EMPIRICAL CHARACTERIZATION OF SOFTWARE QUALITY

Dipartimento di Automatica e Informatica
Politecnico di Torino
Corso Duca degli Abruzzi, 24 - 10129 Torino,
ITALY

Ph.D. Dissertation of:
Syed Muhammad Ali Shah

Advisor:
Prof. Maurizio Morisio

MARCH 2014

ABSTRACT

The research topic focuses on the characterization of software quality considering the main software elements such as people, process and product. Many attributes (size, language, testing techniques etc.) probably could have an effect on the quality of software. In this thesis we aim to understand the impact of attributes of three P's (people, product, process) on the quality of software by empirical means. Software quality can be interpreted in many ways, such as customer satisfaction, stability and defects etc. In this thesis we adopt 'defect density' as a quality measure. Therefore the research focus on the empirical evidences of the impact of attributes of the three P's on the software defect density. For this reason empirical research methods (systematic literature reviews, case studies, and interviews) are utilized to collect empirical evidence. Each of this research method helps to extract the empirical evidences of the object under study and for data analysis statistical methods are used. Considering the product attributes, we have studied the size, language, development mode, age, complexity, module structure, module dependency, and module quality and their impact on project quality. Considering the process attributes, we have studied the process maturity and structure, and their impact on the project quality. Considering the people attributes, we have studied the experience and capability, and their impact on the project quality. Moreover, in the process category, we have studied the impact of one testing approach called 'exploratory testing' and its impact on the quality of software. Exploratory testing is a widely used software-testing practice and means simultaneous learning, test design, and test execution. We have analyzed the exploratory testing weaknesses, and proposed a hybrid testing approach in an attempt to improve the quality.

Concerning the product attributes, we found that there exist a significant difference of quality between open and close source projects, java and C projects, and large and small projects. Very small and defect free modules have impact on the software quality. Different complexity metrics have different impact on the software quality considering the size. Product complexity as defined in Table 53 has partial impact on the software quality. However software age and module dependencies are not factor to characterize the software quality.

Concerning the people attributes, we found that platform experience, application experience and language and tool experience have significant impact on the software quality. Regarding the capability we found that programmer capability has partial impact on the software quality where as analyst capability has no impact on the software quality.

Concerning process attributes we found that there is no difference of quality between the project developed under CMMI and those that are not developed under CMMI. Regarding the CMMI levels there is difference of software quality particularly between CMMI level 1 and CMMI level 3. Comparing different process types we found that hybrid projects are of better quality than waterfall projects. Process maturity defined by (SEI-CMM) has partial impact on the software quality. Concerning exploratory testing, we found that exploratory testing weaknesses induce the testing technical debt therefore a process is defined in conjunction with the scripted testing in an attempt to reduce the associated technical debt of exploratory testing.

The findings are useful for both researchers and practitioners to evaluate their projects

ACKNOWLEDGMENTS

For this research thesis, I am fortunate to have Prof. Maurizio Morisio, who not only gave me the opportunity to undertake the doctoral studies but also supervised me, without the invaluable guidance, supervision and support this thesis would not have happened. Thank you very much Prof. Maurizio Morisio. I still need your supervision and a lot to learn from you. I am thankful to Dr. Marco Torchiano, who is always willing to discuss and guide me, his ideas has been very influential in conducting this research.

I am thankful to my colleagues working at SoftEng for sharing their knowledge with me and offering me to learn from them by discussing the ideas and problem-solution formation. I really appreciate all my colleagues that took part in the internal review of my research work and giving me positive feedback and suggestions to improve my research.

I am thankful to SoftEng body and specifically Prof. Maurizio Morisio who arranged funding for my research work. I think without it, I was not able to perform this research.

I will remain beholden to my parents; they provided me great encouragement, confidence and comfort to undertake my doctoral studies overseas. I thank my brother and sister for their support and love they put during my stay overseas. I give special thanks to my nieces and nephews for providing me ultimate joy in my life.

From every aspect I am thankful to my host country “ITALY”. The years I have spent here in this beautiful city of beautiful country (Torino, Italy) for my studies make me feel it like home. I spent my best time enjoying the great Italian hospitality.

Table of Contents

List of Figures

List of Table

Chapter 1

1. Introduction

The main goal of software engineering research is to provide the evidences that support the practitioners and facilitate them to take correct decision during the software development [1]. These decisions are always dependent on how the data is analyzed and which information is extracted from the data after analysis. This information is used to support the investigation of different themes in this thesis and provides assistance for decision taking. Software is an entity that is dependent on many direct and indirect attributes used in its development. For example, process that is used to develop software is one attribute that software is dependent on. However during the development of software the most important aspect considered is the resultant quality. Consequently it is tried to consider only those attributes that have a relatively better impact on the software quality. The common method to reveal quality of any software is to know the number of defects in it.

1.1 Software elements

A conceptual model of software engineering consist of three elements that are often called software elements and usually denoted by 3Ps i.e. people, process and product. People are involved to carry out the engineering process to produce a software product.

1.1.1 People

People are the primary element and main force of software development. People are involved in all phases of software development e.g. People gather requirements, people interview users (people), people design software, and people write software for people. No people -- no software. The best thing that can happen to any software project is to have people who know what they are doing and have the experience and capability to do it [2]. Concerning people attributes such as experience and capability should have significant impact on the quality.

1.1.2 Process

Process is how we go from the beginning to the end of a project. All projects use a process and selection of process for the project depends on the projects context. Process can be assessed under two dimensions level of maturity and type. Similarly process could be in either dimension level of maturity or type. The level of maturity is expressed in software engineering Institute's Capability Maturity Model (SEI- CMM) and Capability Maturity Model Integration (CMMI) levels and process types are expressed in terms of Waterfall and Agile, etc. Considering these attributes of process, they should have impact on the quality.

1.1.3 Product

The product is the result of a project. The desired product satisfies the customers and fulfills their requirements. So what we are constructing using people and process is a 'Product'. The more emphasis on process and people sometimes causes us to forget the product. This results in a poor product, no money, no more business, and no more need for people and process [2]. There are many attributes of products that should be considered important in order to have a quality product. The product attributes actually are the attributes of software itself e.g. development language, the size, the complexity level of software etc. These attributes should have direct or indirect impact on the quality.

1.1.4 Software quality

Quality has many definitions and often it depends on the context. The most understood definitions by some of the international organizations are as follows.

- The "German Industry Standards DIN 55350 Part 11" defines the quality as "*Quality comprises all characteristics and significant features of a product or an activity which relate to the satisfying of given requirements*".
- The quality defined by 'ANSI Standard ANSI/ASQC A3/1978" is "*Quality is the totality of features and characteristics of a product or a service that bears on its ability to satisfy the given needs*".
- The IEEE Standard (IEEE Std 729-1983) defines the quality as:
    - "*The totality of features and characteristics of a software product that bear on its ability to satisfy given needs: for example, conform to specifications*"
    - "*The degree to which software possesses a desired combination of attributes*"
    - "*The degree to which a customer or user perceives that software meets his or her composite expectations*"
    - "*The composite characteristics of software that determine the degree to which the software in use will meet the expectations of the customer*".
- The Pressman's [3] defines the software quality in term of "*Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software*"
- IEEE Definition of Software Quality in term of customer satisfaction "The degree to which a system, component, or process meets specified requirements"
- IEEE Definition of "Software Quality" in term of Requirements fulfillment "The degree to which a system, component, or process meets customer or user needs or expectations"

There are many definitions regarding the quality, specifically software quality. Besides all the definitions the quality, it is so often depends on the context in which it is required. Hence, in this thesis we have used the quality measure Defect Density (DD) that is usually defined as the number of defects found divided per size. Subsequently the main theme of this research is to understand the impact of people, process and product attributes on the quality defined in term of defect density. Figure 1 shows the general idea of the research conducted in this study. The more detail of the defect density considering its variants and understandings it as an indicator of global quality view of a project and local quality view indicator view is given in Chapter 2.

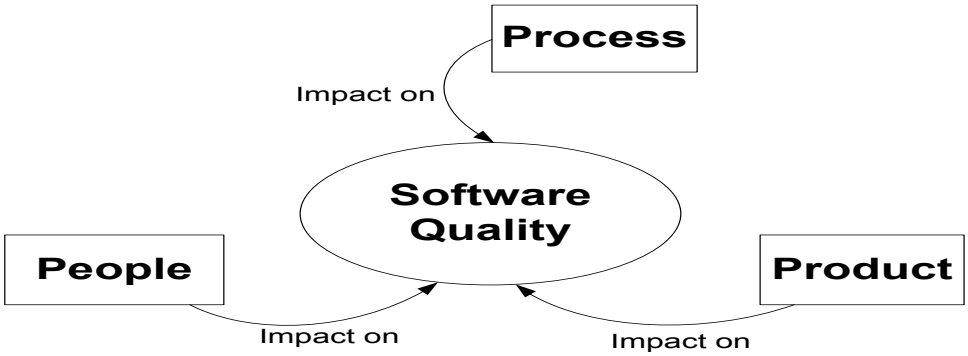

Figure 1. The general idea of the research

1.1.5 Empirical software engineering

Empirical software engineering is a field of quantitative research methods for measuring, assessing, predicting, controlling, and managing software development [4]. It focuses the research on confirming theories e.g. "conventional wisdom", Object Oriented is better or not., exploring relationships e.g. relationship between quality and productivity, evaluating accuracy of models e.g. project quality models and validating measures e.g. code complexity, size etc.

The practical purpose of empirical research is the assessment, evaluation management, prediction and development of software project artifacts. The fundamental elements of empirical studies are research design, measurements, and analysis. The empirical studies are classified into three modes, descriptive, exploratory and confirmatory. The research method types are experiments, case studies and statistical analysis. The level of analysis could vary from individual to team to project and to organization.

The basic requirement for empirical studies is to formulate the hypothesis that should be unambiguous, testable and quantifiable. The definition of dependent (effects) and independent (causes), and the measurement of variables in an efficient, reliable and valid manner [4].

1.2 Software engineering data sets

The data sets are the basic building blocks of empirical studies on which the analysis are performed and results are extracted. The data sets used in this thesis comes from industry or open source. The data set collected and analyzed in this thesis has been used in some previous studies with some other objectives and considered validated. For the industrial projects consultant companies are contacted that helped in providing their data sets e.g. Capers Jones. For the open source projects we collected the defects related data from the portal of apache issues and also we made use of some research repositories such as PROMISE [5]. The important details of the used data sets are given in detail in the chapters of this thesis. Table 1 shows the outline of data set used in this thesis.

In chapter 2 the data set is composed of number of defects and size of releases of three open source projects.

In chapter 3 the data set is from different resources (Systematic Literature Review, Promise, NASA 93) representing many of the product attributes against the defect density of projects.

In chapter 4 the data set is from the industry (NASA 93) comprising of the people attributes against the defect density of projects.

In chapter 5 the data set is from the industry (Capers Jones, NASA 93) comprising of different process attributes against the defects density of projects.

Table 1 the data set used in this thesis

CHAP	Data set name / Description	Sources	Context
2	Camel, Lucene, Whirr	Apache Software Foundation	Large Open Source Projects
3	Project A, Project B, Project C, Project D, ACE, Ant (1.7.0), Apache web server 1.0, Argo UML, Avaya telephony systems, camel, CDK (1.0.1), Ckjm, CMI, Controller, DCF Release 1, DCF Release 2, DCF Release 3, Eclipse, Eclipse 2.0, Eclipse 2.1, Eclipse 3.0, eXpert, FOP (0.94), Forrest, FreeBSD, Freenet (0.7), IBM, Inventory, Ivy, jedit, JEF framework, JEF framework, JEF framework, Jetspeed2 (2.1.2), JM1, Jmol (11.2), KC1, KC3, KC4, Large, Linux kernel Verison 2.4, Linux kernel Version 2.6, log4j, lucene, MC1, MC2, OsCache (2.4.1), pbean, PC2, PC3, PC4, PCI, Pentaho (1.6.0), PL/I data base application, poi, Project 12, Project 28, Project 1, Project 10, Project 11, Project 13, Project 14, Project 15, Project 16, Project 17, Project 18, Project 19, Project 2, Project 20, Project 21, Project 22, Project 23, Project 24, Project 25, Project 26, Project 27, Project 29, Project 3, Project 30, Project 31, Project 4, Project 5, Project 6, Project 7, Project 8, Project 9, Prop 1, Prop 2, Prop 3, Prop 4, Prop 5, Provisioning, Public, Reusable, Rhino, synapse, System A, System A, System B, System C, System C, System D, System E, Tele comm. Sub system ,Telecom System of Ericsson, Tomcat, TV-Browser (2.6), velocity, Xalan, xerces	Electronic Data Bases by Performing Literature Review	Open Source and Industrial projects
	Ckjm, sybkofucha, e learning, kalkulator, workflow, nieruchomosci, wspomaganiepi, Pdf Translator, forrest0.8, Termoproject, sklebagd, Serapion, interface, zuzel 1, Skarbonka, pbean2, systemdata, velocity 1.6, arc, berek, log4j 1.2, synapse 1.2, Redaktor , Ivy 2.0, lucene 2.4, camel1.6, poi 3.0, xerces1.4, jedit4.3, Ant 1.7, Tomcat 6.0.389418, xalan 2.7, Prop 1, Prop 2, Prop 3, Prop 4, Prop 5 and Prop 6.	Promise Data Set	Students, Open Source and Industrial Projects.
	Project 1 to Project 93	NASA 93 Data Set	Industrial Projects from different domains e.g. avionics, mission, simulation, monitoring etc.
4	Project 1 to Project 93	Nasa 93 Data Set	Industrial Projects from different domains e.g.

			avionics, mission, simulation, monitoring etc.
5	Project 1 to Project 60	Caper Jones (industrial data set)	Large Industrial Projects Large
	Qualitative data about Exploratory Testing	Interview, questionnaires Literature review,	Experience based data

1.3 Research questions

The aim and goal of a research question is to draw the main attention behind the investigation [6]. Therefore in every research project the research focus is mostly highlighted by the use of particular research questions [6].

The objective of this thesis is to understand and highlight the impact of different attributes of three main elements of software engineering i.e. 3Ps (product, people, and process) on the projects quality.

Figure 2 shows the research questions and how these research questions are addressed by research papers in relevant chapters of this thesis.

Main Research Question: **What is the impact of software elements on the software quality**?

To answer the main research question, different chapters are formed and in which several research questions are addressed to be answered explicitly.

RQ1: What is the suitable measure of quality in term of defect density?

The answer of RQ1 is to be found in Chapter 2. RQ1 is answered by performing the analysis of different variants of DD. Two concrete variants of Defect Density (standard DD, differential DD) are defined, and analyzed their trend over time on a number of projects, and understand which one is more suitable as an indicator of the quality of software projects.

RQ2: What is the impact of product attributes on the quality?

The answer of RQ2 is given in the Chapter 3. Different research data sets have been used to answer the RQ3 by mean of statistical hypothesis testing i.e. either the product attribute have impact on the quality or not. The first step was to collect the data about different attributes of product; thereafter the statistical analysis is performed to understand the influence of any attribute on the quality.

RQ3: What is the impact of people attribute on the quality?

The answer of RQ3 is given in the Chapter 4 using the data set of 'NASA 93'. To see the impact of people attributes 93 projects from different center of NASA are analyzed to understand the impact of "experience" and "capability" impact on the quality.

RQ4: What is the impact of process attributes on the quality?

Chapter 5 provides the answer of RQ 4. Two different data sets are used to analyze the impact of different process attributes on the quality. The 'NASA 93' data set is used to understand the impact of CMM on the quality and the data set that comes from the industrial partner 'Capers Jones' that are 60 large size projects are used to understand the impact of level of maturity (as measured by the CMMI assessment model) and type (TSP, RUP and the like) on the process quality. In addition the test approach Exploratory Testing (ET) is analyzed to understand its impact on the quality. We

analyzed the software testing approach through a systematic review of literature to understand the consequences of ET on the quality. Afterwards we define a hybrid process in conjunction with scripted and exploratory testing in an attempt to reduce the consequences of ET on quality.

- Syed Muhammad Ali Shah is the primary author of all papers.
- Prof. Maurizio Morisio, who is principal supervisor of this thesis, and is a co-author on all the papers included in the thesis except paper 4 Chapter 5.
- Dr Marco Torchiano is a co author on the papers: paper 1 in Chapter 2, paper 1 and 3 in Chapter 3, paper 1 in Chapter 4, and paper 1, 2 and 3 in Chapter 5.

Paper 3 in Chapter 5, is written with Dr. Antonio Vetro' who contributed on the part of technical debt.

Paper 4 in Chapter 5 is written together with Dr. Cigdem Gencel, Engr. Usman Sattar Alvi and Dr. Kai Petersen, who are also involved in the main idea, in the formation of hybrid process definition and in the static evaluation as well in discussion and writing of the paper.

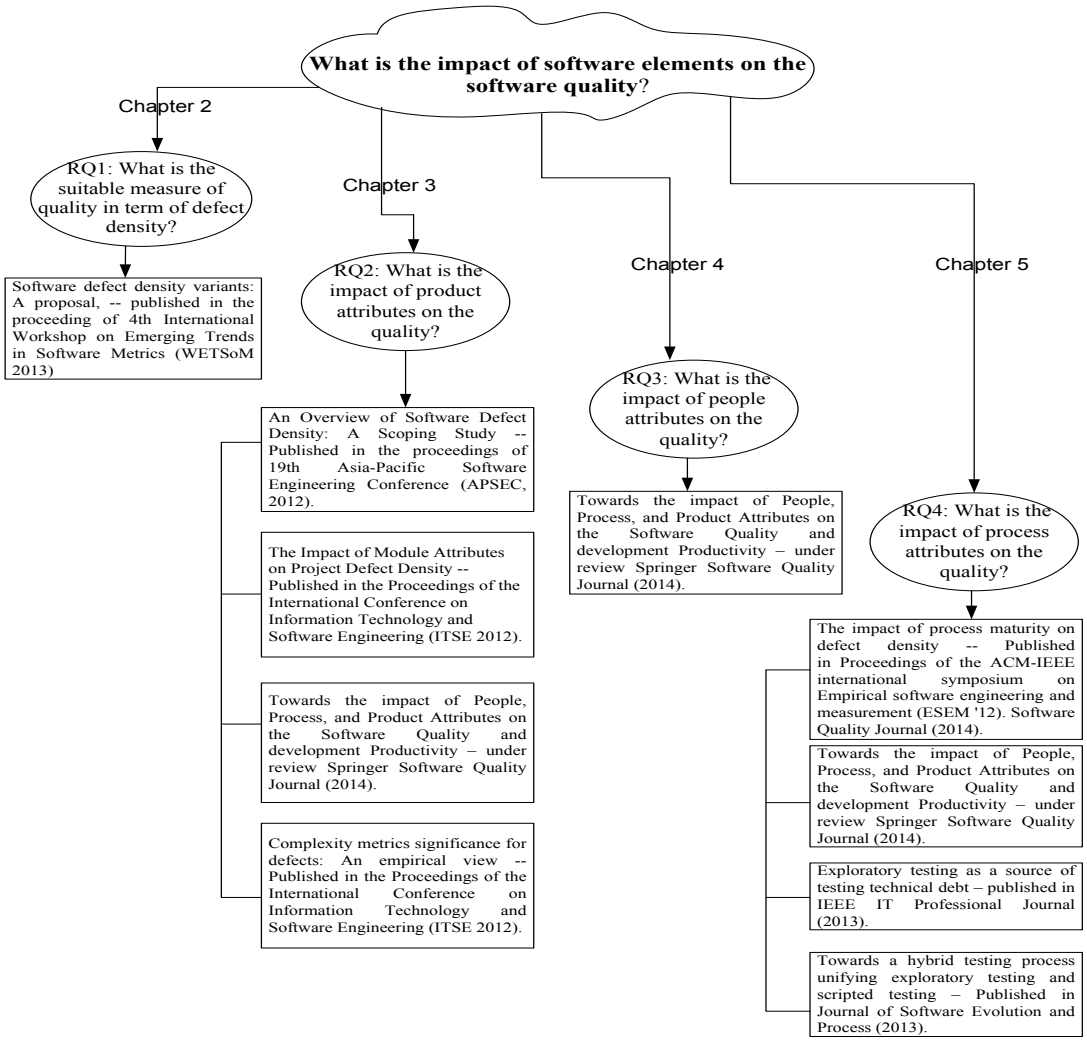

Figure 2 Organization of research questions addressed by research papers in relevant chapters

1.4 Research methodology

The research is defined as a study that goes beyond the influences of personal experience of an individual and it is based on the utilization of some research methods and techniques. Creswell describes three types of methods used for research i.e. Qualitative, Quantitative and Mixed research [6].

The qualitative method of research is relayed on the theory of human perspectives [6] and has different ways of interpretations [7]. Some of the strategies of the qualitative method of research are grounded theory, case study, interviews and ethnography etc.

The quantitative method of research is mainly concerned with quantifying a relationship, comparing two or more groups, use of measurements and observations, hypothesis testing and investigating cause and effect relation [6]. Some of the strategies of the quantitative research are experiments, surveys etc.

The mixed method of research contains both quantitative and qualitative methods in a single study.

The research methodology used in this thesis is both quantitative and qualitative. The research methods used in Chapter 2, 3, 4 are of quantitative nature. As in these chapters first the data is extracted from different source and then statistical analysis is performed to quantify the relationship, comparing groups, performing hypothesis testing and investigating cause and effect relation. In the Chapter 5 the part related to process level of maturity and type is addressed by the quantitative research method, however the exploratory testing related part is addressed by mixed method approach.

1.5 Summary

In this introduction chapter we presented the outline of the research area that is conducted in this thesis. In addition to that we discussed the concepts, research methods, research questions, the data set used in the thesis.

Chapter 2

2. Software defect density variants: A proposal

(Understanding of appropriate software quality measure in term of defect density)

Published at (WETSoM, 13)
Emerging Trends in Software Metrics
S. M. A. Shah, M. Morisio and M. Torchiano

2.1 Introduction

Defect density (DD) is one of the most established measures of software quality. Typically DD is defined as the number of defects found divided by the size in a thousand lines of code. This definition is mostly used among practitioners to calculate and evaluate the quality of their projects at a certain phase of development. It is often used as an indicator of release readiness [8]. In addition the DD is also used to compare subsequent releases of a product to track the impact of defect reduction and quality improvement activities. Hence DD is a popular measure for comparing products [9]. However if we consider a project over a set period of time, what happens to DD? Usually, over subsequent releases code is added, and the same applies for defects. This paper investigates the consequences of this on DD and whether it remains stable, increase or decreases. In other cases code is deleted over releases. This leads us to examine the trend of DD in these cases and whether it is more meaningful to consider all defects in a project, or only the new ones introduced between two releases. The same for size is it better to consider the total size, or only delta between two releases.

This chapter offers an in-depth analysis of different variants of DD, applying them to some sample projects and trying to understand which DD variant is more suitable for the quality evaluation of a project.

2.2 Related work

From literature we found different researchers using different definitions of DD. In the authors recent overview study of defect density they used the cumulative defects of all releases and the size of the last release to define the defect density [10]. They argument that in the meantime the code base may undergo complex transformations e.g. code additions, changes, deletions. Therefore it is difficult to match a defect to the corresponding code base.

In another recent study Zhu and Faller [8] assessed defect density in evolutionary product development. They use aggregated churned LOC as a size measure for calculating the defect density. They argue that for the same code repository, the number of defects of release R_i developed in time period T can be approximated by defects reported in time period T, regardless if those defects come from release R_i or R_{i-1} or any previous one.

Westfall analyzed DD of the releases of a software project over time. For every release she has used the total size of the release (reused code and new code) [9].

In addition Mohagheghi et al. studied a large distributed system by Ericsson. They compared the DD of the system considering the re-used components and non reused components. They found that reused components have lower DD than the non reused components [11].

Kim et al. studied the use of defect density on different phases and artifacts of a software project for quality control activities. They used error defect density, document defect density and delivered defect density for quality controlled activities [12]. In another study done by authors they used the delivered defect density looking for a relationship between process structure and quality of the product. To calculate DD they used the number of defects that are shipped with the product and the total size of the project [13].

2.3 Defect density variants

Since many variants of DD can be used, we give here the precise definition of two of them. Figure 3 shows the general trend of defects and size in the subsequent releases of a project. Most often every release adds both size and defects. In Figure 3 SRi denotes the size of a project at the time of release Ri, and DRi denotes the defects found until release Ri. Where ΔSRi and ΔDRi are the size and defects added between Ri and Ri-1.

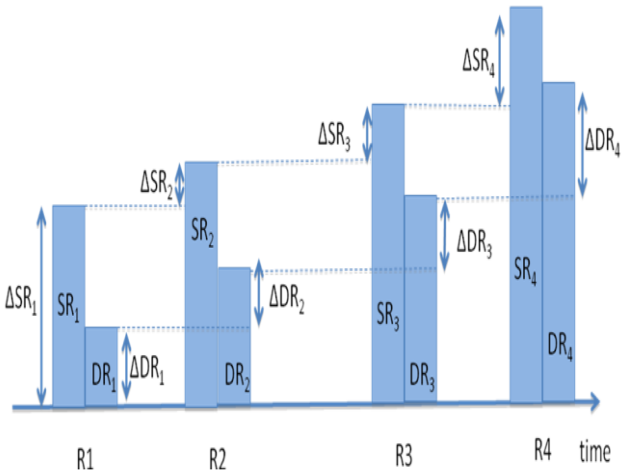

Figure 3 Defects and size in releases of a project

2.3.1 Standard defect density (sDD)

This variant corresponds to the usual and standard definition, that considers a project as a whole at a certain time (release), as used in the study [10].

$$sDD_i = DR_i / SR_i$$

To calculate the subsequent release defect density the cumulative defects of prior releases are used with current release size. For example to calculate the DD of R_{i+2}, all defects from release R_i, R_{i+1} and R_{i+2} are used, where the size is taken at the time of release R_{i+2}.

2.3.2 Differential defect density (dDD)

Standard DD does not distinguish defects belonging to different releases. To overcome this problem we define differential DD. Differential DD considers each release as a new project and is defined as follows:

$$dDD_i = (DR_i - DR_{i-1}) / |SR_i - SR_{i-1}|$$

At the denominator we have the absolute value of the difference in size between two releases. The absolute value is used in case the size decreases, since we are interested in the absolute variation in size. And anyway a negative value of dDD is not acceptable since defects always increase. At the numerator we have, ideally, the defects belonging only to the last release. This is not the case in practice, because some defects will be found later in time, after the release is issued, and other defects belong to previous releases. So as a proxy we have at the numerator $DR_i - DR_{i-1}$

Differential Defect Density at First Release: At the first release we compute dDD_1 using DR_1 and SR_1, as there is no delta *defects* and size for the first release.

2.4 Research design

In this section, we present the research questions, the projects examined and the analysis method.

2.4.1 Research questions

The research questions are:

RQ1: What is the trend of DD variants over a project? Is there any difference?

RQ2: What DD variant is more suitable as an indicator of software quality?

If the DD variants have the same trend and are highly correlated no variants of DD are needed. However if we find a different trend of DD variants we would evaluate which DD variant is more suitable as an indicator of quality.

2.4.2 Studied projects

To evaluate the RQs we need software projects with available size and defect data from the start. Therefore we selected three open source software projects from the "Apache Software foundation" for which we were able to find the total size and defects for every release.

Apache Camel is a versatile open-source integration framework based on known enterprise integration patterns. We analyzed Camel project component 'Camel-Core' that includes forty releases. The releases span over a time period of about five years.

Apache Lucene is a text search engine library written entirely in Java. We analyzed twenty four releases of the Lucene project component 'Lucene-Core' starting from release R2.3.0 to R4.0.0 that span over a time period of more than four years. The first release of Lucene was R1.9, we were unable to find the size of Lucene releases from R1.9 to R2.2. There are total five releases (R1.9, R1.9.1, R2.0.0, R2.1, R2.2) prior to the release R2.3.0, which is the first undertaken release for this

study. However to start and perform our analysis from release R2.3.0 we used the cumulative defects prior to release R2.3.0.

Apache Whirr is a set of libraries for running cloud services. We analyzed nine releases of Whirr project component 'Whirr-Core' that span over a period of approximately two years.

2.4.3 Analysis methods

For the data analysis we used graphical representation (DD values over time) as it gives the immediate comparison. DD values are extremely skewed and required logarithm scale to have a discernible representation.

2.5 Analysis

In this section we analyze the projects over their life time to understand the possible answer of formulated research question for this study.

2.5.1 Project camel

In the Camel project (see Figure 4 and Table 2) we observed two versions v1 and v2. The shifting of version v1 to version v2 is the results of some major changes in the program using the same code base. The v1 consists of 11 releases where v2 consists of 29 releases. We observe the increase of size of over eight times from the first release of v1 (17618 loc) to the last release (145516 loc) of v2. This means that in every release of Camel project there is a fair amount of addition of lines of code.

The range of sDD span from 0.06 to 5.06 defects per thousand lines of code, where the dDD span from 0.05-102.04 defects per thousand lines of code. The average sDD for forty releases is found to be 3.43 where the average dDD of release is found to be 19.88 defects per thousand lines of code. Figure 4 shows the sDD and dDD plot over the project lifetime.

We observe the increasing trend of the sDD over the releases in an entire Camel project lifetime. However considering the dDD plot over the project lifetime, it has high variability, having defect density of 0.06 to above 102.4 defects per thousand lines of code. The very high dDD values are due to the smaller addition of lines of code at any release R_i to compose R_{i+1} compared to high delta defects.

In summary, for the project Camel we observe the continuous and stable trend of increase in sDD over time. Where dDD has high variability over the project lifetime.

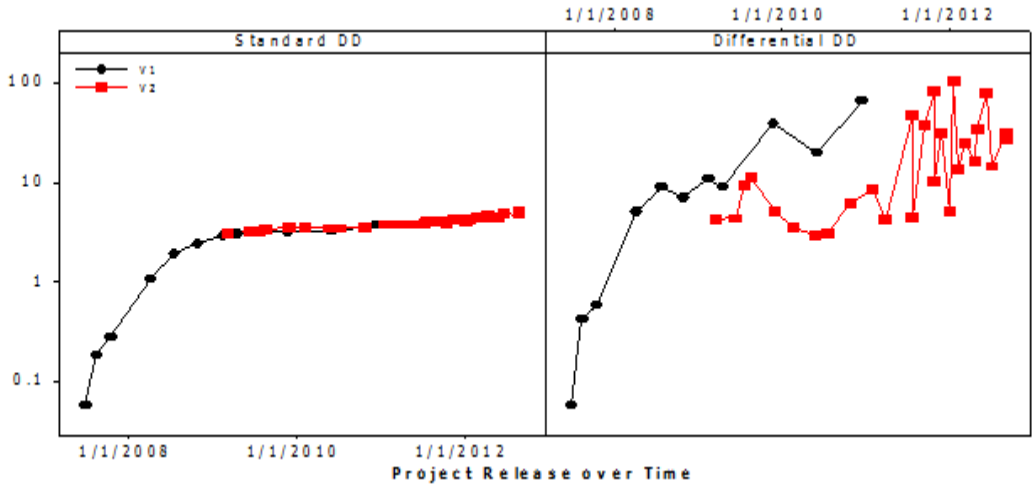

Figure 4 The sDD and dDD plot over Camel project lifetime

Table 2 Camel Project Defect Density Figures

Release	Date	Size	Defects	dDD	sDD
1.0.0	7/2/2007	17618	1	0.06	0.06
1.1.0	8/18/2007	27155	4	0.42	0.18
1.2.0	10/19/2007	35689	5	0.59	0.28

1.3.0	4/7/2008	42547	35	5.10	1.06
1.4.0	7/22/2008	47570	46	9.16	1.91
1.5.0	10/31/2008	53523	41	6.89	2.47
1.6.0	2/16/2009	56170	29	10.96	2.87
2.0M1	3/16/2009	65021	38	4.29	3.06
1.6.1	4/18/2009	58014	17	9.22	3.07
2.0M2	6/15/2009	71367	28	4.41	3.18
2.0M3	7/23/2009	72870	14	9.31	3.31
2.0.0	8/22/2009	73678	9	11.14	3.39
1.6.2	11/24/2009	58246	9	38.79	3.21
2.1.0	12/4/2009	82206	43	5.04	3.56
2.2.0	2/14/2010	87313	18	3.52	3.56
2.3.0	5/26/2010	96403	26	2.86	3.50
1.6.3	6/3/2010	58497	5	19.92	3.28
2.4.0	7/15/2010	103585	22	3.06	3.47
2.5.0	10/28/2010	108778	32	6.16	3.59
1.6.4	12/16/2010	58527	2	66.67	3.79
2.6.0	1/29/2011	112028	28	8.62	3.74
2.7.0	3/21/2011	114825	12	4.29	3.75
2.7.1	4/12/2011	114830	0	0.00	3.75
2.7.2	6/3/2011	114830	0	0.00	3.75
2.7.3	7/19/2011	115244	20	48.31	3.91
2.8.0	7/23/2011	122934	35	4.55	3.95
2.8.1	9/16/2011	123251	12	37.85	4.04
2.7.4	10/24/2011	115268	2	83.33	3.93
2.8.2	10/24/2011	124906	17	10.27	4.12
2.8.3	11/21/2011	125261	11	30.99	4.20
2.9.0	12/31/2011	136618	59	5.20	4.28
2.7.5	1/15/2012	115317	5	102.04	3.97
2.8.4	1/29/2012	126201	13	13.83	4.27
2.9.1	3/5/2012	138060	36	24.97	4.50
2.9.2	4/17/2012	139107	17	16.24	4.59
2.8.5	4/27/2012	126768	19	33.51	4.40
2.8.6	6/9/2012	126831	5	79.37	4.44
2.10.0	7/1/2012	145047	85	14.31	4.98
2.9.3	8/28/2012	140192	35	32.26	4.80
2.10.1	8/28/2012	145516	13	27.72	5.06

2.5.2 Project lucene

In the Lucene project (see Figure 5 and Table 3) we observe three versions v2, v3 and v4, where the version changes occur when the project undergoes some major updates. The v2 consist of 10 releases, v3 consists of 11 releases and v4 consists of 3 releases. We also observe the increase of size but smaller than the Camel project. The size increases about two times from the first observed release of v2 (53142 loc) to the last observed release of v4 (135441 loc). The sDD span from 1.016 to 10.96 defects per thousand lines of code. The average sDD of the Lucene project for twenty four releases is found to be 5.4 defects per thousand lines of code. Where, the average dDD of Lucene project is found to be 41.2 defects per thousand lines of code. Table 3 shows the Lucene project releases defect density figures. Figure 5 shows the sDD and dDD plot over the project lifetime.

Similar to the Camel project we observe an increase of sDD over releases in the Lucene project, where the high variability of dDD over the releases is found for Lucene project.

