# An Evaluation Framework for Software Test Processes

**Dissertation**

zur Erlangerung des akademischen Grades

**doktor ingenieur**
**(Dr.-Ing)**

genehmigt durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von: M.Sc. Ayaz Farooq
geb. am 30.07.1972 in Sargodha (Pakistan)

Gutachter:
Prof. Dr.-Ing. habil. Reiner R. Dumke
Prof. Dr.-Ing. habil. Andreas Schmietendorf
Prof. Dr.-Ing. habil. Fevzi Belli

eingereicht am: 16. Juni 2009
verteidigt am: 29 September 2009

# Abstract

Existing assessment and improvement models of software testing process mainly intend to raise maturity of an organization with reference to testing activities. Such process assessments are based on what testing activities are being carried out, and thus implicitly evaluate process quality. Other test process measurement techniques attempt to directly assess some partial quality attribute such as efficiency or effectiveness using few test measurements. There exists a potential for a formalized method of evaluating test process quality that addresses both implicity and partially of these current evaluations.

The first part of this dissertation surveys and analyzes breadth and depth of existing evaluation approaches in the area of software testing. Strengths and weaknesses of these approaches are highlighted to arrive at a set of requirements for an improved solution. On the other hand, to investigate the scientific and philosophical foundations of evaluation, a short study and analysis into the discipline of evaluation is also made in this part. This research helps identify salient features of an improved test evaluation approach.

Building upon the first phase of research, the second part of the dissertation presents an evaluation framework for specifying and evaluating diverse quality aspects of software test processes. The framework comprises two types of components. Five informative components describe evaluation aspects of testing process, a quality model, a yardstick test process, test metrics, and synthesis techniques. These informative components support a sixth (core) component called evaluation process which details all the steps of test evaluations. The informative components are founded on the theory of evaluation and research into test measurement and evaluation, while the core component derives its concepts from relevant international standards. Combination of this theory, research and best practices helps create a comprehensive test evaluation approach. Finally, the presented approach is exemplified for a domain specific testing approach, i.e., testing process for service-oriented systems.

The main contribution of this dissertation lies in its ability to combine several quality aspects of test processes and, furthermore, in providing an explicit evaluation approach. The developed framework complements the existing maturity models of software testing process by providing a solution that fulfils a subset of requirements at maturity level 4 of TMM/CMMI models.

# Zusammenfassung

Existierende Modelle zur Bewertung und Optimierung des Software-Testprozesses versuchen hauptsächlich den Reifegrad einer Organisation hinsichtlich der Testaktivitäten zu verbessern. Diese Prozessbewertungen entstehen durch Betrachtung der ausgeführten Testaktivitäten und liefern deshalb nur eine implizite Aussage zur Prozessqualität. Andere Prozessmesstechniken versuchen einige Qualitätseigenschaften, wie die Effizienz oder die Effektivität, durch Testmessungen direkt zu bestimmen. Es gibt auch formale Methoden zur Ermittlung der Testprozessqualität, die sowohl die implizite als auch die teilweise direkten Eigenschaften gleichzeitig berücksichtigen können.

Der erste Teil dieser Dissertation gibt einen Überblick zu Bewertungsverfahren auf dem Gebiet des Software-Tests und charakterisiert diese in Bezug auf das durch sie abgedeckte Testspektrum und die erreichbare Testtiefe. Um eine Menge von Anforderungen für einen eigenen verbesserten Ansatz zu finden, wurden die Stärken und Schwächen bekannter Vorgehensweisen herausgearbeitet. Dieser Abschnitt enthält auch eine kurze Betrachtung zu die wissenschaftlichen und philosophischen Grundlagen der Bewertungsdisziplin. Diese Untersuchungen helfen bei der Identifikation entscheidender Merkmale eines verbesserten Testbewertungsansatzes.

Aufbauend auf diese erste Phase der Untersuchungen wird im zweiten Teil der Dissertation ein Bewertungs-Framework zur Spezifizierung und Bewertung verschiedener Qualitätsaspekte von Software-Testprozessen entwickelt. Das Framework umfasst zwei Arten von Komponenten. Fünf informative Komponenten beschreiben die Bewertungs-aspekte des Testprozesses, ein Qualitätsmodell, ein Bewertungsmassstab des Testprozesses, Testmetriken und Synthesetechniken. Diese informativen Komponenten unterstützen eine sechste (Kern- ) Komponente, Bewertungsprozess genannt, die alle detaillierten Schritte der Testbewertung enthält. Die informativen Komponennten sind aus der Theorie der Bewertung und Forschungen zu Testmessungen und Bewertungen abgeleiten worden, während die Kernkomponente auf Konzepten relevanter internationaler Standards beruht. Die Kombination von Theorie, Forschung und Best Praxis hilft so, eine Vorgehensweise, die Ansprüche eines expliziten Testbewertung erfüllt, zu entwickeln. Abschliessend wird die entwickelte Vorgehensweise exemplarisch auf eine domainspezifische Testmethode, den Testprozessen für service-orientierte Systeme, angewandt.

Die Hauptergebnisse dieser Dissertation liegen in der Entwicklung eines Ansatzes zur Kombination verschiedener qualitativer Aspekte von Testprozessen und in der Bereitstellung einer expliziten Vorgehensweise zur Bewertung. Das entwickelte Framework ergänzt die existierenden Reifegradmodelle des Software-Testprozesses um die Erzeugung einer Lösung , die einer Untermenge der Anforderungen des TMM/CMMI Reifegrades der Stufe 4 dieser Modelle entspricht.

# Acknowledgements

First, I am indebted to my supervisor Prof. Dr-Ing. habil. Reiner R. Dumke who provided me an opportunity to undertake this research work. I can't find words to duly express my gratitude for his supervision, guidance, support, and friendship which helped and motivated me all through my work. Thanks also go to Prof. Dr-Ing. habil. Andreas Schmietendorf for his involvement and interest in my research and review of my thesis. Thanks to Prof. Dr-Ing. habil. Fevzi Belli for being so kind in taking time to review my thesis.

I will never forget mentioning Dr. René Braungarten who holds the key role in laying the foundation of my research career. I have learned the very first lessons of doing research from him. Many unnamed teachers starting right from my primary schooling to my college level also share a greate contribution in my academic career. Very special thanks to my colleagues (became friends) in my research group, Dr. Fritz Zbrog and Dr. Martin Kunz. Discussions and feedback from them supported me a lot in building my research approach. Among these names, I cannot ignore mentioning my colleague and friend Konstantina Georgieva with whom I enjoyed a very friendly, peaceful, and pleasant office-sharing environment.

Many thanks to several of my Pakistani friends in this home-away-home. I can't cover names of all of them beyond mentioning Shams, Zahid, Tariq, Rehan, Nasir, Kamran and Zaheer. Their friendship and affection has been a great support for me.

I also find myself thankful to the city of Magdeburg where I have spent six important years of career. I would rather call it my second Heimatstadt (hometown).

Finally, I am grateful to my family who gave me all the support during my journey towards higher studies. Paying any thanks to my mother seems like putting a dot in the sky. Very special thanks to my elder brother Khalid who actually has been the true motivation behind my academic and research career. Thanks to Kiran for being with me, although for a very short time, yet unfolding to me invaluable realities and lessons of life.

# Contents

# 1 Introduction

The purpose of this chapter is to present an overview of the research work at hand and briefly outline structure of the thesis. The chapter begins with a short description of background and motivation behind this work. It then gives a first impression of the addressed research problem, lists the specific research questions, introduces the chosen research approach, and concludes with an outline of subsequent chapters.

## 1.1 Background & Motivation

***Significance of software***

Software is the backbone of computer-based systems which have become an integral part of our everyday life rendering this world-without-computers beyond our imagination. It sits at the heart of almost all kinds of home appliances which have penetrated lives of people across the globe. Software enabled technologies are currently supporting our cars, planes, nuclear power plants, warfare weapons, satellites, space ships and much more. A recent report published by IDC [1] [Microsoft, ] indicates that an overall IT spending, in the western Europe alone, reached 376 billion USD in 2007 and is expected to grow 5.7% a year between now and 2011. According to another finding by European Software Association [ESA, 2008] IT industry, across EU, employs 7.8 million people and the sector contributes 25% of the growth of European economy. These indicators and the results of other studies [Broy et al., 2001], [McGibbon, 2005] further corroborate the economic impact of IT and software industry in Europe and Germany. The EU has already embarked on i2010 [2] strategy to benefit from the fruit of IT progress for improvement of economy, society and personal lives of the general public.

***Software misadventures***

However, our overwhelming dependability on computer systems has left us vulnerable to unimaginable risks. Peter G. Neumann's The Risks Digest forum lists accounts of catastrophic consequences mostly arising out of computer system failures. Although software is not the ultimate culprit in all of these mishaps [Glass, 2008], yet it is behind the considerable proportion of such events/disasters. Partial failure of the toll collection system on German highways causing 10% of all attempts to use the system ending in failure or in people just not paying the toll and of the software program for German employment agency's Arbeitslosengeld II project leaving about 5% of the recipients without money [Weber-Wulff, 2005] are only some examples to mention. Other such incidents where the effects have been widespread and/or most expensive have occurred in space missions[Leveson, 2004], aviation industry [Ladkin, 2006], public mass transport systems [Colville, 2004], financial sector [BBC, 2008], and internet

[1] International Data Corp.

[2] `http://ec.europa.eu/information_society/eeurope/i2010/index_en.htm`

applications [Bennison, 2007] etc. A myriad reports of incidents caused by computer system glitches can also be found elsewhere in [Glass, 1998], [Perrow, 2008] etc.

***Software engineering***

Effective software engineering can help avoid these problems. The term *Software Engineering* was coined in a conference organized by NATO Science Committee in 1968 [Naur and Randell, 1969] for exploiting theoretical and practical knowledge from other engineering disciplines for the construction of software. Fritz Bauer proposed an introductory definition of software engineering in this seminal conference:

> **Definition:** *Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.*

Since then many books, authors and experts on software engineering have provided many competing definitions of this term. However, a more common and comprehensive definition has been given in IEEE Standard Glossary of Software Engineering Terminology [IEEE, 1990]

> **Definition:** *Software engineering. (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).*

The list of initially identified challenges to software engineering such as economics, reliability, and efficiency (some of which although remain partially elusive till today) has been augmented by some of the contemporary challenges of 21st century [Sommerville, 2007, Ch. 1], [Boehm, 2006].

***Software quality***

Quality movement in the field of software engineering did not get real impetus until the monumental works of Garvin [Garvin, 1984], Deming [Deming, 1986], and Juran [Juran, 1988] in the 80s. Although all stakeholders of the software world have always been concerned about quality since the birth of software itself, the exact meaning and ways to achieve it have broadly remained hard-to-achieve targets. The accurate interpretation of software quality is highly perception and context dependent. Quality in the software field is usually attributed to *product* which is believed to be mainly a consequence of design and process quality, to the *process* which is considered to result from maturity of practices, or to the *product in use* which perhaps comes from good understanding and implementation of user requirements. A very promising definition of software quality has been put forward by Pressman [Pressman, 2001, p. 199] in his classical text on software engineering.

> **Definition:** *[Software quality means] conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.*

The extent of the activities for effective management of software quality vary with the size of system involved. While informal approaches may suffice for smaller systems, quality management of larger systems will typically comprise distinct quality assurance, quality planning, and quality control activities [Sommerville, 2007, p. 643]. Regardless of the size of the system, at least some sort of verification, validation, and testing activities are always performed as part of the quality management process. Other higher level approaches include several product and process standards.

***Standardizing software engineering***

Among the most common factors giving rise to software failures are misinterpretation of system requirements, poor communication among people involved in software projects, sloppy development practices, and poor project management etc [Charette, 2005]. Standards are among the main elements of software quality engineering which are believed to assist in reducing many of these failure factors. In the opinion of Sommerville [Sommerville, 2007, p. 646] standards are helpful since they are based on best practices, provide a framework for implementing quality assurance process and assist in establishing systematic and continuous approaches to software development. IEEE and ISO have been leading contributors to the development of software related standards. JTC 1 [iso, ] is the ISO's committee which works on information technology related standards. As of now, the number of standards published directly by this committee stands at 540 while this count reaches to 2250 if the standards published by its sub-committees are also included. This is more than the number of standards published by any ISO committee in any other field. IEEE [iee, ] has also published a number of standards pertaining to software, while Specialist Group in Software Testing [BCS, ] of the British Computer Society is also working on few software testing related standards. Involvement of so many agencies for developing software engineering standards marks the evolving nature of this research area.

ISO 9000 is the series of familiar standards for quality management systems. Its first release in 1987 was followed by several revisions with the latest release appearing in the year of writing this thesis. Some of the well known product quality related standards include ISO/IEC 9126 Software Engineering–Product Quality and its successor ISO/IEC 2500n Software Engineering -Software product Quality Requirements and Evaluation (SQuaRE) series, and ISO/IEC 14598 Information technology–Software product evaluation, while few of those related to software process are IEEE/EIA 12207 Standard for Information Technology–Software life cycle processes, ISO/IEC 15939 Software Engineering - Software measurement process, ISO/IEC 15504 Information Technology–Process Assessment, ISO/IEC 15288 Systems engineering–System life cycle processes, and the regional standard V-Modell XT [iABG, ] Development Standard for IT Systems in Germany (previously V-Modell 97). A more detailed chronicle of commercial and governmental standards relevant to software quality assurance can be found in [Schulmeyer, 2008, Ch. 3]

***Processes improvement***

The well known fact about the common engineering and manufacturing processes is that a better quality process produces equally better products. As the software development is also considered to be an engineering activity, many of the concepts from the engineering and manufacturing world penetrated into the software field. It is a widely held belief that the software product quality is a direct consequence of the process used to develop

it [Humphrey, 1989], [Kenett and Baker, 1999]. Improving the software process, thus, serves as an implicit approach to product quality improvement. Importance of improving the software process was acknowledged in the mid 80s with the initiation of work on Capability Maturity Model (CMM) and the ISO 9000 standard. However software process is a complex human-intensive activity as opposed to other manufacturing processes. Finding the best way to improve the software process has been a great challenge both to the practitioners and researchers. The history of software process research [Fuggetta, 2000] is replete with numerous techniques to manage, control, and improve it. Exemplary efforts in this area include those for establishing [Dumke et al., 2006a], [ISO/IEC, 2004], [Wang and King, 2000], defining [IEEE, 1997b], [IEEE/EIA, 1998], modeling [Zamli, 2004], assessing [ISO/IEC, 1998], [April, 2005], and measuring [Basili and Weiss, 1984], [Ebert and Dumke, 2007] the software process.

***Testing processes***

Since testing is widely acknowledged to consume a considerable proportion of software development resources, improving the testing process also leads to achievement of similar goals as aspired by generic process improvement programs, the difference being in the scale and coverage of activities only. Impressed by similar works over software process, the maturity models and improvement approaches for software testing process began around 1996 with the introduction of Testing Maturity Model (TMM) [Burnstein, 2003] and Test Process Improvement (TPI) model [Koomen and Pol, 1999]. The latest augmentation in this direction is the Test Maturity Model Integration TMMi [Goslin et al., 2008b] as revitalization of TMM. This new maturity model, at the moment, has been worked out only up to maturity level 2 while further development is under way. However there have been few criticisms of both process maturity models[Humphrey et al., 2007] and test maturity models [Farooq et al., 2007]. This thesis aims to address some of the deficiencies in test process assessment approaches, and attempts to build a scientifically rigorous, explicit, and comprehensive method for evaluation of test processes.

### 1.1.1 Background on Test Process Evaluation

***Why improve processes?***

High level business goals, other quality objectives set by business managers, and user's needs are major driving forces behind every test process improvement program. Many times, transforming these implicit or complex intensions into concrete process improvement objectives is in itself a challenge. However, alongside organizational and project specific targets, the most common process improvement goals as identified by Solingen and Berghout [van Solingen and Berghout, 1999, p. 11] are to:

- Increase quality
- Decrease costs
- Shorten project cycle time
- Decrease risks

***Processes improvement defined***

As testing is mainly aimed at finding errors in programs [Myers, 2004], thus additional goals of testing process, alongside those mentioned above, would be to efficiently detect defects, remove errors, and avoid system failures. While setting testing process goals is

easier, it is difficult to perceive them accurately and arrive at a set of process improvement actions. For this we must first try to understand it well what exactly is meant by improving a process. It has been defined in the following manner,

> **Definition:** *Software process improvement (SPI) is a systematic procedure to improve the performance of an existing process system by changing the current processes or updating new processes in order to correct or avoid problems identified in the old system by means of a process assessment.* [Wang and King, 2000, p. 42]

***Test process improvement***

The definition above implies that the process *performance* has to be improved. Cost and time are undoubtedly the biggest factors that businesses want to reduce. But it is not just cost and time expenditures that have to be reduced to a minimum. The system quality must also be improved. So we have to find an optimal balance among these factors. Many diverse techniques have been worked out to meet one or the other set of these goals. Managing and controlling the testing activities provides one way of improving the testing process. Management models [Pol et al., 2002], [Drabick, 2003] and formal models [Cangussu, 2002], [Stikkel, 2006] based on the feedback mechanisms are few examples of this kind. To reduce the cost, time, and effort of generating and executing test cases, model-based testing [Utting and Legeard, 2006] exists to help us. Independent verification and validation (IV&V) and special methods for testing embedded systems [Broekman and Notenboom, 2003] are primarily aimed at reducing risk in systems requiring high accuracy. Assessment is another class of approach to realize improvement of the testing practices. It is seen as a pre-requisite to be able to begin with an improvement activity. This situation has been described by Humphrey [Humphrey, 1989] as the three main questions that should be addressed by any process improvement activity:

1. How good is my current software process?
2. What must I do to improve it?
3. Where do I start?

The first question aims to get a first hand idea of current development/testing activities. The second relates to spotting weaknesses in the existing practices. The third question is about finding out what to do to improve the situation. Assessment is a systematic way to address all these three questions. In the field of software testing research, several assessment techniques have been developed for three main testing entities, i.e. techniques, tools, and processes [Farooq and Dumke, 2008b]. For testing techniques there are empirical analysis and measurements [Farooq and Dumke, 2008b, Ch. 3], for tools there are evaluation approaches to help in their selection [Farooq and Dumke, 2008b, Ch. 4], and for testing processes are assessment models[Farooq and Dumke, 2008b, Ch. 2]. Process assessments typically give an enhanced picture of an overall capability of a variety of testing practices. Wang and King [Wang and King, 2000] define it as:

> **Definition:** *Software process assessment (SPA) is a systematic procedure to investigate the existence, adequacy, and performance of an implemented process system against a model, standard, or benchmark.* [Wang and King, 2000, p. 42]

Assessment based approaches of the testing process have broadly followed the maturity model concept of capability maturity models of software process (CMM/CMMI). These are collections of experience-based best testing practices that are believed to yield desired process improvements. Examples of such models include Testing Maturity Model (TMM), Test Process Improvement (TPI) model, and Test Maturity Model Integration (TMMi) introduced earlier in this chapter. The next section highlights the problems with current test process assessment approaches that this thesis aims to address and explains the followed research approach.

## 1.2 Research Setting

This section is intended to introduce the research project at hand and describe the general and specific research questions addressed by it. Furthermore, the connection of the chosen issue with the contemporary research topics is established. The adopted research approach is also explored within the context of practised software engineering research paradigms.

### 1.2.1 Research Problem

Today's industrial software engineering environments adequately realize the caliber of a well controlled testing process as a means to increasing test efficiency, replying to rapidly increasing complexity, and improving quality of the developed products. Efficient assessment and improvement of the testing processes, therefore, is a primary concern of the today's testing approaches. Small costs, large benefits, and high benefit/cost ratio are probably the most common characteristics sought by software organizations when selecting and undertaking any kind of programs for assessing and improving these processes. However, unfortunately many largescale SPI programs have been found to be highly expensive and difficult to be undertaken by small organizations [Rico, 2004, p. 175]. In general, monolithic process assessment and improvement models have been found to be cost, time, and resource hungry [Staples et al., 2007] and nurture a culture of maturity levels instead of actual process capability [Humphrey et al., 2007]. Furthermore, current process maturity models are only implicit in nature since they assess organizational processes based on their conformance to the set of best practices chosen from the given area. Researchers have already realized these problems and have devised methods by which existing process assessment models can be customized or also have developed new specialized approaches. Current test process assessment and improvement models, which are mostly designed on the pattern of maturity models like CMM/CMMI, suffer from similar kinds of resistance to their adoption [Farooq et al., 2007]. It has been found that the approaches addressing the aforementioned problems in case of testing process are limited.

Given this general characterization of current test process assessment models, the specialized task of this research project is to investigate prevalent forms of test process assessments, diagnose the existing problems in the area, and come up with a light-weight quantitative evaluation methodology for software test processes which is small enough to be reasonably cheap, yet powerful enough to provide meaningful assessments and improvement suggestions. Although few earlier research works have been devoted to these issues, to date no test process evaluation method has been reported in literature which determines

capability of test processes based on *what organizations have achieved* rather than on *what they do*.

### 1.2.2 Research Questions

The main research question investigated by the current thesis is:

**Could a process be defined to allow medium size industrial organizations perform light-weight and explicit evaluation of their testing processes?**

Sine the main research question is quite broad in its perspective, it can be broken down into following secondary research tasks:

Sub-tasks of Research

RT-1. Traverse the area of software testing to identify entities of evaluation. Explore existing evaluation forms and approaches as applied in this field. Investigate characteristics of low-cost and effective test evaluations.

RT-2. Investigate the philosophy of evaluation in general and with special focus to software engineering. Identify core elements of a comprehensive process evaluation approach.

RT-3. Analyze available test process evaluation approaches based on the criterion developed in RT-1/RT-2.

RT-4. Bring forward the concept of a comprehensive approach that fills the gaps marked in RT-3 as well as which satisfies requirements set forth in RT-1/RT-2.

RT-5. How can the idea conceived by RT-4 be transformed into a practical test process evaluation and improvement model?

RT-6. Can the approach developed as a result of RT-5 be validly applied in practical and industrial situations? What are the limitations and the needed adjustments in applying the new approach?

### 1.2.3 State of SE Research

The field of software engineering research is almost as old as the modern computer itself. Despite the many developments and revolutionary inventions made possible by software (which certainly in turn owe to software engineering research), from time to time SE research has been criticized as well–that it is unscientific in approach [Fenton et al., 1994], ignores evaluation [Zelkowitz and Wallace, 1997], is immature [Shaw, 2001] and narrow [Glass et al., 2002], and lacks a decided direction [Poore, 2004]. It has also been observed that a gap exists between the results of academic efforts and the application of these

results in the software industry [Glass et al., 2002] possibly due to irrelevance of research or intransigence of the practitioners. That research model of past is no more a choice of majority of SE researchers today, which Glass [Glass, 1994] once described as *advocacy research* consisting of steps "conceive an idea, analyze the idea, advocate the idea" mainly following the notion that any change is better than status quo. Referring to the knowledge on philosophical and experimental foundations of software engineering research, the next sections explore the connections of current research work to the software engineering patterns of research.

#### 1.2.3.1 Research Paradigms & Phases

The word *paradigm* refers to certain concepts in linguistic and scientific disciplines. Thomas Kuhn, while explaining the philosophy of science in his famous book *The Structure of Scientific Revolutions* [Kuhn, 1996], defines a scientific paradigm as,

> *some accepted examples of actual scientific practice–examples which include law, theory, application, and instrumentation together–provide models from which spring particular coherent traditions of scientific research*

In other words, scientific paradigm explains *how* a scientific work is performed which begins with observation, goes through discovery and analysis, and culminates in some precise results. In simple words, it is a kind of work plan to solve similar problems. An example of a paradigm in the field of computer science or software engineering is the notion of *programming paradigms*. Similarly a *research paradigm* is a way to solve a given class of research problems. Chen [Chen, 2005] defines a research paradigm more precisely as,

> *a dynamical system of scientific works, including their perceived values by peer scientists, and governed by intrinsic intellectual values and associated citation endurance and decay*

Almost three decades old legendary work of Peter Wegner over research paradigms in computer science [Wegner, 1976] is still meaningful today. He mentioned the evolutionary path of prevalent research paradigms as comprising empirical, mathematical, and engineering paradigms. A later explanation of contemporary research paradigms in the field of software engineering was given by Adrion [Adrion, 1993] at the 1992 Dagstuhl Workshop on Future Directions in Software Engineering held in Germany. He named *scientific*, *engineering*, *empirical*, and *analytical* as the four main methods of software engineering research which have been further elaborated in figure1.1. The approach followed by this thesis is highlighted in this figure.

The above mentioned paradigms, or synonymously called methodologies, share some common structure of activities that are applied while following any of these research paths. Glass [Glass, 1995] proposed these patterns to be called research phases. These four phases are,

- **The informational phase**: gathering information via reflection, literature survey, people/organization survey, or poll

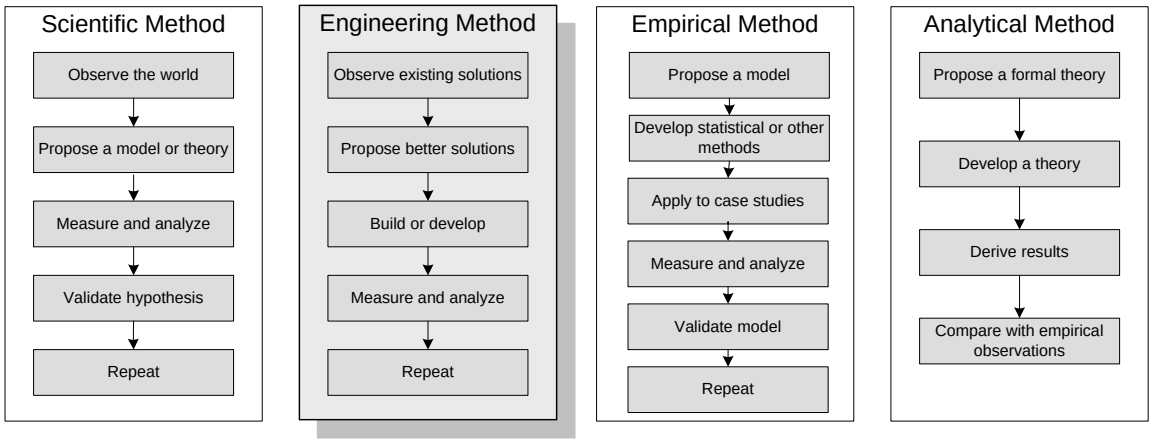


**Figure 1.1:** *Software Engineering Research Methodologies*

- **The propositional phase**: proposing and/or formulating a hypothesis, method or algorithm, model, theory, or solution
- **The analytical phase**: analyzing and exploring a proposition, leading to a demonstration and/or formulation of a principle or theory
- **The evaluative phase**: evaluating a proposition or analytic finding by means of experimentation or observation, perhaps leading to a substantiated model, principle, or theory.

#### 1.2.3.2 Classifications of Research

Hierarchical classification of research topics is a common concept both in computing and non-computing research fields. Journals more often use such schemes for organizing and indexing research works. These classifications also help to establish context and connection of a research work with the relevant and surrounding areas.

Probably the earliest classification of topics in the computing field dates back to 1964 when the first version of ACM's Computing Reviews Classification System appeared. With the emergence of new areas and research, the classification was later revised in 1991 and then in 1998 [ACM, 1998]. Between 2002 and 2004, Glass et al. [Glass et al., 2002], [Glass et al., 2004] developed a broader classification scheme covering the computing field in general and the software engineering field in particular. The scheme is not just limited to the research topics alone, but covers other characteristics of research such as the research approach, method, and relevant disciplines. As an another instance, the content of the IEEE's Guide to the Software Engineering Body of Knowledge [Abran et al., 2004] also serves as a reference resource for software engineering research topics. Some further discussions on possible paradigms, approaches, and methods used in the software engineering research appearing in [Vessey et al., 2005] and [Holz et al., 2006] have presented numerous viewpoints on classifications which overlap with Glass's scheme.

Table 1.1 reproduces above mentioned Glass's classifications of computing research (the areas to which this thesis relates are highlighted). Considering it an adequately comprehensive coverage of research classifications, the thesis at hand uses this classification as a

reference for describing the context of current research work. Glass discussed following aspects of his classifications which have been summarized in table 1.1.

- **Research topic**: It refers to the field of application or the problem type/area addressed. The content of the research work makes up its topic area. This element pertains to the *what* dimension of the research. Topics are divided into major categories and sub-categories.
- **Research approach**: Researchers employ a number of techniques to solve a given problem. The *how* dimension of the research is described through this aspect. This gives the broad or high level description of the adopted research technique. Approaches are also grouped into categories and sub-categories.
- **Research method**: In contrast to a brief view of research techniques, a detailed description is classified as the research method.
- **Reference discipline**: Researchers many times exploit knowledge from disciplines other than their main field of work to establish baselines for their research. For example, for developing formal models of software processes or the alike, a researcher may borrow knowledge from mathematics.
- **Level of analysis**: In information systems, computer science, and software engineering, different objects of interest are studied. These objects may be of technical kind or abstract, or may involve individuals or organization. Level of analysis is related to these objects. For example, Glass [Glass et al., 2004] found that nearly all CS (computer science) and SE research work was conducted at the technical level, studying artifacts or entities.

**Table 1.1:** *Research Classifications*

| **1. Research Topics** | |
|---|---|
| – Problem-solving | – System/software management |
| - Algorithms | - Project/product management |
| - Mathematics/computational sci. | - *Process management* |
| - Methodologies | - *Measurement/metrics* |
| - Artificial intelligence | - Personnel issues |
| – Computer | - Acquisition of software |
| - Principles or architecture | – Organizational |
| - Intercomputer communication | - Organizational structure |
| - Operating systems | - Strategy |
| - Machine-level data/instructions | - Alignment |
| – Systems/software | - Org. learning |
| - Architecture/engineering | - Tech. transfer |
| - Software lifecycle/engineering | - Change management |
| - Programming languages | - IT implementation |
| - Methods/techniques | - IT usage/operation |
| Continued on next page... | |

**Table 1.1 – continued from previous page**

| | |
|---|---|
| - Tools | - Mgmt. of 'computing ' function |
| - Product quality | - Computing as a business |
| - Human-computer interaction | - IT impact |
| - System security | - Legal/ethical/political implications |
| – Data/information | – Societal |
| - Data/file structures | - Cultural implications |
| - Data base/mart org. | - Legal implications |
| - Information retrieval | - Ethical implications |
| - Data analysis | - Political implications |
| - Data security | – Disciplinary issues |
| – Problem domain-specific | - 'Computing' research |
| - Scientific engineering | - 'Computing' curriculum/teaching |
| - Information systems | |
| - Systems programming | |
| - Realtime | |
| - Edutainment | |
| **2. Research Approach** | |
| – Descriptive | – Evaluative |
| - Descriptive System | - Formulative-concept |
| - *Review of Literature* | - Formulative-framework |
| - Descriptive Other | - Formulative-guidelines |
| – Evaluative | - Formulative-model |
| - Evaluative-deductive | - *Formulative-process, method* |
| - Evaluative-interpretive | - Formulative-classification |
| - Evaluative-critical | |
| - Evaluative-other | |
| **3. Research Method** | |
| - Action Research | - Grounded Theory |
| - *Case Study* | - Hermeneutics |
| - Concept Implementation | - Instrument Development |
| - Conceptual Analysis | - Laboratory Experiment-Human |
| - Conceptual Analysis/Mathematical | - Laboratory Experiment-Software |
| - Data Analysis | - *Literature Review/analysis* |
| - Descriptive/Exploratory Survey | - Mathematical Proof |
| - Ethnography | - Protocol Analysis |
| - Field Experiment | - Simulation |
| - Field Study | |
| **4. Reference Discipline** | |
| - Cognitive Psychology | - Not applicable |
| - *Computer Science* | - Science |
| - Economics | - Self-Reference |
| - Management | - *Social and Behavioral Science* |
| - Management Science | - Other |

Continued on next page...

**Table 1.1 – continued from previous page**

| | |
|---|---|
| - *Mathematics* | |
| **5. Level of Analysis** | |
| - *Abstract Concept* | - Organizational Context |
| - Computing Element | - Profession |
| - External Business Context | - Project |
| - Group/Team | - Society |
| - Individual | - System |

#### 1.2.3.3 Characteristics of Research at Hand

In the context of the aforementioned classifications of research efforts, the current work can be considered to match with the following research patterns:

- **Research Paradigm:**
  - Engineering
- **Research Topics:**
  - Process management
  - Measurement/metrics
- **Research Approaches:**
  - Review of literature
  - Formulative-process, method
- **Research Methods:**
  - Case study
  - Literature review/analysis
- **Reference Disciplines:**
  - Computer science
  - Social and behavioral science
- **Analysis Levels:**
  - Abstract concept

## 1.3 Structure of Thesis

The current chapter was dedicated to establishing a foundation for this research work. It has given a very brief historical background first to the software engineering research and then to the related area. The research problem has been defined and the approach and connection of current research is described in different contexts. Highlights of the upcoming chapters are given below while the figure 1.2 visualizes thesis structure.

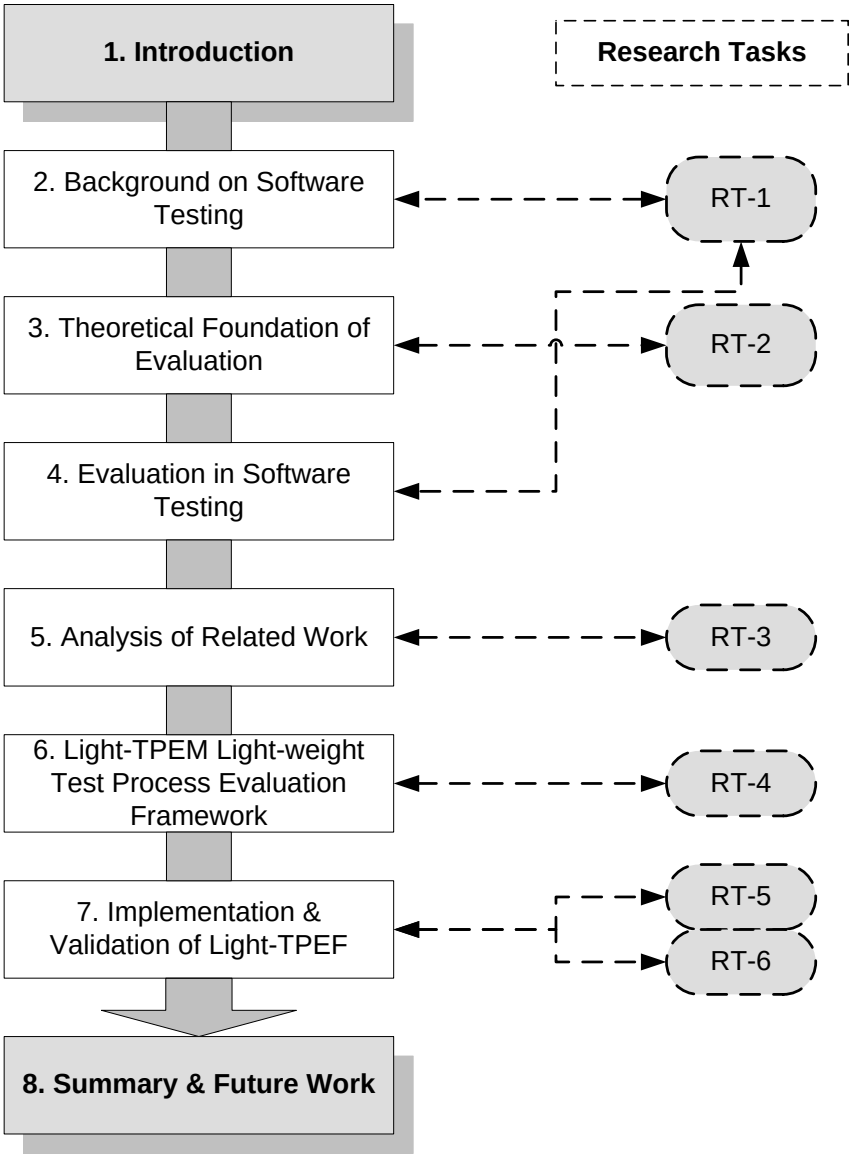


**Figure 1.2:** *Thesis Structure*

***Chapter 2*** explores the field of software testing research and methods with a view to developing a context within which the current research exists. The survey given herein helps to appreciate significance and complexity of the testing aspects. It helps to identify chief entities of evaluation mostly sought by technical and managerial people to determine how well the testing is being done.

***Chapter 3*** introduces the discipline and philosophy of evaluation in general. After a short discussion over evaluation methods, the chapter explains how evaluation is conceived and applied in the field of software engineering. The chapter ends with a list of characteristics and elements of a comprehensive evaluation approach that could be applied to any entity of interest in software engineering.

***Chapter 4*** covers the breadth of available knowledge related to evaluation of software testing entities of interest discovered in chapter 2. It surveys concept, approach, and char-

acteristics of all kinds of process assessment models, assessment and measurement of testing techniques, and qualitative and quantitative evaluation of testing tools. Based on their generality and comprehensiveness, a few of these solutions are shortlisted for a further deeper analysis to be performed later. The chapter concludes by identifying characteristics of successful yet light-weight test evaluation approaches.

***Chapter 5*** is concerned with an in-depth analysis and evaluation of previous work done related to the current research problem. It builds upon research performed in chapter 3 and 4. It first extracts the requirements for a test evaluation approach as established in the previous two chapters. Shortlisted solutions are then analyzed step-by-step for their appropriateness against these criteria. The chapter concludes with a vision of a new evaluation approach which fulfils the discovered shortcomings in the analyzed models.

***Chapter 6*** is the heart of this thesis and presents the structure and content of the proposed test process evaluation framework. All the elements of the framework are explained in sufficient detail using appropriately chosen process modeling techniques. It provides a foundation over which a concrete test evaluation process model could be built. The model is verified to fulfill all the requirements of light-weight test evaluation approaches established at the beginning of the previous chapter.

***Chapter 7*** discusses the implementation and validation of the framework proposed in the previous chapter. The proposed model is verified and validated using assertion techniques. The chapter presents implementation of a workable solution by converting the model into an executable evaluation process model. This process model is demonstrated to fit into an example SOA governance model as an example of a business environment. It serves as a prototype implementation of the developed framework to motivate further industrial implementations of the presented approach.

***Chapter 8*** retrospects the research work with the research goals and takes a look at the prospects of future work in the related areas of research.

# 2 Background on Software Testing

As a first step towards this research work, this chapter recaps some of the fundamental concepts in the field of software testing. The content and organization of this chapter is aimed at building a wide perspective of the area from which this thesis springs out. This synopsis motivates the reader to appreciate the significance of the research problem and connect it to its parent area of knowledge.

## 2.1 Introduction

***Testing defined***

Software testing literature is entangled with somewhat confusing terminologies. Testing, per its classical definition by Glenford Myers [Myers, 2004], *is the process of executing a program for finding errors.* Verification and validation (V&V) are also related activities carried out for the same purpose which may or may not involve code execution. Testing and V&V are mainly support activities of the development process. They serve as evaluation techniques for the software development artifacts as well as tools for quality assurance. Slightly overlapping context descriptions of testing related activities have been found in literature. For instance,

- Guide to the Software Engineering Body of Knowledge (SWEBOK) [Abran et al., 2004, p. 11-1] lists testing related topics inside *software quality* knowledge area. It describes software quality management processes as comprising software quality assurance, verification, validation, reviews, and audits.
- Jeff Tian [Tian, 2005, p. 27] also describes verification, validation and testing as part of quality assurance.
- IEEE/EIA 12207 standard [IEEE/EIA, 1998] organizes software life cycle processes into three categories, namely primary life cycle processes, supporting processes, and organizational life cycle processes. Quality assurance, verification, validation, joint reviews, and audit are listed inside supporting life cycle processes, while quality assurance process may in turn make use of results of other supporting processes such as verification, validation, joint reviews, and audit.

Figure 2.1 gives a visual representation of these relationships among software quality engineering, software quality assurance and software testing discussed above.

***Complexity of testing***

Software testing is a complex and critical task among software development activities. This complexity stems from the characteristics of the software systems themselves. Size, safety criticality, business value and increasing dependence of people on software systems

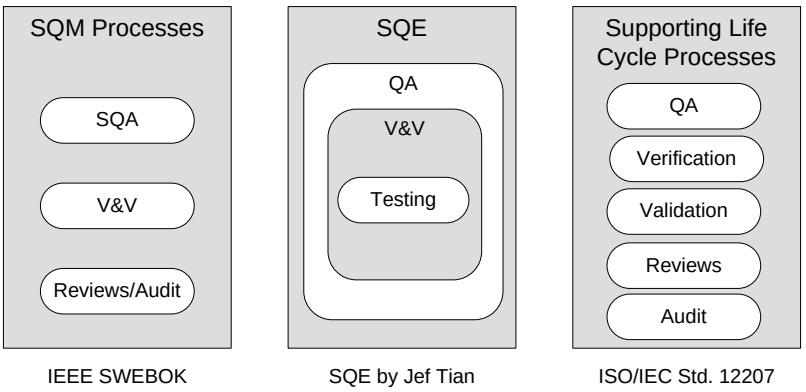


**Figure 2.1:** *Some Context Descriptions of Software Testing*

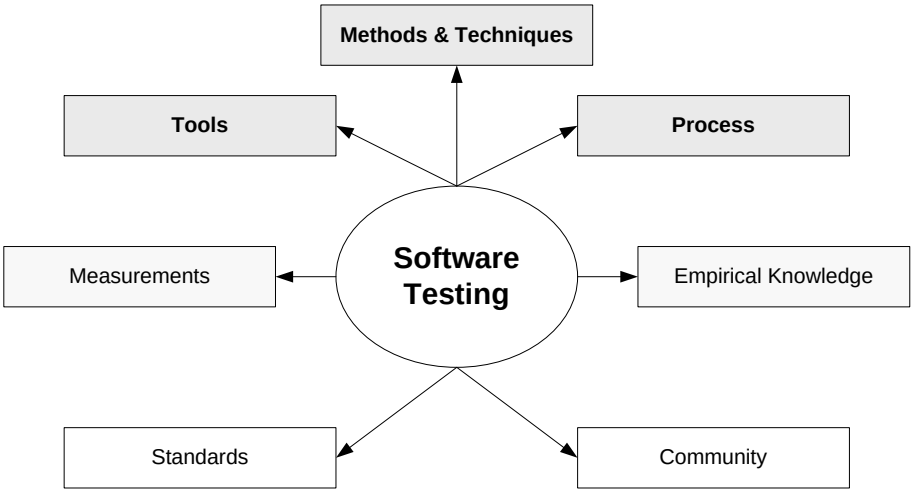


**Figure 2.2:** *Software Testing Elements of Interest*

call for their failure free functioning. Software testing can give us this confidence that a system will work as intended. But this confidence building process costs almost half of the total system development time and money. During this process, testers apply different techniques, avail different tools, sometimes follow some standards, use few measurements and exploit empirical knowledge during the core of their testing tasks. Figure 2.2 presents a visualization of different elements that are involved with and support the task of software testing. Owing to this breadth of the field, software testing occupies a dedicated key area within the software engineering body of knowledge [Abran et al., 2004].

***Research issues in testing***

The area of software testing research is almost as old as the software engineering itself. Historically speaking, an overwhelming portion of software testing research has focused on test case design, static and dynamic testing techniques, problem-centered testing approaches such as for object-oriented design or for embedded systems software, testing tools, and designing effective testing processes. Based upon analysis of several latest articles over past and future research trends in software testing, it can be inferred that the research on fundamental testing issues such as testing methods, tools, and processes has

**Table 2.1:** *Research Issues in Software Testing*

| Reference | Issues Highlighted |
|---|---|
| [Harrold, 2000] | Testing component-based systems<br>*Test effectiveness*<br>*Creating effective testing processes*<br>Testing evolving software |
| [Abran et al., 2004, p. 5-3] | Test selection<br>*Test effectiveness*<br>Test oracles<br>Testing for defect identification<br>Testability<br>Theoretical and practical limitations of testing |
| [Taipale et al., 2005] | Testing automation<br>Standardization<br>*Test process improvement*<br>Formal methods<br>Testing techniques |
| [Bertolino, 2007] | *Test process improvement*<br>*Test effectiveness*<br>Compositional testing<br>*Empirical body of evidence*<br>Model-based testing<br>Test oracles<br>Domain specific test approaches |

somewhat matured. Our focus is now more on advanced and finer problems such as establishing empirical baseline on testing knowledge, test process improvement, standardization, demonstrating effectiveness of testing methods, tools, and processes, and on test automation. Table 2.1 summarizes active research issues in software testing collected from latest literature on testing research.

## 2.2 Testing Process

***Need for process based approach***

With fast growing size of software systems, numerous complexity issues and wealth of professional practices, software development is no longer a programmer oriented activity. Process based software engineering methodology has evolved out of this chaos as a systematic approach that can handle issues related to development methodology & infrastructure, organization, and management of software development activities. Software processes has become a key research area in the field of software engineering today.

***Testing process defined***

Being critical to the quality of the developed product, testing activities occupy major portion of the software development process and are believed to involve heavy expenses, development effort, and time. Owing to their important role, all testing related activities and issues can be seen as a unity or a process. The scope and content of this process can be viewed from two perspectives. First is the purpose for which testing exists. As for its objectives testing process has been defined as,

> **Definition 1:** *Software testing process (by purpose) is the set of activities for revealing defects in software, and for establishing that the software has attained a specified degree of quality with respect to selected attributes.* [Burnstein, 2003, p. 7]

To achieve these objectives diverse forms, phases, and levels of testing activities are performed. It is this content perspective which refers to the type of these activities that are considered to be part of the whole testing process. In the context of the current thesis, this content view brings about the following definition of the testing process,

> **Definition 2:** *Software testing process (by content) encompasses all kinds of verification and validation activities as well as all types of reviews (management reviews, inspections, walk-throughs,...), all phases of testing (planning, design, execution,...) and all levels (unit, integration, system,...) of testing activities.*

***Testing process explained***

Now, similar to the two levels of studying software engineering processes as mentioned in IEEE SWEBOK [Abran et al., 2004, p. 9-1], the test process can also be studied at two levels. The first level refers to technical and managerial activities that are carried out to verify and validate development artifacts throughout the software development lifecycle. The second is the meta-level which involves the definition, implementation, assessment, measurement, management, change, and improvement of the test process itself. This chapter mainly concerns with this meta-level description of the test process which applies to all kinds of testing methods and domains.

Different kinds of meta-level descriptions of test process exist. It is usually described as generic process phases or as a series of various levels of testing. It is commonly studied as an organization of testing techniques [Everett et al., 2007], as a quality assurance approach [Tian, 2005], [Lewis, 2004], or a means to managing different kinds of testing activities [Pol et al., 2002]. A generic very high level structure of test process activities has been given by Tian [Tian, 2005, p. 68]. He divides test process into three main groups of test activities which are,

- *Test planning and preparation*, which sets the goals for testing, select an overall testing strategy, and prepare specific test cases and the general test procedures.
- *Test execution* and related activities, which also include related observation and measurement of product behavior
- *Analysis and follow-up*, which include result checking and analysis to determine if a failure has been observed, and if so, follow-up activities are initiated and monitored to ensure removal of the underlying causes or faults that led to the observed failures in the first place.

Figure 2.3 summarizes these common test process activities. Another detailed level picture of the testing process is the one given by Perry [Perry, 2006, p. 157] who divides it into seven steps, namely; organizing for testing, developing the test plan, verification testing, validation testing, analyzing and reporting test results, acceptance and operational testing, and post-implementation analysis. Perry's partitioning of testing process into these seven steps can be considered to be an expansion of Tian's three phased view discussed above.

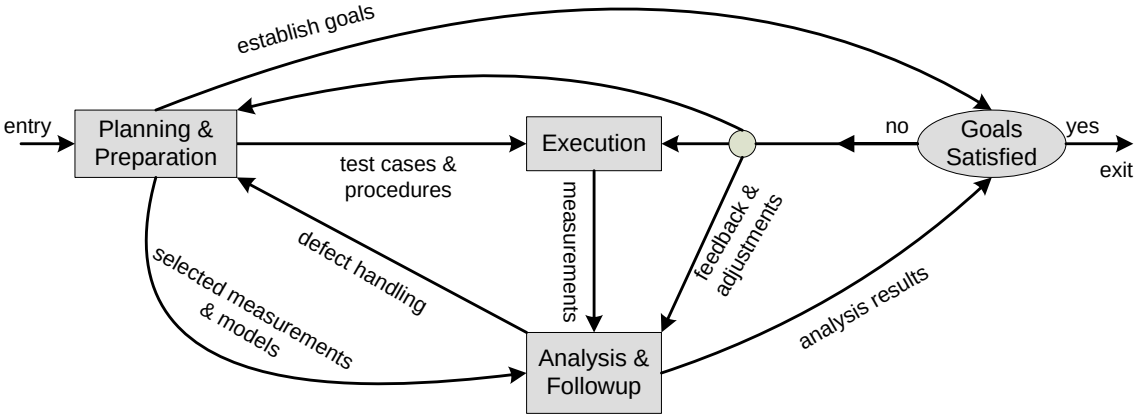


**Figure 2.3:** *Generic Structure of Testing Process [Tian, 2005]*

***Benefits of testing process***

A well established test process can bring about many benefits to all stakeholders. According to Perry [Perry, 2006] these advantages include,

- *Testing is consistent*: Following test process matures the practices. Successful practices can be re-implemented for other projects which reduces variability of activities and increases our confidence.
- *Testing can be taught*: In a heroic testing where no process exists, testing is mainly an art confined to a master tester. Breaking testing into processes makes it understandable and teachable.
- *Test processes can be improved*: By using processes we can identify ineffective areas and activities. Such deficiencies can be removed to make testing cost-effective and improve product quality.
- *Test processes become manageable*: When a process is in place, it can be managed. If it is not, then things are being done in an ad-hoc manner where there can be no management.

### 2.2.1 Research directions

Three main issues concerning test process research are: definition or modeling, evaluation, and improvement.

***Test process definition***

As mentioned above, testing (by definition) is the process of executing code for finding errors, such as unit testing process etc. In this context, the *definition of the test process* refers to the definition of the processes as models, plus any optional automated support available for modeling and for executing the models during the software process (derived from [Acuña et al., 2001]). This may be in the form of a description of part/whole of test process using a suitable process modeling language. An example is model-based testing approaches. Another way to define a test process is to give an activity based description of the process aimed at activity management. This approach to defining testing process is used when testing refers to all sorts of methods for identifying software errors e.g. inspection process, or system testing process. Examples include well known testing standards and other generic and domain-specific test process descriptions.

***Test process evaluation***

*Test process evaluation* is a systematic procedure to investigate the existence, adequacy, and performance of an implemented process system against a model, standard, or benchmark (derived from [Wang and King, 2000, p. 42]). It is the investigation of the current state of the process with a view of finding necessary improvement areas. Process evaluation is typically performed prior to any process improvement initiative. Test process evaluation and improvement is motivated by a concern for cutting on testing costs and improving product quality.

***Test process improvement***

*Test process improvement* is a systematic procedure to improve the performance of an existing process system by changing the current process or updating new processes in order to correct or avoid problems identified in the old process system by means of a process assessment (derived from [Wang and King, 2000, p. 42]). In parallel with the concern for software process improvement, test process improvement also continues to be a major research direction within software testing. It has been ranked by Taiple [Taipale et al., 2005] as one of the top three important issues in software testing research.

In most cases a solution may address more than one of the above mentioned three issues at the same time. For instance, process evaluation and improvement are mutually connected issues of software test process. Any software process improvement initiative needs first an evaluation of the current level of performance of the process. A process evaluation exercise should eventually follow an identification of and suggestions over most important process improvement areas. Therefore, test process evaluation and improvement will be reviewed in the same section in this text. Figure 2.4 gives a classification and summary of existing approaches in this regard.

## 2.2.2 Test Process Definition & Modeling

Existing test process modeling approaches include some *empirical and descriptive*, and *formal and descriptive* process models. According to Wang and King [Wang and King, 2000, p. 40] an empirical process model defines an organized and benchmarked software process and best practices, a descriptive model describes *what to do* according to a certain software process system, while a formal model describes the structure and methodology with an algorithmic approach.

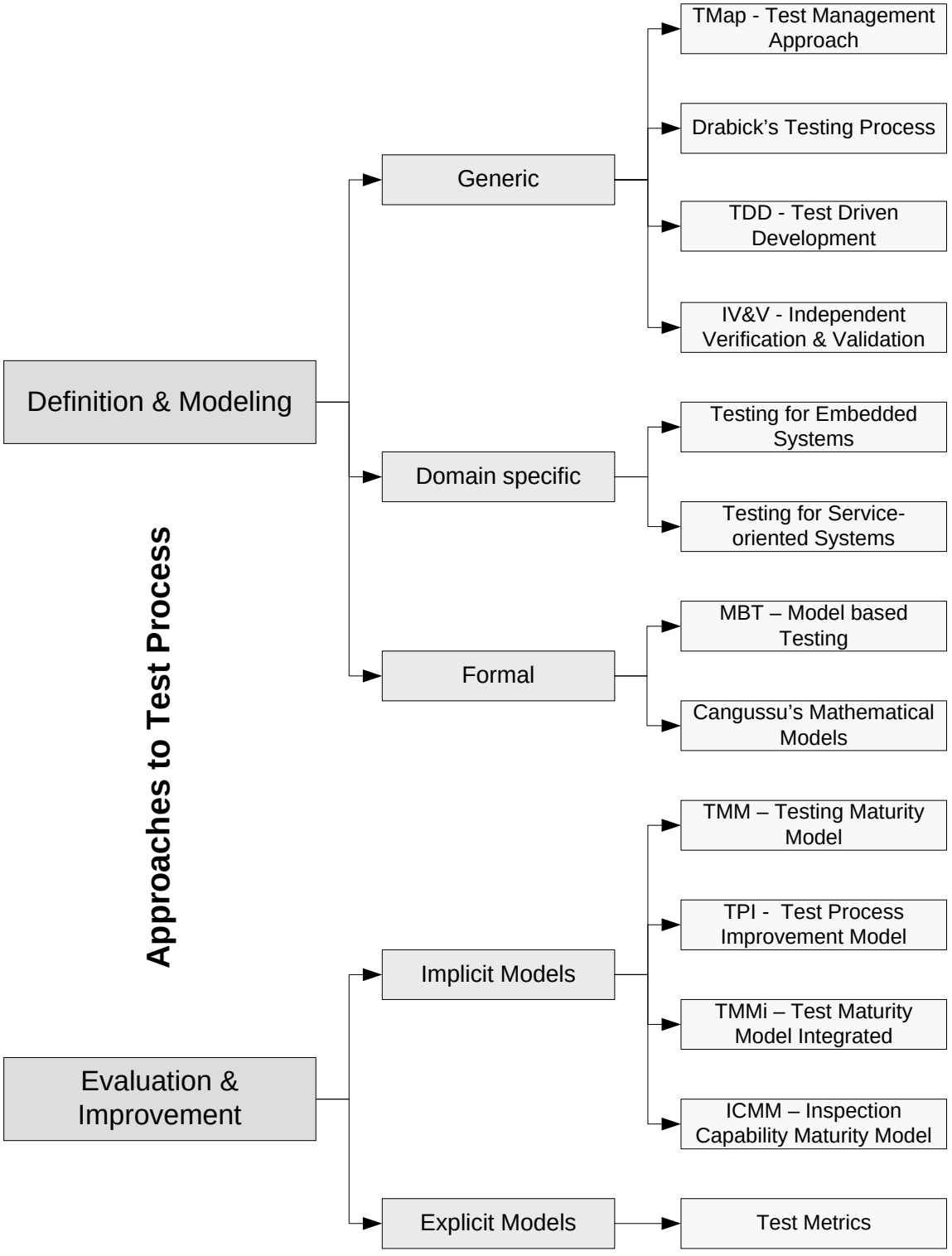


**Figure 2.4:** *Approaches to Software Testing Processes*

#### 2.2.2.1 Generic Test Process Descriptions

An activity-based description of the software test process has been given by Perry [Perry, 2006, Ch. 6]. He divides the test process intro seven steps. The process has been designed to be used by both developers and an independent test team. Since the details of the process activities are very generic in nature, the process must be customized by organization before its actual use.

