Software Qual J (2018) 26:779–807
https://doi.org/10.1007/s11219-016-9334-6

Challenges of software process and product quality improvement: catalyzing defect root-cause investigation by process enactment data analysis

Mehmet Söylemez[1] · Ayca Tarhan[2]

Published online: 22 August 2016

Abstract It is claimed by software quality management that the quality of a software product is highly influenced by the quality of the software process followed to develop it. Since measurement of the software process is a challenging task, it is frequently the defects in the software product that are used to measure development quality. By extracting semantic information from defect records, practitioners can investigate and address root causes of software defects to improve development process and product quality. Investigating root causes requires effort for a detailed analysis into the components of the development process that originated the software defects, and is therefore encouraged only at higher maturity levels by most known process improvement models such as Capability Maturity Model Integration (CMMI). This practice, however, postpones the benefits that root-cause analysis would bring in gaining process awareness to improve the software development process and product quality in emergent organizations or organizations residing at lower maturity levels (MLs). In this article, we present a method for and results from applying root-cause analysis for software defects recorded in a software-intensive project of a CMMI ML3 certified institute. The suggested method combines process enactment data collection and analysis with Orthogonal Defect Classification which is a known technique in defect root-cause analysis. Prior to and after implementing the method in the study, defect attributes were analyzed and compared in order to understand any improvements in development performance and product quality. The results of the comparison indicate that the suggested method was efficient in the effort it required and effective in improving development performance and product quality. Defect triggers have

✉ Ayca Tarhan
atarhan@cs.hacettepe.edu.tr

Mehmet Söylemez
mehmet.soylemez@tubitak.gov.tr

1 TÜBİTAK - BİLGEM/Software Technologies Research Institute, 06100 Ankara, Turkey

2 Computer Engineering Department, Hacettepe University, Beytepe Kampusu, 06800 Ankara, Turkey

become more active in identifying software defects in the earlier phases of software development, and the cost of quality due to software defects has decreased in consequence.

Keywords Causal analysis · Software defect · Orthogonal defect classification · Process enactment · Product quality · Software process · Process improvement

1 Introduction

The main challenges in software industry include the delivery of the developed product within the specified time and budget, and presenting the final product to the customer with a high level of satisfaction (Jones 2008). Meeting these challenges is highly dependent on the quality of the developed product and the software development process itself (Humphrey 1989). Organizations that have a well-defined software development process and that apply systematic quality assurance activities are assumed to deliver high-quality products on time. Identifying the points where the defects occur during the development process and taking corrective and preventive actions reduce costs and result in high-quality and reliable products (ISO 2015).

The evaluation of a software development process to identify the weak points that can lead to defects, on the other hand, is not a straightforward activity. A systematic review of measurement in software engineering (Gómez et al. 2006) reports that software process is the least measured entity and comprises only 21 % of the published studies. In addition to the scarcity of the well-defined methods, this is mostly due to the diverse requirements of knowledge on, for example, process management, measurement, and statistics that need to be considered when empirically evaluating a software development process (Carleton and Florac 1999). Therefore, it is frequently the defects in the software product that are used to measure the quality and understand the process problems that lead to these defects.

Defect root-cause analysis aims to identify weak points in the process relating to the origin of the defects (Andersen and Fagerhaug 2006). This analysis draws attention of the development team to existing problems in the process and develops awareness within the team to prevent the re-occurrence of defects in the software product. Defect records are assumed to hold semantic information related to the causes of defects and are of great value for tracking the defects, taking preventive actions, and improving the process. Causal analysis is also encouraged by the most known process improvement models such as Capability Maturity Model Integration (CMMI) (SEI 2010) typically at higher maturity levels (MLs). In most software projects, the defects are detected in the test and maintenance phases (Boehm 1984; Kan 2002), when the cost of resolving a defect is at maximum. Undertaking a defect analysis in the later phases of the software development lifecycle is difficult since semantic information to support root-cause analysis is frequently not available in the defects records. This situation has two consequences: (i) It is not possible to take actions to timely prevent the defects in the lifecycle, and (ii) the analysis of the defects records through the end of the development process has less value for the project team (in terms of, e.g., time, effort, and cost).

In this research, we investigate whether and how an emerging software institute can benefit from practicing root-cause analysis to improve its development process and product quality. Prior to and after analyzing root causes of defects of critical importance, we intend to understand the existing situations and the improvements in between (if any) in quantitative terms, and use case-study method to structure our research. The institute where the case study is carried out is a part of a research organization and has been certified at ML3 of CMMI. The institute has been developing software-intensive systems in the domain of e-government which connects various Web services from different governmental bodies and/or related agencies. The quality of the systems developed is thus of critical importance for its broad effect on inter-related services and operations that rely on these. A technique used in defect root-cause analysis is *Orthogonal Defect Classification (ODC)* (Chillarege et al. 1992), but the cost of and methods for adopting or supporting it have been rarely reported in the literature (Buglione and Abran 2006; Shenvi 2009). Therefore, we defined and followed a systematic method while carrying out the case study. The suggested method requires collection and analysis of enactment data of software development process and combines the results from this analysis with those from the ODC technique to analyze root causes of software defects.

This article explains the method for and quantitative results from applying root-cause analysis in the institute and the related background. Section 2 provides an overview of related studies with an emphasis on software defects and root-cause analysis, process enactment analysis, and Orthogonal Defect Classification and highlights the contribution of our research in relation to these. Section 3 outlines research design by clarifying the context and design of the study as well as overviewing the method applied in defect causal analysis and process enactment analysis. Section 4 explains step-by-step implementation of the analysis method and provides example artifacts from the implementation. Section 5 summarizes measured benefits and cost of causal analysis for software process and product quality improvement. Section 6 discusses threats to the validity of this research, and Sect. 7 provides overall conclusions.

2 Related work

2.1 Software defects and root-cause analysis (RCA)

A *software defect* is defined by the IEEE as “an incorrect step, process or data definition in a computer program” (IEEE 1990). A defect can be injected into software at any stage of the software development. These defects that enter the software have been classified by Jones as follows (Jones 2008): (i) requirement defects, (ii) design defects, (iii) code defects, and (iv) documentation defects. There are some requirements and challenges faced during defect classification (Chillarege 1996; Wagner 2008). First, the attributes of classification may be ambiguous, non-orthogonal, and unidentifiable. Second, the specified classification system may have too many attributes, which can cause confusion in the root-cause analysis (Freimut 2001). A good classification system should be less time-consuming for the person recording the defect, record the attributes that help improve the content and the process, record attributes in such a way that they will be understood in the same manner by the developers, and provide sufficient information quickly and efficiently during the analysis.