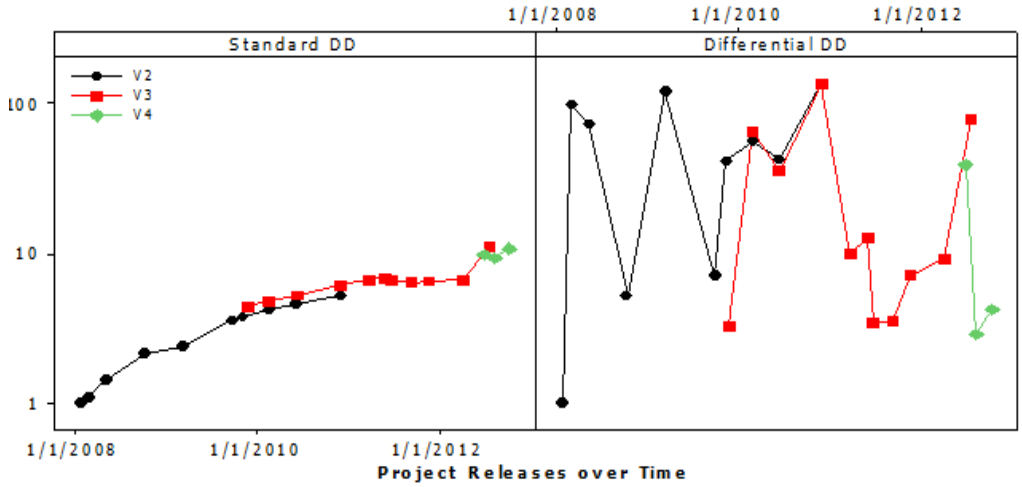

Figure 5 The sDD and dDD plot over Lucene project lifetime

Table 3 Lucene Project Defect Density Figures

Release	Date	Size	Defects	dDD	sDD
2.3.0	1/21/2008	53142	54	1.0161	1.016
2.3.1	2/19/2008	53193	5	98.039	1.109
2.3.2	5/1/2008	53428	17	72.340	1.422
2.4.0	10/5/2008	66027	66	5.238	2.150
2.4.1	3/5/2009	66168	17	120.567	2.402
2.9.0	9/21/2009	88203	159	7.2157	3.605
2.9.1	11/3/2009	88611	17	41.666	3.780
3.0.0	11/22/2009	81290	24	3.278	4.416
2.9.2	2/22/2010	88931	18	56.25	4.239
3.0.1	2/22/2010	81611	21	65.420	4.876
2.9.3	6/6/2010	89407	20	42.016	4.675
3.0.2	6/11/2010	82111	18	36	5.309
2.9.4	11/28/2010	89669	36	137.40	5.263
3.0.3	11/28/2010	82366	34	133.33	6.143
3.1.0	3/26/2011	96222	136	9.8152	6.672
3.2.0	5/31/2011	99110	37	12.811	6.850
3.3.0	6/26/2011	104342	18	3.4403	6.679
3.4.0	9/9/2011	112175	28	3.574	6.463
3.5.0	11/22/2011	117453	38	7.199	6.496
3.6.0	4/6/2012	124809	69	9.380	6.666
4.0 A	7/3/2012	138203	525	39.196	9.818
3.6.1	7/22/2012	124975	13	78.313	10.96
4.0 B	8/13/2012	150344	35	2.8827	9.34
4.0.0	10/12/2012	135441	63	4.2273	10.8

2.5.3 Project whirr

In project Whirr (see Figure 6 and Table 4), we observe a six times increase in size of project from the first release (1047 loc) to the last observed release (7832 loc). The sDD found to be in a range of 3.4 – 11.6 where the dDD is found to be in a range of 1.15 - 70 defects per thousand lines of code. The average sDD of eight releases of Whirr project is found to be 5.4 and average dDD is found to be 12.9 defects per thousand lines of codes. Table 4 shows the Whirr project releases defect density figures. Figure 6 shows the sDD and the dDD plot over the project lifetime. For the sake of clarity the data point in row 9 has been considered an outlier and removed.

For the Whirr project we do not observe the continuous increasing trend of sDD over the time. However it is also found that in Whirr project the differential defect density has high variability over time too.

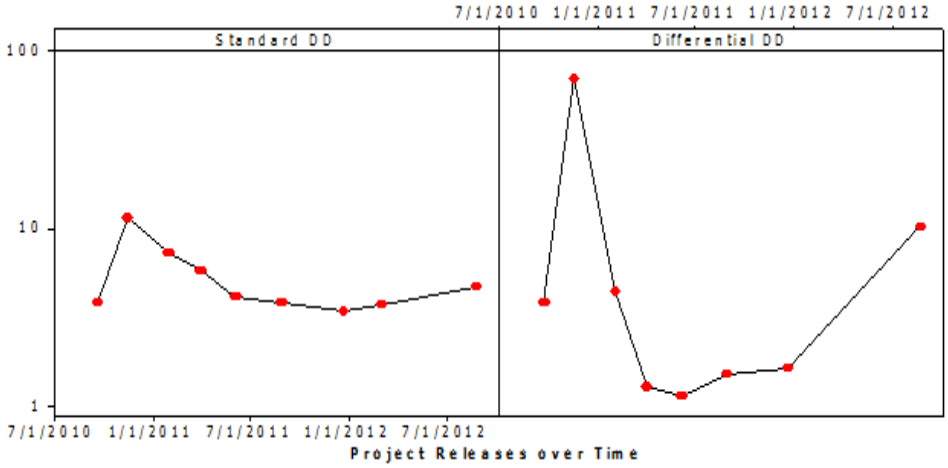

Figure 6 The sDD and dDD plot over Whirr project lifetime

Table 4 Whirr Project Defect Density Figures

Release	Date	Size	Defects	dDD	sDD
0.1.0	9/20/2010	1047	4	3.820	3.820
0.2.0	11/15/2010	947	7	70	11.615
0.3.0	1/30/2011	2299	6	4.437	7.394
0.4.0	3/30/2011	3074	1	1.290	5.855
0.5.0	6/3/2011	4801	2	1.158	4.165
0.6.0	8/27/2011	5455	1	1.529	3.849
0.7.0	12/20/2011	6668	2	1.648	3.449
0.7.1	2/28/2012	6669	2	2000 (outlier)	3.748
0.8.0	8/24/2012	7832	12	10.318	4.724

2.5.4 Validity threats

In this section we discuss the validity threats using the classification proposed by [7]. As for internal validity we do not have control over the exact line of code counting methods adopted by different projects. For example out of the total lines of code how many are the comment lines, blank lines and the physical source lines of code etc.

As for construct validity, we strictly relied on the exact definition of defects in which no accumulation of issues, warnings and temporary problems are considered. The projects come from the Apache Software Foundation, so we count on the strict policies defined by the Foundation, among others for the log of defect data. For dDD, the numerator is a proxy of the defects belonging

to a release. For example in the Lucene release (2.3.0) only 51 new lines of code are added to compose the release (2.3.1) but these additions of new 51 lines of code introduce 5 defects in release 2.3.1. Probably the 5 defects found come from releases before 2.3.0. This concern was also highlighted in the study [8] that if delta loc is used, the defect density would be higher. As for external validity we have used three projects, clearly a low number, so external validity is limited.

2.6 Discussion and conclusion

In this section we discuss the information gained from the analysis by providing answers to our research questions.

RQ1. In all three projects we observe that sDD and dDD behave very differently. sDD has a stable trend, while dDD is unstable and varies widely from release to release.

sDD has a steadily growing trend in two projects, while it has an unclear trend in the Whirr project. However Camel and Lucene are larger and longer projects. So the finding indicated to be further evaluated in other projects, is that sDD grows over time and is not steady.

dDD, as observed, is very unstable over time in all projects. However it seems to have boundaries, the lower at 1 defects/Kloc, and the upper at 100 defects/Kloc. This trend should be further analyzed in similar projects. It is evident that this behavior depends in part on the fact that defects found between two releases may come from any previous release.

RQ2. sDD and dDD have different trends that clearly result from their definition. sDD is a *global* project indicator, while dDD considers a time frame in a project, so it is a *local* indicator.

As for sDD, a reasonable assumption seems to be that low sDD means a higher quality project, and vice versa. However, the steady growth of sDD (as discussed in RQ1, and if confirmed) means either that the quality of a project decreases over time, or that sDD is not a reliable quality indicator.

As for dDD, its high variability could be either normal behavior, or an indicator of a project that is not under control. In the latter case projects should try to reduce dDD as much as possible. It can therefore be deduced that extremely low or high values of dDD could be seen as Technical Debt indicators.

A low dDD could indicate that defects are left to be found in the future, while a high dDD could indicate that debt is repaid finding defects from past releases.

In conclusion it can be observed that both sDD and dDD are useful variants of DD, the former providing a global view of the qualityof project, the latter a local view.

However, further studies are needed to consider the following research questions:

Do the majority of projects show a growing sDD over time?

Is a growing sDD over time correlated with a lower external quality of the project? For instance in term of reliability?

Is an unstable dDD is normal behavior for all projects? Is it related to the type of changes that the project undergoes, Or to some other project characteristics? And can its variability be reduced?

Chapter 3

3. The Impact of Product attributes on Software Quality

Originally published at:
Proceedings of 19th Asia-Pacific Software Engineering Conference (APSEC, 2012). Proceedings of the International Conference on Information Technology and Software Engineering (ITSE 2012). Under Review at: Springer Software Quality Journal (2014).

S. M. A. Shah, M. Morisio and M. Torchiano

3.1 Introduction

Quality of software projects is of concern to all stakeholders e.g. users, practitioners, researchers etc. The very nature of software as a continuously evolving entity makes it possible for several different attributes. The one set of attributes that are considered during the development of software are the product attributes. The product attributes are the attributes of software itself.

There could be different types of product attributes e.g. the external characteristics of product like age, size, development language, environment etc. are considered as external product attributes and the internal structure characteristics are considered to be internal product attributes e.g. module characteristics and the complexity etc. Obviously these attributes should influence either directly or indirectly the quality of the software.

Therefore, many internal properties are used to predict quality e.g. module characteristics and complexity. Considering the module characteristics (size, quality and dependency) should have impact on the software quality. Typically a module is intended at the physical level (a file as part of a project), its size measured in LOC, its quality measured in defects or defect density (DD defined as defects/size). In research studies [14][15], it is found that smaller modules have higher DD compared to the larger modules. Many researchers have made their efforts to highlight different relationship between the size and DD of the modules. For example, in studies [16][17] it is found that the modules DD increases with the increase in the size of the modules. In studies [14][15] it is found that the DD decrease with the increase in the size of the modules. However the studies [18][19] show no significant relation of DD with the size of the modules. In summary the studies show the increase or decrease of DD with size. Although at project level, the consequences of these observations that are proven at module level are not well known. For example if a project is constructed with larger sized modules or small sized modules then what should be the resultant DD of the project. This allows us to devise our research to study the module attributes that have influence on project DD. This paper studies the modules attributes and their effect at project level.

Considering complexity as an internal product attribute can be measured using different techniques applied to source code and design [20][21]. The common understanding about the complexity is its positive correlation with the defects. Although the relation is not always linear, it has significant impact. For the complexity measurement, different complexity metrics have been devised in past years [20][21][22][23][24]. Studies showed that the majority of defects is caused by a small portion of the modules [25]. These modules can be identified before time by examining the complexity to reduce the post release and maintainability work. However it is not straightforward, for the complexity we have different complexity metrics. The selection of appropriate complexity metrics that best relate and indicate with the defects is of concern and requires minimal empirical evaluation for the selection.

The other type of product attributes are the internal attributes of the product e.g. the module characteristics (size, quality and dependency). The relationship between size and quality in software projects is highly debated, both at project and module level. In recent works many researchers characterize the DD of software modules based on different factors like size, complexity etc. [26][27]. While such contributions are very important to understand the internal quality behavior of software, we believe they only tell one part of the story: the perspective of the whole software product instead of individual modules.

The objective of this chapter is to characterize the software projects DD based on different product attributes. The study has a twofold outcome, first it aggregate and analyze DD figures of software projects to answer very simple question, both from researchers and practitioners point of view, such as ‘what is the typical defect density in a project regarding an attribute’? Second it answers the question, ‘what are the attributes to characterize the defect density in a project’?

3.2 Related work

What is the typical defect density of a project, the earliest study conducted by Akiyama’s [28] reports that a 1 KLoC seems to have approximately 23 defects. McConnell [29] reports 1 to 25 defects per thousand lines. Chulani [30] reports it to be 12 defects per thousand lines.

Considering the external attributes to characterize DD, Fenton and Ohlsson study shows that size is a good attribute to characterize defects and DD at module level [27]. Many other studies reports size as a factor to characterize the defects and defect proneness at module level with different implications for open and close source software’s [26][31][32]. Raghunathan et al. compared the quality aspects of both open source and close source software’s and they found no difference of quality of open source and close source software’s [33]. Phipps found that a typical C++ programs had two to three times as many defects per line of code as a typical Java programs [34]. Graves et al. stated that we can characterize the faults based on number of modifications, the size of the modifications and on the age of a file [35]. Zvegintzov stated that the quality of software also increases with the age of software [36]. Cotroneo et al. highlighted the significant correlation of defects with the software aging [37].

Considering the internal attributes to characterize DD, many studies have analyzed modules within a single project. Withrow [16] showed her work by examining the 362 modules of the ADA. She divided the modules into 8 groups based on the module size. She showed that after a certain range of module size (161-250 lines of code) the defects start increasing with the module size. This result also supports the Banker and Kemerer hypothesis [17] where they proposed the optimal module size. The minor size of the module has positive impacts and for greater size, the negative impact starts. Moller and Paulish highlighted that for the module of size smaller than 70 lines of code DD increases significantly and modules that have size greater than the 70 lines of code have similar trends toward DD [38]. Hatton [39] studied 'NASA Goddard' project along with Withrow's data set. He classifies the modules in two categories. For size up to 200 LOC, he suggested that the DD grows logarithmically with the module size and for modules larger than 200 LoC, he observed a quadratic model. Rosenberg [40] has commented on Hatton [39] argument that the observed decrease in DD with rising module sizes is misleading. Shen et al. [14], worked on three separate releases of an IBM software project by studying 108 modules. They highlighted 24 software modules with size exceeding 500 LOC. They affirm that increases in size did not influence the DD. For the remaining 84 modules, they showed that DD declines as size grows. A study done by Basili and Perricone [15] showed the division of 370 modules into 5 groups based on the module size with increment of 50. They observed the trend of having lower DD of larger module. Fenton and Ohlsson [18] have studied the modules of large telecommunication projects. They selected the modules randomly for the study but did not observe significant dependence of module size with DD. In addition many studies analyzed modules from more than one project. Andersson and Runeson [19] replicated the Fenton and Ohlsson [18] study using the data of three telecommunication projects. They were also not so much successful in finding the significance relation between the number of defects and LOC compared to the original study. El Emam et al. studied three software projects written in C++ and Java. They highlighted that there is a continuous relationship between the class size and faults [41]. Koru et al. [42], studied four large open source projects: Mozilla, Cn3d, JBoss, and Eclipse. They observed that smaller classes are more problematic than larger ones. In particular, for open source software the theory of Relative Defect Proneness (RDP) [32] is postulated about the size defect relationship, stating that "smaller modules are less but proportionally more defects prone compared to larger modules".

Considering complexity as internal product attribute many studies show an acceptable correlation between complexity metrics and software defect proneness [43][44][45][46]. English et al, highlighted the usefulness of the CK metrics for identifying the fault-prone classes [47]. Gyimothy et al. studied the object oriented metrics given by CK for the detection of fault prone source code of open source Web and e-mail suite called Mozilla. They found that CBO metric seems to be the best in predicting the fault-proneness of classes and DIT metric is untrustworthy, and NOC cannot be used at all for fault-proneness prediction [46]. Yu et al. examined the relationship between the different complexity metrics and the fault proneness. They used univariate analysis and found that WMC, LOC, CBOout, RFCout LCOM and NOC have a significant relationship with defects but CBOin, RFCin and DIT have no significant relationship [48]. Subramanyam and Krishnan examined the effect of the size along with the WMC, CBO and DIT values on the faults by using multivariate regression analysis for Java and C ++ classes. They conclude that size was a good predictor of defects in both languages, but WMC and CBO could be validated only for C++ [45]. Olague et al. studied three OO metrics suites for their ability to predict software quality in terms of fault-proneness: the Chidamber and Kemerer (CK) metrics, Abreu's Metrics for Object-Oriented Design (MOOD), and Bansiya and Davis' Quality Metrics for Object-Oriented Design (QMOOD). They concluded that CK and QMOOD suites contain similar components and produce statistical models that are effective in detecting error-prone classes. They also conclude that class components in the MOOD metrics suite are not good class fault-proneness predictors [49]. However, Nagappan et al. stated that there is no single set of complexity metrics that could act as a universally best defect predictor [50].

In most of the related work, the product attributes are used to characterize the defects or defect proneness without considering DD. If in some cases DD was used, it was used considering less number of projects. This makes serious concern for the need of a study which characterized product attributes based on DD.

3.3 Data set: systematic literature review

3.3.1 Research design

We followed the framework of Arksey and O'Malley [51] for conducting our scoping study. There are five stages in the adopted framework. We present the stages of the study in the current section (in subsection 3.3.1.1, 3.3.1.2, 3.3.1.3, 3.3.1.4, and 3.3.1.5, respectively).

3.3.1.1 Stage 1: research questions definition

The present paper aims at answering the impact of some important product factors concerning DD.

RQ1: What are the typical figures of DD in software projects?

Such figures provide quality managers and project managers benchmarks to define quality goals upfront, to evaluate the quality of a project during development, and to assess the quality of a project post mortem.

RQ2: Is there a difference in DD between open and closed source project?

Since often the context, motivation, and development process differ between open source and proprietary projects, as the anecdotal story goes one would expect a different quality of products. We aim at finding some evidence, at least in terms of DD.

RQ3: Is there a difference in DD among programming languages?

Different programming languages encompass e.g. varying styles, expressive power and abstraction level. Such differences are likely to influence the DD of projects.

In general (RQ2) & (RQ3) provide project managers evidence on the influence of programming language (RQ3) and reuse of OSS components (RQ2) on project quality.

RQ4: What is the relationship between DD and project size?

A lot of analysis has been conducted on the relationship between module size and DD, but we could not find any on project size and DD. RQ4 provide researchers the evidence of a relationship between project size and DD.

RQ5: What is the relationship of DD and project age?

A reasonable expectation is that the longer a project is released, more defects are found and therefore the higher the defect density. In addition it also provides researchers the evidence of evolution of reliability over time, not only on failure happening (traditional reliability definition) but also on DD.

3.3.1.2 Stage 2: relevant studies identification

We identified the relevant studies by following a three phase search strategy.

Phase 1: The first phase is exploratory and considers papers published in top software engineering journals from 2000 to 2011. The search string is "Defect Density" OR "Fault Density" OR "Reliability". Fault density was used as a possible synonym of defect density, while reliability was used because related papers often present defect data. The search was applied through 'Science direct', 'Springer link', 'IEEE explorer' and 'ACM digital library'. The web site search function "search in all fields" including full text was used for every journal.

The journals considered are:

- System and Software
- IEEE Transaction on Software Engineering
- ACM Transaction on Software Engineering and Measurement
- IEEE Software
- Empirical Software Engineering
- Information and Software Technology

Inclusion and exclusion criteria are explained in 3.3.1.3. The results of Phase 1 are shown in Table 5.

Table 5 Distribution of papers in phase 1

Source	**Scanned**	**Included**
Systems and Software	654	1
IEEE Transaction on Software Engineering	83	0
IEEE Software	55	0
ACM Transaction on Software Engineering and Measurement	6	1
Empirical Software Engineering	173	4
Information and Software Technology	412	2

Phase 2: We search using two publication databases, IEEE explorer and ACM digital library. The search keywords are "Software Defect Density" and "Software Fault Density". We had to add "Software" to filter out excessive non relevant hits. The word 'Reliability' was dropped to narrow down the focus of search as it did not select any relevant paper in phase 1.

Same inclusion and exclusion criteria explained in 3.3.1.3 are used. The result of Phase 2 is shown in Table 6. The table does not consider papers already found in Phase 1.

Table 6 Distribution of papers in phase 2

Source	**Scanned**	**Included**
IEEE explorer	184	8
ACM digital library	2206	2

Phase 3: Phase 1 and phase 2 selected 18 papers. In Phase 3 we followed the references of selected 18 papers to identify other relevant studies. But we did not find any new relevant paper. In addition we particularly searched the Promise proceedings and found one study that indicates the availability of required data set at promise data repository.

This led to a selection of 19 papers in total, out of 3774: 8 from Phase 1 + 10 from Phase 2 + 1 from Phase 3. In total the 19 papers contain DD data about 110 projects. For space constraints the list of projects and their characteristics are not included in the paper but can be found as an electronic resource at: http://softeng.polito.it/syed/AppendixA.pdf. Table 7 reports the selected studies for the analysis.

Table 7 List of selected studies for DD

ID	**Publications**
S1	J.-H. Lo and C.-Y. Huang, "An integration of fault detection and correction processes in software reliability analysis," *Journal of Systems and Software*, vol. 79, no. 9, pp. 1312-1323, Sep. 2006.
S2	A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software

	development: Apache and Mozilla," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 3, pp. 309-346, 2002
S3	E. Weyuker, T. Ostrand, and R. Bell, "Comparing the effectiveness of several modeling methods for fault prediction," *Empirical Software Engineering*, vol. 15, no. 3, pp. 277-295-295, Jun. 2010.
S4	S. Kpodjedo, F. Ricca, P. Galinier, Y.-G. Guéhéneuc, and G. Antoniol, "Design evolution metrics for defect prediction in object oriented systems," *Empirical Software Engineering*, vol. 16, no. 1, pp. 141-175-175, Feb. 2011.
S5	E. Weyuker, T. Ostrand, and R. Bell, "Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models," *Empirical Software Engineering*, vol. 13, no. 5, pp. 539-559-559, Oct. 2008
S6	G. Koru, H. Liu, D. Zhang, and K. El Emam, "Testing the theory of relative defect proneness for closed-source software," *Empirical Software Engineering*, vol. 15, no. 6, pp. 577-598-598, Dec. 2010
S7	T. Illes-Seifert and B. Paech, "Exploring the relationship of a file's history and its fault-proneness: An empirical method and its application to open source programs," *Information and Software Technology*, vol. 52, no. 5, pp. 539-558, May 2010
S8	M. F. Ahmed and S. S. Gokhale, "Linux bugs: Life cycle, resolution and architectural analysis," *Information and Software Technology*, vol. 51, no. 11, pp. 1618-1627, Nov. 2009
S9	P. Abrahamsson and J. Koskela, "Extreme programming: a survey of empirical data from a controlled case study," in *Empirical Software Engineering, 2004. ISESE '04. Proceedings. 2004 International Symposium on*, 2004, pp. 73-82
S10	P. Mohagheghi, R. Conradi, O. M. Killi, and H. Schwarz, "An empirical study of software reuse vs. defect-density and stability," in *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, 2004, pp. 282-291.
S11	A. Mockus and D. Weiss, "Interval Quality: Relating Customer-Perceived Quality to Process Quality," in *Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on*, 2008, pp. 723-732.
S12	A. Gupta, O. P. N. Slyngstad, R. Conradi, P. Mohagheghi, H. Ronneberg, and E. Landre, "A Case Study of Defect-Density and Change-Density and their Progress over Time," in *Software Maintenance and Reengineering, 2007. CSMR '07. 11th European Conference on*, 2007, pp. 7-16
S13	M. Cartwright and M. Shepperd, "An empirical investigation of an object-oriented software system," *Software Engineering, IEEE Transactions on*, vol. 26, no. 8, pp. 786-796, 2000
S14	Hongyu Zhang, "An investigation of the relationships between lines of code and defects," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, 2009, pp. 274-283.
S15	T. Dinh-Trong and J. M. Bieman, "Open source software development: a case study of FreeBSD," in *Software Metrics, 2004. Proceedings. 10th International Symposium on*, 2004, pp. 96-105
S16	N. Fenton, M. Neil, W. Marsh, P. Hearty, L. Radlinski, and P. Krause, "Project Data Incorporating Qualitative Facts for Improved Software Defect Prediction," in *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, 2007, p. 2.
S17	S. Wu, Q. Wang, and Y. Yang, "Quantitative analysis of faults and failures with multiple releases of softpm," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, Kaiserslautern, Germany, 2008, pp. 198-205.
S18	P. Mohagheghi, R. Conradi, and J. A. Borretzen, "Revisiting the problem of using problem reports for quality assessment," in *Proceedings of the 2006 international workshop on Software quality*, Shanghai, China, 2006, pp. 45-50.
S19	M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to

	defect prediction," in Proceedings of the 6th International Conference on Predictive Models in Software Engineering, Timi\şoara, Romania, 2010, pp. 1–10.

3.3.1.3 Stage 3: study selection

The inclusion and exclusion criteria employed in stage 2 are defined below.

Inclusion criteria

The inclusion criteria were applied at three subsequent levels. First we read the papers titles to select those relevant to our study. Then we read the abstracts of previously selected papers and kept the relevant papers only. As a third step we thoroughly read the papers and included only those studies which satisfied the following criteria:

- Are related to software engineering
- Are related to software projects
- Contain directly figures of DD, or contain data that allows to compute DD indirectly (such as number of post release defects and size in LOC)
- Mention that DD was computed after the particular release or the project (operation phase, post release phase, etc).

Exclusion criteria

The studies that did not satisfy the inclusion criteria were excluded.

3.3.1.4 Stage 4: charting the data

DD is the key object of this study we only reported DD in LoCs on post release phase, but despite its apparent simplicity it can take slightly different forms.

Typically DD can be sampled at different times during the evolution of a project; in the meantime the code base may undergo complex transformations, e.g. code additions, changes, deletions. Therefore it is difficult to match a defect to corresponding code base. In this work we consider DD as a cumulative measure, i.e. we count defects since the first release; such a definition considers only post-release defects, therefore temporary problems happening before release should be filtered out. Moreover the data available in the articles is often not as accurate as desired. Such considerations led us to adopt as a reference the definition of defect density as a cumulative metric:

$$DD = \frac{CNDD}{Size}$$

Where CNDD is the cumulative number of post release defects in the observed period, and Size is measured at the end of the observed period in thousands of lines of code (KLoC). On the basis of the above referenced construct we performed a data extraction using either DD figures provided directly in the papers or values computed from the data available in the papers. In addition to the main dependent variable DD, we collected also some context variables:

- Type: whether the project was developed as open or closed source,
- Language: the main programming language used,
- Size: the size of the project in KLoC,
- Age: is the calendar time between first and last release on which DD is computed

Data analysis and hypotheses

To answer the research questions we report descriptive statistics, in particular distribution information, and when applicable we also statistically test some hypothesis. To represent the distribution of DD for the projects we use both cumulative distribution diagrams and box plots. The former report on the horizontal axis the DD values sorted in ascending orders and on the vertical axis the proportion of values not greater than the corresponding DD.

Research questions RQ2 and RQ3 lend themselves to be answered by means of hypothesis testing. The respective null and alternative hypotheses can be formulated as follows.

Concerning RQ2:

$H2_0$: There is no significant difference in term of DD between open source and close source projects.

$H2_a$: There is a significant difference in term of DD between open source and close source projects.

Concerning RQ3:

$H3.1_0$: There is no significant difference in terms of DD among projects adopting different languages.

$H3.1_a$: There is a significant difference in terms of DD among projects adopting different languages.

If the above null hypothesis can be rejected we can conduct a post-hoc investigation of the pair-wise differences; in this case the Bonferroni correction for multiple tests shall be applied. For any pair of languages L1 and L2 we can formulate the hypotheses as:

$H3.2_0$: Projects developed in L1 have not lower defect density as those developed in L2

$H3.2_a$: Projects developed in L1 have lower defect density as those developed in L2

From preliminary analysis we found that the data is not normally distributed, therefore we adopt non-parametric tests. According to the recommendations in [7] we use the Kruskal-Wallis test for differences between three or more groups and the Mann-Whitney test for pair-wise differences. When comparing different groups we will also evaluate the difference from a practical point of view. For this purpose we use the standardized effect size, measured as Cohen's d.

As far as RQ4 and RQ5 are concerned we will conduct a regression analysis. Such analysis helps to determine to which extent the dependent variable varies as a function of one or more independent variables. We will consider both the statistical significance of the model and the practical significance that is expressed by the R^2 statistic.

Since the size of a project may have a huge variability, focusing a pure linear correlation may yield no result. Therefore for RQ4, we perform an additional analysis focusing on size categories and their effect on DD. In order to avoid arbitrary thresholds we identify the classes by means of a clustering algorithm. In particular we use the K-means method to identify k=3 cluster corresponding to small, medium, and large projects.

Finally we analyze the correlation among the context variables. As far as project size is concerned we consider the size categories identified through clustering. In particular, since we deal with categorical or ordinal variables, we build the contingency tables and apply the χ^2 test to detect statistically significant correlations.

In the statistical testing, the significance level is checked by the given p-value. For rejecting or accepting the null hypothesis we used the significance values of α=5%.

3.3.1.5 Results

In performing the data extraction on the paper, we were able to use directly provided figures for 46% of the projects, in another 45% of the cases we computed it starting from a number of defects and code size, and in the remaining 9% of cases we had to extrapolate the values from average values.

We carried on a preliminary analysis to identify possible outliers. As a consequence we discarded a project having DD 120 defects per KLoC.

RQ 1: What are the typical figures of DD in software projects?

Figure 7 presents the cumulative distribution diagram for the defect density of the surveyed projects. The DD is reported on a logarithmic scale, we observe that it spans nearly three orders of magnitudes, from 0.05 to close to 50.0. Table 8, in its first row, reports the central tendency of DD (mean 7.47, median 4.3, standard deviation 7.99). While industry experience is about 1 to 25 errors per 1000 lines of code for delivered projects according to [29], in our data set 89 out of 109 projects are located in that range. The figure also reports, in gray line, the fitted normal distribution. Also based on the results of the Shapiro-Wilk test ($p<0.001$) we confirm that the DD data is not normally distributed.

Table 8 Descriptive statistics of DD of projects

Group		N	Mean	Median	Std Dev
All		109	7.47	4.3	7.99
Type	Closed source	77	8.6	5.4	8.49
	Open source	32	4.66	2.75	5.84
Language	C	43	10.0	7.9	7.98
	Java	45	5.9	3.5	6.22
	C++	11	8.73	1.3	13.17
	Other	10	1.89	1.1	2.83

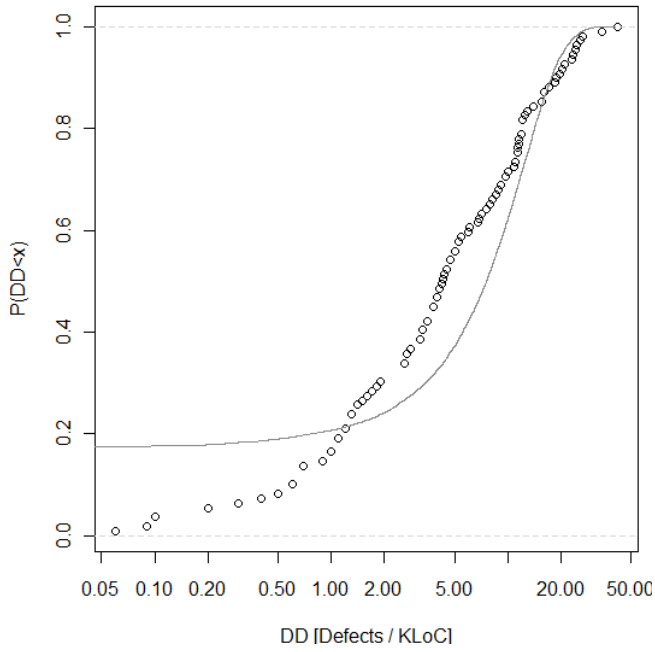

Figure 7 DD cumulative distribution

RQ2: Is there a difference in DD between open and closed source project?

Our data set contains 77 closed source projects, and 32 open source projects. Figure 8 shows the cumulative distribution by type (i.e. open vs. closed source).

In addition Figure 9 contains the box plot. The summary descriptive statistics are reported in Table 8 divided by type. To answer RQ2 we test the hypothesis H20. The Mann-Whitney test reports a p-value = 0.004, which is below the α threshold, therefore we can reject the null hypothesis. Open

source projects in our sample have a DD that is on average 4 Defects/KLoC smaller than closed source ones. In practical terms the difference can be considered of medium size (Cohen's d = 0.5).

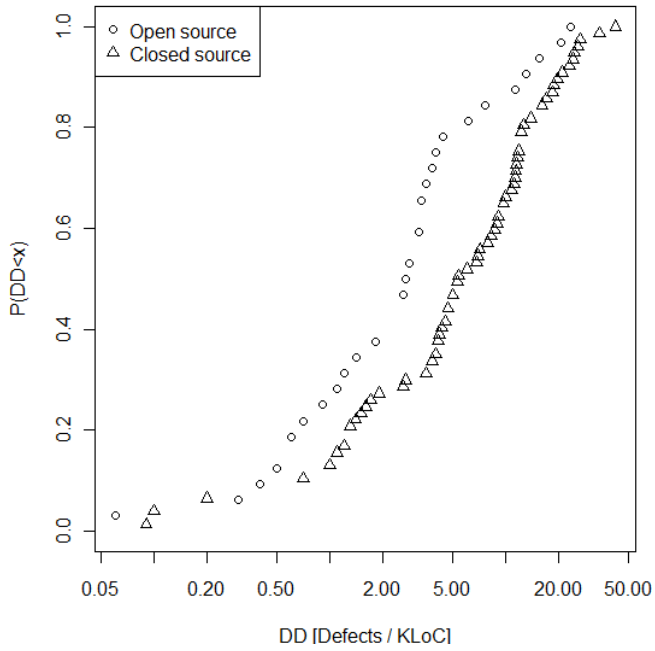

Figure 8 DD cumulative distribution for Close source and Open source projects

Such a result is confirmed by looking at the Figure 9 which shows close source having more DD and suggests a larger variation for closed source projects than open source ones.

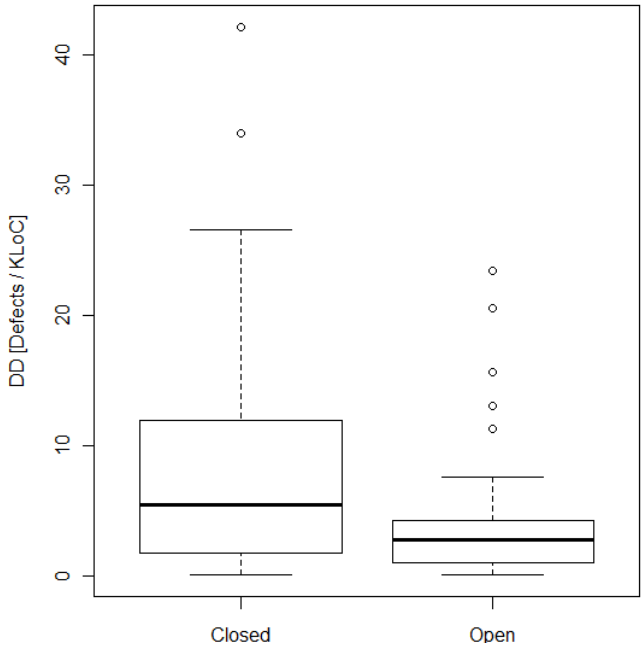

Figure 9 Box plot of Close Source vs. Open Source DD

RQ3: Is there a difference in DD among programming languages?

Our data set contains 43 C projects, 45 Java projects and 11 C++ projects. We excluded from the analysis 10 projects that were coded in other languages (i.e. Perl) or for which it was not possible to

identify a clear major language. Figure 10 presents the cumulative distribution diagram for the three languages under study; Figure 11 shows the box plots. Descriptive statistics are in Table 8, rows 2 and 3. In this case visual analysis does not give clear suggestions.

The first hypothesis concerning RQ3, H3.10 can be tested using the Kruskal-Wallis test. The returned p-value is 0.009, therefore the null hypothesis can be rejected.

Given the above result we proceed with the pair-wise comparisons. In particular we test the $H3.2_0$ for the three possible pairs of languages, by means of the Mann-Whitney test. In assessing this test we adopt an α divided by 3 according to the Bonferroni rule.

For the pair (Java, C) we obtained a p-value of 0.003, therefore we can reject the null hypothesis. For the pairs (Java, C++) and (C++, C) we obtained the p-values 0.375 and 0.119, respectively, therefore we cannot reject the corresponding null hypotheses.

The significant difference can be considered of medium size (Cohen's d = 0.5), C projects have a DD that is an average 4.1 defect per KLoC higher than Java ones. In summary, as regards programming languages, there is evidence that the defect density in Java is lower than in C.