Figure 2.5 gives an overview of the proposed process. It follows the V concept of development/testing. The seven steps as given in [Perry, 2006, p. 157] are being summarized below.

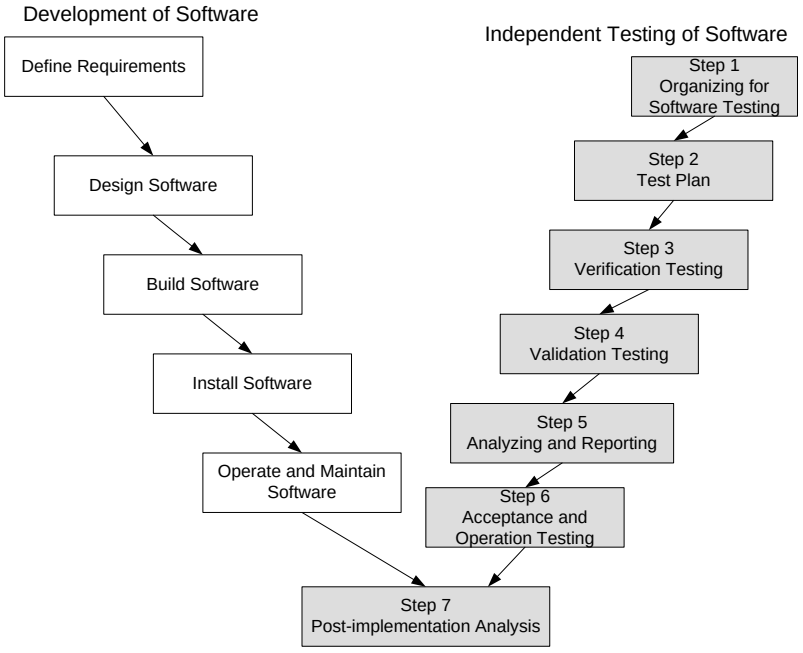


**Figure 2.5:** *V-Diagram for Seven Step Test Process [Perry, 2006]*

1. **Organizing for testing:** This is a kind of preparation step which is aimed at defining the scope of testing activities and responsibilities of whoever will be involved in testing process. Furthermore, the development plan must be analyzed for completeness and correctness which is the basis for the next step of test plan development.

2. **Developing the test plan:** After the preliminary steps, a test plan must be developed that precisely describes testing objectives. A test plan will mention exactly how and what kinds of testing activities will be performed. Possible risks should also be identified at this step.

3. **Verification testing:** The purpose of this step is verify activities and products of each of the design and development process to ensure that software is being constructed correctly. This will enable an early detection of defects before development is complete.

4. **Validation testing:** Dynamic testing of the code using the pre-established methods and tools should be performed now. This step should ensure that the software fulfills the stated requirements.

5. **Analyzing and reporting test results:** Test results should be analyzed to compare the developed product with the intended development goals. Results should be reported with the defect reports etc.

6. **Acceptance and operational testing:** A final step is the testing of the software by the actual users. Upon completion of the acceptance testing, the software must once again be tested in the production environment to observe conflicts or other faults.

7. **Post-implementation analysis:** This step is a kind of post-mortem analysis of the whole testing process. Efficiency and effectiveness of the testing process must be analyzed. This will help us identify lessons learned, and future improvement areas for the test activities.

***TMap-***
***Test Management Approach***

The Test Management Approach (TMap) [Pol et al., 2002] has been developed by a Dutch firm named Sogeti. The TMap approach primarily focuses on structured testing and provides answers to the *what, when, how, where*, and *who* questions of software testing [van Veenendaal and Pol, 1997]. Figure 2.6 gives an overview of TMap. It is founded on following four cornerstones,

L **Lifecycle**: This element sits at the heart of the approach a provides a development process related life cycle model for the testing activities. The lifecycle model is composed of planning, specification, execution, and completion phases under the umbrella of a planning & control phase. The model elaborates on guidelines on objectives, tasks, responsibilities, deliverables and related issues relevant to each of these phases.

T **Techniques**: This component compiles various techniques needed at each phase of the test lifecycle. The techniques cover issues such as defining test strategy, estimating test effort, studying test basis, and test specification etc. This component serves as a knowledge support to the lifecycle element.

I **Infrastructure**: Performing the lifecycle activities using the defined techniques needs some facilities and resources. The infrastructure element exactly addresses the provision of these elements. Two types of facilities fulfill this purpose, the test environment and test tools. The choice of the test environment depends upon the nature of the tests. TMap mentions a laboratory environment for white-box tests, a system test environment for black-box tests, and a production environment for acceptance tests. In case of test tools, TMap divides them according to the lifecycle phase where a test tool is needed or applied. The categories of tools recommended for the testing process include record & playback, comparators, test drivers, simulators, coverage analyzers, and static analyzers.

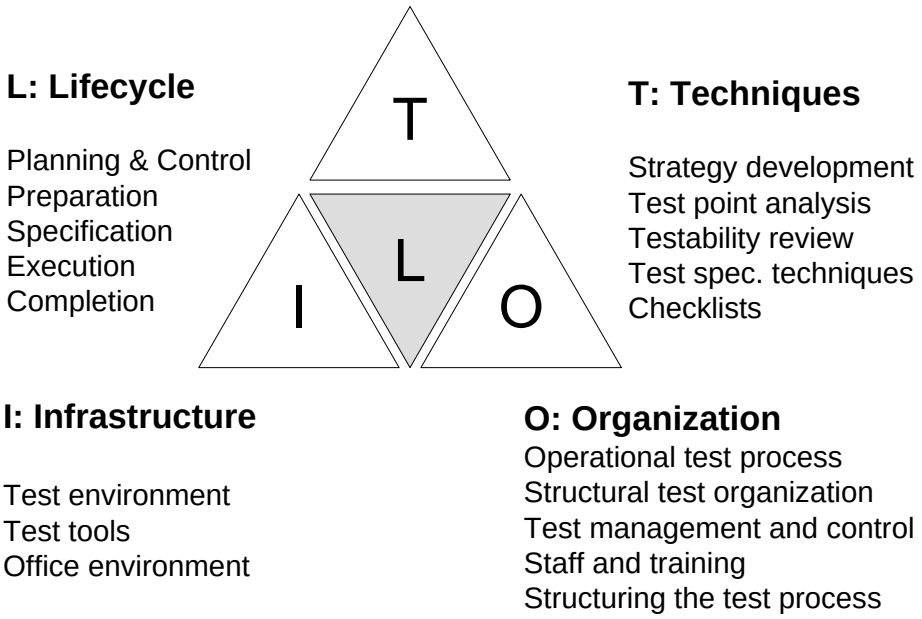


**Figure 2.6:** *Test Management Approach-TMap*

O **Organization**: Apart from necessity of managing technical aspects of establishing a testing process, TMap acknowledges the strong role of organizational support for the success of the process. Contention of interest, organizational structures and management methods, time and resource constraints and lack of expertise affect the testing process. To cope with these issues this cornerstone focuses on the operational test process, the structural test organization, test management, personnel and training, and structuring the test process.

***Drabick's formal testing process***

Drabick [Drabick, 2003] presents a task-oriented process model for formal testing intended for use on medium-to-large software-intensive programs. The model provides a concise framework of testing tasks to assist test engineers. The author of the approach assumes the model to be helpful in a number of ways, for example to

- manage defects
- create efficient test plans
- provide work breakdown structure for the test engineering function, and
- provide a basis for documenting testing processes.

The test process model is composed of a collection of Input-Process-Output (IPO) diagrams. Each IPO diagram lists inputs, process names, and relevant outputs. Figure 2.7 gives structure of the level 0 model for the formal testing. The description is very primitive in nature at this level. This level of detail is not much meaningful and is meant to present only a top-level picture of the test process.

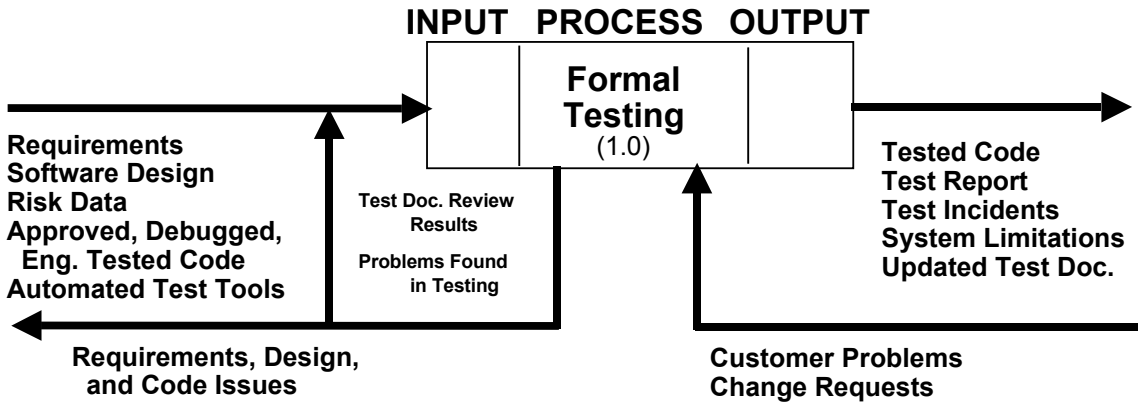


**Figure 2.7:** *Drabick's Formal Software Test Process-Level 0 IPO Diagram [Drabick, 2003]*

Figure 2.8 expands the level 0 description of the model into several sub-processes which are listed below. The proposed model further drills down to level 2 and 3 for each of these processes (which are not given here for the sake of brevity).

1. Extract test information from program plans
2. Create test plan
3. Create test design, test cases, test software, and test procedures
4. Perform formal test
5. Update test documentation

Although the process model contains several useful details of testing activities, yet it speaks nothing about the evaluation of the process itself. It provides no mechanism of evaluating how good the process has been performed or any other form of assessing effectiveness or efficiency of the activities performed.

***Test-driven development***

Agile software development is a conceptual framework for software development that promotes development iterations, open collaboration, and adaptability. Agile methods are development processes that follow philosophies of Agile manifesto and principles. Some examples of these methods include Extreme Programming (XP), Adaptive Software Development (ASD), Scrum, and Feature Driven Development (FDD) etc. Agility, change, planning, communication, and learning are common characteristics of these methods .

Extreme Programming (XP) is a well known and probably the most debated of the Agile methods. Two of the twelve practices of XP include *Test First* and *Refactoring*. The test first principle requires that automated unit tests be written before writing a single line of code to which they are going to be related. Test Driven Development (TDD) [Beck, 2002] has evolved from this test first principle. Although TDD is an integral part of XP but it can also be used in other development methods.

TDD is not a not a testing technique nor a testing method or a process, it is only a style of development. Under this approach software evolves through short iterations. Each iteration

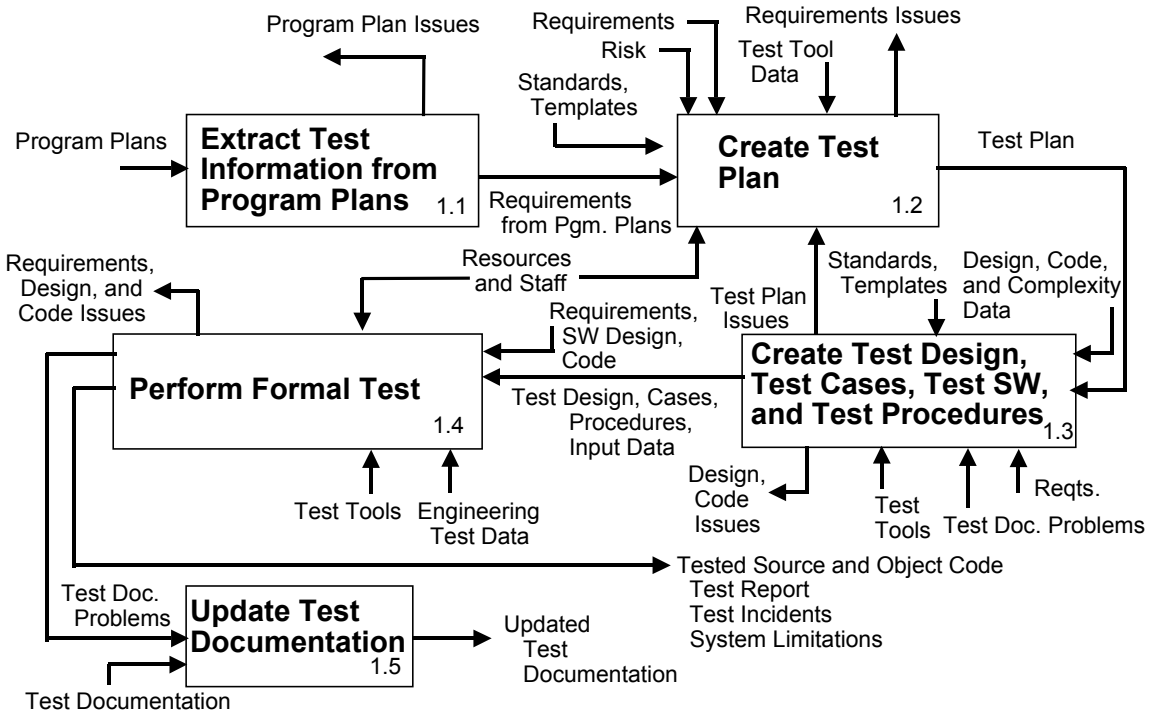


**Figure 2.8:** *Drabick's Formal Software Test Process-Level 1 IPO Diagram [Drabick, 2003]*

involves initially writing test cases that cover desired improvement or new functionality. Necessary code is then implemented to pass these tests and the software is finally refactored to accommodate changes. Test-driven development cycle consists of following sequence of steps; [Beck, 2002]

- **Quickly add a test:** A simple test is written as the first step which covers some aspect of functionality of code.
- **Run all tests and see the new one fail:** Running the test cases in absence of required code should essentially fail. This validates that the test harness is working correctly and that the new test does not mistakenly pass without requiring any new code.
- **Make a little change:** The next step is to implement some code that is just enough to pass the existing tests. This is meant to incrementally add functionality to the developed code.
- **Run all tests and see them all succeed:** If all tests now pass, the programmer can be confident that the code meets all the tested requirements.
- **Refactor to remove duplication:** Refactoring is the process of making changes to existing working code without changing its external behavior. This step removes cleans up the code and any duplication that was introduced getting the test to pass.
- **Repeat:** This test-code-refactor cycle is repeated which leads to an evolution of the whole program, where the program-units are developed gradually.

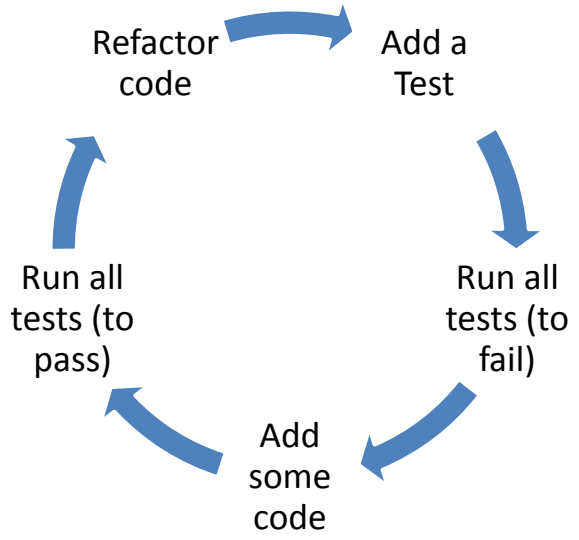


**Figure 2.9:** *Test-driven Development Cycle*

Figure 2.9 summarizes the TDD cycle. As in other conventional development and testing practices, testing under TDD is not done in a linear fashion. The continuous evolution and feedback that is obtained from running tests makes this method circular. Since its inception, a number of techniques and tools have been developed that support TDD style.

Improved quality, testability, extensibility and other benefits are believed to be associated with TDD style of development. Few empirical works have attempted to validate some of these claimed benefits[Siniaalto, 2006]. However certain TDD is limited in certain aspects too. First, it concentrates on automated unit tests to build clean code. It is a fact that not all tests can be automated such as user interface testing. Secondly, in database applications and those involving different network configurations full functional tests are a necessity. Test-first approaches for these kinds of applications are still missing. TDD's lack of proper functional specifications and other documentations also limit this style to small projects. There are some social factors such as developer's attitude and management support which will certainly be a hurdle in adoption of this evolutionary approach.

***Independent Verification & Validation***

Zero defect software is a highly sought goal for some particular kinds of safety critical and complex large applications. Sometimes managerial commitments, financial constraints and developer's or tester's bias may cause adverse affects on testing and may compromise software quality. According to IEEE, independent verification and validation (IV&V) refers to the verification and validation performed by an organization that is technically, managerially, and financially independent of the development organization. But whether IV&V differs from V&V in more than just the independence of its practitioners is still open to debate [Arthur et al., 1999].

IV&V activities have been found to help detect faults earlier in the software development life cycle, reduce the time to remove those faults, and produce a more robust products [Arthur et al., 1999]. The advantages of an independent V&V process are many. In particular, the independence in V&V [Arthur and Nance, 1996],

- provides an objective assessment of the product during its creation,
- adds a new analytical perspective not present in the development environment,
- brings its own set of tools and techniques to bear on ensuring development accuracy and validity,
- introduces "intermediate" users of the system who serve as "beta testers" before the product goes to market, and
- significantly enhances testing and the discovery of design flaws and coding errors.

Several software companies offer IV&V services. NASA's IV&V Facility is a well-known IV&V service provider for NASA's critical projects and missions. Analysis of IV&V approaches for different domains such as simulation and modeling and object-oriented software applications has been performed by various researchers.

#### 2.2.2.2 Domain Specific Test Processes

A very wide variety of software applications are being developed today, for example those for distributed systems, communication systems, and embedded systems etc. Type of the application domain naturally affects scope and range of software testing involved. Certain techniques and levels of testing may no longer be applicable, and new approaches to testing may be required. Testing activities and processes will also be affected. The next two sections will review testing process for embedded systems and service-oriented applications as well-known examples which require specialized testing methods.

***Test process for embedded software***

Many different types of embedded systems exist today such as mobile phones, electrical home appliances, railway signal systems, hearing aids and other health care systems, missile guidance systems, satellites, and space shuttles etc. Zero defect software is needed for such systems since a failure can cause human lives or extremely huge financial losses. Within this context, testing of embedded software becomes very complex and poses much more challenges and requirements on testing than that of other common software applications.

Numerous techniques and tools have been developed to answer specific testing concerns of embedded softwares. Instead of discussing individual techniques a testing method will be reviewed here which covers a wider perspective of embedded software in comparison to specific techniques or tools. The method is called TEmb. TEmb provides a mechanism for assembling a suitably dedicated test approach from the generic elements applicable to any test project and a set of specific measures relevant to the observed characteristics of the embedded system [Broekman and Notenboom, 2003, Ch. 2]. This method actually adapts the concepts of TMap [Pol et al., 2002] approach to the embedded software domain. Figure 2.10 gives an overview of the TEmb method.

The generic elements of the method involve descriptions of *lifecycle*, *techniques*, *infrastructure*, and *organization* issues. The second part of the method involves applying *measures* specific to the system context based on the analysis of *risks* and *system characteristics*. Example of these specific measures include specific test design techniques,

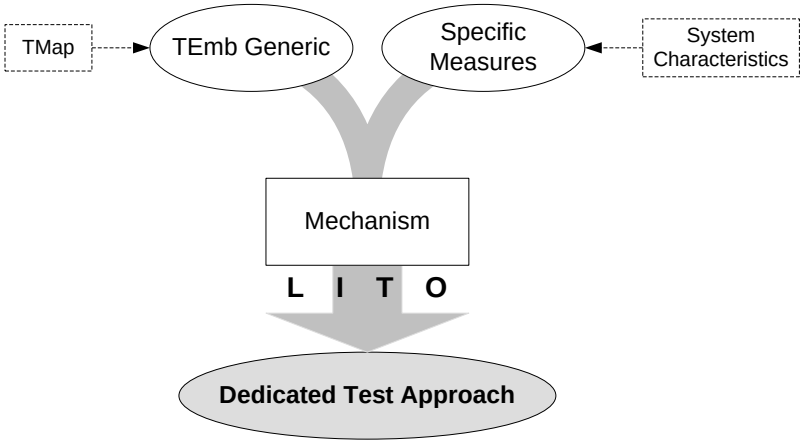


**Figure 2.10:** *TEmb:Test Process for Embedded Systems [Broekman and Notenboom, 2003]*

system modeling, dedicated test tools, and lifecycle etc [Broekman and Notenboom, 2003, p. 18].

***Test process for SOA systems***

Service-oriented computing represents a new generation distributed computing platform [Erl, 2007]. There are two view points to look at a service-oriented architecture (SOA) of applications. On the business side, SOA is a set of services providing some functionality utilized externally by customers or internally by other parts of the organization. This loose coupling of functionality provides support for evolving business requirements to be flexibly implemented. On the technical side, SOA is a hierarchy of services resembling the software components which implement some specific business task. Software systems are then composed of these services serving as primary building blocks which can be discovered, composed, instantiated, and executed at runtime. A generalized definition of SOA covering both these views as given by Bieberstein states that,

> **Definition:** *A service-oriented architecture is a framework for integrating business processes and supporting IT infrastructure as secure, standardized components–services–that can be reused and combined to address changing business priorities.* [Bieberstein et al., 2005, Ch. 1]

Service-oriented architecture (SOA) enables creation of enterprise-wide and cross-enterprise flexible, dynamic business processes and agile applications. The rigor and flexibility of SOA-based systems comes with a price and confronts us with unique challenges [Papazoglou et al., 2007].

One issue among these research areas is the quality assurance and testing of SOA systems. The widespread adoption of SOA-based solutions introduces rising concern for efficient and effective testing methods specific to service-oriented systems. Within the context of testing and quality assurance of SOA-based systems, it has been observed that focus of research has primarily been on developing new testing techniques for services, maturity models of SOA adoption, and SOA related measures etc. The author scanned the literature for articles, conference proceedings, and books related to SOA testing and found that out of the 128 identified sources, 103 presented new service testing techniques, 15 were

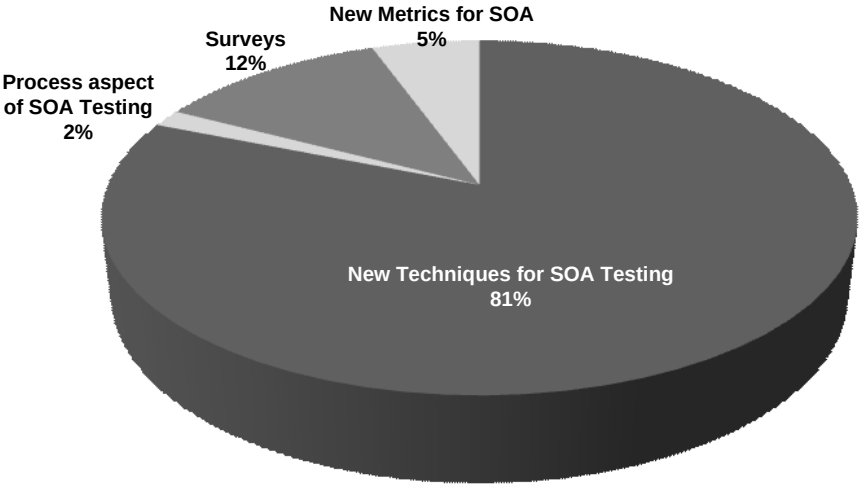


**Figure 2.11:** *Review of 127 Articles on SOA Testing*

about overview of SOA testing issues, new challenges and surveys, 7 discussed new SOA related measures, while only 3 references discussed process dimension of SOA testing. Figure 2.11 summarizes these facts. Thus, it appears that testing of services alone has been focused, until now, within the very broad scope of SOA testing. Perhaps it seems appropriate to believe that SOA testing has not yet received adequate attention as mentioned by [Ribarov et al., 2007].

**TPI SOA Model:**The discussion would be incomplete without mentioning a very recent endeavor [Eggink et al., 2008] for customization of the well-known process improvement model, TPI [Koomen and Pol, 1999], in the SOA context. In addition to making a few structural and textual changes to existing process areas, the model adds following key areas related to SOA testing,

- **Service Registry**:
  Exploitation of service registry for development and test process
- **SOA knowledge**:
  The existence and implementation of special testing knowledge in SOA context
- **Availability of test basis**:
  Existence of details about test objects to serve as test basis
- **Service integration**:
  The level of service testing if it has been modified
- **Quality management**:
  Extent of implementing quality management procedures for test process and SOA services

#### 2.2.2.3 Formal Approaches

Wang and King [Wang and King, 2000, p. 40] define a formal process model as a model that describes the structure and methodology of a software process system with an algo-

rithmic approach or by an abstractive process description language. Formal approaches to software process have been variably applied. Dumke et al. [Dumke et al., 2006a] mention few of such approaches. The same concept has been used in the domain of testing process. The next two sections explain these approaches.

***Model based testing***

A major portion of software testing costs is associated with test case related activities. Test case generation consumes resources such as for their planning, design, and execution. Manual design and execution of test cases is a tedious task. Therefore, automation of test case generation and execution could be an interesting mechanism to reduce the cost and effort of testing. Automatic *execution* of tests is offered by many automated test tools. Model based testing (MBT) [Utting and Legeard, 2006] takes a step forward to automate the *design* process of test cases.

MBT involves creating an abstract model of the system under test which is mostly based on functional requirements. Then a test tool automatically generates test cases from this model of the system. A direct benefit is that overall test design time is reduced and a variety of test cases can be generated from the same model simply by changing test selection criteria. MBT is supposed to offer many benefits such as shorter schedules, lower cost and effort, better quality, early exposure of ambiguities in specification and design; capability to automatically generate many non-repetitive and useful tests, test harness to automatically run generated tests, and convenient updating of test suites for changed requirements [El-Far and Whittaker, 2001]. Utting and Legeard [Utting and Legeard, 2006, p. 27] divide MBT into following five steps,

- **Model:** The very first step is to create an abstract model which describes behavior of the system under test (SUT). This model is abstract in the sense that it mostly covers key aspects of the SUT. Some design language or a test specification language must be used to create this model. Unified Modeling Language (UML), TTCN-3 [1], or Test Modeling Language (TML) [Foos et al., 2008] can be used for this purpose.

- **Generate:** The next step is to generate abstract tests from the model. An automated test case generator tool can be exploited at this step. To reduce the almost infinitely possible test cases, a test selection criteria must be used. In addition to a set of abstract test cases, this step sometimes also produces a requirements traceability matrix and a model coverage report.

- **Concretize:** The abstract test cases from the previous step cannot be executed directly on the SUT. They must be transformed into executable form which is done under this step. A test script generator tool may be used for the purpose.

- **Execute:** This step executes the concrete test cases over the system under test (SUT) with the help of a test execution tool. The step produces the final test results. With *online testing*, the above three steps are merges and tests are executed as they are produced. In case of the *offline testing*, the above three steps will be performed as described.

[1] http://www.ttcn-3.org/

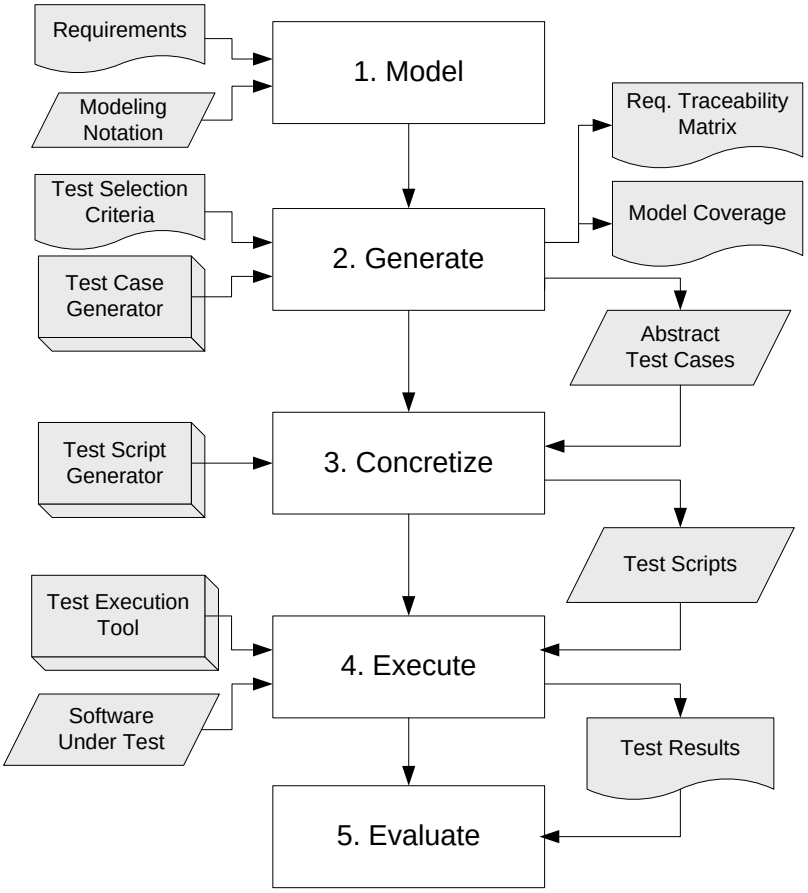


**Figure 2.12:** *Model-based Testing Process*

- **Analyze:** The final step is to analyze the test results. Actual and expected outputs are compared and failure reports are analyzed. The step also involves deciding whether to modify the model, generate more test cases, or stop testing.

Figure 2.12 gives a detailed description of the MBT process with necessary inputs and outputs of each step.

Hundreds of MBT approaches have been developed to date. However, they are not aimed at covering all testing aspects. MBT techniques mainly aimed at functional testing since test cases are derived from functional specification of the system. Only in very few cases have the MBT approaches been used for testing some non-functional characteristics. Furthermore, MBT is a kind of black-box approach since the system model has been derived from the behavioral descriptions. However, MBT can be applied at any testing level (although it has mostly been applied for system level tests). Figure 2.13 summarizes the scope of MBT with reference to different testing aspects.

A comprehensive characterization of these techniques has been given by Neto et al. [Neto et al., 2007]. MBT techniques differ by behavioral model, test generation algorithm, test levels, software domain, or level of automation etc. Choice of a particular MBT approach out of the many can influence efficiency of the overall test process.

***Cangussu's formal models***

A mathematical model of a software process attempts to describe its behavior and pro-

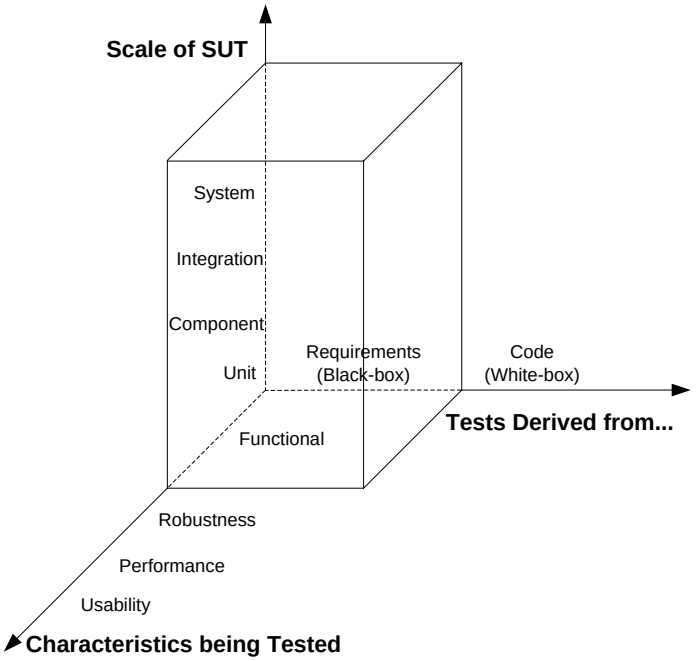


**Figure 2.13:** *Scope of Model-based Testing [Utting and Legeard, 2006]*

vides a feedback mechanism which guides the managers in adjusting model parameters to achieve desired quality objectives. The generic procedure to select, adopt and apply these kinds of models can be summarized in following steps

1. Postulate general class of models
2. Identify model to be tentatively entertained
3. Estimate model parameters
4. Perform diagnose checking (model validation)
5. Use model for prediction or control

Several mathematical models of software test process have been developed by Cangussu et. al[Cangussu, 2002], [Cangussu, 2003]. These mathematical models attempt to predict some aspect of the software test process (with special focus on system test phase) such as effort, schedule slippage, failure intensity or effect of learning etc. Most of these approaches follow a feedback control mechanism as outlined in the figure 2.14.

Now we briefly describe each of Cangussu's approaches one by one.

- **State variable model** [Cangussu et al., 2000]
  This model uses the theory of state variables to capture the dynamic behavior of the software test process by focusing on time and effort required to debug the software. It then applies feedback control for adjusting the variables such as work force and quality of the test process to improve the test process performance, and meeting the deadlines. This model has been validated with data from two large industrial projects.

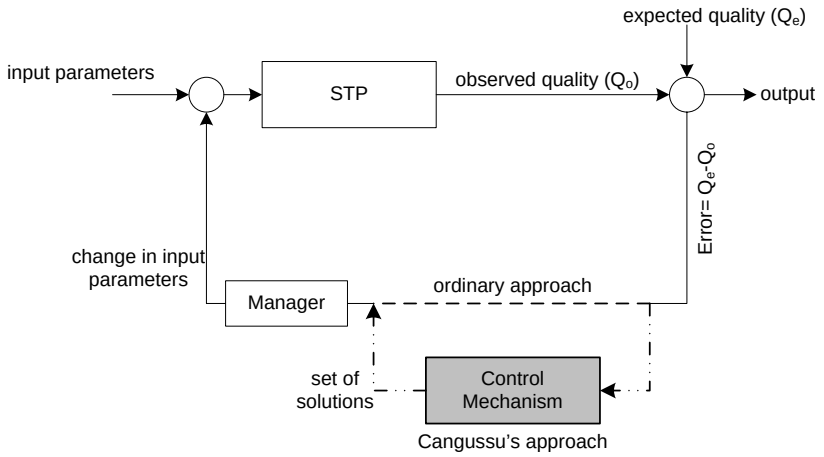


**Figure 2.14:** *Cangussu's Approach of STP Models [Cangussu, 2002]*

- **A state model** [Cangussu et al., 2001a]
  This model attempts to predict completion time and cost to perform software test process. The model provides an automated method for parameter identification. The closed-loop feedback mechanism consisting of determination (based on adjustment of different parameters) of minimum decay rate needed to meet management objectives guides the managers to correct deviations in the software test process.

- **Feedback control model** [Cangussu et al., 2001b]
  Feedback control model is quite similar to formal and state models. It differs only in control variables which in this case are product reliability and failure intensity. These variables are calculated at specific checkpoints within the software test process and result is fed back to the controller to adjust model parameters to meet desired process objectives.

- **A formal model** [Cangussu et al., 2002]
  Current formal model of the software test process is based on the theory of process control. Estimations of the number of remaining errors and schedule slippage are performed at specific checkpoints inside a feedback control structure which helps meet the schedule and quality requirements.

- **Stochastic control model** [Cangussu, 2003]
  The stochastic control model is a variation of state variable model and formal model of the software test process discussed above. This model is designed to account for foreseen and unforeseen disturbances and noise in the data collection process. The model has been verified with some simulation results while still needs validation with actual project data.

- **A quantitative learning model** [Abu et al., 2005]
  This model is also derived from the formal model of the software test process described above. This approach investigates the effect of learning behavior and experience to improve the software test process. Prediction process is improved by

adjusting different model parameters such as initial knowledge and learning rate. The model has been validated with two large industrial case studies.

Some general aspects of concern about such mathematical models are:

- *Model Validation*: Usually these kinds of models are validated through simulation runs, analytical approaches, or empirical investigations and industrial case studies. The models outlined above have been validated through simulation and same two case studies applied to each of these model evaluations. We still need more empirical studies on these models to highlight any new aspects of model behavior and effect of different model parameters.

- *Prediction Quality*: An evaluation of above mentioned mathematical models involves assessment of their prediction quality. Apel [Apel, 2005] mentions some criteria to evaluate prediction quality of such mathematical models.
  - *Prediction Accuracy* answers the question how accurate is the prediction.
  - *Prediction Distance* determines how far in future does the prediction lie.

  The models mentioned above need to be evaluated in the light of these criteria. The only related evaluation reported by authors in this regard is a sensitivity analysis [Cangussu et al., 2003] of the state variable model discussed above. This analysis attempts to quantify effects of parameter variations on the behavior of the model such as its performance.

- *Practical Application/Industrial Acceptance*: The mathematical complexity involved in construction and application of such models may be difficult to be handled by process managers who usually do not have enough background in such areas. In this case, a tool encapsulating mathematical procedures may simplify adoption of these models in industry.

### 2.2.3 Test Process Evaluation & Improvement

***Evaluation models of software process vs. test process***

Evaluation and improvement of software test process is strongly motivated by and borrows common concepts from that of the software process. A large number of methods over assessment and measurement techniques for generic software processes have been developed over the years. Surveys of current software process quality models given in [Komi-Sirviö, 2004, Ch. 3], [Zahran, 1998] highlight the directions of research in software process improvement. Some of these research directions were also followed by researchers working on evaluation and improvement of test process. Analogously, a few surveys have tried to summarize the corresponding models of test process evaluation and improvement [Swinkels, 2000], [Farooq and Dumke, 2007], [Farooq and Dumke, 2008b]. Combining both these surveys a broad picture of available approaches for software process and test process has been developed and given in table 2.2. This discourse could be helpful in establishing connections between the two classes of these models.

**Table 2.2:** *Software Process vs. Test Process Research*

| Model Type | Software Process | Test Process |
|---|---|---|
| Management | Deming's Cycle<br>QIP<br>IDEAL Model<br>ISO 15504 Part 7 | TMap |
| Best Practices | CMMI<br>Bootstrap<br>SPICE<br>ISO 9000-3 | TMM<br>TPI<br>TMMi<br>IEEE Std. V&V<br>IEEE Std. Unit Testing |
| Measurement | SPC<br>GQM<br>PSP | Cangussu's Mathematical Models |
| Product Quality | ISO/IEC 25000<br>IEEE Std. 1061 | – |
| Knowledge Management | Experience Factory (EF) | – |

The categories of the approaches such as management, best practices, etc mentioned in the table have been take from [Komi-Sirviö, 2004]. Since the main direction of this thesis is evaluation and improvement of test process, discussions of the models of test process summarized in table 2.2 needs a detailed analysis. Therefore, such discussions will be deferred until a subsequent chapter (4) which dedicates itself for a meticulous descriptions of these approaches.

## 2.3 Testing Techniques

Within the area of software testing, the terms *technique* and *method* are interchangeably used. Ideally speaking a test technique refers to the way test engineers design or select test cases against which the program is to be tested. In this situation, a technique is in fact some kind of dynamic technique which involves actual code execution. On the other hand, the process of testing may initially and partially be done without executing the program code. In this case, these are called static techniques or rather testing methods. These testing methods (or techniques) define different procedures to find defects in the software design and source code. For the sake of simplicity and ease of understanding, this thesis therefore does not distinguish between the terms *testing technique* and *testing method*.

***Classification of techniques***

Software testing literature contains a rich source of testing techniques, for example Beizer [Beizer, 1990], Perry [Perry, 2006, Ch. 17], Liggesmeyer [Liggesmeyer, 2002], Tian [Tian, 2005, Ch. 8-11], Pezze and Young [Pezzè and Young, 2007]. These techniques are aimed at solving a variety of problems or needed at different testing stages.

Avoiding even the list of such huge collection of testing techniques, only their broad classes will be mentioned here. It is evident that any uniform classification scheme for testing techniques is not possible, and only some pseudo classifications exist such as Abran [Abran et al., 2004], Juristo[Juristo et al., 2004a], Tian[Tian, 2005], and Liggesmeyer [Liggesmeyer, 2002, p. 34] etc. These classifications vary by the way they organize techniques such as based on how tests are designed, the purpose of the technique, or the testing phase or level etc. Liggesmeyer's classification seems to be quite comprehensive in combining few of these criterion and in covering available techniques. Figure 4.9 is a modified version of his classification.

### 2.3.1 Static techniques

Static testing techniques are usually applied at the initial steps in software testing. These are verification techniques which do not employ actual execution of code/program. These techniques attempt to ensure that organizational standards and guidelines for coding and design are being followed. Formal verification, inspection, reviews, and measurement are main types of static techniques. Table 2.3 presents an abridged summary of static testing techniques.

**Table 2.3:** *Summary of Static Testing Techniques*

| Category | Technique | Description |
|---|---|---|
| Verification | Formal Verification | Analyzes correctness of software systems based on their formal specification. |
| | Symbolic Verification | Program is executed by replacing symbolic values in place of original program variables to provide general characterization of program behavior. |
| Analysis | Measurement | Provides quantitative view of various attributes of testing artifacts. |
| | Review | A work product is examined for defects by individuals other than the producer. |
| | Inspection | Disciplined engineering practice for detecting and correcting defects in software artifacts |
| | Walk-through | The producer describes the product and asks for comments from the participants. |
| | Audit | An independent examination of work products to assess compliance with specifications, standards, or other criteria. |
| | Slicing | Technique for simplifying programs by focusing on selected aspects of semantics for debugging. |

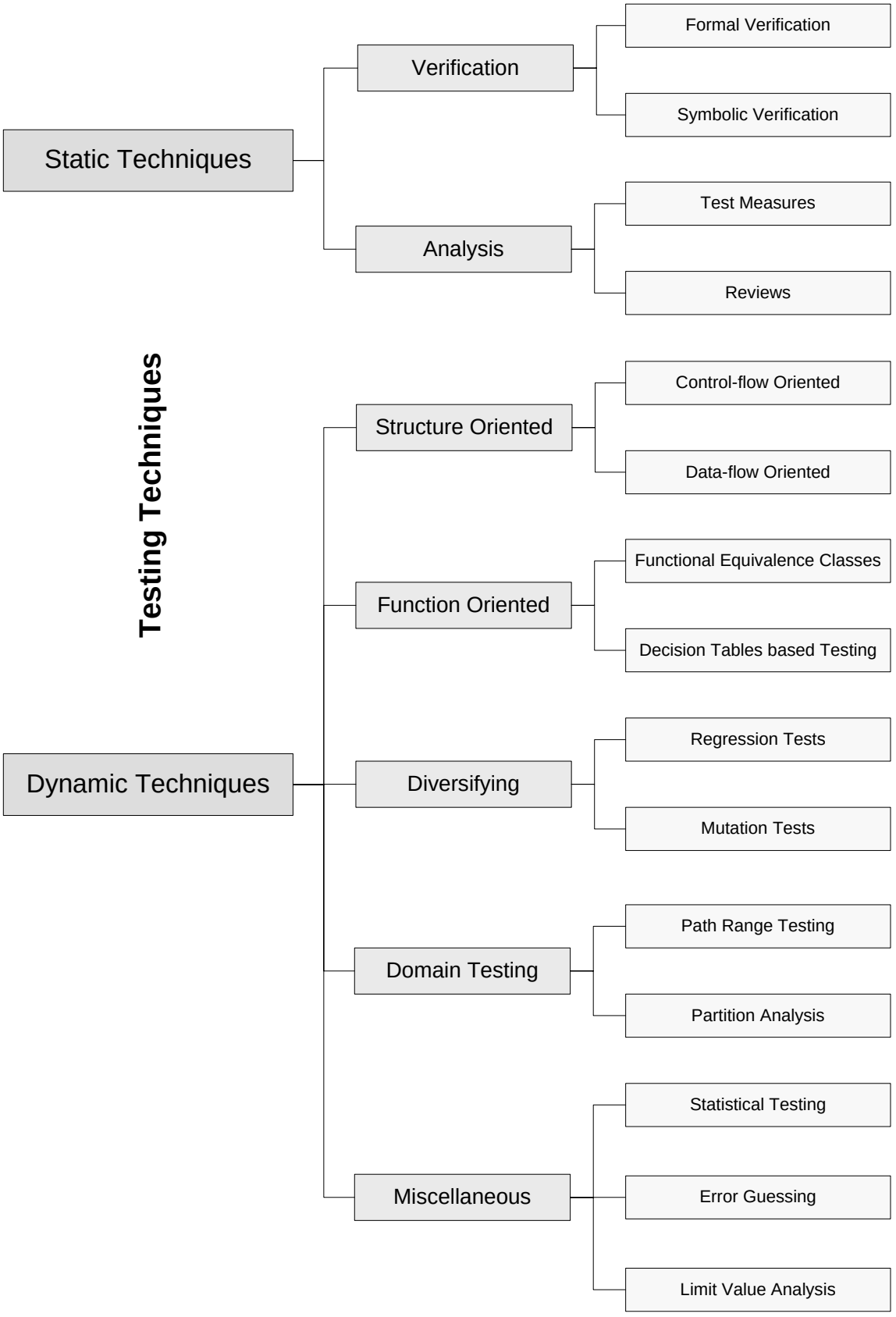


**Figure 2.15:** *Liggesmeyer's Classification of Testing Techniques*

#### 2.3.1.1 Verifying

Formal specifications is a way to precisely describe customer requirements, environmental constraints, and design intentions to reduce the chances of common specification errors. Verifying techniques check the conformance of software design or code to such formal specifications of the software under test. These techniques are mainly focused on investigating functional requirements and aspects such as completeness, clarity, and consistency. Only a few of some well known techniques of this type will be discussed below.

***Formal verification***

Formal verification is the use of mathematical techniques to ensure that a design conforms to some precisely expressed notion of functional correctness. Software testing alone cannot prove that a system does not have a certain defect, neither can it prove that it does have a certain property. The process of formal verification can prove that a system does not have a certain defect or does have a certain property. Formal verification offers rich toolbox of mathematical techniques such as *temporal-logic model checking*, *constraint solving* and *theorem proving* [Lüttgen, 2006]. Clarke [Clarke and Wing, 1996] mentions two well established approaches to verification: *model checking* and *theorem proving*. Two general approaches to model checking are *temporal model checking* in which specifications are expressed in a temporal logic and systems are modeled as finite state transitions while in second approach the specification is given as an automaton then the system, also modeled as an automaton, is compared to the specification to determine whether or not its behavior conforms to that of the specification [Clarke and Wing, 1996]. One of the most important advances in verification has been in decision procedures, algorithms which can decide automatically whether a formula containing Boolean expressions, linear arithmetic, enumerated types, etc. is satisfiable [Heitmeyer, 2005].

***Test process for symbolic testing***

Symbolic testing [Pezzè and Young, 2007, Ch. 19] or symbolic execution is a program analysis technique in which a program is executed by replacing symbolic values in place of original program variables. This kind of testing is usually applied to selected execution paths as against formal program verification. Symbolic execution gives us a general characterization of program behavior which can help us in designing smarter unit tests [Tillmann and Schulte, 2005] or in generating path-oriented test data. Figure 2.16 gives an example of symbolic execution. Although this technique was developed more than three decades before, it has only recently become practical with hardware improvements and automatic reasoning algorithms.

#### 2.3.1.2 Analyzing

Analyzing techniques attempt to find errors in software without executing it. However these techniques are not just limited to checking software entities but also involve reviewing designs and relevant documents. The main premise behind these techniques is that an earlier detection of bugs in software is less expensive than finding and fixing them at later development stages. These techniques analyze requirements, specifications, designs, algorithms, code, and documents. Examples of these techniques are;

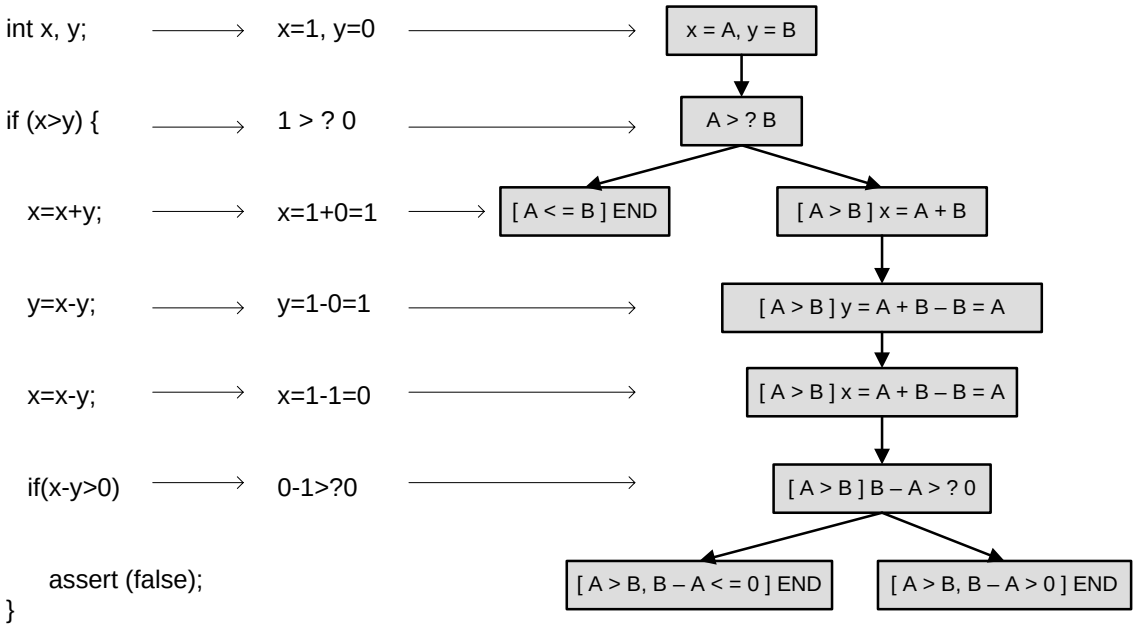


**Figure 2.16:** *An Example of Symbolic Execution*

- test measurements
- inspections
- reviews
- walk-throughs
- audits

***Test measures***

Measurement is a static analysis technique which can give us valuable information even before actually executing dynamic tests. Size, effort, complexity, and coverage like information can readily be obtained with the help of numerous test metrics. A detailed review of test related metrics will be given later in this thesis.(see section 4.2.5)

***Software reviews, inspections and walk-throughs***

Software review as defined by IEEE [IEEE, 1990] is a process or meeting during which a work product or set of work products are presented to project personnel, managers, users, customers, or other interested parties for comment or approval. IEEE standard [IEEE, 1997a] which defines requirements for software reviews describes five types of reviews as *management reviews, technical reviews, inspections, walk-throughs*, and *audits*.

Reviews are usually performed for code, design, formal qualification, requirements, and test readiness etc. Since it is virtually impossible to perform full software testing, reviews are used as an essential quality control technique. It is a common belief that reviews increase the quality of the software product, reduce rework and ambiguous efforts, reduce testing and defines test parameters, and are a repeatable and predictable process [Lewis, 2004].

***Fagan Inspections***

Fagan inspection refers to a structured process of trying to find defects in development documents such as programming code, specifications, designs and others during various

phases of the software development process. In a typical Fagan inspection the inspection process consists of the operations shown in figure 2.17.

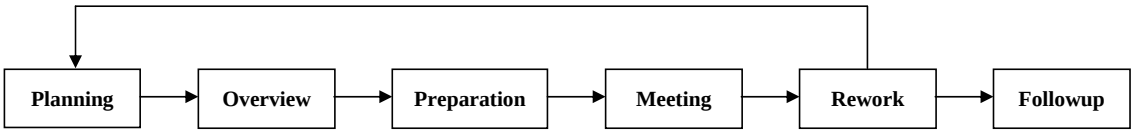


**Figure 2.17:** *Fagan Inspection Basic Model [Fagan, 1986]*

Surveys, state-of-the-art studies, and future research directions within software reviews and inspections have been given by in [Laitenberger, 2002], and [Kollanus and Koskinen, 2007]. Another very recent industrial practice survey of software reviews was performed by [Ciolkowski et al., 2003]. The authors concluded that "companies conduct reviews regularly but often unsystematically and full potential of reviews for defect reduction and quality control is often not exploited adequately".

A recent case study to judge effectiveness of software development technical reviews (SDTR) [Sauer et al., 2000] has concluded that the most important factor in determining the effectiveness of SDTRs is the level of expertise of the individual reviewers. Additionally, this study highlights three ways of improving performance: selection of reviewers who are expert at defect detection; training to improve individuals' expertise; and establishing group size at the limit of performance. Another study [Laitenberger et al., 1999] reports similar results and rates preparation effort as the most important factor influencing defect detection capability of reviews.

## 2.3.2 Dynamic techniques

Dynamic testing techniques involve tests which employ system operation or code execution. Two broad categories of such dynamic methods exist, structural-based and functional-based. Dynamic techniques that exploit the internal structure of the code are known as structural, white-box, glass-box or coverage based tests. In contrast, those that do not involve the internal structure of the code are known as functional, black-box, behavioral or requirement-based tests. These kinds of testing techniques in the coming sections. Table 2.4 presents a very short summary of dynamic testing techniques.

### 2.3.2.1 Structure oriented

Types of testing techniques under this category exploit structural information about the software to derive test cases as well as their coverage and adequacy. In this context, data and control element are two main elements in any computation or information processing task that are grouped through some implemented algorithms. Structural testing techniques [Pezzè and Young, 2007, Ch. 12] are mainly based on this control-flow and data-flow information about our code design.

***Control-flow oriented***

Control-flow testing focuses on the complete paths and the decisions as well as interactions along these execution paths. Control flow elements that may be examined are statements, branches, conditions, and paths. These elements are also generally considered for coverage

**Table 2.4:** *Summary of Dynamic Testing Techniques*

| Category | Technique | Description |
|---|---|---|
| Structured oriented | Data-flow oriented | Select test cases based on program path to explore sequences of events related to the data state. |
| | Control-flow oriented | Select test cases using information on complete paths and the decisions as well as interactions along these execution paths |
| Function oriented | Functional equivalence classes | Input domain of the software under test is partitioned into classes to generate one test case for each class. |
| | Decision tables | Select test cases exploiting information on complex logical relationships between input data. |
| | Cause-and-effect graphs | Causes and effects in specifications are drawn to derive test cases. |
| | Syntax testing | Test cases are based on format specification obtained from component inputs. |
| Diversifying | Regression testing | Selective retesting of a system to verify that modifications have not caused unintended effects. |
| | Mutation testing | Works by modifying certain statements in source code and checking if test code is able to find the errors. |
| | Back-to-back testing | For software subject to parallel implementation, it executes tests on similar implementations and compares the results. |
| Domain Testing | Partition analysis | Compares a procedure's implementation to its specification to verify consistency between the two and to derive test data. |
| Miscellaneous | Statistical testing | It selects test cases based on usage model of the software under test. |
| | Error guessing | Generate test cases based on tester's knowledge, experience, and intuition of possible bugs in the software under test. |

criteria. For most computation intensive applications, which cover most of the traditional software systems, mere state and link coverage would not be enough because of the interconnected dynamic decisions along execution paths [Tian, 2005]. Therefore, control-flow testing is generally a necessary step among the variety of testing techniques for such systems.