Root-cause analysis (RCA) investigates weak points in a production system to identify opportunities for improvement (ISO 2015). For an effective root-cause analysis on negative outcomes, defects records should be well kept and the defect classification system should have features that assist in accurate problem identification. Otherwise, the analysis of a single defect to understand the underlying reasons might be either very costly or not possible at all (Chillarege et al. 1992). In such a case, development team cannot benefit from timely feedback to take corrective and preventive actions to improve the development process. A useful tool to represent the root causes of the defects is Fishbone diagram (Ishikawa 1986), and it is efficient in grouping the root causes and allowing collective decisions to be made.

CMMI models are collections of best practices that help organizations to improve their processes. CMMI for development (SEI 2010) provides a comprehensive integrated set of guidelines for developing products and services. CMMI consists of process areas where domain-related process activities are explained in terms of goals and best practices. Causal analysis of positive or negative outcomes to improve processes is suggested by CMMI at its ML 5 where related requirements are defined by *Causal Analysis and Resolution (CAR)* process area. CAR requires identifying the causes of outcomes from the performance of processes to increase quality and productivity. Table 1 shows specific goals and practices of the process area. In case that the selected outcomes are defects, CAR recommends analyzing the root causes of these defects and proposing actions to prevent or reduce their re-occurrence.

Buglione and Abran (2006) focus on the question of whether the CAR practice is observed only in organizations at high maturity levels or it can be introduced at lower maturity levels. The authors argue that earlier use of RCA may help organizations achieve higher maturity and capability levels faster, and they discuss and analyze possible strategies for achieving an effective process improvement by the application of measurement-based tools of total quality management. This study is remarkable for the suggestions it provides for improving CMMI architecture of version 1.2. In parallel to one of these suggestions, the version 1.3 (SEI 2010) of the model includes "CAR Elaboration" under the generic practices in order to guide implementation of causal analysis in the knowledge of each process area.

Kalinowski et al. (2012) ask several questions and answer them with some guidance, which is based on evidence from a systematic literature review, on how to implement defect causal analysis (DCA) efficiently. The study claims that DCA requires the systematic collection of defect data and works most efficiently when projects have a defined software development process to provide a framework for interpreting causal analysis findings and understanding how to act to influence project outcomes. However, given DCA's potential benefits and return on investment, the authors suggest that even lower maturity organizations implement it and based on the literature, they provide recommendations for approach to follow, data to collect, defect taxonomies to use, tools to categorize defect causes, and expected implementation costs.

Both of the studies above suggest practices to apply RCA in lower maturity contexts, but they do not report quantitative results from the implementation of their suggestions. In this study, we advocate the use of process enactment analysis in combination with ODC to

Table 1 Specific practices (SP) in CMMI's causal analysis and resolution process area

Specific goal 1—determine causes of selected outcomes
SP1.1—select outcomes for analysis
SP1.2—analyze causes
Specific goal 2—address causes of selected outcomes
SP2.1—implement action proposals
SP2.2—evaluate the effect of implemented actions
SP2.3—record causal analysis data

perform causal analysis, and report quantitative results obtained by consequent improvement of development process and product quality in a software institute.

2.2 Process enactment analysis

Feiler and Humphrey (1993) define *process enactment* as “the execution of a process by a process agent according to a process definition.” Also referred to as *process performance*, process enactment may be according to an implicit process definition. A process may be represented in process models, and enactment of a process model allows the performance of a process to be monitored and guided (Feiler and Humphrey 1993). These definitions imply an initial process definition that must be obeyed during process executions, which is typical in organizations that have already defined their processes (e.g., at CMMI ML 3 or higher).

Synchronous to the study above, Wolf and Rosenblum (1993) propose an event-based model of a software process and a technique to capture and analyze process enactment data collected according to this software process model. In a following study, Grundy et al. (1997) highlight the importance of a process-centered software engineering environment, and analyze past event histories of automation-supported process enactments of developers in order to improve software processes. The last two approaches assume the existence of process-centered modeling and execution environments for event-based process enactment and analysis.

Event-based process discovery and enhancement have become popular in the last ten years after the introduction of *process mining* as a technique that supports business process management (Van der Aalst and Weijters 2004). Process mining is implemented for achieving three aims: (i) discovery of process, (ii) verification of process implementation according to identified process, and (iii) enrichment of process by detecting the differences of process implementations. The technique can be applied for the second aim to analyze process enactments for variations from process models or specifications. The application of process mining to software processes, however, is rare and emerging (Rubin et al. 2007; Poncin et al. 2011; Gürgen et al. 2014; Rubin et al. 2014).

Huo et al. (2006) provide a software process recovery method based on mining project enactment data. The goal of the method is to uncover the actual process used in order to provide input to improve the quality of a defined software process. Simultaneously, Kabbaj et al. (2008) present an approach to process enactment evolution based on formal management of process deviations which are operations that violate process constraints. Once a deviation is detected, a deviation-tolerance model is attached to the process and used to decide whether to accept or to reject the deviation. Aslan et al. (2014) report a case study to investigate the effect of process enactment data on product defectiveness analysis in a small software organization, and the results show that the accuracy of predictions has improved after inclusion of process enactment data in the analysis. In a recent study, Kuhrmann et al. (2014) propose a software tool that supports the transformation of a given formal development process into a format that project tools can work with. The Process Enactment Tool Framework (PET) is an instrument to import processes based on a meta-model and provide exports for a specific project environment. The authors advocate their proposal by the rationale that current application life cycle platforms, such as IBM Jazz or Microsoft Team Foundation Server, are powerful in terms of direct enactment but hardly connect to comprehensive process models.

Most of the studies above rely on automated process enactment environments tailored for specific software processes or process models. The existence or use of such environments, on the other hand, is not yet mature or widespread, and the number of studies that report software process enactment analysis is few. Consequently, in this study we focus on manual capture and analysis of software process enactment data as independent of the maturity level or automation infrastructure of the software organization.