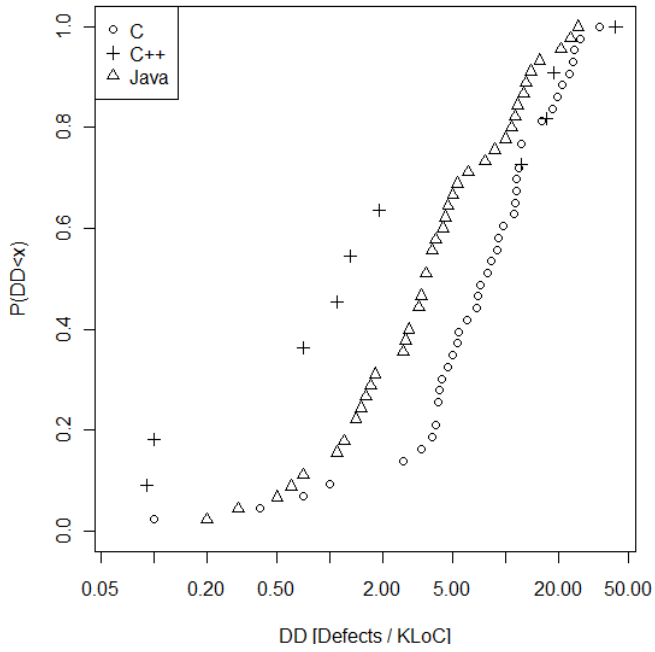

Figure 10 DD cumulative distribution by programming language

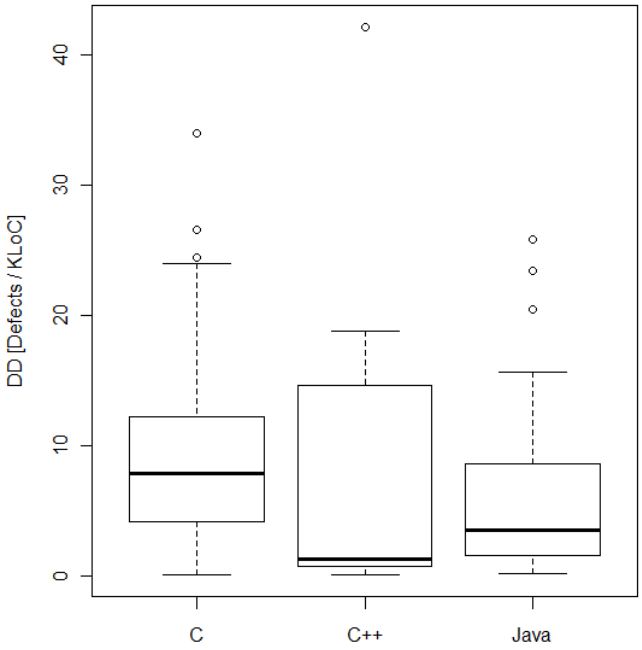

Figure 11 Box plot of DD per programming language

RQ4: What is the relationship between DD and project size?

In our data set, the size is reported for 108 projects. Figure 12 plots DD vs. size of projects in KLoCs. We used a logarithmic scale for both axes to be able to discern the individual points, which would appear flattened against the lower and left borders if a linear scale were used. We performed the regression analysis to find the relation between DD and size of project. The regression equation is:

$$DD = 8.03 - 0.000002\ Size$$

The p-value of the regression is 0.013 and the corresponding adjusted R^2 is 5.5%. There is a negative correlation but it has a limited practical impact.

Since both the DD and size distributions are extremely skewed – actually requiring a dual log scale to have a discernible representation – we also conducted a regression analysis on the log of DD and Size. In this case the regression equation is:

$$\log(DD) = 5.12 - 0.341\ \log(Size)$$

The regression's p-value is smaller than 0.001 and the corresponding adjusted R^2 is 22%. The negative correlation, for the log-values has a higher, though still small practical relevance.

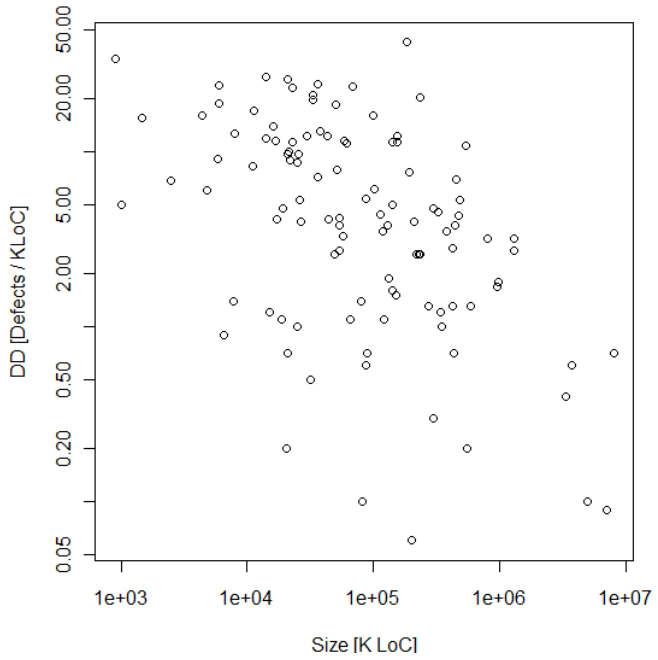

Figure 12 DD vs. size of project in logarithm scale

Finally we investigated a possible relationship between defect density and the category of projects (e.g. small, medium, large). To explore this possibility we identify project clusters by size using the K-means cluster analysis algorithm. The analysis identified 3 clusters that are described in Table 9. The ranges of DD for each Size category are plotted in Figure 13. Visual analysis shows a significant difference among the three groups. We conducted a Kruskal-Wallis test and we obtained a p-value of 0.0002, indicating a significant difference. A pair-wise comparison was then conducted, by means of Mann-Whitney tests and adopting a α divided by 3 according to the Bonferroni rule.

Table 9 Project clusters by size: descriptive statistics

		Defect Density		
Size range	**N**	**Mean**	**Median**	**Std Dev**
0 to 400K LoC	88	8.63	5.3	8.4
400K to 2M LoC	15	3.3	2.8	2.72
Above 2M LoC	5	0.38	0.40	0.28

For all pairs (Small, Medium), (Medium, Large) and (Small, Large) we obtained a p-value of 0.0146, 0.003 and 0.0006 respectively, which can be considered significant. The significant differences can be considered large, Cohen's d is being 0.83, -1.5 and -1.3 respectively. In summary we cannot find a statistical meaningful direct relationship between DD and size. However, by clustering projects we find evidence that the larger the projects, the lower the defect density. In particular we find statistically significant evidence that large projects have a lower defect density than medium and small projects.

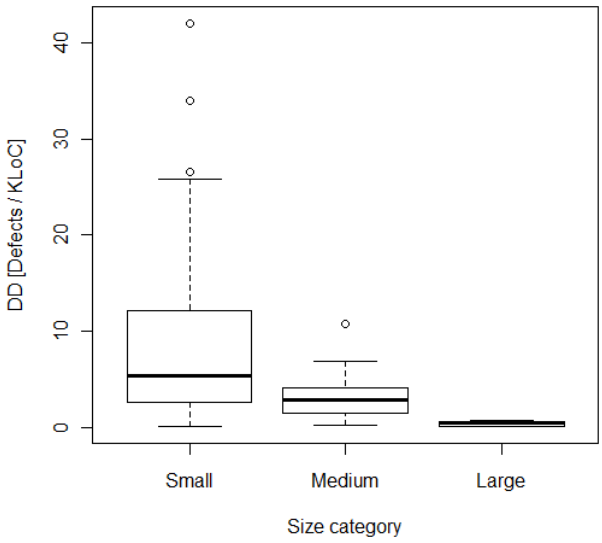

Figure 13 DD for different Size clusters

RQ5: What is the relationship between DD and Age?

In our data set the age (defined as number of years from the first release to the evaluation date mentioned in research studies or the number of years from the first release to the next release) is available for 47 projects. Figure 14 contains the plot of DD vs Age.

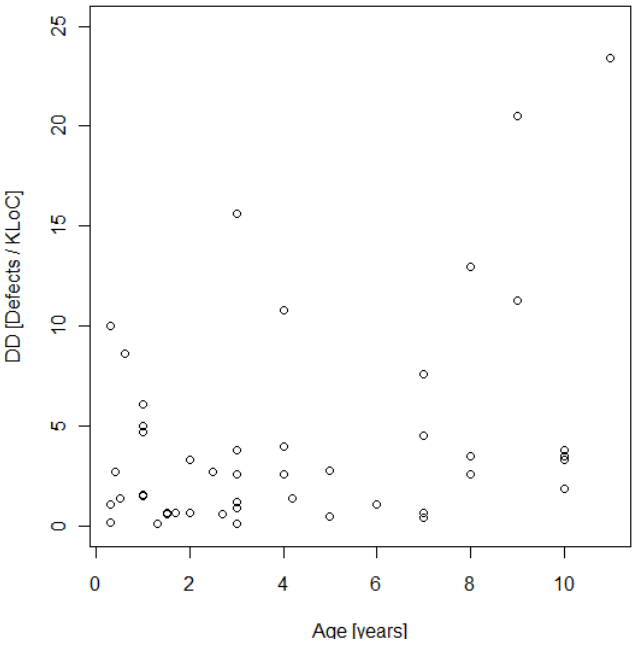

Figure 14DD vs. age of projects

We performed the regression analysis to find the relation between DD and Age of project. The regression equation is:

$$DD = 1.86 + 0.59 \cdot Age$$

The p-value of the regression is 0.011 and the corresponding adjusted R^2 is 13.8%. There is a positive correlation but it has a limited practical impact.

Context variables correlation

We test the presence of correlations among context variable by means of pair-wise contingency tables between: Size category, Type, and Language. Table 10 reports the contingency table for Size category and Type. We can notice how closed source projects are significantly more skewed towards the small projects w.r.t. open source ones (χ^2 test p-value < 0.001).

Table 10 Contingency table for Size category and Type

Size \ Type	**Closed**		**Open**	
Small	63	82%	25	78%
Medium	12	15%	4	12%
Large	2	3%	3	10%
	77	100%	32	100%

As far as programming language and size are concerned we found no statistically significant correlation (χ^2 test p-value > 0.05). Finally,

Table 11 reports the contingency table for Type and Language. We observe a significant difference, in our sample, between the languages used in open vs. closed source projects (χ2 test p-value < 0.05).

Table 11Contingency table for Type and Language

Lang Type	**C**		**C++**		**Java**		**Other**	
Closed	39	91%	11	100%	20	44%	7	70%
Open	4	9%	0	0%	25	56%	3	30%
	43	100%	11	100%	45	100%	10	100%

3.3.2 Discussion

The extraction of DD data from systematically selected articles in the literature allowed us to publish a summary of DD data available. The essential descriptive statistics are presented in Table 8. This is a result per-se, which can be of use to both researchers and practitioners.

In addition we asked ourselves a few research questions whose answers can be summarized for our particular sample of projects with the following pieces of evidence:

- There exists a statistically significant medium sized difference between open and closed source projects: the former have a DD that is 4 defects per KLoC lower than the latter.
- Java projects exhibit a significantly lower DD than C projects, 4.1 defects per KLoC on average
- In general the Size appears to be negatively correlated to DD: the larger the project the lower the DD. In particular, large projects are 10 times less defective than medium ones.

We can offer a few explanations for the above differences, although they are just speculative hypotheses that ought to be verified with further investigations.

The surprising, though not large, the difference between open and closed source projects can be explained by an heavy bias in our sample (see Table 10): the proportion of small projects studied in the literature is larger among closed source projects (82%) than for open source ones (78%).

As far as programming languages are concerned, the difference between C and Java could be explained by the different level of detail and expressive power between the two languages. Moreover the unbalanced use of languages between open and closed source projects (see Table 11) may have some influence.

Finally, the statement that "size is negatively correlated with defect density" is in (only) apparent contrast with several previous studies, e.g. [27] ignoring other additional factors. But we should keep in mind that most of previous studies referred to size and defect density of modules within a project and not of whole projects. Our result can be explained considering that large projects typically need to put a relevant effort on testing while small ones often neglect that phase, the result is a significantly lower post-release DD.

3.3.3 Threats to validity

We discuss in this section validity threat using the classification proposed by [7]. As for internal validity, the key issue is about the soundness in applying the scoping study approach. In this regard we have used known and reliable databases and journals, and repeatable search strings. We believe that inclusion and exclusion of papers in this case is easily repeatable since in the end it consists of checking that a defect density figure is available or not. This may give an impression that the reported data to calculate DD would be "survivorship bias". The selection of DD figures from important software engineering journals and electronic databases assure that nothing relevant has missed However we stress the main goal of our work is to provide an overview from data collected in real projects, as in any overview the composition is subject of debate [52], and actually we acknowledge that this is not the definitive: it is deemed to evolve indefinitely.

On the other hand we may have missed papers that report defect density data but calling it by another name. To increase reliability, the authors have cross checked all major steps in data collection and analysis.

As for construct validity, we have little concern about attributes programming language and development mode (open or closed source), that are hardly subject to ambiguities. On the contrary size, defect density and age are easily subject to ambiguities. It is well known that measuring size in lines of code is subject to variations due to programming language and modes of measuring (with or without comments, with or without blank lines, including libraries or not, etc). In nearly all selected papers the authors do not provide any information on how size was measured, so this poses a threat, of course on the values of size, and indirectly on the values of defect density. Defect density also depends on the measure of the number of defects. Here too there may be differences in the precision of measurement processes used, and on the definition of a defect used. This lack of precision of defect data may give estimates with larger error but following [52] we believe that this is better than having no data at all and relying on intuition. And again the authors of papers hardly describe the measurement process used. A procedural feature in measuring defects is the criterion according to which classify an issue as a defect. Unfortunately there is not a single technique adopted throughout the literature, when a procedure is described at all, therefore we have no way of estimating or balancing the issue, we can but accept it as an additional source of error. Another problem is when in the development process the defect number is computed. We have set as inclusion criterion papers that publish defect density in post release phase. Also at this regard we have no way of double checking whether the authors of papers all use the same meaning for it. The same applies to the attribute age.

As for external validity we underline two problems. On one hand published papers may be subject to publication bias that probably skews data toward projects with lower defect density, reducing the representativeness of the sample. On the other hand the sample is clearly smaller. 109 projects is not a negligible number, but is a very limited percentage of the number of projects released overall.

3.3.4 Conclusions

This study has mined the literature for DD figures that had not been gathered and analyzed before. On 109 software projects the mean DD is 7.47 defects per KLOC with the dispersion (standard deviation) of 7.99. These values are useful for overview purposes.

Besides we have analyzed if size, age, programming language and development mode of project (close vs. open) could be factors for DD. We found that development mode is a factor (open source projects in our sample have a lower defect density), and programming language is sometimes a factor (Java projects have lower DD than C projects, but C++ and Java, C and C++ projects have a similar DD). In addition we found that projects size is relevant (large projects have lower DD figures), while Age is not a factor.

3.4 Data set: promise repository 1, concerning modules structure:

We selected the last releases of 55 software projects from the "*Promise data repository* [5]*."*

3.4.1 Research design

In this section, we present the research questions and the selected projects on which the analysis is performed. The research questions are formalized from the related work, considering the mostly studied attributes (size, quality, dependencies) at modules level. The overall goal of this work is to understand the effect of module attributes on projects DD and how much it is different for different type of projects. Table 12 summarizes the formulated research questions and corresponding hypothesis of the study. We selected the last releases of 38 software projects from the "*Promise data repository"* [5]. The projects inclusion criterion was the availability of the required metrics (defect per module, no of line of code (LoC) of modules, module dependency metrics) to answer our research questions. The projects did not satisfy the inclusion criteria were excluded. The data set contains 23 close source projects, 15 open source software projects and 17 are academic projects that were developed by the students.

Table 12. Research questions and hypothesis

RQ 1: What is the distribution of modules on size in projects?
The goal here is to characterize the modules on size of different projects, and then check the following hypothesis.
RQ 2: What is the distribution of defect free modules in a project?
The aim here is to find the difference in DD of projects by defect free modules, checking the following hypothesis
H2.1: Projects have the same distribution of defect free modules.
H2.2: Projects with lower DD have a larger percentage of defect free modules.
RQ 3: How modules dependencies affect the projects DD?
The goal here is to find the influence of modules dependencies on the projects DD, checking the following hypothesis
H3.1: Projects having higher dependencies of modules have higher DD.
H3.2: Projects having lower dependencies of modules have lower DD.
RQ 4: How defect density of modules affects the defect density of projects?
The goal here is to find the difference in DD of projects by modules DD, checking the following

hypothesis.

H4.1: Projects with more DD have a larger percentage of modules with higher DD.

Notion of Defects: In this work, we consider only post release defects, therefore temporary problems, non defect items like issues, warnings; temporary problems and further enhancements are not included. We believe on the authenticity and reliability of post release defects of the projects available at Promise repository as the data set is publicly available and used in many prior research studies.

Notion of Modules: In this work, the module is assumed to be a smallest unit of functionality i.e. set of declarations and subroutines usually belonging to one file.

Notion of Defect Density: DD is the key object of this study the DD is reported in LoCs on defects. For each project we successively extract and add all the defects related to each module, to obtain the total number of defects in a project. In a similar way (addition of all modules LoC) we calculated the total size of the project in LoC. To obtain the DD per thousand lines of code, we multiply it by 1000.

For the data analysis we used both graphical representation and the mathematical calculation "percentage". The former gives us the immediate comparison and the later shows weight and the influence to characterize. Where applicable we also used the statistical methods for data analysis.

Concerning RQ1, we carried out the preliminary analysis of three types of projects (student, open source and close source) to categorize the projects in small, medium and large category using the k mean clustering algorithm. Afterward we find the percentage of distribution of very small modules in all categories of projects then we compare the percentage of modules and DD of projects. To find the statistical significant difference between the two groups we used the non parametric test Mann Whitney. For projects type student, we found 7 small projects, 7 medium and 3 large projects. For projects type open source, we found 6 small, 5 medium and 4 large projects. In projects type close source, we found 14 small, 3 medium and 6 large projects. Table 13 shows the three categories of software projects i.e. small, medium and large it shows that the large projects in term of size have lower DD in all types. We clustered the modules of each type of projects into 5 categories (very small, small, medium, large, and very large) based on size using the k mean clustering algorithm. We observed that the mean DD of very large modules is less than the DD of other categories of modules.

Table 13 Categories of software's in term of size

Type	**Category**	**No of Projects**	**Avg size in LoC**	**Avg DD [KLOC]**
Student	Small	7	4350	4.55
	Medium	7	12079	1.4
	Large	3	40984	1.2
Open Source	Small	6	28639	6.15
	Medium	5	114838	5.27
	Large	4	285061	1.22
	Small	14	12240	4.67

Close Source	Medium	3	70784	2.38
	Large	6	437029	1.6

Table 14 reports the DD of very small, small, medium, large and very large modules of each type of project. The preliminary observation shows that in all types of projects there is a smaller percentage of very large modules.

Concerning RQ 2: We first extract those modules that are defect free and then find out their percentage in the projects. For the second hypothesis we used a k mean clustering algorithm to cluster the projects based on DD and then performed the analysis observing the DD and percentage of defect free modules.

Concerning RQ 3: To find the module dependencies we used two metrics suggested by Martin[1] to calculate the module dependencies. The metrics we used are:

- Afferent Coupling (AC): The number of modules that depend on M.
- Efferent Coupling (EC): The number of modules that M depends upon.

We first extracted the dependencies of each module using the above defined two metrics. Afterwards we average the module dependencies of each project to find average module dependencies of a project. Then we performed the analysis observing, how DD of projects is affected by the module dependencies using a statistical measure regression analysis.

Table 14 DD of different categories of modules

Type	**Category**	**Range in Loc**	**No of Module**	**Avg DD**	**% of Module**
Student	V Small	1 – 205	640	7.22	67.4
	Small	206 – 565	204	1.6	21.4
	Medium	575 – 1250	78	1.15	8.2
	Large	1347 – 2781	22	0.77	2.3
	V Large	3211- 5924	5	1.48	0.5
Open Source	V Small	1-283	4732	33.2	73.6
	Small	284 – 850	1207	2.86	18.7
	Medium	853 – 1902	329	2.06	5.1
	Large	1940 – 4114	122	1.38	1.9
	V Large	4202- 3175	32	0.6	0.5
	V Small	1-132	30179	5.6	82
	Small	133-465	5209	1.39	14.2

[1] http://www.objectmentor.com/resources/articles/oodmetrc.pdf

Close Source	Medium	466-1263	1065	0.77	2.9
	Large	1266-2927	222	0.66	0.6
	V Large	2928 – 9878	40	0.61	0.1

Concerning RQ4: For the analysis we consider only top five projects with higher DD. Consequently we observe the distribution of percentage of modules in these projects that have higher DD.

3.4.2 Results

RQ 1: What is the distribution of modules on size in projects?

From Table 14 we found that in all three categories of projects the distribution of modules on size is very different. Hence the distribution of modules on size is very different in all categories of projects; we can reject our hypothesis H1.1, that the projects have not same distribution of modules on size. Considering hypothesis H1.2, we found that very large module have very small percentage in all types of projects. For projects (student, open source) it counts 0.5%, and for close source projects the percentage is 0.1%. Hence we can reject our hypothesis H1.2, that the projects have not more percentage of large modules. The secondary observation we obtain is the higher percentage of very small modules in all types of projects. It is above 60%. Figure 15 shows the box plots of categories of very small modules in all categories of projects. In all categories of projects, we found the distribution of very small modules of large sized project less than the small and medium sized projects. We notice that there is more variation of percentage of very small module in student projects as compared to open and close source projects.

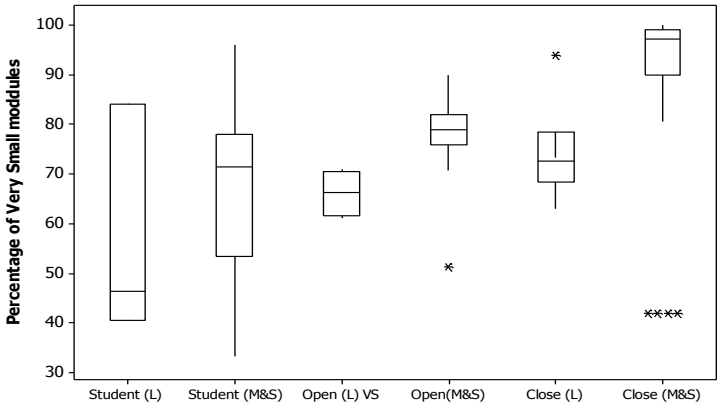

Figure 15 Percentage of Very Small modules in projects

Afterward we compare the distribution of very small modules and DD of large sized projects, with the distribution of very small modules and DD of small and medium sized projects. Table 15 reports the distribution of very small modules and DD of large sized projects compared to small and medium sized projects. It shows that there is a smaller percentage of very small modules in the larger sized projects compared to the small and medium sized projects of all types.

Table 15 Distribution of very small modules and DD in large sized projects vs. small and medium sized projects

Type	% of VS modules on Large Projects	DD of Large Projects	% of VS modules in Small & Medium Projects	DD of Small & Medium Projects

Student	57	1.23	67	2.98
Open Source	66	1.22	77.6	5.7
Close Source	74.3	1.6	91.2	4.27

We used Mann Whitney test to see the significant difference of percentage of very small modules between large sized projects with small and medium sized projects. We obtained p value = 0.0068 which is below the significant value 0.05. This confirms that the distribution of very small modules in large projects is different compared to small and medium size project in all types. Recalling Table 14 we had observed that large projects have lower DD and hereinafter we found that projects having more percentage of very small modules have higher DD. In particular, when we looked for a comparison between the large sized projects with small and medium sized projects, we found that large projects have a smaller percentage of very small modules and their corresponding DD is also lower than small and medium size projects. Similarly if the projects are constructed by the higher percentage of very small modules then the overall DD of the project is higher. Based on the empirical evidence one might be able to state that large projects have a smaller percentage of very small modules that would result in lower DD of large projects. However there could be many other factors affecting the DD of larger projects e.g. the larger project is normally taken more seriously, have rigorous testing etc.

RQ 2: What is the distribution of defect free modules in a project?

We found that the distribution of defect free modules in different types and categories of projects is very different. The difference of distribution allows us to reject our formulated hypothesis H2.1. Table 16 reports the distribution of defect free modules in each category of projects by type. Comparing overall by category, it shows that the large projects of all types have a higher number of defect free modules compared to medium and small category of projects and they make a larger percentage of code size as well. Similarly comparing the defect free modules by type, we found that close source projects have more percentage of defect free modules than open source and student projects. The defect free modules found in close source, open source and student projects are 81%, 59% and 69.3% respectively. Thus it shows that there is more quality attention given to the units of large projects compared to the medium and small sized projects. In addition the close source projects have more percentage of defect free modules compared to open source and student projects. This shows that close source projects have better construction quality.

Table 16 Distribution of defect free modules and percentage of code of defect free

Type	**Category**	**% of Defect Free Modules**	**% of code of Defect free modules**
Student	Small	59	38.1
	Medium	70.4	41.1
	Large	78.5	56
Open Source	Small	56	39.7
	Medium	54	35.4
	Large	67	53.7
	Small	81.4	70.7

Close Source	Medium	75.3	55.7
	Large	87.6	75.3

Concerning hypothesis H2.2, we performed the regression analysis to understand the relationship of defect free modules with the projects DD. For student and open source projects we obtain the R^2 = 38.4% and 54.7% respectively having a partial impact. However for close source projects we obtain R^2 = 5.9% having a very limited impact. To test our hypothesis H2.2 we cluster the projects based on the DD using k means clustering algorithm for all types of projects. Table 17 reports the DD and percentage of defect free modules in projects. In all types of projects we found that the projects having a higher percentage of defect free modules have lower DD compared to projects having a smaller percentage of defect free modules. Thus we accept our hypothesis H2.2 that project with lower DD have a larger percentage of the defect free modules.

Table 17 Projects Defect density Vs % defect free modules

Type	**No of Projects**	**Defect Density**	**% of Defect Free**
Student	4	5.8	41.2
	5	2.3	61
	8	1.2	84
Open Source	6	8.8	22.2
	5	4.54	58.2
	4	0.5	92.7
Close Source	13	4.3	82.3
	3	4.0	56.3
	7	1.93	93.28

RQ 3: How modules dependencies affect the projects DD?

In our data set the module dependency metrics are reported for 36 projects. We answer RQ3 considering those 36 projects, in which there are 17 students, 14 open source and 5 close source projects. Table 18 reports the average module dependencies of each category of project along with the DD. Observing the Table 18 we can't find any difference of module dependencies and the projects DD in different categories of projects. This makes us to perform our analysis only on the project types, to understand the impact of module dependencies on the projects DD.

Student projects: The obtained R^2 value is found to be 27.7%. There is a positive correlation but it has a limited practical impact. The regression equation is:

$$DD = 3.11 - 1.10AC + 0.825EC$$

Open source projects: The obtained R^2 value is found to be 10.6% having limited practical impact. The regression equation is:

$$DD = 10.9 - 0.582\ AC - 0.564\ EC$$

Close source projects: The obtained R^2 value is found to be 60.5% having a significant practical impact. The regression equation is:

$$DD = -\ 1.52 + 0.889AC + 0.012EC$$

Table 18 Projects module dependencies and DD

Type	Category	Project	Afferent coupling	Efferent coupling	Defect Density
Student	Small	7	2.5	4.0	4.55
	Medium	7	4.2	3.8	1.4
	Large	3	4.7	6.1	1.2
Open Source	Small	6	3.4	6.0	6.15
	Medium	5	5.3	5.0	5.27
	Large	3	6.6	6.56	4.24
Close Source	Large	5	2.6	12.2	1.01

Considering all the observation, we do not find any significant impact of modules dependencies on the projects DD in our sample of data. The relationship of module dependencies and DD of projects in type (student, open source) is found to be limited, however considering close source projects there is some practical impact. Thus we cannot accept or reject our formulated hypothesis H3.1 and H3.2.

RQ 4: How defect density of modules affects the defect density of projects?

To answer the RQ4, we selected top 5 projects from each type that have higher DD. Table 19 reports the projects having higher DD in each type. For projects (student, open source) the average percentage of module with higher DD is 51% and 59 % respectively. On the contrary for the close source projects the average percentage of module with higher DD is only 19.6%. Thus our formulated hypothesis is accepted for the students and open source projects that the projects with more DD have the more percentage of modules with higher DD; however the hypothesis H4.1 was not found true in the close source projects.

Table 19 Projects with higher DD vs. modules with higher DD

Type	Projects Avg DD	Projects DD	% of modules with higher DD	Avg % of module with higher DD
Student	5.53	11.5	56	51
		5.1	66.6	
		4.8	51.2	
		3.4	45.5	

		2.7	35.7	
Open Source	10.11	15.6	50	59
		13.0	91.7	
		11.3	74.3	
		6.1	60	
		4.4	20	
Close Source	6.69	8.0	35.2	19.6
		7.2	14.7	
		6.2	21.6	
		6.0	12.7	
		6.0	14	

3.4.3 Threats to validity

We discuss in this section validity threat using the classification proposed by [7].

Internal validity: In this study we only focus our observation towards some basic module attributes like size, quality, dependencies etc., to find their impact on projects DD. However there are many other module attributes that should have an influence on projects DD e.g. testing effort, testers experience and testing techniques etc. We acknowledge all other module attributes but for this study we only focus on the studied ones. Construct validity: In this research, we are dependent on the data logs provided by the *Promise data repository*. Surely, some potential concerns can be raised about the given data set e.g. how many modules may be left, how many defects may be raised and fixed before data collection and how many defects may not be recorded in logs etc. We consider this threat, but as the data set is publicly available and has been used in many previous studies, we believe its authenticity. External validity: In this study our findings are based on a small set of projects i.e. 54 software projects of different nature considering the impact of only a few attributes. Although this number is small but not negligible, this adds to the confidence by presenting some module attributes and their impact on projects DD.

3.4.4 Discussion and conclusion

This study has two folded outcomes, first how different the internal structural properties of three types (student, opens source, close source) of projects, secondly how module attributes affects the quality (in term of DD) of projects. The results show that the module attributes have some impact on projects DD. We found that the projects have not the same distribution of modules on size. In all types of projects there is a very small percentage of very large modules. The percentages of very small modules are less in large projects compared to medium and small sized projects. The empirical evidence shows that DD of the project increases with more percentage of very small modules (RQ1). The quality of the module has significant impact on the project quality (RQ2). We found that the module dependencies have not significant influence on the projects DD for student and open source projects; however module dependencies have some impact on close source projects DD. Having only 5 close source projects for the analysis does not add much confidence in the results (RQ3). We found that it is not always true that modules with higher DD would result in higher DD of projects as it is found true for student and open source projects but not for close source projects (RQ 4). The projects

DD can be predicted by using the modules attributes (percentage of very small and defect free modules) as the significant relationship between projects DD and attributes is found (RQ5). Authors want to give some suggestions to practitioners aimed to assess their projects based on our empirical findings.

(1) More percentage of very small modules affect negatively to the projects DD.
(2) Module compositions have not much effect on projects DD.
(3) Modules DD may not always be significant to predict the projects DD.
(4) The module attributes (% of very small modules, % of defect free modules) can be used to access the projects DD.

These are based on the validated data set of projects that have been used previously and available for research purposes at Promise *Repository*. The artifacts of this study also give direction to the researchers that different types of projects have different internal structure and their characteristics are quite different from each other. In addition it also shows the importance regarding the previously conducted studies (where mostly the relationship of different module attributes has been shown) but their impact on overall project was unknown. Therefore, we recommend researcher to study more module attributes and their impact on projects. We think that the main limitation of this study is the very few projects and module attributes under study. More attributes like (testing efforts, development process, testing techniques, experience of the team) should be studied to find their impact on the projects DD.

3.5 Data set: promise repository 1, concerning complexity metrics:

3.5.1 Research design

In this section, we present the research question and the data set. One research question is formulated for this research.

RQ1: Do complexity metrics have an effect on defects?

We selected the last releases of 38 software projects constituting 27,734 modules from the "*Promise data repository"* having the required metrics freely available for research evaluation purposes [5]. The data set contains 6 proprietary software projects, 15 open source software projects and 17 are academic software projects that were developed by the students. We downloaded the CVS files and found 18 complexity metrics defined in Table 20. The values of complexity metrics were available against the defects for every module. Table 20 shows the metrics used in the study.

Table 20. The metrics used in the study

The metrics suggested by Chidamber and Kemerer [21] are.
Weighted Methods per class (WMC): WMC is the number of methods defined in each class.
Depth of Inheritance Tree (DIT): It is the measure of the number of ancestors of a class.
Number of Children (NOC): It is the measure of a number of direct descendants of the class.
Coupling between Objects (CBO): It is the number of classes coupled to a given class.
Response for a Class (RFC): It is the measure of different methods that can be executed when an object of that class receives a message.
Lack of Cohesion in Methods (LCOM): It is the number of pairs of member functions without shared instance variables, minus the number of pairs of member functions with shared instance

variables.

Henderson Sellers defined one complexity metric [23].

Lack of cohesion in methods (LCOM3): According to study [23] LCOM3 is defined as.

$$LCOM3 = \frac{\left(\frac{1}{a}\sum_{j=1}^{a}\mu(A_j)\right) - m}{1-m}$$

m - number of methods in a class; a - number of attributes in a class; μ(A) - number of methods that access the attribute A.

Bansiya and Davis [22] suggested the following quality metrics suite.

Number of Public Methods (NPM): It counts all methods in a class that are declared as public. This metric is also known as Class Interface Size (CIS).

Data Access Metric (DAM): It is the measure of the ratio of the number of private (protected) attributes to the total number of attributes declared in the class.

Measure of Aggregation (MOA): It is the count of the number of class fields whose types are user defined classes.

Measure of Functional Abstraction (MFA): It is the ratio of the number of methods inherited by a class to the total number of methods accessible by the member methods of the class.

Cohesion among Methods of Class (CAM): It computes the relatedness among methods of a class based upon the parameter list of the methods.

Data Access Metric (DAM): It is the measure of the ratio of the number of private (protected) attributes to the total number of attributes declared in the class.

Measure of Aggregation (MOA): It is the count of the number of class fields whose types are user defined classes.

Measure of Functional Abstraction (MFA): It is the ratio of the number of methods inherited by a class to the total number of methods accessible by the member methods of the class.

Cohesion among Methods of Class (CAM): It computes the relatedness among methods of a class based upon the parameter list of the methods.

Tang et al [53] extended the Chidamber & Kemrer metrics suite focusing on the quality.

Inheritance Coupling (IC): It provides the number of parent classes to which a given class is coupled.

Coupling Between Methods (CBM): It measures the number of new/redefined methods to which all the inherited methods are coupled.

Average Methods Complexity (AMC): It measures the average method size for each class.

Following two metrics were suggested by Martin [24].

Afferent Coupling (Ca): It is the number of classes that depend upon the measured class.
Efferent coupling (Ce): It presents the number of classes that the measured class is depended upon.
The one metric was suggested by McCabe [20].
McCabe's Cyclomatic Complexity (CC). It is equal to the number of different paths in a method (function) plus one. It is defined as CC = E-N+P; where E is the number of edges in the graph, N is the number of nodes of the graph; P is the number of connected components. It is only suitable for the methods; therefore it is converted to the class size metrics, by calculating the arithmetic mean of the CC value in the investigated class.

3.5.2 Results

We carried out the preliminary analysis to identify the three categories of software projects small, medium and large using the K mean clustering algorithm. We found 24 software projects in the small category, 7 software projects in medium and 7 software projects in the large category. The average defects found in the small category of software projects are 52.5, for the medium category of software projects it is 519.14 and 508.2 defects for a large category of software projects. Table 21 shows the three categories of software projects i.e. small, medium and large.

Table 21Categories of software's in term of size

Category	**No**	**Avg defects**	**Avg size**
Small [1-60KLoC]	24	52.5	17241 LoC
Medium [60–300 KLoC]	7	519.14	140743 LoC
Large [above 300KLoC]	7	508.2	427354 LoC

RQ 1: Do complexity metrics have an effect on defects?

We attempted to find the linear correlation between the complexity metrics and defects. We selected Pearson correlation coefficient which best suited to find the linear relation between the two variables. In no case we found the strong correlation among complexity metrics and defects. To study any possible relationship of defects with complexity metrics, we cluster the modules into three categories based on the values, using the K mean clustering algorithm. In order to understand the clusters behavior, we performed the preliminary analysis of the identified clusters for each complexity metric based on the project category. We found three types of behaviors of complexity metrics and grade them as effective, untrustworthy and not useful indicators of defects.

Effective indicators:

We extract those complexity metrics where higher values result in higher defects. We called these complexity metrics effective indicators of defects and these metrics exhibit the below phenomenon. Table 22 reports the complexity metrics, effective indicators of defects in small, medium and large projects.