***Data-flow oriented***

Data-flow testing [Pezzè and Young, 2007, Ch. 13], [Beizer, 1990, Ch. 5] is based on principle of selecting paths through the program's control flow in order to explore sequences of events related to the status of data objects, for example, pick enough paths to assure that every data object has been initialized prior to use or that all defined objects have been used for something. It attempts to test correct handling of data dependencies during program execution. Program execution typically follows a sequential execution model, so we can view the data dependencies as embedded in the data flow, where the data flow is the mechanism that data are carried along during program execution [Tian, 2005]. Data flow test adequacy criteria improve over pure control flow criteria by selecting paths based on how one syntactic element can affect the computation of another.

#### 2.3.2.2 Function oriented

IEEE [IEEE, 1990] defines function oriented testing or black-box testing as:

- *Testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions.*
- *Testing conducted to evaluate the compliance of a system or component with specified functional requirements.*

This type of testing does not exploit any knowledge about inner structure of the software. It can be applied towards testing of modules, member functions, object clusters, subsystems or complete software systems. The only system knowledge used in this approach comes from requirement documents, specifications, domain knowledge or defect analysis data. This approach is specifically useful for identifying requirement or specification defects. Several kinds of functional test approaches are in practice such as,

- decision tables
- functional equivalence classes
- domain testing
- transaction-flow based testing
- array and table testing
- limit testing
- boundary value testing
- database integrity testing
- cause-effect analysis
- orthogonal array testing
- exception testing
- random testing

Out of these, only a few commonly used techniques will be discussed in the coming sections.

***Functional equivalence classes***

This technique is used for minimizing the test cases that need to be performed in order to adequately test a given system. It produces a partitioning of the input domain of the software under test. The finite number of equivalence classes that are produced allow the tester to select a given member of an equivalence class as a representative of that class and the system is expected to act the same way for all tests of that equivalence class. A more formal description of equivalence classes has been given by Beizer [Beizer, 1995]. While Burnstein [Burnstein, 2003] regards derivation of input or output equivalence classes mainly a heuristic process, Myers [Myers, 2004] suggests some more specific conditions as guidelines for selecting input equivalence classes.

***Cause-and-effect graphing analysis***

Equivalence class partitioning does not allow combining conditions. Cause-and-effect graphs can be used to combine conditions and derive an effective set of test cases that may disclose inconsistencies in a specification. Based on some empirical studies, Paradkar [Paradkar, 1994] relates some experiences of using cause-effect graphs for software specification and test generation. He finds it very useful in reducing the cardinality of the required test suite and in identifying the ambiguities and missing parts in the specification. Nursimulu and Probert [Nursimulu and Probert, 1995] point out ambiguities and some known drawbacks to cause-effect graphing analysis.

***Syntax testing***

Syntax testing [Beizer, 1995], [Liggesmeyer, 2002], also called grammar-based testing, is a testing technique for testing applications where the input data can be described formally. Some example domains where syntax testing is applicable are GUI applications, XML/HTML applications, command-driven software, scripting languages, database query languages and compilers. According to Beizer [Beizer, 1995], syntax testing begins with defining the syntax using a formal meta-language such as Backus-Naur form (BNF) which is used to express context-free grammars and is a formal way to describe formal languages is the most popular. Once the BNF has been specified, generating a set of tests that covers the syntax graph is a straightforward matter.

The main advantage with syntax testing is that it can be automated, easily making this process easier, reliable and faster. Tools exist that support syntax testing. Marquis et al. [Marquis et al., 2005] explain a language called SCL (structure and context-sensitive) that can describe the syntax and the semantic constraints of a given protocol, and constraints that pertain to the testing of network application security. Their method reduces the manual effort needed when testing implementations of new (and old) protocols.

#### 2.3.2.3 Diversifying

The diversifying test techniques pursue quite different goals. Diversifying test techniques do not serve in contrast to the structure-oriented or function-oriented test techniques. A goal of the diversifying test techniques is to sometimes avoid the often hardly possible evaluation of the correctness of the test results against the specification. Differ-

ent types of diversifying techniques are back-to-back test, mutation test, and regression tests [Liggesmeyer, 2002]. Only regression testing, being probably the most widely researched technique in this category, will be discussed next.

***Regression testing***

Regression testing is defined by IEEE [IEEE, 1990] as selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements. Regression tests may apply at any level of testing such as unit tests etc to confirm no undesired changes have occurred during functional improvements or repairs. The main issues in regression testing include a) removal of redundant and obsolete test cases and b) test case selection to reduce cost and time of retesting.

The new version of software involves structural or other changes to modules which renders some of the previous test cases non-executable. Redundant test cases are those that are still executable but are irrelevant with rest to testing criteria. Re-executing all test cases other than obsolete and redundant affects regression testing complexity, effort and cost. We must select a suitable subset of these test cases. A number of techniques exist which attempt to reduce the test suite in this case. Some of these approaches are;

- Test case prioritization
- Test case selection
  - Code based
  - Specification based
  - Control-flow based
  - Data-flow based
  - Random sampling

Several regression testing techniques exist for specific problem situations. Muccini et al. [Muccini et al., 2005] explore how regression testing can be systematically applied at the software architecture level in order to reduce the cost of retesting modified systems, and also to assess the regression testability of the evolved system. Few other recently developed regression testing techniques include a scenario-based functional regression testing [Paul, 2001], regression testing for web-applications based on slicing, agile regression testing using record & playback, and regression testing technique for component-based software systems by enhancing change information etc.

**Regression test selection and prioritization:** Rothermel et al. [Rothermel et al., 2001] analyze few techniques for test case prioritization based on test case's code coverage and ability to reveal faults. Their analysis shows that each of the prioritization techniques studied improved the rate of fault detection of test suites, and this improvement occurred even with the least expensive of those techniques. Harry Sneed [Sneed, 2004] considers a problem which arises in the maintenance of large systems when the links between the specification based test cases and the code components they test are lost. It is no longer possible to perform selective regression testing because it is not known which test cases to run when a particular component is corrected or altered. To solve this problem, he proposes applying static and dynamic analysis of test cases.

**Analysis of regression test techniques:** Several other research works have performed cost-benefit or effectiveness analysis of regression test selection techniques. These include [Graves et al., 2001] and [Rothermel et al., 2004]. These studies reveal that very few safety-based regression test selection techniques exist as compared to coverage-based techniques. Although the safety-based techniques were most effective in detecting faults, yet such techniques could not considerably reduce the test suite. The minimization techniques produced smallest and least effective set suites while safe and data-flow techniques had nearly equivalent behavior in terms of cost effectiveness.

#### 2.3.2.4 Domain Testing

Selection of appropriate test data from input domain maximizing fault detection capability and minimizing costs is one major problem in black-box test design approach. Domain testing [Liggesmeyer, 2002, p. 190] attempts to partition the input domain and to select best representatives from these partitions to achieve these goals. Path analysis, partition testing, and random testing are usually used to short-list test data in domain testing.

Much research effort has been devoted to comparative analysis of these different domain testing approaches and varying opinions have been held by researchers. According to Gutjahr [Gutjahr, 1999], "in comparison between random testing and partition testing, deterministic assumptions on the failure rates systematically favor random testing, and that this effect is especially strong, if a partition consists of few large and many small sub-domains". He maintains that partition testing is better at detecting faults than random testing. In a later work, Ntafos [Ntafos, 2001] conclude that although partition testing generally performs better than random testing, the result can be reversed with a little addition in number of test cases.

## 2.4 Testing Tools

***Why test tools?***

With the growth in size, maturity of practices, and increased workload, software organizations begin to feel a need for automating (some of the) testing procedures. A test tool is defined to be an automated resource that offers support to one or more test activities, such as planning and control, specification, constructing initial test files, execution of tests, and analysis [Pol et al., 2002, p. 429] etc. Supporting the testing process with tools can possibly increase the efficiency of test activities, reduce the effort required for executing routine test activities, improve the quality of software and the test process, and provide certain economic benefits. In summary, test tools automate manual testing activities thereby enabling efficient management of the testing activities and processes. But the level of test automation depends upon many factors such as type of application under development, testing process, and type of development and target environment. One hundred percent automatic testing has been regarded as a dream of modern testing research by Bertolino [Bertolino, 2007].

***Is a tool inevitable?***

The first thought that should concern someone while considering a tool implementation is determining whether it is really inevitable that a tool should be used. If the answer is

positive, one must then look around for resources where we can find some appropriate tools. With the advancement of research and technology we expect to come across a lot of tools of different kinds. At this stage it will be interesting to organize this list in some fashion which could facilitate one in grasping an overview of available tools.

Despite the long list of possible benefits expected of test tools, it is not wise to instantly start using a tool in all kinds of testing problems. The decision to use a test tool warrants careful cost-benefit analysis. Some testing tools may be very expensive in terms of money and effort involved and an organization may even be doing well without application of a sophisticated tool. Different sets of circumstances exist which may encourage or discourage adopting a testing tool. Ramler and Wolfmaier [Ramler and Wolfmaier, 2006] analyze trade-off between automated and manual testing and present a cost model based on opportunity cost to help decide when tests should be automated. Some situations that motivate organizations in automating testing tasks include,

- Test practices are mature
- Large size of the software
- Large number of tests required
- Time crunch

In opinion of Lewis [Lewis, 2004, p. 321] the few circumstances where using a testing tools may not be a wise choice include,

- Lack of a testing process
- Education and training of testers
- Technical difficulties with tool
- Organizational issues
- Ad hoc testing
- Cost
- Time crunch
- Organizational culture

## 2.5 Summary

As part of the initial research phase, this chapter concentrated on a preliminary review of software testing area. The chapter has partially addressed the first sub-question of the research task. The fulfilled part of the research question has been *emphasized* as shown below,

RT-1. *Traverse the area of software testing to identify entities of evaluation.* Explore existing evaluation forms and approaches as applied in this field. Investigate characteristics of low-cost and explicit test evaluations.

This chapter opened with a brief introduction to the field of software testing. Serving as a quick scan of the area, a summary of research issues in software testing was given right in the beginning. Intending to cover breadth of knowledge in the software testing area, the chapter went through all the major branches of this area. Software testing process was discussed in the first place since this is the broader class of research problem of this thesis. A definition and explanation of the software testing process as followed in this thesis was given. Past and current directions of research in this sub-area were mentioned followed by an extensive coverage of all testing approaches which could be classed under the *testing process* category. The chapter postponed review of evaluation and improvement models of testing process which due to their size and special relevance to thesis deserve a separate chapter ( 4). Focus then shifted to the second major sub-area of testing techniques. Classifications of testing techniques along with summaries of significant static and dynamic techniques among them were given. The area of software testing techniques was found to be as much researched as the process aspect. Finally, the topic of test tools was examined. Rationale for using or implementing any test tools got deliberated. Evaluation aspects of these tools were discussed as well.

The treatment of the software testing area given in this chapter enables us to grasp the complexity and importance of evaluation in this field. The chapter has marked three areas of testing as key elements of interest for evaluations– processes, techniques, and tools. Existing evaluation methods for these testing aspects will be presented in later chapters (chapter 4 and 5). Below is a summary of observations made in this chapter.

Observations on Software Testing

- The all-time chief knowledge areas of software testing include techniques and methods of testing, evaluation of test effectiveness, and test tools.
- While testing stands for evaluating software product quality, evaluation of the testing itself for its effectiveness has been a topic of constant debate.
- Techniques, tools, and the testing process are significant elements of evaluation in the software testing area.

# 3 Theoretical Foundations of Evaluation

In an attempt to establish theoretical foundations behind evaluation of test processes, this thesis exploits some fundamental knowledge from a non-computing field; a subject area from the social sciences called *Evaluation*. This chapter explains some of the key concepts of this discipline and explores the necessary elements of a typical evaluation task. The relevance and application of this knowledge area to the field of software engineering is also summarized.

## 3.1 Introduction

The practice of evaluating something is a quite natural and almost spontaneous phenomena in humans and even among animals. Sitting at the lunch table we evaluate our food by its appearance, smell and taste, while taking a bath we judge temperature of the water before pouring it over our body, watching a movie we evaluate it if was good or bad, arriving at a place on an excursion trip we evaluate the atmosphere and weather against our aesthetic sense, and even when we throw some eatable before an animal it first sniffs and touches it before beginning to eat it. Unlike these kinds of primitive and involuntary evaluations that we make in our everyday lives, we need much more systematic efforts when it comes to evaluating more complex entities such as success of a project or of a social welfare program, or efficiency of an industrial production process. Well organized evaluation procedures are usually undertaken to evaluate individuals, programs, projects, policies, products, equipments, services, and organizations etc. The issues like goals and scope of such extended evaluations, the parameters and criteria to be considered, and techniques & processes to be applied are studied today under a well established discipline called *Evaluation*. What is precisely meant by evaluation has been described in several definitions of this term reflecting the approach followed. Only a few of these from the renowned subject matter gurus are given below.

> **Definition 1:** *Evaluation is the process of determining the merit, worth, or value of something, or the product of that process.* [Scriven, 1991]

> **Definition 2:** *Evaluation is the systematic assessment of the operation and/or the outcomes of a program or policy, compared to a set of explicit or implicit standards, as a means of contributing to the improvement of the program or policy.* [Weiss, 1998]

> **Definition 3:** *Evaluation is the systematic process of delineating, obtaining, reporting, and applying descriptive and judgmental information about some*

> *object's merit, worth, probity, feasibility, safety, significance, and/or equity.* [Stufflebeam and Shinkfield, 2007]

Under these definitions is hidden a complete art, science, and philosophy of all kinds of evaluative undertakings. The expansiveness of the field cannot even be summarized here, only some relevant concepts will be explained next.

### 3.1.1 Evaluation Concepts

***Program evaluation***

The term *evaluation* can refer to a range of evaluative works. However, in the context of social science, evaluation refers to a special kind of it called *program evaluation* which is perhaps the largest area studied and discussed in the evaluation literature. A program in this context is a collection of organized human efforts for the well being of the common people. Hence program evaluation is defined to be use of social research procedures to systematically investigate the effectiveness of social intervention programs [Rossi et al., 1999]. Although the origin of disciplined evaluation procedures cannot be precisely determined in history, the birth of modern program evaluation occurred around 1960s due to the initiation of several social welfare programs in the United States. Colossal amounts of money spent on such programs raised questions about their legitimacy, effectiveness, and success which could not be answered by the traditional unsystematic approaches. Governmental, political, managerial, and intellectual concerns about these programs mandated a need for systematic evaluations of these programs for their effectiveness, efficiency, and success.

***Types of evaluation***

Notable variations exist in evaluation approaches based on their goals and nature. Scriven's division of evaluations into two broad groups [Scriven, 1996b] was the first endeavor in this regard. He classified evaluations as formative (those performed during the program to improve it) and summative (those performed after the program is finished to assess its effectiveness). Apart from these large differences the two types differ over their intended use, methodology, and characteristics of results etc. Scriven's classification was challenged by Chen [Chen, 1996] who provided an extended view which based the decompositions on program stage (process or outcome) and evaluation functions (improvement or assessment). Chen's classification is given in table 3.1.

***Evaluation models***

Under the broad types of evaluation discussed above, there exist several sets of assumptions corresponding to different evaluation scenarios. The best ways to guide the planning and implementation in each of these evaluation situations are grouped in the form of different *evaluation models*. Stufflebeam [Stufflebeam, 2001] critically analyzed twenty two evaluation models with a view to deciding which one of them are worthy of continued application and which are best abandoned. The models that he reviewed are named below. Among the best of these program evaluation approaches include Client-centered, Constructivist, and Outcome assessment.

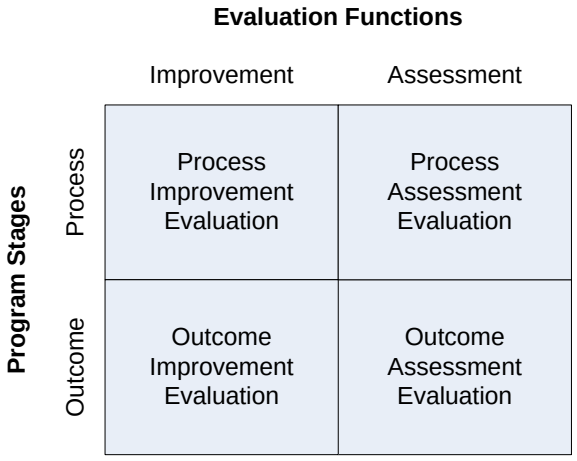


**Figure 3.1:** *Chen's Classification of Evaluation Types [Chen, 1996]*

- Public relations-inspired studies
- Politically controlled studies
- Objective testing programs
- Management information systems
- Benefit-cost analysis approach
- Decision-oriented studies
- Deliberative democratic evaluation
- Program theory-based evaluation
- Criticism and connoisseurship
- Consumer-oriented studies
- Utilization-focused evaluation
- Experimental studies
- Accountability studies
- Outcome evaluation
- Performance testing
- Clarification hearing
- Case study evaluations
- Accreditation approach
- Constructive evaluation
- Mixed-methods studies
- Objectives-based studies
- Client-centered studies

In a similar work, Posavac et al. [Posavac and Carey, 2003] reviewed thirteen such models, only the names of which are being listed here.

- Industrial inspection model
- Objectives-based evaluation
- Theory-driven evaluation
- Improvement-focused model
- Social science model
- Black box evaluation
- Goal-free evaluation
- Traditional model
- Fiscal evaluation
- Accountability model
- Expert opinion model
- Naturalistic model
- Empowerment model

***Evaluation theory***

One of the Merriam-Webster online dictionary's definitions of theory is that *it is a belief, policy, or procedure proposed or followed as the basis of action*. A theory enables us to analyze a phenomenon and understand the research findings. With reference to the discipline of evaluation, [Clarke and Dawson, 1999, p. 30] distinguishes between two types of theories; theory *about* evaluation and theory *in* evaluation. Mentioning the former kind of theory, [Shadish et al., 1991, p. 34] write:

> *Evaluation theory tells us when, where, and why some methods should be applied and others not, suggesting sequence in which methods could be applied, ways different methods can be combined, types of questions answered better or less well by a particular method, and benefits to be expected from some methods as opposed to others.*

The other type, theory in evaluation, is the application of program theory (how the program is supposed to behave) for program evaluation. However, the term *evaluation theory* most commonly refers to theory of evaluation, i.e. how the evaluation has to be conducted. [Shadish et al., 1991] summarize the different evaluation theories presented by seven well known theorists. None of the seven theories consider entirely generic evaluation situations as they do not avoid references to social programs. However, Scriven's theory can be assumed to be at the highest level of abstraction as he describes principles, concepts, and methods for any scenario of knowledge construction in evaluation. His theory is discussed with little more detail in the next section.

### 3.1.2 Evaluation Components

Scriven's theory of evaluation as mentioned in [Scriven, 1996a] and also by [Shadish et al., 1991, Ch. 3] attempts to clarify the logic behind evaluations. In Scriven's words "*the most common type of evaluation involves determining criteria of merit (usually from needs assessment), standards of merit (frequently as a result of looking for appropriate comparisons), and then determining the performance of the evaluand so as to compare it against these standards*". He further explains that two more steps are involved before and after these three key evaluation activities. Prior to anything, an object

of evaluation is to be selected, and after the third step of performance measurement, the results have to be accumulated and transformed to a shape from which judgements could be drawn. These five elements of evaluation are further explained below:

- **Target**
  It is the object under evaluation. [Scriven, 1991, p. 139] calls it *evaluand*, a generic term for whatever is being evaluated, be it concrete or abstract, for example, person, performance, program, proposal, product, possibility etc. Gaining an understanding of the evaluand is the first step in the development of rest of the evaluation components. Delimitation of the scope and range of the target helps in determining the boundaries of the evaluation program.

- **Criteria**
  Once the nature of the evaluand has been established, we become able to determine the possible good or bad characteristics of our target of evaluation. For example, for a train, punctuality, safety, and comfort are some examples of criteria of its merit. Such characteristics of the target that are to be assessed make up the criteria element. With the elicitation of the criteria we come up with the set of attributes that are interesting to us and are to be evaluated.

- **Standard/yardstick**
  The ideal form of the target based on the optimality of defined criteria makes up the standard or yardstick against which a real target is to be matched. Acceptable ranges for certain performance indicators is an example of yardstick. In case of criteria of punctuality mentioned in the previous element a typical yardstick could be a deviation of plus/minus 2 minutes, for example. Since multiple sets of criterion are most commonly involved, we need to develop a balanced combination of optimal values of the required attributes.

- **Assessment techniques**
  The criteria set forth for the target must be somehow assessed to find out how well the target rates against each of the criteria. Different assessment techniques or methods such as qualitative or quantitative may be used for this purpose.

- **Synthesis techniques**
  Application of the assessment techniques makes available a lot of data and/or raw information which is not much meaningful in its that state. Synthesis techniques provide means to harmonize assessment data and transform it to a final evaluative judgement so it can be compared against the yardstick. Different types of mathematical or statistical techniques can be applied to synthesize assessment data.

In addition to Scriven's identification of key elements of evaluation, common sense knowledge of evaluative thinking was applied in the context of information systems evaluation to draw a framework of analysis [Grembergen, 2001, Ch. 4]. This framework uses *why, what, which aspects, when, who,* and *how* questions to come up with a list of elements of evaluation as *purpose, subject, criteria, time, people,* and *methodologies*. Application of the philosophy of evaluation to software architectures [Lopez, 2003], and software processes [Ares et al., 2000] also drew quite similar elements of evaluation with the addition

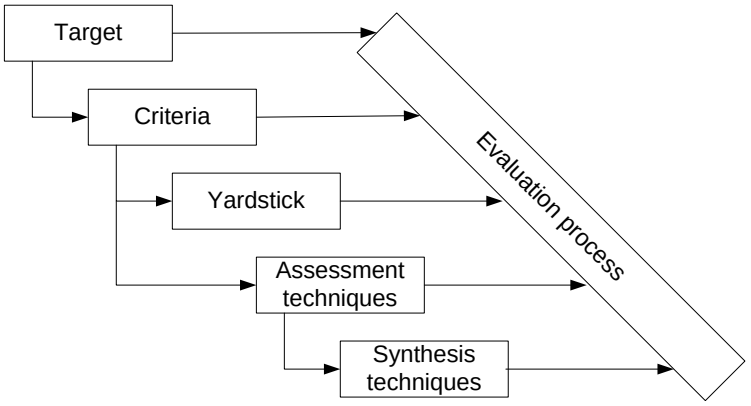


**Figure 3.2:** *Interrelationships among Components of Evaluation*

of *evaluation process* component which is aimed at providing guidelines for the conduct of all evaluation activities. This element can define a series of activities and sub-activities that are performed from the beginning of the evaluation till it finishes. Planning, execution, and evaluation results reporting are generic broad level activities involved here. There may be slightly varying organization of these activities but the main purpose of this element is to provide express guidelines for the evaluation steps.

Thus combining all these views, six generic components can be traced in all kinds of evaluative works, whether they are related to the field of social sciences or otherwise. Figure 3.2 summarizes these components and draws the relationships that exist among them.

## 3.2 Evaluation in Software Engineering

***SE's connections to other disciplines***

The field of software engineering cannot solve all of its problems in isolation from other knowledge domains. It has a strong multidisciplinary nature [Wang, 2008] and incorporates laws, theories and experiences from many other science and engineering disciplines such as philosophy, mathematics, computing, linguistics, information science, cognitive informatics, system science, management science, economics, sociology, and engineering organization. Evaluation as a discipline of social sciences presents one such example of non-software field which can serve as foundation for evaluation problems focused on different software engineering artifacts such as products and processes. A short discussion below reviews the status of evaluation in software engineering and the application of theories and knowledge from other disciplines for solving this class of problems.

***SE research problems***

Lázaro and Marcos [Lázaro and Marcos, 2005] have distinguished software engineering research problems as either *engineering problems* (concerned with the formulation of new artifacts) or *scientific problems* involving analysis of existing artifacts. The role of evaluation and assessment in software engineering is realized by long-standing works of many researchers as [Kitchenham et al., 1997], [Budgen, 2000],

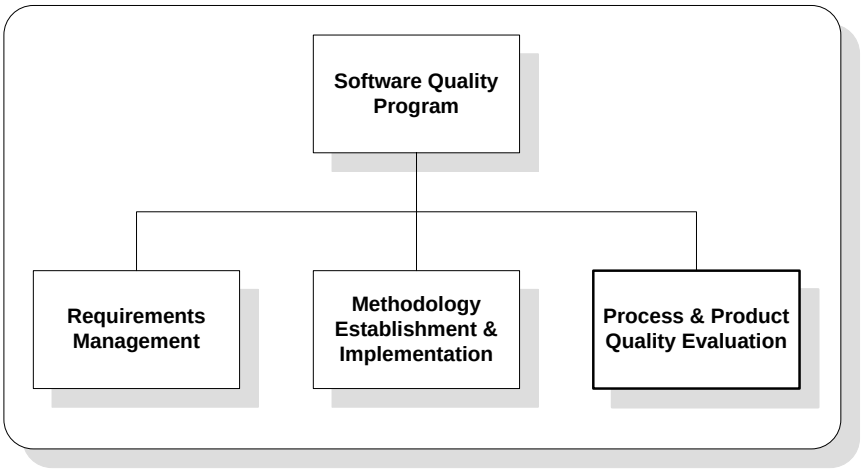


**Figure 3.3:** *Software Quality Elements [Kenett and Baker, 1999]*

[Singpurwalla and Wilson, 1999], and [Dumke et al., 2004]. Despite these works, one of the criticisms to software engineering research is that it ignores evaluation [Zelkowitz and Wallace, 1997]. This opinion is further strengthened by a survey conducted by Glass et al. [Glass et al., 2004] in which it was found that 79% of approaches in the field of general computer science and 55% of approaches in software engineering were formulative in nature while only about 14% approaches were evaluative works. Perhaps still today many research efforts follow the research model that Glass [Glass, 1994] once described as *advocacy research* consisting of steps, "conceive an idea, analyze the idea, advocate the idea" ignoring the comparative evaluation among the proposed and existing approaches.

***Evaluation for quality assurance***

Evaluation is an important tool of software quality assurance. A typical software quality program involves i) establishment, implementation, and control of requirements, ii) establishment and control of methodology and procedures, and iii) software quality evaluation [Kenett and Baker, 1999, p. 4]. Figure 4.2 summarizes this observation. The software quality evaluation component is aimed at evaluating products (both in-process and at completion), activities and processes (for optimization and compliance with standards), and methodologies (for appropriateness and technical adequacies).

***Evaluation terminology in SE***

When it comes to software engineering in general and software process in particular, the terms *evaluation* and *assessment* are interchangeably used in literature and practice. We however differentiate between them and follow the viewpoint of Kenet and Baker [Kenett and Baker, 1999] which seems quite logical specially in view of available process evaluation approaches. The nature of the software evaluation, according to him, may be qualitative ("*assessment*") or quantitative ("*measurement*"). "Measurement encompasses quantitative evaluations that usually use measures which can be used to directly determine attainment of numerical quality goals. On the other hand, any evaluative undertaking that requires reasoning or subjective judgment to reach a conclusion as to whether the software meets requirements is considered to be an assessment. It includes analysis, audits, surveys, and both document and project reviews" [Kenett and Baker, 1999]. Per-

haps it would be more appropriate not to call them as two different types of evaluations but rather as two different methods of gathering data for the purpose of evaluation. This text will follow this distinction between qualitative and quantitative methods while studying and analyzing evaluative works in the discussed areas.

It has been observed that evaluation approaches of software processes and products are largely based on empirical methodologies and lack well formed theoretical foundations.

## 3.3 Evaluation Theory Applied in SE

To the best of the author's knowledge, only a couple of SE literary material could be found which attempted to exploit theory of evaluation in some manner. These examples will be discussed one-by-one below.

***Meta-evaluation of software architectures***

Designing an architecture for the software system is among the initial steps in the system development life cycle. Success of a developed system heavily depends upon the underlying architecture which if poorly designed will not be able to provide intended functionality. Software architectures for larger or complex systems are themselves complex too, one needs to make judicious choice among candidate design alternatives which strongly affect the quality aspects of the developed systems. A number of methods have been developed to formally compare different architectural decisions specially with a view to ensure high quality or reduce risks. Architecture Tradeoff Analysis Method (ATAM) [Kazman et al., 2000] is one such method developed by Software Engineering Institute (SEI). This method is aimed at analyzing a software architecture to determine if it will be capable of satisfying established quality goals and also to compare interactions between them. It guides the selection of suitable architecture taking into consideration various trade-offs, quality requirements, and risks. In this sense, it is an evaluation method for software architectures.

An evaluation of the ATAM method (an evaluation of the evaluation, hence the name meta-evaluation) has been performed by Lopez et al. [Lopez, 2003] to check for its comprehensiveness, weaknesses, or possible improvements. The authors have performed this evaluation using the concepts of evaluation theory. They have analyzed the ATAM method against the six key elements of evaluation guided by the principles and theory of evaluation. The findings of their study are summarized in table 3.1.

The study has enabled a systematic characterization of the method, identification of missing or incomplete elements, and a feedback about possible enhancements and improvements of ATAM approach.

***Meta-evaluation of software processes***

A number of software process assessment and improvement approaches have been developed to date. All of these methods follow slightly or greatly different paths to achieve the same goal of improving the software process. Explanations, analysis, and comparisons of these approaches have been given in many literary works. At times, these models may look quite similar in structure and purpose making it hard to decide which one of them may be suitable for an organizational setting. A

**Table 3.1:** *Evaluation theory perspective of ATAM [Lopez, 2000]*

| | **ATAM-Architecture Tradeoff Analysis Method** |
|---|---|
| **Target** | Architecture (not explicitly defined) |
| **Criteria** | General criteria: quality attributes, Specific criteria: architectural decisions, responses |
| **Yardstick** | Particular scenarios and architecture styles |
| **Assessment techniques** | Presentations, interviews, questionnaires |
| **Synthesis techniques** | Not explicitly distinguished |
| **Evaluation process** | Structured steps and tasks described in ATAM documentation |

few techniques have been developed to serve this purpose. These include an evaluative framework for software process improvement models by Saiedian and Chennupati [Saiedian and Chennupati, 1999], an object-oriented software measurement and evaluation framework by Dumke and Foltin [Dumke and Foltin, 1999], metrics-based evaluation frameworks for XP (eXtreme Programming) [Williams et al., 2004] and RUP (rational unified process) [Krebs et al., 2005], a technique to compare them based on their return on investment [Rico, 2004] etc. Any comparisons among these models mostly consider the value offered by these models to an organization while they pay little attention to the content of the models themselves.

Comparing them from the content perspective, many of these approaches focus heavily on developing standard/yardstick and the details of the evaluation process itself which are just two among the six core elements of evaluation described earlier. Analyzing for comprehensiveness or treatment of the problem, Ares et al. [Ares et al., 2000] have compared different software process evaluation approaches with reference to principles of evaluation to discover existence of sound theoretical and scientific foundations behind them. Like the analysis of ATAM discussed earlier, the authors also have taken an evaluation theory perspective of the software process assessment methods and have analyzed them with reference to six key components of evaluation. Their findings as outlined in table 3.3 show some missing or inadequately provided evaluation components in all of the considered approaches.

The authors further describe a process evaluation method which is aimed to be comprehensive enough to provide all the necessary evaluation components. A brief summary of method components that they defined is given in table 3.2. Although the developed method seems to follow a systematic and logical approach to process evaluation yet it is marked with some visible shortcomings. For example, *criteria* is defined as comprising three elements; activities, software process model, and structure. Criteria should be characteristics of the target which is of interest for assessment purposes. Criteria definition is quite implicit in nature as given by the method. The *yardstick* is defined as a process model quite similar to SPICE or CMMI. It is not clear if the defined process model comes from any set of well established best practices or only from the author's personal experiences.

**Table 3.2:** *Evaluation theory based software process evaluation method [Ares et al., 2000]*

| | **Developed Method** |
|---|---|
| **Target** | Technical, management, and strategic processes |
| **Criteria** | Criteria tree comprising activities, software process model, and structure |
| **Yardstick** | Software process model |
| **Assessment techniques** | Questionnaires, interviews, and document inspections |
| **Synthesis techniques** | Criteria grouping and datum-by-datum analysis |
| **Evaluation process** | Planning, examination, and decision making as main phases |

Furthermore, the software process model is exemplified only for the requirements analysis process (table 7 in [Ares et al., 2000]), no information is given for other processes. The last method component is *evaluation process* itself which gives some high level information for evaluation phases and sub-phases without any details about specific inputs/outputs and entry/exit criterion.

## 3.4 Summary

As part of the investigating the scientific foundation and cross-disciplinary analysis of the research problem, this chapter concentrated and resolved the second sub-question of the current research task.

RT-2. Investigate the philosophy of evaluation in general and with special focus to software engineering. Identify core elements of a comprehensive process evaluation approach.

The chapter set about an inquiry into the baseline philosophy of evaluative undertakings. Following this trail led to *Evaluation*, which is a discipline in its own right. It is a field of social sciences which is concerned with studying the issues involved in evaluation of social programs. Key concepts of this discipline such as evaluation types, models, and theories were briefly explained. Focal point of the discussion remained theory of evaluation, which is concerned with the rationale and design of any kind of evaluations. Out of this deliberation emerged six elementary components of any generic evaluative design. The next part of the chapter explored connections between the field of Evaluation and Software Engineering and that how one can benefit from the former to solve some of the problems of the later field. An abridged analysis of software engineering research problems emphasized the fact that evaluation of existing artifacts is given lesser attention as compared to development

of new ones. Evaluation was identified mainly as a tool for quality assurance. Finally, the chapter inquired about the software engineering evaluation approaches that could possibly have exploited theoretical foundations of evaluation provided by the discipline and field of Evaluation. Only two such instances were found and explained in brief.

In finding conceptual requirements underlying any comprehensive evaluation approach for software processes or test processes, this chapter discovered the following mandatory components for the composition of a solution.

Components of Comprehensive Test Process Evaluations

- ✓ The test process evaluation approach must explicitly define and delimit the target, i.e. the entities interesting for evaluation.
- ✓ The evaluation approach must explicitly define particular criteria to be considered about the entities of interest.
- ✓ The test process evaluation approach must provide a standard or yardstick against which the target process could be compared.
- ✓ The approach must describe and explain the type of techniques that will be used to assess the target against the criteria.
- ✓ The approach must specify the synthesis techniques that will be used to integrate the collected assessment or measurement data.
- ✓ Finally, the test process evaluation approach must define the steps of the evaluation process

**Table 3.3:** *Evaluation theory perspective of software process assessment methods [Ares et al., 2000]*

| | ISO 9000 | TickIT | CMM | Bootstrap | Trillium | STD | ISO 15504 |
|---|---|---|---|---|---|---|---|
| **Target** | Quality system | Quality System | Management process | Technical and management process and methodology | Telecommunications | Technical, management and strategic process | Technical, management and strategic process |
| **Criteria** | Implicit | Implicit | Implicit | Defined | Implicit | – | Defined |
| **Yardstick** | Developed | Developed | Developed | – | Developed | – | Not developed |
| **Assessment techniques** | Not prepared | Not prepared | Not prepared | Developed | Not prepared | – | Not proposed |
| **Synthesis techniques** | Generic description | Generic description | Generic description | Developed | – | – | Developed |
| **Evaluation process** | Partially developed | Partially developed | Developed | Developed | Partially developed | Partially developed | Not developed |

# 4 Evaluation in Software Testing

The purpose of this chapter is to provide a detailed level survey of related work. Through the analysis of contemporary research problems, the chief objects of evaluation in the field of software testing are identified first. For each of these objects, the chapter contains findings from an exhaustive survey of available evaluation techniques and models. After reading this chapter, the reader should be familiar with existing forms and approaches of evaluation as prevalent in the area of software testing.

## 4.1 Introduction

The fundamental purpose behind all kinds of software testing is to evaluate the products of the software development process. A large portion of the research in the field of software testing has concentrated on *how* to perform it. Chapter 2 in this thesis has discussed various topics which are strongly related to this aspect of software testing. Since testing is an expensive and difficult piece of work, the huge costs and effort required to perform it motivate us to evaluate the testing itself. In this case, it will be equivalent to finding out *how well* the testing is being performed. Despite the availability of so many scattered approaches in this direction, evaluation of software testing artifacts remains an elusive research problem. Such evaluations when made available and if they are comprehensive enough, can help reduce uncertainties about the testing activities such as their adequacy, time, cost, and quality of the developed products.

**Objects of evaluation in software testing**

But the first question is which testing artifacts can be and should be evaluated? Chapter 2 has partially developed the answer to this question. The topics contained therein consist mainly of test levels, test techniques, test measures, test process, and test tools. Among these, *test techniques* is one element of evaluation, for example, we need to know how much effective is our technique in terms of effort and defect finding capability. *Test tools* are another target of measurement. We need to assess and analyze our tools themselves for their efficiency. *Test process* is perhaps the most substantial element to evaluate since it may cover other elements of evaluation under its umbrella. By evaluating test process we try to find out how much effective and efficient is it in terms of money, time, effort, and defect identification and removal. The next sections discuss these three testing artifacts and survey existing forms of evaluations for each of them.

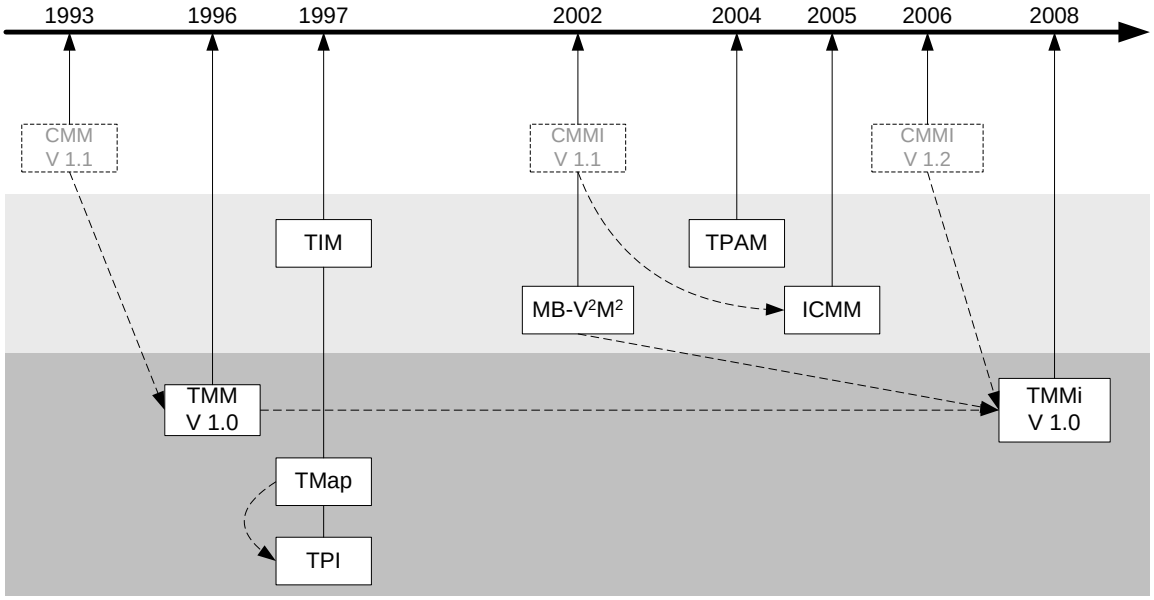


**Figure 4.1:** *History of Test Process Assessment Models & Dependencies*

## 4.2 Evaluation of Processes

***History of test process assessment/improvement models***

Motivated by Capability Maturity Model (CMM) and a lack of any maturity models in the field of the software testing, Testing Maturity Model (TMM) introduced in 1996 was the first model of its kind. It was followed by Test Process Improvement (TPI) model appearing in 1997. In the same year, another related approach Test Improvement Model (TIM) [Ericson et al., 1997] was published in an article which later disappeared into the ocean of research without enjoying any significant appreciation. Two later approaches similarly introduced in short articles, one in 2002 named as Metrics-based Verification & Validation Maturity Model ($MB - V^2M^2$) [Jacobs and Trienekens, 2002], and another in 2004 as Test Process Assessment Model (TPAM) [Chernak, 2004] met the same fate as TIM. Another similar maturity model specifically for the inspection process, called Inspection Capability Maturity Model (ICMM) [Kollanus, 2005], was developed in 2005. The latest well organized and detailed development (other than this thesis) in this regard is the Test Maturity Model Integration (TMMi) which is still under development. Figure 4.1 summarizes time-line of these test process evaluation and improvement models.

TIM, TPAM, and $MB - V^2M^2$ appear to have vanished from literature probably due to their insignificance or incompleteness. These three models will be ignored here from further discussion. An overview of the four *living* test process assessment models is summarized in table 4.1.

### 4.2.1 Testing Maturity Model (TMM)

***Model concept***

Testing Maturity Model (TMM) was developed by Ilene Burnstein [Burnstein, 2003] to

**Table 4.1:** *Comparison of Test Process Assessment Models*

| Model | Dependency | Approach | Scope |
|---|---|---|---|
| TMM<br>Testing Maturity Model | CMM | Implicit | General |
| TPI<br>Test Process Improvement | TMap | Implicit | Structured testing |
| ICMM<br>Inspection Capability Maturity Model | CMMI | Implicit | Software inspections |
| TMMi<br>Test Maturity Model Integration | CMMI | Implicit | General |

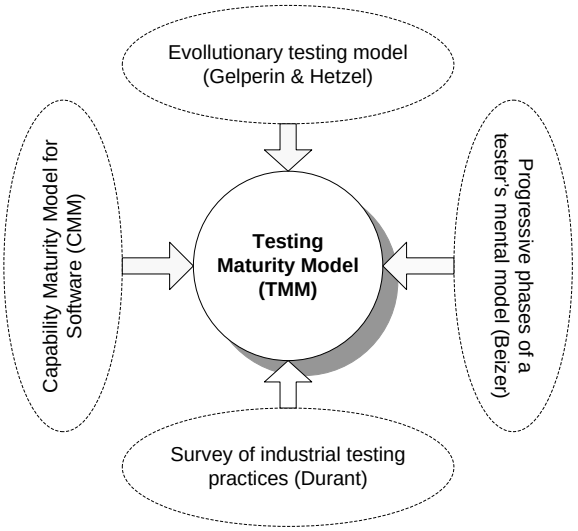


**Figure 4.2:** *Inputs to the Testing Maturity Model*

assist and guide organizations focusing on test process assessment and improvement. Since release of its first Version 1.0 in 1996 no further release has appeared. The principal inputs to TMM were Capability Maturity Model (CMM) V 1.1, Gerlperin and Hetzel's Evolutionary Testing Model [Gelperin and Hetzel, 1988], survey of industrial testing practices by Durant [Durant, 1993] and Beizer's Progressive Phases of a Tester's Mental Model [Beizer, 1990]. The figure summarizes these inputs. It is perhaps the most comprehensive test process assessment and improvement model to date.

***Model structure in brief***

TMM derives most of its concepts, terminology, and model structure from CMM. The model consists of a set of maturity levels, a set of maturity goals and sub-goals and associated activities, tasks and responsibilities (ATRs), and an assessment model. The model description follows a staged architecture for process improvement. The maturity levels along with relevant maturity goals are shown in figure 4.3.

***Model structure in detail***

TMM reference model is organized in two parts. The first part contains five maturity

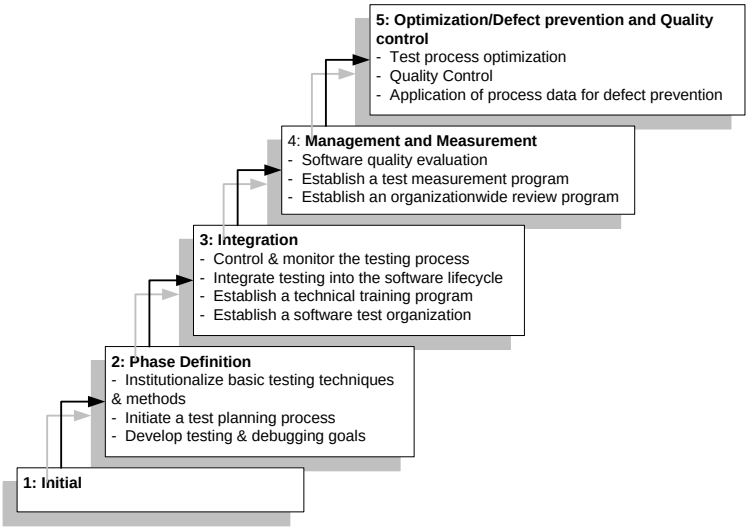


**Figure 4.3:** *TMM Maturity Levels*

levels which define evolutionary path to test process improvement. The contents of each level are described in terms of testing capability organizational goals and sub-goals. Under each maturity goal, several questions are provided which correspond to testing practices. A maturity goal is expected to be achieved if all the questions pertaining to it can be answered affirmatively. Level 1 contains no goals and therefore every organization is at least at level 1 of test process maturity. This part one with the questionnaire is mainly of use for the process assessments.

The second part contains recommendations, roles, and responsibilities for the key players in the testing process–the managers, developers/testers, and users/clients. The model organizes sets of TMM activities, tasks, and responsibilities (ATRs) to be performed by these three key players. Appropriate ATRs are given for each maturity goal as process improvement suggestions to achieve that goal. Relationships between its model elements have been summarized in figure 4.4.

The author of the TMM also provides an assessment model [Burnstein, 2003, Ch. 16]. The model follows the class of team-based self assessments. Information regarding selection and training of assessment team, assessment procedure, and the assessment questionnaire is provided. Details of the ranking procedure for rating maturity goals and sub-goals are also included.

***Short critical review***

A comparison of TMM with other test process improvement models has been performed by Swinkels [Swinkels, 2000]. It is a kind of characteristics-based comparison which takes into consideration the model elements and coverage of the process improvement issues. He concludes that TMM and other test process improvement models of its era appear to complement each other. Another detailed criticism of TMM has been performed by the author of the thesis in [Farooq et al., 2007]. The analysis which is based on content and structure of the model comes up with some suggestions to improve the model structure, to update the assessment model, and to expand the process areas.

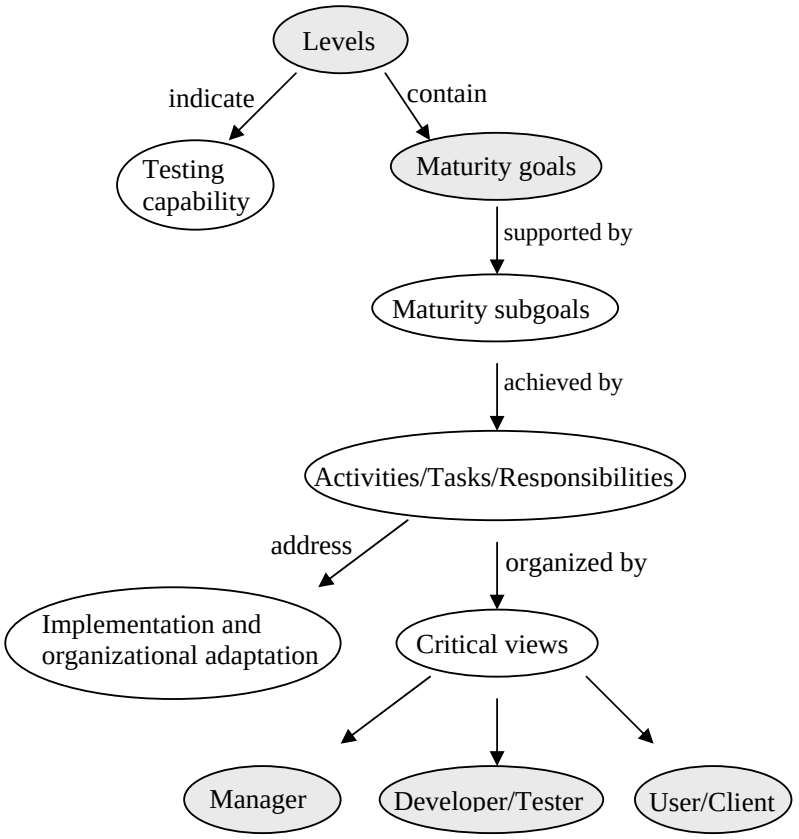


**Figure 4.4:** *Structure of Testing Maturity Model [Burnstein, 2003]*

### 4.2.2 Testing Process Improvement (TPI) Model

***Model concept***

Test Process Improvement (TPI) [1][Koomen and Pol, 1999] model is an industrial initiative to provide test process improvement guidelines based on the knowledge and experiences of a large number of professional testers. The first release of this model appeared in 1997. The model has been designed in the context of structured high level testing. It is strongly linked with the Test Management Approach (TMap) [Pol et al., 2002] test methodology.

***Model structure in brief***

The model elements include several key areas, each with different levels of maturity. A maturity matrix describes the connections between key areas and levels. For every key area, several checkpoints have been defined corresponding to each maturity level; questions that need to be answered positively in order to classify for that level. Improvement suggestions, which help to reach a desired level, are also part of the model. Relationships among TPI model elements are summarized in figure 4.5.

***Model structure in detail***

TPI model consists of 20 key areas which are organized by means of the four cornerstones of structured testing as defined by TMap: life cycle, organization, infrastructure, and techniques. These key areas are listed in table 4.2. Level of achievement relevant to these

[1] http://www.sogeti.nl/Home/Expertise/Testen/TPI.jsp

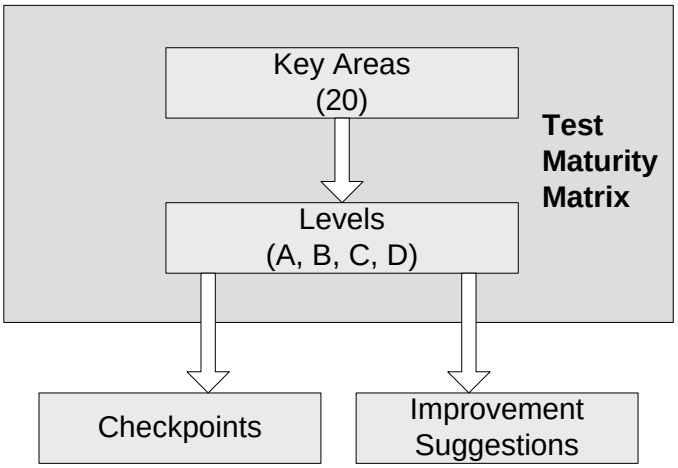


**Figure 4.5:** *Structure of Test Process Improvement (TPI) Model*

**Table 4.2:** *TPI Key Areas*

| Cornerstone | Key areas |
|---|---|
| Life cycle | Test strategy, Life-cycle model, Moment of involvement |
| Techniques | Estimating and planning, Test specification techniques, Static test techniques, Metrics |
| Infrastructure & tools | Test tools, Test environment, Office environment |
| Organization | Commitment and motivation, Test functions and training, Scope of methodology, Communication, Reporting, Testware management, Test process management |
| All key areas | Evaluation, Low-level testing |

key areas is defined through maturity levels. There can be three to four maturity levels for each key area. Each level consists of certain requirements (defined in terms of checkpoints) for the key area. With the design of TPI, all key areas need not be and cannot be at the same maturity level, so an overall maturity level for all key areas is not provided by the model. Improvement suggestions are intended to help fulfill checkpoints for all levels and key areas. TPI does not provide any assessment model. It seems that TPI is aimed at self-assessment.

***Short critical review***

Two world-wide surveys on adoption of TPI by software industry have been reported in [Koomen, 2002], [Koomen and Notenboom, 2004]. These surveys reported positive improvements and better control of the testing process by the organizations applying the model. Both of these surveys are not enough detailed, for example, they do not provide any information about the characteristics of the survey respondents. Critical review and comparisons of TPI with other test process improvement models given in [Swinkels, 2000], [Goslin et al., 2008b, p. 70] have found it a complementary approach to TMM. Another

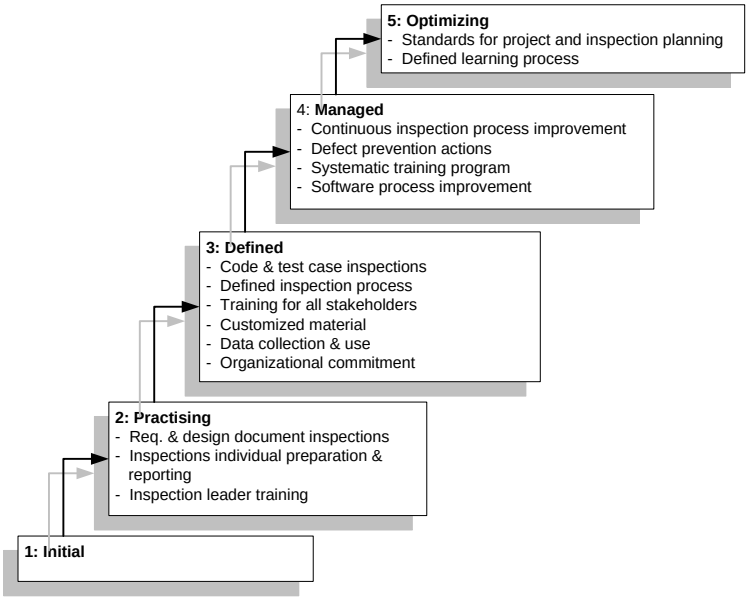


**Figure 4.6:** *ICMM Maturity Levels*

comparison among the test process improvement models has been performed by author in [Farooq and Dumke, 2008a]. The comparison frames the approaches in an evaluation framework made out of critical success factors for process improvement. The findings show that TPI diverges from process assessment standards as well as ignoring some other process improvement aspects.

### 4.2.3 Inspection Capability Maturity Model (ICMM)

***Model concept***

Kollanus [Kollanus, 2005] developed the Inspection Capability Maturity Model (ICMM) to help organizations assess and improve the level of inspection process. He developed this method due to CMMI's lack of not explicitly addressing software inspections and TMM's limitation of not being able to assess reviews/inspections independently from the rest of the testing process.

***Model structure***

ICMM derives its concept, terminology, and model structure from CMMI. The model consists of five maturity levels (initial, practising, defined, managed, optimizing) and altogether 15 process areas. The maturity levels along with relevant process areas are shown in figure 4.6. Maturity level 2 mandates existence of only base inspection practices. The process areas within this level cover only inspection of requirements/documents and some fundamental issues about conducting inspections. Maturity level 3 calls for establishing well organized inspection program covering all sorts of inspections. Level 4 is about evaluating and improving the inspection process itself while level 5 concerns following standard approaches in the inspection process. The description of process areas inside these maturity levels includes only some guidelines without their division into any further elements such as generic/specific goals/practices. ICMM does not provide any information about a corresponding assessment model.