2.3 Orthogonal defect classification (ODC)

Root-cause analysis requires an investigation into process components, which is time-consuming. Therefore, it is not easily and widely practiced in software organizations. *Orthogonal Defect Classification (ODC)* was proposed in the beginning of 1990s to reduce the cost of causal analysis by bridging the statistical defect model and defect root-cause analysis (Chillarege et al. 1992). Typically, a statistical defect model supports quantitative analysis using measures such as defect density and defect detection rates, but since it does not set a relation to the system that originate the defects, it falls short in identifying the root causes. When successfully initiated, the ODC technique eliminates these disadvantages by categorizing defects and reducing the cost of analysis based on predefined attributes. The main purpose of ODC is to extract semantic information from software defects to take actions against their re-occurrence in order to improve the process. In this sense, ODC can be used as a technique to realize the specific goals defined by CAR process area of CMMI.

The defect type, trigger, and origin are important attributes of the ODC technique. The *defect types* that can be used independently from a specific product are (Chillarege et al. 1992): Interface, Function, Build/Package/Merge, Assignment, Documentation, Checking, Algorithm, and Timing/Serialization. These can be used to classify all defects that emerge in various phases of software development life cycle. The *defect origins* are the phases of the software development process in which the defects are created, and can be Requirements, Design, Coding, and Maintenance. Defect type and origin are selected at the stage of defect resolution by the person that repairs the defect. The root causes of defect types lie in the changes made by the developer to repair the defects. A *defect trigger* is an attribute that represents the condition that allows the identification of a software defect and is selected by the person recording the defect. The defect triggers can be categorized into elements such as requirement and design review, code inspection, unit test, functional test, and system test. While the identification of the defect type and origin is crucial throughout the development for an effective root-cause analysis, the selection of the defect trigger provides information about the integrity of the validation process (Chillarege et al. 1992).

Following the classification of the defects using the ODC technique, the distributions of the defects by the defect attributes (i.e., type, trigger, and origin) are obtained providing quantitative results from the semantic data. This eliminates the problems of high cost and slow feedback to the developers observed in root-cause analysis. The results are evaluated by the development team who may then identify the root causes of the software defects and take preventive actions.

In the literature, there exist studies that reveal qualitative and quantitative effects of applying ODC on software process and product quality improvement. We identified 15 such studies that report improvement via root-cause analysis by either using or adapting the ODC technique. Table 2 lists these studies with the attributes of year, reference, title, aim, and states of evidence for qualitative results and the cost of implementation. We observe from the table that only few studies report the cost of initiating or using ODC, or suggest or explain the details of defect causal analysis method that they follow.

In Table 2, majority of the studies (11 out of 15) state quantitative results of process improvement from ODC-based implementations. This finding shows that the ODC

Table 2 List of related studies that use or adapt ODC for software process improvement

No	Reference	Title	Aim of study	Quantitative results on process improvement?	Specify cost for initiating or using ODC?
R1	(Bassin and Santhanam 1997)	Use of software triggers to evaluate software process effectiveness and capture customer usage profiles	Illustrate the use of ODC triggers to capture customer usage in a way directly meaningful to product development; evaluate the effectiveness of product development activities; and to target specific areas of improvement	Yes	No
R2	(Bridge and Miller 1998)	Orthogonal defect classification using defect data to improve software development	Illustrate how ODC can be used to measure development progress with respect to product quality and identify process problems	No	Yes
R3	(Bassin et al. 1998)	Evaluating software development objectively	Illustrate this ODC's value for software management and technical decision support with three case studies	Yes	No
R4	(Butcher et al. 2002)	Improving software testing via ODC: three case studies	Demonstrate the use of ODC to improve software testing by three case studies	Yes	Yes
R5	(Chillarege and Prasad 2002)	Test and development process retrospective—a case study using ODC triggers	Gain an understanding of the test and development process effectiveness via a case study of a product development retrospective analysis by using ODC	Yes	No
R6	(Jalote and Agrawal 2005)	Using defect analysis feedback for improving quality and productivity in iterative software development	Discuss the role of defect analysis as a feedback mechanism to improve the quality and productivity in an iteratively developed software project	Yes	No
R7	(Shenvi 2009)	Defect prevention with orthogonal defect classification	Demonstrate, through a case study, the structured process for defect prevention using some attributes of ODC for defect classification and its related interpretations for causal analysis, action planning and results tracking	Yes	No
R8	(Kumaresh and Baskaran 2010)	Defect analysis and prevention for software process quality improvement	Gain a deeper understanding of the effectiveness of the software process by examining the details of defects detected in five projects by using ODC; and eliminate similar defects due to process improvements and newer methodologies	Yes	No

Table 2 continued

No	Reference	Title	Aim of study	Quantitative results on process improvement?	Specify cost for initiating or using ODC?
R9	(Li et al. 2011)	Analysis of software process effectiveness based on orthogonal defect classification	Improve the software process by analyzing software process effectiveness by ODC attribute measurement method	Yes	No
R10	(Kanniappan and Mishra 2011)	Orthogonal defects classification of observed defects in C-DAC Noida projects	Propose an approach based on ODC for defect analysis of C-DAC Noida projects, which enables project teams to evaluate the effectiveness and completeness of the verification processes and helps in predicting the quality of deliverables	*Source could not be reached*	*Source could not be reached*
R11	(Dubey 2012)	Towards adopting ODC in automation application development projects	Examine and adopt ODC from the perspective of automation application development (AAD) processes; and improve the design process	No	No
R12	(Chillarege 2013)	Using ODC to diagnose an agile enterprise application development Project	Present a case study using ODC as a diagnostic tool to understand the quality and productivity issues in a agile enterprise Web development project	No	No
R13	(Si and Yan 2013)	Research on quality assurance method based on software defect analysis	Define a software metrics framework based on ODC; evaluate effectiveness and efficiency of quality assurance activities by using the framework; determine the direction of the quality assurance improvements; and help the software process integral improvement	Yes	No
R14	(Söylemez and Tarhan 2013)	Using process enactment data analysis to support orthogonal defect classification for software process improvement	Analyze and compare ODC defect trigger and origin attributes of a project before and after identifying and applying improvement actions in order to understand the improvement in development performance	Yes	Yes
R15	(Huang et al. 2015)	The impact of software process consistency on residual defects	Investigate the effect of process consistency regarding to CMMI maturity levels on residual defects based on defect data of 70 software systems developed by 29 Chinese aviation organizations collected from acceptance tests in 5 years	Yes	No

technique is beneficial when adopted successfully. However, the number of studies that report the cost of practice or method for adoption is scarce. Other than our study (R14), two of the studies (R2 and R4) report the cost of implementation, while the other two (R10 and R13) suggest methods for adopting the ODC technique. R15, on the other hand, is the only study that considers process consistency evaluation in defect causal analysis. We should note that each of these studies addresses a specific challenge of applying defect root-cause analysis via ODC implementation. In other words, neither of the studies holistically address the challenges of presenting a repeatable method, reporting quantitative benefits of improvement, and clearly revealing implementation costs. There seems a need for guiding studies that combat these challenges, and this is where our study contributes to the literature and practice.