High Complexity → High Defect

Table 22 Complexity metrics effective indicators of defects

Project type	**Complexity metrics**

Small	LCOM
Medium	WMC, CBO, RFC, CA, CE, NPM, DAM, MOA, IC, Avg CC
Large	WMC, CBO, RFC, CA, NPM, AMC

Untrustworthy indicators:

We classify those complexity metrics that have no fixed criterion of increase in defect with the increase in complexity metric value. We called these complexity metrics untrustworthy indicators of defects. Table 23 reports the complexity metric untrustworthy indicator of defects in small, medium and large category of projects. For the untrustworthy indicators we observe two different behaviors of complexity metrics. (a) Medium complexity value resulted in high defects: Medium Complexity → High Defects. (b) High complexity values resulted in high defects but corresponding medium cluster value resulted in lower defects: High Complexity → High Defects, AND Medium Complexity → Low Defects.

Table 23 Complexity metrics untrustworthy indicators of defects

Project type	**Complexity metrics**
Small	WMC, NOC, CBO,RFC, CE, Avg CC
Medium	DIT, NOC, LCOM, CBM, AMC
Large	DIT, NOC, LCOM, CE, LCOM3, DAM, MOA, MFA, CAM, IC, Avg CC

Not useful indicators:

We classify those complexity metrics where smaller values resulted in high defects, as not useful indicators of defects. Table 24 reports the complexity metrics not useful indicator of defects in small, medium and large projects. These complexity metrics exhibit the phenomenon: Low Complexity → High Defects

Table 24Complexity metrics not useful indicators of defects

Project type	**Complexity metrics**
Small	WMC, DIT, CA, NPM, LCOM3, DAM, MOA, CAM, IC, CBM, AMC
Medium	LCOM3, MFA, CAM
Large	CBM

Hypothesis testing

For hypothesis testing, we only consider the effective indicator of complexity to verify that the distribution of defects among high, medium and low complexity. We did not perform the analysis on the untrustworthy and not useful indicators because it does not seem to be very meaningful. We take support of statistical hypothesis testing to confirm the difference of defect in three categories of complexity metrics values i.e. high, medium and low. Using statistical techniques, we will test the null hypothesis H_0. We will accept and reject it based on the favor of the alternative hypothesis.

- H_0: There is no significant difference of defects among high, medium and low complexity of effective indicators.
- H_1: There is a significant difference of defects among high, medium and low complexity of effective indicators.

Selection of statistical test

We first examined the distribution of the samples to choose the appropriate statistical test for the analysis. We applied the Ryan-Joiner test for the normality check and found that for every sample (p - value <0.01). The results showed that none of the sample under study has a normal distribution of data. This made us to select the non parametric test for the hypothesis testing. According to the recommendation for not normal samples, we chose non parametric test Kruskal-Wallis test for differences between three or more samples. We compare the defects of high, medium and low complexity cluster of each effective metric. In each category of projects the obtained p value was found less than 0.05 meaning there is a significant difference of defects among high, medium and low complexity value of effective indicators.

Threats to validity

This section discusses the validity threat as classified and proposed by the study [7]. As for construct validity, we collected the CVS logs from the Promise research data repository. Although we have much confidence in the correctness and accuracy of the provided data but still we have no control to decide that up to which level the data is authentic e.g. how many module's data may be left to record, correctness of the measurement of complexity metric values etc. As for external validity, our findings are based on the large data set of modules i.e. 27,734 of 38 software projects. Although this number is not small but still there can raise some concerns on the generalization of the findings when the projects are categorized into small, medium and large. We have 24 projects from small, 7 projects are from medium and 7 projects are from large category which is fairly small samples.

3.5.3 Conclusion

The findings have important implications as they are based on the complete set of complexity metrics belonging to one particular project. Similarly 38 such projects were selected which have all the complexity metrics available and then combined to perform the analysis collectively. The artifact of this study is very vital and beneficial for both researchers and practitioners. The primary contribution of this research is the identification of having no linear relation of any complexity metrics with defects. However based on the complexity metrics high, medium and small value clusters we find that there are some complexity metrics which higher values resulted in a higher number of defects. These complexity metrics are called effective indicators of defects. Consequently the complexity has an effect on defects but not as large as one might expect. The researchers can use the effective complexity metrics for the predicting and estimating of defects and can use in predictive models. The statistical analysis adds confidence that there is a quite significant difference among the defects of effective complexity metrics high, medium and low values. The practitioners can assess their projects' quality based on the effective complexity metrics. The categorization of projects is quite useful as it gives practitioners a view to select the appropriate effective complexity metric when assessing their project based on size.

3.6 Data set: NASA 93

For the study we used the software projects from the data set NASA-93 that has been used in many previous studies. The data set includes 93 software projects from different centers of NASA.

3.6.1 Data collection

For the study we used the software projects from the data set NASA-93 that has been used in many previous studies. The data set includes 93 software projects from different centers of NASA and Table 25 shows the types and number of projects. The mean project size was approximately 94 Kloc and projects are developed using C programming language. The data set is publicly available under the Promise Software Engineering Repository [5]. The data set was primary organized for the cost estimation of software projects.

Table 25 Project types

Project Type	Number of Projects
Avionics Monitoring	30
Mission Planning	20
Avionics	11
Monitor and Control	8
Operating System	4
Simulation	4
Real Data Processing	3
Data Capture	3
Application ground	2
Batch Processing	2
Science	2
Utility	2
Launch Processing	1
Communication	1

3.6.2 Product complexity (PC):

Product Complexity is assessed by considering the five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations [54]. The product complexity rating was done by subjective weighted average of these areas. Table 53 shows that how complexity rating is characterized.

3.6.3 Data analysis method

For the data analysis we report the descriptive statistics and where applicable we also used the statistical techniques. We used interval plots that illustrate both measure of central tendency and variability of the data to present the data distribution. In the interval plots on the y-axis the defect density figures are reported for the software quality graphs and lines of code developed per month are reported on the y-axis for the development productivity graphs. The average software quality is found to be 41.42 defects per thousand lines of code as provided in Table 26 shows the descriptive statistics of the software quality in term of defect density. Table 27 shows the categorization of product regarding the product complexity.

Table 26 Dependent variables descriptive statistics

Dependent Variable	N	Mean	Std Deviation	Minimum	Maximum
Software Quality (defects per thousand lines of code)	93	41.42	12.55	24.42	93.25

Table 27 project categorization regarding complexity

Product Complexity (PrC)	Extreme High	5
	Very High	17
	High	58
	Medium	10
	Low	3

Impact of Product Complexity (PrC) on Quality of software

The data set contain 5 projects having extreme high complexity, 17 projects having very high complexity, 58 projects having high complexity, 10 projects having medium complexity and 3

projects having low complexity. The summary descriptive statistics are reported in Table 28 divided by type. Figure 16 shows that there is a high variation of software quality in the projects having very high PrC, Considering the Figure 16 we can visually analyze the following trends.

- Extreme High PrC →Lower than average software quality
- Very High PrC → Lower than average software quality
- High PrC → Higher than average software quality
- Medium PrC → Higher than average software quality
- Low PrC → Higher than average software quality

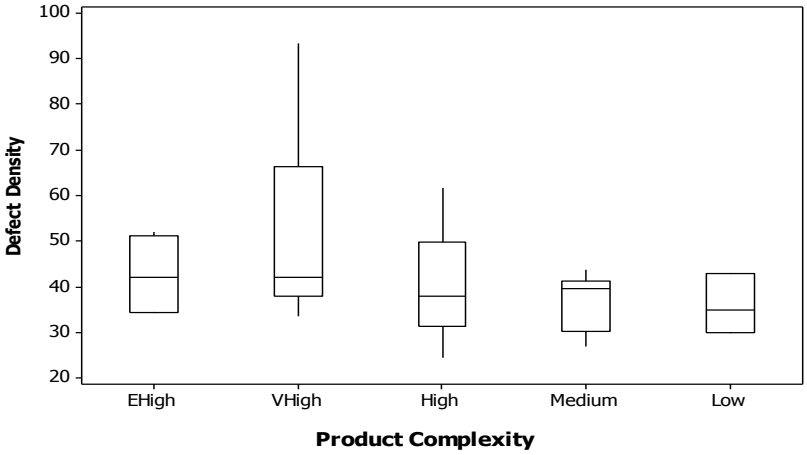

Figure 16 Box plot of product complexity against the quality of software

Table 28 Descriptive statistics for product complexity

Product Complexity	No	Quality	Std Deviation
Extreme High	5	42.62	8.45
Very High	17	50.84	18.57
High	58	39.71	10.43
Medium	10	36.38	6.32
Low	3	35.90	6.59

We performed Mann-Whitney test for the pair wise comparison between two neighboring pairs of product complexity to find the significant difference of software quality and development productivity between them. Concerning software quality and development productivity we made four pair's (EHigh, VHigh), (VHigh, High), (High, Medium) and (Medium Low).

Concerning software quality for every pair except (VHigh, High) we obtained the p-value above the α threshold. Similarly concerning development productivity for every pair except (VHigh, High) we obtained the p-value above the α threshold. For the pair (VHigh, High) we obtained the p-values below the α threshold. Therefore the only significant difference of software quality and development productivity is found in the pair (VHigh, High).

Product complexity has partial impact on the software quality. The partial impact of product complexity (only at higher level) shows that product complexity start influencing software quality and development productivity after some complexity threshold.

Chapter 4

4. The Impact of People attributes on Software Quality

Towards the Impact of
People, Process and Product
Attributes on the Software Quality
and Development Productivity, Under
Review at: Springer Software Quality
Journal (2014).

S. M. A. Shah, M. Morisio and M. Torchiano

4.1 Introduction

Many researchers studied the impact of people related attributes like experience, skills and capability and their impact on software quality. Humphrey found that the average defect injection rate for the developers is 120 defects per KLOC, or one defect in every eight lines of code. He observed the high variation of injection of defects among the developer where 10% of the developers injected 29 defects/KLOC and the top 1% injected 11defects/KLOC" [55]. Acuna et al. analyzed the relationships between personality, team processes, task characteristics, product quality and satisfaction in software development teams. They found that the teams exhibit a significant positive correlation between the personality factor extraversion and software product quality [56]. Hazzan and Hadar found evidence that human aspects are the source of the majority of problems associated with software development projects [57]. Gorla and Lin surveyed 112 Information systems project managers and found that organizational attributes are more important than technical attributes impacting software quality in IS projects [58].

Krishnan et al. examined the relationship of product size, personnel capability, software process, usage of tools, and higher front-end investments on productivity and conformance quality. The study identified several quality drivers in software products e.g. higher personnel capability, deployment of

resources in initial stages of product development (especially design) and improvements in software development process factors are associated with higher quality products [59]. Shendil and Madhavji identified that the individual developers productivity could be improved as a consequence of (i) a growing stock of knowledge and experience gained by repeatedly doing the same task (ii) due to technological and training programs supported by the organization [60].

4.2 Independent variables

We studied 5 independent variables (Analyst Capability, Programmer Capability, Application Experience, Platform Experience, Language and Tool Experience) and their impact on the dependent variables "software quality".

4.2.1 Analyst capability (AC)

Analysts are the persons responsible for working on requirements, high level design and detail design. To consider the analysts capability, major attributes like analysis and design ability, thoroughness and efficiency, and the ability to communicate and cooperate are assessed for the rating. Table 29 shows the analysts capability in ordinal scale, the analysts that fall in the 15th percentile are rated very low and those that fall in the 95th percentile are rated as very high [54].

Table 29 Analyst / Programmer Capability levels

	Very Low	**Low**	**Medium**	**High**	**Very High**
Analyst | Programmer Capability	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile

4.2.2 Programmer capability (PC)

The programmer capability is evaluated as a team rather than as individuals. The attributes considered in the rating are ability, thoroughness and efficiency, and the ability to communicate and cooperate. The criteria rating for the programmer capability was same as adopted for the analyst's capability as shown in Table 29. Programmers that fall in the 15th percentile are rated very low and those that fall in the 95th percentile are rated as very high [54].

4.2.3 Application experience (AE)

Application experience rating is dependent on the level the project team developing the system. Table 30 shows the rating of application experience in ordinal scale. The experience of 2 month is characterized as very low where the experience of 6 years is characterized as very high as shown in Table 30.

4.2.4 Platform experience (PE)

The platform experience is the rating of working with powerful platforms, including database, networking, graphical user interface, and distributed middle ware capabilities. Same procedure of rating is followed; 2 month is characterized as very low where the experience of 6 years is characterized as very high as shown in Table 30.

4.2.5 Language and tool experience (LTE)

The language and tool experience is the rating of working with programming languages and the assistive tools for the developing a software system. This rating is important as experience in programming with a specific language and supporting tools also affect the quality. Same procedure of rating is followed 2 month is characterized as very low where the experience of 6 years is characterized as very high as shown in Table 30.

Table 30 Experience levels

	Very Low	Low	Medium	High	Very High
Application | Platform | Language and Tool Experience	2 months	6 months	1 year	3 years	6 years

The categories and number of projects on each category is given in Table 31.

Table 31 Independent variables categories

Independent Variables	Categories	N
Analyst Capability (AC)	Very High	10
	High	51
	Medium	32
Programmer Capability (PC)	Very High	10
	High	39
	Medium	44
Application Experience (AE)	Very High	12
	High	46
	Medium	34
	Low	1
Platform Experience (PE)	High	22
	Medium	14
	Low	53
	Very Low	4
Language and tool Experience (LTE)	High	69
	Medium	14
	Low	6
	Very low	4

4.2.6 Correlation among the variables

To find the relationship and possible interaction between the independent variables we used the Kendall's rank coefficient tau–b. Kendall's rank coefficient tau–b provides a distribution free test of independence and a measure of the strength of dependence between two ordinal variables. It can be seen in Table 32 that there is only one correlation near 60% (0.595), which is between PC and AE.

Table 32 Correlation among the variables

Attributes	AC	PC	AE	PE	LTE
AC	1	0.40	0.53	0.02	0.24
PC		1	0.59	0.14	0.16
AE			1	-0.01	0.11
PE				1	0.39
LTE					1

4.3 Results

Impact of Analyst Capability on software Quality

The data set contain 10 projects developed by analyst having very high capability, 51 projects developed by analyst having high capability and 32 projects developed by analyst having medium capability. The summary descriptive statistics are reported in Table 33 divided by type. Figure 17

shows that there is a high variation of software quality of projects developed by analyst having medium capability. Considering the Figure 17 we can visually analyze the following trends.

- Very High AC → Lower than average software quality
- High AC → Higher than average software quality
- Medium AC → Lower than average software quality

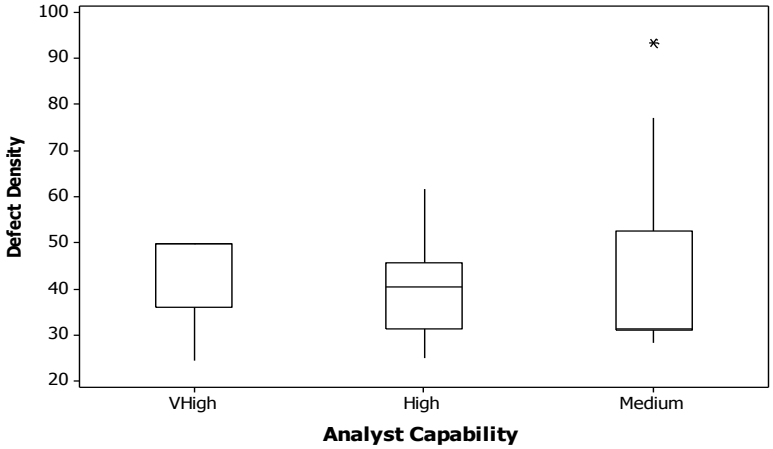

Figure 17 Box plot of analyst capability against quality of software

Table 33 Descriptive statistics for analyst capability

Analyst Capability	No	Quality	Std Deviation
Very High	10	42.79	9.89
High	51	40.24	9.27
Medium	32	42.86	17.18

We performed Mann-Whitney test for the pair wise comparison between two neighboring pairs of analyst capability to find the significant difference of software quality and development productivity between them.

Concerning software quality we made the two pairs (VHigh, High) and (High, Medium). For every pair considering software quality we obtained the p-value above the α threshold, indicating that there is no significant difference of software quality between the pairs.

Impact of Programmer Capability on software Quality

The data set contain 10 projects developed by programmer having very high capability, 39 projects developed by programmer having high capability and 44 projects developed by programmer having medium capability. The summary descriptive statistics are reported in Table 34 divided by type. Figure 18 shows that there is a high variation of software quality in projects developed by programmers having medium capability. Considering the Figure 18 we can visually analyze the following trends.

- Very High PC → Higher than average quality
- High PC → approx average quality
- Medium PC → Lower than average quality

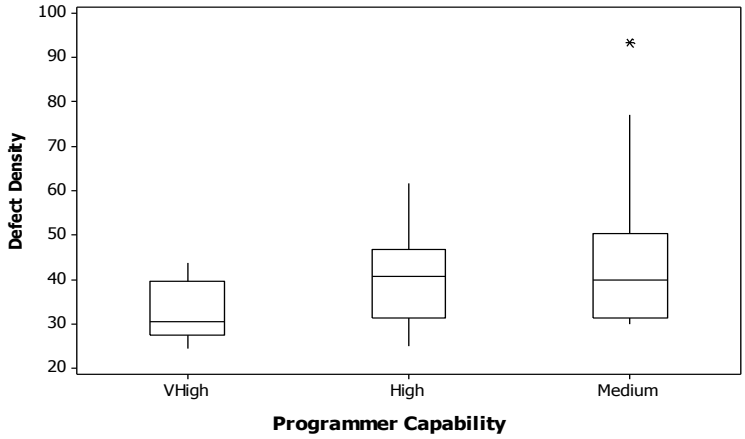

Figure 18 Blox plot of analyst capability against quality of software

Table 34 Descriptive statistics for programmer capability

Programmer Capability	No	Quality	Std Deviation
Very High	10	33.15	7.06
High	39	40.97	9.45
Medium	44	43.69	15.04

We performed Mann-Whitney test for the pair wise comparison between two neighboring pairs of programmer capability to find the significant difference of software quality between them. Concerning software quality we made the two pair (VHigh, High) and (High, Medium).

Concerning software quality only the pair (VHigh, High) we obtained the p-value below the α threshold. Therefore there is only a significant difference of software quality between the pair (VHigh, High).

Impact of Application Experience on software Quality

The data set contain 12 projects developed considering very high level of application experience, 46 projects developed considering high level of application experience, 34 projects developed by considering medium level of application experience and 1 project is developed considering low level of application experience. For the analysis we do not take the one data point of project developed considering low application experience. The summary descriptive statistics are reported in Table 35 divided by type. Figure 19 shows that, there is a high variation of software quality in projects considering medium level of application experience. Considering the Figure 19 we can visually analyze the following trends.

- Very High AE → Higher than average software quality
- High AE →Lower than average software quality
- Medium AE → Average software quality
- Low AE → Lower than average software quality

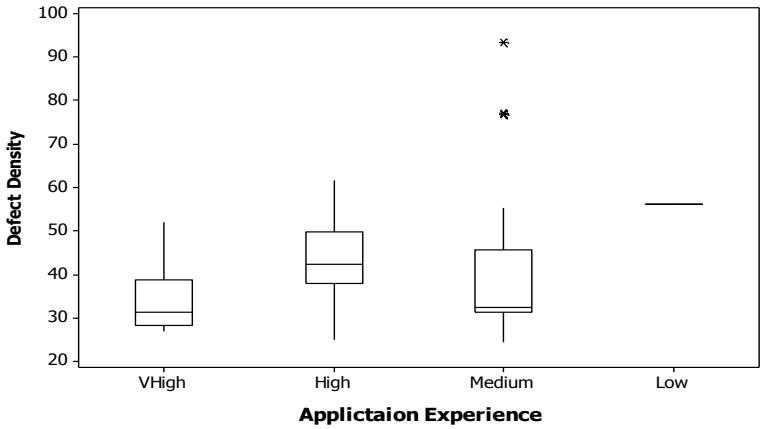

Figure 19 Box plot of application experience against software

Table 35 Descriptive statistics for application experience

Application Experience	No	Quality	Std Deviation
Very High	12	33.97	7.64
High	46	43.20	8.96
Medium	34	41.20	16.75
Low	1	56.00	*

We performed the pair wise comparison by mean of Mann-Whitney test between two neighboring pairs of application experience to find the significant difference of software quality between them. Concerning software quality we made two pairs (VHigh, High) and (High, Medium).

Concerning software quality for both pairs we obtained the p-value below the α threshold. Therefore we can indicate that there is a significant difference of software quality between the pair (VHigh, High) and (High, Medium).

Impact of Platform Experience on software Quality

The data set contain 22 projects developed considering high level of platform experience, 53 projects developed considering medium level of platform experience, 14 projects developed by considering low level of application experience and 4 projects are developed considering very low level of application experience. The summary descriptive statistics are reported in Table 36 divided by type. Figure 20 shows that there is a high variation of software quality in projects considering low platform experience. Considering the Figure 20 we can visually analyze the following trends.

- High PE = Approx average software quality
- Medium PE = Higher than average software quality
- Low PE = Lower than average software quality
- Very Low PE = Lower than average software quality

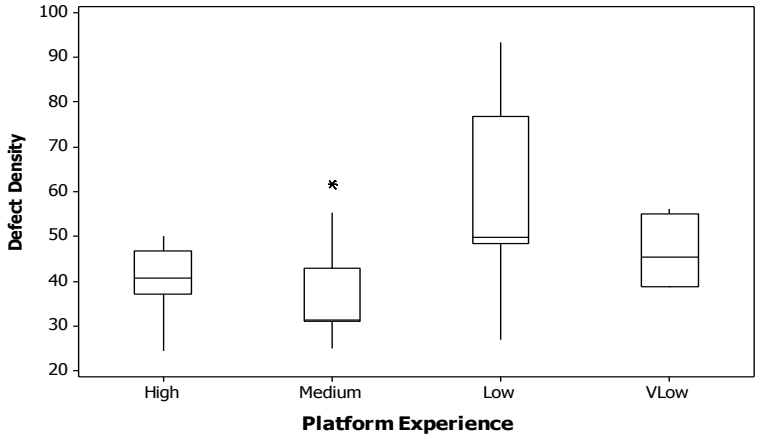

Figure 20 Box plot of platform experience against software quality

Table 36 Descriptive statistics for platform experience

Platform Experience	No	Quality	Std Deviation
High	22	40.84	6.59
Medium	53	37.31	9.86
Low	14	56.45	17.78
Very Low	4	46.44	8.88

We performed the pair wise comparison by mean of Mann-Whitney test between two neighboring pairs of platform experience to find the significant difference of software quality between them. Concerning software quality we made three pairs (High, Medium), (Medium, Low) and (Low, VLow).

Concerning software quality for all pairs except (Low, VLow) we obtained the p-value below the α threshold. Therefore we can indicate that there is a significant difference of software quality between the pair (High, Medium) and (Medium, Low).

Impact of Language and Tool Experience on software Quality

The data set contain 69 projects developed considering high level of language and tool experience, 14 projects developed considering medium level of language and tool experience, 6 projects developed by considering low level of language and tool experience and 4 projects are developed considering very low level of language and tool experience. The summary descriptive statistics are reported in Table 37 divided by type. Figure 21 shows that there is a high variation of software quality of projects considering low language and tool experience. Considering the Figure 21 we can visually analyze the following trends.

- High LTE = Higher than average software quality
- Medium LTE = Lower than average software quality
- Low LTE = Lower than average software quality
- Very low LTE = Lower than average software quality

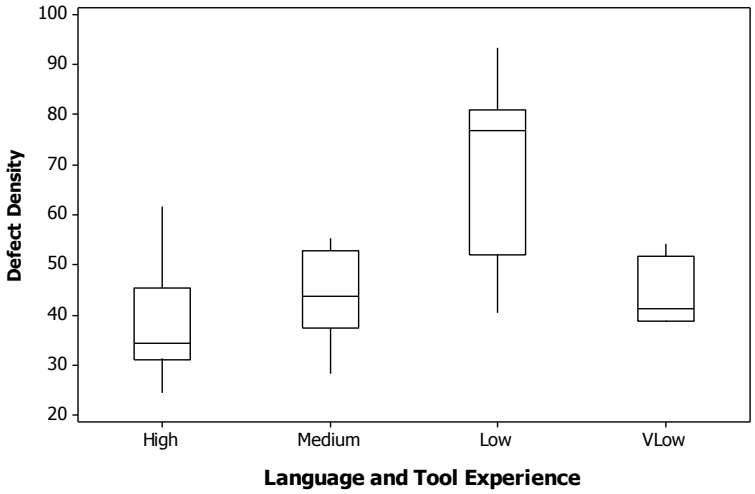

Figure 21 Interval plot of language and tool experience against software quality

Table 37 Descriptive statistics for language and tool experience

Language and tool Experience	No	Quality	Std Deviation
High	69	38.20	253.0
Medium	14	44.28	51.2
Low	6	70.07	206.9
Very Low	4	43.88	194.3

We performed the pair wise comparison by mean of Mann-Whitney test between two neighboring pairs of language and tool experience to find the significant difference of software quality between them. Concerning software quality we made three pairs (High, Medium), (Medium, Low) and (Low, VLow).

Concerning software quality for all pairs we obtained the p-value below the α threshold. Therefore we can indicate that there is a significant difference of software quality between the pair (High, Medium), (Medium, Low) and (Low, VLow).

Table 38 reported the summary of impact of independent variables on the dependent variables. Here in Table 38 the "partial impact" means at least some levels of independent variable have statistical significant impact on the dependent variables. The "significant impact" means all levels of independent variables have statistical significant impact on the dependent variables. The "No impact" means that levels of independent variables have no statistical significant impact on the dependent variables.

Table 38 Summary observation of Impact of independent variables on dependent variable

Variables	Software Quality
Analyst Capability	No impact
Programmer Capability	Partial impact
Application Experience	Significant Impact
Platform Experience	Significant Impact
Language and tool Experience	Significant Impact

4.4 Discussion

Summary of Results: The analysis of 93 projects allows us to understand the impact of different attributes on the software quality. The results are important for both researchers and practitioners. The practitioners can use the results to understand the impact of any attribute on the software quality. In order to attain increase of software quality in their projects; they must put sufficient attention to the identified influencing attributes. The researchers can use the results in order to find the reasons that why one particular attribute has an impact on the software quality or not. We can summarize the following evidence that we found after analyzing the particular data set.

- Analyst capability has no significant impact on the software quality.
- Programmer capability has partial impact on software quality.
- Application experience has significant impact on software quality
- Platform experience and language and tool experience has significant impact on the software quality

Analyst capability has no impact on the software quality. Typically there are about 85% of defects originates from the requirements [61]. Having this high percentage of defect introduction does not able to classify the impact of analyst capability on the software quality.. Hence the impact of analyst's capability on software quality is further subject to investigation.

Programmer's being the main workforce of software development, and software quality highly dependent on them. Though, in this study we found partial impact of programmer's capability on the software.

4.5 Conclusion

This study analyzed the attributes that are related to people, and their impact on the software quality using the statistical significant evidences. The evidences show that there are some attributes that have significant impact on the software quality. Therefore practitioners can benefit from the findings to attain increase in software quality by selecting influencing attributes.

Chapter 5

5. The Impact of Process attributes on Software Quality

Published in Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM '12).*
Towards the Impact of People, Process and Product Attributes on the Software Quality and Development Productivity, Under Review at: Springer Software Quality Journal (2014).*
Published in IEEE IT Professional Journal (2013).**
Published in Journal of Software Evolution and Process (2013)***

(S. M. A. Shah, M. Morisio, and M. Torchiano)*
(S. M. A. Shah, M. Torchiano, A. Vetro, and M. Morisio)**
(S. M. A. Shah, C. Gencel, U. S. Alvi, and K. Petersen)***

5.1 Introduction to software process

The relationship between quality of the product and quality of the process is a key issue in all the engineering disciplines. In software engineering the attention to process started to be widespread

from the 90's thanks to the work of Watts Humphrey, who applied to software engineering process concepts developed in other disciplines [62]. The CMMI [63], main result of this work, proposes a capability assessment model and an improvement path for organizations.

On a parallel track, a large variety of software processes have been proposed over the years, starting from the Waterfall model [64] up to the PSP [65], TSP [66], RUP [67], MSF [68] and Agile [69]. Behind many of these proposals stands the assumption that more sophisticated processes will lead to higher quality products. This view has been later challenged by the agile movement that insists more on *low ceremony* approaches. Our research is focused on finding empirical evidence of the effect of process choices on product quality. Product quality can be measured in several ways: reliability at function or system level, user satisfaction, defect density. In this study we take the pragmatic, view of quality in terms of defect density. Defect density (DD) is defined as the total number of found defects divided by the size of the software [70].

5.1.1 Related work

The Capability Maturity Model Integration (CMMI) has been widely adopted as a guideline to improve the overall software quality. There is research evidence that a higher CMMI level is linked to better quality [71][72][73][74]. Li et al, highlighted the experience of Neusoft Group, where defect density decreased from 0.85 defects per KLoC in 2000 to 0.1 defects per KLoC in 2005 as a result of CMMI adoption [75]. According to Jones [76] the CMMI 1 to CMMI 5 levels has 0.75, 0.44, 0.27, 0.14, and 0.05 delivered defects per KLoC.

To date, there are various software development processes and an even larger numbers of hybrids in use. Software development process research literature contains different claims for the quality [77][78][79][80][81]. As reported in [82], if a well structured TSP is used, it has a positive impact decreasing the DD, in particular, and increasing the software quality in general. Abrahamsson and Koskela obtained the system defect density of 1.43 defects per KloC from a controlled case study on extreme programming in Agile setting [83].

The study [84] reported the IBM experience of the Agile software process that reduced the DD and increased the overall quality. The survey conducted by Ramasubbua and Balan [85] on 112 projects showed that the combination of CMMI 5 with Agile had a significant and mostly positive impact on the project DD. Mohan et al. suggested the use of RUP to achieve increased reliability with higher productivity and lower defect density [86]. Bhat and Nagappan observed a significant increase in quality of two Microsoft projects developed using TDD (Test Driven Development) compared to same projects developed in a non-TDD fashion [87]. In one review study, Mitchell and Seaman [88] performed a systematic review comparing Waterfall vs. Iterative and Incremental development but the data set did not demonstrate any difference in quality.

Jones et al. [89] classify CMMI not assessed, CMMI level 1 and Waterfall projects under the low quality category. For average quality, they classify CMMI 1, 2 and Agile projects. For high quality, they classify CMMI level 3, 4, 5, Hybrid process, TSP and RUP projects. Li et al, highlighted the experience of Neusoft Group, where defect density decreased from 0.85 defects per thousand lines of code (kloc) in 2000 to 0.1 defects per kloc in 2005 as a result of CMMI adoption [75]. Jones et al. classify CMMI not assessed, CMMI level 1 and Waterfall projects under the low quality category. For average quality, they classify CMMI 1, 2 and Agile projects. For high quality, they classify CMMI level 3, 4, 5, Hybrid process, TSP and RUP projects [89]. Subramanian et al. showed that CMM levels do associate with IS implementation strategies and higher CMM levels relate to higher software quality and project performance. [90]. Tufail and Malik observed that the adoption of an agile process improved the quality of the software produced by the software house by reducing the percentage of serious errors and defects and by increasing the ratio of passed to failed test cases [91]. Li et al. compared software quality assurance processes and software defects of the project between a 17- month phases with a plan-driven process, followed by a 20-month phase with Scrum. The results of the study did not show a significant reduction of defect densities or changes of defect profiles after Scrum was used. [92].

Rubin studied the impact of improvement in the software process on software engineering productivity and quality in 300 organizations and found that those that had embarked on significant process improvement efforts were substantially gained in productivity and quality [93]. Sussy et al. presented a case study that describes introduction of team software process to proved that training in team software process had a positive impact on getting better estimations, reducing costs, improving productivity, and decreasing defect density [82]. Sison compared the programs (small, large) developed by students using pair programming technique. The results suggested that pair programming increase software quality without decreasing productivity [94]. Rafique and Misic performed a systematic meta-analysis of 27 studies that investigate the impact of Test-Driven Development (TDD) on external code quality and productivity. The results indicate that, in general, TDD has a small positive effect on quality but little to no discernible effect on productivity [95].

5.1.2 Data set: Capers Jones & Associates LLC

We selected 61 software projects having available required metrics for our analysis. Data on these projects was kindly provided by Capers Jones & Associates LLC (http://www.namcook.com/) in October 2011. The company uses these projects metrics as a reference for quality prediction and benchmarking.

5.1.2.1 Research design

In this section, we present the research questions, the data set and the metrics used.

Research questions

Two primary research questions were formulated for this research.

RQ1: Do different CMMI levels affect defect density?

The goal here is to first characterize the defect density for different CMMI levels, and then check if these levels are significantly different.

RQ2: Do structured software processes affect defect density?

The goal is to characterize the effect of a structured software process on defect density.

Metrics

The CMMI levels are expressed on an ordinal scale 1 to 5. However, the dataset contains also companies that were not assessed. These appear as having level 0. In practice we have two merged scales: a nominal scale (assessed, not assessed) and an ordinal scale 1 to 5, only for assessed companies.

The software process is expressed as a nominal scale, first by (structure and without structure) and then by structured types (TSP, PSP, etc).

Defect density is calculated by dividing the number of found defects, by the code size in LOCs. When size is expressed in terms of function points as defined by the International Function Point Users Group (IFPUG), it is converted to KLoCs using the logical code statements by using the proprietary method of conversion [96].

Analysis method

To answer our research questions we adopted both visual and statistical analysis of the data and hypothesis testing.

For the purpose of visual representation, we use both probability distribution diagrams and box plots: the former allow a fine grained appraisal of the distribution, while the latter allow for a more immediate comparison. In the probability distribution diagrams we reported the DD values in ascending order on the horizontal axis and the cumulative frequency on the vertical axis.

The research questions are addressed by means of statistical hypothesis testing; therefore we formulated null and alternative hypotheses as follows.

Concerning RQ1

$H0_0$: There is no significant difference in terms of DD between projects assessed under CMMI and projects not assessed under CMMI.

$H0_a$: There is significant difference in terms of DD among projects assessed under CMMI and projects not assessed under CMMI.

$H1_0$: There is no significant difference in terms of DD among projects developed under different CMMI levels.

$H1_a$: There is a significant difference in terms of DD among projects developed under different CMMI levels.

If the above null hypothesis $H1_0$ can be rejected, we can conduct a post-hoc investigation of the pair-wise differences; in this case, the Bonferroni correction for multiple tests shall be applied. For any pair of CMMI levels (L1, L2) we can formulate the hypotheses as:

$H1.1_0$: $DD_{L1} = DD_{L2}$ (Projects developed in L1 have the same defect density as those developed in L2)

$H1.1_a$: $DD_{L1} \neq DD_{L2}$ (Projects developed in L1 have not the same defect density as those developed in L2)

Concerning RQ2

$H2_0$: There is no significant difference in terms of DD among projects developed with and without structured software processes.

$H2_a$: There is a significant difference in terms of DD among projects developed with and without structured software processes.

$H3_0$: There is no significant difference in terms of DD among projects adopting different structured software processes.

$H3_a$: There is a significant difference in terms of DD among projects adopting different structured software processes.

If the above null hypothesis $H3_0$ can be rejected, we can conduct a post-hoc investigation of the pair wise differences; in this case, the Bonferroni correction for multiple tests shall be applied. For any pair of structured software processes (P1, P2) we can formulate the hypotheses as:

$H3.1_0$: $DD_{P1} = DD_{P2}$ (Projects developed in P1 have the same defect density as those developed in P2)

$H3.1_a$: $DD_{P1} \neq DD_{P2}$ (Projects developed in P1 have not the same defect density as those developed in P2)

According to the recommendations in [7] we use the Kruskal-Wallis test for differences between three or more groups and the Mann-Whitney test for pair wise differences. To evaluate the practical difference comparing different groups, we use the standardized effect size measure like Cohen's d.