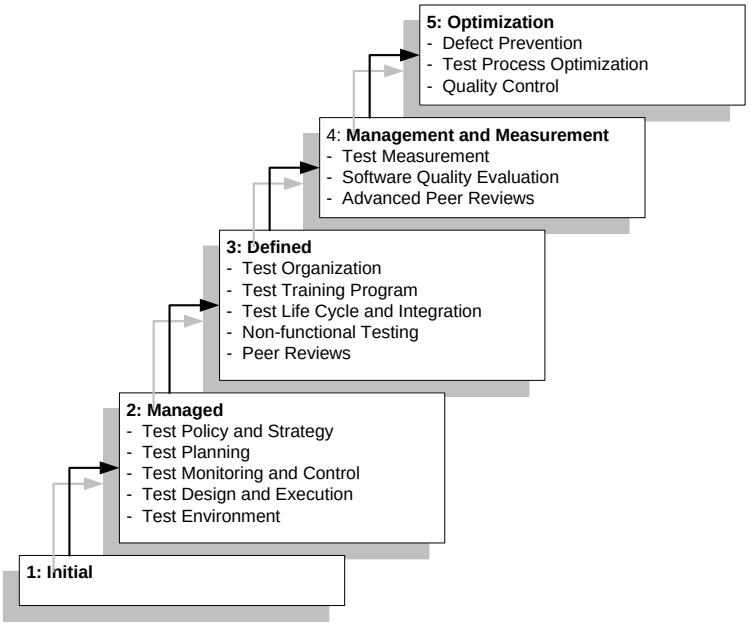


**Figure 4.7:** *TMMi Maturity Levels*

***Short critical review***

In the first place, the ICMM model is described in only a very short article [Kollanus, 2005]. The model (with its maturity levels and process areas) rather gives only a skeleton of a full inspection maturity model. A later article [Kollanus, 2009] has discussed application of this model in a number of case studies where the approach has been found to work well to identify strengths and weaknesses of industrial inspection practices.

### 4.2.4 Test Maturity Model Integration (TMMi)

***Model concept***

TMMi is being developed by a non-profit organization called TMMi Foundation. This framework is intended to complement Capability Maturity Model Integration (CMMI) with a special focus on testing activities and test process improvement in both the systems engineering and software engineering discipline. An initial version 1.0 [Goslin et al., 2008b] of this framework was released in February 2008. The current version follows staged representation and provides information only up to maturity level 2 out of the five proposed levels. The assessment framework itself is not part of TMMi and has not been released yet.

***Model structure in brief***

TMMi borrows its main principles and structure from Capability Maturity Model Integration (CMMI), Gelperin and Hetzel's Evolution of Testing Model [Gelperin and Hetzel, 1988], Beizer's testing model [Beizer, 1990], IEEE Standard for Software Test Documentation [IEEE, 1998], and ISTQB' Standard Glossary of terms used in Software Testing [ISTQB, 2006]. Similar to CMMI, this framework defines three types of components. The model defines exactly the same elements defined by CMMI, maturity levels, process areas, generic and specific goals etc. The maturity levels and process areas proposed by this model have been summarized in figure 4.7.

***Model structure in detail***

A detailed structure of the model has been shown in figure 4.8. TMMi defines three types of components–required, expected, and informative components which have been summarized to illustrate their relationship in figure 4.8. These components are shortly explained as,

- **Required**: These components describe what an organization must achieve to satisfy a process area. Specific and generic goals make up required component of TMMi.
- **Expected**: These components describe what an organization will typically implement to achieve a required component. Expected components include both specific and generic practices.
- **Informative**: These components provide details that help organizations get started in thinking about how to approach the required and expected components. Subpractices, typical work products, notes, examples, and references are all informative model components. Informative components describe other components.

TMMi defines five maturity levels. A maturity level within this framework indicates the quality of organizational test process. To each maturity level several process areas are associated which in turn involve several generic and specific goals and generic and specific practices. To reach a particular maturity level, an organization must satisfy all of the appropriate goals (both specific and generic) of the process areas at the specific level and also those at earlier maturity levels. All organizations possess a minimum of TMMi level 1, since this level does not contain any goals that must be satisfied.

***Short critical review***

Test Maturity Model Integration is no doubt a long awaited enhancement to its predecessor Testing Maturity Model. Below are presented some critical observations of TMMi.

- The model description is yet incomplete since the currently available document only provides information up to maturity level 2.
- The assessment framework for TMMi is also not part of current release and is not yet publicly available.
- The current release of TMMi provides only a *staged* model representation. This same limitation was also observed for TMM [Farooq et al., 2007]. A *continuous* representation on the other hand lets an organization to select a process area (or group of process areas) and improve processes related to it. While staged and continuous representations have respective pros and cons, the availability of both representations provides maximum flexibility to organizations to address their particular needs at various steps in their improvement programs.
- TMMi is designed to be a complementary model to CMMI. The model description [Goslin et al., 2008b, p. 6] states that "*in many cases a given TMMi level needs specific support from process areas at its corresponding CMMI level or from lower CMMI levels. Process areas and practices that are elaborated within the CMMI are*

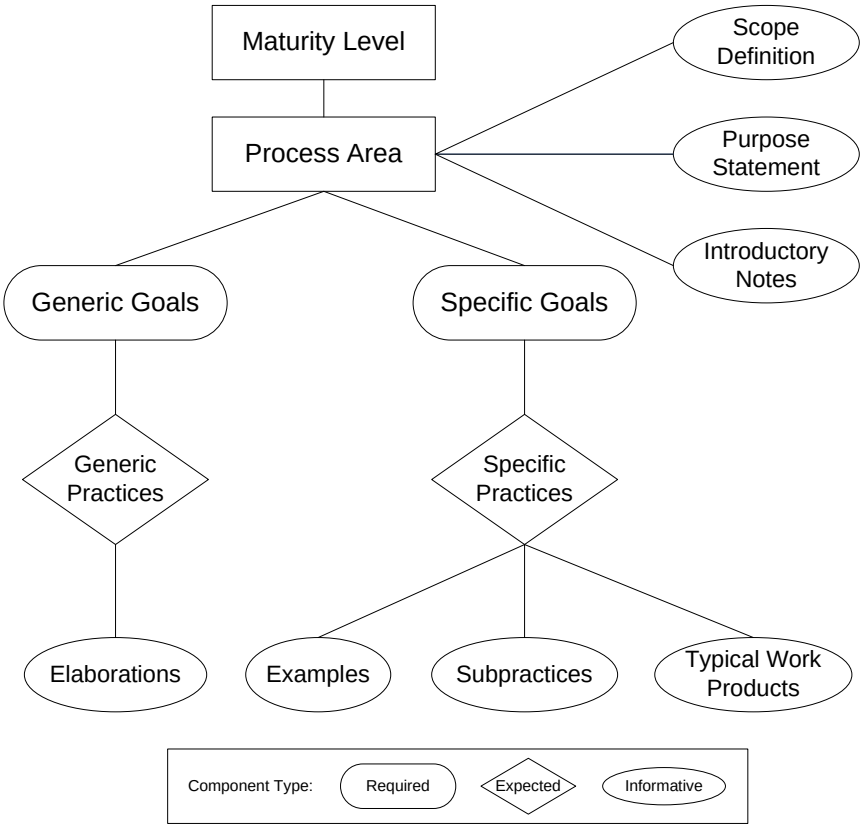


**Figure 4.8:** *Structure of Test Maturity Model Integration (TMMi)*

*mostly not repeated within TMMi; they are only referenced*". Now there are organizations which offer independent software testing services. Such or other organizations may solely want to concentrate on improvement of their testing process only. Strong coupling and references between TMMi and CMMI may limit independent adoption of this framework without implementing a CMMI process improvement model.

### 4.2.5 Test Process Metrics

***Concept of test metrics***

Quantitative approaches to process management work by evaluating one or more of its attributes through measurement. The measurement information so obtained reflects some key characteristics of measured process such as size, involved effort, efficiency, and maintainability etc. The objectivity of the information provides possibility of precise and unbiased evaluation as compared to that obtained through assessments. Although several measurement tools and frameworks [Dumke, 2005], [Dumke et al., 2006c] exist for the generic software process and can possibly be tailored to test process with minor or major changes, but very few have been developed solely for the test process. Measurement techniques for software test process exist broadly in the form of metrics for the test process. This section analyzes available metrics in this area.

***Types of test metrics***

Like other knowledge areas within software engineering, testing related measures are very

helpful to managers to understand, track, control, and improve the testing process. For example, metrics of testing costs, test effectiveness, tester productivity, testability, test cases, coverage, defects and faults and other similar aspects can give us very valuable insight about many different aspects of software testing. Realizing necessity of such measurements, a number of test process metrics have been proposed and reported in literature. The table 4.3 provides a non-comprehensive list of test metrics definitions found in the literature.

**Table 4.3:** *Resources of Test Metrics Definitions*

| Reference | Types of Metrics Discussed |
|---|---|
| [Peng and Wallace, 1994] | test cases, coverage, failure |
| [Liggesmeyer, 1995] | test complexity |
| [Suwannasart et al., 1999] | Miscellaneous |
| [Kan, 2002, Ch. 10] | Miscellaneous |
| [Hutcheson, 2003, Ch. 5] | size, cost, defect |
| [Kan et al., 2001] | test progress |
| [Pol et al., 2002] | miscellaneous |
| [Burnstein, 2003, p. 266] | testing status, tester productivity, testing costs, errors/failures/faults, test effectiveness, metrics at test process maturity levels |
| [Chen et al., 2004] | quality metrics, time-to-market metrics, cost-to-market metrics |
| [Abran et al., 2004, p. 5-7] | program under test, tests performed |
| [Verma et al., 2005] | coverage metrics |
| [Rajan, 2006] | coverage metrics |
| [Harris, 2006] | coverage metrics |
| [Whalen et al., 2006] | coverage metrics |
| [Sneed, 2005] | test cases, costs, coverage, test effectiveness |
| [Sneed, 2007] | product, project, progress, process |
| [Afzal, 2007] | test progress, cost, quality |

Nonetheless, we can distinguish several of these metrics which are meaningful at the process level only, for example few maturity level metrics and process progress and effectivity metrics. Availability of so many metrics may sometimes confuse practitioners rather than help them. A well organized list of these metrics may help a test manager better understand metrics available at hand and to select them according to particular situations and needs. Feeling this need, the author and his colleagues [Farooq et al., 2008a] presented a classification of test metrics considering various test contexts. Existing test related metrics and test metrics classifications were also reviewed. Figure 4.9 shows Farooq et al.'s [Farooq et al., 2008a] classification of test metrics. Another related approach to classify software process metrics was presented by Dumke et al. [Dumke et al., 2006b].

***Short critical review***

An examination of literature on test related metrics has revealed that research in this context

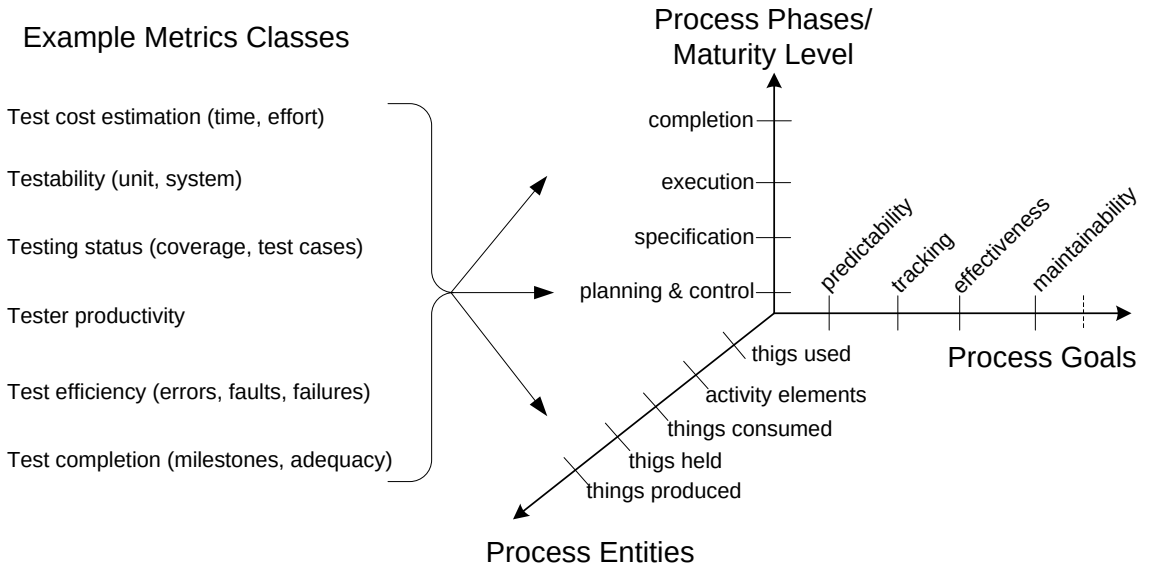


**Figure 4.9:** *Classification of Test Process Metrics [Farooq et al., 2008a]*

is as yet immature. Each set of existing test metrics have been defined only in a confined context, serving the need of some particular analysis problem of a given testing aspect. We still lack widely known common set of test metrics. Moreover, existing test metrics remain poorly validated both from theoretical and empirical point of view.

## 4.3 Evaluation of Techniques

***Why evaluate techniques***

There are many reasons why the evaluation of testing techniques should be carried out. Issue of technique selection is one reason. The fault finding capability of candidate testing techniques needs to be assessed first. This kind of information is useful before one has implemented a given technique, but the same information is also useful (as a post mortem analysis) when one is finished with testing. This post-implementation assessment and analysis is needed for subsequent improvement of the technique to increase its effectiveness. This section surveys testing techniques, empirical knowledge about them, and existing ways for assessing them from different quality perspectives.

Although that the criteria for evaluating static or dynamic techniques are nearly the same, the methods are slightly different. Furthermore, literature is divided over surveys and comparisons of evaluation techniques for both kinds of testing. Therefore, evaluation of both families of static and dynamic techniques will be given separately in the next sections.

### 4.3.1 Evaluation of Static Techniques

***Evaluation criteria***

The criterion here means *what* aspect of techniques is usually evaluated. Review of literature shows that the evaluation criteria for static testing techniques has largely been their ability for detecting defects, costs incurred, or expended time and effort. Lam-

sweerde [van Lamsweerde, 2000] mentions few qualitative criterion for evaluating specification techniques, namely constructibility, manageability, evolvability, usability, and communicability. Some of these attributes are applicable to other static techniques as well. Rico [Rico, 2004] mentions several high level criteria such as return on investment (ROI), benefit/cost ratio, return on investment percentage, and net present value.

***Evaluation methods***

The method refers here as to *how* the techniques are evaluated. Wu et al. [Wu et al., 2005] incorporate number of remaining faults in a Bayesian network model of the inspection process to measure its effectiveness. Another example of a similar model-based approach in this direction is [Freimut and Vollei, 2005] who used historical project data and expert opinions to evaluate inspections. Empirical techniques for evaluating inspection effectiveness in isolation or in comparison to testing or other forms of technical reviews include works by Eickelmann [Eickelmann et al., 2002] and Kelly [Kelly and Shepard, 2002] etc. In addition to these, behavioral theory has been used to analyze effectiveness of software development technical reviews. One of the very few approaches for evaluating formal verification techniques is by [Wang et al., 1998] which involves an experiment for measuring effectiveness of design validation techniques based on automatic design error injection and simulation.

***Evaluation results***

Summarizing studies conducted by various researchers to evaluate the effectiveness of inspections as compared to testing , Eickelmann et al. [Eickelmann et al., 2002] mention that inspections are two times more effective than tests to identify errors, cause four times less effort than tests and are 7.4 times more productive than tests. However a recent case study [Chatzigeorgiou and Antoniadis, 2003] has identified that project planning methodologies, as currently applied in software project management, do not account for the inherent difficulties in planning software inspections and their related activities. As a result, inspection meetings accumulate at specific periods towards the project deadlines, possibly causing spikes in the project effort, overtime costs, quality degradation and difficulties in meeting milestones.

Finally, analysis of literature on software reviews and inspections has revealed that current research in this area is now not focusing much on developing new inspection or review techniques. Rather, the modern (and some past) research effort is now being devoted mainly to studying factors that influence success and efficiency of reviews and inspections and to evaluating (relative) effectiveness of these techniques in comparison to other testing and related techniques.

### 4.3.2 Evaluation of Dynamic Techniques

***Types of evaluation***

A rich body of research work is available concerning evaluation of dynamic testing techniques as compared to static techniques. This research work has mainly been triggered by a need to select an appropriate technique among the many competing ones or due to an interest in validating usefulness or effectiveness of a technique in question. For a given testing problem, there may exist several techniques of the same kind which differ by the underlying mechanism. For instance, several regression testing techniques are available, they belong to same family, yet they follow a different way to solve the problem at hand.

Contrary to this are techniques which solve the same testing problem, but exploit totally different set of information for the purpose. As an example, the control-flow and data-flow based techniques derive test cases quite differently. Following this distinction, Juristo et al. [Juristo et al., 2004a] identify two classes of evaluation studies on dynamic techniques as *inter-family*, and *intra-family*,

- Intra-family studies
  - studies on data-flow testing techniques
  - studies on mutation testing techniques
  - studies on regression testing techniques
- Inter-family studies
  - comparisons between control-flow, data-flow and random techniques
  - comparisons between functional and structural control-flow techniques
  - comparisons between mutation and data-flow techniques
  - comparisons between regression and improvement techniques

This section deals with wider range of *intra*-family studies over the state of research covering all dynamic testing techniques.

***Evaluation methods***

Three directions of research have been found related to evaluation of dynamic techniques,

1. actual evaluations and comparisons of testing techniques based either on analytical or empirical methods,
2. evaluation frameworks or methodologies for comparing and/or selecting testing techniques
3. surveys of empirical studies on testing techniques which have summarized available work and have highlighted future trends

During the past few decades, a large number of theoretical and empirical evaluations of numerous testing techniques have been executed. Morasca and Capizzano [Morasca and Serra-Capizzano, 2004] present an analytical technique that is based on the comparison of the expected values of the number of failures caused by the applications of testing techniques, based on the total ordering among the failure rates of input sub-domains. They have also reviewed other approaches that compare techniques using expected number of failures caused or the probability of causing at least one failure.

The second stream of research in evaluation of dynamic technique is developing framework or guidelines for comparing and thus selecting an appropriate testing technique for a given problem domain. Probably the most commonly considered attributes of test techniques are their efficiency, effectiveness, and applicability in detecting errors in programs. However, the major problems with these comparison frameworks are that they treat all types of faults and the underlying programs on which these techniques are to be evaluated as equal which can affect validity of such comparison results.

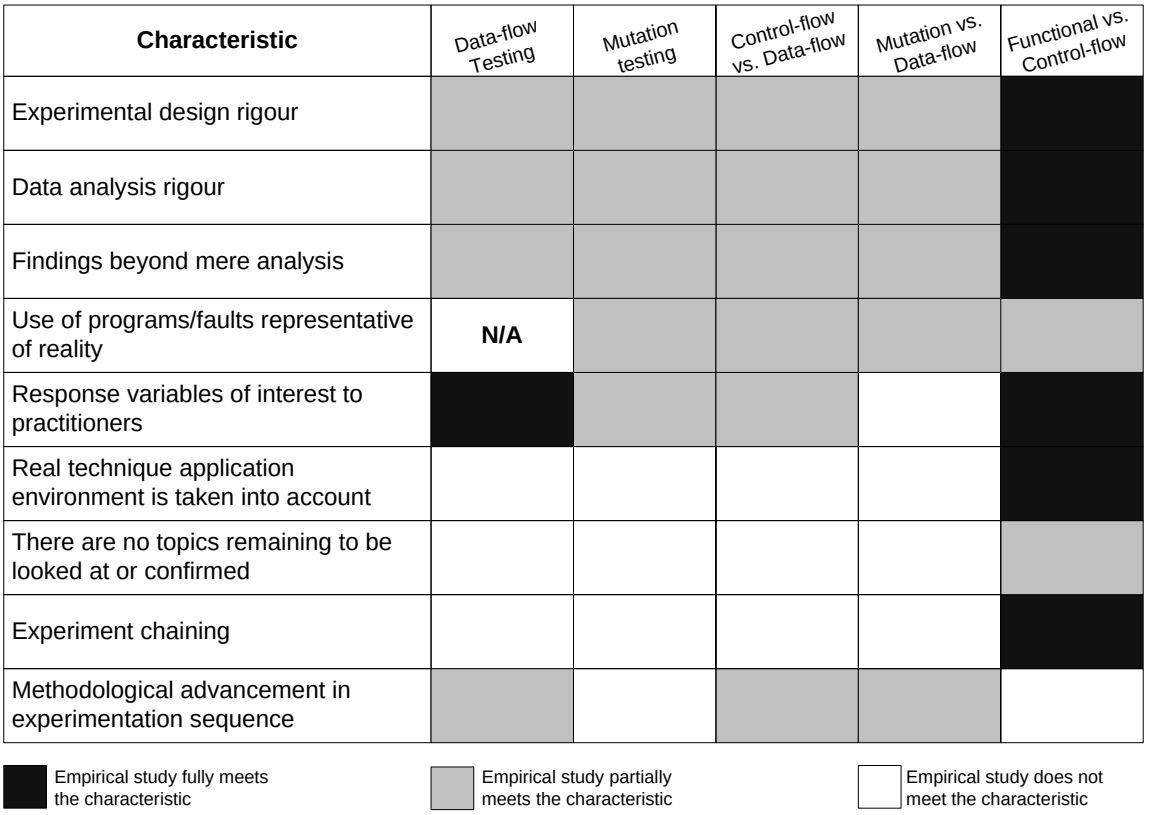


**Figure 4.10:** *Study Maturity by Families*

***Evaluation results***

Juristo [Juristo et al., 2002], [Juristo et al., 2004a] performed very comprehensive analysis of several years of empirical work over testing techniques. She has highlighted following issues with current studies namely,

- informality of the results analysis (many studies are based solely on qualitative graph analysis)
- limited usefulness of the response variables examined in practice, as is the case of the probability of detecting at least one fault
- non-representativeness of the programs chosen, either because of size or the number of faults introduced
- non-representativeness of the faults introduced in the programs

An analysis of the maturity of empirical studies of various testing techniques has been given in [Juristo et al., 2004b]. Figure 4.10 has been adapted from the summary given therein. Additionally, Briand and Labiche [Briand and Labiche, 2004] discussed issues facing empirical studies of testing techniques. Criteria to quantify fault-detection ability of a technique is one such issue, while threats to validity arising out of the experimental setting (be it academic or industrial) is another. They suggest using (common) benchmark systems for such empirical experiments and standardizing the evaluation procedures.

## 4.4 Evaluation of Tools

***Why evaluate tools?***

Evaluation of testing tools is important for many reasons. Due to an overwhelming number of testing tools available in the market, the decision to select the best tool remains elusive. Subjective and objective evidence about candidate tools before is needed before arriving at a final choice among them. Only systematic guidelines and precise criteria to compare and evaluate tools is the befitting solution to this problem. This kind of quality evaluation, when at hand, establishes our confidence in the capability of the tool in solving our testing issues. This section deals with existing research work which has focused on developing procedures and criteria for testing tools evaluation.

***Evaluation Criteria***

Many different subjective and objective criteria have been suggested in tool evaluation techniques. Typical sets of criteria which could contribute to a tool's evaluation or affect its selection are;

- Quality attributes
  - Reliability
  - Usability
  - Efficiency
  - Functionality
  - Maintainability
  - Portability
- Vendor qualifications
  - Profile
  - Support
  - Licensing
- Cost
  - Purchasing & installation
  - Training
- Organizational constraints
- Environmental constraints
  - Lifecycle compatibility
  - Hardware compatibility
  - Software compatibility

***Levels of tool evaluation***

Evaluation of tools may be meaningful at three different stages in development and test process. First and perhaps the most important stage is when one feels that he/she needs a tool to automate testing tasks, has several candidate tools available at hand, and wants to make a judicious choice of selecting and implementing the most relevant tool matching specific needs and constraints. This is the pre-implementation stage. Second is the in-process stage. It is when we are in the middle of our test process and we want to track and control progress of our testing tasks. At this state it would be interesting to see number of test cases run in comparison to time, number of faults detected etc. A quite similar and third level of evaluation will be helpful when we are finished with a project and we want to assess what we have spent for a tool and what have we gained. If a tool is found to do well according to our cost benefit analysis, it will likely be re-implemented for next projects or otherwise. This third point of tool evaluation is a kind of post-implementation evaluation.

### 4.4.1 Pre-Implementation Analysis/ Tool Selection

***Evaluation methods***

Most test tool evaluation approaches belong to the type of pre-implementation analysis which involves assessing a tool based on certain criteria. The assessment results are used by a subsequent tool selection process. IEEE Standard 1209 [IEEE, 1992] distinguishes between evaluation and selection as, "evaluation is a process of measurement, while selection is a process of applying thresholds and weights to evaluation results and arriving at decisions".

A short discussion of some well known such evaluation techniques is given below.

- **IEEE Standard 1209, Recommended Practice for the Evaluation and Selection of CASE Tools** [IEEE, 1992]: This standard comprises three main sections; *evaluation process, selection process*, and *criteria*. The evaluation process provides guidelines on determining functionality and quality of CASE tools. The section on selection process contains guidelines on identifying and prioritizing selection criteria and using it in conjunction with evaluation process to make a decision about a tool. The third section of the standard is criteria which is actually used by evaluation and selection process. It presents a framework of tool's quality attributes based on ISO 9126-1 standard.

- **Lewis' Methodology to Evaluate Automated Testing Tools** [Lewis, 2004, Ch. 30]: Lewis provides step-by-step guidelines for identifying tool objectives, conducting selection activities, and procuring, implementing, and analyzing the tool.

- **Task Oriented Evaluation of Illes et al.** [Illes et al., 2005]: They have defined functional and quality criteria for tools. Quality criteria has been specified using set of several quality attributes and sub-attributes influenced from ISO 9126-1 standard. The functional criteria are based on a task oriented view of the test process and tools required for each test process phase are described. Their approach attempts to avoid laboratory test by forming the criteria which can be analyzed based on tool vendor's provided instructions.

- **Miscellaneous**: Some un-structured guidelines in this regard have been presented by Fewster and Graham [Fewster and Graham, 1999, Ch. 10], Spillner [Spillner et al., 2007, Ch. 12] et al. and Perry [Perry, 2006, Ch. 4]. The authors have discussed various implications involved with selecting, evaluating, and implementing test tools. Perry [Perry, 2006] suggests considering development life cycle, tester's skill level, and cost comparisons for tools. Another similar example is Schulmeyer and Mackenzie's test tool reference guide [Schulmeyer and MacKenzie, 2000, p. 65].

### 4.4.2 In-Process & Post-Implementation Analysis

***Evaluation methods***

Dumke [Dumke and Grigoleit, 1997] provides an example of scenarios to evaluate efficiency of a similar class of tools (CAME-computer assisted software measurement and evaluation tools). However, very few specific methods exist for an in-process evaluation of test tools. In this regard, a quantitative criteria presented by Michael et al. [Michael et al., 2002] can be used both during the test process and also as a post-implementation analysis. They proposed several metrics for the purpose which are named below.

- Tool Management
- Human Interface Design
- Maturity & Customer Base
- Maximum Number of Parameters
- Test Case Generation
- Estimated Return on Investment
- Maximum Number of Classes
- User Control
- Ease of Use
- Tool Support
- Response Time
- Reliability
- Features Support

***Short critical review***

In contrast to many evaluation works over testing techniques, search into existing literature resources over evaluation of test tools returned very few results. It seems that the development of new testing tools has been given far more attention than analysis, measurement, and comparison among existing tools. Based on the above discussion it can be observed that systematic test tool selection and evaluation involves several steps, which include;

1. Principal decision to use a tool
2. Understanding concerned testing tools
3. Identification of tool requirements
4. Pre-evaluation

5. Selection
6. Post-evaluation

## 4.5 Typical Characteristics of Test Evaluations

After the widespread survey of all forms of prevalent test evaluations, it has become possible now that the chief characteristics of these evaluations be identified which embody their strength, effectiveness, and applicability. With the list of these characteristics, we would be able to arrive at a superset of common features that form the criteria upon which we can analyze selected evaluation approaches.

### 4.5.1 Measurement

The application of software measurement for software engineering evaluations is widely accepted as an effective technique. International standards on process and product quality give pivotal place to measurement. The most well known process maturity model (CMMI) contains a dedicated measurement & analysis process area, while part 3 and 4 of its counterpart standard for product quality (ISO 9126) also concentrate on product measurements. Software measurement has successfully been exercised in a variety of evaluation approaches [Dumke et al., 2006a], [Ebert et al., 2004] and is seen as one of the critical success factors for process evaluation and improvement [Dyba, 2005].

Apart from its unquestionable significance for process, product, and resource evaluations, software measurement has been a key player in all forms of test evaluation approaches discussed in this chapter. All the assessment models of test process place measurement as a requirement in higher maturity levels. For example, it is mentioned as a maturity goal at level 4 in TMM, as a key area in TPI, and as a process area at level 4 in TMMi. In case of test process evaluations, measurement has been found to facilitate understanding, controlling, monitoring, predicting, and making objective decisions during the course of action of process activities. Measurement and analysis of test process activities is a helpful tool for process managers who need to constantly track process progress and to identify improvement needs. A majority of evaluation approaches of test techniques reviewed in this chapter also used test measurements. The empirical evaluations of both static and dynamic techniques used some type of test measurements to compare their relative effectiveness. The methodologies to evaluate test tools specifically at the in-process and post-implementation levels heavily used cost and defect measurements for finding tool's effectiveness or efficiency.

As a consequence of these observations, exploitation of test measurements is regarded as an important element for any test evaluation approach to be effective and efficient. References to other research/practice resources are also being reiterated below which motivate the choice of this requirement,

- Requirement of a test measurement program in TMM [Burnstein, 2003, Ch. 16] maturity goal 3.4 (maturity level 3) and maturity goal 4.2 (maturity level 4)
- Requirement of a test measurement program in TMMi [Goslin et al., 2008b] process areas 4.1 and 4.2 (maturity level 4)

- Application of test metrics mandated by TPI [Koomen and Pol, 1999] metrics key area
- Requirement of metrics-based process assessment by [Hamann, 2006, p. 48]

### 4.5.2 Compliance with Standards

Software engineering standards represent the sets of best practices to do some activity. A large number of standards have been developed by various organizations. Well known software testing related standards have mostly been developed by IEEE and Special Interest Group in Software Testing of British Computer Society. On many occasions, other standards designed mainly for the general classes of software engineering problems have also be used to develop testing related approaches.

In the field of software testing, many models and techniques for the evaluation of process and tools contain references to international software engineering standards. For example, the concepts, structures, and contents of the Testing Maturity Model (TMM) and Test Maturity Model Integration (TMMi) both follow the ideas of CMM/CMMI process maturity and improvement standards. The Test Process Improvement (TPI) model is also linked strongly with the TMap (Test Management Approach) pseudo standard. These references of the test process maturity models to standard models on one hand offers familiarity of the approach while on the other hand facilitates its integration into any existing process improvement program. The existing evaluations of testing techniques are mostly based on some kind of experimental analysis and do not refer to any standards in this regard. In case of evaluation of test tools, the IEEE Standard 1209 Recommended Practice for the Evaluation and Selection of CASE Tools can be cited as an example of standard-based test tool evaluations.

Therefore, referencing and complying with relevant international software engineering or testing standards is an unavoidable necessity in developing any test evaluation approach. References to other research/practice resources are also being reiterated below which motivate the choice of this requirement,

- Requirement of following process assessment standards [Braungarten, 2007, Section 3.4]
- Reference to use of *best practices* as SPI's strength [van Solingen, 2000, Section 4.5.3]
- Remark about immaturity level of software testing by [Bertolino, 2004]
- Active application of software testing standards identified by testing practices survey made in Australia by [Ng et al., 2004]
- Test standardization marked as the second most important research issue identified in a survey by [Taipale et al., 2005]
- Active interest in standards based software testing exemplified by an under-development software testing standard ISO/IEC 29119 System and Software Engineering: Software Testing

### 4.5.3 Implicitness vs. Explicitness

In the context of social and natural sciences, any kind of evaluation is distinguished either as implicit (relatively automatic and perhaps unconscious) or as explicit (deliberate, controlled and conscious). However, in the field of software engineering it means slightly different for an approach to be of implicit or explicit type. Implicit models use subjective methods for assessing quality of an object while explicit models use objective and direct measurement techniques to evaluate their targets. Based on the peculiarities of the discovered evaluation methods in the area of software testing, the division between implicit and explicit models as held by the author of the thesis could be summarized like this,

- *Implicit models subjectively evaluate the practices of testing processes against a reference standard, and solely focus on the maturation of the processes. Primarily, they investigate **how** testing is being done.*
- *Explicit models are those that directly and objectively evaluate actual performance of the testing processes. Primarily, they investigate **how well** testing is being done.*

Attempting to match the evaluation approaches of test processes, techniques, and tools reviewed earlier in this chapter, it can be observed that test process maturity models or other process assessment/improvement models fit well into the *implicit* category. These models attempt to subjectively evaluate practices of an actual testing process against those practices which are believed to enable efficient testing. Such maturity models are based on the widely held belief that a better software process produces better products. However, it is still a kind of indirect way to evaluate testing processes. In contrast to the processes, most evaluation methods for test techniques are of explicit kind since they directly attempt to measure some quality attribute of the techniques through the use of test metrics. This avoids the possibly biased human judgements and provides an objective view of the quality of the evaluated object. The evaluation techniques for test tools almost follow the same style of explicit evaluations.

A very recent example of an explicit approach to assessment of software processes is that of Tarhan and Demirörs [Tarhan and Demirörs, 2008] where they have applied several process metrics for a quantitative analysis of process progress. Although that the benefits of the implicit models of test process evaluation cannot be ignored, an explicit or at least a hybrid kind of test evaluation similar to Tarhan and Demirörs' approach mentioned above has the potential of providing effective and accurate results.

References to various research/practice resources are being reiterated below which motivate the choice of this requirement,

- Need for explicit process quality evaluations mentioned in [van Solingen, 2000, Section 4.5.1, 4.5.2]
- Application of direct process measurement approach by [Tarhan and Demirörs, 2008]
- Need for explicit assessment of testing practices mentioned as one of the seven principles of software testing by [Meyer, 2008]

### 4.5.4 Cost

***SME defined***

All across the globe as well as in Europe, a vast majority of software development companies exist on the scale of small and medium-sized enterprizes. Small and medium-sized enterprizes are companies the number of whose employees or turnover is below certain thresholds. European Commission categorizes such companies more precisely as *micro*, *small*, and *medium-sized* enterprizes distinguishing them on the basis of *staff headcount*, *annual turnover*, and *annual balance sheet*. As per these categories, micro, small, and medium-sized enterprizes (SMEs) are those companies who have fewer than 250 employees, do not earn more than 50 million euro, and whose annual balance sheet does not exceed 43 million euro. These thresholds are slightly different for companies in the United States or few other countries to be classed as SMEs. Following any interpretation of SMEs, majority of all sorts of businesses across the globe are SMEs with their proportions varying between 70-90% for different countries.

***Software SMEs***

IT business or more specifically that of software development is an ideal choice for people having motivation to set up a small technology firm. Unlike other businesses requiring considerable investments, infrastructure, and personnel, setting up a small software firm is a much simple piece of chore. Perhaps that is the reason behind the fact that most software companies fall in the category of SMEs. The often quoted proportion of SMEs in the IT/software business is between 80-90% for various countries.

***Organizational challenges***

Both large and small software development companies have to meet almost the same set of technical and business requirements. They strive to improve the quality of their products and processes, keep up with the new technologies, maintain competitive advantage, and sustain economic fluctuations. However, SMEs have limited resources to face all these challenges. Cost is the first major issue for SMEs. Their monetary resources are highly limited and they cannot make any long term investments the fruit of which could be realized only after a considerable period of time. Lack of human resources is another obstacle. Often a person is playing more than one role which limits his or her ability to perform any tasks requiring dedicated effort. Absence of adequately trained staff marks another disadvantage that SMEs have over larger firms which leaves SMEs unable to undertake complex tasks to optimize the quality of their services.

***Software process initiatives for SMEs***

It is a strong perception in software development industry that CMMI and other similar software process assessment/improvement approaches are not suited to SMEs. In a recent survey [Staples et al., 2007], organizational size, cost, and time ranked as the top three hitches for the organizations not adopting CMMI. Acknowledging the need for specialized methodologies matching the constraints of SMEs, several software process assessment and improvement approaches have been developed. Several regional solutions in this direction include process improvement models such as CMM Fast-Track, MoProSoft, Agile SPI, QuickLocus, and COMPETISOFT [Oktaba and Piattini, 2008], and process assessment models such as Micro-Evaluation, MARES, SPM, RAPID, FAME, and EAP [Zarour et al., 2007]. In contrast to all these assessment and improvement mod-

els for software process, unfortunately no specialized evaluation model for test processes could be found in literature.

***Requirements on assessment models for SMEs***

The organizational challenges facing SMEs mandate some special requirements on the process evaluation models suited to them. Even if the size of the organization is small, they need the same level and content of the assessment results and improvement suggestions as sought by larger companies. Instead, their assessments have to be even more effectively designed and implemented. As analyzed by Richardson [Richardson, 2002] and Zarour [Zarour et al., 2007] some of the characteristics that any SPI model for SMEs should support are that they,

- relate to the company's business goals
- focus on the most important software processes
- give maximum value for money
- propose improvements which have maximum effect in as short a time as possible
- provide fast return on investment
- be process-oriented
- relate to other software models
- be flexible and easy-to-use

Some of these requirements are already on the priority list of large-scale process assessment/improvement models. Still few others are vague as they are subjective and opinion centric in nature rather than objective. For example, relating to business goals and providing effective improvement suggestions are the features claimed by almost all assessment/improvement approaches, yet the validity of the claim is difficult to be judged objectively. What remain as the visible characteristics are the time, personnel, and any specialized assessor training needed for the assessments which are chief characteristics of light-weight process assessments. It can be concluded that expensive and large-scale evaluation models of software process or testing process are inappropriate for SMEs unless they are adapted to specific constraints. Light-weight evaluation approaches of test process are, therefore, an obligation for small companies while a choice for larger ones.

References to various research/practice resources are being reiterated below which motivate the choice of this requirement,

- Cost as a barrier to adoption of software testing approaches observed in a testing practices survey made in Australia by [Ng et al., 2004]
- Personnel costs identified as the highest cost item in testing in a dissertation on testing practices by [Taipale, 2007]

## 4.6 Summary

Extending the survey performed in chapter 2, the current chapter identified all existing forms of evaluation in the testing area. As a result, the chapter has partially addressed the second part of the first sub-question of the research task. The fulfilled part of the research question has been *italicized* as shown below,

> RT-1. Traverse the area of software testing to identify entities of evaluation. *Explore existing evaluation forms and approaches as applied in this field. Investigate characteristics of low-cost and effective test evaluations.*

This chapter was solely dedicated to the survey of all forms of evaluations prevailing in the area of software testing. An earlier chapter (chapter 2) had already given a glimpse of the possible entities of testing that could be evaluated. Evaluation of testing process was considered first. History of the test process evaluation approaches was crawled in depth. It resulted in identification of seven approaches most of which were maturity models. Out of these, three significant and living models were selected for further detailed elaboration. The concept, structure, and applications of these models were discussed. Focus of the chapter then moved on to evaluation of testing techniques. A previous chapter had divided these techniques into two broad classes as static and dynamic. Based on the observation of available evaluations and upon intrinsic characteristics of their application, it seemed appropriate to discuss evaluations of these two kinds of techniques separately. It was observed that in the category of static techniques, reviews and different forms of inspections caught more attention than any other static techniques. The dynamic techniques were evaluated much more often than the static techniques. The evaluations were largely focused on finding effectiveness and efficiency of these techniques. Testing tools was the third element of testing that was subjected to evaluations. Corresponding to different stages in the testing process where a tool is involved, different evaluation methodologies, guidelines were discovered. One of these guidelines for evaluation and selection of tools was found in the form of an IEEE standard. Diagnosis of these numerous forms of test evaluations enabled the author to discover characteristics of successful test evaluations which were given in the last section of the chapter. These characteristics are being summarized below and form part of the criteria against which the related literature will formally be analyzed in the next chapter.

Desired Characteristics of Test Process Evaluations

- ✓ The test evaluations must exploit quantitative assessment techniques.
- ✓ The test evaluations must follow standard approaches wherever applicable.
- ✓ The test evaluations must be light-weight in nature.

✓ The test evaluations must be explicitly designed.

# 5 Analysis of Related Work

The previous chapter provided an in-depth survey of existing evaluation approaches in the area of software testing. Moving a step further, the current chapter critically reviews and objectively analyzes these approaches for finding their strengths and weaknesses. The chapter is aimed to establish a relationship between available evaluation approaches and the research problem at hand.

## 5.1 Introduction

After an exhaustive review of testing literature for finding available test evaluation approaches it was discovered that alongside existence of a couple of models and techniques the evaluation and improvement of testing process remains an active research issue in the testing community. It is necessary that these existing approaches be analyzed in the purview of the current research problem. This analysis has to be based on a clearly established yardstick.

In finding an answer to the sub-questions of research (RQ-1 and RQ-2), the previous chapters have already determined few criteria of merit relating to foundation, structure, content, and applicability of the approach. These aspects of test evaluations are a direct consequence of the research for finding a solution to the current problem sought by this thesis. The requirements for a possible solution mentioned in section 3.4 and 4.6 are being summarized here in the form of a definite criteria set,

Criteria of Evaluation for Related Work

**C1 –** Scientific rigor:
*The test process evaluation approach must have been developed on a sound theoretical and scientific foundations of evaluation.*

- **C1.1:** *The approach must explicitly define and delimit the target, i.e. the entities interesting for evaluation.*
- **C1.1:** *The approach must explicitly define particular criteria to be considered about the entities of interest.*
- **C1.1:** *The approach must provide a standard or yardstick against which the target process could be compared.*
- **C1.1:** *The approach must describe and explain the type of techniques that will be used to assess the target against the criteria.*
- **C1.1:** *The approach must specify the synthesis techniques that will be used to integrate the collected assessment or measurement data.*

**C1.1:** *The approach must explain the steps of the evaluation process.*

**C2 –** Utilization of Measurement
*The test process evaluation approach must exploit quantitative assessment techniques.*

**C3 –** Utilization of Standards
*The test process evaluation approach must have been developed using standard approaches wherever applicable.*

**C4 –** Size
*The test process evaluation approach must be light-weight in nature.*

**C5 –** Style
*The test process evaluation approach must make explicit assessments of process quality.*

Building upon the previous analysis of test process evaluation models by Swinkels [Swinkels, 2000] and by the author himself [Farooq et al., 2007], [Farooq and Dumke, 2007], this chapter critically analyzes the selected approaches against the criteria of merit presented above. The author of the thesis was unable to find any other survey/analysis of test process assessment/improvement approaches other than the three just mentioned. This analysis will be a sort of characteristics-based comparison [Halvorsen and Conradi, 2001] of test process evaluation/improvement models. Such kinds of criteria-based analyses have earlier been performed of SPI approaches in other PhD dissertations involving research over evaluation and improvement of software process. These research works are being mentioned very briefly here,

[van Solingen, 2000]:
Criteria for a product-focused SPI approach for embedded systems derived from strengths and weaknesses of SPI methodologies

[Komi-Sirviö, 2004]:
Criteria for development and evaluation of SPI methods derived from critical success factors of existing SPI approaches

[Hamann, 2006]:
Criteria for an integrated approach for SPI derived from analysis of software process assessment models

[Braungarten, 2007]:
Criteria for development and evaluation of software measurement process improvement model derived from a survey of software measurement literature and measurement process improvement models

## 5.2 Analysis of Existing Approaches

Chapter 4 has revealed three types of test evaluation approaches, for test processes, test techniques and test tools. By design, the evaluation/improvement models at the process level encompass other elements of evaluation such as techniques and tools as well. Only these process level approaches will be analyzed here as examples of evaluation models covering all aspects of software testing. Details about underlying principles, structure, and content of shortlisted models among these have already been presented in chapter 4. This section will analyze only these selected models (TMM, TPI, and TMMi) against the criteria mentioned above.

### 5.2.1 Analysis of Testing Maturity Model (TMM)

***C1–scientific rigor***

Development of TMM was a result of a university research project completed in 1996. It was backed by practical testing knowledge extracted from a diverse set of resources. The fact mentioned earlier in chapter 4 is being reiterated here that this model's scientific foundations were built upon Beizer's progressive phases of a tester's mental model [Beizer, 1990] while the testing foundations were taken from Gelperin and Hetzel's evolutionary testing model [Gelperin and Hetzel, 1988]. Furthermore, the structural elements were based on the concepts of Capability Maturity Model (CMM) and the best approaches derived from Durant's survey of industrial testing practices [Durant, 1993]. This combination of research and practice thus works as the backbone of this maturity model making it a well founded approach.

Although that the author of the TMM did not make any specific reference to the concepts of evaluation as mentioned in chapter 3, the model yet seems to provide the six evaluation components mandated by this first criterion (3.4). A little careful comparison of TMM model against these evaluation components, shows that it actually contains all of these components. For example, the target of the TMM is clearly all types of software testing processes. The maturity goals correspond with the broad categories of test process issues that have to be considered for evaluation. Thy make up the set of criteria aspired by the evaluators. The overall maturity model including the maturity sub-goals and the TMM questionnaire give a picture of an ideal testing process at different levels of maturity. It represents the standard or yardstick against which an actual instance of a testing process is to be compared. The model also describes an assessment procedure which is aimed at internal and self evaluations. The procedure describes assessment techniques (questionnaire, interviews, forms, document reviews etc), techniques of synthesizing results (ranking procedures for sub-goals and goals etc), and all relevant details of the assessment steps. Based on these discussions, it can be assumed that TMM model fully satisfies all the required elements of criterion of scientific rigor.

***C2–utilization of measurement***

To analyze the involvement of quantitative techniques, two elements of TMM need to be considered, its assessment model, and the TMM reference model itself. Considering the TMM reference model, the role of measurement for process monitoring, control and evaluation is emphasized in three maturity goals at two different maturity levels. Particularly, the maturity level 4 concerns with management and measurement of testing processes. Two

**Table 5.1:** *Mapping between Evaluation Components and TMM*

| Evaluation Component | Testing Maturity Model |
|---|---|
| Target | Software testing processes |
| Criteria | Maturity goals |
| Yardstick | Sub-goals, Questions |
| Assessment techniques | Questionnaires, Interviews, Forms, Document reviews |
| Synthesis techniques | Ranking procedure, Traceability matrix |
| Evaluation process | Assessment procedure |

maturity goals at this level emphasizes the use of measurements for controlling and monitoring the testing process. References to measurement made by TMM are summarized below through the extracts obtained from [Burnstein, 2003, pp. 648-651],

***Maturity Goal 3.4**: Control and monitor the testing process*
*The purpose of this maturity goal is to promote development of a monitoring and controlling system for the testing process so that deviations from the test plans can be detected as soon as possible,...*

- *Maturity subgoal 3.4.1*
  *An organizationwide committee or group on the controlling and monitoring of testing is formed and provided with funding and support. The committee develop, documents, distributes, and supports...and measurements for controlling and monitoring of testing.*
- *Maturity subgoal 3.4.2*
  *Test-related measurements for controlling and monitoring are collected for each project.*

***Maturity Goal 4.2**: Establish a test measurement program*
*The purpose of test measurement program is to identify, collect, analyze, and apply measurements to support an organization in determining test progress, evaluating the quality and effectiveness of its testing process, assessing the productivity of its testing staff, assessing the results of test improvement efforts, and evaluating the quality of its software products.*

- *Maturity Subgoal 4.2.1*
  *An organizationwide committee or group focusing on developing a test measurement program is formed and provided with funding and support. The committee develops, documents, distributes, and supports procedures, goals, policies, and measurement as applied to software artifacts and the test process...*
- *Maturity Subgoal 4.2.2*
  *A test measurement program is developed according to policy with a measurement reporting system....*

- *Maturity Subgoal 4.2.3*
  *Training, tools, and other resources are provided to support the test measurement program.*

***Maturity Goal 4.3**: Software quality evaluation*
*The purpose of software quality evaluation maturity goal is to relate software quality issues to the adequacy of testing process, define and promote use of measurable software quality attributes...*

- *Maturity subgoal 4.3.1*
  *An organizationwide committee or group focusing on software quality evaluation is formed and provided with funding and support. The committee develops, documents, distributes, and supports ... and measurement for software quality evaluation.*
- *Maturity subgoal 4.3.3*
  *Quality goals are developed for each project according to policy. The testing process is structured, measured, and evaluated to ensure that quality goals are achieved....*

There is no doubt that the mentioned measurement activities are necessary for effective management of the testing process, but these are only very abstract and high-level guidelines. They merely tell *what* has to be done, the information on *how* it can be done is missing.

The *how* element of this test measurement has fractionally been explained by Burnstein [Burnstein, 2003, Ch. 9, 11] when she exemplifies several measurements organized by the applicable TMM level or by their purpose for controlling and monitoring testing process. She further lists some steps of a general measurement program without any reference to testing related issues. It still lacks particular details of test measurement program as required by maturity goal 4.2 explained above. It can be concluded that apart from stressing the need for test measurement and partly describing the peculiarities of the measurement process, the TMM model lacks guidelines on designing and implementation of measurements for the test process evaluations.

As for the TMM assessment model [Burnstein, 2003, Ch. 16] notable lacks are observed from measurement point of view. The model consists of three major components a) team training and selection criteria, b) the assessment procedure, and c) the assessment instrument (questionnaire). The TMM questionnaire itself is composed of eight parts. Three of these parts comprise maturity goal questions, testing tool questions, and testing trends questions while rest of the parts are of informative nature. This structure of assessment components is visualized in figure 5.1.

Performing the assessment involves determining the answers to these questions through interviews, inspections, and other such subjective data gathering techniques. The questions themselves are somewhat objective or subjective in nature. However, the assessment procedure by no means exploits any process measurements for ranking maturity subgoals and goals. In fact, with the nature of questions the role of measurement even seems irrelevant in determining their answers. In the absence of measurements for assessing and evaluating testing process the criteria of measurement is found to have failed.

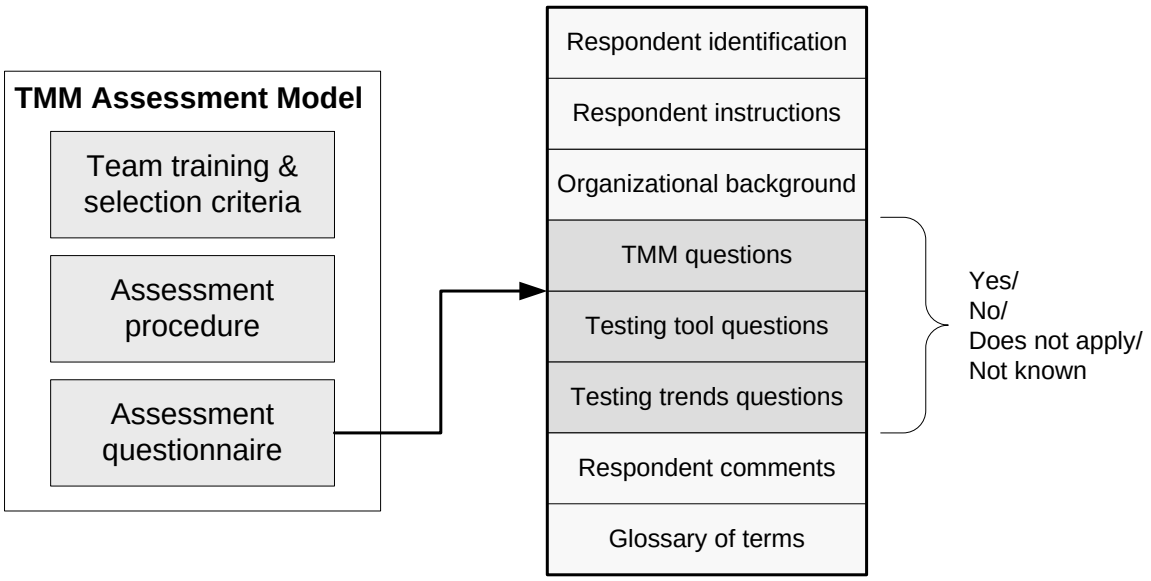


**Figure 5.1:** *Components of TMM Assessment Model*

***C3–utilization of standards***

The concept and structure of TMM's maturity model and assessment model need to be considered for compliance to relevant standards. The use of standards makes such a model to be easily integrated with other standard or well-known process assessment and improvement approaches. The structural framework of TMM clearly resembles that of CMM. There is one-to-one correspondence between the both. First considering the TMM from maturity model point of view, it can be observed that TMM was developed to serve as a complementary model to CMM. Perhaps that was the reason that it derived most of CMM's concepts of building a maturity model. TMM also has five levels of maturity, follows a staged improvement model, has maturity goals/subgoals similar to process areas/goals of CMM, and defines activities/tasks/responsibilities (ATRs) resembling key practices of CMM. In essence, TMM provides an expanded picture of testing related process areas of CMM. This matching structure of TMM makes it a suitable choice for organizations already using CMM/CMMI-like process improvement models.

Considering the construction of TMM's assessment model, Burnstein [Burnstein, 2003, p. 549] mentions the underlying principles in these words,

> The *CMM and SPICE Assessment Models were used to guide development of the TMM Assessment Model. The goal was to have the resulting TMM-AM compliant with the Capability Maturity Model Appraisal Framework so that organizations would be able to perform parallel assessment in multiple process areas.*

Accordingly, TMM defines a SPICE like assessment model which is a kind of team-based self assessment instrument. TMM explains assessment procedures, although in not enough detail as SPICE, yet to be sufficient enough for evaluating the testing process. The style of the TMM's maturity and assessment model makes it familiar to people and organizations already using CMM/CMMI. Based on the observations, it can be assumed that TMM followed standard approaches in its design and implementation and hence the criteria C-3 is fairly fulfilled.

***C4–size***

TMM is based on the principle of self (team-based) assessment and does not require an external certification body for the evaluations. The assessment procedure is carried through several short steps. The necessary assessment questionnaire, ranking procedure for rating maturity goals and subgoals, and the guidelines on determining overall maturity levels are all part of the assessment model. Appropriate format of the assessment outputs are exemplified with some sample forms describing test process profile. The structure and details of the assessment model give the reflection that the evaluations can be performed in a relatively short period of time. However, some aspects of the assessment team may raise some issues. For example, writing about the size of the team Burnstein [Burnstein, 2003, p. 550] mentions,

> *The size of a TMM assessment team may vary, and depends on the scope of the assessment, the experience level of the team, and the size and complexity of the organizational projects being assessed. A team size of 4-8 members is suggested.*

The mentioned team size may be enough large for small organizations. A short training for the team has also been suggested by TMM. Most small and medium companies would be reluctant to dedicate a few of their staff members for the assessment, sacrificing their other precious responsibilities. However, despite these observations the overall consideration of the assessment model finds it a light-weight evaluation and improvement model.