This study supports the application of Orthogonal Defect Classification by conducting process enactment analysis in order to catalyze defect root-cause investigation for software process and product quality improvement. It also shows quantitative effects of process improvement achieved by the root-cause analysis on software development performance and product quality in a software institute, and reports the experienced costs of implementation. The initial results were reported in (Söylemez and Tarhan 2013) where the effect of causal analysis and process improvement was shown only on development performance, based on the attributes of defect trigger and origin. This article presents the benefits of defect root-cause analysis on both development performance and product quality. In addition, it provides detailed explanation on the steps and the assets of the causal analysis method by giving examples where necessary, and highlights the positive effect of using process enactment data analysis on identifying root causes and suggesting action items for improvement.

3 Research design

3.1 Research context and design

The case study was conducted in a software project (will be called "Project X" hereafter) of a CMMI ML 3 certified institute. For the software-intensive systems developed in the institute, software verification and validation (V&V) activities have been regularly undertaken in parallel to the development, and the defects have been systematically recorded. The Project X has had many versions and over time 5200 defects have been recorded for five modules of the project (A, B, C, D, and E). The defect records are kept in different CASE tools as the result of V&V activities, but they have not yet been subject to causal analysis. Considering multi-stakeholder nature of e-government applications and related Web services that are being developed, practicing root-cause analysis to understand and improve software processes and resulting product quality would be a valuable contribution for the institute.

In a typical software-intensive project of the institute, there are main releases and sub-releases. The content and the date for receiving the main release are determined in advance. New functions are added to the software in the main release. Prior to the main release, a preliminary release is issued on which system tests can be conducted. Once all the tasks planned for the main release are verified and validated, the preliminary release is completed and the main release is installed on a real platform. Then, sub-releases are used to make improvements to the system that has been installed on the real platform. In sub-releases, no new functions are added to the main release, and only improvements are made.

Project X uses an incremental and iterative software development life cycle, thus allowing the developers to add new attributes to the software at each increment and constantly improve the product through iterative cycles. The incremental approach also makes it possible to obtain rapid feedback from the early stages onwards, which would make root-cause analysis even more valuable. Project X is carried out by a team of 22 people including a project leader, a project leader assistant, four team leaders, two training staff, and 14 developers.

In this research, we aimed to investigate whether, how, and with what cost the software institute can benefit from practicing root-cause analysis to improve its development process and product quality, when a well-defined method for causal analysis is followed. We intended to understand the situations prior to and after analyzing root causes of defects of critical importance from Project X and the improvements in between in quantitative terms, and used case-study method to structure our research. Accordingly, we identified three research questions (RQs):

RQ-1 What are product quality and performance of development process with respect to software defects prior to conducting root-cause analysis?

RQ-2 What are product quality and performance of development process with respect to software defects after conducting root-cause analysis and implementing improvement actions?

RQ-3 What is the cost of conducting root-cause analysis by following the analysis method that we defined?

The structure of the research design is shown in Fig. 1. We defined two analysis units regarding the states of prior to and after causal analysis (i.e., pre- and post-process improvement), and adopted a holistic analysis view (Yin 2013) that would allow us to understand the impacts of process improvements on development performance and product quality. While selecting the defects and the modules in the analysis units, we considered the existence of defect data and process enactment data, and the accessibility of process performers. We defined and followed a method to carry out the causal analysis and explain it in the next subsection. The method allowed us to take advantages of process phase decomposition, process modeling, qualitative data collection, and process verification as well as the ODC technique.

In order to respond to RQ-1, the high-priority defects that were recorded in the development and maintenance stages of modules B and C in versions 1 through 6 of the Project X were addressed. In each version, new components were developed in addition to

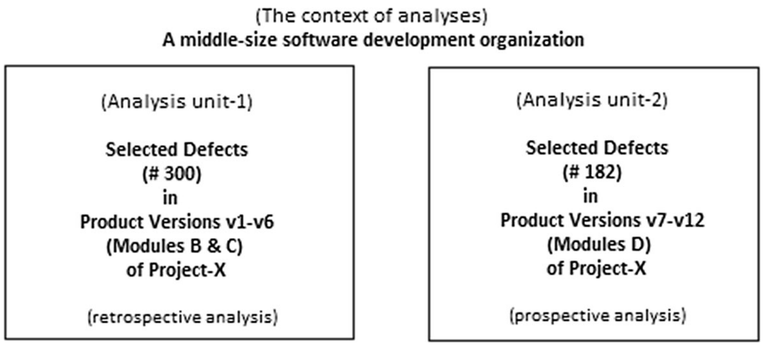


Fig. 1 Structure of research design

the maintenance of existing components. For each version, 50 defects were selected randomly and a total of 300 defects were individually reviewed by the first author who was also a developer of the modules B and C. In this initial analysis (i.e., pre-improvement), the attributes of the ODC technique were used to orthogonally classify the selected defects which were analyzed by type, trigger, and origin. The results of this implementation and process enactment data where the defects occurred were then evaluated together to identify the root causes of the defects and the improvement opportunities. To eliminate the root causes of the defects, suggestions were made for process improvement and taken into practice prior to developing Module D by versions 7 through 12 of the software.