In the statistical testing, the significance level is checked by the given p-value. For rejecting or accepting the null hypothesis, we used the significance value α=5% / number of tests (Bonferroni correction).

5.1.2.2 Results

RQ1 Do different CMMI levels affect defect density?

The data set contains 32 projects from companies that are assessed under CMMI and 29 projects from companies that are not assessed under CMMI. Figure 22 contains the box plot of DD vs. projects (CMMI assessed, CMMI not assessed). It shows that projects assessed under CMMI have higher variance of DD. Figure 23 shows the cumulative distribution of the two project groups. Table 39 in its first section, reports the descriptive statistics of DD of projects (CMMI assessed, CMMI not assessed).

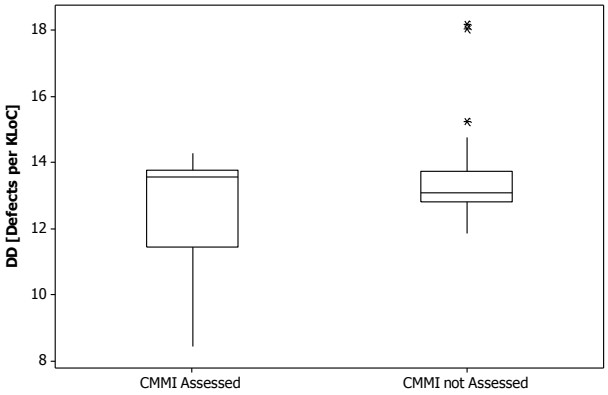

Figure 22 Box plot of DD of projects assessed under CMMI vs. projects not assessed under CMMI.

Table 39 Descriptive statistics of DD of projects

Group		N	Mean	Median	Std Dev
CMMI	Assessed	32	12.5	13.5	1.86
	Not Assessed	29	13.5	13.1	1.44
Maturity Level's	CMMI1	9	13.9	13.9	0.25
	CMMI 3	14	12.1	13.1	2.2
	CMMI 5	9	11.69	11.5	1.42
Software Process	With Structure	57	12.9	13.15	1.75
	Without Structure	4	14.5	14.7	0.78
Structured Software Process	Water Fall	12	13.8	13.8	0.24
	Agile	7	13.43	13.35	2.8
	Rational Unified Process	7	13.01	12.8	0.45
	Microsoft Solution Frame work.	2	12.89	12.89	0.29
	Hybrid Process	5	12.1	11.5	1.06
	Team Software Process	4	11.51	11.9	2.19
	V Model	3	11.22	11.41	2.45

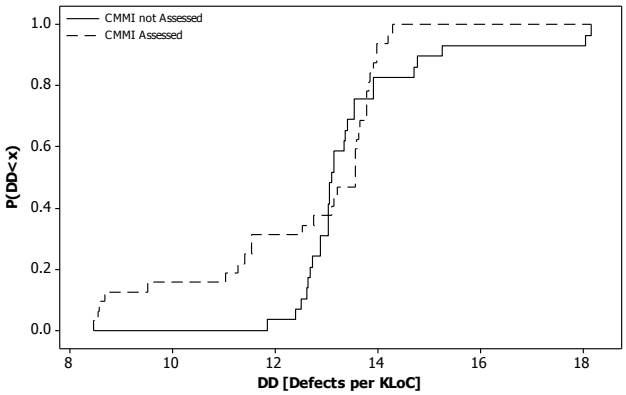

Figure 23 DD cumulative distribution for CMMI assessed and CMMI not assessed projects

For RQ1, observing the box plots in Figure 22 it appears that they overlap, but CMMI assessed projects are more skewed towards lower DD. We then test the hypothesis H00 with Mann-Whitney test for differences. The test reports a p-value = 0.745 which is above the α threshold. Therefore, we cannot reject the null hypothesis, indicating that there is no significant difference of DD between the projects assessed under CMMI and projects that are not assessed under CMMI.

As a next step, we focus our attention on projects with CMMI assessment. The data set contains 9 CMMI 1 projects, 14 CMMI 3 projects, and 9 CMMI 5 projects. Figure 24 contains the box plot of DD vs. CMMI levels. It shows that higher levels of CMMI seem to have a lower DD, but higher variance.

Figure 25 reports the cumulative distribution DD for different CMMI levels. Table 39 in its second section reports the corresponding descriptive statistics. To test H10 we select Kruskal-Wallis's test to see the significant difference of DD in different CMMI levels. The test reports a p-value = 0.0009, which is below the α threshold. Therefore, we can reject the null hypothesis.

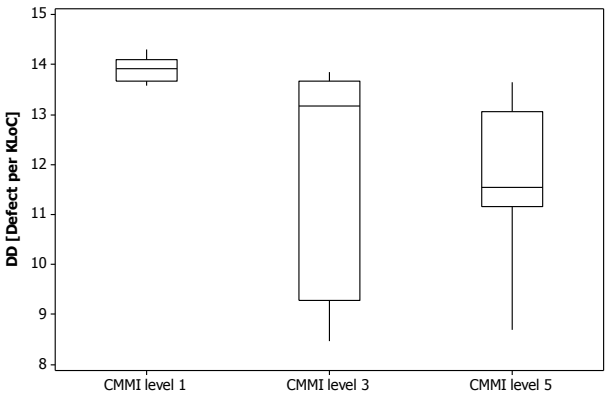

Figure 24 Box plot of DD of different CMMI levels

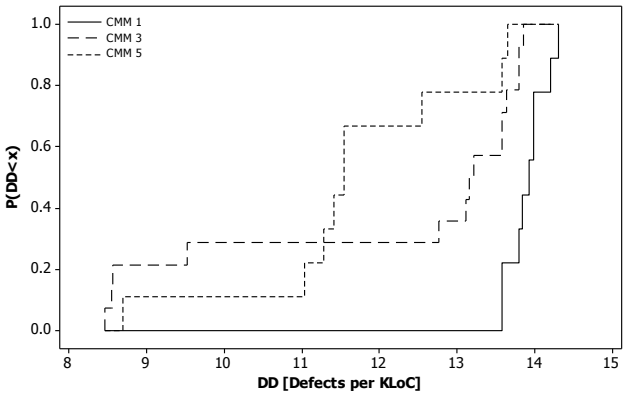

Figure 25 DD cumulative distribution for different CMMI levels

Given the above result we proceed with the pair wise comparisons. In particular, we test the $H1.1_0$ for the two possible adjacent pairs of CMMI levels, i.e. (CMMI 1, CMMI 3), (CMMI 3, CMMI 5) by means of the Mann-Whitney test. In assessing the test results, we adopt an α divided by 2 according to the Bonferroni rule.

For the pair (CMMI 3, CMMI 5) we obtained the p-value 0.3, which is larger than 0.025, therefore we cannot reject the corresponding null hypotheses.

For the pair (CMMI 1, CMMI 3) we obtained a p-value of 0.0021, therefore we can reject the null hypothesis. The significant difference can be considered of medium size (Cohen's d = 0.75), CMMI 3 projects have a DD that is on average 1.8 defects per KLoC lower than CMMI 1.

In summary, concerning CMMI levels, there is evidence that the defect density of CMMI 3 projects is lower than CMMI 1 projects.

RQ2 Do structured software processes affect defect density?

The data set contains 57 projects developed with structured software processes and 4 projects developed without structured software process. The third section of Table 39 reports the descriptive statistics of DD of projects developed with and without structured software process.

Figure 26 reports the box plot of DD vs software process structuredness. It shows that projects developed with structured process seem to have lower DD. Figure 27 presents the cumulative distribution of DD figures of projects developed with and without structured software process.

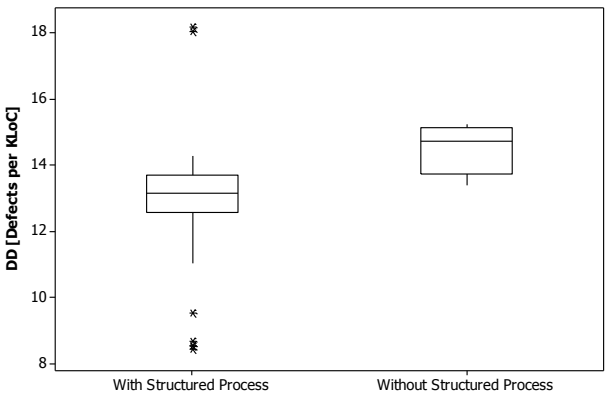

Figure 26 Box plot of DD of projects with and without structured process.

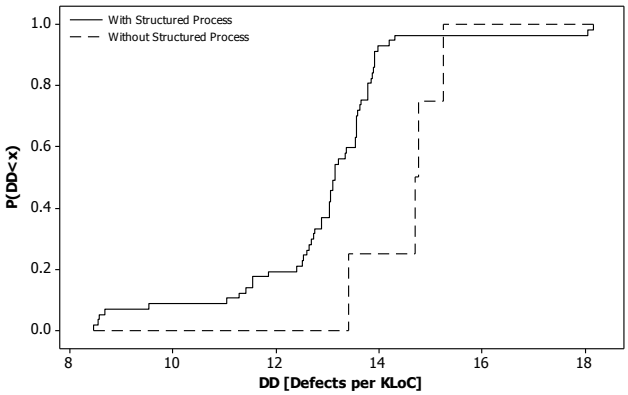

Figure 27 DD cumulative distribution for projects with and without structured process

Concerning hypothesis $H2_0$, we select the non-parametric Mann-Whitney. The test reports a p-value = 0.013, which is below the α threshold. Therefore, we can reject the null hypothesis. The significant difference can be considered of large size (Cohen's d = 1.18), projects developed with structured process have a DD that is on average 1.6 defects per KLoC lower than the projects developed without structured process.

Considering the projects developed with structured software process, the data set contains 12 projects using Waterfall, 7 Agile projects, 7 projects adopting RUP, 2 projects using MSF, 5 Hybrid projects, 4 projects using TSP, and 3 projects using the V model. Figure 28 contains the box plot of DD vs. process type. It shows that V model, TSP and Hybrid process have lower DD, but higher variance.

Figure 29 reports the cumulative distribution DD of different structured software process. The fourth section of Table 39 reports the relative descriptive. To test H30 we selected Kruskal Wallis' test. The test reports a p-value = 0.004, which is below the α threshold. Therefore, we can reject the null hypothesis.

Given the above result, we precede with the pair wise comparisons. In particular, we test the $H3.1_0$ for all possible pairs of structured software processes i.e. (Waterfall, Agile), (Waterfall, RUP), (Waterfall, MSF), (Waterfall, Hybrid), (Waterfall, TSP), (Waterfall, V model), (Agile, RUP), (Agile, MSF), (Agile, Hybrid), (Agile, TSP), (Agile, V model), (RUP, MSF), (RUP, Hybrid), (RUP, TSP), (RUP, V model), (MSF, Hybrid), (MSF, TSP), (MSF, V model), (Hybrid, TSP), (Hybrid, V model) and (TSP, V model) by means of the Mann-Whitney test. In assessing this test, we adopted an α divided by 21 according to the Bonferroni rule.

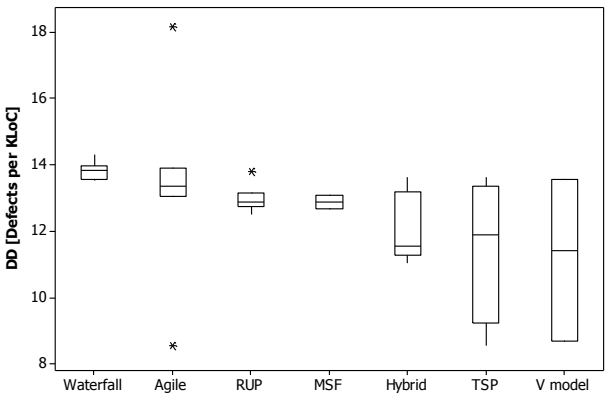

Figure 28 Box plot of DD for different structured processes

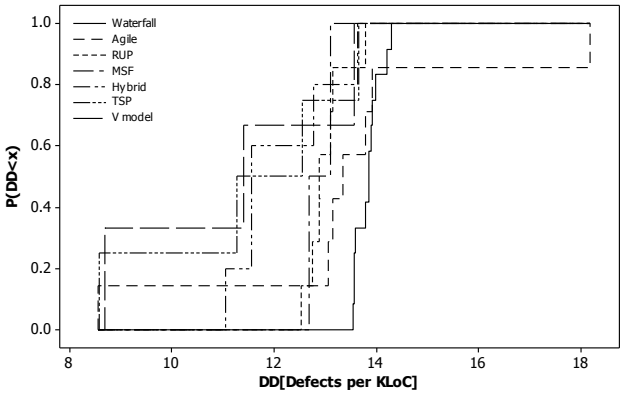

Figure 29 DD cumulative distribution for structured processes

For all pairs except (Waterfall, RUP) we obtained p-values > 0.002 therefore we cannot reject the corresponding null hypotheses. For (Waterfall, RUP) we obtained p–value < 0.002 therefore we can reject the corresponding null hypothesis indicating that they have statistical different DD. The significant difference can be considered of large size (Cohen's d = 3.7), Waterfall projects have a DD that is on average 0.7 defects per KLoC higher than RUP.

5.1.2.3 Discussion

The extraction of DD figures from industrial projects allowed us to publish a summary of DD data available. Table 39 presented the essential descriptive statistics. These results can be of useful to both researchers and practitioners. In addition, we can summarize the following pieces of evidence:

- There exists no statistically significant difference of DD between the projects assessed under CMMI and the projects not assessed under CMMI. However, visual analysis of the box plots Figure 22suggests that CMMI assessed projects tend to be on the low DD side.
- There exists a statistically significant medium sized difference of DD between CMMI 1 and CMMI 3 based projects: the former have a DD that is 1.8 defects per KLoC higher than the latter.
- There exists a statistically significant large sized difference of DD between the projects developed with vs. without structured software process: the former has a DD that is 1.6 defects per KLoC lower than the latter.
- There exists no statistical significant difference of DD between different structured processes except (Waterfall, RUP). Waterfall projects have DD that is on average 0.7 defects per KLoC higher than RUP.

Overall these results confirm that having a process (as suggested by CMMI, RQ1, or any structured process, RQ2) has a positive effect on quality measured in terms of DD. Higher CMMI levels have an effect on quality, but probably smaller than one might expect, especially considering that we couldn't find any difference between levels 3 and 5. Also adopting a specific process (Waterfall, Agile, TSP....) does not produce specific effects on quality. With this respect the most surprising comparison is Waterfall vs Agile, where again there seem to be no difference in product quality.

So the key factor for quality (evaluated in terms of DD) seems to be having or not a process. Of course we acknowledge that our analysis is partial: a process may have an effect on other properties

(cost, time to market, customer satisfaction, etc). Further studies should be dedicated to analyze the effect of processes on those qualities.

Threats to validity

We discuss in this section validity threats using the classification proposed by [7].

As for internal validity, there could be two level of indirection and sources of error. First we rely on a data set collected by others, who in their turn rely on how companies have collected the data. Unfortunately, as any secondary study, we have no control on these aspects.

As for construct validity, there are concerns about the conformance of projects classified according to CMMI levels or as adopting Structured Processes. For structured process we group all procedural software development processes (V model, TSP, Hybrid Process (RUP + TSP + Agile), MSF (Microsoft solution framework), RUP, Agile and Waterfall etc) into structured category. The structured processes sometime are customized for particular project needs and this trend is also observed for some projects. On the contrary DD have low conformance, collection and transformation of size from function points to LoC can subject to ambiguities.

As for external validity we face the problem of generalizing the results. The study samples 61 projects, which are are not a negligible absolute number but may offer a limited representation of industrial projects in general.

5.1.2.4 Conclusion

This study performed a statistical analysis of the product DD along two dimensions of process quality - i.e. level of maturity and type - in 61 industrial projects.

The DD values are useful as a proxy of the quality of the products and are widely used for benchmarking and evaluation purpose.

Our results could not completely confirm some previous studies [62] [75] that reported a steady quality increase with process assessment levels. Our results partially support the observed increase in quality, by moving to the higher levels of maturity e.g. CMMI 1 to CMMI 3. However, corroborates the findings from previous studies [82][84][85] where increase of quality is observed with the adoption of increasingly more structured software processes.

5.1.3 Data Set: NASA 93:

5.1.3.1 Process maturity (PM)

The process maturity is determined by the procedure organized by the Software Engineering Institute Capability Maturity Model (SEI-CMM). The process maturity can be assessed in two ways [54]. The first is based on the results of the evaluation of the CMM. The second is the assessment based on the 18 key process areas (KPAs) in the SEI–CMM. The process maturity determining is decided by the percentage of compliance for each of the KPAs.

Table 40 Independent variables categories

Independent Variables	Categories	N
Process Maturity (PM)	High	43
	Medium	30
	Low	20

5.1.3.2 Results

Impact of Process Maturity (PM) on software quality

The data set contain 43 projects developed under high PM, 30 projects are developed under medium PM and 20 projects are developed under low PM. The summary descriptive statistics are reported in Table 41 divided by type. Figure 30 shows that there is a high variation of software quality in the projects developed under low PM. In addition high variation is also observed for the software quality in the projects developed under medium PM. Considering the Figure 30 we can visually analyze the following trends.

- High PM → Higher than average software quality
- Medium PM → Lower than average software quality
- Low PM → Lower than average software quality

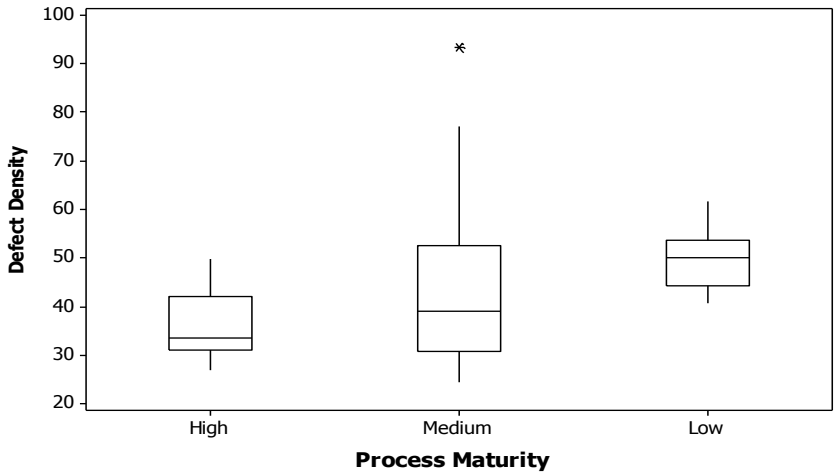

Figure 30 Interval plot of process maturity against quality of software

Table 41 Descriptive statistics for process maturity

Process Maturity	No	Quality	Std Deviation
High	43	36.26	6.96
Medium	30	43.18	17.53
Low	20	49.86	6.80

We performed the pair wise comparison between two neighboring pairs of process maturity to find the significant difference of software quality between them. Concerning software quality we made two pair's (High Vs Medium) and (Medium Vs Low).

We performed the Mann-Whitney test for the pair wise comparison. Concerning software quality we found the p-value above the α threshold for the pair (High, Medium) and below the α threshold for the pair (Medium, Low).

Therefore we can indicate that there is a significant difference of software quality between the pair (Medium, Low).

The partial impact of process maturity on the software quality and development productivity shows that there is an impact but smaller than one might except which also confirms the findings of Caper Jones data set.

5.2 Introduction of exploratory testing

The cost of software testing is believed to range between 40 and 80% of the total cost of development with safety critical systems closer to the high end [97]. According to NIST [98] inadequate software testing infrastructure is estimated to cost US society about 59.5 billion USD annually.

Among the several attempts to reduce such costs, automated software testing deserves a special mention [99]. Automated software testing provides quick verification and reduces the testing execution effort but it requires a significant upfront investment to set up the infrastructure. The alternative, i.e. manual testing, is very labor intensive and requires human testers to execute the tests, e.g. test case based testing and exploratory testing. Despite the potentially higher costs, the latter approach is more common in industry compared to the former [100].

The different types of manual testing approaches are typically selected based on the context; for instance structured and prescriptive techniques such as Test case based testing are adopted when a system undergoes the acceptance testing [101].

In general, any technique typically leads to specific advantages, while it brings some drawbacks which often are not immediately apparent but tend to surface later. A testing approach introduces testing induced Technical Debt (TD) whose value is not known at the time of testing and appears in later phases of the life cycle.

We analyzed the software testing approach "*Exploratory Testing (ET)*" through a systematic review of literature to understand the consequences of ET as Technical debt. The evidence shows that ET is used as an alternative to any structured software testing approach to speed up the testing tasks and proved to be cost effective at the time of testing. Nevertheless ET also has many weaknesses that are not apparent at the time of testing but prompt up in later phases of system life cycle. These weaknesses incur increased rework and cost, and hence are considered to be the sources of TD. In addition we propose the possible solutions to embark upon these weaknesses that indeed help to reduce the testing technical debt of ET.

5.2.1 Exploratory testing

Exploratory testing (ET) is an approach that does not rely on the formal test case definition. In fact, the tester, instead of designing test cases, runs and evaluates the software behavior basing tests on his intuition and knowledge. The formal definition of ET was proposed by James Bach as [102]:

"Exploratory testing is simultaneous learning, test design, and test execution"

Given its widespread adoption in industrial practices to speed up verification activities of newly developed functionalities, several researchers have begun to focus their studies on ET. ET is considered as a cost effective practice due to the lack of test case documentation and planning. Moreover it also has good defect detection ability and is a very flexible process [103].

Practitioners tend to consider ET as a valid alternative to systematic testing approaches when not enough time is available [102]. In such context it is not possible to define plans and write test cases on the basis of requirement and design specification, and later abide. Then, adopting ET, testers utilize their own intuition and experience without the need of any documented guideline, testing only particular scenarios or functionalities. Such an unconstrained and creative approach makes ET attractive for testers. In addition, as a consequence ET provides rapid feedback, simultaneous learning and diversity in testing [102].

The benefits listed above makes ET very attractive. The one example of ET adoption is the testing of whole web site testing carried out in just two days [104]. Such a practice makes very serious concerns about the overall effectiveness of the testing approach. This may probably introduce increased work, problems and cost in the future, when the complexity of the application will need to be managed in a more structured way also in testing.

On the other hand ET cannot be applied to some applications, e.g. safety critical systems, where the testing procedure must strictly adhere to well defined procedure to document the testing done, in order to conform to given requirements, e.g. safety integrity level.

5.2.2 Technical debt

The term Technical Debt refers to an increased cost of changing or maintaining a system in the future due to expedient shortcuts taken during development [105]. In other words, Technical Debt is a techno-financial instrument that makes quick development affordable at present time, at the cost of compound interests to be paid later [106]. Technical debt can be related to different activities: architecture, design, documentation, testing, and so on. We focus on Testing Technical Debt. We define one source of testing technical debt in the following way:

"Feature testing to attain one aspect while simultaneously ignoring -- either knowingly or not -- other aspects that may prompt up later with increased rework and cost"

There are different ways to deal with technical debt, Buschman [106] proposes three main categories of approaches:

- Pay the interests: we suffer the consequences of the debt and to cope with it we incur in repeated additional costs, e.g. in absence of automated tests we keep on carrying on expensive manual regression tests.
- Repay the debt: we get rid of the origin of the debt at the cost of significant extra rework effort, e.g. a structural limitation in the program is removed through an additional reengineering activity.
- Convert the debt: we replace the source of technical debt with another solution still implying some debt, though typically smaller, e.g. a flexible though hard-to-maintain run-time customization module is replaced with a rigid development-time one.

While the above definition may seem negative, contracting debt is not necessarily a bad thing. As in real-life situations a number of readers would not own a house without a mortgage, so most projects could not be successful.

5.2.3 Systematic literature review

In order to collect evidence about the effects of ET on technical debt we conducted a systematic literature review. The goal we set for the SLR was to assess the potential negative effects of applying the ET technique.

We followed a simple procedure: (i) first we searched in the most important publication databases --- IEEE explorer, ACM digital library, Springer link and Science Direct -- with selection limited to software/computer science studies. The search was done using the keyword "Exploratory Testing" and the option "search in all fields" was used. At this stage we obtained 95 articles. Then we screened out the irrelevant paper first looking at the title and then reading the abstracts; at this stage we had 32 articles left. Eventually we read through the remaining papers and decided whether to include them on the basis of the actual content; as a result we included 8 papers. The number of articles retained at each stage and grouped by source database is reported in Table 42. The selection criteria we used in the above filtering were: (i) the paper deals with Exploratory Testing, (ii) it contains empirical evidence base on a valid study, (iii) the evidence concerns potential drawbacks induced by ET.

Table 42 Distribution of selected papers.

Source	Found	Scanned	Included
IEEE explorer	7	7	5
ACM digital	47	8	2

library			
Springer Link	39	15	1
Science direct	2	2	0

While examining the articles we had a twofold objective: first, identify which are the areas of potential negative impact of ET and, second, verify the existence of empirical evidence supporting such a link. Table 43 summarizes the practices impacted by ET and the relative supporting evidence as revealed in the surveyed articles.

Table 43 Supporting evidence Vs practices impacted by ET

Empirical Evidence	**Weaknesses**				
	Test Planning	**Test case Definition**	**Test result Assessment**	**Human Dependence**	**Documentation**
Controlled Experiments	E1	E1	E1		
				E2	
		E3			
				E4	
		E5			
Interviews	I1	I1		I1	I1
Case Study			CS1	CS1	CS1
Action Research	AR1				

Surveyed publications

E1 J. Itkonen, M. V. Mantyla, and C. Lassenius, "Defect Detection Efficiency: Test Case Based vs. Exploratory Testing," in *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, 2007, pp. 61–70.

E2 L. Shoaib, A. Nadeem, and A. Akbar, "An empirical evaluation of the influence of human personality on exploratory software testing," in Multitopic Conference, 2009. INMIC 2009. IEEE 13th International, 2009, pp. 1–6.

E3 T. D. Hellmann and F. Maurer, "Rule-Based Exploratory Testing of Graphical User Interfaces," in AGILE Conference (AGILE), 2011, 2011, pp. 107–116.

E4 S. Al-Azzani and R. Bahsoon, "Using implied scenarios in security testing," in Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems, Cape Town, South Africa, 2010, pp. 15–21.

E5 L. H. O. do Nascimento and P. D. L. Machado, "An experimental evaluation of approaches to feature testing in the mobile phone applications domain," in Workshop on Domain specific approaches to software test automation: in conjunction with the 6th ESEC/FSE joint meeting, Dubrovnik, Croatia, 2007, pp. 27–33.

I1 J. Itkonen and K. Rautiainen, "Exploratory testing: a multiple case study," in Empirical Software Engineering, 2005. 2005 International Symposium on, 2005, p. 10 pp.

CS1 J. Itkonen, M. V. Mantyla, and C. Lassenius, "How do testers do it? An exploratory study on manual testing practices," in Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on, 2009, pp. 494–497.

AR8 J. Tuomikoski and I. Tervonen, Absorbing Software Testing into the Scrum Method, 2009. Lecture Notes in Business Information Processing, 2009, Volume 32, Part 4, 199-215.

5.2.4 ET as a source of testing technical debt

Does ET represent an archetypal example of technical debt inducing practice? Shall it be repaid later in the application life cycle?

To answer these questions we conducted a systematic literature review concerning ET (see side box) and collected evidence about its weaknesses. We examined eight articles reporting empirical evidence: five controlled experiments (sources: E1 to E5), one case study (source: CS1), one interview (source: I1), and one action research study (source: AR1). We were particularly interested in those aspects that in the short term represent a cost saving but might imply a debt whose accrued interests ought to be repaid later. In order to understand how these weaknesses may induce technical debt we need to:

- identify the different aspects of testing that are affected by ET in a potentially negative way,
- understand the effects of ET on those aspects, and
- evaluate them in terms of technical debt.

We categorized the different type of debt in which testers might occur when practicing ET and, after we identified its general weaknesses, we mapped them to the technical debt instances. Hereinafter, we list the weaknesses of ET identified as a source of technical debt. We will discuss later on how to tackle the debt.

5.2.4.1 Test Planning

Typically test planning defines all those aspects related to testing strategy, resource utilization, responsibilities, risks, testing priorities, budget, and timeline [107]. Moreover planning is the prerequisite for monitoring and tracking the test activities, enabling awareness and visibility of the process.

ET effects: Different type of evidence is available in the literature (sources: E1, I1, AR8) on the lack of any planning of ET activities, given the vocation to personal freedom and creativity of this practice. ET lack of test planning results into having no control over the testing and increases the likelihood of double testing or overlooking important tests. Moreover the lack of plans makes the coverage of system functionalities unknown, therefore it is not possible to accurately estimate the overall quality of the system and make accurate plans for effort maintenance. In addition, due to the lack of planned responsibilities the tester's progress is difficult to track and responsibilities remain undefined. For example if the defects appear in a tested application during operations no one has the responsibility for that test.

TD implications: According to established approaches testing without a test plan cannot be managed in an effective way: for example, overruns of one or two hundred percent have been reported and test managers have difficulty understanding and monitoring testing as well [107]. Therefore, the higher cost of tests execution is the first type of debt introduced by ET. Yet another consequence of the lack of planning is the higher number of residual defects due to unmanaged functionality coverage and to the inappropriate handling of defects.

5.2.4.2 Test cases definition

A test case defines how the implementation of a given specification can be validated and typically includes preconditions, test steps, input data, and expected output. The documentation of test case provides a structure, guidance and traceability to the testing tasks [107].

ET effects: On ET, testing is based on simultaneous design and execution utilizing human intuitions; therefore it is performed without neither defining nor documenting the test cases. If we consider that we often need to re-execute the tests [108] (e.g., to verify that a modification did not introduce new errors), the absence of documented test cases will make the re-execution quite hard or even impossible where tests were traced/recorded. Moreover, we found evidence that the lack of test cases implies that ET cannot assure 100% testing of all functionalities. There are evidences where functionalities remained without going under the course of testing. (sources: E1, I1).

TD implications: In ET, re-execution of tests (regression testing) is quite hard and expensive due to the absence of test cases. In addition to that, considering that some functionalities are not tested, residual defects could prompt in later phases like in production and maintenance, not only causing the malfunctioning of the system but also higher cost of defect removal.

5.2.4.3 Test result assessment

The outcome of a test is assessed in order to verify the correctness of functionality. A test oracle is required to determine whether the results of a program execution in a test are correct or not; an oracle can be just the expected result or a criterion to take a formal decision.

ET effects: In ET a formal oracle is absent and it is replaced by the tester's own judgment, therefore the evidence suggests that it is not possible to judge formally the correctness of an output (sources: E1, CS1). The main effect is the high probability of judging the incorrect behavior of an application as a correct one or vice versa. The evidence indicates [109] that these sorts of oracle mistakes are responsible for possible residual defects.

TD implications: The test result assessment based on the missing oracle in ET would result in additional rework due to more residual defects directly affecting the maintenance costs.

5.2.4.4 Human dependence

Testing as many software development activities is a human intensive activity. Naturally the outcome of any task depends on individual feature, e.g. skill, but too much variability may represent a problem. In particular we found evidence that ET is highly human dependent specifically on two factors. A first factor is experience: different studies (sources: I1, CS1, E4) reported that ET is highly dependent on the experience of the tester. A second factor is personality extraversion: testers that possess an extrovert personality achieve the highest ET effectiveness (source: E2). Though, this is a trait that is not apparent and noticeable at the time of testing and selection of exploratory tester.

ET effects: Experience has positive effects on various parameters such as domain knowledge, speed of completing tasks, ability to identify meaningful patterns, superior recall [110]. Further, experienced testers identify more potential categories of defects [111]. This makes ET more a form of test for experienced staff. Therefore ET could be suboptimal if it is performed by some junior testers and possibly re-work or re-tests might be necessary.

TD implications: The major consequences of being ET highly human dependent might be the accumulation of residual defects due to a sub-optimal tester fitness and in general a non uniform test accuracy over the whole system. Thus the probability of rework to fix defects not found in testing is high and dependent on the difficulty of the fix task.

5.2.4.5 Documentation

The documentation covers all the required information related to system development; it includes typically preparation -- e.g. planning, requirements, and design -- and final documents -- logs and reports --. Not only insufficient but also unmaintained documentation may introduce weaknesses that are identified as a source of error [112].

ET effects: In ET, there is typically not much available documentation. No test guidance is available and other testing artifacts are not produced; usually only failed tests are reported. ET limited documentation can provoke problems in the testing phase particularly because in following up a failure report is difficult to understand what has been tested and what has not (sources: I1, CS1) and in the maintenance phase too, where having no comprehensive documentation might slow down the activities. Since ET test cases are born out of intuition and experience in tester's mind and there remain confined without any documented trace, other testers will not be able to reproduce them due to the lack of documentation and identifying the root cause of problems may result extremely difficult. Such problems are even more pressing when a tester leaves the test team and new resources have to repeat tests or to perform regression testing: the lack of documentation makes knowledge transfer within the company very difficult or even impossible. Moreover, insufficient or missing documentation might lead to difficulty in estimating the effort required in maintenance phase (or in allocating the proper time for testing) because no or inconsistent logs of past activities are available.

TD implications: The main consequence of lack of documentation in ET is increased and repeated costs in the maintenance phase. In addition, lack of documentation jeopardizes knowledge management in the company with additional costs for new testers. On top of those costs there might also be estimation errors in effort planning due to the lack of proper logs of the past testing activities.

5.2.5 Tackling the exploratory testing debt

Figure 31 summarizes how exploratory testing influences the testing activities and what is the result in terms of technical debt on the basis of the SLR conducted.

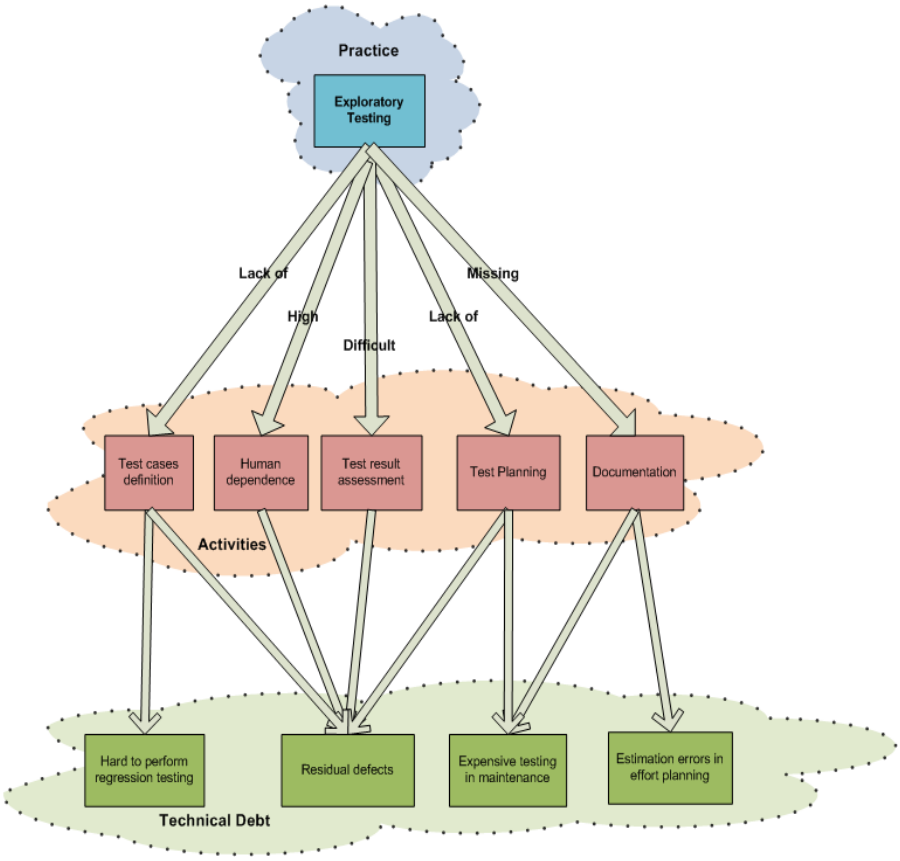

Figure 31 Induced TD by the implication of ET

In presence of technical debt, either induced by ET or other practices, the management ought to cope with it: that is paying the interests, retiring the debt, or converting it [2].