**C5–style**

TMM's assessment procedure involves collecting answers to the questions designed for each of the subgoal/goal. These questions check the existence of best testing practices that are believed to have worked well. Based on the responses to these questions, subgoals and then goals are ranked by a pre-determined scheme from which an overall maturity level is calculated. Since the descriptions of goals/subgoals are themselves somewhat abstract and implicit, so are the questions prepared to check if they are satisfied are not. In this way the testing process is assessed by conformance of actual testing practices with these sets of questions/best practices. In essence, this kind of evaluation actually determines *what* is being done. It is equivalent to the interpretation of implicit approaches explained earlier in a previous chapter in section 4.5.3. The TMM's assessment techniques do not consider what test process has achieved, i.e. the actual performance of the testing process which can be gathered through direct process measurements. Based on this point of view of implicit and explicit process evaluations, TMM is considered to be an implicit model.

### 5.2.2 Analysis of Test Process Improvement Model (TPI v1.0)

**C1–scientific rigor**

Testing professionals working for IQUIP, a Dutch firm now known as Sogeti, designed TPI mainly based on their industrial knowledge and experience gained within a specific field of application (administrative automation). Although the concept of the TPI model originated out of the personal reflections of its designers, yet it compares well with most other maturity models of its kind. For example, its key areas are similar to maturity goals of TMM and process areas of CMM. The concept of checkpoints is a renamed version

of TMM's activities/tasks/responsibilities and CMM's key practices. Thus, TPI's model structure resembles to that of other well established approaches in its area.

Comparing the model structure with evaluation components as outlined in the sub-criteria of first criterion C1, it is observed that TPI does not cover all of these components. First, although that TPI model applicable to broad testing situations it is mainly designed for structure testing. This serves the target of the model. The 20 key areas relate to different aspects of the testing process to be considered for improvement and hence for evaluation too. The key areas are the criteria of TPI's evaluations. The checkpoints compose an ideal picture of a key area at a given level. Together they make the yardstick with which the actual test activities have to be matched. Except for mentioning interviews and document reviews for conducting assessment, and a maturity matrix to draw final evaluation results, TPI lacks further details on assessment and synthesis techniques. However, the whole evaluation process component is provided in enough detail when its authors explain the application of the model [Koomen and Pol, 1999, Ch. 6].

**Table 5.2:** *Mapping between Evaluation Components and TPI*

| **Evaluation Component** | **Test Process Improvement** |
|---|---|
| Target | Structured testing |
| Criteria | Key areas |
| Yardstick | Checkpoints |
| Assessment techniques | — |
| Synthesis techniques | — |
| Evaluation process | Assessment procedure |

It is observed that even if the TPI model is not a direct consequence of some scientific principles of process assessment and improvement, it has been demonstrated to work well in practice without any critical failures [Koomen and Notenboom, 2004]. The model has been found to partially comply with the requirement of scientific rigor.

**C2–utilization of measurement**

Like measurement related maturity goals and subgoals of TMM, TPI also has a key area named Metrics. This key area concerns with the use of metrics for progress monitoring, configuration management, and defect and change management. The model mentions examples of product, process, system, and organization metrics that should be used for monitoring and controlling the test process. Collecting sets of metrics along the project phases is a trivial task that somewhat mature organizations typically perform. However, this key area only partially addresses this aspect of test process while further guidelines on using these metrics for drawing meaningful process indicators are missing from TPI descriptions.

The TPI assessment procedure mainly mentions using qualitative assessment techniques such as interviews and analysis of documents. Although the checkpoints are designed to be as objective as possible, yet the procedure for determining if a checkpoint is met does not use any quantitative, viz. measurement techniques. Describing the information collection phase of the process assessment, TPI model says that [Koomen and Pol, 1999, p. 63],

By interviewing the participants, studying the documentation, and optionally by witnessing the process, the necessary information is collected. If the person who is interviewed gives too positive or too negative an account of a situation, a biased view can be presented. The interviewer should be aware of this. Studying the documentation can help to recognize this bias.

No matter that the TPI's process area acknowledge the significance of measurement for process monitoring, control and improvement, the assessment procedure still relies on qualitative and subjective information for deciding whether the checkpoints are met or not. Based on these findings, it can be inferred that TPI does not fulfil the requirement of measurement criteria.

**C3–utilization of standards**

The model is based on the methodology for structured testing called Test Management Approach (TMap) [Pol et al., 2002]. TPI serves as an example of implementation of TMap concept. The key areas of TPI are organized around the four corner stones defined by TMap–lifecycle of test activities (L), organization (O), infrastructure(I), and tools (T).

Although that TMap is well known in software testing industry across few countries of western Europe, it is not an international standard as yet. TPI could be a favorable choice of those professionals and companies already familiar with TMap concepts but may not be of others too.

Additionally, TPI's model elements have a close resemblance with other well known maturity model concepts in its area. However, the concept of its assessment procedure is not in line with any process assessment or measurement standards. In view of these considerations, TPI is not considered to be utilizing relevant international standards. The criteria C3 is therefore absent.

**C4–Size**

This criteria relates to the size of the effort spent on assessment procedures or the implementation of the model itself. A number of issues such as planning, assessment, and improvement actions that arise in the implementation of TPI model have been collected in a so called *change process* by the TPI's developers. The process is designed to be applicable to both small and large kinds of organizational settings and requirements.

Figure 5.2 details the steps inside this change process. Starting with the initial phases of awareness and target definition, the process iterates through assessment, improvement, and evaluation phases. Only broad guidelines for conducting these phases are available without reference to any low-level information such as the roles, techniques, and specific activities. The whole implementation process for the TPI-based process improvement does not entail significant overhead and the costs can be expected to be from low to medium scale.

The TPI's assessment procedure is mentioned to be consisting of preparation, information collection, analysis, and reporting sub-phases. The details given thereof are, although quite brief, yet are adequate set of instructions about the assessment steps, the techniques to be used, and the results to be produced. Thus, the TPI's assessment method matches with the principle of (team-based) self-assessment. Considering the information found about the implementation and assessment model, TPI seems to satisfy the requirement of being a light-weight solution.

**C5–style**

The assessment procedure of the model involves evaluating the validity of the defined

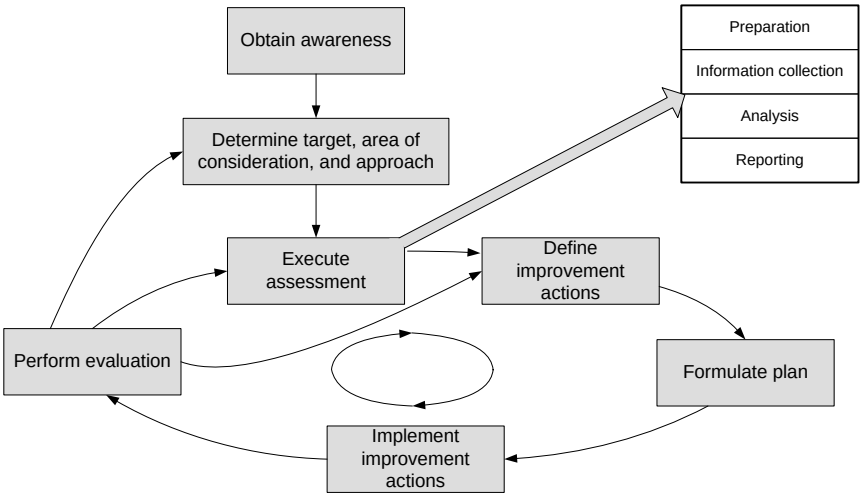


**Figure 5.2:** *TPI Change Process [Koomen and Pol, 1999, p. 56]*

checkpoints for various key areas and levels. TPI's checkpoints have been developed in the form of statements relevant to key areas and their levels. These checkpoints are no different than maturity questions of TMM. For example, one of the checkpoints says something like this [Koomen and Pol, 1999, p. 85],

Checkpoint for key area Test Strategy, level A, Strategy for single high-level test
- A motivated consideration of the product risks takes place, for which knowledge of the system, its use and its operational management is required.
- There is a differentiation in test depth, depending on the risks and, if present, the acceptance criteria: not all subsystems are tested equally thoroughly and not every quality characteristic is tested (equally thoroughly).
- etc.

Like TMM, TPI also assesses the quality of the testing process based on existence of the practices mentioned in checkpoints, not on the actual performance and achievements of the process itself. Therefore, the same argument of implicity of evaluation methods holds true for TPI leading to rejection of criteria C5.

### 5.2.3 Analysis of Test Maturity Model Integration (TMMi v1.0)

Although the development of the TMMi is still in process and only few interim documents have been released at the time of writing this thesis, a short analysis of this model will be included here based on whatever information is currently available about its structure and contents.

***C1–scientific rigor***

TMMi is not a product of a purely practice-oriented approach, its foundations are routed in an international research project which investigated supporting TMM with software metrics. Furthermore, other guiding principles have been taken from almost the same set of testing literature which formed the basis of TMM. Uniquely from other test improvement

models, TMMi also takes inputs from few international testing standards. TMMi has another advantage over TMM in that it includes new sets of best industrial testing practices extracted from surveys and the experiences of its development team.

Like TMM, this model also did not make any specific reference to evaluation theory mentioned in chapter 3. Nonetheless, TMMi's model elements match well with the six evaluation components mandated by the first criterion (C1). For instance, the target of TMMi is the whole testing process covering all levels and types of testing procedures. The different process areas represent the criteria which have to be considered. The generic and specific goals and practices, the sub-practices, and typical work products relevant to each of these process area describe the best way to testing. This serves as a yardstick against which an actual instance of testing process has to be compared. TMMi authorizes two kinds of assessment methods, an informal assessment without resulting in any capability determination, and a formal (external) assessment producing a formal capability rating against the model. Both these types of methods use interviews, questionnaires, document reviews, surveys etc as assessment techniques. All assessment related guidelines are given in a separate document titled TMMi Assessment Method Application Requirements (TAMAR) whose initial version is available in [Goslin et al., 2008a]. This TAMAR gives information on assessment techniques and synthesis techniques, and the evaluation process. Thus all six components of evaluation are provided by the TMMi model.

**Table 5.3:** *Mapping between Evaluation Components and TMMi*

| **Evaluation Component** | **Test Maturity Model Integration** |
|---|---|
| Target | Software testing processes |
| Criteria | Process areas |
| Yardstick | Generic/specific goals/practices, |
| Assessment techniques | Questionnaires, Interviews, Surveys, Document reviews |
| Synthesis techniques | TAMAR |
| Evaluation process | TAMAR |

Review of this information on TMMi's foundation, development, and model structure leads to satisfaction of the criteria of scientific rigor.

***C2–utilization of measurement***

It is expected that the measurement will occupy a significant place both in the reference model and any associated assessment instrument since the model's foundations come from a research project about metrics based maturity model for software testing process. Although that the full TMMi reference model is not yet available, the names of the process areas as such as test monitoring and control, test measurement, and software quality evaluation suggest strong association to process and product measures. However, any opinion about (level of) the involvement of measurement in TMMi will be premature since sufficient information about the model is not available yet and hence the criteria C2 will be left undetermined.

***C3–utilization of standards***

The model is claimed to be successor of TMM and a complementary approach to CMMI. Hence it inherits many of the concepts and elements from both of these two models. Like TMM/CMMI it develops the concept of a matured testing process growing through a gradual and incremental implementation of best testing practices. The number of levels and the degree of organizational maturity required to achieve those levels is almost the same as could have been for TMM/CMMI. Division of maturity levels into process areas and then into generic and specific goals exactly follows the CMMI structure. The specific and generic practices are also not unknown to organizations having used TMM or CMMI. Thus the model structure fully matches with the well established standard process improvement approaches.

An initial document titled TMMi Assessment Method Application Requirements (TAMAR) [Goslin et al., 2008a] is currently available which outlines key features of TMMi assessment method. The assessment method mentioned thereof is aimed to comply with the requirements of ISO 15504 process assessment standard and therefore uses ISO/IEC 15504 standard (specially its part 2) as its input. Section 5 of the TAMAR provides its one-to-one correlation with ISO/IEC 15504-2. In this way, TMMi's assessment method is in accordance with international standards on software process assessment.

The structure of the TMMi reference model and the style of assessment method qualifies TMMi against the requirement of following standard approaches.

***C4–size***

The five maturity levels of TMMi contain sixteen process areas altogether. Level 2 encompasses five process areas. Each of these process areas define several generic practices, specific practices (and sub-practices). Level 2 alone contains 60 generic practices, 69 specific practices (and at least more than 250 sub-practices). Each of the specific practices requires preparation of 2 to 4 items of typical work products (documents etc). The rest of the levels when developed would proportionally have similar number of elements. Implementation of TMMi at any given level means fulfilling all of these required elements and preparation of documents which would cost a considerable amount of time and effort often suited only to large organizations.

As for the assessment model, TMMi authorizes two types of assessment methods [Goslin et al., 2008a]. An 'Informal Assessment' performed by a single experienced assessor which is fast and brief scan of organizational activities carried through interviews, questionnaires, surveys, and document checks. This type of assessment is aimed at a raw evaluation of the organizational maturity in various process areas. No formal maturity level score against the model is produced in this case. This is a kind of light-weight and internal assessment. The other type is 'Formal Assessment' which is performed by an external assessment team lead by an accredited assessor. This type of assessment fulfils the requirements of the ISO/IEC 15504 standard for software process assessment. Accuracy of the data gathered through interviews, questionnaires, and document checks is cross verified with the use of multiple techniques. This formalized assessment which produces maturity level ratings is much more expensive since it has to check the existence of all the relevant practices, and work products in the course of process execution and can be carried out by external assessing bodies. Based on the choice of the method, the assessment procedure can be low or high cost, but the implementation of the model itself is beyond any doubt an expensive task. The requirement of the being a light-weight approach will not therefore be

**Table 5.4:** *Analysis of Test Process Evaluation & Improvement Models*

| Criteria | TMM | TPI | TMMi |
|---|---|---|---|
| **C1: Scientific rigor** | | | |
| **C1.1: Target** | √ | √ | √ |
| **C1.2: Criteria** | √ | √ | √ |
| **C1.3: Yardstick** | √ | √ | √ |
| **C1.4: Assessment techniques** | √ | × | √ |
| **C1.5: Synthesis techniques** | √ | × | √ |
| **C1.6: Evaluation process** | √ | √ | √ |
| **C2: Utilization of Measurement** | × | × | — |
| **C3: Utilization of Standards** | √ | × | √ |
| **C4: Size** | √ | √ | × |
| **C5: Style** | × | × | × |

considered met by TMMi.

***C5–style***

Despite that this model is not fully developed, yet it is clear that it is being developed on the lines of TMM/CMMI. The same argument of implicity that is applicable to TMM also seems to hold true for TMMi. Therefore, the TMMi model like other maturity models of its kinds will not be considered explicit.

## 5.3 Summary

Extending the survey of the test evaluation approached explained and reviewed in chapter 4, this chapter attempted a more objective analysis and evaluation of these approaches using the set of criteria established in chapter 2, and 3. In doing so, the chapter has achieved the third sub-question of this research work,

> RQ-3. Analyze available test process evaluation approaches based on the criterion developed in RQ-1/RQ-2.

This chapter marks the conclusion of first phase of the research project, i.e. observing the existing solutions. The chapter started with a summary of the criteria against which the test process evaluation approaches were to be analyzed. The choice of the approaches to be considered had already been reduced to Testing Maturity Model (TMM), Test Process Improvement Model (TPI), and Test Maturity Model Integration (TMMi). The chapter deeply analyzed all three of these approaches highlighting or reproducing the elements which supported or caused those criteria to fail. Summary of the findings is presented

in table 5.4. The — sign in the table marks that criteria could not be evaluated due to insufficient information, the other signs are self explanatory.

As the table shows none of the reviewed evaluation and improvement models of the testing process conform fully to all the set forth criteria of merit. Among them, only the Testing Maturity Model (TMM) appears to be the closest match with those criteria. The analysis has shows that TMM is still an excellent source for defining an improvement path for the testing process. Nonetheless, the identified lacks in this model create the possibility of a complementary evaluation model, which once appropriately developed, can fill the identified gaps in the current test process evaluation approaches.

# 6 Light-TPEF: The Test Process Evaluation Framework

The main contribution of this research work is contained in this chapter. It proposes the concept of an evaluation framework to address the research problem at hand. The framework serves as a design document for the construction of a concrete evaluation model and its implementation. All the elements of the framework are described that have to be utilized to evaluate software testing processes and later to develop an executable evaluation process model.

## 6.1 Introduction

The previous chapter has deeply analyzed mainstream evaluation and/or assessment models of software testing processes. Based on the criteria set forth at the beginning of this thesis, none of these models could exhibit full conformance to all of these requirements. The analyzed approaches, broadly following the maturity model concept, cover most of these criteria with few exceptions. Specially, despite stressing the use of measurements for controlling and monitoring progress of the testing process, the models exploit limited role of measurements for evaluating the process. Furthermore, the nature/style of evaluations has been implicit in contrast to explicitness as marked by research problem. These two aspects have come up as chief shortcomings of the existing models. The current chapter proposes a solution to these shortcomings in the form of an evaluation framework for test processes.

***Framework, what it means exactly?***

Before moving on further, the term *framework* deserves little clarification here which has often been used as a buzzword, especially in the software engineering field. The term *conceptual framework* as mentioned in Wikipedia [1] *is used in research to outline possible courses of action or to present a preferred approach to an idea or thought.* Although that the solution presented here is deemed to be this kind of conceptual framework, it is more than just an abstract framework in its essence. It is rather a concrete approach supported with full guidelines to implement it in a practical environment. The next sections incrementally build this approach and describe its details with appropriate techniques.

[1] `http://en.wikipedia.org/wiki/Conceptual_framework`

## 6.2 Concept and Design

***Framework concept***

To be able to address the requirements of an improved evaluation approach, a framework is first proposed which takes inputs from diverse sources of knowledge and practice. The approach followed in the framework enables satisfaction of generally considered key requirements of test measurement. Figure 6.1 summarizes principal inputs to the framework and the well-known test measurement related requirements that are believed to be implicitly met by the use of this approach. The four inputs are;

- **Software testing knowledge areas**: This element provides information about possible entities of evaluation, existing issues and requirements of test evaluations, classes of test measures commonly applied in practice which can answer different evaluation perspectives and suitability of an approach applicable to the testing area.
- **Evaluation theory**: To provide scientific rigor to the development of evaluation framework, this element provides philosophy and rationale behind all the components required to build a comprehensive solution. Generic evaluation principles are extracted from this non-software area of knowledge to identify and fill possible gaps in the existing evaluation approaches.
- **ISO/IEC 14598-5 Standard**: This standard is titled *Information technology–Software product evaluation–Part 5: Process for evaluators*. Part 5 of this standard takes evaluator's view of the evaluation process and describes activities needed to perform an independent product evaluation in connection with quality model as defined in ISO/IEC 9126 standard. Although that this standard is mainly designed for software product evaluations, it has recently been used by Trudel [Trudel et al., 2006] and Schmietendorf [Schmietendorf, 2008] to evaluate software processes and business processes respectively. Application of this standard to process evaluations and its close resemblance with the evaluation theory components motivates the author to customize the guidelines of this standard for test evaluations.
- **IEEE 1061 Standard**: Again fulfilling the requirement of following pertinent standard approaches, this standard will be used to extract knowledge as to how the measurements will be used to derive various quality characteristics of the measured testing process.

Although that the framework originally strikes the primary research problem of the thesis at hand, yet the developed approach can possibly enable an organization to satisfy measurement related maturity goals contained in maturity models. For example, implementation of test process measurement based on this framework tends to partially meet with the requirements of following maturity goals;

- **TMM Level 3 Maturity Goal 3.4- Control and Monitor the Testing Process**: This maturity goal calls for establishing a system to control and monitor the progress of testing processes. The subgoals outline a path to achieve this requirement by setting

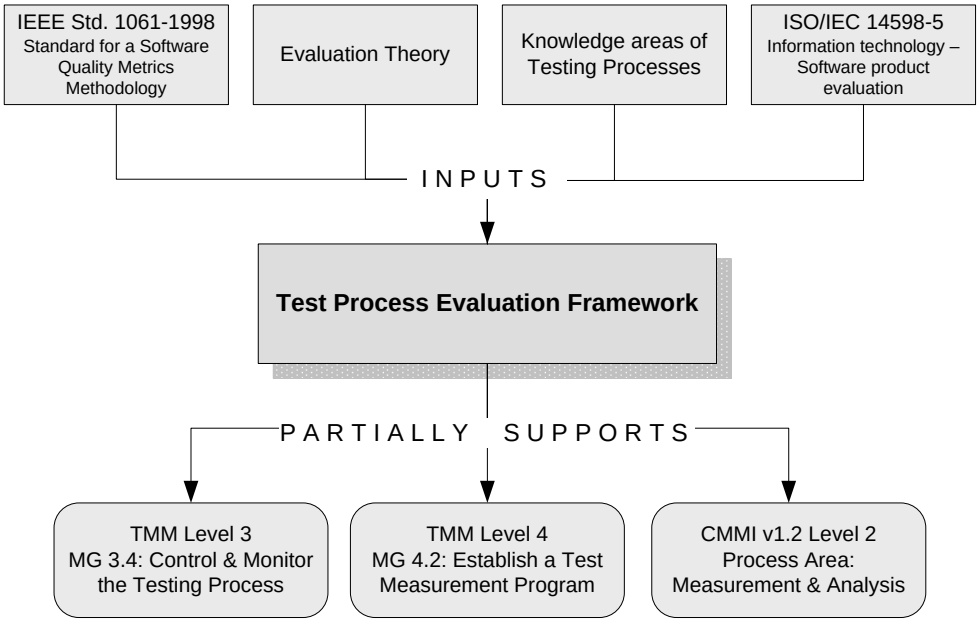


**Figure 6.1:** *Concept of Evaluation Framework*

up a responsible organization-wide committee, collecting and analyzing test measurements, and utilizing tools and other resources to support these activities. Specifically with reference to utilizing test-related measurements for project monitoring the TMM only tells 'what' has to be done while the question of 'how' it has to be done is answered by the proposed framework.

- **TMM Level 4 Maturity Goal 4.2- Establish a Test Measurement Program**: This maturity goal is quite similar to maturity goal 3.4 discussed before except that it necessitates a more consistent measurement program that uses organization-wide test measures to aid decision making process. A method following the proposed framework will be capable of meeting this requirement of test measurement program.

- **CMMI v1.2 Process Area: Measurement & Analysis**: This process area specifies a measurement process with much more specific details than TMM maturity goals and subgoals discussed above. An adapted version of mentioned measurement process makes one part of the of framework, which additionally provides some other information to support these activities. This process area can also be considered to be implicitly supported by the test evaluation framework.

Following the intentions of framework, the design and development of the test evaluation approach to be explained in the next sections is briefly summarized here in following steps;

1. A textual description of the five primary components of evaluation framework which serve as information support to the one core component

2. Graphical and semi-formal (textual) representation of the core component (evaluation process itself) to define all the workable and necessary steps

3. As a proof of concept, transformation and implementation of a concrete executable evaluation process model based on the framework

### 6.2.1 Framework presentation

Since the framework exists as a source of practical information that is expected to be used by technical and managerial people, it must be described in such a way to allow easy understanding and communication of information among involved people. A variety of process modeling notations and languages exist that exactly serve these purposes. Some kinds of descriptive, graphical, and semi-formal (also textual) representations will be used to provide unique views to the framework elements and their interrelationships.

**Choice of graphical modeling notation**

Several graph-based process modeling languages [Zamli, 2004] have been developed which describe software processes at different levels of abstraction. Since software processes are a kind of businesses too, apart from these predominantly technical representations, software processes today need to be described in way understandable to the business people as well. Business process modeling approaches [Lu and Sadiq, 2007], on the other hand, are oriented to business environments where process activities are performed by different roles and business units. Business Process Modeling Notation (BPMN) [OMG, 2006], as an intermediate solution addressing both software technology levels and business levels, supports a vast range of abstraction levels. The official specification of this industrially standardized notation states that *"The primary goal of BPMN is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes."* Since the evaluation process for test processes can integrate with software processes and hence with business processes, BPMN will be used to model process activities.

**Choice of textual modeling notation**

Complementing the graphical presentation of the model structure, the process architecture is sometimes modeled using some certain specification elements. Supporting the various levels of details and abstractions, the processes can be modeled using either of IPO (input, process, output) paradigm, ETXM (entry, task, exit, measure), ETVX (entry, task, validation, exit), or EITVOX (entry, input, task, validation, output, exit) paradigm etc [Humphrey, 1989, Ch. 13]. Although that the ETVX or EITVOX are usually the choice of most high maturity organizations [Paulk and Chrissis, 2002], the IPO model yet being quite concise in nature enjoys many interdisciplinary applications. It has recently been used by Drabick [Drabick, 2003] to model software testing processes. For the sake of simplicity an adapted version of this model–SIPO (support, input, process, output) as shown in figure 6.2 will be used in this thesis to describe the evaluation process.

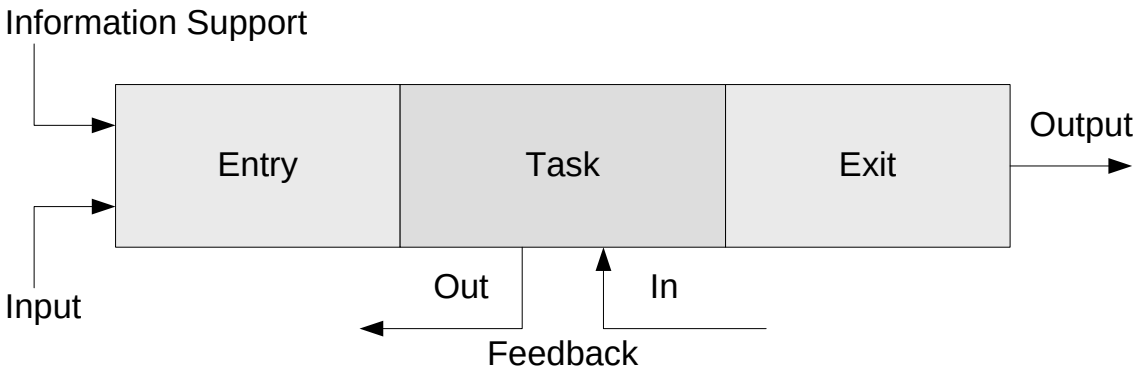


**Figure 6.2:** *Basic (adapted) Process Architecture Elements*

## 6.3 Framework Components

**Core component**

Stretching the conceptual view of the framework described earlier in section 6.2 and discussed in [Farooq et al., 2008d], [Farooq et al., 2008c], this section explains all the six individual components of this framework as shown in figure 6.3. There is one core component, Evaluation Process, that is derived from applicable international standards. This component furnishes the *'what'* aspect of the information needed to evaluate the testing processes. The scope of the information provided in this component is not much different from what is found in the ISO/IEC 15939 Software engineering – Software measurement process standard or the *Measurement and Analysis* process area of Capability Maturity Model v1.2. However, some content is simplified to reduce the overhead involving bureaucratic activities while on some other occasions further low-level activities are defined to specify the evaluation steps. BPMN diagrams and SIPO process architecture will be used to sketch this component.

**Support components**

The other five support components deal with the *'how'* aspect of the evaluation process and support the execution of activities defined in the core component. They are highly customized within the context of testing processes and their issues. The components are informative in nature and encircle all the details of testing knowledge needed to evaluate and assess the capabilities or in-capabilities of testing processes. A characterization of which testing entities could be evaluated, which criterion of merit are generally considered, which types of test measurements could be used, formulas of how the measurements can be synthesized, and visualization of evaluation results to be produced. The information from each one or more of these components is fed to each phase of the evaluation process thus enhancing those high level evaluation guidelines to produce a concrete working solution. Bare textual descriptions will be used to render these components. The next sections indulge themselves into further details of first the support components and then finally the core component.

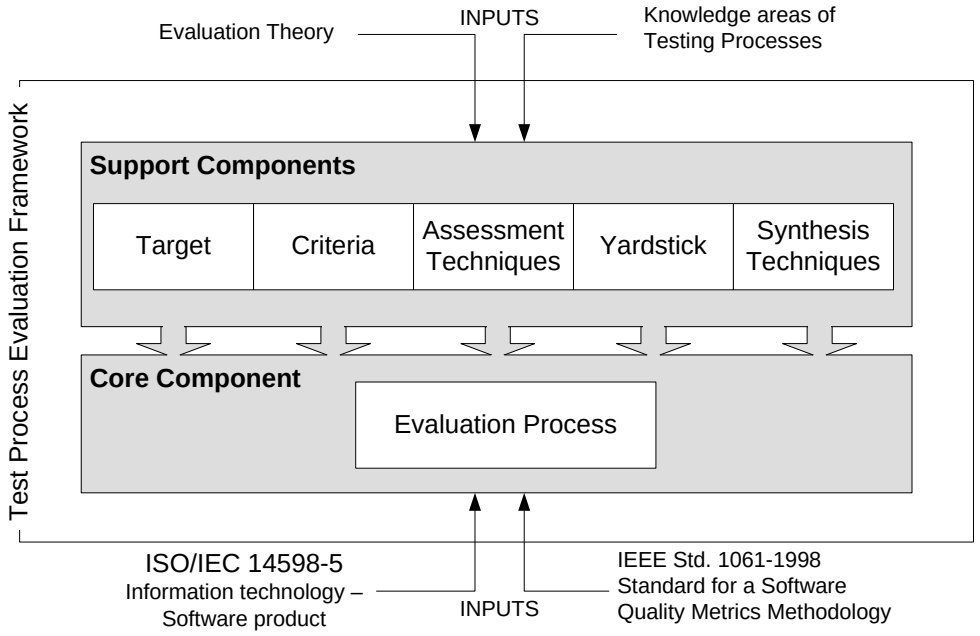


**Figure 6.3:** *Components of Evaluation Framework*

## 6.3.1 Support Components

Titles of support components are preceded with labels S1, S2, S3, S4, or S5. The labels will be used to refer to these components later on in the description of core component.

### 6.3.1.1 S1–Target

For the evaluations to be objective and specific, scope of the evaluations must be defined and the target must be delimited. The first question here arises, *what objects of test process can be and should be evaluated?*. A formalization of software processes provided by Dumke et al. [Dumke et al., 2005], [Ebert and Dumke, 2007] can be taken as starting source for this purpose where he classifies components of software products, processes and resources. They define a software product ($SP$) as a collection of programs and documentations. They write it in a formal manner as,

$$\begin{aligned} SP &= (M_{SP}, R_{SP}) \\ &= (\{programs, documentation\}, R_{SP}) \end{aligned}$$

They divide programs and documentation into further components as,

$$\begin{aligned} programs \subseteq \{&sourceCode, objectCode, template, macro, library, script \\ &plugIn, setup, demo\} \end{aligned}$$

$$documentation \subseteq \{userManual, referenceManual, developmentDocumentation\}$$

whereas $R_{SP}$ describes the set of relations over the $SP$ elements.

The software product is developed through implementation of a software development process. According to Dumke et al. [Dumke et al., 2005] the development process ($SD$) involves aspects such as different kinds of development methods, lifecycle phases through which the development iterates, lifecycle models adopted as a development approach, and management methods used at various levels. The define it as,

$$\begin{aligned} SD &= (M_{SD}, R_{SD}) \\ &= (\{developmentMethods, lifecycle, softwareManagement\} \cup M_{SR}, R_{SD}) \end{aligned}$$

They divide these three elements into further components such as,

$$\begin{aligned} developomentMethods &\subseteq \{formalMethods, informalMethods\} \\ &= SE - Methods \end{aligned}$$

$$lifecycle \subseteq \{lifecyclePhase, lifecycleModel\}$$

$$\begin{aligned} softwareManagement &= developmentManagement \subseteq \{projectManagement, \\ &\quad qualityManagement, configurationManagement\} \end{aligned}$$

Alongside the development process, several kinds of resources are exploited. Dumke et al.'s [Dumke et al., 2005] approach defines them as comprising personnel resources (developers, testers etc.), software resources in the form of CASE (computer-aided software engineering) tools, and platform resources representing the hardware and any other development environments. Their definition of software resources ($SR$) looks like,

$$\begin{aligned} SR &= (M_{SR}, R_{SR}) \\ &= (\{personnelResources, softwareResources, platformResources\}, R_{SR}) \end{aligned}$$

The resource types are further categorized as,

$$softwareResources = \{COTS\} \cup \{ICASE\}$$

$$\begin{aligned} personnelResources &= \{analyst, designer, developer, acquisitor, reviewer, \\ &= programmer, tester, administrator, qualityEngineer \\ &= systemProgrammer, chiefProgrammer, customer\} \end{aligned}$$

Equivalently, testing is as complex a job as is the whole development process itself. To identify the elements of evaluation, a structuring of the test process entities in a precise

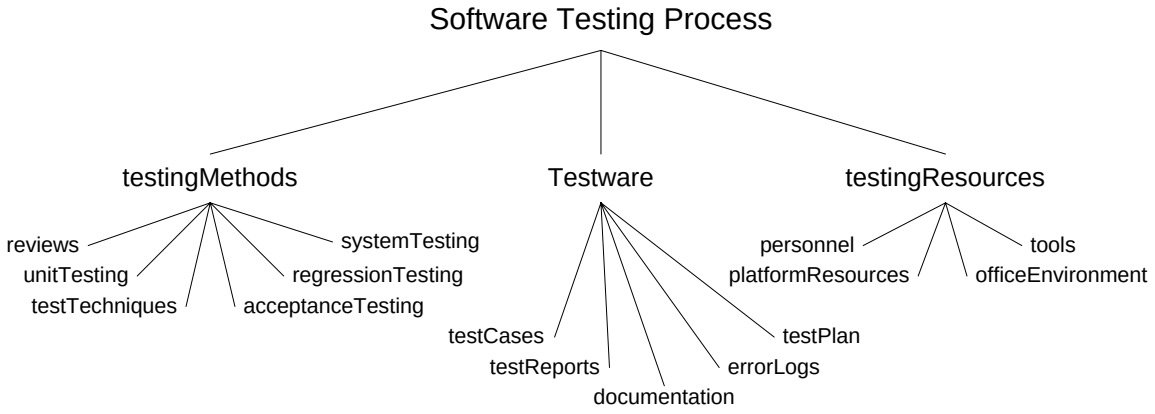


**Figure 6.4:** *Entities of Evaluation in Software Testing Process*

and comprehensible manner is needed. In doing so, without imposing any particular process model for test processes, a structured list of the testing entities of evaluation will be developed here.

First, several methods are applied all along the way of testing process. Seen from the perspective of these methods, testing process may be decomposed into various types and levels of organized activities which although cannot be detached from the testing process but a clear boundary can still be drawn between them. Examples are technical reviews, inspections, audits, unit testing, system testing etc or any particular testing techniques. One may be interested in finding effectiveness of inspections, efficiency of unit testing, or costs attributed to any particular testing technique that has been applied. This makes the first component of the testing process regarded here as *testing methods*. During the course of its execution, test process would usually produce some artifacts such as test specifications, test plans, error logs, defect reports, test cases, tested product etc. Testware refers to all such elements. Measurement and evaluation of this testware can give key insights to the size, duration, effort, and cost incurred during the testing process. Testware makes the second dimension of evaluation about testing process. Still a third class of elements of evaluation comprises resources involved with the testing process. These can be testing personnel, test tools, computers and hardware, as well as office environment. Evaluation of these aspects is necessary to assess cost effectiveness of the process. Figure 6.4 summarizes these three entities of evaluation for software testing processes.

#### 6.3.1.2 S2–Criteria

After having identified and defined artifacts involved in software testing process, we need to define their criterion of merit or worth that concern us. These must be established clearly in the light of general process goals, strategic business goals, and any particular technical requirements. For example, it is very common to say that testing be done in an efficient manner, with minimum time and resource consumptions. It is also not uncommon at all to require testing to be done effectively, to find and remove as many defects as possible. Several other criterion may relate to individual testing phases and techniques, testing tools and personnel etc. Such descriptions of requirements are generally outlined in the form of quality models for a given domain. A large number of techniques exist that attempt to eval-

uate some entity of testing process (see section 4.3). For example, some techniques focus on comparing efficiency of test case selection techniques, while others dedicate themselves to analyze effectiveness of reviews, inspections, unit and system testing. However, these examples represent only a subset of testing entities each being evaluated against a single criteria.

**Process quality attributes**

Similar to the case of software process, defining overall quality characteristics of testing process is a difficult task. In this connection, ISO 9126 software product quality standard (now replaced by ISO/IEC 25000 SQuaRE) is a well known quality model specifying product quality attributes and sub-attributes. Since the software processes are considered software too [Osterweil, 1987], the concept of product quality model could also be mapped to software testing process. Examples already exist where ISO 9126-like quality attributes and sub-attributes have been defined and their use demonstrated for software process by Kenett & Baker [Kenett and Baker, 1999], Tyrrell [Tyrrell, 2000], Satphathy [Satpathy et al., 2000], Güceğlioğlu and Demirörs [Güceğlioğlu and Demirörs, 2005a], [Güceğlioğlu and Demirörs, 2005b], and Kandt [Kandt, 2005]. For the sake of brevity, only the top level process quality attributes mentioned by these authors have been summarized in table 6.1. Recently, a similar concept has been illustrated to assess the quality of business processes by defining business process metrics, Rud et al. [Rud et al., 2007f], [Rud et al., 2007c], [Rud et al., 2007a], Demirörs and Güceğlioğlu [Demirörs and Güceğlioğlu, 2006], and Vanderfeesten [Vanderfeesten et al., 2007] being few examples. Wille [Wille et al., 2004] also used a similar hierarchical quality model for evaluation of agent-based systems.

**Table 6.1:** *Definitions of Software Process Quality Attributes*

| Reference | Process Quality Attributes |
|---|---|
| [Kenett and Baker, 1999, p. 101] | Size, Defect, Effort, Duration, Cost, Customer satisfaction |
| [Tyrrell, 2000] | Effectiveness, Maintainability, Predictability, Repeatability, Quality, Improvement, Tracking |
| [Satpathy et al., 2000] | Functionality, Usability, Efficiency & estimation, Visibility & control, Reliability, Safety, Scalability, Maintainability |
| [Güceğlioğlu and Demirörs, 2005a], [Güceğlioğlu and Demirörs, 2005b] | Maintainability, Reliability, Functionality, Usability |
| [Kandt, 2005] | Effectiveness, Efficiency, Predictability |

**Evaluation criteria for test processes**

These successful implementation of ISO 9126 standard like quality model concept for software process motivates one to develop a similar approach (as its special kind) for the software testing process. Consequently, ISO 9126-like hierarchy of typical evaluation attributes and sub-attributes for any kind software test process will be defined here. This

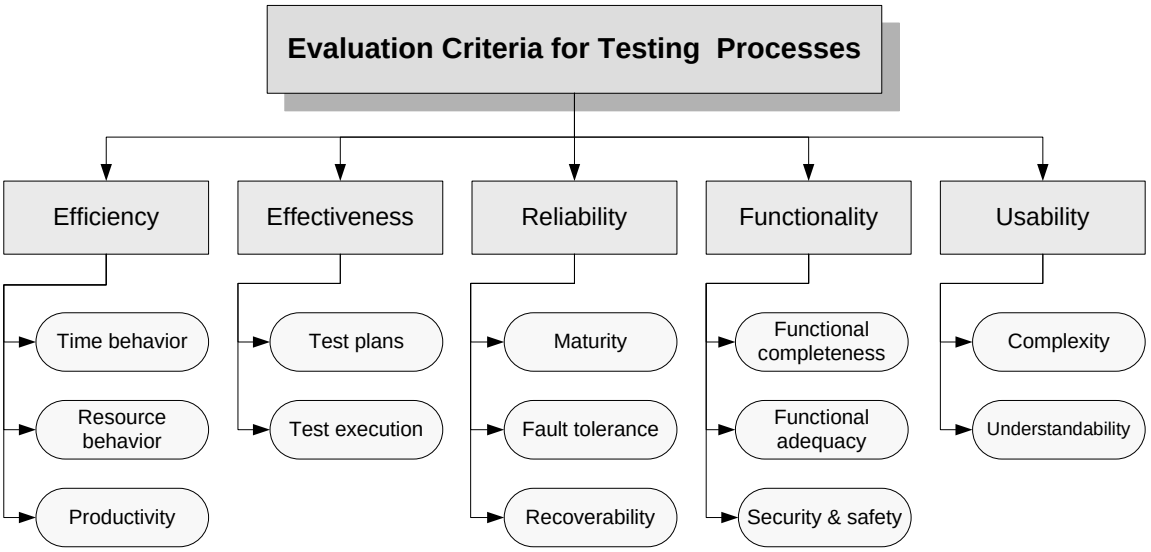


**Figure 6.5:** *Candidate Evaluation Criteria for Test Processes*

serves as a pre-defined generic evaluation model of the test process which needs to be customized according to particular evaluation situations. The finally refined model will make up the criteria against which a particular instance of a testing process has to be evaluated. The list of attributes/sub-attributes is explained next.

**Efficiency**

The definition of this attribute is derived directly from ISO 9126 standard where it applies to software product. Efficiency of the testing process is its capability to provide appropriate performance, relative to the amount of resources used, under stated conditions.

An efficient testing process is one which is as low-cost as possible. At the highest level of evaluation this cost undoubtedly refers to monetary costs. The top management usually counts this single cost indicator. However, there are other lower level cost drivers which contribute towards an overall monetary of cost of performing the testing activities. The time behavior of the testing activities and tools is one of them. Testing tasks also consume several kinds of resources during the course of their execution. The work performance or productivity of the testing staff also matters to complete intended testing procedures. Time behavior, resource behavior, and productivity make up the three sub-attributes of efficiency.

**Effectiveness**

It is the capability of the testing process to find and remove faults and defects. It relates to doing the testing *well* without reference to time or the resources consumed. This attribute is not only relevant to testing activities but is equally applicable to individual techniques, and tools for analyzing their required capabilities.

It is the implied goal of every testing process to render a software product as much defect free as possible. Errors in the code create faulty situations in the software programs which gives rise to failures. It is, therefore, a chief task of the testing process to detect and remove any defects present in the software. Effectiveness of testing process is meaningful at two key stages of the process, test planning and test execution. At the test planning phase it is mainly related to the necessity and sufficiency of the designed tests. While during test execution the efficiency concerns with defect detection capability of activities, techniques,

and tools.

**Reliability**

Test process may also be assessed indirectly by assessing its products. For testing process, tested program and code is the product. Since testing is a way to gain confidence in the quality of software products, we can rely on a testing process when it produces reliable (tested) products. Therefore, the concept of test process reliability is borrowed here from that of product reliability given in ISO 9126 standard and will be evaluated using product's fault tolerance, recoverability, and predictability. Maturity of the practices is one direct measure of test process reliability.

**Functionality**

This attribute will also be assessed indirectly from product quality evaluation. Software testing stands there to validate that the right product is being built and to verify that it is being built the right way (verification and validation). A software testing process that fulfills its functionality produces a (tested) product that satisfies functional and non-functional requirements. Therefore, this attribute can be assessed using product's functional completeness, adequacy, security and safety.

**Usability**

Before testing can begin, a testing process should be planned and defined. The activities of the testing process should be well designed to avoid complex couplings between them. The test process should be adequately documented to be understood well by its implementors. The usability of the testing process refers to its capability to be understood and implemented comfortably. This can be evaluated in terms of its understandability and complexity.

#### 6.3.1.3 S3–Assessment techniques

Having prepared the list of test artifacts and criterion of evaluation, the next task is to determine how good is that artifact with respect to its relevant criteria. A level (in ordinal scale) or a numeric score can be used to represent the extent to which a criteria holds true. Test process assessment models typically employ a varied set of techniques for this purpose. These may include measurements, questionnaires, interviews, document inspections, and simple observations etc. It is beyond any doubt that the use of measurement as a tool for process assessment is a powerful way to gather accurate status information about process progress. The use of test measurements is being advocated here as a primary means of test process data collection.

No definitive list of test metrics to be used will be specified here since the choice of a metric is limited by its availability or other constraints. However, several examples and classes of existing test metrics will be discussed here as guidelines. Several sources of test process metrics definitions and their application have been summarized in [Farooq et al., 2008a]. In developing a categorization of these test metrics based on their inherent characteristics or intended use, Dumke et al.'s [Dumke et al., 2006b] structuring of the software process metrics can provide a foundation where they discuss empirical aspects of software process by the use of different kinds of process measurements. Within the present context, the first aspect to be considered is the nature and type of measurement-based information available or applicable to test process. It can be distinguished as test related measurement experiences and test process metrics. For example, some testing prin-

ciples [Davis, 1995, Ch. 6], and testing laws [Kit and Finzi, 1995, Ch. 1], and various rules of thumb [Boehm and Basili, 2001] are a form of static quantitative experiences that guide the test professionals. On the other hand, test process metrics and related thresholds are a kind of dynamic quantitative information that highlight state and progress of the test process. Table 6.2 is a condensed reproduction of the test metrics definitions mentioned earlier 4.3 in this thesis. Although the list mentions most commonly used metrics, new metrics may also be defined as the need may arise.

**Table 6.2:** *Notable Resources of Test Metrics Definitions*

| Reference | Types of Metrics Discussed |
|---|---|
| [Peng and Wallace, 1994] | test cases, coverage, failure |
| [Liggesmeyer, 1995] | test complexity |
| [Kan et al., 2001] | test progress |
| [Pol et al., 2002] | miscellaneous |
| [Burnstein, 2003, p. 266] | testing status, tester productivity, testing costs, errors/failures/faults, test effectiveness, metrics at test process maturity levels |
| [Chen et al., 2004] | quality metrics, time-to-market metrics, cost-to-market metrics |
| [Abran et al., 2004, pp. 5-7] | program under test, tests performed |
| [Sneed, 2005] | test cases, costs, coverage, test effectiveness |
| [Sneed, 2007] | product, project, progress, process |
| [Afzal, 2007] | test progress, cost, quality |

Figure 6.6 gives the classification contexts for the test process metrics. Process evaluation goals is one of them. This goal aspect is equivalent to the evaluation criterion listed in section 6.3.1.2 above. Test metrics can be mapped to each of these criterion. Secondly, several entities pertaining to the test process itself can be measured as mentioned in section 6.3.1.1. Although not all of the process entities may be of concern in a particular situation, tracing the metrics back to the list of entities may help us identify if we are ignoring some process aspects in our measurements. Some broad classes of these measurable entities include [Florac and Carleton, 1999];

- Things received or used (resources etc)
- Activities and their elements (testing, inspections etc)
- Things consumed (effort, time, money etc)
- Things held or retained (tools, experience etc)
- Things produced (test cases, tested components, defect reports etc)

Test process phases/level and test process maturity levels give another dimension to structuring test process metrics. Metrics relating to test cost and effort or testability may be of concern at initial process phases while productivity and test efficiency kind of metrics

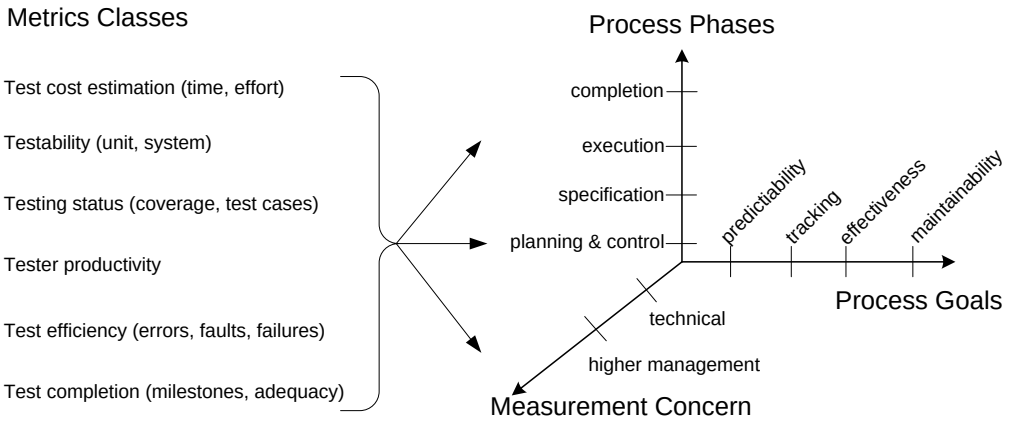


**Figure 6.6:** *Classifications of Test Process Metrics*

will be available towards later phases of the test process. Burnstein [Burnstein, 2003, p. 372] also lists different sets of metrics appropriate at different test process maturity levels. Figure 6.6 shows some example test process metrics classes which have been derived from available metrics definitions. These metrics classes relate to all three classification contexts. For example, metrics of test cost estimation are related to resources (process entity), calculated during planning & control step (process phase), and assess predictability and tracking (process goals). Undoubtedly, not all test managers may be concerned about all of these metrics. Most of the time they may just be interested in cost and effort related metrics.

#### 6.3.1.4 S4–Synthesis techniques

Collection of process measurements is as much an important task as is to convert this data into useful information about process quality. Organizations often complain that they collect lots of metrics but do not know how to use them. Synthesis techniques exactly serve this purpose. They organize and combine data gathered with the application of assessment techniques and transform them into qualitative judgements about the measured process. Systematic mathematical, statistical, or other methods are used for this purpose. Additionally, this section also includes explanation of any other techniques used to connect information provided by any of these support components.

**Measurement conversion & normalization**

This framework recommends collecting a wide variety of test process metrics and setting threshold values for each of them. All of these metrics may belong to different scale types. After the metrics values have been recorded and before they can be used for further measurement analysis, they need to be transformed to a common scale and normalized to a value between 0 and 1. Zuse [Zuse, 1998, Ch. 5] provides guidelines and rules to follow for such conversions. Normalization of measures is also a widely used method to get a consistent quality score for a measured attribute. Depending upon the type of problem and context, several different kinds of normalization methods exist [Zuse, 1998, pp. 231-238]. Among these, the normalization by a ratio scale transformation methods seems most

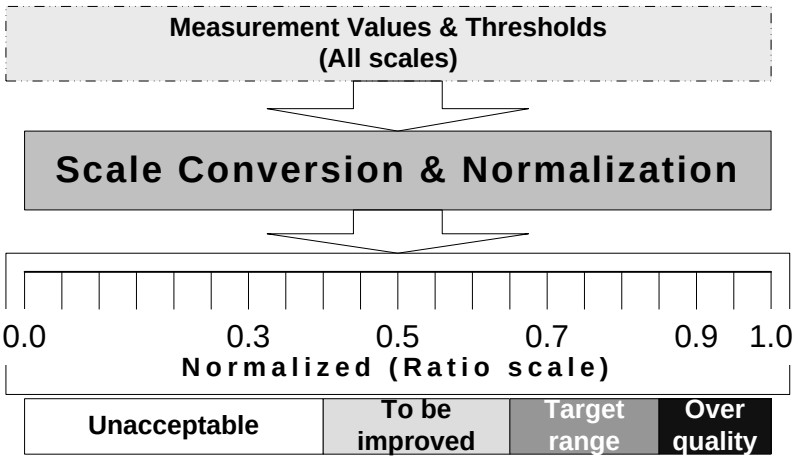


**Figure 6.7:** *Conversion and Normalization of Test Process Metrics*

appropriate since the empirical relational system of the measure is maintained in this way and the method is equally applicable to other scale types. Figure 6.7 shows a schematic diagram of test metrics transformation and normalization.

**Private quality indexes**

One or more test metrics may be associated with each of the sub-attributes for each of the test process entity. This is visualized in figure 6.8. These metrics will be used to calculate a numerical quality score for each of the sub-attributes called as private quality index. This concept of quality index as a hybrid measure is quite common in use in the area of software quality evaluations, such as the example of SAP using it for evaluating product, process, and project quality [Limböck, 2009], and Kozlov using it for software process quality [Kozlov, 2005].

It is assumed that $m_1, m_2, ..., m_i$ metrics are defined relevant to each sub-attribute/test process entity. Priority weights $w_1, w_2, ..., w_i$ (a number between 0 and 1) should also be assigned to each of these metrics corresponding to the significance of the metric. This concept of quality weights is influenced from Scriven's weight and sum methodology [Scriven, 1981] where a similar application was discussed. Private quality index (P.QI) is defined here to be

$$P.QI = \frac{\sum_{ind=1}^{i} w_{ind} \times m_{ind}}{\sum_{ind=1}^{i} w_{ind}}$$

The design of the formula results a value always between 0 and 1 for P.QI which will further be used for higher level quality indexes.

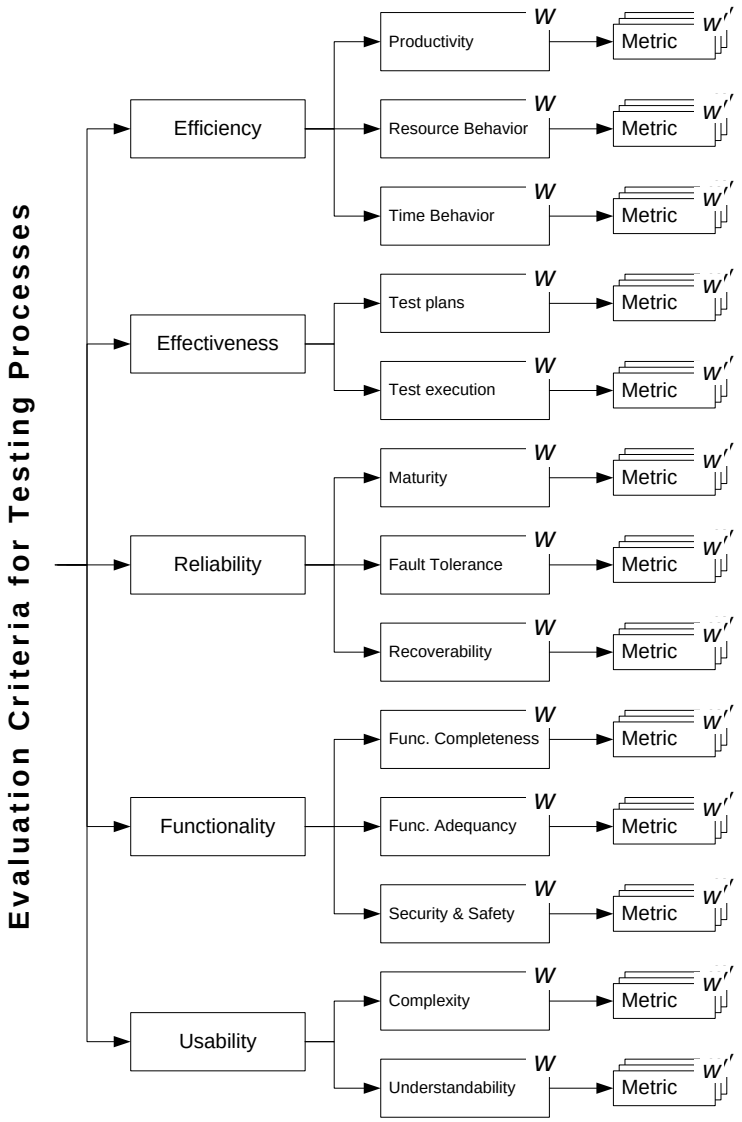


**Figure 6.8:** *Evaluation Criteria for Testing Processes with Metrics*

### Integral quality indexes

Measurement of evaluation attributes for each test process entity using sub-attributes will be carried out next. The above mentioned calculations will result in a series of private quality indexes $P.QI_1, P.QI_2, ..., P.QI_j$ for a given test process element against a given evaluation attribute. Priority weights should also be defined for each of the sub-attributes referring to their significance value. Then the formula for calculating an overall integral quality index (I.QI) for a given test process element for a given attribute can be written as:

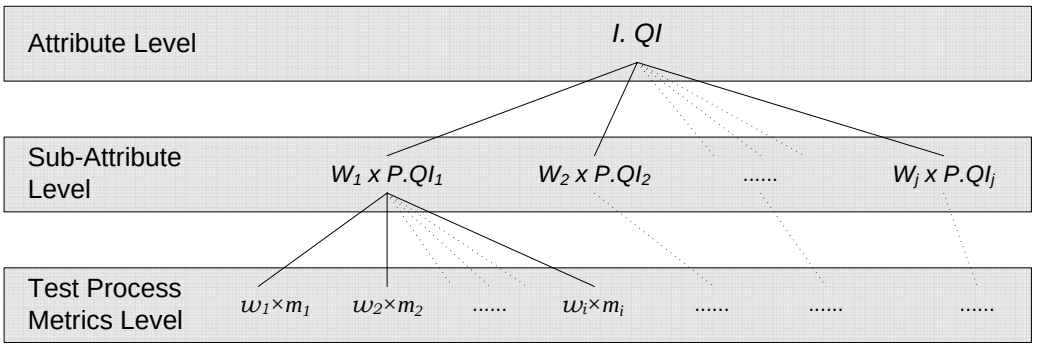


**Figure 6.9:** *Relationship between Quality Indexes*

$$I.QI = \frac{\sum_{ind=1}^{j} W_{ind} \times P.QI_{ind}}{\sum_{ind=1}^{j} W_{ind}}$$

The value for I.QI will always lie between 0 and 1. Calculation of integral quality indexes will give us attribute level view of test process quality. Figure 6.9 shows hierarchy and relationships among these quality indexes. The procedure for developing these different levels of quality scores from the test metrics could also be based on multi-criteria evaluation methodology [Blin and Tsoukiàs, 2001] or the analytic hierarchy process [Saaty, 2000] or its variant which is very commonly used in many different fields involving decision situations including quantification of overall software quality [McCaffrey, 2005].