In order to answer RQ-2, the ODC technique was applied to Module D that was developed in the versions 7 through 12 of the Project X. Following a briefing on the attributes of the defects by the first author, the developers entered the attributes of the type, trigger, and origin in the defect life cycle into the defect tracking tool. From Module D that was developed by following the improved process, 182 high-priority defects were selected and analyzed by using the ODC technique. The results of this analysis were compared with those of the initial analysis in order to understand the effect of defect root-cause analysis on development performance and product quality.

To answer RQ-3, the effort spent by process performers and the authors during the execution of the case study was recorded. The cost of causal analysis was calculated by summing the individual efforts.

3.2 Defect causal analysis method

The method for defect causal analysis has been defined to provide a systematic, practical roadmap to analyze the root causes of defects and recommend improvement actions in the software institute. The method combines process enactment data analysis with Orthogonal Defect Classification by executing a number of activities and making use of several assets. The activities of the method comply with the specific practices of the CAR process area of CMMI.

The analysis method is given in Fig. 2 in Event-Driven Process Chain (EPC) notation (Hommes 2004). The first stage in the method is to choose the defect data set for the root-cause analysis. This set may include customer defects, system test defects, development defects, or the combination of these elements for one or more modules or projects. It is

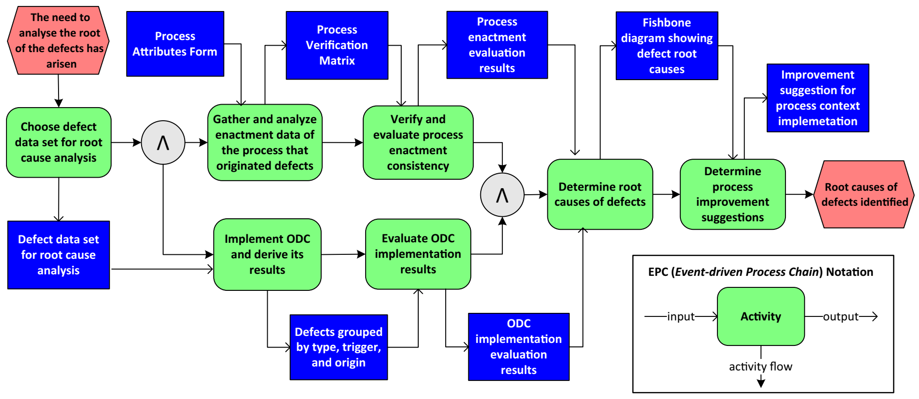

The need to analyse the root of the defects has arisen
Process Attributes Form
Process Verification Matrix
Process enactment evaluation results
Fishbone diagram showing defect root causes
Improvement suggestion for process context implemetation
Choose defect data set for root cause analysis
Λ
Gather and analyze enactment data of the process that originated defects
Verify and evaluate process enactment consistency
Λ
Determine root causes of defects
Determine process improvement suggestions
Root causes of defects identified
Defect data set for root cause analysis
Implement ODC and derive its results
Evaluate ODC implementation results
Defects grouped by type, trigger, and origin
ODC implementation evaluation results
EPC (Event-driven Process Chain) Notation
input
Activity
output
activity flow

Fig. 2 Defect root-cause analysis method

Table 3 Artifacts used in the analysis method

Artifact name	Description
ODC method requirements	For the implementation of ODC, defect attributes (e.g., type and trigger) must have been recorded. These attributes are expected to have orthogonal characteristics
Defects classified according to defect attributes	Defects are classified according to their types and triggers
Results of the ODC application	This artifact holds the results of evaluation regarding the classification of the defects, the extraction of semantic information from the defects, and the possible root causes of the defects
Suggestions for improving the ODC application	Problems that occur during the implementation of ODC are evaluated to make improvement suggestions for the future use of ODC. Since ODC is flexible, the values of defect attributes (e.g., defect types) may differ from one organization to another
Process enactment attributes form (PEAF)	This artifact contains the components of the process model and certain attributes of the environment in which the process is enacted
Process similarity matrix (PSM)	This artifact provides information about the current state of the process enactments to identify the weak points at which the defects might occur
Results of process enactment	This artifact holds the results from identifying what works well and what has failed during process enactments. The results offer a general idea about how the process has worked
Suggestions for improving the process	As a result of the analysis of the process enactment according to the expectations of the process model, suggestions are made to correct the defects and improve the process
Fishbone diagram showing the causes for defects	This diagram shows the root causes of defects that were identified through the analysis of the results from the ODC and process implementations

important that enactment data of the process in which these defects originated are collected and available. The flows upper side and lower side, which succeed the activity of choosing defect data set for the root-cause analysis in the figure, indicate process enactment analysis and the ODC implementation, respectively. Steps to analyze process enactments are described in the next subsection. The results obtained by both the flows are used to identify the root causes of the defects and represent them as a Fishbone diagram. The causes on this diagram are then used to determine the opportunities for software process improvement.

Artifacts (i.e., inputs and outputs) of the analysis method are described in Table 3. All the artifacts were subject to human cognition for analysis in the study by team members and the authors. Some of the artifacts and their analysis are exemplified in Sect. 4 from the activity implementations.

3.3 Process enactment analysis method

Data from process enactments to produce the software product are analyzed for identifying inconsistencies and variations, if any, with respect to expected process attribute values. Process enactments to gather data should be selected in such a way that they produce the part of software for which root causes of detected defects are being investigated. In other words, it is the “process enactments” that produce “defective” or “defect-free” software.

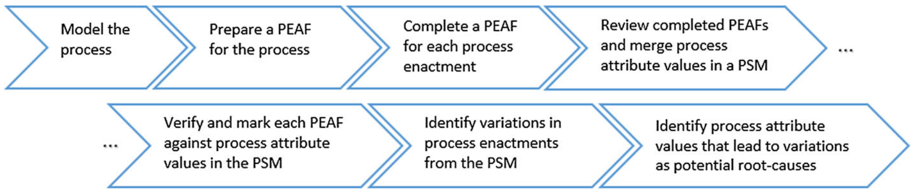

Model the process
Prepare a PEAF for the process
Complete a PEAF for each process enactment
Review completed PEAFs and merge process attribute values in a PSM
...
...
Verify and mark each PEAF against process attribute values in the PSM
Identify variations in process enactments from the PSM
Identify process attribute values that lead to variations as potential root-causes