The first possible choice is to simply accept the very nature of the technical debt: trade smaller cost at present for larger costs in the future; that means paying the debt and the interests. The installments may take different forms that, based on the evidence discussed above, appear to be mainly: a larger amount of rework needed for defect removal, the allocation of unanticipated and thus more expensive tasks, additional difficulties in performing regression testing. Another immediate consequence of ET is represented by a higher number of residual defects, which also causes bad advertisement with the customers; this may represent a significant cost.

The possible alternative is represented by debt conversion that is transforming some part of the approach that brings immediate advantages but induces debt into a different one that implies more immediate costs but less debt. To achieve debt conversion, a basic transformation consists in providing ET with a semi-formalized structure that overcomes or partially reduces the weaknesses. We can see alternative for structuring the ET:

> *"Use ET in conjunction with other testing approaches where ET is unified by other approaches in formal software testing process"*

We will explain the solutions with a scenario: unifying ET with Requirement Based Testing [113] as a solution.

Requirements Based Testing generates test cases from a set of requirements [113] that may be used to structure ET in such a way that it would be documented and executed to test the basic requirements first. Afterwards ET would be performed in the usual fashion utilizing the human intuition and giving freedom to the testers to design their test based on the experience and on the fly, to identify more focused defects. This combination might lower the risk of having residual defects on the most used part of the systems (i.e. the basic requirements) but at the same time leaves the testers free to explore unusual scenarios based on their creativity. Moreover, the presence of a plan to test the basic requirements might drive down the debt associated to the lack of planning: it would help controlling the status of the testing process, the chance of double testing and overlooking of important test would be reduced and performing the regression testing should be much easier. It also helps to assess the test results based on requirements and the functionality coverage of testing will be computed as percentage of requirements tested.

5.2.5.1 Considerations

- The key issue is the understanding of exploratory testing as an approach for testing. We emphasized only on the free exploratory testing which is most widely used as industrial practice [103]. So we enforce that using free ET could lead to such implications of TD. However using other styles of ET or ET conducted through session based management system might not have such implications but subject to the further direction of research.
- We emphasize our findings based on the academic literature supported by empirical evidences published in electronic databases. There is also much literature available online in forms of test blogs, wiki's and consultant websites. We didn't consider such gray literatures which are not supported through empirical evidences.
- We emphasize that the outcome of this study is just the hypothetical understanding based on the evidences from the literature. However we state that these hypothesis are ought to be verified further with empirical evidences.

5.2.6 Summary

Exploratory testing is widely used in the industry; practitioners view ET as a cost effective substitute for their daily testing activities where they are not bound to follow the structured and systematic way of testing. However, empirical evidence reported in the literature provides a more comprehensive picture of ET that takes in consideration also the technical debt implication of this technique. While ET has some manifest and immediate positive aspects and benefits, evidence suggests that it also brings some deferred drawbacks that might consist in a technical debt burden.

Such a picture allows practitioners to take an informed decision about ET adoption: therefore when planning to adopt ET, in addition to the specific benefits, testers and managers should be aware of the future TD that are packaged with the benefits of ET. Since, most of us need a mortgage to own

a house, so most projects could not be successful without any compromises i.e. contracting some technical debt.

Any solution is a point in a continuum, at one end of the spectrum there is pure Exploratory Testing, cheaper in the short term, with no upfront costs, but bearing a significant debt, while at the other end is a structured and possibly automated testing approach, more expensive, with important upfront costs, and with a limited debt. The difficult work of the project manager is to find the right compromise that best suits the context of the project.

5.3 ET in conjunction with other testing approaches

5.3.1 Introduction

Software testing aims to verify whether software behaves as intended and identifies potential problems. A recent survey [114] indicates that testing is the main approach being used in industry to identify defects. Hence, there is a need to understand how to improve the efficiency and effectiveness of testing approaches. Two widely used testing processes in industry are scripted testing (ST) (also referred to as prescriptive or test case based testing in International Organization for Standardization/International Electro technical Commission [ISO/IEC] 29119 Software Testing Standard) and exploratory testing (ET) [115].

Scripted testing follows a prescriptive process, in which test cases are designed prior to test execution to structure and to guide the testing tasks. Many of the existing studies on ST have a focus on automated test case design, generation and prioritization, or testing technique selection [116][117][118] [119]. In a sense, ST is a plan-driven process for testing.

On the other hand, in ET, the tests are not defined in advance in an established test plan but are dynamically designed, executed, and modified [9]. Exploratory testing is also referred to as ad hoc testing [120] as it relies on the implicit and informal understanding of the testers. Because the literal meaning of ad hoc may correspond to sloppy and careless work, the term 'exploratory' was introduced by a group of experts instead of 'ad hoc' [102]. As the testers can freely explore an application by utilizing human intuition and experience [102] [121], and it is not explicit how they make this exploration, the tasks are performed manually rather than with an automation support.

Scripted testing and ET provide various benefits and weaknesses. A few studies [117] [115][103] mentioned that ET makes better use of testers' creativity and skills to discover the bugs that prescriptive testing may not uncover because of its mechanical nature. Agruss and Johnson [120] and Bach [122] claimed that software testing might benefit through using these approaches in combination. In general, there is a general interest in industry for a hybrid testing (HT) approach unifying the two approaches, which is, for example, visible in lively discussions in industry oriented blogs (see e.g., [123]).

In this study, our aim is to address the need for a systematic and repeatable investigation of such a hybrid process. To this end, we first explored the weaknesses and strengths of ST and ET by reviewing the literature and getting feedback from industry. Then, based on the signified findings by comparing the two approaches, we propose an HT process that unifies ET and ST in a way that some major weaknesses of ET and ST are minimized in a compromise form.

With these objectives, we formulated the research questions (RQs) for this study as follows:

- RQ1: What are the strengths of ST and ET?
- RQ2: What are the weaknesses of ST and ET?
- RQ3: What are the improvement opportunities for testing process by addressing some major weaknesses of ST and ET through unifying their processes in a hybrid form?

It is important to point out that this paper does not focus on individual testing techniques that can be used within the testing process. For example, common testing techniques in ST for black-box testing include, boundary value analysis [124], equivalence partitioning [124], and decision tables [125]. For ST, the commonly used white-box testing techniques include decision coverage [126], path coverage [126], multiple condition/decision coverage (MC/DC) [127], and data flow coverage [128]. One example of a technique in ET is smoke testing [103]. However, instead, our focus here is on the overall 'testing process' that fulfills the characteristics of ST and ET mentioned previously.

In order to answer our RQs, we used systematic literature review (SLR) ([129]) and interviews as the main research methods. Our research process is shown in Figure 1 and was inspired by the

technology transfer model proposed by Gorschek et al. [130].

Our work starts off with the clear contrast between ET and ST. Consequently, companies could make conscious decisions on which process to choose based on evidence. This implies understanding the strengths and weaknesses of the approaches that are reported in the literature.

Hence, the first phase of this exploratory research was investigating the strengths and weaknesses of ST and ET. Furthermore, we interviewed practitioners with extensive experience of ET and ST in order to identify their perspective on strengths and weaknesses and then compared the outcomes of the interviews to those of the literature review. Through interviews, we also could identify the connections between the strengths and weaknesses of ST and ET that later on helped in identifying the improvement opportunities for an HT process.

After having identified strengths and weaknesses, we mapped the strengths of one process to the weaknesses of the other and vice versa. Practitioners with extensive experience in both HT and ST were involved in this mapping. They also reviewed the final mapping to improve the reliability of the results. The outcome of P1 and P2 provided two major results that are helpful in working towards an HT: (i) clearly establishing the need for an HT; and (ii) knowing how the strengths and weaknesses of ET and ST relate to each others' help in (i) connecting them to the activities of the HT process to check whether weaknesses are addressed and strengths are supported; and in (ii) providing input to questions to be asked when evaluating an HT. The details of this step are given in Section 3.

With the input of the previous phase, we designed the HT process in the third phase. We identified the process fragments and high-level structure of the process as suggested in [131]. The initial design was created by mapping the activities of ET and ST to the strengths and weaknesses identified. Having designed an initial version of the solution (HT process), we iteratively improved the design of the process with the practitioners' input. Code signing the HT process with very experienced practitioners in both HT and ET improves the credibility of the solution proposed.

As the outcome towards a practically applicable and useful HT process, we provide valuable directions based on making strengths and weaknesses between the two processes as well as how they relate to each other explicit. Furthermore, the HT process proposed was designed with practitioner input. In future work, the process should be further evolved in controlled experiments, case studies, and action research.

5.3.2 PHASE 1: Indentifying strengths and weaknesses of exploratory testing and scripted testing

In order to answer our RQs (RQ1: What are the strengths of ST and ET? and RQ2: What are the weaknesses of ST and ET?), we first performed an SLR (see [129] for guidelines of how to conduct systematic reviews). Then, we made semi-structured interviews with practitioners to investigate further the strengths and weaknesses of ST and ET in practice. This provides the input for comparing the two processes.

5.3.2.1 Systematic literature review

Systematic literature review has several advantages over regular reviews where the research design of the literature is often not presented in sufficient detail. In particular, systematic reviews have the following advantages: (i) reduction of bias due to well-defined criteria for selecting studies; (ii) availability of guidelines of how to aggregate evidence from primary studies; (iii) rigor and documentation of design decisions make the review repeatable and extendable; and (iv) the documentation of every step of the review allows for replication (cf. [129] [132]).

In the succeeding texts, we present the details of the search, data extraction, and data synthesis processes of this SLR.

Search process

The basic steps we followed during the search process were as follows:

- Develop the review protocol.
- Perform the search.
- Review search results using the selection and quality assessment criteria.
- Select the primary studies and finalize the review.

In the succeeding texts, we first present the search strings, the selection criteria and procedure, the quality assessment checklist, and the data sources used for the search process. Then, we provide the results of the search and the selected primary studies. Finally, we discuss the data extraction and data synthesis processes, which led to the conclusions of the SLR.

Search strings: We formulated the keywords and the search strings according to our RQs. We used the synonyms and alternative terms for the keywords referring to linguistic dictionaries while limiting them within the context of software engineering. When deciding on the keywords, we also checked the general terminology used in the testing field (e.g., ISO/IEC 29119 and some key publications such as [115]) not to miss any important keyword. Furthermore, we asked an expert in the area to recap the design of the literature review as well as the list of included papers after the review to make sure that no important study is missed. We did not include keywords for specific testing techniques, as here, our focus was on the studies about test processes of ST and ET. To form the search strings, Boolean operators 'AND' and 'OR' were used to intersect or incorporate the search results for different keywords (Table 44). In [129], it is proposed that pilot searches should be carried out in order to identify primary studies by using the defined search strings as defined in review protocol. The search strings were verified by conducting trail searches, and a preliminary search is carried out in order to identify the relevant literature by the help of the Blekinge Institute of Technology (Sweden) (BTH) librarians. We chose the start year of the search from 2000 when ET was introduced (hence, we assumed that significant work should have been published afterwards) and the end date as January 2010.

Table 44 Keywords: (A1 or A2 or A3 OR A4 or A5 or A6) and (B1 or B2 or B3 or B4 or B5 or B6 or B7 or B8].

ID	Keyword
A1	Exploratory testing
A2	ET
A3	Ad hoc testing
A4	Test case based testing
A5	TCBT
A6	Scripted testing
B1	Weakness
B2	Complexity
B3	Shortcoming
B4	Problem
B5	Issue
B6	Strength
B7	Efficiency
B8	Benefit

Data sources

Search for the primary studies was carried out by using the following electronic resources: IEEE Xplorer, Association for Computing Machinery Digital Library, Engineering Village, Google Scholar, and Institute for Scientific Information (ISI), Scopus, and Springer Link. 'Zotero' reference management tool [133] was used to manage and keep the track of the primary studies.

Selection procedure and criteria

The selection of the primary studies included two consecutive steps. The inclusion and exclusion criteria were applied to titles and abstracts. After having identified the potentially relevant studies, the full text of the studies was read. In this step, further studies were excluded as it was not clear from the title and the abstract that they were irrelevant.

Our inclusion criteria when selecting the primary studies were the following:

- Studies provide full text and available for access.
- Studies peer-reviewed by other researchers (journal/conference/workshop papers and thesis).
- Studies published as a book or a book chapter.
- Technical reports (including work in progress) and research theses, for example, PhD (gray literature).

- Studies using the research methods: literature review, experiment, case study, field observation, survey, interviews, experience reports, and expert opinion.
- Studies that provide discussion on the strengths and/or weaknesses for ST and ET processes.

Our criteria to exclude the studies were the following:

- Studies not published in English language.
- Studies that were the duplicates of already included studies.
- Reports on blogs and private Web pages.
- Studies without any evaluation, comparative analysis, or relation to practical experience.

For the articles meeting our inclusion/exclusion criteria, we further applied the following quality assessment criteria:

- Research methodology: Is the research methodology mentioned and described (including research goal, data collection, analysis, etc.)?
- Results: Does the study report on the strengths and weaknesses of ST and ET processes based on a sound research process?
- Validity: Does the study discuss validity threats/limitations to the study?

Search Conduct

We performed the search using the data sources and the search strings. We review the search results and by manually going through the titles and abstracts applying the inclusion and exclusion criteria, which at the end left us with 100 studies for further review. After reading the full-text of the articles, 19 studies remained. The list of studies was cross-checked among the two reviewers, and the final list was agreed upon after discussion. We also consulted an external expert for reviewing the list of identified list of primary studies. He mentioned three more studies of relevance. We reviewed these studies and decided to include them in the primary studies list, which led to a final list of 21 studies to be input to the data extraction and analysis step. The selected primary studies are given in Table 45. The primary studies included 10 conference papers, 3 journal papers, 4 books, 2 technical reports, 1 licentiate thesis, and 1 book chapter. Fifteen of the studies were published after 2004. In year 2009, five studies were published that shows an increasing trend in discussing either the strengths or weaknesses of ST and ET.

Data extraction

Two authors (Syed Muhammad Ali Shah and Usman Sattar Alvi) were the review team implementing the systematic review process. They designed the data extraction form (Table 46) to obtain the required information from the primary studies in order to be able to answer RQ1 and RQ2. One of the other authors, who was not in the review team, reviewed the designed form to check relevancy of the data to be extracted and any missing information that needs to be captured. Then, the forms were slightly revised afterwards to include categories of relevant area of study that helped in uniformity of coding.

Table 45 Included papers

No.	Published Scripted testing/ Exploratory testing venue	Title	Method	Scripted testing		Exploratory testing	
				S	W	S	W
S1	Conference	Itkonen J, Mantyla M, Lassenius C, (2007) Defect Detection Efficiency: Test Case Based vs. Exploratory Testing. First Intern. Symposium on Empirical Software Engineering and Measurement. 20–21 September, Madrid. pp. 61–70.	Controlled experiment	√		√	
S2	Conference	Itkonen J, Mantyla M, Lassenius C, (2009) How do testers do it? An exploratory study on manual testing practices. 3rd Intern. Symposium on	Field observation			√	√

		Empirical Software Engineering and Measurement. ESEM 2009. pp. 494–497					
S3	Technical Report	Agruss C, Johnson B, (2000) Ad Hoc Software Testing, A perspective on exploration and improvisation, Florida Institute of Technology, USA, pp. 68–69.	Expert opinion			√	√
S4	Conference	Itkonen J, Rautiainen K, (2005) Exploratory testing: a multiple case study. Intern. Symposium on Empirical Software Engineering. 17–18 November, pp. 10.	Case study			√	√
S5	Journal	Ahonen J J., Junttila T, and Sakkinen M, (2004) Impacts of the Organizational Model on esting: Three Industrial Cases. Empirical Software Engineering. Springer, Netherlands, vol. 9, pp 275–296.	Case study	√	√		
S6	Conference	Andersson C, Runeson P, (2002) Verification and Validation in Industry: A Qualitative Survey on the State of Practice. Proc. of the Intern. Symposium on Empirical Software Engineering, IEEE Computer Society. 3–4 October, Washington, DC, pp. 37.	Survey		√		
S7	Thesis	Itkonen J, (2008) Do test cases really matter? An experiment comparing test case based and exploratory testing. Licentiate Thesis. Helsinki University of Technology, Finland.	Controlled experiment	√	√	√	√
S8	Book	Kaner, (1988) Testing Computer Software. TAB Professional & Reference Books.	Experience report			√	
S9	Book Chapter	Bach J, (2004) Exploratory Testing. In: Smith J (ed) The Testing Practitioner, E. van Veenendaal, edn. UTN Publishers, Den Bosch, pp 253–265.	Experience report			√	
S10	Book	Kaner C, Bach J, Pettichord B, (2002) Lessons Learned in Software Testing, John Wiley & Sons, Inc, New York.	Controlled experiment			√	
S11	Conference	Shoaib L, Nadeem A, Akbar A, (2009) An empirical evaluation of the influence of human personality on exploratory software testing. IEEE 13th Intern. Conf. on Multitopic. 15 January, Islamabad, Pakistan. pp. 1–6.	Controlled experiment			√	
S12	Technical report	Bourque and Dupuis, (2004) Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE Computer Society, Los Alamitos, California.	Experience report		√		
S13	Book	Tinkham A, Kaner C, (2003) Learning Styles and Exploratory Testing. Portland. Oregon. USA.	Expert opinion			√	
S14	Book	Ryber T, (2007) Essential Software Test Design, Fearless Consulting.	Expert opinion	√			
S15	Conference	Fraser G, Gargantini A, (2009) Experiments on the test case length in specification based test case generation. ICSE Workshop on Automation of Software Test, 18–19 May, Vancouver, Canada, pp 18–26.	Controlled experiment	√			
S16	Conference	Grechanik M, Qing Xie, Chen Fu, (2009a) Maintaining and evolving GUI-directed test scripts. IEEE 31st	Case study	√	√		

		Intern. Conf. on Software Engineering. 16–24 May, Vancouver, Canada, pp. 408–418.				
S17	Conference	Grechanik M, Qing Xie, Chen Fu, (2009b) Experimental assessment of manual versus tool-based maintenance of GUI-directed test scripts. IEEE Intern. Conf. on Software Maintenance. 20–26 September, Edmonton, Canada, pp. 9–18	Controlled experiment		√	
S18	Conference	Ng S, Murnane R T K, Grant D, Chen T, (2004) A preliminary survey on software testing practices in Australia. Australian Software Engineering Conference. 27 September Hawthorn, Australia, pp 116–125.	Survey	√	√	
S19	Journal	Yamaura, (1998) How to design practical test cases, Software, IEEE, vol.15, 1998, pp. 30–36.	Case Study	√	√	
S20	Conference	Taipale O, Smolander K, Kalviainen H, (2006) Factors affecting software testing time schedule. Proc. of the Australian Software Engineering Conference. 18–21 April, Australia, pp.9.	Survey		√	
S21	Conference	Do H, Rothermel G, (2006) An empirical study of regression testing techniques incorporating context and lifetime factors and improved cost-benefit models. In: Proc. of the 14th ACM SIGSOFT Intern. Symp. On Foundations of Software Engineering. 5–11 November, New York, pp. 141–151.	Controlled experiment		√	
S22	Journal	Houdek F, Schwinn T, Ernst D, (2002) Defect detection for executable specifications – an experiment. International Journal of Software Engineering and Knowledge Engineering, vol. 12,(6): pp.637-655	Controlled experiment			√

ST, scripted testing; ET, exploratory testing; S, Strength; W, Weakness.

Table 46 Data extraction form

General information
Title of the article
Name of the author(s)
Date of publication
Venue of publication
Data source used to retrieve the research article
Specific information
Study environment: industry/academia/consultancy
Empirical methods used: experiment, case study, survey, field observation, interview, and literature review
Type of study participants: researchers, industry professionals, students
Relevant area of research study with details: ET, ST, weaknesses of ET, strengths of ET, strengths of ST, weaknesses of ST, and comparison of ST and ET

ST, scripted testing; ET, exploratory testing.

Data analysis and results

For data analysis and synthesis, we used Noblit and Hare's meta-ethnography method [134], which includes a set of techniques for synthesizing qualitative studies. In particular, we used the lines-of-argument synthesis strategy that involves building a general interpretation grounded in the findings of the primary studies [135]. It is essentially interpretive and seeks to reveal similarities and

discrepancies among accounts of a particular phenomenon [136].

In lines-of-argument synthesis strategy, we first identified the 'first order constructs' and the 'second order constructs', and then we came up with the third order interpretations [137][135]. The first order constructs refer to free codes from primary studies (i.e., each individual strength and weakness as stated in primary studies). From these free codes, we identified the 'second order constructs' that refer to descriptive themes in software engineering (e.g., less bogus defects and defect detection effectiveness). We then further interpreted these to develop third order (or synthetic) constructs. Thereby, four main categories were identified for the strengths and weaknesses: (i) testing quality; (ii) nature of the process (structuredness /flexibility); (iii) cost-effectiveness; and (iv) customer satisfaction. The two reviewers worked together during the analysis phase and made decisions for each construct after joint discussion. An example of how first, second, and third order constructs relate is shown in Figure 32.

The third order constructs and their links to second order constructs arising directly from the literature are presented in the following tables (Table 47- Table 50).

We further made a quantitative analysis to provide some quantitative information regarding the percentage of studies with respect to specific types of strengths and weaknesses in addition to types of empirical methods used in those studies.

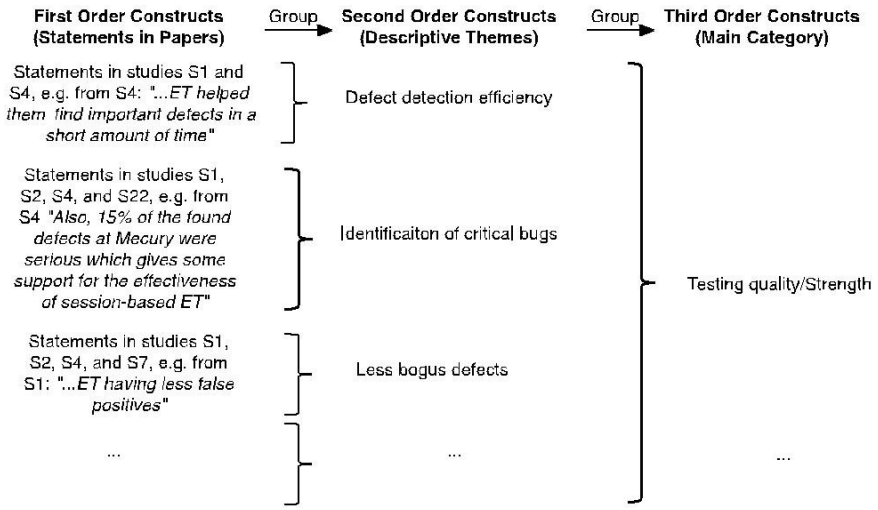

Figure 32 Lines-of-argument synthesis strategy analysis example.

The strengths of ET with respect to the main categories are shown in Table 47. In total, we identified 11 references that discuss the strengths of ET (Table 45).

Analyzing the studies found for the strengths, we identified that 82% of the references (cf. S1 , S2, S4, S7, S8, S9, S10, S13, and S22) highlight the strengths of ET related to testing quality (defect detection effectiveness/functionality coverage). The research methods used include controlled experiments, case studies, field observations, and personal experiences and opinions. Of the references, 36% (cf. S9, S2, S13, and S22) identifies various strengths of ET related to cost-effectiveness by conducting controlled experiments, field observations, and personal experiences and opinions. Of the references, 36% (cf. S9, S11, S3, and S4) states strengths related to the flexibility of ET in test analysis. The research methods used in these studies are case studies, controlled experiments, and personal experiences and opinions.

Table 48 shows the identified strengths of ST. We found eight references discussing the strengths of ST (Table 45).

The research methods used in the identified studies for the strengths of ST include case studies, surveys, controlled experiments, and personal experiences and opinions. Of the references, 38% (cf. S1, S7, and S14) highlights the strengths related to testing quality (defect detection effectiveness/functionality coverage). Of the references, 75% (cf. S1, S14, S15, S16, S18, and S19) mentions strengths of ST related to process flexibility. Of the references, 38% (cf. S7, S14, and S18)

poses ST as good for customer satisfaction especially when there is a need to fulfill legal requirements.

Table 49 shows the identified weaknesses of ET. We found four references that discuss the weaknesses of ET based on case studies, controlled experiments, field observations, and personal experiences and opinions (Table 45). Among the identified four references, 75% states issues related to testing quality (cf. S2, S3, and S7). Of the cited references, 100% (cf. S2, S3, S4, and S7) highlights various weaknesses particularly related to process flexibility. Moreover, some issues related to customer satisfaction are reported by 50% of references (cf. S3 and S4).

Table 50 presents the identified weaknesses of ST. In total, 10 references were identified for the weaknesses of ST (Table 45). The research methods used in the identified studies are controlled experiments, surveys, personal experiences, and case studies. Of the references, 70% (cf. S12, S7, S16, S19, S5, S21, and S6) states that main problems reside in the quality of the design of the test cases. Of the references, 30% (cf. S7, S18, and S17) highlights issues related to cost-effectiveness. Of the references, 10% (cf. S7) mentions the issues related to process flexibility.

Table 47 Strengths of ET

Main category	**Strengths of exploratory testing**
Testing quality (defect detection effectiveness / functional coverage)	Less bogus defects (reduced number of false-positives) (cf. [S1, S2, S4, S7]) • Identification of critical bugs in the system in shorter time (cf. [S1, S2, S4, S22]) • High defect detection efficiency (cf. [S1, S4]) • Investigation and isolation of defects becomes easier as tester directly observes system behavior (cf. [S4, S8, S9, S10, S13]) • Better regression testing (only if test steps are recorded and can later be replayed) (cf. [S1, S4, S8, S10]) recorded and can later be replayed) (cf. [S1, S4, S8, S10]) Cost-effectiveness • Rapid feedback on a new product or a feature as testing can be started immediately without extensive planning and coding of test suites (cf. [S9, S13]) • Quick learning of a new product by the tester who is exploring the system (cf. [S2, S9]) • Low reliance on comprehensive documentation as no documentation is needed, the experience of the tester guides the session (cf. [S9, S13]) • Easy maintenance as there is no need to maintain large test suites including a vast amount of test code (cf. [S9]) • More time allocation in actual testing of the product given that no comprehensive documentation/test code needs to be produced (cf. [S9, S22])
Nature of process (flexibility)	• Simultaneous learning and testing as the tester is exploring the system's functionality while testing (cf. [S4, S9]) • Free exploration as the tester can freely explore the system (e.g., conduct unusual test scenarios) (cf. [S4, S9]) • Improvising on scripted tests as scripted tests are not blindly followed, testers can improvise and explore freely (cf. [S9])

	• Interpreting vague test instructions is possible in ET as the tester can complement with own experience (written automated test scripts based on oracles often require precise instructions) (cf. [S3]) • Diversification in testing as the freedom in writing tests leads to dissimilar results (cf. [S9]) • Utilization of testers' skills as the tester is not restricted by pre-defined rules of how to create test cases (cf. [S3, S11]) • Better product analysis as the product is explored from a usage perspective (cf. [S3]) • Improving existing tests as ET can be used to planning additions and improvements to already existing automated test suits (cf. [S4]) • Identifying missing tests that are overlooked by following a ST approach (additional tests can be found through ET) (cf. [S4]) • Cross-checking the work of another tester (ET should be used complementary to other test activities and can serve as a cross-check to ST test output) (cf. [S3, S9]) • Investigating a particular risk in order to plan a prescriptive test (cf. [S3])

ST, scripted testing; ET, exploratory testing.

5.3.2.2 Interviews

We conducted semi-structured interviews with practitioners in industry to further investigate the experiences and opinions of the domain experts for the weaknesses and strengths for ET and ST as a complementary to what we identified in the literature performing an SLR. In the succeeding texts, we discuss the details of the data collection and the analysis phases of the systematic review.

Data collection:

Four data collection instruments were designed by the two authors of this paper, who also performed the SLR. We first designed the questionnaires with open-ended questions based on the weaknesses and strengths of ET and ST as identified in the literature. In order to assure the quality of the instruments, first, another author of this paper cross-checked the questionnaire. Then, to check whether we need to add more relevant and follow up questions, we piloted the questionnaire with two industry practitioners having the knowledge on both ET and ST. Afterwards, we finalized the instruments.

Table 48 Strengths of scripted testing

Main category	**Strengths of scripted testing**
Testing quality (defect detection effectiveness/functional coverage)	● Higher testing functionality coverage by making conscious/planned coverage decisions (cf. [S1,S7]) ● Complex relationships of a function to be tested identified, cf. [S1,S7] ● Most of the test conditions captured (e.g., all decisions are covered, all combinations of valid and invalid input samples of different valid and invalid classes) (cf. [S14]) ● Test cases depict the overall picture of the perceived quality (cf. [S14])
Nature of process (structured/ guided)	● Oracles availability for the validation of the expected output against the actual value obtained from the test (cf. [S14, S19]) ● Detailed information and guidance available for the tester for test execution (e.g., through testing techniques

	giving concrete guides of how to achieve specified coverage criteria) (cf. [S1, S18, S19]) ● Resource independence in execution as tests can be run automatically when scripted (cf. [S15, S16]) ● Repeatability of the same tests (e.g., for regression testing) (cf. [S1]) ● Reusability of the test cases (cf. [S1]) ● Better risk management (cf. [S14]) ● Better analysis of the system specification from diverse angles as problems in the specification become visible when deriving tests from it (cf. [S15, S18, S19]) ● Quality of the test cases can be validated (e.g., through test case reviews) (cf. [S14]) ● Better tracking of progress (e.g., completed x% of the implemented test cases in the regression test suit) (cf. [S19]) ● Early quality prediction based on test case metrics (cf. [S14, S19])
Customer satisfaction	● Required when legal and regulatory requirements are to be addressed (cf. [S7, S14]) ● Better serves in acceptance testing (cf. [S14, S18]) ● Better serves in release testing (cf. [S7, S14])

ST, scripted testing; ET, exploratory testing.

Table 49 Weaknesses of ET

Main category	**Weaknesses of exploratory testing**
Testing quality (defect detection effectiveness/functional coverage)	● Hard to assess whether all new functionalities and features are tested (cf. [S2, S3]) ● The quality of testing not known because of the dependency on the skills of the testers (cf. [S3]) ● Unavailability of oracles (cf. [S7])
Nature of process (unstructured/ ad hoc)	● Difficulty in prioritizing and selecting the appropriate tests (cf. [S2]) ● Difficulty in reevaluating the test (cf. [S7]) ● Difficulty in monitoring and keeping track of the progress (cf. [S7, S4]) ● Lack of effective risk management (cf. [S7]) ● Repeatability of the tests is challenging because there is no documentation (cf. [S3]) ● Investigating and isolating the actual cause of the problem taking longer time (cf. [S7])
Customer satisfaction	● Not suitable for acceptance, performance, and release testing (cf. [S3]) ● Less accountability and audit ability (cf. [S3, S4])

ST, scripted testing; ET, exploratory testing.

Table 50 Weaknesses of scripted testing

Main category	Weaknesses of scripted testing
Testing quality (defect detection effectiveness/functional coverage)	● Defect detection effectiveness and functionality coverage rely on the quality of the test case design (cf. [S7]) ● Dependency on testers' skills, experience, and domain knowledge for test case design (cf. [S7]) ● Test cases being prone to human error (e.g., coding mistakes in written test cases) (cf. [S5, S12, S19]) ● Quality of the test cases not known until their execution (cf. [S6, S19]) ● The possibility of redesigning the test cases under time constraints to cause low quality design (cf. [S16, S20, S21]) ● Not suitable for regression testing when test cases are not well maintained/updated (erosion of regression test suit) (cf. [S21])
Cost-effectiveness	● Exhaustive and protracted (cf. [S7]) ● Designing and documenting require considerable effort (cf. [S18]) ● Often overruns the assigned budget and time (cf. [S7, S18]) ● Test cases not sufficient for the entire system life cycle (cf. [S18]) ● Durability of the test cases not known (cf. [S7]) ● Reusability and maintenance of test cases can be quite expensive (cf. [S17]) ● Redesign or revision due to poor quality of the test cases increase the cost more (cf. [S17])
Nature of process (inflexibility)	● Prescriptive process does not give freedom to the testers (even in cases where the test cases quality is not good) (cf. [S7]) ● The testers skills not utilized during test execution (cf. [S7]) ● Difficulty in prioritizing the test cases (cf. [S7])

We conducted interviews with five persons having worked as software testers, test managers, practitioners, or consultants. Our sampling of the interviewees was purposeful as we focused on practitioners with a very high level of experience in both types of processes (minimum 10 years), that is, ET and ST processes. In order to make this research more authentic and reliable, we selected interviewees who hold a senior position in reputable organizations. The experience adhered by such professionals was of great essence as they are also involved in interacting with stakeholders. By conducting interview of such people, it gave us broader insights of the problem domain from multiple perspectives. Given that a high requirement was put on the experience, the number of people to ask was limited, and it was a challenge to identify a high number of them. Hence, we focused on senior testers and also on people known in the testing domain with respect to their knowledge on ST and ET (two interviewees were, e.g., identified through keynotes they gave on the topic). The people interviewed fulfilled our criteria, but their number was limited given the previously mentioned requirements. Some diversity was achieved by interviewing people from different companies.

Interviewee 1 has been working as a test manager in Logica AB (Sweden) for the last 2 years. In the past, he worked for a number of companies including Microsoft and UIQ Technologies. Interviewee 2 has been working as a consultant for Telenor AB (Sweden) for the last 2 years. Interviewee 3 is the owner of DevelopSense (Canada) and has been providing consultancy, training, coaching, and other services in software testing. Interviewee 4 has been working for Maquet Critical Care AB (Sweden) as a test manager for the last 6 years. Interviewee 5 is the founder of Satisfice Inc.

(USA), which is dedicated to teaching and consulting in software testing and quality assurance. Most of his experience is with market-driven software companies such as Apple Computer and Borland.

Four of the interviews were conducted face-to-face and one online through Skype because of geographical distance. We presented the interviewees the aims of this research before the interviews. The duration of each interview was between 60 and 90 min. We took notes and recorded the interviews using a digital recorder. The data collected from the interviews were transcribed* in order to eliminate any irrelevant information.

Data analysis and results

The transcribed outputs of the interviews were qualitatively analyzed by applying the notice, collect, and think technique [138]. This is a nonlinear qualitative analysis model and consists of three phases: noticing, collecting, and thinking phases. These phases are iterative, recursive, and holographic in nature.

First, the two authors who also performed the SLR analyzed the interviews. Then, another author of this paper made an independent analysis. The results were cross-checked, and then after a discussion, the codes, the main categories, and the connections in between the main strengths and weaknesses were agreed upon solving very few disagreements also by consulting the interviewees.

In the noticing phase, all the relevant information highlighted by the interviewees regarding the strengths and weaknesses were noted using a heuristic coding approach. For example, during the noticing phase, for ET, we captured the following codes from the interviewees: 'less time', 'less documentation', 'more focused documentation', 'more time on actual testing', 'better resource utilization', and 'rapid feedback and quick learning of the product'. As for ST, we identified the codes as 'time consuming', 'exhaustive', 'too much documentation', taking time', 'less costly if test cases can be automatically generated', and 'time depends on the quality desired.'

Then, during the collecting phase, we sorted the weaknesses and strengths and categorized them under main categories based on the similarities and differences between them. Thereby, we identified 'cost-effectiveness' as a main category.

In the thinking phase, both the codes and the main categories were reexamined. Here, we observed that some of the strengths and weaknesses have connections. For example, one of the interviewees mentioned that even though ST takes more time because of too much documentation (hence, less cost-effective), ST was required especially in cases where there was a need to have documented proof of testing where legal and regulatory requirements were to be met. This was a good example showing why one approach should not replace the other, but rather a hybrid process, which optimizes the strengths and minimizes the weaknesses of both approaches, is required. Thereby, we used these insights for identifying improvement opportunities for an HT process as a complementary to what has been captured from the SLR.

In the succeeding texts, we summarize the results of the analysis for the strengths and weaknesses of ET and ST as experienced in industry. However, this time we preferred reporting the strengths and weaknesses in a narrative form instead of reporting them only independently as we did for the SLR (Table VIII shows the additional categories identified in comparison to SLR findings). This is because of that, through interviews, we also could capture the totality of philosophy as expressed by the interviewees for the strengths and weaknesses of ST and ET that might help in identifying the improvement opportunities for a hybrid process.