#### 6.3.1.5 S5–Yardstick

This component represents the ideal picture of testing process against which a real instance of it is to be compared. It serves as a kind of reference standard or a yardstick which is used by every process assessment model. CMMI and TMM both use a maturity model for this purpose. This thesis introduces the concept of process quality profiles to be used for describing an idealized testing process. These profiles consist of pictures of the to-be-achieved testing process' quality at the attribute, sub-attribute or even at the metrics level. These quality-related requirements are represented in the form of numerical quality scores. This concept resembles Watt S. Humphrey's quality profile [Humphrey, ] where he uses it to suggest a process that should consistently produce high quality programs.

The test process quality profiles used in this approach consist of integral and private quality indexes which correspond to quality views at the attribute and sub-attribute levels respectively. These quality indexes for the ideal test process are calculated from threshold values of test metrics as against metrics values collected from an actual instance of testing process. The profiles are graphically shown with the help of Kiviat diagrams where each spoke represents a evaluation attribute/sub-attribute. The length of the spoke is taken as unity, and a quality index value which is between 0 and 1 is marked at an appropriate point

on this spoke. Joining these points gives a picture of the test process quality. Figure 6.10 and 6.11 show imaginary Kiviat diagrams at the attribute and sub-attribute level. Values of quality indexes for a measured test process are drawn accordingly and can easily and instantly be compared with the reference standard.

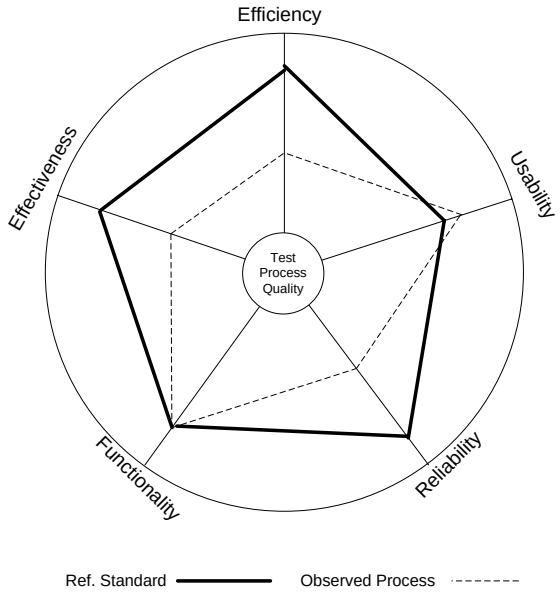


**Figure 6.10:** *Process Quality Profile: Attribute Level*

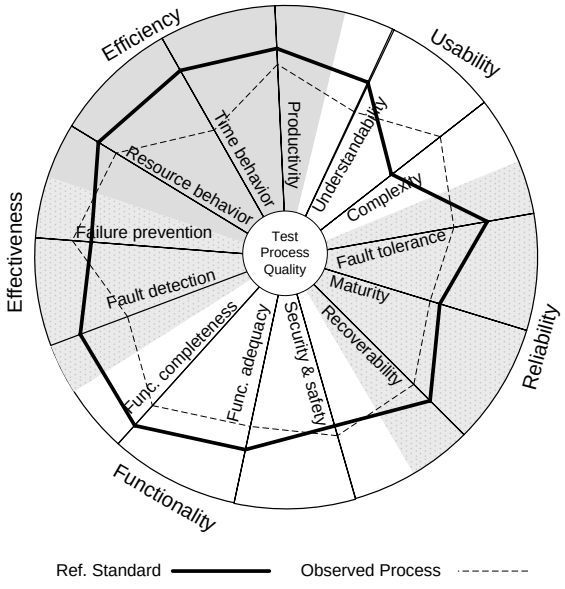


**Figure 6.11:** *Process Quality Profile: Sub-attribute Level*

**Evaluation matrix**

Not all of the evaluation attributes and/or sub-attributes proposed in section 6.3.1.2 may be associable to each element of the software test process mentioned in section 6.3.1.1. The relevance of each quality attribute to test process elements must be defined in some fashion. An evaluation matrix is created to establish connection between these two aspects. This matrix has to be derived based on requirement specifications, test process goals, and test specifications. Application of this kind of matrix as a method to specify and evaluate software quality requirements supporting managerial decision-making during the software life cycle has earlier been found in [Salvaneschi, 2005].

Furthermore, importance of different evaluation attributes may vary depending upon a particular scenario. For instance, for testing of a usual software application, efficiency and visibility & control of the test process may be more important as compared to other evaluation attributes while for testing of a safety critical application, functionality and reliability attributes may receive more concern. To express this significance weights can be assigned to the evaluation attributes. These weights can range from 0 (non-relevance) to 1 (applicable and fully important). The weights must be selected very carefully based on expert opinions or through a more sophisticated technique such as analytic hierarchy process (AHP) [Saaty, 2000]. Figure 6.12 shows this evaluation matrix may look like. A first view of this quality matrix contains only weights of sub-attributes corresponding to relevant process elements. Other views of this matrix are formed in the later stages of the evaluation process when quality indexes .

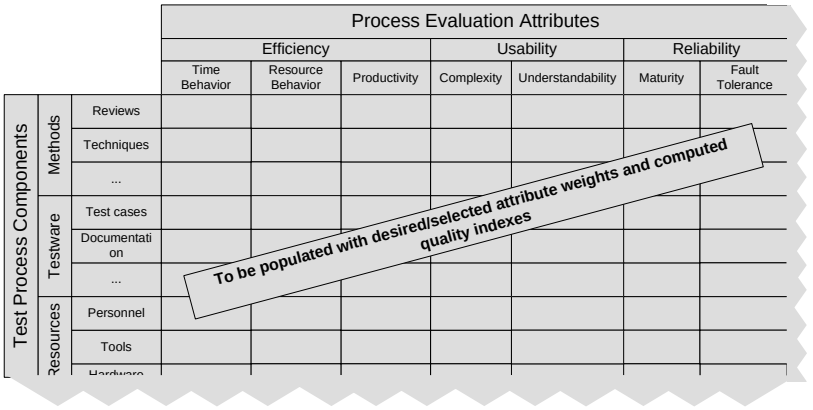


**Figure 6.12:** *A sketch of the Evaluation Matrix*

### 6.3.2 Core Component

As mentioned in section 6.2.1 above about the presentation format of the framework, BPMN has been opted for graphical notation to be used to explain phases of evaluation process. Recalling the figure 6.3 found earlier in this chapter about framework components, it is found that core component of this evaluation framework has been derived from ISO/IEC 14598-5 Information technology–Software product evaluation–Part 5: Process for evaluators and IEEE Std. 1061-1998 Standard for a Software Quality Metrics Methodology and is reinforced by support components $S1, S2, S3, S4, S5$ explained in sections 6.3.1.1 through 6.3.1.5. To maintain simplicity and conciseness, the evaluation process has been designed to comprise three sub-processes as seen in figure 6.13. It shows inputs and outputs to each sub-process in the form of data objects while the support components $S1, S2, ...S5$ etc are shown as text annotations attached through an association to respective sub-processes.

The first sub-process, *specify evaluation process requirements*, involves defining and prioritizing precise evaluation goals in the form of evaluation attributes/sub-attributes, finalizing the entities of interest for the evaluation, and developing relationships between the both. It takes candidate evaluation goals ($G$), candidate test entities ($E$), and testing process requirements ($R$) as input and produces selected evaluation goals ($G'$), selected test entities ($E'$), evaluation criteria ($EC$) consisting of attributes and sub-attributes of concern, and an initial evaluation matrix ($EvalMat$) defining relationships among test entities and evaluation attributes/sub-attributes. These outputs are fed into the second sub-process, *specify software measures*, which selects and defines test measures to answer evaluation goals, sets measurement thresholds, and establishes and prioritizes relationships among measures and evaluation attributes/sub-attributes. This sub-phase produces a set of selected measures ($M'$), measurement thresholds ($MT$), and a second version of evaluation matrix ($EM'$) which is now enhanced by association of measures to their respective attributes/sub-attributes of evaluation. The outputs from these first two sub-processes are used by the third and final sub-process, *perform process measurement & evaluation*, in which actual measurement values are collected and are processed to produce different views of quantitative judgements about test process. A final evaluation report ($ER$) is output that contains details of strong and weak areas of test process and improvement suggestions.

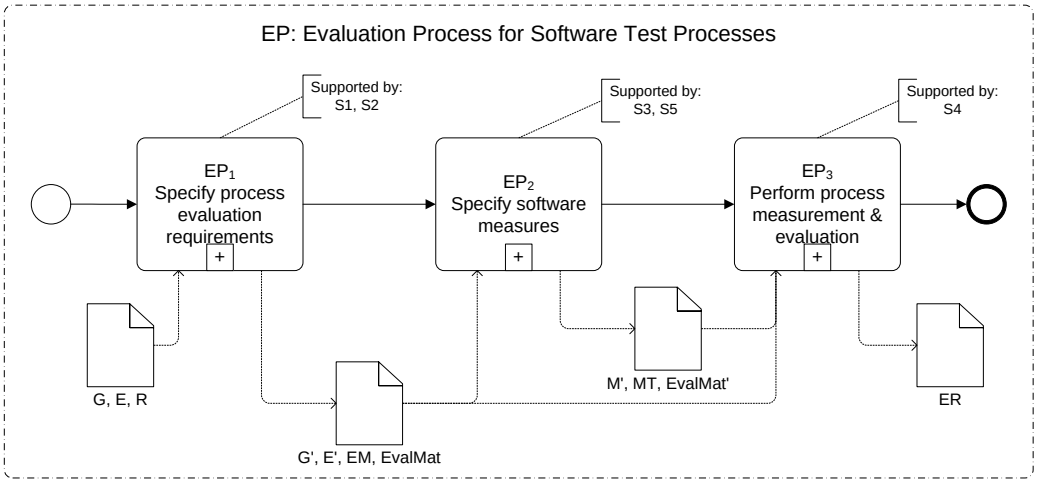


**Figure 6.13:** *BPMN Diagram of Top Level Evaluation Process*

**Table 6.3:** *Top Level Evaluation Process*

| **EP**: Evaluation Process for Software Test Processes | |
|---|---|
| Support items | $S1, S2, S3, S4, S5$ |
| Inputs | $G$ – goals |
| | $E$ – candidate test entities |
| | $R$ – test requirements |
| Process | - Specify process evaluation requirements |
| | - Specify software measures |
| | - Perform process measurement & evaluation |
| Outputs | $ER$ – evaluation requirements |
| | $ImprovementSuggestions$ |

The graphical representation of the evaluation process by means of BPMN diagrams is supplemented by a textual/semi-formal description using a customized SIPO process modeling methodology. As introduced earlier in section 6.2.1 and summarized in figure 6.2 above, the element 'S' has been appended to 'IPO' methodology to refer to support component $S1, S2, ...S5$ etc which guide the various phases of evaluation process. Each process phase modeled with BPMN diagrams in this text will also be exemplified by corresponding SIPO model. Sub-processes which contain only atomic phases (tasks) will not be modeled here using BPMN, they will be modeled using SIPO methodology alone. Table 6.3 is the SIPO model of the top level evaluation process shown by figure 6.13.

#### 6.3.2.1 Sub-process: Specify process evaluation requirements

This section explodes the sub-process, *specify process evaluation requirements*, into four sub-processes. Figure 6.14 and table 6.3.2.2 visualize these details. The first sub-process, *stipulate information needs*, produces a list of selected evaluation goals ($G'$) based on diverse sources of information such as test requirements ($R$), and organizational/business goals ($G$). The next sub-process, *delimit entities of evaluation*, shortlists entities of test

process ($E'$) to be evaluated from a list of possible test entities ($E$). The third sub-phase, *prescribe evaluation goals*, uses the list of selected evaluation goals ($G'$) and an initial version of evaluation criteria ($EC$) to develop a refined evaluation criteria ($EC'$) consisting of evaluation attributes and sub-attributes of concern for present project. The fourth and last sub-process in this sequence, *prioritize evaluation goals*, defines relationships among evaluation attributes/sub-attributes and testing entities and prioritizes these relationships. This phase uses the set of selected goals ($G'$), the selected test entities ($E'$), and refined evaluation criteria ($EC'$) to produce an evaluation matrix ($EvalMat$) which captures priorities of attribute-entity relationships.

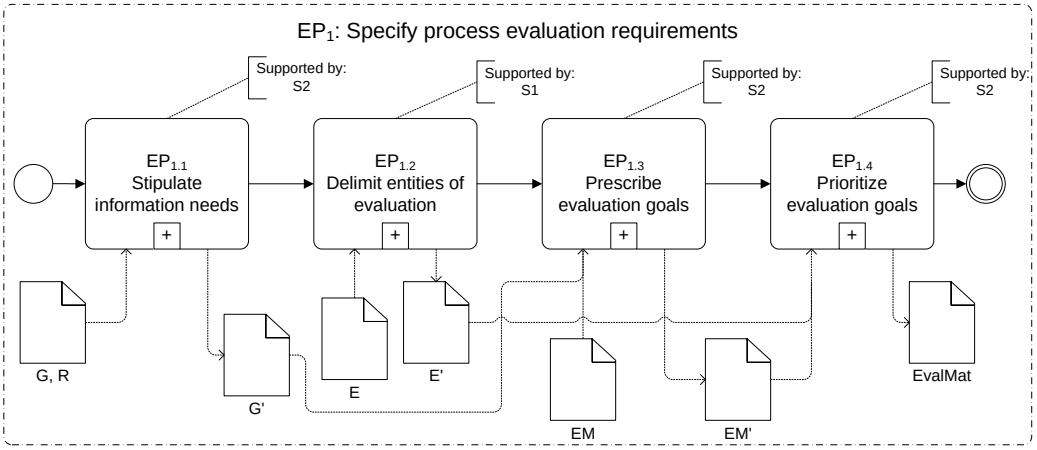


**Figure 6.14:** *BPMN Diagram of Specify process evaluation requirements Sub-process*

**Table 6.4:** *Sub-process: Specify process evaluation requirements*

| **EP**$_1$: Specify process evaluation requirements | |
|---|---|
| Support items | $S1$ – target |
| | $S2$ – criteria |
| Inputs | $G$ – goals |
| | $E$ – candidate test entities |
| | $R$ – any test process requirements |
| Process | - Stipulate information needs |
| | - Delimit entities of evaluation |
| | - Prescribe evaluation goals |
| | - Prioritize evaluation goals |
| Outputs | $G'$ – refined goals |
| | $E'$ – selected test entities |
| | $EC$ – candidate evaluation criteria |
| | $EvalMat$ – evaluation matrix |

**Stipulate information needs**

For the sake of brevity, BPMN diagrams will be sacrificed while describing the sub-processes which contain only atomic level activities (tasks), they will be modeled using only SIPO methodology. Table 6.3.2.1 models the inner lifecycle of *stipulate information needs* sub-process. The tasks begin with identification of all possible kinds of evaluation requirements. These requirements are refined based on strategic, business, and project goals resulting in a list of selected evaluation goals ($G'$).

| **EP**$_{1.1}$: Stipulate information needs | |
|---|---|
| Support items | $S2$ – criteria |
| Inputs | $G$ – goals |
| | $R$ – test process requirements |
| Process | - Identify a list of possible evaluation requirements |
| | - Identify business/internal evaluation requirements |
| | - Identify information needs to be addressed |
| | - Document results |
| Outputs | $G'$ – refined goals |

**Delimit entities of evaluation**

Table 6.3.2.1 summarizes the low-level tasks that make up this sub-process which is responsible for defining scope of evaluation. Using the information given in target ($S1$) component and taking the set of candidate test entities, the tasks shortlist the set of test entities of interest for present evaluation. A set of selected entities ($E'$) is the output from this sub-process.

| **EP**$_{1.2}$: Delimit entities of evaluation | |
|---|---|
| Support items | $S1$ – target |
| Inputs | $E$ – candidate test entities |
| Process | - Identify all possible entities of evaluation |
| | - Identify present entities of interest |
| | - Document results |
| Outputs | $E'$ – selected test entities |

**Prescribe evaluation goals**

This sub-process concentrates on defining the evaluation goals and attributes. With the help of criteria ($S2$) component, the preliminary evaluation criteria ($EC$) and input of selected evaluation goals ($G'$) from the previous sub-process, top-level evaluation attributes are first defined based on current project and business contexts. Relevant sub-attributes are then assigned to these attributes and defined accordingly producing a selected and defined evaluation criteria ($CM'$).

| **EP**$_{1.3}$: Prescribe evaluation goals | |
|---|---|
| Support items | $S2$ – criteria |
| Inputs | $G'$ – refined goals |
| | $EC$ – evaluation criteria |
| Process | - Identify all possible evaluation attributes |
| | - Identify evaluation attributes in accordance with business needs |
| | - Define evaluation attributes |
| | - Identify evaluation sub-attributes |
| | - Define evaluation sub-attributes |
| | - Document results |
| Outputs | $EC'$ –refined evaluation criteria |

**Prioritize evaluation goals**

The outputs from the last two sub-processes, the set of selected entities ($E'$) and selected evaluation criteria ($EC'$) still stand alone in their positions and need to be connected to each other reflecting the relevance and importance of each attribute for each test entities. In doing this, the tasks of this sub-process use the information provided by target ($S1$) and criteria ($S2$) components. Relative significance of attributes/sub-attributes are defined with the help of quality weights and are recorded in a so called high-level evaluation matrix ($EvalMat$).

| **EP**$_{1.4}$: Prioritize evaluation goals | |
|---|---|
| Support items | $S1$ – target |
| | $S2$ – criteria |
| Inputs | $E'$ – selected test entities |
| | $EC'$ – refined evaluation criteria |
| Process | - Connect evaluation attributes/sub-attributes with entities |
| | - Prioritize evaluation attributes/sub-attributes |
| | - Build high-level evaluation matrix |
| | - Document results |
| Outputs | $EvalMat$ – evaluation matrix |

#### 6.3.2.2 Sub-process: Specify software measures

The *specify software measures* sub-process is mainly concerned with establishing a system of test measures. Figure 6.15 and table 6.5 exemplify its structure. It is composed of three further sub-processes. The first sub-process, *select and define measures*, exploits the list of selected test entities ($E'$) and candidate test measures ($M$) to arrive at a list of selected test measures ($M'$). The support component, *assessment techniques* ($S1$), serves as guidelines for activities of this sub-process. The subsequent sub-process, *determine measurement thresholds*, takes the list of selected measures ($M'$) as input and assigns the

acceptable values ranges to these measures. This phase uses the information provided in the process *yardstick* ($S5$) support component and produces a list of measurement thresholds ($MT$). The third sub-process, *define relationships between measures & attributes*, is about assigning metrics to evaluation attributes/sub-attributes and prioritizing these relationships. These activities are governed by the information given in *assessment techniques* ($S3$) and *yardstick* ($S5$) support components. The inputs to this sub-process selected evaluation criteria ($EC'$) and selected measures ($M'$) are used to produce a next version of evaluation matrix ($EvalMat'$) which has measures attached to attributes/sub-attributes.

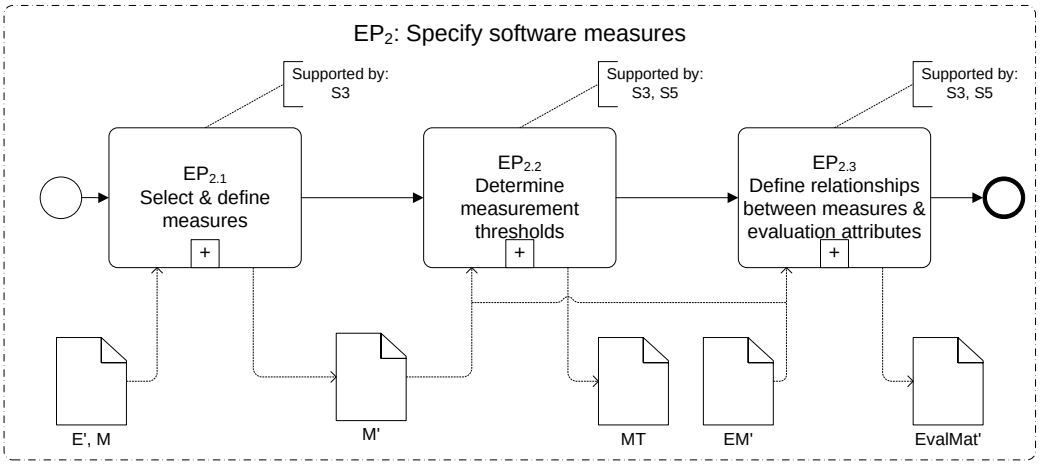


**Figure 6.15:** *BPMN Diagram of Specify software measures Sub-process*

**Table 6.5:** *Sub-process: Specify software measures*

| **EP**$_2$: Specify software measures | |
|---|---|
| Support items | $S3$ – assessment techniques |
| | $S5$ – yardstick |
| Inputs | $E'$ – selected test entities |
| | $EC'$ – refined evaluation criteria |
| | $EvalMat$ – evaluation matrix |
| Process | - Select and define measures |
| | - Determine measurement thresholds |
| | - Define relationships between measures & attributes |
| Outputs | $M'$ – selected measurements |
| | $MT$ – measurement thresholds |
| | $EvalMat'$ – evaluation matrix with measurements |

**Select and define measures**

It is the first elements of the above mentioned sub-process *specify software measures*. The tasks exploit the information from assessment techniques ($S3$) component and the inputs selected test entities ($E'$) and candidate test measures ($M$). Specific and applicable measures to be used are selected and defined. The tasks of this sub-process produce a set of selected test measures ($M'$).

| **EP**$_{2.1}$: Select and define measures | |
|---|---|
| Support items | $S3$ – assessment techniques |
| Inputs | $E'$ – selected test entities |
| | $M$ – candidate measures |
| Process | - Identify list of all candidate measures |
| | - Select and define specific measures |
| | - Document results |
| Outputs | $M'$ – selected measures |

**Determine measurement thresholds**

This sub-process is supported by assessment techniques ($S1$) and yardstick ($S5$) components and takes as input the list of selected test measurements to assign measurement thresholds to each of them. These desired value serve as measurement goals which help to develop a picture of ideal testing process sought by the organization. The tasks generate a set of measurement thresholds ($MT$).

| **EP**$_{2.2}$: Determine measurement thresholds | |
|---|---|
| Support items | $S3$ – assessment techniques |
| | $S5$ – yardstick |
| Inputs | $M'$ – selected measurements |
| Process | - Define measurement thresholds |
| | - Document results |
| Outputs | $MT$ – measurement thresholds |

**Determine measurement thresholds**

After the measures have been defined, they need to be linked with the attributes for which they provide some evaluative value. Information provided by criteria ($S2$) and assessment techniques ($S3$) components will be helpful in establishing this. The tasks will need the selected evaluation criteria ($EC'$) and selected measures ($M'$) elements. These measures could also be (optionally) prioritized reflecting their relative significance. These relationships will be recorded in the form of a low-level evaluation matrix ($EvalMat'$) which will be output to the next phase.

#### 6.3.2.3 Sub-process: Perform process measurement & evaluation

The third and last element of the top level evaluation process is the *perform process measurement & evaluation* sub-process. This phase involves collection of actual measurements, building qualitative views about the worth of the testing process, and reporting final evaluation results and suggesting improvements. Figure 6.16 and table 6.6 summarize configuration of this sub-process. It consists of another three sub-processes. The first of them, *collect measurement data*, is responsible for managing the collection of measurement values and related issues of refining and validating them. It does so with the help of guidelines

| **EP**$_{2.3}$: Define relationships between measures & attributes | |
|---|---|
| Support items | $S2$ – criteria |
| | $S3$ – assessment techniques |
| Inputs | $EC'$ – selected evaluation criteria |
| | $M'$ – selected measurements |
| Process | - Assign measures to sub-attributes |
| | - Prioritize measures |
| | - Build low-level evaluation matrix |
| | - Document results |
| Outputs | $EvalMat'$ – evaluation matrix with measurements |

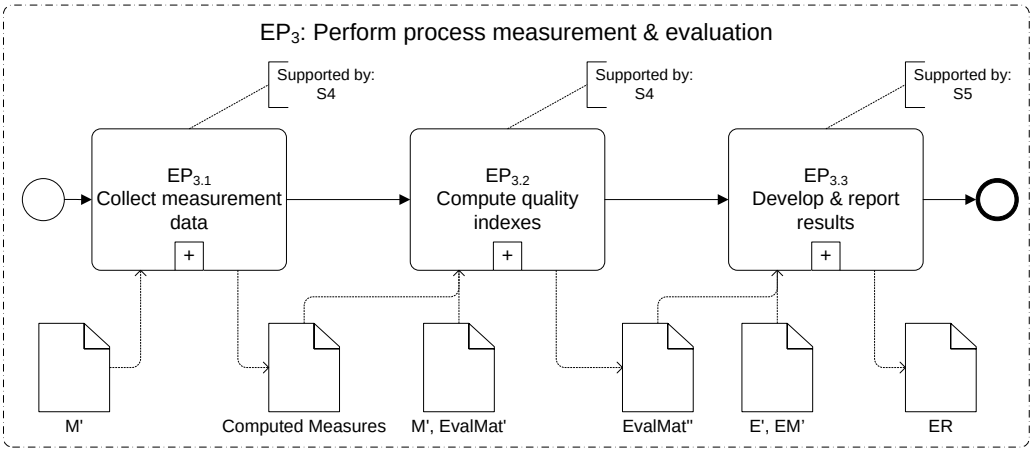


**Figure 6.16:** *BPMN Diagram of Perform process measurement & evaluation Sup-process*

from the *synthesis techniques* ($S4$) component, and with inputs of selected test entities ($E'$) and selected measures ($M'$). This phase results in a set of computed test measures. The next sub-process, *compute quality indexes*, is dedicated to transform computed test measures into qualitative judgements about the state of the testing process. These are in the form of numerical quality scores referred by quality indexes in this text. Using the information from synthesis techniques ($S4$) support component and inputs as selected measures ($M'$), their computed measurement values, and evaluation matrix ($EvalMat'$), this sup-process delivers a further refined version of evaluation matrix ($EvalMat''$) which contains values of quality indexes relevant to all attributes/sub-attributes and test entities. The third and last of these sub-processes is the *develop and report results* phase. It is about describing aforementioned qualitative information about test process at a higher level of abstraction highlighting strong and weak process areas. This phase is also responsible for developing appropriate improvement suggestions. The whole evaluation process culminates with completion of activities within this sub-process. The inputs to this sub-process are selected test entities ($E'$), refined evaluation criteria ($EC'$), and evaluation matrix ($EvalMat''$) while evaluation report ($ER$) is the final output.

**Table 6.6:** *Sub-process: Perform process measurement & evaluation*

| **EP**$_3$: Perform process measurement & evaluation | |
|---|---|
| Support items | $S4$ – synthesis techniques |
| Inputs | $E'$ – selected test entities |
| | $M'$ – selected measured |
| | $EC'$ – refined evaluation criteria |
| | $EvalMat'$ – evaluation matrix with measurements |
| Process | - Collect measurement data |
| | - Compute quality indexes |
| | - Develop and report results |
| Outputs | $ER$ |

| **EP**$_{3.1}$: Collect measurement data | |
|---|---|
| Support items | $S4$ – synthesis techniques |
| Inputs | $M'$ – selected measurements |
| Process | - Collect base measurements |
| | - Compute measurement values |
| | - Normalize measurement values |
| | - Document results |
| Outputs | Computed measurement values |

**Collect measurement data**

For performing the actual measurement and evaluation the first step is the collection and refinement of the crude measurement data. This typically involves capturing base measures out of which other hybrid measures are computed. These measurements have to be normalized to a common ratio scale to enable computation of next higher level of quality score out of these measurements. The guidelines given in synthesis techniques ($S4$) component will help guide this process. The tasks use the set of selected measurements ($M'$) to produce a set of computed measurements.

**Compute quality indexes**

The measurements collected and computed in the previous sub-process are converted into quantitative evaluations at the sub-attribute and attribute level which are here called quality indexes. Computations of these quality indexes need guidelines and formulas provided in synthesis techniques ($S4$) component. The tasks additionally use the set of selected measurements ($M'$) and the evaluation matrix ($EvalMat'$) to yield another version of evaluation matrix ($EvalMat''$) containing this time the computed quality indexes.

**Develop and report results**

This is the final phase of the whole evaluation process which converts the all the process measurements into interpretable quality judgements which are meaningful to test managers or other such people. The results are compared against the established goals and

| **EP**$_{3.2}$: Compute quality indexes | |
|---|---|
| Support items | $S4$ – synthesis techniques |
| Inputs | Computed measurement values |
| | $M'$ – selected measurements |
| | $EvalMat'$ – evaluation matrix with measurements |
| Process | - Compute private quality indexes |
| | - Compute integral quality indexes |
| | - Document results |
| Outputs | $EvalMat''$ – evaluation matrix with all quality indexes |

benchmarks. The yardstick ($S5$) support component provides ways through which such comparisons could be executed. This gives rise to identification of strong and weak areas of process and an evaluation report ($ER$) about needed improvements.

| **EP**$_{3.3}$: Develop and report results | |
|---|---|
| Support items | $S5$ – yardstick |
| Inputs | $E'$ – selected test entities |
| | $EC'$ – selected evaluation criteria |
| | $EvalMat''$ – evaluation matrix with all quality indexes |
| Process | - Draw desired-quality graph |
| | - Draw measured-quality graph |
| | - Identify strong and weak areas of process |
| | - Identify potential improvements |
| | - Document results |
| Outputs | $ER$ – evaluation report |

## 6.4 Summary

Moving towards the next phase of research project, i.e. proposal and development of an improved solution, this chapter addressed the fourth sub-question of current research task,

> RT-4. Bring forward the concept of a comprehensive approach that fills the gaps marked by research on RT-3 as well as which satisfies requirements set forth in RT-1/RT-2.

After a step-by-step analysis of the current test evaluation approaches for provision of established criteria, the previous chapter highlighted the areas where the available models exhibited weaknesses. This provided a rationale for an improved and complemental solution for the discussed problem domain. It was concluded that in addition to special consideration for measurement based explicit evaluations of test process, the solution must

**Table 6.7:** *Mapping of Method Criteria to Evaluation Framework*

| Criteria | Test Process Evaluation Framework | |
|---|---|---|
| | Status | Found in |
| **C1: Scientific rigor** | | |
| **C1.1: Target** | √ | Section 6.3.1.1 |
| **C1.2: Criteria** | √ | Section 6.3.1.2 |
| **C1.3: Yardstick** | √ | Section 6.3.1.3 |
| **C1.4: Assessment techniques** | √ | Section 6.3.1.4 |
| **C1.5: Synthesis techniques** | √ | Section 6.3.1.5 |
| **C1.6: Evaluation process** | √ | Section 6.3.2 |
| **C2: Utilization of Measurement** | √ | Section 6.3.1.4 |
| **C3: Utilization of Standards** | √ | Section 6.3.2 |
| **C4: Size** | √ | Overall model design |
| **C5: Style** | √ | Overall model approach & Section 6.3.1.4 |

be derived from science and theory of evaluations, be based on any relevant standards, and that it be low cost light-weight approach.

To care for these requirements, the proposed approach was built from an in-depth survey of the software testing field (rendered in chapter 2), perusal over the theory of systematic evaluations (accomplished in chapter 3), and guidelines derived from ISO standard on software measurement process and IEEE standard on applying software quality metrics. The design and content of the intended approach matched with the some requirements of TMM maturity level 3 (about using measurements for controlling and monitoring test process progress) and 4 (setting up a test measurement program).

The intended solution initially introduced as an abstract concept was filled with enough details to form a complete framework and to be able to

conceptual framework was upgraded to an evaluation model for test process with enough details to implement it in practical environments. The widely known graphical tool–Business Process Modeling Notation (BPMN) and an adapted version of a simple textual process modeling methodology–Input, Process, Output (IPO) were chosen as representation techniques for the evaluation framework.

The chapter introduced the framework as a collection of six components altogether, divided into two groups. The design of these six components was in strict correlation to the six core elements of evaluation required by the first criterion (C1). The first five of these components (target, criteria, yardstick, assessment techniques, synthesis techniques), named support components, were of informative nature and were explained with all relevant details derived from research on testing knowledge areas earlier in the thesis. These five components were connected to and supported the sixth component, evaluation process, which was called a core component. The process modeling techniques chosen earlier (BPMN and SIPO) were used to steps of the test evaluation process. While the structure

of the model was in line with the first criterion (C1), the content of the model components helped meet rest of the criteria (C2 through C5).

The chapter has presented a novel approach to evaluation of software testing processes in accordance with the set forth criteria and the marked shortcomings in the available solutions of its kinds. Table 6.7 maps the required criteria to the elements of the developed evaluation framework. The table clearly signs the achievement of the intended model goals. Consideration of the implementation scenarios and exercising the concept for a practical use is now left for the next chapter.

# 7 Implementation & Validation of Light-TPEF

The framework introduced in the previous chapter provides only principal structure of the solution to the research problem. The approach cannot really be appreciated without an operational example of its application. This chapter concerns with the practical implementation of the framework concept. Development of a working evaluation process model using appropriate technologies is explained. The chapter further discusses integration of the model in an actual business process and another example case study.

## 7.1 Introduction

Although that the presented model seems to provide structure of a test process evaluation approach to support the improvement of software testing processes, yet its correctness, adequacy and generality needs to be tested to gain confidence in the suitability of the solution. Furthermore, applicability and acceptance of a scientific research work depends on its ability to demonstrate its worth for practice. Without provision of any kind of proof of concept, the research at hand would merely be of the often criticized [Glass et al., 2004] '*formulate process/method/algorithm*' type missing any evaluations.

Software engineering researchers adopt a number of ways to validate developed methods or processes. The choice of a particular techniques is driven by the nature of the addressed problem, the nature of the developed solution, the research environment, time and other such elements. Experimentation or empirical validation is one broad class of such techniques. Zelkowitz and Wallace [Zelkowitz and Wallace, 1997] identified three categories of experimental validation techniques as *observational* (collect data as a project develops), *historical* (collect data from completed projects) and *controlled* (multiple instances of data collection with statistical analysis). Other classes of validation approaches mentioned by Shaw [Shaw, 2001] include *persuasion, implementation, evaluation, analysis*, and *experience*. However, the author of the thesis reorients these types of validation techniques by stressing that every research work first needs to be analyzed if it really solves the problem that it purports to solve, a kind of validation to be named as *internal/theoretical*. Next, to prove its value for practice, another *practical* validation can be performed for which any of the techniques classed by Shaw [Shaw, 2001] or Zelkowitz and Wallace [Zelkowitz and Wallace, 1997] mentioned earlier could be followed. The internal/theoretical validation is implicitly performed as the main and sub-questions of research are mentioned across previous chapters and that the Chapter 6 begins with explanation of connection between research problem/research questions and the to-be-proposed evaluation framework. In its summary in section 6.4 the chapter also provides a one-to-one mapping between required criteria and the elements of the developed model which ad-

dress them. It is a kind of assertion (from the class of observational approaches mentioned in [Zelkowitz and Wallace, 1997]) technique which is generally considered to have a value judgement. Therefore, the requirement of first kind of validation is assumed to be met already. The next sections discuss the development of a working solution that is exemplified to fit in current industrial environments.

## 7.2 Development of a Working Solution

***Goal***

The description of the evaluation process given in the last chapter has a semi-abstract nature. It is a kind of meta-level information in the sense that it just imposes a structure or framework on a possible concrete evaluation process. However, the modeling of the evaluation process with BPMN diagrams and SIPO methodology provides a partial picture of an approach from which an actual evaluation process could be instantiated. The goal of this section to discuss how existing modeling techniques and technologies can be used to convert it into a fully working evaluation process model suited to a process oriented software engineering environment.

***Process model***

A software process model is a representation of the software engineering process using a process modeling language. Software process models embody the sequence of activities, roles/responsibilities performing those activities, artifacts produced or consumed in activities, and any tools exploited during process tasks. Process models impose an overall structure over the process activities, while a process is an actual instantiation of it. Thus, process models can be reused by creating more than one instantiations of it. Software process models are developed to meet several objectives. They help us to facilitate understanding and communication among different stakeholders. This way processes can be manually or automatically monitored, measured, and evaluated thus supporting their efficient management.

***Process modeling languages***

Software process models can be designed using a diverse set of process modeling languages. Many different types of process modeling languages and notations exist [Zamli, 2004] that can be used based on the target of application. These include, among others, rule-based, graphical, executable, non-executable, state-based, and hybrid languages. However, more generally they can be classed as either executable, non-executable, or simulated languages. Non-executable or non-enactable languages are used when one is only interested in semantics of process activities with the goal of understanding process structure. Executable or enactable languages are used when the activities of software engineers are to be supported through a model which automates some of the process tasks. Such enactment is even more useful in a distributed environment where well organized synchronization among process activities is needed.

***Business process***

Today's dynamic markets and economy have posed great challenges for companies in providing better and specific products to their customers. The concept of business process management evolved around 1990s to enable companies to gain competitive advantages in

the markets. In this context, a *business process* is defined by Weske [Weske, 2007] to be *a set of activities that are performed in coordination in an organizational and technical environment which jointly realize a business goal*. A *business process model* is a description of these business activities modeled using an appropriate business modeling language or notation. Modeling of business processes offers similar benefits [Havey, 2005, Ch. 1] as do the software process models. There exist several modeling languages and notations for business processes too [Lu and Sadiq, 2007] some of which are for non-executable graphical representations only while others are with executable features. Among these, BPMN being a widely adopted graphical process modeling notation has already been used to provide details of the test evaluation process.

***XPDL***

Different business process modeling tools support a slightly varying sets of modeling notations. It is essential that a process designed by one tool is still usable by another tool which uses a different modeling elements. XML Process Definition Language (XPDL) [WfMC, 2008] is an XML-based file format standardized by the Workflow Management Coalition (WfMC) that can be used to interchange process models between various design tools. The XPDL and the BPMN specifications address the same modeling problem from different perspectives. The official XPDL specification [WfMC, 2008] explains relationships between each of BPMN and XPDL modeling elements. Several tools exists that can convert a business process represented in BPMN to an equivalent in XPDL format. The test evaluation process can also be transformed to XPDL version through the use of these tools. Although that the XPDL process is not designed to be executable, yet some execution engines are available that can directly enact a process stored in this format.

***WS-BPEL***

Web Services Business Process Execution Language Version 2.0 (WS-BPEL or shortly only BPEL) [OASIS, 2007] is an XML-based standard executable language that can model behavior of both abstract and concrete (executable) business processes. BPEL Abstract Processes are partially specified, serve communication and descriptive roles, and lack operational details. The BPEL Executable (concrete) Processes on the other hand contain full details of the process behavior and specifications. BPEL does not contain elements to represent the graphical aspects of a process behavior. However, a mapping between the BPEL elements and the corresponding BPMN graphical equivalent is provided in BPMN specification [OMG, 2006]. Many tools exist that offer transformation of processes modeled with BPMN into WS-BPEL.

***Approach***

In today's enterprise environments there is a need to reduce the gap between the organizational aspects of business problems and the technical aspects of information technology that is used to solve these problems. Software processes, test processes, as well as the evaluation processes of the kind developed by in this thesis being on the software side can be represented as a business process on the other end thus bridging the gap between information technology and business issues. To achieve this, the approach summarized in figure 7.1 can be followed. Several process modeling tools support designing business processes using BPMN notation. They provide to serialize these processes in XPDL process exchange language. Other transformation tools can convert a BPMN process either into BPEL or XPDL process. With the inclusion of semantic information about process execution, either of the XPDL or BPEL processes can be executed by a process execution

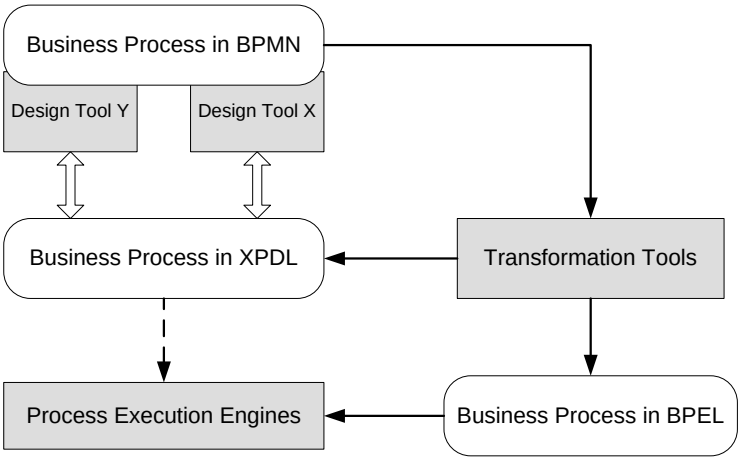


**Figure 7.1:** *Relationships among Business Process Technologies*

engine.

***Implementation***

Since the Light-TPEF is roughly an abstract structure of evaluation procedures, first a precise software process and/or a business process needs to be constructed out of it. Figure 7.2 outlines the approach followed in this thesis for construction of an executable business process based on the Light-TPEF framework. In addition to introducing the evaluation framework, the previous chapter, by the use of BPMN diagrams for describing Light-TPEF framework, has also partially developed this business process. The developed business process next needs to be transformed to an executable form using an appropriate process modeling language. A well known open source business process modeling tool called TIBCO [1] Business Studio™is being used here to first model the Light-TPEF and then to convert it to XPDL. A process execution engine will then be able to execute this evaluation process in an actual business setting. TIBCO is itself capable of executing the processes. Another option used in this approach is to send the XPDL version of the Light-TPEF to another simpler process execution engine. Enhydra Shark [2] is being referred here to enact the developed XPDL evaluation process. Shark is a Java-based open source workflow server which can directly execute processes written in XPDL format.

Figure 7.3 and 7.4 show screen shots of the Light-TPEF process model developed inside TIBCO Business Studio. Only the top level view of the evaluation process is visible in the first figure while the second shows a sub-process of the test evaluation process. This tool organizes related processes and sub-processes into a so called process package. All process packages are part of the root level project. The tool saves each process package in a separate XPDL file. Figure 7.5 shows the XPDL-based view of the designed test evaluation process. The actual enactment of the process is not provided here as it can only be done in a real problem setting which needs appropriate inputs from an implemented testing process.

[1] `http://www.tibco.com/devnet/business_studio/default.jsp`
[2] `http://www.enhydra.org/workflow/shark/index.html`

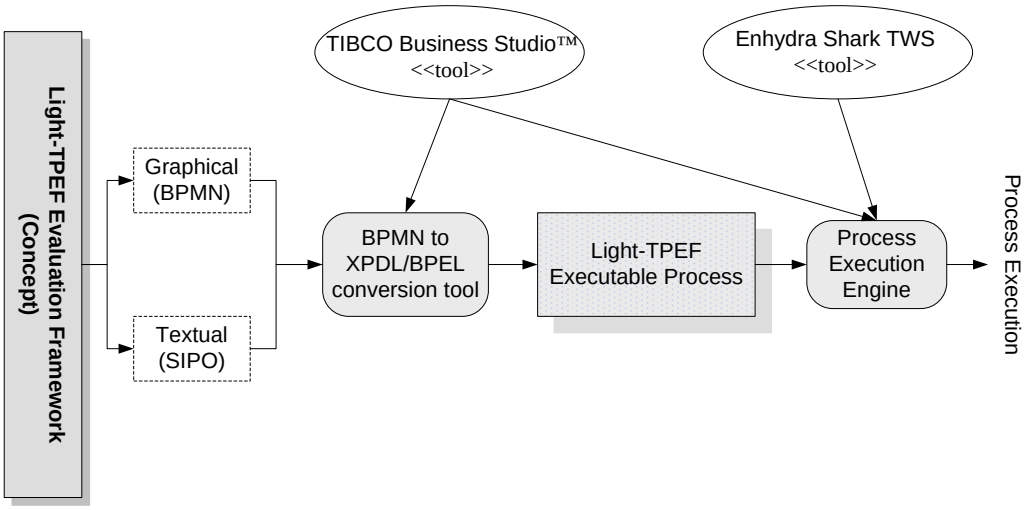


**Figure 7.2:** *Implementation Approach for the Evaluation Process*

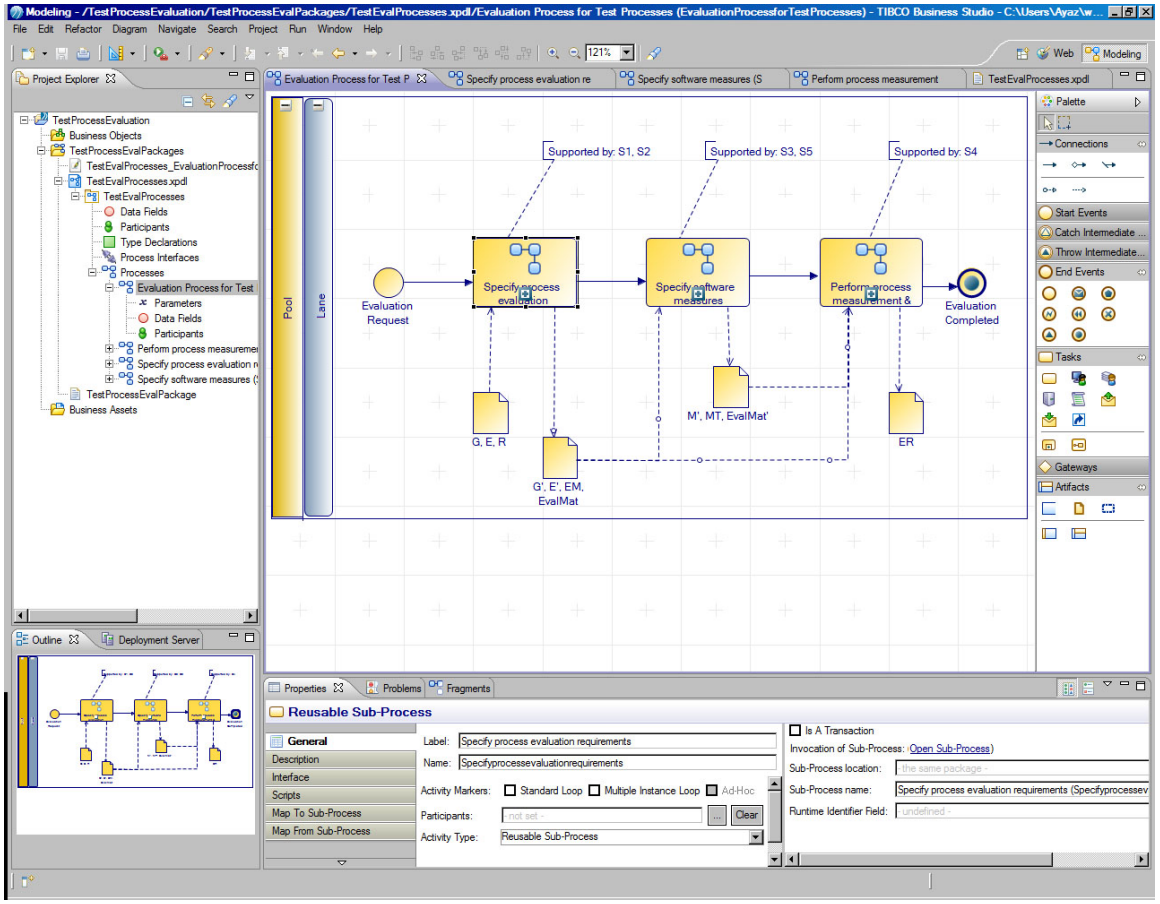


**Figure 7.3:** *Implementation of top-level Evaluation Process in TIBCO Business Modeler*

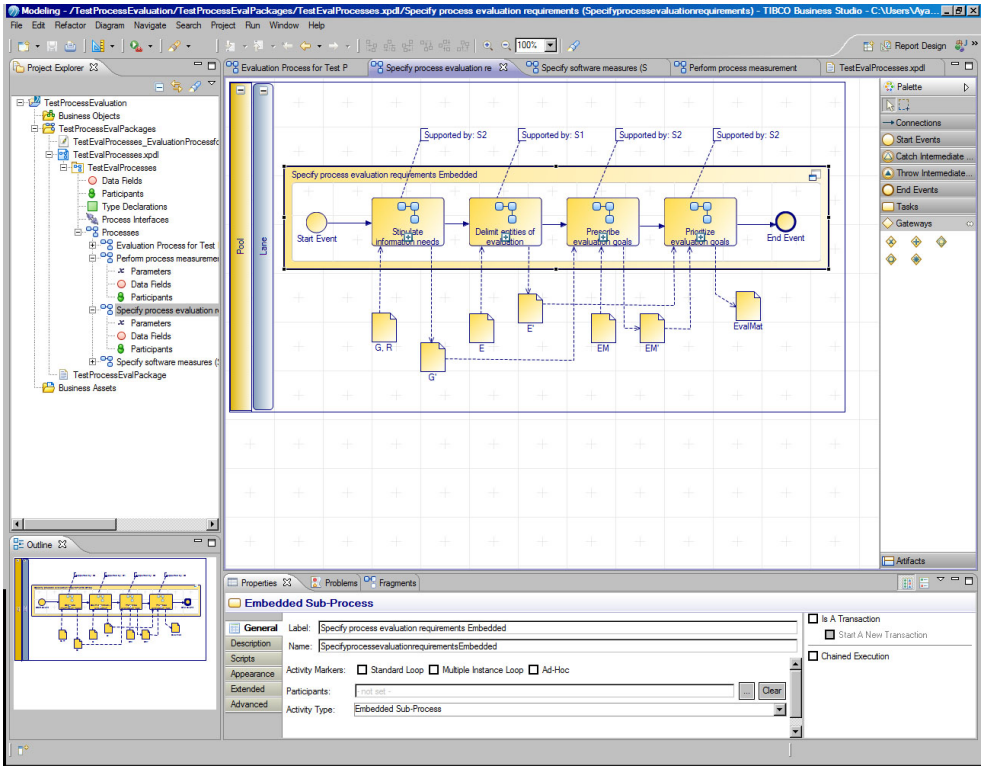


**Figure 7.4:** *Implementation of an Embedded Sub-process in TIBCO Business Modeler*

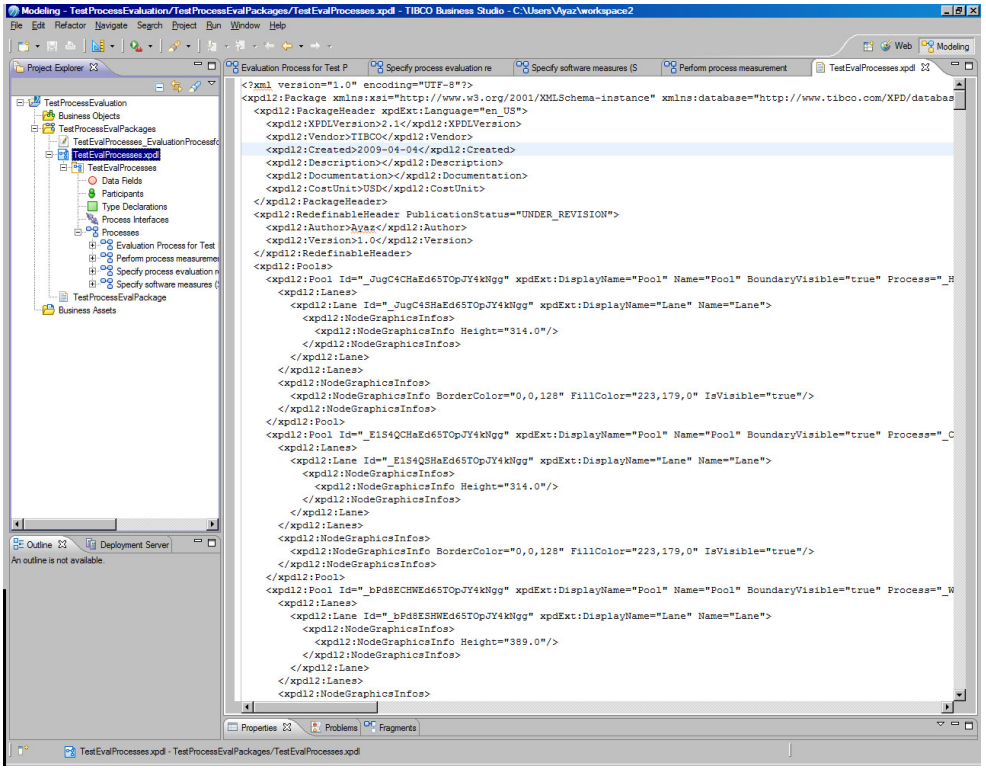


**Figure 7.5:** *XPDL view of Evaluation Process in TIBCO Business Modeler*

## 7.3 SOA Testing Background

This section introduces the application environment and related fundamentals of the chosen area in which the developed Light-TPEF approach will be exemplified. Being the latest trend in the development of modern business applications, systems based on service-oriented architecture (SOA) motivated the author to select it as an application scenario for the test evaluation framework.

### 7.3.1 SOA Revisited

***Service-oriented computing elements***

Service-oriented computing represents a new generation of the older systems built around the concepts of distributed computing and modular programming. It is completely new art and science of system architecture, design, and development involving unique design principles, design patterns, and new concepts, technologies, and frameworks. This new style of software system structure is aimed at improving efficiency, agility, and productivity and enables creation of enterprisewide and cross-enterprise flexible, dynamic business processes and agile applications. [Erl, 2007] defines and describes following primary elements of a service-oriented computing platform (also summarized in the accompanying figure 7.6),

- **Service-oriented architecture**:
  With the proliferation of literature about this new paradigm the term "service-oriented architecture" or the SOA have wrongly become synonym with the service-oriented computing itself. An SOA is an architectural model whose implementation can consist of a combination of technologies, products, APIs, and supporting infrastructure extensions etc.