Fig. 3 Process enactment analysis method

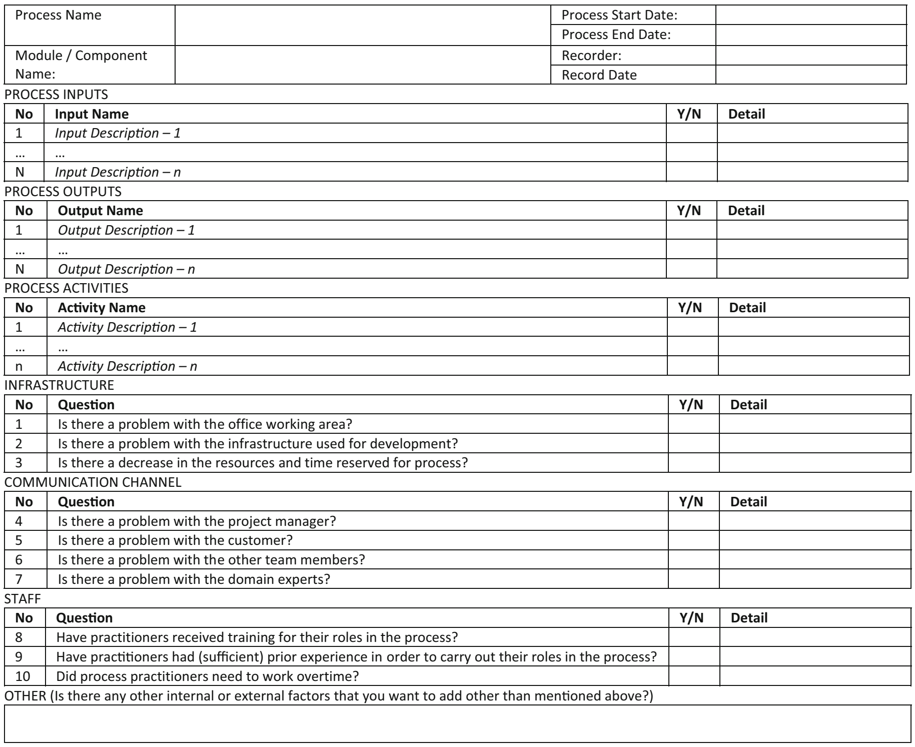
By gathering and analyzing the process enactments' data, we identify problematic enactments and underlying process attribute values that are the likely causes of software defects. Problematic process enactments are the ones with missing process attributes (e.g., lacking of a process input or skipping of a process activity) in execution. Consequently, the missing process attribute values in the enactments constitute *potential* root causes that lead to variations (or nonconformances) in the process execution, which in turn might produce defective software. The steps of process enactment analysis are shown in Fig. 3. We should note that the potential root causes identified by this analysis should be reviewed and discussed with process performers in order to understand whether they are the intended causes (i.e., the enactment is varied for a specific and known reason) or not. The causes that are not intended by process performers are then included in the finalized list of root causes.

The process enactment data are collected by process performers via Process Enactment Attributes Form (PEAF) during the development and then gathered and verified against process attribute values (e.g., the values of inputs, activates, and outputs) via Process Similarity Matrix (PSM) (Tarhan and Demirors 2012).

The main requirement for the analysis of process enactments is the distinctiveness of process components. The input, activity, and output components are clearly modeled by either the process owner or the person who is responsible for root-cause analysis. The suggested notation for modeling is the Event-driven Process Chain (Hommes 2004), but any other notation that supports the modeling of the stated components is appropriate. Then, a PEAF for the process is prepared by the person who is responsible for the analysis, based on the components depicted in the process model. This form is completed by the process performers for each enactment of the process. It is used for recording information about the process enactment to clarify whether the process proceeds according to the modeled process. After the PEAFs are completed by process performers, the information obtained from these forms is used in the Process Similarity Matrix to create an overall picture of process enactments.

A PEAF, template of which is given in Fig. 4, captures the instant values of process attribute values for a process enactment and is adapted from an asset called Process Execution Record. Process performers record the actual values of process attributes for a specific process enactment on this form. As independent of whether a process is defined or not, completed PEAFs constitute traces and provide detailed information regarding the process execution. Recorded values are merged as a list of process attribute values that are later entered into a PSM.

A PSM, template of which is shown in Fig. 5, is used to verify the values of process attributes that are inputs, outputs, activities, infrastructure, communication channel, and staff, for process enactments. The software process owner enters the values of process attributes in the rows of the matrix and the numbers of process enactments in the columns. He or she reviews each process execution, questions the process attribute values, and marks each cell with an "O" if the value applies. The completed matrix helps practitioners see



Process Name		Process Start Date:	
		Process End Date:	
Module / Component Name:		Recorder:	
		Record Date	

PROCESS INPUTS

No	Input Name	Y/N	Detail
1	*Input Description – 1*		
…	…		
N	*Input Description – n*		

PROCESS OUTPUTS

No	Output Name	Y/N	Detail
1	*Output Description – 1*		
…	…		
N	*Output Description – n*		

PROCESS ACTIVITIES

No	Activity Name	Y/N	Detail
1	*Activity Description – 1*		
…	…		
n	*Activity Description – n*		

INFRASTRUCTURE

No	Question	Y/N	Detail
1	Is there a problem with the office working area?		
2	Is there a problem with the infrastructure used for development?		
3	Is there a decrease in the resources and time reserved for process?		

COMMUNICATION CHANNEL

No	Question	Y/N	Detail
4	Is there a problem with the project manager?		
5	Is there a problem with the customer?		
6	Is there a problem with the other team members?		
7	Is there a problem with the domain experts?		

STAFF

No	Question	Y/N	Detail
8	Have practitioners received training for their roles in the process?		
9	Have practitioners had (sufficient) prior experience in order to carry out their roles in the process?		
10	Did process practitioners need to work overtime?		

OTHER (Is there any other internal or external factors that you want to add other than mentioned above?)

Fig. 4 Process enactment attributes form template

variations in the process enactments with respect to process attribute values and identify inconsistencies in the process. The variations and the inconsistencies observed in the process are the likely causes that originate the software defects. The final list of causes is used to initiate process improvements, and the process variants are useful to cluster process enactments and data for quantitative process management (Tarhan and Demirors 2011).

4 Implementation

This section provides detailed information about the implementation of the case study in parallel with the activities of the defect causal analysis method.