Strengths and weaknesses of ET: The interviewees were of the opinion that unstructured and flexible process in ET could provide either strengths or weaknesses depending on the conditions. As for the strengths, they mentioned that a tester could freely explore different areas of the product and that ET was a process of simultaneous learning and testing. The interviewees had an agreement on the cost-effectiveness of ET because of less time spent on documentation (i.e., focused documentation for only logs, test notes, and videos after the execution), better resource utilization, rapid feedback, and quick learning of the product. Related to this, three of the interviewees mentioned that defect detection efficiency was likely to be high in ET as more time was spent on actual testing rather than on test design and comprehensive documentation.

Moreover, three interviewees were of the opinion that ET could achieve better regression testing and help in identifying most of the critical bugs. Three interviewees stated that ET was handy in investigating more risky parts of the software. Two interviewees claimed that customers were more

satisfied as more bugs and also critical ones could be identified. All five interviewees highlighted one key strength of ET as a better utilization of the testers' skills. The reason was stated as testers to become more responsible, engaged, motivated, and creative, while they were given freedom. On the other hand, the interviewees also emphasized that this strength could also become a major weakness in some situations as the quality of testing became dependent on only testers' skills and the domain knowledge. According to three interviewees, the availability of an oracle becomes an issue when the application is too complex, the skills and the domain knowledge of the testers are insufficient, and if the time is running out, and functional specifications have not been updated. Moreover, they mentioned that the flexibility in the process caused significant difficulties in terms of managing, prioritizing, and tracking the tests. Four interviewees were of the opinion that managers and organizations were reluctant to implement ET because they thought they might lose control over testing. Two interviewees added that automation support was not possible for ET.

All four interviewees agreed on the fact that using ET alone is not suitable in some cases, and it should be used as a complementary approach to prescriptive approaches. One of the interviewees stated that conducting only ET on complex application alone was not suitable and should be combined with other test approaches in order to ensure testing of critical functionality of complex and real time applications. One of the interviewees emphasized that ET was an approach and not a technique and, therefore, it was already being used with prescriptive techniques as ST. Two of the interviewees raised the need to have a more structured process for ET for better management. They also mentioned that ET could serve well in terms of testing quality if used together with a prescriptive approach such as ST.

Strengths and weaknesses of ST: Similar to ET, all interviewees stated that the structured and formal process in ST could provide either strengths or weaknesses depending on the conditions. As for one major strength, three of the interviewees mentioned that ST was required especially in cases where there was a need to have documented proof of testing where legal and regulatory requirements were to be met. Furthermore, one interviewee added that ST also served well for the acceptance testing. All interviewees were of the opinion that ST provided better test guidance to testers on specifying desired outputs in test oracles and also could support testers in creative testing. All interviewees mentioned that quality of testing (functionality coverage and defect detection efficiency) was depended on test case design quality. Moreover, two interviewees said that test case design quality was dependent on skills, experience, and domain knowledge of the designer, as well as on previously produced documents, such as software requirements specification or test plan. They stated that the test quality would be high if the design quality was high. Another benefit, pointed out by an interviewee, was early quality assurance with respect to requirements specifications. He stated that bugs could be found before testing starts when designing test cases from requirements specifications. On the other hand, two of the interviewees stressed the fact that the quality of the test case design could not be known before testing. Three interviewees mentioned that a tester was not free to make decisions even if the test cases were not designed properly.

Three of the interviewees stated that most of the time, they experienced good functionality coverage in their companies when using ST. They added that this was because of documenting the test cases in correspondence with the requirement specification provided better functionality coverage. One stated that he experienced low defect detection efficiency. Two of the interviewees mentioned that finding defects by ST was difficult as it might be impossible to follow each and every step of the test case. About increasing testing quality, all interviewees were of the opinion that the quality of testing would increase if ET were used as a complementary approach to ST.

Low cost-effectiveness and difficulty in managing large number of test cases were stated as two major weaknesses of ST. All interviewees were of the opinion that designing, documenting, and executing test cases were too much time consuming and costly. One interviewee mentioned the need that the test cases should be updated continuously in the software development life cycle as the requirements change. Moreover, two interviewees added that the test cases required revision and/or redesign in cases of low quality design. These last two requirements bring more management overhead and thus cost.

5.3.3 Summary of the systematic literature review and interview results

We performed qualitative comparative analysis [139] to identify commonalities and diversities between the results obtained from the SLR and the industrial interviews.

The results of industrial interviews showed that most of the weaknesses and strengths identified from literature have also been experienced in industry. Therefore, we also distinguish findings reported both in the literature and by the interviewees from the new findings identified during the interviews. Furthermore, in the following paragraphs, we also discuss the new and more insights that we captured from the interviews providing a bigger picture with connections between the strengths and weaknesses in addition to what has been reported as individual strengths and weaknesses in the literature.

The weaknesses of ET were attributed to ET being an unstructured and ad hoc process (which causes difficulties in planning, managing, and tracking the testing process) or related to dependency of testing quality on the skills, experience, and domain knowledge of the testers.

For ST, many weaknesses were reported to be related to cost-effectiveness and dependency of testing quality on test case design quality. As for the strengths, many strengths for ET were reported as being related to cost-effectiveness, process flexibility, and testing quality; whereas for ST having a defined and repeatable process, testing quality, and being independent from the skills of testers during the test execution.

During the interviews, we identified some more aspects, which have not been reported in literature. For example, focused documentation was found to be strength for ET. As for ST, another strength identified is early quality assurance. One of the interviewees stated that bugs could be found before testing starts when designing test cases from requirement specifications.

On the other hand, one weakness identified for ET is the reluctance of managers in organizations to implement ET because of having the fear to lose control over testing. Another weakness of ET is the difficulty in interpreting the test results because these are generated based on the testers' own experience and intuition. We also found that the interviewees do not believe that automation support is possible in ET.

Furthermore, from the interview results, we also could identify the conditions for when strength of one approach could become a weakness and vice versa. For example, one significant conclusion is that quality of testing in ET and ST depends on some conditions. A few studies in literature reported ST to perform well for functionality coverage but poor for defect detection efficiency in comparison to ET.

However, the interviews revealed that quality of testing in ST depends on the test case design, which depends on the skills, experience, and domain knowledge of the test designers as well as the previous documents from which the product requirements are inherited. On the other hand, the quality of the testing in ET depends on skills, experience, and domain knowledge of the testers who execute the tests.

Therefore, when the testers lack some of these attributes, for example, domain knowledge and experience, it would be better to use either ST alone or ET as a support for ST. Or, if there is a doubt about the quality of previous documents (such as requirements specification) from which the test cases are to be derived, then ET might work better if the testers have domain knowledge and experience.

Another significant conclusion from the interviews is that all interviewees emphasized using ET as a complementary approach to ST as they all believe that this would bring many benefits and help in overcoming major weaknesses. Hence, we identified the following improvement opportunities for designing an HT process:

- Utilizing the skills and the domain knowledge of the testers during both design and test execution. In ST, the quality of testing depends on the 'test case design', and the test case design quality depends on the test case designer skills, experience, and the domain knowledge as well as the previous documents from which the product requirements are inherited. In ET, the testing quality depends on the skills, experience, and domain knowledge of the testers who execute the tests. Therefore, there is a need to increase the utilization of all available test skills and expertise both in design and execution.
- Defining a structured process with some level of flexibility. This is required to enable better management and increased motivation of the testers by incorporating the creativity and skills of them as well as overcoming the risk of not being able to take an action when they encounter poor test case design. The defined process should also require more focused and less documentation in order to increase cost-effectiveness.

In the next section, we present the mapping of strengths of one approach to the weaknesses of the other to identify how to design the HT process by incorporating different aspects of ST and ET to overcome the weaknesses in the compromise form.

5.3.4 PHASE 2: Mapping exploratory testing and scripted testing in relation to strengths and weaknesses

A mapping process is a method of identifying problems and their solutions in a structured way. In this investigation, we used mapping process [139] [27] as an important feature of research technique evaluation method, which helps to develop the mechanisms that support to find the solution of one testing approach weaknesses considering other approach strengths. For this, we list down one approach weaknesses against the other approach respective strengths.

Table 51 shows the mapping of the identified strengths of ET as candidate solutions to the weaknesses of ST. Table 52 shows the mapping of the identified strengths of ST as candidate solutions to the weaknesses of ET. Observe that the benefits and weaknesses were previously categorized into testing quality, cost-effectiveness, nature of process, and customer satisfaction. The categories were used to match related benefits and strengths to each other. As an example, the ST issue of 'Prescriptive process does not give freedom to the testers' under the category of the nature of process is addressed in ET through 'free exploration'.

Overall, the intention is to leverage on the benefits listed on the right column of Table 51 and Table 52 by defining a structured prescriptive process, which at the same time gives flexibility to testers to conduct ET. In other words, by having both aspects in one compromise process would aid in overcoming some weaknesses of ST and ET, whereas the strengths of both processes are utilized.

In the following section, describing the phase3 of this research, the hybrid process incorporating ST and ET is presented. We provide rationales on how the different activities map to the strengths and weaknesses identified earlier.

5.3.5 PHASE 3: Designing the hybrid testing process

We designed the process iteratively. Our design started out with creating an initial version of the process based on the results of phase1 and phase 2. We start by presenting the design rationales for our initial process.

Table 51 Mapping of the strengths of exploratory testing to the weaknesses of scripted testing

Weaknesses of scripted testing	Strengths of exploratory testing as Candidate Solutions
Testing quality	Testing quality
● Defect detection effectiveness and functionality coverage rely on the quality of the test case design	● Less bogus defects (reduced number of false-positives)
● Test case design depends on the skill, experience, and domain knowledge of the testers	● Identification of critical bugs in the system in shorter time
● Test cases are prone to human mistakes	● High defect detection efficiency
● Quality of the test cases not known until their execution	● Investigation and isolation of defects become easier as tester directly observes system behavior
● Redesigning the test cases under time constraints may cause low quality design	● Better regression testing (only if test steps are recorded and can later be replayed)
● Not suitable for regression testing when test cases are not well maintained/ updated (erosion of regression test suit)	
Cost-effectiveness	Cost-effectiveness
● Exhaustive and protracted	● Rapid feedback on a new product or a feature
● Designing and documenting require considerable effort	● Quick learning of a new product by the tester who is exploring the system
● Often overruns the assigned budget and time	● Low reliance on comprehensive documentation
● Test cases are not sufficient	● Easy maintenance as there is no need to maintain large test suites

for the entire system life cycle	
● Durability of the test cases are not known	● More time allocation in actual testing of the product
● Reusability and maintenance of test cases can be quite expensive	● Focused documentation
● Difficulty in prioritizing the test cases	
● Redesign or revision due to poor quality of the test cases increase the cost more	
Process (inflexible)	Process (flexible)
● Prescriptive process does not give freedom to the testers	● Free exploration
● The testers skills not utilized during test execution	● Simultaneous learning and testing
● Difficulty in prioritizing the test cases followed	● Improvising on scripted tests as scripted tests are not blindly
	● Interpreting vague test instructions is possible in exploratory testing
	● Diversification in testing
	● Better utilization of the skills of testers
	● Better product analysis
	● Improving existing tests
	● Identifying missing tests that are overlooked by following a scripted testing approach
	● Cross-checking the work of another tester
	● Investigating a particular risk in order to plan a prescriptive test

5.3.5.1 Method engineering for initial hybrid testing process

Design goals: In order to identify the candidate solution, we take into consideration all the weaknesses and strengths of both approaches identified through SLR and from interviews. If one approach lack in providing some of the aspects in a candidate solution, it is taken from other approach and so forth. In other words, by having both aspects in one compromise process would aid in overcoming some weaknesses of ST and ET, whereas the strengths of both processes are utilized. From the comparative analysis, we showed that weaknesses in one approach are potentially improved through strengths in the other process.

Table 52 Mapping of the strengths of scripted testing to the weaknesses of exploratory testing

Weaknesses of exploratory testing	Strengths of scripted testing as Candidate Solutions
Testing quality	Testing quality
● Hard to assess whether all new functionalities and features are tested	● Higher testing adequacy by making conscious/planned coverage decisions (functionality coverage)
● The quality of testing not known because of the dependency on the skills of the testers	● Complex relationships of a function to be tested identified
● Unavailability of oracles	● Most of the test conditions captured (e.g., all decisions are covered, all combinations of valid and invalid input samples of different valid and invalid classes)
● Difficulty in interpreting the test results	● Test cases depict the overall picture of the perceived quality
	● Early quality assurance
Process (ad hoc and unstructured)	Process (structured and guided)
● Difficulty in prioritizing and selecting the appropriate tests	● Oracles availability for the validation of the expected output against the actual
● Difficulty in reevaluating the test	● Detailed information and guidance available for the tester for test execution
● Difficulty in monitoring and keeping track of the progress	● Resource independence in execution
● Lack of effective risk management	● Repeatability of the same tests
● Repeatability of the tests is challenging	● Reusability of the test cases

because there is no documentation ● Investigating and isolating the actual cause of the problem taking longer time ● Fear to lose control over testing ● Automation support not possible	● Better risk management ● Better analysis of the system specification from diverse angles ● Quality of the test cases can be validated ● Better tracking of progress ● Early quality prediction based on test case metrics
Customer satisfaction ● Not suitable for acceptance, performance, and release testing ● Less accountability and audit ability	Customer satisfaction ● Required when legal and regulatory requirements are to be addressed ● Better serves in acceptance testing ● Better serves in release testing

5.3.5.2 Process definition

We based the HT process on ISO/IEC 29119 (2009), which is a software testing standard aiming to provide one definitive standard that captures vocabulary, processes, documentation, and techniques for the entire software testing lifecycle. The testing processes in this standard include organizational, management, and fundamental test processes.

When defining the HT process, we considered only the management and fundamental processes as given below. Organizational processes were not in the scope of the HT process definition, as these processes include definition of organizational test policy and test strategy that are outside of the main research focus of this paper.

- Management processes:
  - Test planning
  - Test monitoring and control
  - Test completion
- Fundamental processes:
  - Test design and implementation
  - Test environment setup
  - Test execution
  - Test incident reporting

In order to incorporate ET concepts into HT process definition, we used the session-based test management process defined by Bach [140]. The reason for choosing this process definition was that during our interviews, we identified that it is a well known approach in industry. In session-based test management, a test session is the basic testing work unit. This session is an uninterrupted block of reviewable and chartered test effort, that is, each session is associated with a test mission. Every test session is debriefed after execution. The debriefing occurs as soon as possible after the session. The test outcomes, issues, bugs, and related information are stored on the 'session sheets'.

As we previously reviewed the strengths and weaknesses with respect to testing quality, cost-effectiveness, structuredness of testing process, and customer satisfaction, we discuss how these four attributes were incorporated in the HT process design (also referred to as fragment selection in method engineering [131]). Hereafter, this reasoning has been taken into the collaborative design activity with the practitioners

The bullets listed showed the initial idea of the process, in which it is tried on how to incorporate these four main attributes in the HT process. Hereafter, this is presented to the interviewees to obtain the feedback:

- Testing quality: Following Sections 5.3.2, we found out that testing quality (defect detection effectiveness and functionality coverage) depends on a couple of conditions for both ST and ET. For example, testing quality for ST depends on the test case design quality, which depends on the test designer skills. As for ET, the quality depends on the skills and the domain knowledge of the testers. Considering different quality aspects of each approach, in HT process, we need to adopt these aspects of both processes. For this, we unify the subsection's 'test design' and 'test execution' of both approaches in a formal manner. The idea is to achieve better coverage by defining requirement-based test cases (RBTC) [141] and test missions. For example, through the

requirements, one can check whether all highly prioritized requirements have been tested. In order to achieve the defect detection effectiveness, we allow the testers to explore the product under testing freely and to utilize their intuitions and experience in identifying defects. In addition, HT also allows testers to execute the designed RBTC and test missions. Following the proposed HT process, our proposition is that the strengths of both the approaches are aligned, and the testing performed would be planned, and effective with the focus on complex function and having ability to identify critical defects.

- Cost-effectiveness: Following Sections 5.3.2 , we found that ST is not a cost-effective approach where ET is cost-effective. Scripted testing highly relied on the test design phase where ET is meant to be simultaneous test design and execution. The HT process is meant to have cost-effectiveness by adopting both ST and ET attributes. For this, we tried to lessen the contribution of test design phase by introducing RBTC [141] and test missions in the HT process. The consideration of high-level test cases such as RBTC and test missions lessen the dependability on the formal test case design, which includes each aspect of conditions in the code, input data, and GUI under test. Thus, our design proposition is that the use of high-level test cases in the form of RBTC and test mission took less time

  in design, without much compromising on the benefits of the test design phase of ST. In complement to RBTC and test missions, we introduce a step of free exploration that could allow more time being spent on the actual testing task, rather than designing the test. Subsequently, the time saved in the test design phase should make the HT process more cost-effective in comparison to ST, and the introduction of free exploration may help to attain better quality in a form of defect detection efficiency (as is evident from our literature review).
- Unstructured process: Following the findings shown in Table 49, ET has no process structure, it is meant to be free exploration only, whereas ST has a structured process. This had negative consequences, such as difficulty to prioritize tests, reevaluating tests, monitoring progress, and so on. The attempt is to design HT in a way of not having a strict process but a semi-structured process that adopts strengths of both the approaches. Thus, considering the structure of ISO/IEC 29119 in conjunction of ET strength-free exploration, we aimed to provide HT a semi-structured process that would have a formal structure with free exploration being a part it. We also achieve this by allowing flexibility in work flows. The process is also designed so that practitioners are able to decide which activities are emphasized, depending on testing outcomes, type of systems, and type of tests. Further, the process is iterative in nature.
- Customer satisfaction From Sections 5.3.2 , we observe that customers are very reluctant with ET, while they are satisfied with ST. The primary reason of customers not being satisfied with ET is the lack of a formal test design phase, on which they can evaluate their product, and which can be used for to document the fulfillment of contractual requirements. In the HT process definition, the attention is given to make such a process, which could satisfy the customers. Therefore, we include the definition of test design phase that could allow overcoming the reluctance of customers. This may help the HT process to be useful for legal requirements and acceptance/release testing. In addition, it also allows test managers to have control of their testing activities.

5.3.5.3 Collaborative design

We co designed the HT process with the help of practitioner feedback. The practitioner feedback was collected by conducting semi-structured interviews with testing experts. We conducted four face-to-face semi-structured interviews to receive feedback on the mapping of the strengths and weaknesses of ST and ET, and also for the proposed HT process. Here, we should mention that the development and refinement of the HT process was an iterative process considering the feedback of the interviewees.

Two of the interviewees are working for Logica AB (Sweden) as a test manager and a project manager. The other two interviewees work as test managers for Maquet Critical Care AB (Sweden) and Toolaware (Sweden). Three interviewees being involved in the collaborative design have also participated in the interviews.

A data collection instrument was designed to receive feedback and suggestions for the proposed

HT process. To assure the quality of the instrument, all the questions were cross-checked by the authors of this paper. All the interviews were presented with the RQs before the interviews. A number of scenarios were shown in order to validate or grasp the improvement opportunities in the HT process. Approximate duration of each interview was between 30 and 45 min. The data were collected manually by taking notes and also by recording with the consents of the interviewees. The data collected were transcribed, and the irrelevant materials were omitted (i.e., the key points of the interview were separated from the general discussion).

The feedback given by the practitioners, as well as how it has been utilized in the process definition is presented in the following:

Feedback of interviewee 1: Interviewee 1 suggested that the strengths and weaknesses of both test approaches were concise and detailed. Her concern was how in reality the strengths of each testing approach will work out on real projects and provide benefits. She added that the weaknesses of ET and ST were generic, and that in practice, there could be many ways to deal with such issues by other means. However, she affirmed providing a solution inferred from strengths of both test approaches and found attempting to resolve the weaknesses in this way as quite innovative. She also had some reservations on the debriefing session because she considered that managing the test team might even take more time because of having debriefing session. She recommended involving test leaders in HT process. Reflection on feedback: the debriefing session was not removed based on the feedback by the practitioner, the reason being that Interviewee 4 provided use-ful suggestions of how to utilize the debriefing session better. Overall, the practitioner agreed with the main idea of formation of HT process keeping the previously mentioned context as no further changes were suggested. We highlight that when executing the process, the suggestion of the practitioner should be followed to involve test leaders.

Feedback of interviewee 2: Interviewee 2 said that mapping the strengths of both testing approaches to the weaknesses was a good way to compare both testing approaches. He mentioned that mapping was an ideal way of presenting the solution based on theoretical constructs, but practically, this mapping might not provide with 100% solution. He stated that it was a high-level presentation of strengths to weaknesses, but still all strengths of both test approach might have several weaknesses that may be associated with other indirect measures. He said that RBTC should only be used complementary, specifically where GUI testing was required, and test cases were hard to codify. Reflection on feedback: given our design, RBTC is complementary and can always be combined with free exploration, which indicates that our design addresses the practitioner's concerns. As the practitioner highlights, different emphasis might be given depending on the type of testing conducted (e.g., GUI testing).

Feedback of interviewee 3: Interviewee 3 highlighted that mapping strengths to weaknesses was an appropriate way of defining a compromise process based on ET and ST. He evaluated the mapping process and mentioned that the approach was quite elaborative. When we presented him with the initial process flow description, he added that he was not fond of flow boxes connected to each other telling him what to do, and he was of the opinion that the context should decide which box should be used in a specific situation. He also recommended the introduction of free exploration in order to learn about the application, that is, before, after, or during the execution of RBTC. He added that free exploration would provide an edge to the testers as they would be able to immediately look for any major abnormality in a very short span of time. Reflection on feedback: the flow boxes were retained for the purpose of presenting the process in this paper. It is important, however, to illustrate the flexibility of the flow through the process, which makes it semi-structured as pointed out earlier. Hence, formal descriptions (flow boxes or activity diagrams) might not be suited to represent the process to practitioners. Rather, a narrative form should be preferred. Free exploration has been emphasized in our process more based on this interviewee's feedback.

Feedback of interviewee 4: Interviewee 4 was of the opinion that there should be more flexibility in using any sort of test cases, not only RBTC. He also suggested that these RBTC should be made more generalized, and one should not limit to RBTC only. He said that it should be up to the testers or managers to decide upon what they need and require out of testing. And, he highlighted that performing ET at the beginning of testing life cycle could provide many benefits, and therefore, it should also be incorporated in the HT process. He pointed out that exit criteria should be explicitly discussed. He also recommended that upon the conclusion of every debriefing session, more test missions should be drafted based on the testers report and intuitions and that these newly devised test missions should become the input for further session executions. Reflection on feedback: The

flexibility of the process is illustrated by showing different alternative paths through the process. Furthermore, the debriefing session is retained for the purpose specified by the interviewee.

After evaluating the mapping and the HT process, the HT process was refined based on the feedback received.

5.3.5.4 Defined hybrid testing process

Considering the design rationales, as well as the feedback by the practitioners, the brief descriptions of each sub process in HT (Figure 33) are given in the following paragraphs:

Test planning: The purpose of test planning in HT process is to plan, document, and communicate all the necessary and required information to all the stakeholders about what is going to happen regarding testing. HT test planning is inherited from the ST process. In order to have an improved planning process, the strengths of ET planning are also incorporated. These include specification of the scope and time, allocation of resources, risk planning for risk management, and mitigation.

Test mission design and implementation: HT test design, introducing the RBTC [141] and test missions would help in enabling high functionality coverage and defect detection effectiveness in addition to cost-effectiveness through reducing the test bed size. The RBTC specify those test cases that are defined only from the requirement specification. The 'test mission' is a concrete instruction for testing and the problem being looked for.

Test environment setup: For HT, there is a freedom for the selection of test environment. Based on the test case design and implementation, the test environment in which the test will be executed is established and maintained.

Test execution: Both RBTC and the test missions are executed, which were designed in test design phase. First, a tester has given the freedom to freely explore the application in order to learn and obtain knowledge about it. After that, RBTC and then the test missions are executed, and the execution artifacts are recorded. A session is a particular time slot assigned to a specific test mission in which test mission has to be executed. A session time is an uninterrupted block of test time. A session time may last from 30 to 90 min.

Test incident reporting: The purpose of test incident reporting in HT is to report the issues identified in the test execution to the relevant stakeholders in order to conduct further actions on the reported problems. The session sheet taken from the ET is used to report all incidents happened during the testing, and it has information about tested area, test notes, issues, faults, bugs, failures relevant information, or any other ambiguities related to the functionality. This provides focused documentation related to the testing with all relevant information.

Debriefing: The purpose of debriefing session in HT is to obtain the input of a tester on the test mission, which was assigned to him, and to discuss about his observations. A debriefing session should also provide coaching to the tester regarding further test activities that needed to be performed. If required, a debriefing session can lead to the derivation of many test missions. After the completion of session, a debriefing session is set up between the tester and a test lead.

Test completion: The purpose of test completion criteria is to make sure that the useful test assets such as test plans, test cases, and session sheets are made available, and all the results are documented, recorded, and communicated to the relevant stakeholders. Test completion criteria are met when an agreement has been reached that the testing being performed and managed is complete.

Test monitoring and control: The purpose of HT monitoring and control is to ensure whether all the activities as specified in test plan are aligned with the actual execution of those activities. Hybrid testing monitoring and control provides assurance of whether or not the testing being performed is in line with the defined test plan. All the processes within the HT process, that is, tests design, test execution, test incident reporting, and test completion are being monitored and controlled.

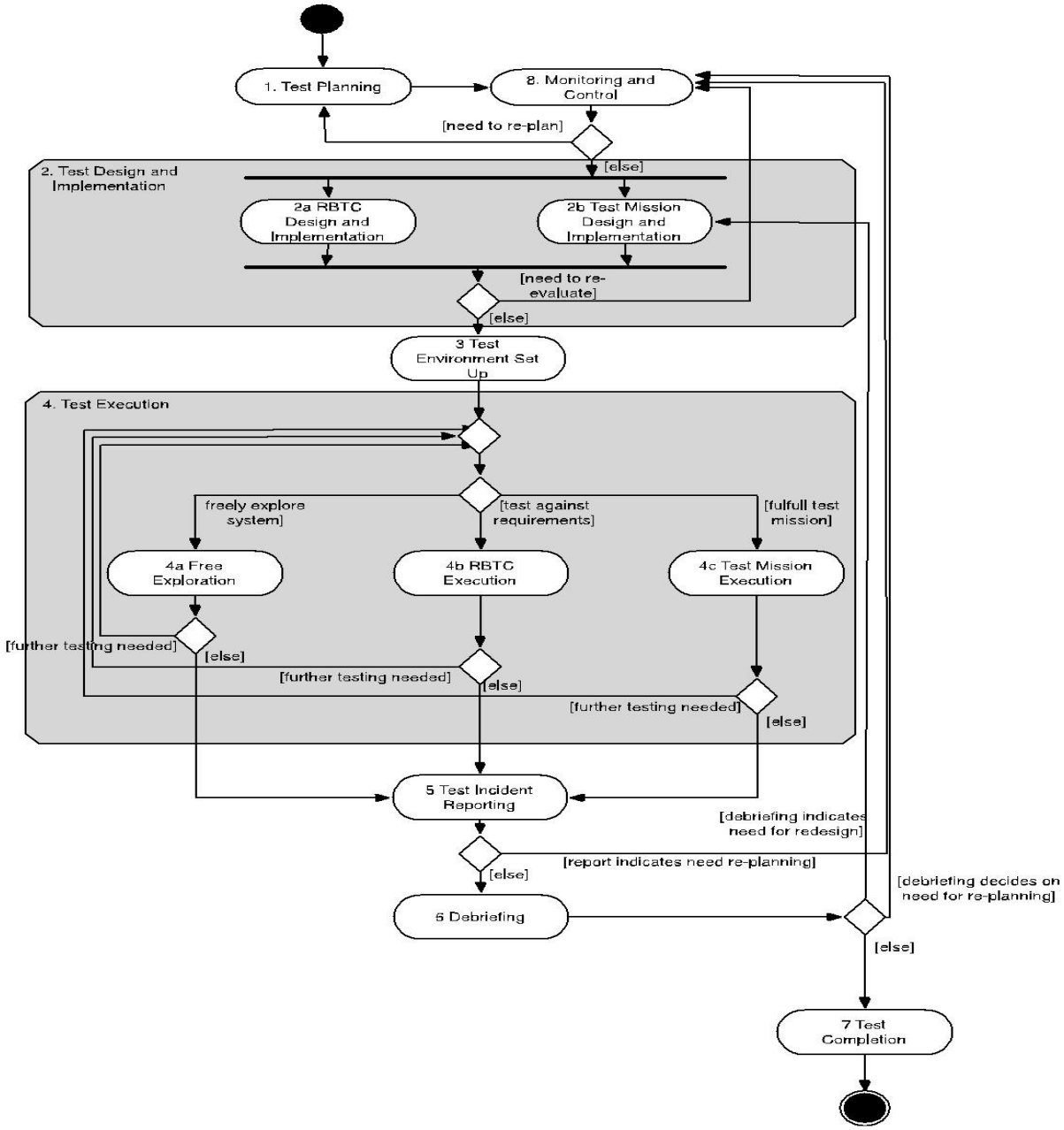

Figure 33 Process of hybrid testing.

The flow of the process is designed to be flexible and iterative (Figure 33). In the beginning of the process, test planning influences monitoring and control (e.g., which test targets should be monitored), while defining the targets test planning can be influenced and refined.

After having specified the plan and how to monitor and control, test design and implementation are conducted, and both RBTC design and test mission design are executed. With these activities completed, the outcome can be monitored and controlled, and eventually updates are made in the designs.

Thereafter, the test environment is set up. This is the prerequisite to conduct test execution. The test execution part is highly flexible. One can, for instance, start with an exploratory session, followed by test mission execution and RBTC. Another scenario is to only do free exploration. How much effort is spent and how many executions of the particular activities are conducted are not pre-specified and might vary

with the testing context (e.g., type of testing performed or the type of system to be tested).

After having completed the test execution, test incidents are reported, and debriefing is conducted. At any point, one can return to the monitoring and control activity and, depending on the outcome, decide on how to continue in the process. That is, it is possible to continue at any point in the process after completing monitoring and control. We have not illustrated this in the Figure to sustain its readability.

5.3.6 Threats to validity

Because the HT process definition is based on the results of the SLR and interviews, the validity threats for each indirectly influence the validity of the proposed HT process. The internal and external validity threats for the SLR, the interviews, and the experiment are discussed in the following paragraphs.

5.3.6.1 Systematic literature review

For the SLR, one of the validity threats was associated with the possibility of missing any important publication. In order to eliminate this threat, when designing the search strings, we used the synonyms and alternative terms for the keywords referring to linguistic dictionaries while limiting them within the context of software engineering. When deciding on the keywords, we also checked the general terminology used in the testing field (e.g., testing standards such as ISO/IEC 29119 and key publications) not to miss any important keyword. The search strings were verified by conducting trail searches, and a preliminary search was carried out in order to identify the relevant literature by the help of the BTH (Sweden) librarians. Furthermore, we asked an expert in the area to review the design of the literature review as well as the list of included papers after the review to make sure that no important study is missed.

The quality of the data extraction form was checked by one of the other authors of this paper, who was not in the review team. The reviewer checked, in particular, the relevancy of the data to be extracted as well as whether any important information that needs to be captured is missing. Then, the forms were slightly revised after the pilot searches to include categories of relevant areas to study that helped in the uniformity of the coding.

To avoid selection bias during the selection process of the primary studies, two reviewers worked together to decide on the inclusion and exclusion of the studies. In addition to this, we also asked an external reviewer to check the final list of primary studies included in the SLR. As for the analysis phase, one threat could have been an individual bias when identifying the codes and the main categories for the strengths and weaknesses. In order to reduce this threat, a pair of reviewers worked together and identified the constructs after joint discussion.

5.3.6.2 Industrial interviews

For the interviews, the possibility of missing any important question in the questionnaires was one of the potential validity threats. In order to avoid this, we designed the questionnaires based on the findings of the SLR. Furthermore, we also included open-ended questions to identify additional strengths and weaknesses by letting the interviewees discuss their experiences.

Another threat could be the misinterpretation of the question and answers during the interviews. This threat was minimized by reviewing of the questionnaire. A number of senior software engineering students studying at BTH (Sweden) were asked to review the questions for ensuring the clarity of the meaning before conducting the actual interviews. A recording device was used to record the interviews, and the transcribed interviews were shared with the interviewees to avoid any misunderstanding.

Another threat was related to the fact that the data were gathered in the form of qualitative information during the interviews. A risk of misinterpretation of qualitative data exists because of the possibility of multiple interpretations. This risk was reduced by cross-checking the findings and also by getting feedback on our interpretations from the interviewees (member checking).

During the analysis phase, the two authors who also performed the SLR analyzed the interviews. To avoid researcher bias, another author of this paper made an independent analysis. The results were cross-checked, and then after a discussion, the codes, the main categories, and the connections in between the main strengths and weaknesses were agreed upon, solving very few disagreements also by consulting the interviewees.

There is also a threat to external validity because of a low number of interviewees. It was essential to involve practitioners with a vast amount of experience in ST and ET, as this provides the greatest potential to obtain additional experience-based insights complementing the results of the literature

review. This constraint limited the number of persons we could involve in the research process. Overall, it was a trade-off between the levels of experience of practitioners versus the number of practitioners involved. It is important to highlight that for P1 and P2, both the literature review and the practitioners, complement each other. Having only one source would increase the risk of losing valuable information. Using source triangulation reduces the threat related to the number of responses. We required detailed and qualitative insights to design our HT process; therefore, we chose a qualitative data collection instrument (interview) over a sampling-based instrument (questionnaire). In S3, the HT process was co designed with the practitioners, also involving both sources (practitioners and literature). The practitioners only had one contradiction in opinion of how to design the actual process (i.e., whether to have a debriefing session or not). Other suggestions were valuable complements to our suggested process (e.g., what to emphasize in GUI testing).

5.3.7 Discussion and conclusions

The conclusion is divided into two parts. The first part summarizes the results, and the second part presents the implications for practitioners and researchers.

5.3.7.1 Summary of findings

This study has mainly two contributions. First, the strengths and weaknesses of ST and ET were identified. Second, by bringing into light the improvement opportunities for a new testing process through unification of ST and ET in a compromise form, an HT process was defined in collaboration with practitioners.

What are the strengths of ST and ET?: The identified strengths and weaknesses were recognized under four main categories: (i) testing quality (defect detection effectiveness/functionality coverage); (ii) nature of the process (structure/flexibility); (iii) cost-effectiveness; and (iv) customer satisfaction. Major strength categories for ST were found to be related to the nature of the process, testing quality, and customer satisfaction. The structured and guided process of ST provides benefits such as repeatability of the tests, reusability of the test cases, early quality assurance, oracles availability for validating the testing quality, better risk management, independency from the testers' skills, and automation of the testing process. Moreover, good functionality coverage and increased customer satisfaction during product acceptance are two other identified strengths. As for ET, cost-effectiveness, the nature of the process, and testing quality were the main strength categories recognized. Exploratory testing was stated to be cost-effective because of less time being spent on documentation (i.e., focused documentation for only logs, test notes, and videos after the execution), better resource utilization, rapid feedback, and quick learning of the product. As for the testing quality, better defect detection effectiveness, better regression testing, and more critical bug detection were found to be the major strengths. Because the process of ET is flexible, the skills of testers are better utilized as they can freely explore the defects; and thus, the testers become more responsible, engaged, motivated, and creative, while they are performing the tests. Tables V and IV provide an explanation of the strengths.

What are the weaknesses of ST and ET?: For ST, major weaknesses were found to fall under testing quality, nature of the process, and cost-effectiveness categories. One of the major weaknesses was identified as the dependency of testing quality on the test case design, which depends on the skills, experience, and the domain knowledge of the designer as well as the previously produced documents. Testers, being not free to make decisions even if they see the problem about the test cases, were another weakness attributed to inflexibility of the test process. As for the cost-effectiveness, ST found to be time consuming and costly as it requires designing, documenting, executing, and managing large numbers of test cases, which should also be updated continuously in the software development life cycle as the requirements change. Moreover, the cost increases if test cases require revision and/or redesign in cases of low quality design.