- **Service-orientation**:
  It refers to a kind of style or design paradigm to provide functionality around the concept of independently existing services. With reference to other design paradigms, service-orientation broadly targets the separation of concerns.

- **Service-oriented solution logic**:
  It originates from the application of service-oriented design principles to built the solution logic

- **Services**:
  A service provides some capability in a distinct functional context. It is a primary building block of service-oriented systems and the resulting software programs as services can be discovered, composed, instantiated, and executed at runtime.

- **Service compositions**:
  Services may embody in themselves other services to provide some functionality. Service composition thus involves systematic coordination among services to address some related group of required functionalities.

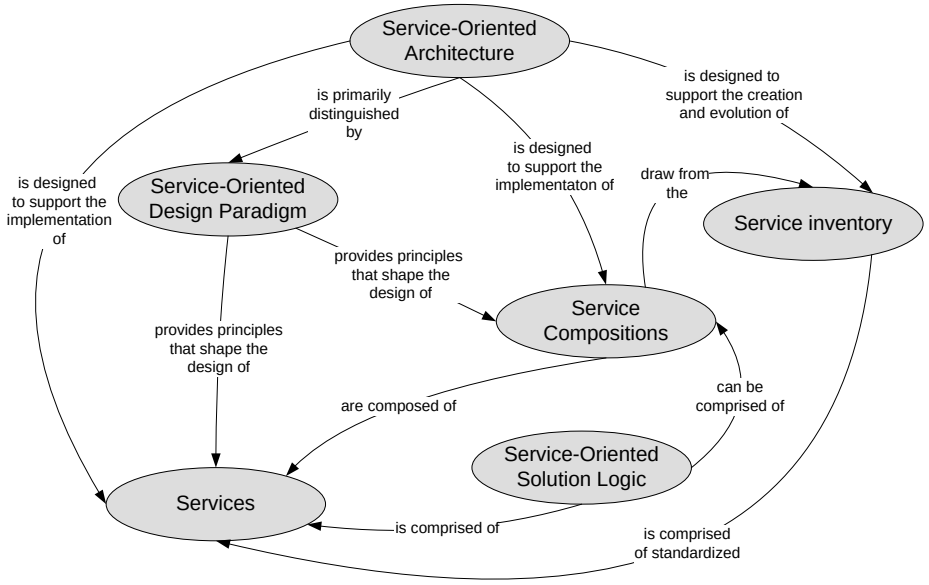


**Figure 7.6:** *Elements of Service-oriented Computing [Erl, 2007, p. 41]*

- **Service inventory**:
  A service inventory is an independently standardized and governed collection of complementary services within a boundary that represents an enterprise or a meaningful segment of an enterprise.

***SOA lifecycles***

A short description of SOA-based systems has already appeared in this text in section 2.2.2.2. The present discussion is aimed at drawing a picture of SOA systems so as to discover points where any kind of software testing could be involved and later to develop a picture of a SOA testing process. An analysis of SOA-related lifecycle phases can reveal these points. However there exist several perspectives to look at lifecycles in an SOA. One is to look at the service alone. A service is usually created, used, and enhanced but does not always live forever. It may cease to exist at some point in time when it no longer provides a business value. Thus, the service goes through few phases along its lifetime. However, the lifecycle of a service may be visualized from two other fundamental perspectives. From the point of view of provider, a service goes through vision, requirement analysis, design, development, testing, deployment, operation, maintenance, and phaseout cycles. From the view point of a consumer, it iterates through vision, discovery, binding, invocation, change, and monitoring. Another perspective to SOA lifecycle is held by IBM's experts [Jr. et al., 2005] and [Woolf, 2008] where they interpret it with reference to adoption of a complete SOA solution as well as a SOA governance lifecycle. For the SOA lifcycle, they divide it into model (capture business design), assemble (transform business to IT design), deploy (application hosting), and manage (maintain operational environment and policies) stages. Based on these lifecycle scenarios of a SOA-based system different levels and types of testing can be distinguished.

***SOA research issues***

The rigor and flexibility of SOA-based systems comes with a price and confronts us with

unique challenges [Stojanovic and Dahanayake, 2005][Papazoglou et al., 2007]. Some example research issues in this context are business related (SOA strategy selection etc.), engineering (process and lifecycle, development, quality assurance, testing, and maintenance etc.), operations (service monitoring and support etc.), and cross-cutting issues (governance and stakeholder management etc.) [Kontogiannis et al., 2008]. One issue among these engineering related research areas is quality assurance and testing. Software testing itself is a complex task and its scope and objectives vary with the applicable software engineering dimensions such as technology (object-oriented, component-based, services-based etc.), development methodology (waterfall, agile, etc.), and application systems (information systems, embedded systems etc.). The widespread adoption of SOA-based solutions introduces rising concern for efficient and effective testing methods but unfortunately testing SOA has not yet received adequate attention [Dustdar and Haslinger, 2004][Ribarov et al., 2007].

### 7.3.2 ITIL & SOA

ITIL stands for Information Technology Infrastructure Library. It is developed by the United Kingdom's Office of Government Commerce (OGC). ITIL is a widely accepted approach to information technology service management (ITSM). The current version 3 of the ITIL library consists of five components which have been briefly described below [Buchsein et al., 2008],

- **Service Strategy**:
  Covers strategy development, demand management, and service portfolio management for the prioritization of service provider investments in services

- **Service Design**:
  guidance on the design of IT services, processes, and other aspects of the service management effort. Beginning with the service portfolio management it provides guidelines on service level management and supplier management.

- **Service Transition**:
  Covers issues such as change management, service assessment configuration management, and release & deployment management.

- **Service Operation**:
  Provides guidelines on event management, incident management, and request fulfilment for the delivery of agreed levels of services both to end-users and the customers

- **Continual Service Improvement**:
  Contains set of best practices for service measurement, service reporting, and service improvement to aligning IT services to changing business needs.

These components tend to cover the gap between business and technology and focus on processes needed to deliver effective services to business customers. The version 3 reflects a key improvement with increased focus on service management taking the lifecycle approach to guiding IT services.

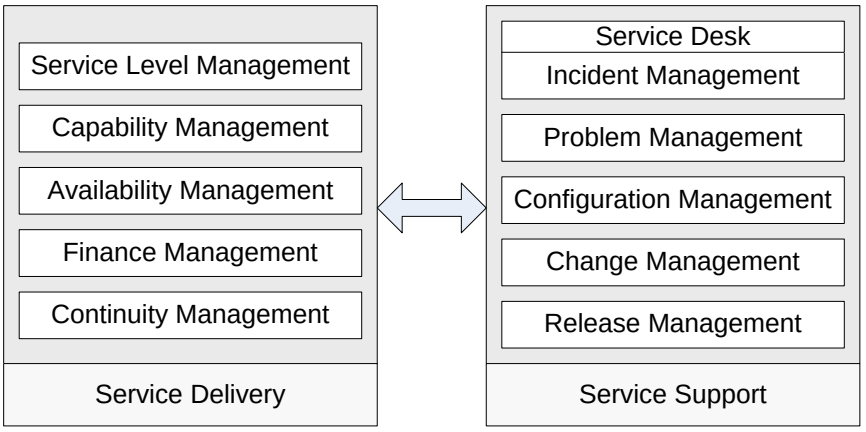


**Figure 7.7:** *ITIL Process Description for Service Management [Schmietendorf, 2007, p. 85]*

ITIL and SOA are deeply connected to each other. The focus of ITIL v3 towards services makes it more important for SOA. One key issue of SOA is governance while ITIL is also mainly about IT service governance. [Schmietendorf and Dimitrov, 2006] mentions an approach to manage service-oriented architectures with the application of ITIL. He outlines to use ITIL for service support, service delivery, and as ITIL-conformance management of SOA. Figure 7.7 summarizes the ITIL's role for service management.

### 7.3.3 Existing Research on SOA Testing

Survey of SOA testing issues and research have been found in a couple of articles such as [Parveen and Tilley, 2008], [Canfora and Penta, 2006],[Ribarov et al., 2007]. It has been observed that the task of SOA testing is significantly different from that of ordinary software applications and even of distributed systems. It is further complicated by a number of factors such as SOA testing perspectives (developer, provider, integrator, etc), the elements to be tested (governance, architecture, technology etc), the SOA peculiarities (absence of user interface and underlying code, etc), and testing levels and types (unit, service-level, integration, functional, non-functional) etc. Figure 7.8 visualizes these aspects involved within testing of SOA-based systems.

***Service testing process***

Instead of covering all aspects of SOA testing which seems a considerably challenging task, only a component of SOA, a service, will be considered here from testing viewpoint. Testing of services closely resembles that of component-based systems. It is rather even more complex due to peculiar characteristics of SOA systems. For example, lack of user interface, distribution of functionality, composition and integration of services, and high demands on performance and quality call for specialized testing techniques. Based on the analysis of SOA and service test methodologies discussed in [Canfora and Penta, 2006], [Ribarov et al., 2007], [THBS, 2007], following chief levels of service testing can be identified,

**Reviews**:
When requirements are being established and service is in design, traditional review

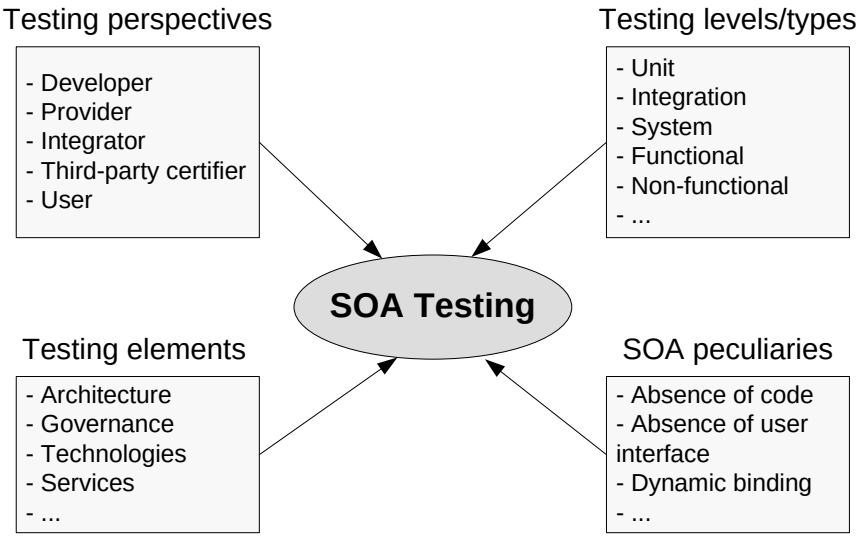


**Figure 7.8:** *Aspects of SOA Testing*

and inspection of documents and code is the first testing activity towards service testing. No specialized review techniques for SOA systems have been found in literature.

**Unit testing**:
Unit testing of a service performed by its developers is much like unit testing of ordinary software components which is aimed at verifying basic functionality of the service in isolation. Code reviews, white-box and black-box testing techniques can be applied at this stage. Despite the absence of any user interface, black-box tests can be generated from service specifications. It is the most common type of service testing which is widely supported by both commercial and open-source SOA testing tools.

**Integration testing**:
A service also needs to be tested if it works well in collaboration with any other intended services or components. Service integration testing establishes that service interface follows defined format and standards. This is perhaps the most important phase of service testing since SOA systems are meant to be an integration of several services. Evolveability of services, unavailability of all the constituting components or services, and dynamic and late binding nature of services sometimes complicates this phase of service testing. Examples of integration test strategies for SOA include [Huang et al., 2008] and [Bucchiarone et al., 2007].

**System testing**:
Systems testing is aimed at testing the whole service-oriented system as a unity. This testing level involves verifying the functional and non-functional characteristics of the system. Acceptance testing is yet another type of SOA system testing.

**Functional testing**:
It applies both to an individual component/service or the whole SOA system to

verify its conformance to business requirements or technical designs. In context of services, this type of testing can be performed by sending a request to a service and then analyzing the received response. Services can exhibit a number of faults [Brüning et al., 2007] all along their lifecycle phases. These may occur during publishing, discovery, composition, binding, and execution of services. Service specifications help to generate test cases for the tests against these and other types of functional misbehavior. Growing size of a service-oriented system needs more sophisticated services which makes design of adequate tests covering all functional characteristics a great challenge. Examples of functional testing of SOA systems include [Sinha and Paradkar, 2006] and [Dranidis et al., 2007].

**Non-functional testing**:
Quality of the service is a crucial factor for its existence. Non-functional testing is aimed at verifying QoS (quality-of-service) attributes [O'Brien et al., 2007] of services. It checks to see if a service meets the demands on its reliability, efficiency, usability, maintainability, security, and portability etc which are usually defined in service level agreements (SLAs). The dynamics of service bindings and external factors such as network or server load complicate this testing step. However, today there are several tools on the market which support this type of service testing. Approaches of non-functional testing of SOA systems include [Rud et al., 2007a] and [Fu et al., 2004].

**Regression testing**:
After any phase of service testing is completed regression testing of services is performed to ensure that any changes to service code, interface, or otherwise does not introduce any new defects. It should be particularly performed after unit, integration, functional and non-functional testing phases. Examples of regression test techniques for services include [Penta et al., 2007] and [Ruth and Tu, 2008].

**Acceptance testing**:
Finally, a passage through user acceptance test moves a service-oriented system to its implementation. Early involvement of business users and operational stakeholders is a key to a comfortable user acceptance test of services.

Like traditional software testing, these testing activities inevitably consume at least some portion of an service development project resources. Since optimization of development effort, time, and cost has always been prime concern of software companies, the choice eventually falls on analyzing testing processes for its efficiency and effectiveness. As has been observed and mentioned earlier in section 2.2.2.2 that alongside invention of numerous testing technique for SOA, empirical knowledge about their efficiency and effectiveness has yet to be established. While there exist many techniques for evaluation of SOA implementation [Rud et al., 2007e], products [Rud et al., 2006], and resources [Zenker et al., 2007] etc, dedicated techniques for the evaluation of service test techniques and processes has been an ignored issue as yet, with the exception of an initial model called TPI SOA.

***TPI SOA***

TPI SOA [Eggink et al., 2008] is a recent customization of the well known process improvement model, TPI [Koomen and Pol, 1999], in the SOA context. The TPI SOA model

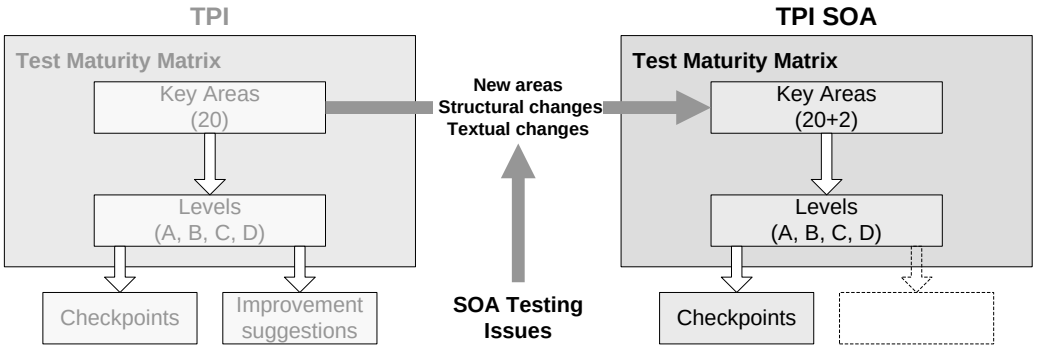


**Figure 7.9:** *From TPI to TPI SOA*

introduces five new process areas in addition to making few structural and textual changes to existing process areas. The newly added key areas are shortly described below while the figure summarizes the shift of TPI to TPI SOA.

- **Service Registry**:
  Exploitation of service registry for development and test process
- **SOA knowledge**:
  The existence and implementation of special testing knowledge in SOA context
- **Availability of test basis**:
  Existence of details about test objects to serve as test basis
- **Service integration**:
  The level of service testing if it has been modified
- **Quality management**:
  Extent of implementing quality management procedures for test process and SOA services

The new model is no doubt an initial endeavor to the complex world of SOA testing. The new model lacks key areas specific to SOA governance testing, SOA metrics, and testing the underlying technology and architecture etc. The model seems to cover only a few *service* testing issues. Although that checkpoints can also serve as improvement guidelines, dedicated improvement suggestions like TPI are not part of the currently available TPI SOA.

## 7.4 Light-TPEF: Application in an SOA Industrial Environment

This part of the thesis describes an industrial development situation as an example implementation scenario of Light-TPEF approach. The considered case is a service-oriented development environment at the software division of a company in eastern Germany. The identity of the company is withheld for the sake of anonymity and to avoid influences of any company-related policies on the applicability of current research work.

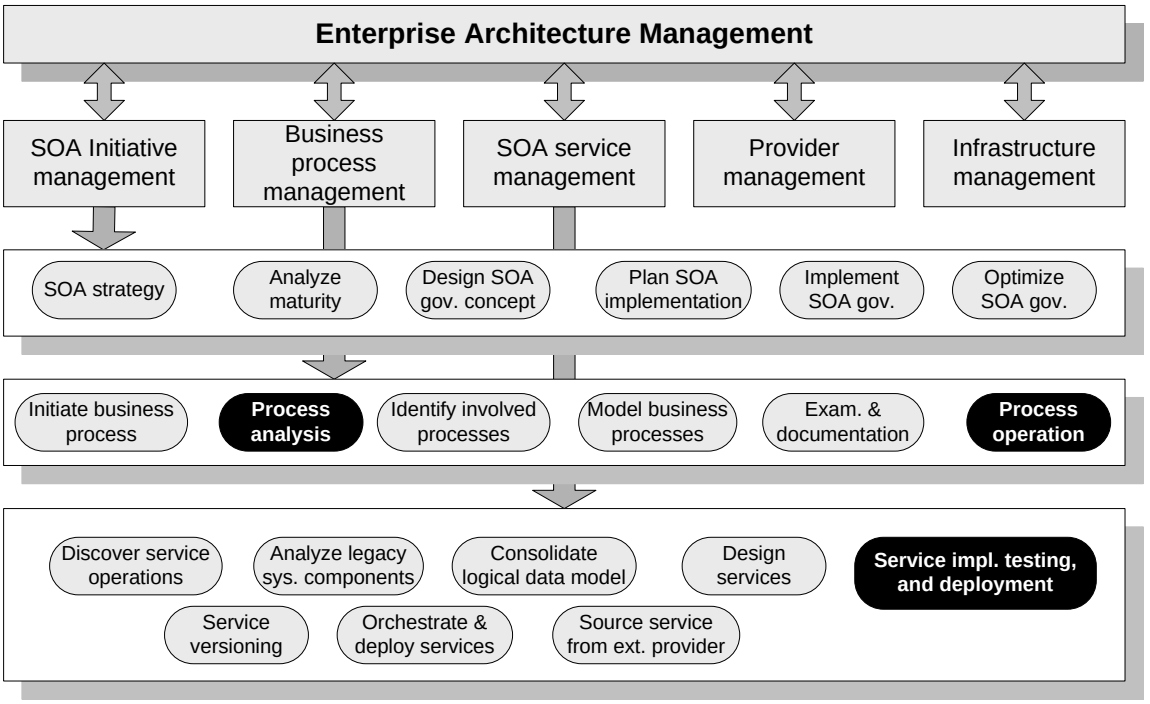


**Figure 7.10:** *SOA Governance Model*

## 7.4.1 Background of the Industrial Environment

***Example SOA governance model***

Figure 7.10 shows an outline of this company's approach to managing technical and business/organizational issues related to implementation of SOA as its development strategies. This is called a SOA governance model which is variably defined in literature and practice to comprise some necessary components. It is so because implementation of SOA governance in enterprizes is a customization of technical and business approaches to solve particular SOA related issues. The SOA governance model shown in figure 7.10 comprises five modules, SOA initiative management (introducing SOA governance in an organization), business process management (for planning, implementation, and operation of business processes), SOA service management (covering service derivation, implementation, orchestration, and deployment), provider management, and infrastructure management. Each of these modules consist of several building blocks which organize activities to be carried out. The building blocks are defined in a structured manner with the details about their task descriptions, inputs/outputs and involved roles. The figure 7.10 also highlights three building blocks of interest to the current example.

The first building block of interest in the current situation is *process analysis*. It is aimed at identifying and analyzing requirements for all new processes or adjusting pre-existing processes with any new requirements that grow as the business process evolves. The first activity inside this building block determines the boundary conditions for the process such as time, technical and business constraints. The functional and non-functional requirements for the process are identified next. These requirements are derived from requirement specification documents and service-level agreements (SLAs).

These requirements are then assessed to find out effort involved and then prioritizing them. The next phase captures and records all relevant data to these processes. Summary of the process analysis building block is given below,

Process analysis

Participating roles:

Business project team

Activities:

- Determination of boundary condition
- Determination of functional and non-functional requirements
- Estimation of requirements
- Process survey

The next building block of interest, *process operation*, is concerned with the evaluation of business processes. The first activity measures the time related aspects of the process phases. The measurement data so obtained is statistically processed to draw any conclusions about its progress. Summary of this building block follows next,

Process operation

Participating roles:

Business process owner, IT process owner, SOA service manger, System and database administrator

Activities:

- Business activity monitoring
- Processing of results
- Interception

The service management module includes a building block responsible for implementation, testing, and deployment of services. Specifically, the test part starts after the service has been implemented. Service code is reviewed and service is tested. Successful acceptance tests send the service to deployment step. Several roles are involved at this stage such as service manager, testing specialist etc. However, testing of service-oriented systems is not as simple as seems to be from the description of this building block. Service testing involves several levels of testing described earlier but not visible here. Considering the complexity and breadth of testing issues involved in SOA perspective, this building block however seems to provide insufficient information. Its summary is given below,

Service implementation, testing, and deployment

Participating roles:

IT domain leader, Business analyst, SOA service manager, Testing specialist, Deployment manager, Business process owner, Head of line of business, IT process owner, IT project manager

Activities:

- Plan implementation project
- Negotiate implementation project plan
- Start implementation project
- Implement service
- Test service
- Review code
- Perform user acceptance test
- Deploy service
- Update service meta-data and endpoint reference

### 7.4.2 Adaptation of Light-TPEF for the considered case

***Light-TPEF Elements revisited***

The previous chapter has presented, Light-TPEF, as a framework to explicitly evaluate diverse quality aspects of all the elements of software testing processes. The framework is particularly aimed at addressing the implicity and partiality of current test evaluation approaches as well as providing a lightweight test measurement and evaluation methodology. The four inputs to the framework, namely research over software testing knowledge areas (chapter 2), research on evaluation theory [Shadish et al., 1991, Ch. 3], [Scriven, 1996a] concepts (chapter 3), IEEE Std. 1061-1998 Standard for a Software Quality Metrics Methodology, and ISO/IEC 14598-5 Information technology–Software product evaluation provide theoretical and practical foundations to address the challenges envisaged for the framework.

To this end, the framework defines one core component, evaluation process, which has been derived from IEEE 1061 and ISO/IEC 14598-5 standards and describes all the steps necessary to select, measure, and evaluate different test process entities. Five support and informative components have been derived from software testing knowledge areas and evaluation theory concepts to guide the steps of the core evaluation process component. The framework uses BPMN (as a graphical technique) and SIPO, a variant of IPO methodology (as a semi-formal textual technique) to describe the core component, while the support components use ordinary textual descriptions. Figure 7.11 outlines the Light-TPEF once again to reiterate the concept and to be able to visualize how it can be adapted to evaluating various specialized testing processes such as for embedded systems, distributed systems, or service-oriented systems.

***Adaptation outline for service testing process***

Taking a different perspective than the TPI SOA, the present framework, Light-TPEF, can

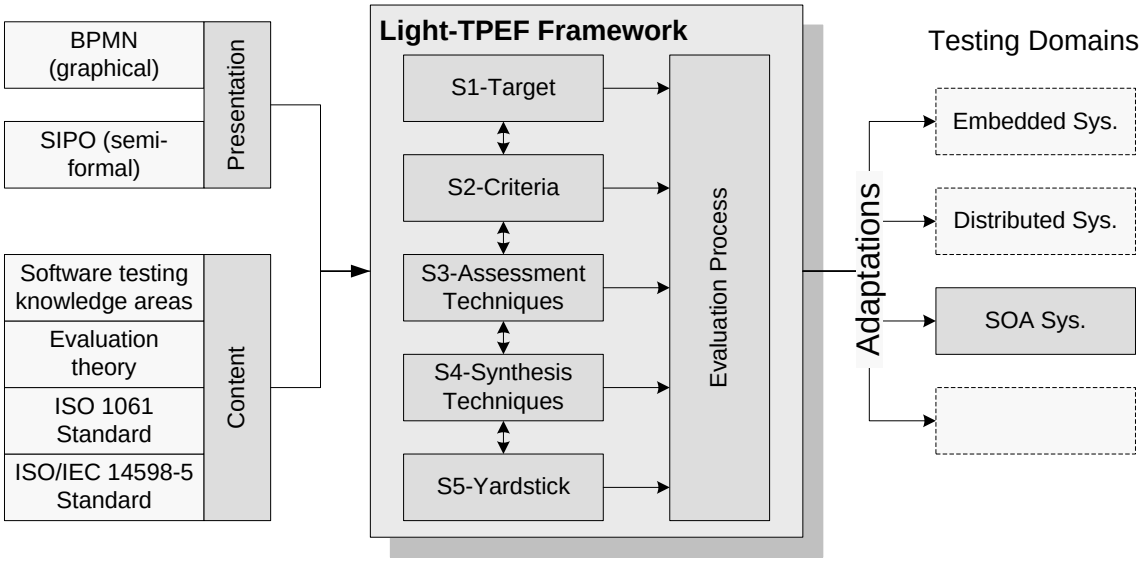


**Figure 7.11:** *Light-TPEF: Flashback and customizations*

help make evaluations of service testing activities. Figure 7.12 outlines connections among the service development, service testing, and the present framework. While the SOA lifecycle on the left part of the figure covers the phases of the complete SOA adoption and related issues, the service provider lifecycle next to it gives the lifecycle of a single service or group of services. Between the requirement analysis and testing/deployment stages of this service are involved the above mentioned testing levels. This offers the activity view of service testing. These activities have to be managed through a service testing process. The management view of the service testing process can comprise analysis & design, implementation & execution, and evaluation & reporting engulfed by a planning & control phase. An important role of planing & control phase is to control and monitor the progress of the testing process. As a most specific solution to fulfil this task, the Light-TPEF framework can complement this phase of the testing process. Challenges to adopting this approach for service test processes have earlier been discussed by the author [Farooq et al., 2008b]. Addressing these challenges, next part of this thesis presents a case where the Light-TPEF approach could be used in SOA development and testing situations.

***Adaptations needed***

Not all components of the Light-TPEF are affected by the type of testing required in a given context. For example, in case of SOA testing, we need to extend and redefine the *S1:Target* component with specifics of SOA testing elements, the *S2:Criteria* component with some of the SOA quality attributes, and the *S3:Assessment techniques* component with SOA related metrics. The rest of the framework components are quite generic and are independent of the choice of the testing background.

The first adaptation is with the first support component of the framework, Target. While the testware and testing resources elements remain same for service testing, the testing phases are slightly different which are the target of evaluations. The specialized levels of testing involved in this case are unit, integration, system testing, as well as testing for governance, underlying technologies, or normally uncommon non-functional testing. The figure 7.13 captures these elements as a summary information.

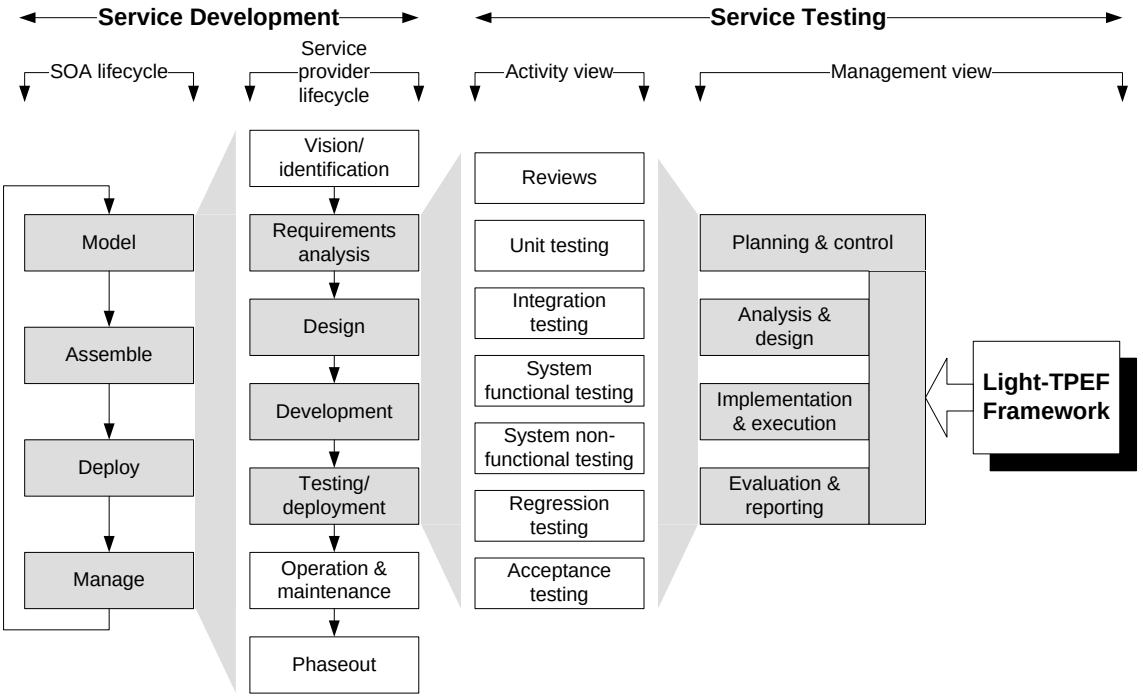


**Figure 7.12:** *Light-TPEF vs. Service Testing Process*

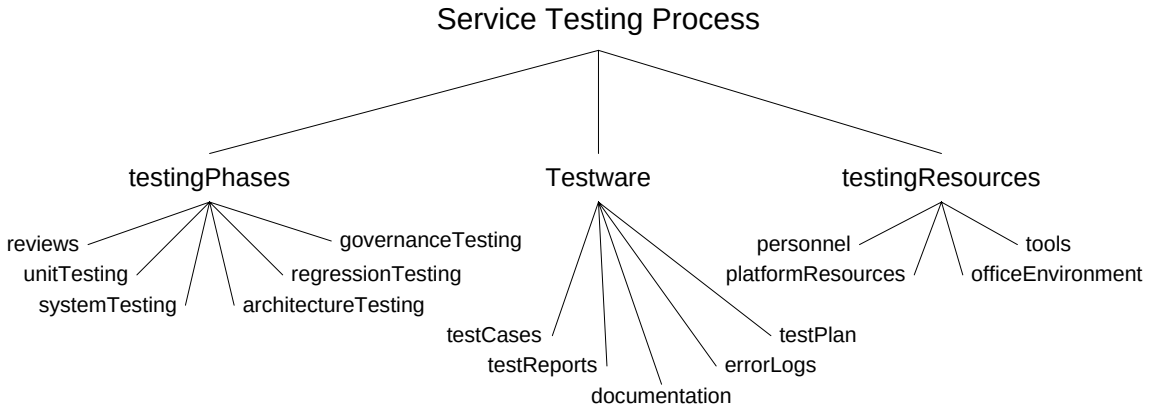


**Figure 7.13:** *Elements of Service Testing Process*

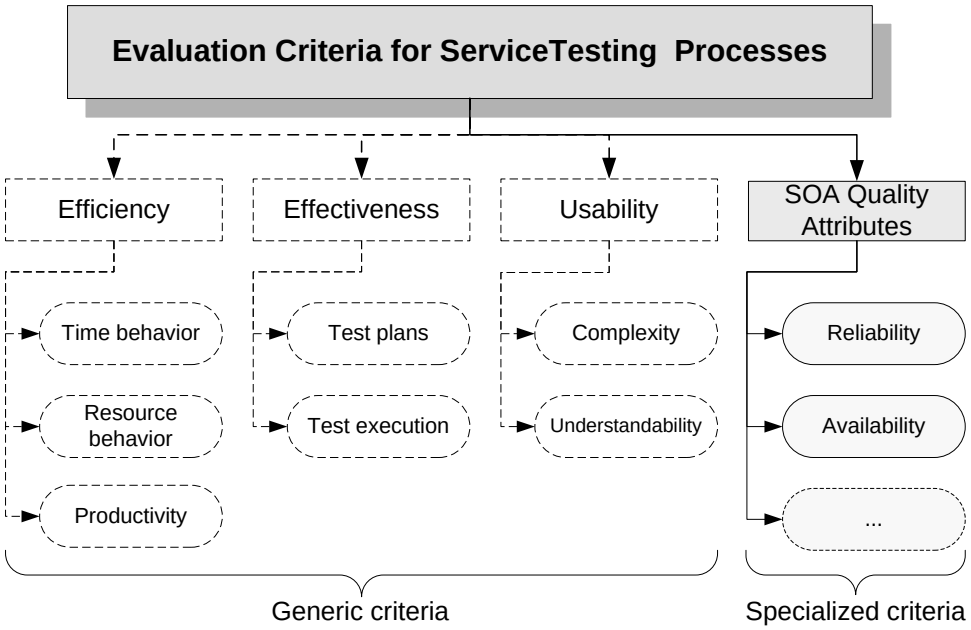


**Figure 7.14:** *Evaluation Criteria for Service Testing Process*

The second required adjustment is with the evaluation criteria. The generic evaluation criteria element S2 mentioned in section 6.3.1.2 is a set of generic quality attributes for the software testing process. This set contains two product specific quality attributes, reliability and functionality, as an evaluation criteria for the testing process. These attributes need to be replaced with SOA specific quality attributes [O'Brien et al., 2005] such as reliability and usability etc. In this way, the evaluation criteria for the service or SOA testing process will be a combination of generic and specific attributes as visualized in figure 7.14.

The third area where Light-TPEF should include SOA specific adjustments is the component on assessment techniques. The section 6.3.1.3 mentions a classification of several test metrics which are connected with the evaluation criteria attributes. In case of service/-SOA testing, specific product, process, and resource metrics such as [Rud et al., 2006], [Rud et al., 2007b], [Rud et al., 2007d] will add to the list of existing test metrics to help evaluate generic and service/SOA specific quality criteria mentioned in the previous section.

***Connections between Light-TPEF and the SOA governance model***

With the incorporation of the mentioned adaptations, Light-TPEF can augment the SOA governance modules mentioned in the previous section. It can be observed that there exist few relationships among some of these modules. Figure 7.15 highlights these relationships among *Process analysis, Process operation, and Service implementation, testing, & deployment* building blocks. It can be observed as summarized earlier that the task of testing services is considerably large and complex to be viewed as a sizable testing process. Eventually, it will need to be monitored as a process for its effectiveness, efficiency and other such concerns. The process analysis and process operation building blocks from the business process management module can help guide this monitoring and control. Specifically, the *process operation* building block as summarized above is about monitoring process

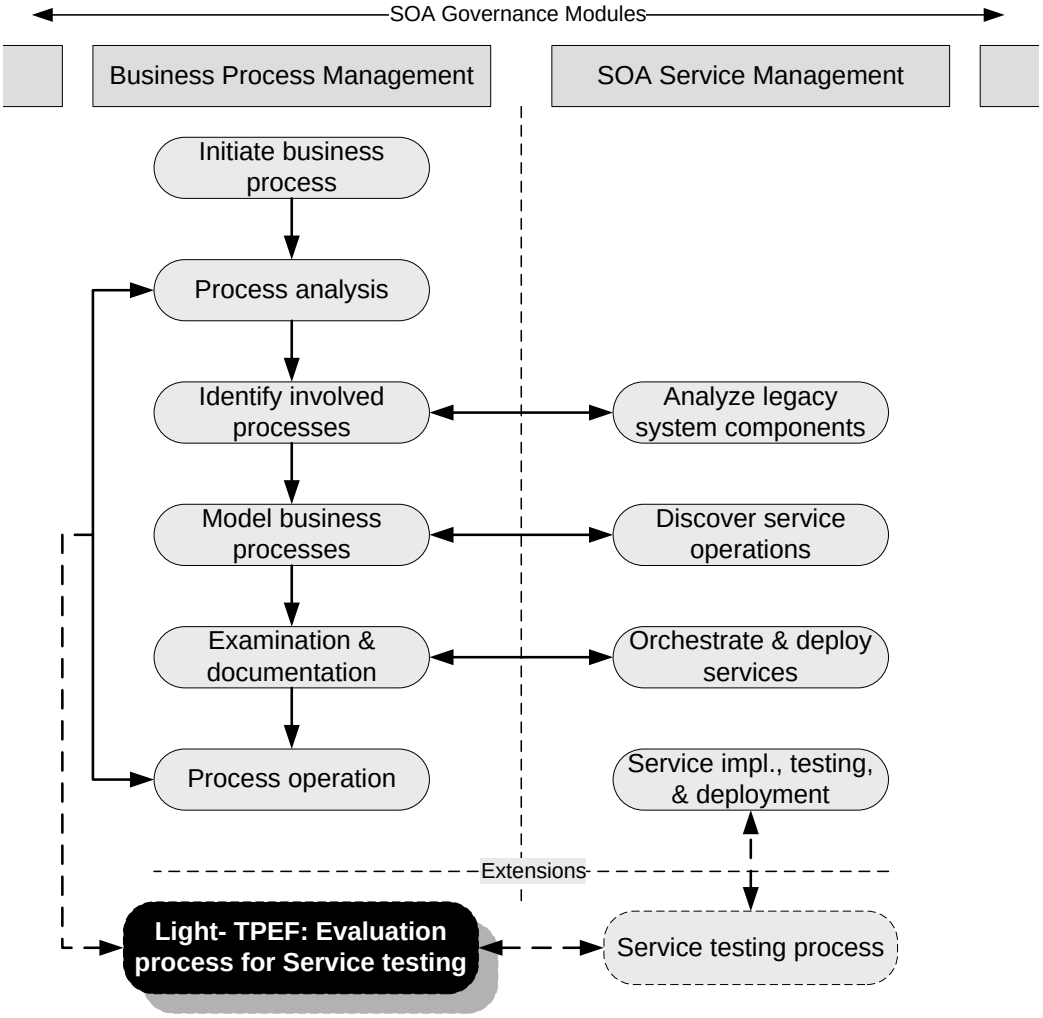


**Figure 7.15:** *Relationships among SOA Governance Modules*

progress. However, it is not enough detailed to cover all the aspects of measuring and evaluating the service test process tasks. As a more specialized approach, the Light-TPEF approach developed in this thesis can fill this gap. It can serve as a bridge between the business process management and SOA service management modules by sitting between the process analysis and process operation building blocks and the service implementation and testing building block. The figure 7.15 shows two extended building blocks, service testing process and the Light-TPEF as evaluation process for service testing process which enhance the connections between SOA business process management and service management.

## 7.5 Summary

Concluding the last phase of the research project at hand, this chapter concentrated and resolved last two sub-questions of the current research task.

RT-5. How can the idea conceived by RT-4 be transformed into a practical test process evaluation and improvement model?
RT-6. Can the approach developed as a result of RT-5 be validly applied in practical and industrial situations? What are the limitations and the needed adjustments in applying the new approach?

This chapter focused on the implementation and validation of the evaluation framework developed in this thesis. The considered implementation scenario was to convert the model into an executable business process to allow for semi automatic capturing of test metrics and fully automatic generation of evaluation results. Thus it would provide continuous test process measurement. Fundamental concepts of software process model, process modeling languages, and the business processes where explained in brief. Discussions also included short introduction of XPDL (XML process definition language) as an exchange format for process designs, and WS-BPEL (business process execution language) as an example of well-known executable process modeling language. Connections among these technologies in the context of current task were also established.

The chapter later outlined the implementation approach to be followed. It was constituted in converting the existing BPMN process descriptions into their equivalent XPDL or BPEL processes. It was done through one of the many open source process modeling and execution engines called TIBCO Business Studio. The XPDL version of the evaluation process was then sent to another process execution engine (Enhydra Shark) which was capable of directly executing the XPDL-based processes.

Principle of using this implementation scheme in an industrial environment was also explained. In a chosen industrial domain, testing of SOA-based systems came up as an example scenario. After briefly introducing the testing processes in the context of SOA-based development settings, the chapter explained integration of the implemented solution into SOA governance model. The integration seemed to be smooth and provided a useful example where the test evaluation framework provided an added value to the current models.

# 8 Summary & Future Work

## 8.1 Summary

In the context of evaluation and improvement of software testing processes, this dissertation attempts to find an answer to the following primary research question,

**Could a process be defined to allow medium size industrial organizations perform light-weight and explicit evaluation of their testing processes?**

As a result of this research a framework for the explicit evaluation of testing processes was developed. Organized research steps were needed to be followed to perform this PhD research which was to span over a few years. Addressing this challenge, the author chose to follow the *engineering paradigm* as the research method. It was one of the four common software engineering research methodologies; the choice being motivated by the type of research problem and solution that it presented. This thesis ran only a first cycle of this method consisting of observation of existing solution, proposal of a better solution, development of a better solution, and its measurement and analysis. A summary of each part of this research is given below,

*Chapter 1*

***Setting ground for research***

This chapter set the background of the current research project. Beginning with an introduction to the role of the software, the chapter led the reader to the definition and explanation of the research problem. The chapter established the research pattern and plan to be followed for solving the problem. Connections and various contextual aspects of the research project with reference to software engineering research environments were given.

*Chapter 2 through chapter 5*

***Observation of existing solutions***

A first-hand knowledge of the problem domain is an initial step for dissecting any research problem. A survey of the software testing knowledge and practice areas was performed (chapter 2). It enabled identification of core entities of software testing interesting in the context of the current problem. It was felt that some level of interdisciplinary inspection was needed to extract philosophy and principles of evaluation. It was done through a brief study of a non-software engineering discipline, evaluation research (chapter 3). The inquisition brought forward key elements of any evaluative work, some of which are often appear to be ignored by evaluation methods in the field of software testing and software engineering. After discovering these key components of evaluation, it seemed logical to review currently available evaluation methods and models in the testing domain. An

exhaustive survey of evaluation methods for testing processes, techniques and tools was performed (chapter 4). At this stage of research, precise requirements for a test evaluation approach in the light of the research problem were developed. The available test evaluation methods were judged based on this set of requirements (chapter 5). The results of the analysis showed a need for an improved solution that would satisfy the identified shortcomings in the current approaches.

*Chapter 6*

***Proposal & development of a better solution***

An improved solution was imposed in the form of an evaluation framework which took inputs from testing foundations, theory of evaluations, and well accepted process measurement standards. The framework consisting of five support components and one core component was enhanced to be described as a complete guide for evaluating software test processes. This was done with the help of graphical and textual process modeling notations. As a counter check, the solution was matched with the criteria set in the earlier phase of the research and was found to be consistent with its goals.

*Chapter 7 through chapter 8*

***Prototypical implementation and validation***

This phase of research attempted to transform the proposed framework as an exemplary workable solution. Open source process modeling tools were used to create a business process representing the evaluation framework. The business process was described in the form of an executable process model. As an industrial application of the propose concept, the service-oriented development environment at an international company was considered for an example. Adaptation of the developed framework in the context of SOA governance model at this company was explained. Finally, theoretical validation of the model for its consistency was performed using assertion techniques. Further extensive validity checks of the approach were left for future research, nonetheless the current research task had been successfully accomplished at this stage.

## 8.2 Thesis Contributions

This dissertation has made following main contributions in the field of evaluation and improvement of software test processes,

**Contribution # 1** A contemporary augmentation to the many surveys of the software testing field, but with special focus to candidate elements for evaluation.

**Contribution # 2** A first-of-its-kind survey of evaluation approaches in software testing, specifically a first chronological and comprehensive account of test process assessment and improvement models.

**Contribution # 3** An explicit and light-weight evaluation framework for software test processes covering all possible entities of test evaluation.

**Contribution # 4** Development of a test measurement approach that can potentially help organizations fulfill part of the requirements at maturity level 4 of the Testing Maturity Model.

**Contribution # 5** Presentation and implementation of a working evaluation process model in modern business process environments.

## 8.3 Future Work

The previous chapter has discussed a so called internal/theoretical validation of the evaluation framework concept using the assertion techniques. Additionally, an implementation environment discussed earlier provides only its pilot application. This could be improved by a full scale application of the approach in multiple projects and over a longer period of time. It is an ongoing process which needs considerable time to adequately validate the approach for truly realizing the its practical significance. It can provide important feedback about improving and adjusting the framework accordingly.

Recently an approach called causal network-based process model (CNPM) has been developed by Richter and Dumke [Richter and Dumke, 2008] for an explicit analysis of process models. Application of this approach for analysis of Capability Maturity Model Integration (CMMI) has discovered few inconsistencies and incompleteness of this process model. This capability of CNPM method as a validity check for process model creates the possibility of analyzing the current framework developed in this thesis. A causal network model representation of the test process evaluation framework is expected to be able to enhance the current approach by identifying any of its incompleteness or potential improvement areas.

The implementation of the evaluation framework given in chapter 7 represents it as an executable business process model. With the current design of the process model, the test metrics have yet to be collected through a manual process. This allows for a semi automated but continuous measurement and evaluation of the testing process. A related but alternative approach can improve this situation by providing a fully automated continuous measurement. If a model of software testing process and the current evaluation framework are described using some formal process modeling language, they can be merged into an integrated test process and evaluation process model through the use of some modeling tool. This can be converted into an executable business process model on the lines of implementation discussed in chapter 7. With this approach the test measurements can be defined in the model and can be automatically collected. The measurements can be synthesized according the procedures already defined in the framework. Thus it will eliminate the need to manually calculate and record test metrics and enable a completely automatic and continuous measurement and evaluation of testing processes.

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| **ACM** | Association for Computing Machinery |
| **AHP** | Analytic Hierarchy Process |
| **ASD** | Adaptive Software Development |
| **ATR** | Activities, Tasks, Responsibilities |
| **ATAM** | Architecture Tradeoff Analysis Method |
| **BNF** | Backus-Naur Form |
| **BPEL** | Business Process Execution Language |
| **BPMN** | Business Process Modeling Notation |
| **CAME** | Computer Assisted Software Measurement |
| **CASE** | Computer Aided Software Engineering |
| **CMM** | Capability Maturity Model |
| **CMMI** | Capability Maturity Model Integration |
| **CNPM** | Causal Network-Based Process Model |
| **COTS** | Commercial off-the-shelf |
| **CS** | Computer Science |
| **DoD** | Department of Defense |
| **ETXM** | Entry, Task, Exit, Measure |
| **ETVX** | Entry, Task, Validation, Exit |
| **EITVOX** | Entry, Input, Task, Validation, Output, Exit |
| **EIA** | Electronic Industries Alliance |
| **EU** | European Union |
| **EF** | Experience Factory |
| **FDD** | Feature Driven Development |
| **GQM** | Goal, Question, Metric |
| **GUI** | Graphical User Interface |
| **HTML** | Hypertext Markup Language |
| **IBM** | International Business Machines |
| **ICMM** | Inspection Capability Maturity Model |
| **IDC** | International Data Corporation |
| **IDEAL** | Initiating, Diagnosing, Establishing, Acting, Learning |
| **IEC** | International Electrotechnical Commission |
| **IEEE** | Institute of Electrical and Electronics Engineers |
| **IPO** | Input Process Output |
| **ISO** | International Organization for Standardization |
| **ISTQB** | International Software Testing Qualifications Board |
| **IT** | Information Technology |
| **ITIL** | IT Infrastructure Library |
| **IV&V** | Independent Verification & Validation |
| **JTC** | Joint Technical Committee |

| | |
|---|---|
| **NATO** | North Atlantic Treaty Organization |
| **MBT** | Model based Testing |
| $MB - V^2M^2$ | Metrics-based Verification and Validation Maturity Model |
| **NASA** | National Aeronautics and Space Administration |
| **PMBOK** | Project Management Body of Knowledge |
| **PSP** | Personal Software Process |
| **QIP** | Quality Improvement Paradigm |
| **ROI** | Return on Investment |
| **RT** | Research Question |
| **RUP** | Rational Unified Process |
| **SD** | Software Development Process |
| **SE** | Software Engineering |
| **SEI** | Software Engineering Institute |
| **SIPO** | Support, Input, Process, Output |
| **SLA** | Service Level Agreement |
| **SME** | Small and Medium sized Enterprise |
| **SOA** | Service-oriented Architecture |
| **SP** | Software Product |
| **SPI** | Software Process Improvement |
| **SPICE** | Software Process Improvement and Capability Determination |
| **SQuaRE** | Software product Quality Requirements and Evaluation |
| **SR** | Software Resources |
| **STD** | Software Technology Diagnostic |
| **STDR** | Software Development Technical Review |
| **SUT** | System under Test |
| **SW-CMM** | Software Capability Maturity Model |
| **SWEBOK** | Software Engineering Body of Knowledge |
| **TAMAR** | TMMi Assessment Method Application Requirements |
| **TDD** | Test Driven Development |
| **TIM** | Test Improvement Model |
| **TMap** | Test Management Approach |
| **TML** | Test Modeling Language |
| **TMM** | Test Maturity Model |
| **TMMi** | Test Maturity Model Integration |
| **TPAM** | Test Process Assessment Model |
| **TPEF** | Test Process Evaluation Framework |
| **TPI** | Test Process Improvement |
| **UML** | Unified Modeling Language |
| **V&V** | Verification and Validation |
| **W3C** | World Wide Web Consortium |
| **XML** | Extensible Markup Language |
| **XP** | Extreme Programming |
| **XPDL** | XML Process Definition Language |
| **XT** | Extreme Tailoring |

# Bibliography

**Note** All links to web resources have been checked and found to be working in a final step on 16.06.2009.

[iee, ] IEEE Standards Association. Available at `http://standards.ieee.org/`.

[iso, ] JTC 1- Information Technology, International Organization for Standardization. Available at `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_tc_browse.htm?commid=45020`.

[Abran et al., 2004] Abran, A., Bourque, P., Dupuis, R., and Moore, J. W., editors (2004). *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, Piscataway, NJ, USA.

[Abu et al., 2005] Abu, G., Cangussu, J. W., and Turi, J. (2005). A quantitative learning model for software test process. In *HICSS '05: Proceedings of the 38th Annual Hawaii International Conference on System Sciences - Track 3*, page 78.2, Washington, DC, USA. IEEE Computer Society.

[ACM, 1998] ACM (1998). The ACM computing classification system. Available at `http://www.acm.org/about/class`.

[Acuña et al., 2001] Acuña, S. T., Antonio, A. D., Ferré, X., López, M., and Maté, L. (2001). The software process: Modelling, evaluation and improvement. *Handbook of Software Engineering and Knowledge Engineering*, pages 193–237.

[Adrion, 1993] Adrion, W. R. (1993). Research methodology in software engineering: Summary of the Dagstuhl workshop on future directions in software engineering. *SIGSOFT Softw. Eng. Notes*, 18(1):35–48.

[Afzal, 2007] Afzal, W. (2007). Metrics in software test planning and test design processes. Master's thesis, Department of Software Engineering and Computer Science, Blekinge Institute of Technology, Ronneby, Sweden.

[Apel, 2005] Apel, S. (2005). Software reliability growth prediction-state of the art. Technical report, IESE-Report No. 034.05/E Fraunhofer Institute of Experimental Software Engineering.

[April, 2005] April, A. (2005). *$S3^m$-Model to Evaluate and Improve the Quality of Software Maintenance Process*. PhD thesis, University of Magdeburg, Magdeburg, Germany.

[Ares et al., 2000] Ares, J., Vazquez, R. G., Juzgado, N. J., Lopez, M., and Moreno, A. M. (2000). A more rigorous and comprehensive approach to software process assessment. *Software Process: Improvement and Practice*, 5(1):3–30.

[Arthur et al., 1999] Arthur, J. D., Groner, M. K., Hayhurst, K. J., and Holloway, C. M. (1999). Evaluating the effectiveness of independent verification and validation. *Computer*, 32(10):79–83.

[Arthur and Nance, 1996] Arthur, J. D. and Nance, R. E. (1996). Independent verification and validation: a missing link in simulation methodology? In *WSC '96: Proceedings of the 28th conference on Winter simulation*, pages 230–236, Washington, DC, USA. IEEE Computer Society.

[Basili and Weiss, 1984] Basili, V. R. and Weiss, D. M. (1984). A methodology for collecting valid software engineering data. *IEEE Trans. Software Eng.*, 10(6):728–738.

[BBC, 2008] BBC (2008). Software blamed for LSE failure, BBC News, 9 Sept. 2008. Available at `http://news.bbc.co.uk/2/hi/business/7605871.stm`.

[BCS, ] BCS. SIGiST Specialist Group in Software Testing, British Computer Society. Available at `http://www.testingstandards.co.uk/`.

[Beck, 2002] Beck, K. (2002). *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Beizer, 1990] Beizer, B. (1990). *Software Testing Techniques*. Van Nostrand Reinhold, New York, USA.

[Beizer, 1995] Beizer, B. (1995). *Black-box testing: techniques for functional testing of software and systems*. John Wiley & Sons, Inc., New York, NY, USA.

[Bennison, 2007] Bennison, M. J. (2007). Software bug took Skype out. *The Risks Digest*, 24(80).

[Bertolino, 2004] Bertolino, A. (2004). The (im)maturity level of software testing. *SIGSOFT Softw. Eng. Notes*, 29(5):1–4.

[Bertolino, 2007] Bertolino, A. (2007). Software testing research: Achievements, challenges, dreams. In *FOSE '07: Proceeding of International Conference on Software Engineering: Future of Software Engineering*, pages 85–103, Washington, DC, USA. IEEE Computer Society.

[Bieberstein et al., 2005] Bieberstein, N., Bose, S., Fiammante, M., Jones, K., and Shah, R. (2005). *Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap*. The developerWorks Series. IBM Press.

[Blin and Tsoukiàs, 2001] Blin, M.-J. and Tsoukiàs, A. (2001). Multi-criteria methodology contribution to the software quality evaluation. *Software Quality Journal*, 9(2):113–132.

[Boehm, 2006] Boehm, B. (2006). A view of 20th and 21st century software engineering. In *ICSE '06: Proceeding of the 28th international conference on Software engineering*, pages 12–29, New York, NY, USA. ACM Press.

[Boehm and Basili, 2001] Boehm, B. and Basili, V. R. (2001). Software defect reduction top 10 list. *Computer*, 34(1):135–137.

[Braungarten, 2007] Braungarten, R. (2007). *The SMPI model: A stepwise process model to facilitate software measurement process improvement along the measurement paradigms*. PhD thesis, University of Magdeburg, Magdeburg, Germany.

[Briand and Labiche, 2004] Briand, L. and Labiche, Y. (2004). Empirical studies of software testing techniques: challenges, practical strategies, and future research. *SIGSOFT Softw. Eng. Notes*, 29(5):1–3.

[Brüning et al., 2007] Brüning, S., Weißleder, S., and Malek, M. (2007). A fault taxonomy for service-oriented architecture. In *HASE '07: Proceedings of the 10th IEEE High Assurance Systems Engineering Symposium*, pages 367–368, Washington, DC, USA. IEEE Computer Society.