4.1 Select the defect set to explore the causes

The defect set should be chosen for product part(s) produced by the processes that are subject to improvement. For example, if the causes for the requirement analysis process are to be investigated, the defect set should be chosen to include the ones that are originated from the enactment of this process. For the initial analysis within the case study (i.e., pre-improvement), 300 defects with a high and critical severity were chosen from versions V1 through V6 of Project X. The common characteristic of the selected defects was that they were all obtained from the development and maintenance phases of the modules B and C.

			Process Enactments			
Process Attributes			PE1	PE2	…	PEm
1	**Inputs**					
	1.1	*Input Description - 1*				
	…	…				
	1.n	*Input Description – n*				
2	**Outputs**					
	2.1	*Output Description - 1*				
	…	…				
	2.n	*Output Description - n*				
3	**Activities**					
	3.1	*Activity Description - 1*				
	…	…				
	3.n	*Activity Description - n*				
4	**Infrastructure**					
	4.1	Is there a problem with the office working area?				
	4.2	Is there a problem with the infrastructure used for development?				
	4.3	Is there a decrease in resources and time reserved for process?				
5	**Communication Channel**					
	5.1	Is there a problem with the project manager?				
	5.2	Is there a problem with the customer?				
	5.3	Is there a problem with the other team members?				
	5.4	Is there a problem with the domain experts?				
6	**Staff**					
	6.1	Have practitioners received training for their roles in the process?				
	6.2	Have practitioners had (sufficient) prior experience to carry out their roles in the process?				
	6.3	Did process practitioners need to work overtime?				

Fig. 5 Process similarity matrix template

4.2 Apply ODC to the defect set and analyze the results

The application of the ODC technique requires the analysis of the defects data regarding the attributes of type, trigger, and origin in order to identify the root causes for the selected defects. The ODC attributes were defined for each defect in the selected defect set, retrospectively. First, the *defect type* was identified by analyzing the traces and explanations given for the correction of defects on the defect recording tool. Where the defect type could not be identified, information about the defect was obtained from the person who corrected the defect and the defect type was identified. The process of defect type identification took an average of five minutes when there was no need to consult with the person who corrected the defect. As a result, 300 defects were orthogonally classified. The *defect trigger* was obtained from the defect record, activity information, and the person who reported the defect. Information about defect origins was obtained from the activity of defect correction. The *defect origin* was identified as follows: requirement, if there was a change in the requirements; design, if there was a change in the design; and coding, if there was a change in the code. If the defect occurred in the maintenance phase after passing the acceptance test, the defect origin was classified as maintenance. Since the process of identifying defect origins required a detailed analysis, an average of twelve minutes was allocated for each defect to obtain the relevant information.

Following the identification, the analysis of the ODC technique was performed by classifying the defects according to the attributes through the extraction of the derived metric values. Figure 6 shows the distribution of the defect types with respect to defect triggers. The distribution of the defects by the defect origin was as follows: Requirements Engineering: 34 %, Technical Solution: 32 %, Coding: 26 %, and Maintenance: 8 %.

A general evaluation of Fig. 6 indicates that the defects in types other than Documentation and Timing/Serialization were mostly discovered in the system test stage. The Timing/Serialization defects could not be discovered in the system test stage and therefore reached the customer. The Requirement, Design, and Code reviews were found inefficient

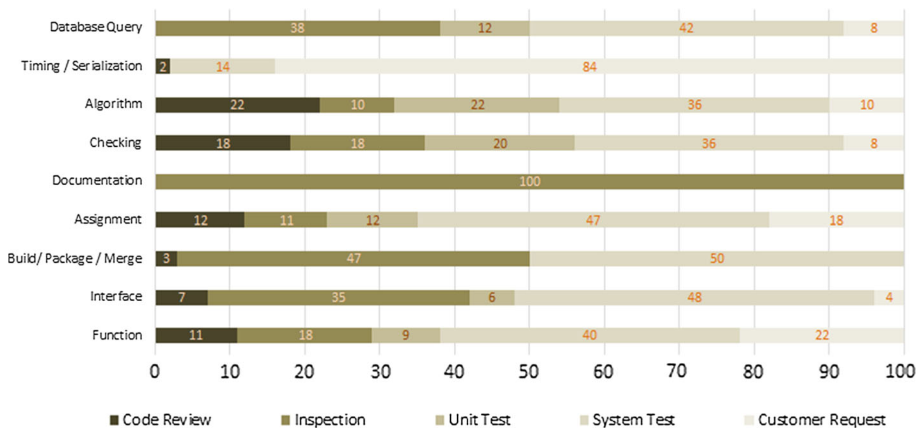


Fig. 6 Distribution of the defect types with respect to defect triggers (pre-improvement) (*color codes* and numbers per defect type highlight defect triggers and % distributions of defects that are detected by these triggers.)

in discovering Function and Interface type defects. The Assignment, Checking, and Algorithm defects were mostly discovered in the system test stage, which indicated that code reviews and unit tests are not adequate. The Inspection trigger showed that developers found the defects when inspecting the code. Since these defects could not be found before the code completion and were only discovered during the inspection stage, they have a higher effect on the cost. In other words, the later the defect was discovered, the higher the correction cost would be.

4.3 Evaluate the results of the ODC application

After defects are classified according to the attributes of the ODC, semantic information is extracted from the defect set and root causes of the defects are identified in order to suggest improvement actions to prevent the occurrence of future defects due to the same causes.

In the case study, three developers reviewed and discussed the results from the initial ODC implementation. The following evaluations were made after the analysis of the defects with respect to the ODC attributes:

- Requirement reviews can be more effective.
- Design reviews can be more effective.
- Code reviews can be more regular and effective.
- The effectiveness of unit tests can be increased.
- Defects in database queries can be reduced.
- The number of changes in the design due to misunderstandings in requirement stage can be reduced.
- The effectiveness of unit tests can be increased to discover code-related defects in Algorithm, Assignment, and Checking defect types.

The root causes of the defects can be identified by grouping the defects on a Fishbone diagram; this is a recommended approach for the ODC technique. The causes identified after the ODC implementation (pre-improvement) is shown by solid lines in Fig. 8.