On the other hand, major weaknesses of ET were identified as related to the nature of the process, testing quality, and customer satisfaction categories. The unstructured and ad hoc processes are found to cause difficulties in managing the testing process and risk, in prioritizing and selecting the appropriate tests, and in repeating the tests. Moreover, these also, in turn, create the fear of losing control over testing. As for testing quality, the dependency on the skills, experience, and domain

knowledge of the testers are among the major weaknesses identified. These become more significant especially when the application to be tested is too complex. In addition, ET found to be not suitable for acceptance, performance, and release testing, which in turn lowers the accountability and hence customer satisfaction. Table 49 and Table 50 provide an explanation of the weaknesses.

What are the improvement opportunities for testing processes by addressing some major weaknesses of ST and ET through unifying their processes in a hybrid form?: The second contribution of this study is the identification of the improvement opportunities for the testing process through unification of ST and ET into a resultant HT approach. We defined the HT process considering ISO/IEC 29119, which is an upcoming software testing standard. The industrial evaluation of the proposed HT process was performed through interviews in industry. The practitioners stated that the HT process has merits to resolve some major issues of ST and ET test approaches and invited us to their companies for dynamically validating the HT process. The details of the identification of improvement opportunities through mapping ET and ST strengths and weaknesses to each other are provided in Table 51and Table 52.

Our study contributes to highlight the importance of experience. In order to further understand the merits of HT, we recommend taking the following actions. First, experiments have to be designed and the performance of testers with different experience levels for the different testing approaches has to be compared. Second, experience shall not be treated as a variable stating total experience in years. Instead, experience should be broken down in different kinds of experiences (e.g., programming, testing, and methodologies) relevant to testing to understand its impact on ET and HT processes. Furthermore, we plan to evaluate the hybrid testing process in further trials through action research.

5.3.7.2 Implications for research and practice

We discuss the implications for research and practice the findings from two perspectives, practitioners and researchers.

- Practitioners: Given the analysis of strengths and weaknesses of ST and ET (P1 and P2), a clear need has been established for hybrid processes. This leads to the proposition that practitioners can benefit from using a hybrid development process, hence, utilizing the strengths of both types of processes and addressing the weaknesses. The hybrid process presented in this paper is flexible baseline (indicated by different paths one can take through the process) of an HT process. The process has been co designed with very experienced practitioners knowing both, ET and ST. This study makes their experience, as well as the experience reported in literature, accessible to other practitioners. Practitioners are now in need to adopt and refine the process in practice, as this is the prerequisite to extend and mature it. In particular, empirical evidence provided on the potential and usefulness of a hybrid process could speed up the technology transfer of HT processes. In particular, we found that there is an increasing trend of publications related to ST and ET studies discussing strengths and weaknesses, indicating that with evidence, the interest in adoption and evaluations increases.

- Researchers: We presented an approach that uses systematic review and practitioner input to design a new solution (HT process), the approach being based on the technology transfer model by Gorschek et al. [18]. Researchers might find the approach valuable in designing solutions combining evidence-based methods (here systematic review) and practitioner input in an exploratory way. The HT process needs further evaluation. Researchers hence should focus on conducting empirical studies with industry practitioners putting the process into action. In particular, researchers should evaluate the variances of the test process (e.g., testing with and without debriefing), how the activities and the flow through the process should differ for different types of testing (e.g., which activity in test execution is emphasized in terms of effort spent and number of executions depending on the type of testing, such as GUI testing versus unit testing), and what the longitudinal effects are of using an HT process. For these future activities, our research laid the foundations to continue such research.

6. Conclusion of the thesis

The collection of data related to different attributes of three elements of software allows us to perform the analysis to understand the impact of these attributes on the quality. Therefore first we selected an appropriate measure that should be suitable indicator of quality in term of defect density (see Section 2) and then asked ourselves few research questions regarding the impact of attributes of elements on the quality whose answer can be summarized for the particular sample of projects. In general the thesis answers the main research question that "What is the impact of software elements on the software quality" and then answers the sub questions very explicitly.

Concerning the product attributes, the question we asked ourselves was that "What is the impact of product attributes on the quality?"

The concrete answers we found for this research question is given below.

- There exists a statistically significant medium sized difference of quality between open and closed source projects: the former have a DD that is 4 defects per KLoC lower than the latter.
- Java projects exhibit a significantly lower DD than C projects, 4.1 defects per KLoC on average
- In general the Size appears to be negatively correlated to DD: the larger the project the lower the DD.
- In particular, large projects are 10 times less defective than medium ones.
- Age is not a factor to characterize the quality
- Very small modules on size have significant impact on the projects quality. The more percentage of very small modules resulted in lower quality.
- Defect free modules have significant impact on the projects DD. The more percentage resulted in higher project quality.
- The attribute module dependencies have no significant impact on the projects DD.
- In small projects we found LCOM as effective indicator for the quality
- In the medium category of project we found WMC, CBO, RFC, CA, CE, NPM, DAM, MOA, IC, and Avg CC as effective indicators of quality.
- In the large category of projects we found WMC, CBO, RFC, CA, NPM, AMC and Avg CC as effective indicators of quality.
- Product Complexity (PrC) as defined in Table 53 has partial impact on the software quality.

Concerning the product attributes, the question we asked ourselves was that "What is the impact of people attributes on the quality?"

The concrete answers we found for this research question is given below.

- Analyst Capability (AC) as defined in Table 29 has no impact on the software quality.
- Programmer Capability (PC) as defined in Table 29 has partial impact on the software quality
- Platform Experience (PE) as defined in Table 30 has significant impact on the software quality.
- Application Experience (AE) as defined in Table 30 has significant impact on the software quality
- Language and Tool Experience as defined in (LTE) Table 30 has significant impact on the software quality

Concerning the process attributes, the question we asked ourselves was that "What is the impact of process attributes on the quality?"

The concrete answers we found for this research question is given below.

- We found statistically not significant difference of quality between the projects developed under CMMI and those that are not developed under CMMI.
- Considering the CMMI levels, the pair (CMMI 1, CMMI 3) is characterized by a statistically significant different quality. CMMI 1 exhibiting lower quality than CMMI 3
- By comparing different software processes with each other we found that Hybrid process exhibits statistically significant higher quality than Waterfall.
- Process Maturity (PM) defined by Software Engineering Institute Capability Maturity Model (SEI-CMM) has partial impact on the software quality.

Concerning exploratory testing:

- ET also has many weaknesses that are not apparent at the time of testing but prompt up in later phases of system life cycle.
- These weaknesses incur increased rework and cost, and hence are considered to be the sources of TD.
- We propose the possible solutions to embark upon these weaknesses that indeed help to reduce the testing technical debt of ET in a form of hybrid process.
- We found that both ST and ET provide strengths and weaknesses and these depend on some particular conditions, which prevents preference of one approach to another.
- The mapping showed that it is possible to address the weaknesses in one process by the strengths of the other in a hybrid form.

- With the input from literature and industry experts a flexible and iterative hybrid process was designed.

This study has performed a statistical analysis on different attributes of process, product and people considering different data set. Overall the results in this thesis indicate the impact of attributes of elements on the software quality. The results show that there are some attributes that have significant impact on the quality where other has partial and no impact on the software quality. The empirical findings in this thesis about the impact of attributes of elements on software quality are useful for both practitioners and researchers to evaluate their projects.

The organizational managers who tend to increase the quality of their software should utilize these findings and select only those attributes that have impact on the software quality. The researchers can use the results in order to device more hypothesis in order to find the reasons that why one particular attribute has lower impact on the software quality and other have higher impact.

Concerning the hybrid process, practitioners can clearly benefit from using a hybrid process given the mapping of advantages and disadvantages of both test approaches and would get exploratory testing advantages without the induction of technical debt.

7. Future work

After a look at the literature we believe that empirical research on process characterization is limited. There is a need of further empirical evidence with precise methodology to give managers a broad perspective in making appropriate decisions when selecting software processes.

We would like to continue our future studies with the same attention considering more numbers of attributes that are not covered in this thesis e.g. testing efforts, quality effort, code churn and code history that should probably have impact on the quality.

Concerning the exploratory testing in future, we highlight the importance to evaluate the HT process first in controlled experiments and in industrial environments.

An experimental setup should focus on comparing ET, ST, as well as HT in relation to testing effectiveness (ability to identify critical defects) and efficiency (time needed for test design and execution).

Industrially focused studies need to focus on practitioners executing the process and learning how the process is tailored based on the context (e.g., different organizational test policies, types of system, and so forth). Earlier, we mentioned two types of tailoring, namely process structure (activities to be executed) and process flow (order of activities and relative effort spent on them).

8. References

[1] N. E. Fenton, *Software Metrics: A Rigorous and Practical Approach*. International Thomson Computer Press, 1996.

[2] Dwayne Phillips, "PEOPLE, PROCESS, AND PRODUCT," *http://dwaynephillips.net/CutterPapers/ppp/ppp.htm*, 2013. .

[3] R. S. Pressman, *Software Engineering: A Practitioner's Approach*. USA: McGraw-Hill, 2005.

[4] Yves Le Traon and Lionel Briand, "Empirical Software Engineering Quantitative methods for measuring, assessing, predicting, controlling, and managing software development," *http://www.irisa.fr/triskell/perso_pro/yletraon/cours/CoursEmpirirical%20Software%20Engineering/IntroTypeExperimentsYvesCours2.pdf*, 2013. .

[5] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, "The PROMISE Repository of empirical software engineering data," *West Virginia University, Department of Computer Science*. .

[6] J. W. Creswell, *Research design – Qualitative, quantitative and mixed method approaches*, Second edition. United Kingdom/India: Sage Publications, 2003.

[7] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in software engineering: an introduction*. Norwell, Massachusetts. USA: Kluwer Academic Publishers, 2000.
[8] Y. Zhu and D. Faller, "Defect density assessment in an evolutionary product development environment: a case study in medical imaging," *Softw. IEEE*, vol. PP, no. 99, pp. 1–1, 2012.
[9] L. Westfall, "Defect Density," *http://www.westfallteam.com/Papers/defect_density.pdf*, Mar-2013. .
[10] S. M. A. Shah, M. Morisio, and M. Torchiano, "An Overview of Software Defect Density: A Scoping Study," *Softw. Eng. Conf. APSEC 2012 19th Asia-Pac.*, vol. 1, pp. 406–415, 4.
[11] P. Mohagheghi, R. Conradi, O. M. Killi, and H. Schwarz, "An empirical study of software reuse vs. defect-density and stability," in *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, 2004, pp. 282–291.
[12] C. J. Kim, S.-M. Kim, and K.-W. Song, "Measurement of Level of Quality Control Activities in Software Development [Quality Control Scorecards]," *Converg. Hybrid Inf. Technol. 2008 ICHIT 08 Int. Conf. On*, pp. 763–770, 28.
[13] S. M. A. Shah, M. Morisio, and M. Torchiano, "The impact of process maturity on defect density," in *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, Lund, Sweden, 2012, pp. 315–318.
[14] V. Y. Shen, Tze-jie Yu, S. M. Thebaut, and L. R. Paulsen, "Identifying Error-Prone Software—An Empirical Study," *Softw. Eng. IEEE Trans. On*, vol. SE-11, no. 4, pp. 317–324, 1985.
[15] V. R. Basili and B. T. Perricone, "Software errors and complexity: an empirical investigation0," *Commun ACM*, vol. 27, no. 1, pp. 42–52, 1984.
[16] C. Withrow, "Error density and size in Ada software," *Softw. IEEE*, vol. 7, no. 1, pp. 26–30, 1990.
[17] R. D. Banker and C. F. Kemerer, "Scale Economies in New Software Development," *Softw. Eng. IEEE Trans. On*, vol. 15, no. 10, pp. 1199–1205, 1989.
[18] N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *Softw. Eng. IEEE Trans. On*, vol. 26, no. 8, pp. 797–814, 2000.
[19] C. Andersson and P. Runeson, "A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems," *Softw. Eng. IEEE Trans. On*, vol. 33, no. 5, pp. 273–286, 2007.
[20] T. J. McCabe, "A Complexity Measure," *Softw. Eng. IEEE Trans. On*, vol. SE-2, no. 4, pp. 308–320, 1976.
[21] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *Softw. Eng. IEEE Trans. On*, vol. 20, no. 6, pp. 476–493, 1994.
[22] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *Softw. Eng. IEEE Trans. On*, vol. 28, no. 1, pp. 4–17, 2002.
[23] B. Henderson-Sellers, *Object-Oriented Metrics, measures of Complexity*. Prentice Hall, 1996.
[24] R. Martin, "OO Design Quality Metrics - An Analysis of Dependencies," presented at the Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, OOPSLA'94.
[25] N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *Softw. Eng. IEEE Trans. On*, vol. 26, no. 8, pp. 797–814, 2000.
[26] A. G. Koru, Dongsong Zhang, and Hongfang Liu, "Modeling the Effect of Size on Defect Proneness for Open-Source Software," in *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, 2007, p. 10.
[27] N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *Softw. Eng. IEEE Trans. On*, vol. 26, no. 8, pp. 797–814, 2000.
[28] F. Akiyama, "An Example of Software System Debugging," *Inf. Process.*, vol. 71, pp. 353–379, 1971.
[29] S. McConnell, *Code Complete*, 2nd ed. Washington: Microsoft Press, 2004.
[30] S. Chulani, "Constructive Quality Modeling for Defect Density Prediction: COQUALMO." IBM Research, Center for Software Engineering.
[31] G. Koru, H. Liu, D. Zhang, and K. El Emam, "Testing the theory of relative defect proneness for closed-source software," *Empir. Softw. Eng.*, vol. 15, no. 6, pp. 577–598–598, Dec. 2010.
[32] A. Koru, K. E. Emam, D. Zhang, H. Liu, and D. Mathew, "Theory of relative defect proneness," *Empir. Softw Engg*, vol. 13, no. 5, pp. 473–498, 2008.

[33] S. Raghunathan, A. Prasad, B. K. Mishra, and Hsihui Chang, "Open source versus closed source: software quality in monopoly and competitive markets," *Syst. Man Cybern. Part Syst. Hum. IEEE Trans. On*, vol. 35, no. 6, pp. 903– 918, 2005.
[34] G. Phipps, "Comparing observed bug and productivity rates for Java and C++," *Softw Pr. Exper*, vol. 29, no. 4, pp. 345–358, 1999.
[35] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *Softw. Eng. IEEE Trans. On*, vol. 26, no. 7, pp. 653–661, 2000.
[36] N. Zvegintzov, "Software should live longer," *Softw. IEEE*, vol. 15, no. 4, pp. 19, 21, 1998.
[37] D. Cotroneo, R. Natella, and R. Pietrantuono, "Is software aging related to software metrics?," in *Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second International Workshop on*, 2010, pp. 1–6.
[38] K.-H. Moller and D. J. Paulish, "An empirical investigation of software fault distribution," in *Software Metrics Symposium, 1993. Proceedings., First International*, 1993, pp. 82–90.
[39] L. Hatton, "Reexamining the fault density component size connection," *Softw. IEEE*, vol. 14, no. 2, pp. 89–97, 1997.
[40] J. Rosenberg, "Some misconceptions about lines of code," in *Software Metrics Symposium, 1997. Proceedings., Fourth International*, 1997, pp. 137–142.
[41] K. El Emam, S. Benlarbi, N. Goel, W. Melo, H. Lounis, and S. N. Rai, "The optimal class size for object-oriented software," *Softw. Eng. IEEE Trans. On*, vol. 28, no. 5, pp. 494–509, 2002.
[42] A. G. Koru, Dongsong Zhang, K. El Emam, and Hongfang Liu, "An Investigation into the Functional Form of the Size-Defect Relationship for Software Modules," *Softw. Eng. IEEE Trans. On*, vol. 35, no. 2, pp. 293–304, 2009.
[43] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial use of metrics for object-oriented software: an exploratory analysis," *Softw. Eng. IEEE Trans. On*, vol. 24, no. 8, pp. 629–639, 1998.
[44] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *Softw. Eng. IEEE Trans. On*, vol. 22, no. 10, pp. 751–761, 1996.
[45] R. Subramanyam and M. S. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects," *Softw. Eng. IEEE Trans. On*, vol. 29, no. 4, pp. 297–310, 2003.
[46] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *Softw. Eng. IEEE Trans. On*, vol. 31, no. 10, pp. 897–910, 2005.
[47] M. English, C. Exton, I. Rigon, and B. Cleary, "Fault detection and prediction in an open-source software project," in *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, Vancouver, British Columbia, Canada, 2009, pp. 1–11.
[48] Ping Yu, T. Systa, and H. Muller, "Predicting fault-proneness using OO metrics. An industrial case study," in *Software Maintenance and Reengineering, 2002. Proceedings. Sixth European Conference on*, 2002, pp. 99–107.
[49] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes," *Softw. Eng. IEEE Trans. On*, vol. 33, no. 6, pp. 402–419, 2007.
[50] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceedings of the 28th international conference on Software engineering*, Shanghai, China, 2006, pp. 452–461.
[51] H. Arksey and L. O'Malle, "Scoping studies: towards a methodological framework," *Int. J. Soc. Res. Methodol.*, vol. 8, no. 1, pp. 19–32, 2005.
[52] W. F. Tichy, "Should computer scientists experiment more?," *Computer*, vol. 31, no. 5, pp. 32–40, 1998.
[53] Mei-Huei Tang, Ming-Hung Kao, and Mei-Hwa Chen, "An empirical study on object-oriented metrics," in *Software Metrics Symposium, 1999. Proceedings. Sixth International*, 1999, pp. 242–249.
[54] "COCOMO® II Cost Driver and Scale Driver Help," *http://sunset.usc.edu/research/COCOMOII/expert_cocomo/drivers.html*, Feb-2013. .
[55] W. S. Humphrey, "The Quality Attitude," Mar-2004. .

[56] S. T. Acuna, M. Gomez, and N. Juristo, "How do personality, team processes and task characteristics relate to job satisfaction and software quality?," *Inf Softw Technol*, vol. 51, no. 3, pp. 627–639, 2009.
[57] O. Hazzan and I. Hadar, "Controversy Corner: Why and how can human-related measures support software development processes?," *J Syst Softw*, vol. 81, no. 7, pp. 1248–1252, 2008.
[58] N. Gorla and S.-C. Lin, "Determinants of software quality: A survey of information systems project managers," *Inf Softw Technol*, vol. 52, no. 6, pp. 602–610, 2010.
[59] M. S. Krishnan, C. H. Kriebel, S. Kekre, and T. Mukhopadhyay, "An Empirical Analysis of Productivity and Quality in Software Products," *Manage Sci*, vol. 46, no. 6, pp. 745–759, 2000.
[60] K. Shendil and N. H. Madhavji, "Personal `progress functions' in the software process," *Softw. Process Workshop 1994 Proc. Ninth Int.*, pp. 117–121, 5.
[61] I. Hooks and K. Farry, "Customer-centered products: Creating successful products through smart requirements management," American Management Association, New York, 2001.
[62] W. S. Humphrey, *Managing the Software Process*, Addison-Wesley Professional. 1989.
[63] D. M. Ahern, A. Clouse, and R. Turner, *CMMI Distilled: A Practical Introduction to Integrated Process Improvement*, 3rd ed. Addison-Wesley, 2008.
[64] W. W. Royce, "Managing the development of large software systems: concepts and techniques," in *Proceedings of the 9th international conference on Software Engineering*, Monterey, California, United States, 1987, pp. 328–338.
[65] W. S. Humphrey, *Introduction to the Personal Software Process*. Addison-Wesley, 1997.
[66] W. S. Humphrey, *Introduction to the Team Software Process*. Addison-Wesley, 1999.
[67] P. Kruchten, *The Rational Unified Process: An Introduction*, 2nd ed. Boston MA USA: Addison-Wesley, 2000.
[68] M. Turner, *Microsoft® solutions framework essentials: building successful technology solutions*. Redmond WA USA: Microsoft Press, 2006.
[69] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2003.
[70] P. D. Ronald and B. Finkbine, "Metrics and Models in Software Quality Engineering," *SIGSOFT Softw Eng Notes*, vol. 21, no. 1, p. 89, 1999.
[71] M. M. Hinkle, "Software Quality, Metrics, Process Improvement, and CMMI: An Interview with Dick Fairley," *IT Prof.*, vol. 9, no. 3, pp. 47–51, 2007.
[72] I. Sommerville, *Software Engineering*, 8th ed. Pearson Education, 2007.
[73] P. Miller, "An SEI Process Improvement Path to Software Quality," in *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*, 2007, pp. 12–20.
[74] P. Monteiro, R. J. Machado, and R. Kazman, "Inception of Software Validation and Verification Practices within CMMI Level 2," in *Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on*, 2009, pp. 536–541.
[75] Meng Li, He Xiaoyuan, and A. Sontakke, "Defect Prevention: A General Framework and Its Application," in *Quality Software, 2006. QSIC 2006. Sixth International Conference on*, 2006, pp. 281–286.
[76] C. Jones, "The Pragmatics of Software Process Improvements," *Softw. Eng. Tech. Counc. Newsl. Tech. Counc. Softw. Eng IEEE Comput. Soc.*, vol. 14, no. 2, 1996.
[77] P. Abrahamsson, J. Warsta, M. T. Siponen, and J. Ronkainen, "New directions on agile methods: a comparative analysis," in *Proceedings of the 25th International Conference on Software Engineering*, Portland, Oregon, 2003, pp. 244–254.
[78] P. J. Agerfalk and B. Fitzgerald, "Flexible and distributed software processes: old petunias in new bowls?: Introduction," *Commun. ACM*, vol. 49, pp. 26–34, 2006.
[79] B. Boehm, "Get ready for agile methods, with care," *Computer*, vol. 35, no. 1, pp. 64–69, 2002.
[80] D. H. Kitson and S. M. Masters, "An analysis of SEI software process assessment results: 1987-1991," in *Software Engineering, 1993. Proceedings., 15th International Conference on*, 1993, pp. 68–77.
[81] M. C. Paulk, "Extreme programming from a CMM perspective," *Softw. IEEE*, vol. 18, no. 6, pp. 19–26, 2001.
[82] B. O. Sussy, J. A. Calvo-Manzano, C. Gonzalo, and S. F. Tomas, "Teaching Team Software Process in Graduate Courses to Increase Productivity and Improve Software Quality," in

Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International, 2008, pp. 440–446.

[83] P. Abrahamsson and J. Koskela, "Extreme programming: a survey of empirical data from a controlled case study," in *Empirical Software Engineering, 2004. ISESE '04. Proceedings. 2004 International Symposium on*, 2004, pp. 73–82.

[84] J. C. Sanchez, L. Williams, and E. M. Maximilien, "On the Sustained Use of a Test-Driven Development Practice at IBM," in *AGILE 2007*, 2007, pp. 5–14.

[85] N. Ramasubbu and R. K. Balan, "The impact of process choice in high maturity environments: An empirical analysis," in *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, 2009, pp. 529–539.

[86] K. K. Mohan, A. K. Verma, A. Srividya, G. V. Rao, and R. K. Gedela, "Early Quantitative Software Reliability Prediction Using Petri-nets," in *Industrial and Information Systems, 2008. ICIIS 2008. IEEE Region 10 and the Third international Conference on*, 2008, pp. 1–6.

[87] T. Bhat and N. Nagappan, "Evaluating the efficacy of test-driven development: industrial case studies," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, Rio de Janeiro, Brazil, 2006, pp. 356–363.

[88] S. M. Mitchell and C. B. Seaman, "A comparison of software cost, duration, and quality for waterfall vs. iterative and incremental development: A systematic review," in *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, 2009, pp. 511–515.

[89] C. Jones, J. Subramanyam, and O. Bonsignour, *The Economics of Software Quality*, Addison-Wesley Professional. 2011.

[90] G. H. Subramanian, J. J. Jiang, and G. Klein, "Software quality and IS project performance improvements from software development process maturity and IS implementation strategies," *J Syst Softw*, vol. 80, no. 4, pp. 616–627, 2007.

[91] R. Tufail and A. A. Malik, "A Case Study Analyzing the Impact of Software Process Adoption on Software Quality," *Front. Inf. Technol. FIT 2012 10th Int. Conf. On*, pp. 254–256, 17.

[92] J. Li, N. B. Moe, and T. Dybå, "Transition from a plan-driven process to Scrum: a longitudinal case study on software quality," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, Bolzano-Bozen, Italy, 2010, pp. 1–10.

[93] H. A. Rubin, "Software process maturity: measuring its impact on productivity and quality," *Softw. Eng. 1993 Proc. 15th Int. Conf. On*, pp. 468–476, 17.

[94] R. Sison, "Investigating the Effect of Pair Programming and Software Size on Software Quality and Programmer Productivity," *Softw. Eng. Conf. 2009 APSEC 09 Asia-Pac.*, pp. 187–193, 1.

[95] Y. Rafique and V. B. Mišić, "The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis," *Softw. Eng. IEEE Trans. On*, vol. 39, no. 6, pp. 835–856, Jun. 2013.

[96] C. Jones, *Patterns of Software Systems Failure and Success*. International Thompson Computer Press, 1995.

[97] S. Eldh, H. Hansson, Sasikumar Punnekkat, A. Pettersson, and D. Sundmark, "A Framework for Comparing Efficiency, Effectiveness and Applicability of Software Testing Techniques," in *Testing: Academic and Industrial Conference - Practice And Research Techniques, 2006. TAIC PART 2006. Proceedings*, 2006, pp. 159–170.

[98] NIST-Final Report, "The Economic Impacts of Inadequate Infrastructure for Software Testing," Table 8-1, National Institute of Standards and Technology, 2002.

[99] F. Ricca, M. Torchiano, M. D. Penta, M. Ceccato, and P. Tonella, "Using acceptance tests as a support for clarifying requirements: A series of experiments," *Inf. Softw. Technol.*, vol. 51, no. 2, pp. 270–283, 2009.

[100] C. Andersson and P. Runeson, "Verification and validation in industry - a qualitative survey on the state of practice," in *Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n*, 2002, pp. 37– 47.

[101] J. Itkonen, "Do test cases really matter? an experiment comparing test case based and exploratory testing," Ph.D thesis, Helsinki University of Technology, Finland (2008).

[102] J. Bach, *"Exploratory Testing", in The Testing Practitioner*, Second ed., E. van Veenendaal Ed. Den Bosch: UTN Publishers, 2004.

[103] J. Itkonen, M. V. Mantyla, and C. Lassenius, "How do testers do it? An exploratory study on manual testing practices," in *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, 2009, pp. 494–497.
[104] J. Vaga and S. Amland, "Managing high-speed web testing," in *Software quality and software testing in internet times*, Springer-Verlag New York, Inc., 2002, pp. 23–30.
[105] T. Klinger, P. Tarr, P. Wagstrom, and C. Williams, "An enterprise perspective on technical debt," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, Waikiki, Honolulu, HI, USA, 2011, pp. 35–38.
[106] F. Buschmann, "To Pay or Not to Pay Technical Debt," *IEEE Softw*, vol. 28, no. 6, pp. 29–31, 2011.
[107] "IEEE Guide for Software Verification and Validation Plans." 1994.
[108] M. J. Arafeen and Hyunsook Do, "Adaptive Regression Testing Strategy: An Empirical Study," in *Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on*, 2011, pp. 130–139.
[109] A. Memon, I. Banerjee, and A. Nagarajan, "What test oracle should I use for effective GUI testing?," in *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, 2003, pp. 164– 173.
[110] S. . Sim, S. Ratanotayanon, O. Aiyelokun, and E. Morris, "An Initial Study to Develop an Empirical Test for Software Engineering Expertise," Institute for Software Research, University of California, Irvine, CA, USA, UCI-ISR-06-6, 2006.
[111] P.-L. Poon, T. H. Tse, S.-F. Tang, and F.-C. Kuo, "Contributions of tester experience and a checklist guideline to the identification of categories and choices for software testing," *Softw. Qual. Control*, vol. 19, no. 1, pp. 141–163, 2011.
[112] B. Muranko and R. Drechsler, "Technical Documentation of Software and Hardware in Embedded Systems," in *Very Large Scale Integration, 2006 IFIP International Conference on*, 2006, pp. 261–266.
[113] C. Pecheur, F. Raimondi, and G. Brat, "A formal analysis of requirements-based testing," in *Proceedings of the eighteenth international symposium on Software testing and analysis*, Chicago, IL, USA, 2009, pp. 47–56.
[114] G. Scanniello, F. Fasano, A. Lucia, and G. Tortora, "Does software error/defect identification matter in the Italian industry?," *IET Softw.*, vol. 7(2), pp. 76–84, 2013.
[115] J. Itkonen, M. V. Mantyla, and C. Lassenius, "Defect Detection Efficiency: Test Case Based vs. Exploratory Testing," in *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, 2007, pp. 61–70.
[116] A. Bertolino, "Software Testing Research: Achievements, Challenges, Dreams," in *Future of Software Engineering, 2007. FOSE '07*, 2007, pp. 85–103.
[117] L. Copeland, *A Practitioner's Guide to Software Test Design*. Boston: Artech House Publishers, 2004.
[118] N. Juzgado, A. Moreno, and S. Vegas, "Reviewing 25 years of testing technique experiments.," presented at the Empirical Software Engineering, 2004, vol. 9(1–2), pp. 7–44.
[119] S. Vegas, N. Juristo, and V. R. Basili, "Identifying Relevant Information for Testing Technique Selection: An Instantiated Characterization Scheme. .," *Springer Heidelb.*, Germany 2003.
[120] C. Agruss and B. Johnson, "Ad hoc software testing, a perspective on exploration and improvisation," Florida Institute of Technology, USA, Techical report, 2000.
[121] L. Shoaib, A. Nadeem, and A. Akbar, "An empirical evaluation of the influence of human personality on exploratory software testing," in *Multitopic Conference, 2009. INMIC 2009. IEEE 13th International*, 2009, pp. 1–6.
[122] J. Bach, *Exploratory testing*. In Veenendal EV (eds.), 2005.
[123] Thomson A., "How to choose between exploratory and scripted testing," .
[124] T. Murnane, K. Reed K, and R. Hall R, "On the learnability of two representations of equivalence partitioning and boundary value analysis.," presented at the Australian Software Engineering Conference (ASWEC 2007), 2007, vol. 274–283.
[125] M. S. C. . Sharma, "Automatic generation of test suites fromdecision table – theory and implementation.," pp. 459–464.
[126] P. Ammann and J. Offutt, *Introduction to Software Testing. :* Cambridge University Press, 2008.

[127] A. Dupuy and N. Leveson, “An empirical evaluation of the mc/dc coverage criterion on the hete-2 satellite software.,” 2000.
[128] L. C. Briand, Y. Labiche, and Q. Lin, “Improving the coverage criteria of uml state machines using data flow analysis.,” 2010, pp. 177–207.
[129] B. Kitchenham and S. Charters, “Guidelines for performing Systematic Literature Reviews in Software Engineering,” Software Engineering Group, School of Computer Science and Mathematics, Keele University, Keele,, UK, Vol 2.3 EBSE Technical Report, EBSE-2007-01, 2007.
[130] T. Gorschek, C. Wohlin, and S. Larsson, “, C. Wohlin, P. Carre, and S. Larsson, ‘A Model for Technology Transfer in Practice,’ Software, IEEE, vol. 23, no. 6, pp. 88-95, 2006.,” *IEEE Softw.*, vol. 23, no. 6, pp. 88–95, 2006.
[131] B. Henderson-Sellers, C. Gonzalo, and J. Ralyte, “Comparison of method chunks and method fragments for situational method engineering.,” presented at the Australian Conference on Software Engineering (ASWEC 2008), 2008, pp. 479–488.
[132] K. Petersen, “Measuring and predicting software productivity: a systematic map and review.,” *Inf. Softw. Technol.*, pp. 317–343, 2011.
[133] Zotero., “A reference management tool, project of Roy Rosenzweig Center for History and New Media, funded by the Andrew W. Mellon Foundation, the Institute of Museum and Library Services, and the Alfred P. Sloan Foundation.” .
[134] G. W. Noblit and R. D. Hare, *Meta-Ethnography: Synthesizing Qualitative Studies*. London: Sage, 1998.
[135] Dixon-Woods M, Agarwal S, Jones D, Young B, and Sutton A, “Synthesising qualitative and quantitative evidence: a review of possible methods.,” *J. Health Serv. Res. Policy*, no. 10(1), pp. 45–53B, 2005.
[136] Barnett-Page E and Thomas J, “Methods for the synthesis of qualitative research: a critical review.,” *BMC Med. Res. Methodol.*, vol. 9(1), no. 59, 2009.
[137] Britten N,, Campbell R,, Pope C,, Donovan J,, Morgan M,, and Pill R., “Using meta ethnography to synthesise qualitative research: a worked example.,” *J. Health Serv. Res. Policy*, no. 7(4), pp. 209–215, 2002.
[138] Seidel JV., “Qualitative data analysis.,” *Qualis Res. Colo. Springs Colo.*, 1998.
[139] Given LM., *The Sage Encyclopedia of Qualitative Research Methods*. SAGE: Los Angeles, 2008.
[140] J. Bach, “Session-based test management,” *Software Testing Quality Engineering magazine*, vol. 2, no. 6, 2000.
[141] Tahat L H, Bader A, Vaysburg B, and Korel B, “Requirement-based automated black-box test generation.,” presented at the 25th International Computer Software and Applications Conference (COMPSAC 2001), 2001, pp. 489–495.

9. Appendix

Table 53 Product complexity criterion

	Control Operations	**Computational Operations**	**Device-dependent Operations**	**Data Management Operations**	**User Interface Management Operations**
Very Low	Straight-line code with a few non-nested structured programming operators: DOs, CASEs, IFTHENELSEs. Simple module	Evaluation of simple expressions: e.g.,*A=B+C*(D-E)*	Simple read, write statements with simple formats.	Simple arrays in main memory. Simple COTS-DB queries, updates.	Simple input forms, report generators.

	composition via procedure calls or simple scripts.				
Low	Straightforward nesting of structured programming operators. Mostly simple predicates	Evaluation of moderate-level expressions: e.g., *D=SQRT(B**2 -4.*A*C)*	No cognizance needed of particular processor or I/O device characteristic s. I/O done at GET/PUT level.	Single file sub setting with no data structure changes, no edits, no intermediate files. Moderately complex COTS-DB queries, updates.	Use of simple graphic user interface (GUI) builders.
Medium	Mostly simple nesting. Some intermodule control. Decision tables. Simple callbacks or message passing, including middleware-supported distributed processing	Use of standard math and statistical routines. Basic matrix/vector operations.	I/O processing includes device selection, status checking and error processing.	Multi-file input and single file output. Simple structural changes, simple edits. Complex COTS-DB queries, updates.	Simple use of widget set.
High	Highly nested structured programming operators with many compound predicates. ueue and stack control. Homogeneous, distributed processing. Single processor soft real-time control.	Basic numerical analysis: multivariate interpolation, ordinary differential equations. Basic truncation, roundoff concerns.	Operations at physical I/O level (physical storage address translations; seeks, reads, etc.). Optimized I/O overlap.	Simple triggers activated by data stream contents. Complex data restructuring.	Widget set development and extension. Simple voice I/O, multimedia.
Very High	Reentrant and recursive coding. Fixed-priority interrupt handling. Task synchronization , complex callbacks, heterogeneous distributed processing.	Difficult but structured numerical analysis: near-singular matrix equations, partial differential equations. Simple parallelization.	Routines for interrupt diagnosis, servicing, masking. Communicati on line handling. Performance-intensive embedded systems.	Distributed database coordination. Complex triggers. Search optimization.	Moderately complex 2D/3D, dynamic graphics, multimedia.

	Single-processor hard real-time control.				
Extra High	Multiple resource scheduling with dynamically changing priorities. Microcode-level control. Distributed hard real-time control.	Difficult and unstructured numerical analysis: highly accurate analysis of noisy, stochastic data. Complex parallelization.	Device timing-dependent coding, micro-programmed operations. Performance-critical embedded systems.	Highly coupled, dynamic relational and object structures. Natural language data management.	Complex multimedia, virtual reality.