[Broekman and Notenboom, 2003] Broekman, B. and Notenboom, E. (2003). *Testing Embedded Software*. Addison-Wesley, Great Britain.

[Broy et al., 2001] Broy, M., Hartkopf, S., Kohler, K., and Rombach, D. (2001). Germany: Combining software and application competencies. *IEEE Software*, 18(4):93–95, 100.

[Bucchiarone et al., 2007] Bucchiarone, A., Melgratti, H., and Severoni, F. (2007). Testing service composition. In *ASSE'07: Proceedings of the 8th Argentine Symposium on Software Engineering*.

[Buchsein et al., 2008] Buchsein, R., Victor, F., Günther, H., and Machmeier, V. (2008). *IT-Management mit ITIL V3 Strategien, Kennzahlen, Umsetzung*. Vieweg+Teubner.

[Budgen, 2000] Budgen, D. (2000). Evaluation and assessment in software engineering. *J. Syst. Softw.*, 52(2-3):93–94.

[Burnstein, 2003] Burnstein, I. (2003). *Practical Software Testing: A Process-oriented Approach*. Springer Inc., New York, NY, USA.

[Canfora and Penta, 2006] Canfora, G. and Penta, M. D. (2006). Testing services and service-centric systems: Challenges and opportunities. *IT Professional*, 8(2):10–17.

[Cangussu, 2002] Cangussu, J. W. (2002). *A Mathematical Foundation for Software Process Control*. PhD thesis, Purdue University, West Lafayette, IN, USA.

[Cangussu, 2003] Cangussu, J. W. (2003). A stochastic control model of the software test process. In *ProSim'03: Proceedings of the Workshop on Software Process Simulation Modeling*.

[Cangussu et al., 2000] Cangussu, J. W., DeCarlo, R., and Mathur, A. (2000). A state variable model for the software test process. In *Proceedings of 13th International Conference on Software & Systems Engineering and their Applications, Paris-France*.

[Cangussu et al., 2001a] Cangussu, J. W., DeCarlo, R., and Mathur, A. P. (2001a). A state model for the software test process with automated parameter identification. In *Proceedings of 2001 IEEE International Conference on Systems, Man, and Cybernetics*, pages 706–711, Los Alamitos, CA, USA. IEEE Computer Society.

[Cangussu et al., 2002] Cangussu, J. W., DeCarlo, R. A., and Mathur, A. P. (2002). A formal model of the software test process. *IEEE Trans. Softw. Eng.*, 28(8):782–796.

[Cangussu et al., 2003] Cangussu, J. W., DeCarlo, R. A., and Mathur, A. P. (2003). Using sensitivity analysis to validate a state variable model of the software test process. *IEEE Trans. Softw. Eng.*, 29(5):430–443.

[Cangussu et al., 2001b] Cangussu, J. W., Mathur, A. P., and DeCarlo, R. A. (2001b). Feedback control of the software test process through measurements of software reliability. In *ISSRE '01: Proceedings of the 12th International Symposium on Software Reliability Engineering*, page 232, Washington, DC, USA. IEEE Computer Society.

[Charette, 2005] Charette, R. N. (2005). Why software fails. *IEEE Spectrum*, 42(9):36–43.

[Chatzigeorgiou and Antoniadis, 2003] Chatzigeorgiou, A. and Antoniadis, G. (2003). Efficient management of inspections in software development projects. *Information & Software Technology*, 45(10):671–680.

[Chen, 2005] Chen, C. (2005). Measuring the movement of a research paradigm. In *VDA 2005: Proceedings of Conference on Visualization and Data Analysis*, pages 63–76. SPIE and IS&T.

[Chen, 1996] Chen, H.-T. (1996). A comprehensive typology for program evaluation. *American Journal of Evaluation*, 17(1):121–130.

[Chen et al., 2004] Chen, Y., Probert, R. L., and Robeson, K. (2004). Effective test metrics for test strategy evolution. In *CASCON '04: Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research*, pages 111–123. IBM Press.

[Chernak, 2004] Chernak, Y. (2004). Introducing TPAM: Test process assessment model. *Crosstalk-The Journal of Defense Software Engineering*. June Issue.

[Ciolkowski et al., 2003] Ciolkowski, M., Laitenberger, O., and Biffl, S. (2003). Software reviews: The state of the practice. *IEEE Software*, 20(06):46–51.

[Clarke and Dawson, 1999] Clarke, A. and Dawson, R. (1999). *Evaluation Research: An Introduction to Principles, Methods and Practice*. Sage Publications, Bonhill Street, London, UK.

[Clarke and Wing, 1996] Clarke, E. M. and Wing, J. M. (1996). Formal methods: state of the art and future directions. *ACM Comput. Surv.*, 28(4):626–643.

[Colville, 2004] Colville, J. (2004). Sydney trains disrupted by software glitch. *The Risks Digest*, 23(35).

[Davis, 1995] Davis, A. M. (1995). *201 principles of software development*. McGraw-Hill, Inc., New York, NY, USA.

[Deming, 1986] Deming, W. E. (1986). *Out of the Crisis*. MIT Press.

[Demirörs and Güceğlioğlu, 2006] Demirörs, O. and Güceğlioğlu, A. (2006). A case study for measuring process quality attributes. Technical report, Middle East Technical University, Informatics Institute.

[Drabick, 2003] Drabick, R. D. (2003). *Best Practices for the Formal Software Testing Process: A Menu of Testing Tasks*. Dorset House.

[Dranidis et al., 2007] Dranidis, D., Kourtesis, D., and Ramollari, E. (2007). Formal verification of web service behavioural conformance through testing. *Annals of Mathematics, Computing & Teleinformatics*, 1(5):36–43.

[Dumke, 2005] Dumke, R. R. (2005). Software measurement frameworks. In *Proceedings of the 3rd World Congress on Software Quality*, pages 72–82, Erlangen, Germany. International Software Quality Institute GmbH.

[Dumke et al., 2006a] Dumke, R. R., Braungarten, R., Blazey, M., Hegewald, H., Reitz, D., and Richter, K. (2006a). Software process measurement and control - a measurement-based point of view of software processes. Technical report, Dept. of Computer Science, University of Magdeburg, Germany. Available at `http://ivs.cs.uni-magdeburg.de/sw-eng/agruppe/forschung/paper/FormalM.pdf`.

[Dumke et al., 2006b] Dumke, R. R., Braungarten, R., Blazey, M., Hegewald, H., Reitz, D., and Richter, K. (2006b). Structuring software process metrics. In *IWSM/MetriKon 2006: Proceedings of the 16th International Workshop on Software Metrics and DASMA Software Metrik Kongress*, pages 483–497, Aachen, Germany. Shaker Verlag GmbH.

[Dumke et al., 2006c] Dumke, R. R., Braungarten, R., Kunz, M., Schmietendorf, A., and Wille, C. (2006c). Strategies and appropriateness of software measurement frameworks. In *MENSURA 2006: Proceedings of the International Conference on Software Process and Product Measurement*, pages 150–170, Spain. Servicio de Publicaciones de la Universidad de Cádiz.

[Dumke et al., 2004] Dumke, R. R., Côté, I., and Andruschak, O. (2004). Statistical process control (SPC) - a metric-based point of view of software processes achieving the CMMI level four. Technical report, Dept. of Computer Science, University of Magdeburg, Germany. Available at `http://ivs.cs.uni-magdeburg.de/sw-eng/agruppe/forschung/paper/SPCPreprint.pdf`.

[Dumke and Foltin, 1999] Dumke, R. R. and Foltin, E. (1999). An object-oriented software measurement and evaluation framework. In *FESMA 99: Proceedings of the 2nd European Software Measurement Conference*, pages 59–68.

[Dumke and Grigoleit, 1997] Dumke, R. R. and Grigoleit, H. (1997). Efficiency of CAME tools in software quality assurance. *Software Quality Journal*, 6(2):157–169.

[Dumke et al., 2005] Dumke, R. R., Schmietendorf, A., and Zuse, H. (2005). Formal descriptions of software measurement and evaluation-a short overview and evaluation. Technical report, Dept. of Computer Science, University of Magdeburg, Germany. Available at `http://ivs.cs.uni-magdeburg.de/sw-eng/agruppe/forschung/paper/ProcPreprintFinal.pdf`.

[Durant, 1993] Durant, J. (1993). Software testing practices survey report. Technical report, Software Practices Research Center.

[Dustdar and Haslinger, 2004] Dustdar, S. and Haslinger, S. (2004). Testing of service-oriented architectures - a practical approach. In *Net.ObjectDays: Proceedings of 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a NetworkedWorld*, volume 3263 of *Lecture Notes in Computer Science*, pages 97–109. Springer.

[Dyba, 2005] Dyba, T. (2005). An empirical investigation of the key factors for success in software process improvement. *IEEE Trans. Softw. Eng.*, 31(5):410–424.

[Ebert et al., 2004] Ebert, C., Dumke, R., Bundschuh, M., and Schmietendorf, A. (2004). *Best Practices in Software Measurement*. Springer Verlag.

[Ebert and Dumke, 2007] Ebert, C. and Dumke, R. R. (2007). *Software Measurement: Establish Extract Evaluate Execute*. Springer-Verlag Berlin Heidelberg.

[Eggink et al., 2008] Eggink, J. M., Wilhelmus, L., and Hulleman, R. (2008). TPI® SOA model version 1.0. Sogeti Netherlands, October 2008. Available at `http://www.sogeti.nl/images/TPI%20SOA%20Model%20v1.0_tcm6-47915.pdf`.

[Eickelmann et al., 2002] Eickelmann, N. S., Ruffolo, F., Baik, J., and Anant, A. (2002). An empirical study of modifying the Fagan inspection process and the resulting main effects and interaction effects among defects found, effort required, rate of preparation and inspection, number

of team members and product. In *SEW'02: Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop*, page 58, Washington, DC, USA. IEEE Computer Society.

[El-Far and Whittaker, 2001] El-Far, I. K. and Whittaker, J. A. (2001). *Encyclopedia of Software Engineering*, chapter Model-based Software Testing, pages 825–837. Wiley.

[Ericson et al., 1997] Ericson, T., Subotic, A., and Ursing, S. (1997). TIM a test improvement model. *J. Softw. Test., Verif. Reliab.*, 7(4):229–246.

[Erl, 2007] Erl, T. (2007). *SOA Principles of Service Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[ESA, 2008] ESA (2008). European software industry: Looking for a competitive advantage. European Software Association. Available at `http://www.europeansoftware.org/documents/softwarestrategywhitepaperFINAL.pdf`.

[Everett et al., 2007] Everett, G. D., Raymond, and Jr., M. (2007). *Software Testing: Testing Across the Entire Software Development Life Cycle*. Wiley InterScience, Hobokon, NJ, USA.

[Fagan, 1986] Fagan, M. E. (1986). Advances in software inspections. *IEEE Trans. Softw. Eng.*, 12(7):744–751.

[Farooq and Dumke, 2007] Farooq, A. and Dumke, R. R. (2007). Research directions in verification & validation process improvement. *SIGSOFT Softw. Eng. Notes*, 32(4):3.

[Farooq and Dumke, 2008a] Farooq, A. and Dumke, R. R. (2008a). Developing and applying a consolidated evaluation framework to analyze test process improvement approaches. In Cuadrado-Gallego, J., Braungarten, R., Dumke, R., and Abran, A., editors, *Software Process and Produce Measurement*, volume 4895 of *Lecture Notes in Computer Science*, pages 114–128. Springer-Verlag Berlin/Heidelberg.

[Farooq and Dumke, 2008b] Farooq, A. and Dumke, R. R. (2008b). Evaluation approaches in software testing. Technical report, Dept. of Computer Science, University of Magdeburg, Germany. Available at `http://ivs.cs.uni-magdeburg.de/sw-eng/agruppe/forschung/TR_Farooq.pdf`.

[Farooq et al., 2008a] Farooq, A., Dumke, R. R., Schmietendorf, A., and Hegewald, H. (2008a). A classification scheme for test process metrics. In *SEETEST 2008: Proceedings of South East European Software Testing Conference*, Heidelberg, Germany. dpunkt.verlag.

[Farooq et al., 2008b] Farooq, A., Georgieva, K., and Dumke, R. R. (2008b). Challenges in evaluating SOA test processes. In Dumke, R., Braungarten, R., Büren, G., Abran, A., and Cuadrado-Gallego, J., editors, *Software Process and Product Measurement*, volume 5338 of *Lecture Notes in Computer Science*, pages 107–113. Springer-Verlag Berlin/Heidelberg.

[Farooq et al., 2008c] Farooq, A., Georgieva, K., and Dumke, R. R. (2008c). A meta-measurement approach for software test processes. In *INMIC 2008: Proceedings of 12th IEEE International Multitopic Conference*, pages 333–338. IEEE Computer Society.

[Farooq et al., 2007] Farooq, A., Hegewald, H., and Dumke, R. R. (2007). A critical analysis of the Testing Maturity Model. *Metrics News, Journal of GI-Interest Group on Software Metrics*, 12(1):35–40.

[Farooq et al., 2008d] Farooq, A., Schmietendorf, A., and Dumke, R. R. (2008d). A quantitative evaluation framework for software test process. In *CONQUEST 2008: Proceedings of the International Conference on Quality Engineering in Software Technology*, pages 1–14, Aachen, Germany. Shaker Verlag GmbH.

[Fenton et al., 1994] Fenton, N., Pfleeger, S. L., and Glass, R. L. (1994). Science and substance: A challenge to software engineers. *IEEE Software*, 11(4):86–95.

[Fewster and Graham, 1999] Fewster, M. and Graham, D. (1999). *Software test automation: effective use of test execution tools*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

[Florac and Carleton, 1999] Florac, W. A. and Carleton, A. D. (1999). *Measuring the Software Process: statistical process control for software process improvement*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Foos et al., 2008] Foos, R., Bunse, C., Höpfner, H., and Zimmermann, T. (2008). TML: an XML-based test modeling language. *SIGSOFT Softw. Eng. Notes*, 33(2):1–6.

[Freimut and Vollei, 2005] Freimut, B. and Vollei, F. (2005). Determining inspection cost-effectiveness by combining project data and expert opinion. *IEEE Trans. Softw. Eng.*, 31(12):1074–1092.

[Fu et al., 2004] Fu, C., Ryder, B. G., Milanova, A., and Wonnacott, D. (2004). Testing of java web services for robustness. In *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, pages 23–34, New York, NY, USA. ACM.

[Fuggetta, 2000] Fuggetta, A. (2000). Software process: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 25–34, New York, NY, USA. ACM Press.

[Garvin, 1984] Garvin, D. A. (1984). What does "product quality" really mean? *Sloan Management Review*, 26(1):25–43.

[Güceğlioğlu and Demirörs, 2005a] Güceğlioğlu, A. and Demirörs, O. (2005a). A process based model for measuring process quality attributes. In Richardson, I., Abrahamsson, P., and Messnarz, R., editors, *Software Process Improvement*, volume 3792 of *Lecture Notes in Computer Science*, pages 118–129. Springer-Verlag Berlin/Heidelberg.

[Güceğlioğlu and Demirörs, 2005b] Güceğlioğlu, A. and Demirörs, O. (2005b). Using software quality characteristics to measure business process quality. In van der Aalst, W., Benatallah, B., Casati, F., and Curbera, F., editors, *Business Process Management*, volume 3649 of *Lecture Notes in Computer Science*, pages 374–379. Springer-Verlag Berlin/Heidelberg.

[Gelperin and Hetzel, 1988] Gelperin, D. and Hetzel, B. (1988). The growth of software testing. *Commun. ACM*, 31(6):687–695.

[Glass, 1994] Glass, R. L. (1994). The software-research crisis. *IEEE Software*, 11(6):42–47.

[Glass, 1995] Glass, R. L. (1995). A structure-based critique of contemporary computing research. *J. Syst. Softw.*, 28(1):3–7.

[Glass, 1998] Glass, R. L. (1998). *Software Runaways: Monumental Software Disasters*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Glass, 2008] Glass, R. L. (2008). Software: Hero or zero? *IEEE Software*, 25(3):96, 95.

[Glass et al., 2004] Glass, R. L., Ramesh, V., and Vessey, I. (2004). An analysis of research in computing disciplines. *Commun. ACM*, 47(6):89–94.

[Glass et al., 2002] Glass, R. L., Vessey, I., and Ramesh, V. (2002). Research in software engineering, an analysis of the literature. *Information and Software Technology*, 40(1):491–506.

[Goslin et al., 2008a] Goslin, A., Olsen, K., O'Hara, F., Miller, M., Thompson, G., and van Veenendaal, E. (2008a). *TMMi Assessment Method Application Requirements (TAMAR) Version 1.0*. TMMi® Foundation. Available at `http://www.tmmifoundation.org/downloads/tmmi/TMMi.TAMAR.pdf`.

[Goslin et al., 2008b] Goslin, A., Olsen, K., O'Hara, F., Miller, M., Thompson, G., and Wells, B. (2008b). *Test Maturity Model Integration-TMMi*. TMMi® Foundation. Available at `http://www.tmmifoundation.org/downloads/resources/TMMi%20Framework.pdf`.

[Graves et al., 2001] Graves, T. L., Harrold, M. J., Kim, J.-M., Porter, A., and Rothermel, G. (2001). An empirical study of regression test selection techniques. *ACM Trans. Softw. Eng. Methodol.*, 10(2):184–208.

[Grembergen, 2001] Grembergen, W. V., editor (2001). *Information technology evaluation methods and management*. John Wiley & Sons, Inc., New York, NY, USA.

[Gutjahr, 1999] Gutjahr, W. J. (1999). Partition testing vs. random testing: The influence of uncertainty. *IEEE Trans. Softw. Eng.*, 25(5):661–674.

[Halvorsen and Conradi, 2001] Halvorsen, C. P. and Conradi, R. (2001). A taxonomy to compare SPI frameworks. In *EWSPT '01: Proceedings of the 8th European Workshop on Software Process Technology*, pages 217–235, London, UK. Springer-Verlag.

[Hamann, 2006] Hamann, D. (2006). *Towards an Integrated Approach for Software Process Improvement: Combining Software Process Assessment and Software Process Modeling*. PhD thesis, Fraunhofer-Institute of Experimental Software Engineering, Univ. of Kaiserslautern, Kaiserslautern, Germany.

[Harris, 2006] Harris, I. G. (2006). A coverage metric for the validation of interacting processes. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 1019–1024, 3001 Leuven, Belgium, Belgium. European Design and Automation Association.

[Harrold, 2000] Harrold, M. J. (2000). Testing: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 61–72, New York, NY, USA. ACM Press.

[Havey, 2005] Havey, M. (2005). *Essential Business Process Modeling*. O'Reilly Media, Inc.

[Heitmeyer, 2005] Heitmeyer, C. (2005). A panacea or academic poppycock: Formal methods revisited. In *HASE '05: Proceedings of the Ninth IEEE International Symposium on High-Assurance Systems Engineering*, pages 3–7, Washington, DC, USA. IEEE Computer Society.

[Holz et al., 2006] Holz, H. J., Applin, A., Haberman, B., Joyce, D., Purchase, H., and Reed, C. (2006). Research methods in computing: what are they, and how should we teach them? *SIGCSE Bull.*, 38(4):96–114.

[Huang et al., 2008] Huang, H. Y., Liu, H. H., Li, Z. J., and Zhu, J. (2008). Surrogate: A simulation apparatus for continuous integration testing in service oriented architecture. In *SCC 2008: Proceedings of IEEE International Conference on Services Computing*, pages 223–230, Los Alamitos, CA, USA. IEEE Computer Society.

[Humphrey, ] Humphrey, W. S. The software quality profile. Available at `http://www.sei.cmu.edu/publications/articles/quality-profile/index.html`.

[Humphrey, 1989] Humphrey, W. S. (1989). *Managing the software process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Humphrey et al., 2007] Humphrey, W. S., Konrad, M. D., Over, J. W., and Peterson, W. C. (2007). Future directions in process improvement. *Crosstalk-The Journal of Defense Software Engineering*. February Issue.

[Hutcheson, 2003] Hutcheson, M. L. (2003). *Software Testing Fundamentals: Methods and Metrics*. John Wiley & Sons, Inc., New York, NY, USA.

[iABG, ] iABG. V-Modell XT, iABG Industrieanlagen-Betriebsgesellschaft mbH. Available at `http://v-modell.iabg.de/`.

[IEEE, 1990] IEEE (1990). Std 610.12-1990:IEEE standard glossary of software engineering terminology.

[IEEE, 1992] IEEE (1992). Std 1209:IEEE recommended practice for evaluation and selection of CASE tools.

[IEEE, 1997a] IEEE (1997a). Std 1028:IEEE standard for software reviews.

[IEEE, 1997b] IEEE (1997b). Std 1074: IEEE standard for developing software life cycle processes.

[IEEE, 1998] IEEE (1998). Std 829: IEEE standard for software test documentation.

[IEEE/EIA, 1998] IEEE/EIA (1998). Std 12207: Standard for information technology-software life cycle processes.

[Illes et al., 2005] Illes, T., Herrmann, A., Paech, B., and Rückert, J. (2005). Criteria for software testing tool evaluation-a task oriented view. In *Proceedings of the 3rd World Congress for Software Quality*. Available at `http://www.softwareforschung.de/fileadmin/_primium/downloads/publikationen/IHPR2005.pdf`.

[ISO/IEC, 1998] ISO/IEC (1998). Standard 15504 : Information technology - software process assessment – part 7 : Guide for use in process improvement.

[ISO/IEC, 2004] ISO/IEC (2004). Standard 90003: Software engineering – guidelines for the application of ISO 9001:2000 to computer software.

[ISTQB, 2006] ISTQB (2006). Standard glossary of terms used in software testing.

[Jacobs and Trienekens, 2002] Jacobs, J. C. and Trienekens, J. J. M. (2002). Towards a metrics based verification and validation maturity model. In *STEP '02: Proceedings of the 10th International Workshop on Software Technology and Engineering Practice*, page 123, Washington, DC, USA. IEEE Computer Society.

[Jr. et al., 2005] Jr., R. H., Kinder, S., and Graham, S. (2005). IBM's SOA foundation. IBM White Paper. Available at http://www.ibm.com/developerworks/webservices/library/ws-soa-whitepaper/.

[Juran, 1988] Juran, J. M. (1988). *Juran on Planning for Quality*. Free Press.

[Juristo et al., 2002] Juristo, N., Moreno, A. M., and Vegas, S. (2002). A survey on testing technique empirical studies: How limited is our knowledge. In *ISESE '02: Proceedings of the 2002 International Symposium on Empirical Software Engineering*, page 161, Washington, DC, USA. IEEE Computer Society.

[Juristo et al., 2004a] Juristo, N., Moreno, A. M., and Vegas, S. (2004a). Reviewing 25 years of testing technique experiments. *Empirical Softw. Engg.*, 9(1-2):7–44.

[Juristo et al., 2004b] Juristo, N., Moreno, A. M., and Vegas, S. (2004b). Towards building a solid empirical body of knowledge in testing techniques. *SIGSOFT Softw. Eng. Notes*, 29(5):1–4.

[Kan, 2002] Kan, S. H. (2002). *Metrics and Models in Software Quality Engineering Second Edition*. Addison-Wesley Pub. Company, Inc.

[Kan et al., 2001] Kan, S. H., Parrish, J., and Manlove, D. (2001). In-process metrics for software testing. *IBM Systems Journal*, 40(1):220–241.

[Kandt, 2005] Kandt, R. K. (2005). *Software Engineering Quality Practices (Applied Software Engineering)*. Auerbach Publications, Boston, MA, USA.

[Kazman et al., 2000] Kazman, R., Klein, M., and Clements, P. (2000). ATAM: Method for architecture evaluation. Technical report, Software Engineering Institute, SEI. Available at http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00tr004.pdf.

[Kelly and Shepard, 2002] Kelly, D. and Shepard, T. (2002). Qualitative observations from software code inspection experiments. In *CASCON '02: Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research*, page 5. IBM Press.

[Kenett and Baker, 1999] Kenett, R. S. and Baker, E. R. (1999). *Software Process Quality Management and Control*. Marcel Dekker Inc., New York, NY, USA.

[Kit and Finzi, 1995] Kit, E. and Finzi, S. (1995). *Software testing in the real world: improving the process*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

[Kitchenham et al., 1997] Kitchenham, B., Brereton, P., Budgen, D., Linkman, S., Almstrum, V. L., and Pfleeger, S. L. (1997). Evaluation and assessment in software engineering. *Information and Software Technology*, 39(11):731–734.

[Kollanus, 2005] Kollanus, S. (2005). ICMM- inspection capability maturity model. In Kokol, P., editor, *Proceedings of IASTED International Conference on Software Engineering, part of the 23rd Multi-Conference on Applied Informatics*, pages 372–377. ACTA Press.

[Kollanus, 2009] Kollanus, S. (2009). Experiences from using ICMM in inspection process assessment. *Software Quality Journal*, 17(2):177–187.

[Kollanus and Koskinen, 2007] Kollanus, S. and Koskinen, J. (2007). Survey of software inspection research: 1991-2005. Technical report, University Of Jyväskylä, Department of Computer Science and Information Systems. Available at http://users.jyu.fi/~kolli/research/Inspection_survey_WP.pdf.

[Komi-Sirviö, 2004] Komi-Sirviö, S. (2004). *Development and Evaluation of Software Process Improvement Methods*. PhD thesis, Faculty of Science, University of Oulu, Oulu, Finland.

[Kontogiannis et al., 2008] Kontogiannis, K., Lewis, G. A., and Smith, D. B. (2008). A research agenda for service-oriented architecture. In *SDSOA '08: Proceedings of the 2nd international workshop on Systems development in SOA environments*, pages 1–6, New York, NY, USA. ACM.

[Koomen, 2002] Koomen, T. (2002). Worldwide survey on Test Process Improvement. Technical report, Sogeti. Available at `http://www.sogeti.se/upload/vara_tjanster/dokument/tpi_survey2002.pdf`.

[Koomen and Notenboom, 2004] Koomen, T. and Notenboom, E. (2004). Worldwide survey on Test Process Improvement. Technical report, Sogeti. Available at `http://www.sogeti.nl/Home/Expertise/Testen/tpi_survey_uk.jsp`.

[Koomen and Pol, 1999] Koomen, T. and Pol, M. (1999). *Test Process Improvement: a Practical Step-by-Step Guide to Structured Testing*. Addison-Wesley, New York, NY, USA.

[Kozlov, 2005] Kozlov, D. (2005). A formal model for evaluation of the software development process based on quality indexes. In *QAOOSE 2005: Proceedings of the 9th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*. Available at `http://www.iro.umontreal.ca/~sahraouh/qaoose2005/paper8.pdf`.

[Krebs et al., 2005] Krebs, W., Ho, C.-W., Williams, L., and Layman, L. (2005). Rational unified process evaluation framework version 1.0. Technical report, IBM Corporation and North Carolina State University, USA. Available at `ftp://ftp.ncsu.edu/pub/unity/lockers/ftp/csc_anon/tech/2005/TR-2005-46.pdf`.

[Kuhn, 1996] Kuhn, T. S. (1996). *The Structure of Scientific Revolutions Third Edition*. The University of Chicago Press, Chicago, IL, USA.

[Ladkin, 2006] Ladkin, P. B. (2006). A380 delivery delays attributed partly to design SW problems. *The Risks Digest*, 24(45).

[Laitenberger, 2002] Laitenberger, O. (2002). A survey of software inspection technologies. *Handbook on Software Eng. and Knowledge Eng.*, 2:517–555.

[Laitenberger et al., 1999] Laitenberger, O., Leszak, M., Stoll, D., and Emam, K. E. (1999). Quantitative modeling of software reviews in an industrial setting. In *METRICS '99: Proceedings of the 6th International Symposium on Software Metrics*, page 312, Washington, DC, USA. IEEE Computer Society.

[Leveson, 2004] Leveson, N. G. (2004). Role of software in spacecraft accidents. *Journal of Spacecraft and Rockets*, 41(4):564–575.

[Lewis, 2004] Lewis, W. E. (2004). *Software Testing and Continuous Quality Improvement, Second Edition*. Auerbach Publications, Boca Raton, FL, USA.

[Liggesmeyer, 1995] Liggesmeyer, P. (1995). A set of complexity metrics for guiding the software test process. *Software Quality Journal*, 4(4):257–273.

[Liggesmeyer, 2002] Liggesmeyer, P. (2002). *Software-Qualität. Testen, Analysieren und Verifizieren von Software*. Spektrum Akademischer Verlag, Berlin, Germany.

[Limböck, 2009] Limböck, G. (2009). Measuring the quality of standard software: the SAP quality index. In *Software Quality Days 2009*.

[Lopez, 2000] Lopez, M. (2000). An evaluation theory perspective of the architecture tradeoff analysis methodsm (ATAM). Technical report, Software Engineering Institute, SEI. Available at `http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00tr012.pdf`.

[Lopez, 2003] Lopez, M. (2003). Application of an evaluation framework for analyzing the architecture tradeoff analysis method. *J. Syst. Softw.*, 68(3):233–241.

[Lüttgen, 2006] Lüttgen, G. (2006). Formal verification & its role in testing. Technical Report YCS-2006-400, Department of Computer Science, University of York, England. Available at `http://www-users.cs.york.ac.uk/~luettgen/publications/pdf/York-YCS-2006-400.pdf`.

[Lu and Sadiq, 2007] Lu, R. and Sadiq, S. W. (2007). A survey of comparative business process modeling approaches. In Abramowicz, W., editor, *Business Information Systems*, volume 4439 of *Lecture Notes in Computer Science*, pages 82–94. Springer-Verlag Berlin/Heidelberg.

[Lázaro and Marcos, 2005] Lázaro, M. and Marcos, E. (2005). Research in software engineering: Paradigms and methods. In *CAiSE Workshops Vol. 2, Proceedings of the 17th International Conference, CAiSE 2005, Porto, Portugal*, pages 517–522. FEUP Edições, Porto.

[Marquis et al., 2005] Marquis, S., Dean, T. R., and Knight, S. (2005). SCL: a language for security testing of network applications. In *CASCON '05: Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, pages 155–164. IBM Press.

[McCaffrey, 2005] McCaffrey, J. (2005). Test run: The analytic hierarchy process. *MSDN Magazine*. Available at `http://msdn.microsoft.com/en-us/magazine/cc163785.aspx`.

[McGibbon, 2005] McGibbon, S. (2005). Growth and jobs from the European software industry. *European Review of Political Technologies*, 3:1–13.

[Meyer, 2008] Meyer, B. (2008). Seven principles of software testing. *Computer*, 41(8):99–101.

[Michael et al., 2002] Michael, J. B., Bossuyt, B. J., and Snyder, B. B. (2002). Metrics for measuring the effectiveness of software-testing tools. In *ISSRE '02: Proceedings of the 13th International Symposium on Software Reliability Engineering (ISSRE'02)*, page 117, Washington, DC, USA. IEEE Computer Society.

[Microsoft, ] Microsoft. The economic impact of IT, software, and the Microsoft ecosystem on the global economy. International Data Corp. Available at `http://www.microsoft.com/downloads/details.aspx?FamilyId=BB95083E-2BCA-4C60-832C-9B35A2A6BC6D&displaylang=en`.

[Morasca and Serra-Capizzano, 2004] Morasca, S. and Serra-Capizzano, S. (2004). On the analytical comparison of testing techniques. In *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, pages 154–164, New York, NY, USA. ACM.

[Muccini et al., 2005] Muccini, H., Dias, M. S., and Richardson, D. J. (2005). Reasoning about software architecture-based regression testing through a case study. In *COMPSAC'05: Proceedings of the 29th Annual International Computer Software and Applications Conference*, volume 02, pages 189–195, Los Alamitos, CA, USA. IEEE Computer Society.

[Myers, 2004] Myers, G. J. (2004). *The Art of Software Testing*. John Wiley & Sons, Inc., New York, NY, USA.

[Naur and Randell, 1969] Naur, P. and Randell, B., editors (1969). *Software Engineering: Report on a Conference Sponsored by NATO Science Committee (in October in Germisch, Germany)*. Scientific Affairs Division, NATO, Burssels, Belgium.

[Neto et al., 2007] Neto, A. C. D., Subramanyan, R., Vieira, M., and Travassos, G. H. (2007). Characterization of model-based software testing approaches. Technical report, PESC/COPPE/UFRJ, Siemens Corporate Research. Available `http://www.cos.ufrj.br/uploadfiles/1188491168.pdf`.

[Ng et al., 2004] Ng, S. P., Murnane, T., Reed, K., Grant, D., and Chen, T. Y. (2004). A preliminary survey on software testing practices in australia. In *ASWEC '04: Proceedings of the 2004 Australian Software Engineering Conference*, page 116, Washington, DC, USA. IEEE Computer Society.

[Ntafos, 2001] Ntafos, S. C. (2001). On comparisons of random, partition, and proportional partition testing. *IEEE Trans. Softw. Eng.*, 27(10):949–960.

[Nursimulu and Probert, 1995] Nursimulu, K. and Probert, R. L. (1995). Cause-effect graphing analysis and validation of requirements. In *CASCON '95: Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research*, page 46. IBM Press.

[OASIS, 2007] OASIS (2007). Web services business process execution language WS-BPEL version 2.0. OASIS Standard.

[O'Brien et al., 2005] O'Brien, L., Merson, P., and Bass, L. (2005). Quality attributes and service-orinted architectures. Technical report, Software Engineering Institute, SEI.

[O'Brien et al., 2007] O'Brien, L., Merson, P., and Bass, L. (2007). Quality attributes for service-oriented architectures. In *SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments*, page 3, Washington, DC, USA. IEEE Computer Society.

[Oktaba and Piattini, 2008] Oktaba, H. and Piattini, M. (2008). *Software Process Improvement for Small and Medium Enterprises Techniques and Case Studies*. Information Science Reference, IGI Global.

[OMG, 2006] OMG (2006). Business process modeling notation (BPMN) specification. Final Adopted Specification, February 2006. Object Management Group.

[Osterweil, 1987] Osterweil, L. (1987). Software processes are software too. In *ICSE '87: Proceedings of the 9th international conference on Software Engineering*, pages 2–13, Los Alamitos, CA, USA. IEEE Computer Society Press.

[Papazoglou et al., 2007] Papazoglou, M. P., Traverso, P., Dustdar, S., and Leymann, F. (2007). Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45.

[Paradkar, 1994] Paradkar, A. (1994). On the experience of using cause-effect graphs for software specification and test generation. In *CASCON '94: Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research*, page 51. IBM Press.

[Parveen and Tilley, 2008] Parveen, T. and Tilley, S. (2008). A research agenda for testing SOA-based systems. In *Proceeding of 2008 2nd Annual IEEE Systems Conference*, pages 1–6, Los Alamitos, CA, USA. IEEE Computer Society.

[Paul, 2001] Paul, R. (2001). End-to-end integration testing. In *APAQS '01: Proceedings of the Second Asia-Pacific Conference on Quality Software*, page 211, Washington, DC, USA. IEEE Computer Society.

[Paulk and Chrissis, 2002] Paulk, M. C. and Chrissis, M. B. (2002). The 2001 hight maturity workshop. Technical report, Software Engineering Institute, SEI. Available at `http://www.sei.cmu.edu/pub/documents/01.reports/pdf/01sr014.pdf`.

[Peng and Wallace, 1994] Peng, W. W. and Wallace, D. R. (1994). *Software Error Analysis*. Silicon Press, Summit, NJ, USA.

[Penta et al., 2007] Penta, M. D., Bruno, M., Esposito, G., Mazza, V., and Canfora, G. (2007). *Test and Analysis of Web Services*, chapter Web Services Regression Testing, pages 205–236. Springer.

[Perrow, 2008] Perrow, C. (2008). Software failures, security, and cyber attacks. In *SHB 2008: Interdisciplinary Workshop on Security and Human Behaviour*. Available at `http://www.cl.cam.ac.uk/~rja14/shb08/perrow.pdf`.

[Perry, 2006] Perry, W. E. (2006). *Effective methods for software testing*. Wiley Publishing Inc., Indianapolis, IN, USA, third edition.

[Pezzè and Young, 2007] Pezzè, M. and Young, M. (2007). *Software Testing and Analysis: Process, Principles, and Techniques*. John Wiley & Sons, Inc, Hobokon, NJ, USA.

[Pol et al., 2002] Pol, M., Teunissen, R., and van Veenendaal, E. (2002). *Software Testing-A Guide to the TMap Approach*. Addison-Wesley, New York, NY, USA.

[Poore, 2004] Poore, J. H. (2004). A tale of three disciplines...and a revolution. *Computer*, 37(1):30–36.

[Posavac and Carey, 2003] Posavac, E. J. and Carey, R. G. (2003). *Program Evaluation: methods and Case Studies*. Pearson Education, Inc., Upper Saddle River, NJ, USA.

[Pressman, 2001] Pressman, R. S. (2001). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education.

[Rajan, 2006] Rajan, A. (2006). Coverage metrics to measure adequacy of black-box test suites. In *ASE '06: Proceedings of the 21st IEEE International Conference on Automated Software Engineering*, pages 335–338, Washington, DC, USA. IEEE Computer Society.

[Ramler and Wolfmaier, 2006] Ramler, R. and Wolfmaier, K. (2006). Economic perspectives in test automation: balancing automated and manual testing with opportunity cost. In *AST '06: Proceedings of the 2006 international workshop on Automation of software test*, pages 85–91, New York, NY, USA. ACM Press.

[Ribarov et al., 2007] Ribarov, L., Manova, I., and Ilieva, S. (2007). Testing in a service-oriented world. In *InfoTech-2007: Proceedings of the International Conference on Information Technologies*.

[Richardson, 2002] Richardson, I. (2002). SPI models: What characteristics are required for small software development companies? *Software Quality Journal*, 10(2):101–114.

[Richter and Dumke, 2008] Richter, K. and Dumke, R. R. (2008). A causal-based approach for process improvement. *Software Measurement News-Journal of the Software Metrics Community*, 13(2):27–48.

[Rico, 2004] Rico, D. F. (2004). *ROI of Software Process Improvement: Metrics for Project Managers and Software Engineers*. J. Ross Publishing, Inc.

[Rossi et al., 1999] Rossi, P. H., Freeman, H. E., and Lipsey, M. W. (1999). *Evalution: A Systematic Approach*. Sage Publications, Thousand Oaks, CA, USA.

[Rothermel et al., 2004] Rothermel, G., Elbaum, S., Malishevsky, A. G., Kallakuri, P., and Qiu, X. (2004). On test suite composition and cost-effective regression testing. *ACM Trans. Softw. Eng. Methodol.*, 13(3):277–331.

[Rothermel et al., 2001] Rothermel, G., Untch, R. H., Chu, C., and Harrold, M. J. (2001). Prioritizing test cases for regression testing. *IEEE Trans. Softw. Eng.*, 27(10):929–948.

[Rud et al., 2007a] Rud, D., Kunz, M., Schmietendorf, A., and Dumke, R. R. (2007a). Performance analysis in WS-BPEL-based infrastructures. In *UKPEW 2007: Proceedings of 23rd Annual UK Performance Engineering Workshop*, pages 130–141.

[Rud et al., 2007b] Rud, D., Mencke, S., Schmietendorf, A., and Dumke, R. R. (2007b). Granularitätsmetriken für serviceorientierte architekturen. In *MetriKon 2007: Proceedings of the DASMA Software Metrik Kongress*, pages 297–308, Aachen, Germany. Shaker Verlag GmbH.

[Rud et al., 2006] Rud, D., Schmietendorf, A., and Dumke, R. R. (2006). Product metrics for service-oriented infrastructures. In *IWSM/MetriKon 2006: Proceedings of the International Workshop on Software Measurement and DASMA Software Metrik Kongress*, pages 161–174, Aachen, Germany. Shaker Verlag GmbH.

[Rud et al., 2007c] Rud, D., Schmietendorf, A., and Dumke, R. R. (2007c). Performance annotated business processes in service-oriented architectures. *International Journal of Simulation: Systems, Science & Technology*, 8(3):61–71.

[Rud et al., 2007d] Rud, D., Schmietendorf, A., and Dumke, R. R. (2007d). Resource metrics for service-oriented infrastructures. In *SEMSOA 2007: Workshop on Software Engineering Methods for Service Oriented Architecture*, pages 90–98.

[Rud et al., 2007e] Rud, D., Schmietendorf, A., Kunz, M., and Dumke, R. R. (2007e). Analyse verfügbarer SOA-reifegradmodelle-state-of-the-art. In *BSOA 2007: 2. Workshop Bewertungsaspekte serviceorientierter Architekturen*, pages 115–126, Aachen, Germany. Shaker Verlag GmbH.

[Rud et al., 2007f] Rud, D., Schmietendorf, A., Kunz, M., and Dumke, R. R. (2007f). Processqualität bei dem Übergang zur serviceorientierten architektur. In *MetriKon 2007: Proceedings of the DASMA Software Metrik Kongress*, pages 141–153, Aachen, Germany. Shaker Verlag GmbH.

[Ruth and Tu, 2008] Ruth, M. E. and Tu, S. (2008). Empirical studies of a decentralized regression test selection framework for web services. In *TAV-WEB '08: Proceedings of the 2008 workshop on Testing, analysis, and verification of web services and applications*, pages 8–14, New York, NY, USA. ACM.

[Saaty, 2000] Saaty, T. (2000). *Fundamentals of the Analytic Hierarchy Process*. RWS Publications, Pittsburgh, PA, USA.

[Saiedian and Chennupati, 1999] Saiedian, H. and Chennupati, K. (1999). Towards an evaluative framework for software process improvement models. *J. Syst. Softw.*, 47(2–3):139–148.

[Salvaneschi, 2005] Salvaneschi, P. (2005). The quality matrix: A management tool for software quality evaluation. In *SE 2005: Proceedings of the The IASTED International Conference on Software Engineering*, pages 394–399. ACTA Press.

[Satpathy et al., 2000] Satpathy, M., Harrison, R., Snook, C., and Butler, M. J. (2000). A generic model for assessing process quality. In *IWSM '00: Proceedings of the 10th International Workshop on New Approaches in Software Measurement*, pages 94–110, London, UK. Springer-Verlag.

[Sauer et al., 2000] Sauer, C., Jeffery, D. R., Land, L., and Yetton, P. (2000). The effectiveness of software development technical reviews: A behaviorally motivated program of research. *IEEE Trans. Softw. Eng.*, 26(1):1–14.

[Schmietendorf, 2007] Schmietendorf, A. (2007). *Eine strategische Vorgehensweise zur erfolgreichen Implementierung serviceorientierter Architekturen in großen IT-Organisationen*. Shaker Verlag.

[Schmietendorf, 2008] Schmietendorf, A. (2008). Assessment of business process modeling tools under consideration of business process management activities. In Dumke, R., Braungarten, R., Büren, G., Abran, A., and Cuadrado-Gallego, J., editors, *Software Process and Product Measurement*, volume 5338 of *Lecture Notes in Computer Science*, pages 141–154. Springer-Verlag Berlin/Heidelberg.

[Schmietendorf and Dimitrov, 2006] Schmietendorf, A. and Dimitrov, E. (2006). Management serviceorientierter architekturen auf der grundlage von ITIL. In *CECMG Jahrestagung 2006*.

[Schulmeyer, 2008] Schulmeyer, G. G., editor (2008). *Handbook of Software Quality Assurance (4th ed.)*. Artech House, Inc., Norwood, MA, USA.

[Schulmeyer and MacKenzie, 2000] Schulmeyer, G. G. and MacKenzie, G. R. (2000). *Verification and Validation of Modern Software Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Scriven, 1981] Scriven, M. (1981). The "weight and sum" methodology. *American Journal of Evaluation*, 2(1):85–90.

[Scriven, 1991] Scriven, M. (1991). *Evaluation Thesaurus*. Sage Publications Inc., Thousand Oaks, CA, USA, 4th edition.

[Scriven, 1996a] Scriven, M. (1996a). The theory behind practical evaluation. *Evaluation*, 2(4):393–404.

[Scriven, 1996b] Scriven, M. (1996b). Types of evaluation and types of evaluator. *American Journal of Evaluation*, 17(1):181–161.

[Shadish et al., 1991] Shadish, W. R., Cook, T. D., and Leviton, L. C. (1991). *Foundations of Program Evaluation: Theories of Practice*. Sage Publications, Inc., Newbury Park, CA, USA.

[Shaw, 2001] Shaw, M. (2001). The coming-of-age of software architecture research. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, page 656, Washington, DC, USA. IEEE Computer Society.

[Singpurwalla and Wilson, 1999] Singpurwalla, N. D. and Wilson, S. P. (1999). *Statistical Methods in Software Engineering: Reliability and Risk*. Springer.

[Sinha and Paradkar, 2006] Sinha, A. and Paradkar, A. (2006). Model-based functional conformance testing of web services operating on persistent data. In *TAV-WEB '06: Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications*, pages 17–22, New York, NY, USA. ACM.

[Siniaalto, 2006] Siniaalto, M. (2006). Test driven development: Empirical body of evidence. Technical report, ITEA, Information Technology for European Advancement. Available at `http://www.agile-itea.org/public/deliverables/ITEA-AGILE-D2.7_v1.0.pdf`.

[Sneed, 2004] Sneed, H. M. (2004). Reverse engineering of test cases for selective regression testing. In *CSMR'04: Proceedings of the Eigth Euromicro Working Conference on Software Maintenance and Reengineering*, page 69, Los Alamitos, CA, USA. IEEE Computer Society.

[Sneed, 2005] Sneed, H. M. (2005). Measuring the effectiveness of software testing: converting software testing from an art to a science. In *Proceedings of MetriKon 2005: DASMA Software Metrik Kongress*, pages 145–170, Aachen, Germany. Shaker Verlag GmbH.

[Sneed, 2007] Sneed, H. M. (2007). Test metrics. *Metrics News, Journal of GI-Interest Group on Software Metrics*, 12(1):41–51.

[Sommerville, 2007] Sommerville, I. (2007). *Software Engineering*. Pearson Education Limited, Harlow, England, 8th edition.

[Spillner et al., 2007] Spillner, A., Rossner, T., Winter, M., and Linz, T. (2007). *Software Testing Practice: Test Management*. Rocky Nook Inc., Santa Barbara, CA, USA.

[Staples et al., 2007] Staples, M., Niazi, M., Jeffery, R., Abrahamsd, A., Byatte, P., and Murphy, R. (2007). An exploratory study of why organizations do not adopt CMMI. *J. Syst. Softw.*, 80(6):883–895.

[Stikkel, 2006] Stikkel, G. (2006). Dynamic model for the system testing process. *Information and Software Technology*, 48(7):578–585.

[Stojanovic and Dahanayake, 2005] Stojanovic, Z. and Dahanayake, A. (2005). *Service-oriented Software System Engineering Challenges and Practices*. Idea Group Inc., Hershey, PA, USA.

[Stufflebeam and Shinkfield, 2007] Stufflebeam, D. and Shinkfield, A. J. (2007). *Evaluation Theory, Models, and Applications*. Jossey-Bass, San Francisco.

[Stufflebeam, 2001] Stufflebeam, D. L. (2001). Evaluation models. *New Directions for Evaluation*, 89.

[Suwannasart et al., 1999] Suwannasart, T., Prapass, and Srichaivattana (1999). A set of measurements to improve software testing process. In *NCSEC'99: Proceedings of the 3rd National Computer Science and Engineering Conference*.

[Swinkels, 2000] Swinkels, R. (2000). A comparison of TMM and other test process improvement models. Technical report, Frits Philips Institute, Technische Universiteit Eindhoven, Netherlands. Available at `http://is.tm.tue.nl/research/v2m2/wp1/12-4-1-FPdef.pdf`.

[Taipale, 2007] Taipale, O. (2007). *Observations On Software Testing Practice*. PhD thesis, Lappeenranta University of Technology, Lappeenranta, Finland,.

[Taipale et al., 2005] Taipale, O., Smolander, K., and Kälviäinen, H. (2005). Finding and ranking research directions for software testing. In Richardson, I., Abrahamsson, P., and Messnarz, R., editors, *Software Process Improvement*, volume 3792 of *Lecture Notes in Computer Science*, pages 39–48. Springer-Verlag Berlin/Heidelberg.

[Tarhan and Demirörs, 2008] Tarhan, A. and Demirörs, O. (2008). Assessment of software process and metrics to support quantitative understanding. In Cuadrado-Gallego, J., Braungarten, R., Dumke, R., and Abran, A., editors, *Software Process and Product Measurement*, volume 4895 of *Lecture Notes in Computer Science*, pages 102–113. Springer-Verlag Berlin/Heidelberg.

[THBS, 2007] THBS (2007). SOA test methodology. Torry Harris Business Solutions. Available at `http://www.thbs.com/pdfs/SOA_Test_Methodology.pdf`.

[Tian, 2005] Tian, J. (2005). *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*. Wiley-IEEE Computer Society Pres, Los Alamitos, CA, U.S.A.

[Tillmann and Schulte, 2005] Tillmann, N. and Schulte, W. (2005). Parameterized unit tests. In *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 253–262, New York, NY, USA. ACM Press.

[Trudel et al., 2006] Trudel, S., Lavoie, J.-M., Paré, M.-C., and Suryn, W. (2006). PEM: The small company-dedicated software process quality evaluation method combining CMMI and ISO/IEC 14598. *Software Quality Journal*, 14(1):7–23.

[Tyrrell, 2000] Tyrrell, S. (2000). The many dimensions of the software process. *Crossroads*, 6(4):22–26.

[Utting and Legeard, 2006] Utting, M. and Legeard, B. (2006). *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[van Lamsweerde, 2000] van Lamsweerde, A. (2000). Formal specification: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 147–159, New York, NY, USA. ACM Press.

[van Solingen and Berghout, 1999] van Solingen and Berghout, E. (1999). *The Goal/Question/-Metric Method: A Practical Guide for Quality Improvement of Software Development*. McGraw-Hill, London, UK.

[van Solingen, 2000] van Solingen, R. (2000). *Product Focused Software Process Improvement: SPI in the Embedded Software Domain*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

[van Veenendaal and Pol, 1997] van Veenendaal, E. and Pol, M. (1997). A test management approach for structured testing. *Achieving Software Product Quality*.

[Vanderfeesten et al., 2007] Vanderfeesten, I. T. P., Cardoso, J., and Reijers, H. A. (2007). A weighted coupling metric for business process models. In Eder, J., Tomassen, S. L., Opdahl, A. L., and Sindre, G., editors, *Proceedings of the CAiSE'07 Forum at the 19th International Conference on Advanced Information Systems*, pages 41–44.

[Verma et al., 2005] Verma, S., Ramineni, K., and Harris, I. G. (2005). An efficient control-oriented coverage metric. In *ASP-DAC '05: Proceedings of the 2005 conference on Asia South Pacific design automation*, pages 317–322, New York, NY, USA. ACM Press.

[Vessey et al., 2005] Vessey, I., Ramesh, V., and Glass, R. L. (2005). A unified classification system for research in the computing disciplines. *Information & Software Technology*, 47(4):245–255.

[Wang et al., 1998] Wang, L.-C., Abadir, M. S., and Zeng, J. (1998). On measuring the effectiveness of various design validation approaches for powerpc microprocessor embedded arrays. *ACM Trans. Des. Autom. Electron. Syst.*, 3(4):524–532.

[Wang, 2008] Wang, Y. (2008). *Software Engineering Foundations : A Software Science Perspective*. Auerbach Publications, Boca Raton, FL, USA.

[Wang and King, 2000] Wang, Y. and King, G. (2000). *Software engineering processes: principles and applications*. CRC Press, Inc., Boca Raton, FL, USA.

[Weber-Wulff, 2005] Weber-Wulff, D. (2005). Two german projects: Toll and dole. *The Risks Digest*, 23(65).

[Wegner, 1976] Wegner, P. (1976). Research paradigms in computer science. In *Proceedings of Int'l Conf. Software Engineering*, pages 322–330.

[Weiss, 1998] Weiss, C. H. (1998). *Evaluation*. Prentice Hall, Upper Saddle River, NJ, USA, second edition.

[Weske, 2007] Weske, M. (2007). *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[WfMC, 2008] WfMC (2008). Process definition interface – XML process definition language version 2.1. Workflow Management Coalition Standard.

[Whalen et al., 2006] Whalen, M. W., Rajan, A., Heimdahl, M. P., and Miller, S. P. (2006). Coverage metrics for requirements-based testing. In *ISSTA '06: Proceedings of the 2006 international symposium on Software testing and analysis*, pages 25–36, New York, NY, USA. ACM Press.

[Wille et al., 2004] Wille, C., Dumke, R. R., and Brehmer, N. (2004). Evaluation of the agent academy: Measurement intentions and results. In *IWSM/MetriKon 2004: Proceedings of the International Workshop on Software Metrics and DASMA Software Metrik Kongress*, pages 309–319, Aachen, Germany. Shaker Verlag GmbH.

[Williams et al., 2004] Williams, L., Layman, L., and Krebs, W. (2004). Extreme programming evaluation framework for object-oriented languages version 1.4. Technical report, IBM Corporation and North Carolina State University, USA. Available at `ftp://ftp.ncsu.edu/pub/unity/lockers/ftp/csc_anon/tech/2004/TR-2004-18.pdf`.

[Woolf, 2008] Woolf, B. (2008). *Exploring IBM SOA Technology & Practice, How to Plan, Build and Manage a Service Oriented Architecture in the Real World*. Maximum Press., FL, USA.

[Wu et al., 2005] Wu, Y. P., Hu, Q. P., Ng, S. H., and Xie, M. (2005). Bayesian networks modeling for software inspection effectiveness. In *PRDC '05: Proceedings of the 11th Pacific Rim International Symposium on Dependable Computing*, pages 65–74, Washington, DC, USA. IEEE Computer Society.

[Zahran, 1998] Zahran, S. (1998). *Software process improvement: practical guidelines for business susccess*. Addison-Wesley Longman Ltd., Essex, UK, UK.

[Zamli, 2004] Zamli, K. Z. (2004). A survey and analysis of process modeling languages. *Malaysian Journal of Computer Science*, 17(2):68–89.

[Zarour et al., 2007] Zarour, M., Desharnais, J.-M., and Abran, A. (2007). A framework to compare software process assessment methods dedicated to small and very small organizations. In *ICSQ'07: Proceedings of International Conference on the Software Quality*. Available at `http://www.lrgl.uqam.ca/publications/pdf/1095.pdf`.

[Zelkowitz and Wallace, 1997] Zelkowitz, M. and Wallace, D. (1997). Experimental validation in software engineering. *Information and Software Technology*, 39(1):735–743.

[Zenker et al., 2007] Zenker, N., Kunz, M., and Rautenstrauch, C. (2007). Service oriented architectures: Resource based evaluation of a SOA. In Schmietendorf, A., Mevius, M., and Dumke, R. R., editors, *BSOA 2007: 2 Workshop Bewertungsaspekte serviceorientierter Architecturen*, pages 23–32. Shaker Verlag.

[Zuse, 1998] Zuse, H. (1998). *A Framework of Software Measurement*. Walter de Gruyter & Co., Berlin, Germany.

# Statement of Autonomy

I hereby declare that the this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the reward of any other degree or diploma at the University of Magdeburg or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at the University of Magdeburg or elsewhere, has been explicitly acknowledged in this thesis.

I also declare that the intellectual content of this thesis is the product of own work, except to the extent that assistance from others in the project's design and conception or in style, presentation, and linguistic expression is acknowledged.

Magdeburg, June 16, 2009

Ayaz Farooq