4.4 Gather and analyze process enactments originating the defects

Data from process enactments are analyzed in comparison with the expected outcomes (e.g., process definitions) to identify the weak points. In the case study, the first stage in this activity was to model the Requirement Engineering and Technical Solution processes that are implemented for product development in Project X. The reason for focusing on these two processes was that the defects that were discovered in the later stages of development mostly originated in the earlier stages. The Requirement Engineering process was divided into two phases: preliminary analysis and analysis. Following the modeling of these phases, the inputs, activities, and outputs were determined separately. The Technical Solution process was also divided into two phases: Creating the Design and Applying the Design.

Following the modeling of the Requirement Engineering and Technical Solution processes, the input, activity, and output components in these models were moved to their PEAFs. The prepared forms were distributed to process performers for completion during their process enactments. The PEAFs created for preliminary analysis and analysis phases of the Requirement Engineering process were completed by 7 performers to provide information about their process enactments. The information obtained from these forms was coded into the PSM that was prepared for the Requirement Engineering process. The PEAFs created for Creating the Design and Applying the Design phases of the Technical Solution process were completed by 10 performers to provide information about their process enactments. The information obtained from these forms was coded into the PSM that was prepared for the Technical Solution process. The PSM for Creating the Design phase is shown in Fig. 7.

4.5 Evaluate deviations in process enactments

The deviations in the process enactments are evaluated using the information encoded in the PSM. Each column of the PSMs corresponds to the state of a process enactment by a process performer. The analysis of all the columns allows the rapid and efficient acquisition of the information regarding the points where the process enactments deviate from the expectations of the process model. Contextual factors that may have an effect on the process (included in the PEAF under the headings of Infrastructure, Communication Channels, and Employees) are also analyzed.

The identification of a process deviation (or variation) depends on the perspective of process owner and process performers. In the case study if a process attribute value of input, output, activity, or staff was missing in more than half of the enactments, then it was marked as a potential root cause (as indicated by "*" in the leftmost column in Fig. 7). For the process attributes of infrastructure and communication channel, if an attribute value was missing in more than one-third of the enactments, then it was marked as a potential root cause. The potential causes were identified and elicited from the PSM this way and discussed with process owners and performers to finalize the root causes.

The results of the analysis of the Requirement Engineering process enactments via the PSMs showed that:

- Risks should be thoroughly considered.
- Process entities should be used more systematically.
- The completeness of requirements should be ensured.
- The verification of the requirements by the customer should be improved.
- The software requirement review should be more systematic.

			Process Enactments									
Process Attributes			PE1	PE2	PE3	PE4	PE5	PE6	PE7	PE8	PE9	PE 10
1	**Input**											
	1.1	Alternative design opinion. DAR (Decision Analysisand Resolution) process area			o		o		o	o	o	o
*	1.2	Selection criteria			o						o	o
*	1.3	Selection criteria, Configuration Management process area					o					
*	1.4	Alternative design for each alternative design opinion	o		o				o			o
*	1.5	Chosen high level design, high level design template					o					
*	1.6	High level design document, V&V (Verification and Validation) process area					o			o	o	o
	1.7	High level design	o	o		o	o	o		o	o	
	1.8	Design model	o		o	o	o	o	o	o		
	1.9	Software design document template, design model and database model			o	o	o	o	o	o	o	o
*	1.10	Software design document, V&V process area								o		
2	**Output**											
*	2.1	Selection criteria for each alternative design			o		o					
	2.2	Alternative designs	o		o				o	o	o	o
*	2.3	High level design for each alternative design	o				o					
	2.4	Chosen high level design	o	o	o	o	o	o	o			
	2.5	High level design document	o	o			o			o	o	o
*	2.6	Reviewed high level design document										
	2.7	Design model	o	o		o	o	o	o	o	o	o
	2.8	Database model	o	o	o	o	o	o	o	o	o	o
	2.9	Software design document	o	o	o	o		o	o	o	o	o
	2.10	Reviewed software design document										
3	**Activities**											
	3.1	Selection criteria was determined between alternative designs			o	o	o				o	o
	3.2	Alternative designs were developed	o	o	o	o			o	o	o	o
*	3.3	High level design alternative was developed	o			o	o					
	3.4	Solutions for product components were chosen	o	o	o		o	o	o			
	3.5	High level design document was prepared	o	o			o			o	o	o
	3.6	High level design document was reviewed				o	o			o	o	o
	3.7	Design model was developed	o	o	o	o	o	o	o	o	o	o
	3.8	Database model was developed	o	o	o	o	o	o	o	o		
	3.9	Software design document was prepared	o	o	o	o	o	o	o	o	o	o
*	3.10	Detail software design was reviewed			o							
4	**Infrastructure**											
	4.1	Is there a problem with the office working area?										
	4.2	Is there a problem with the infrastructure used for development?			o		o					
	4.3	Is there a decrease in resources and time reserved for process?										
5	**Communicat-ion Channel**											
	5.1	Is there a problem with the project manager?										
*	5.2	Is there a problem with the customer?	o	o			o		o			
	5.3	Is there a problem with the other team members?										
	5.4	Is there a problem with the domain experts?		o								
6	**Staff**											
	6.1	Have the practitioners received training for their roles in the process?		o		o	o		o	o	o	o
	6.2	Have the practitioners had (sufficient) prior experience to carry out their roles in the process?	o	o	o	o	o		o			
	6.3	Did process practitioners need to work overtime?	o	o	o				o	o	o	o

Fig. 7 Process similarity matrix for 'Creating the Design' activity (* in the *leftmost column* indicates a problematic process attribute value in process enactments)

- The number of rapid requirement changes should be reduced.
- Interface requirements should be more detailed.
- The experience of the implementers should be extended.

The results of the analysis of the Technical Solution process enactments via the PSMs showed that:

- The stage of creating alternative designs should be improved.
- The determination of high-level designs should be undertaken on a regular basis.
- High-level and detailed designs should be reviewed on a regular basis.
- The detailed design should be reviewed before application.
- Product components should be reviewed on a regular basis.
- Unit tests should test all cases that a piece of code can produce.

4.6 Identify the root causes of defects

This activity includes the analysis of information from both the application of the ODC technique (as identified in Sect. 4.3) and the evaluation of the PSMs for deviations in process enactments (as identified in Sect. 4.5). After the root causes of the defects were identified by using the evaluation results from the two activities, the causes were analyzed

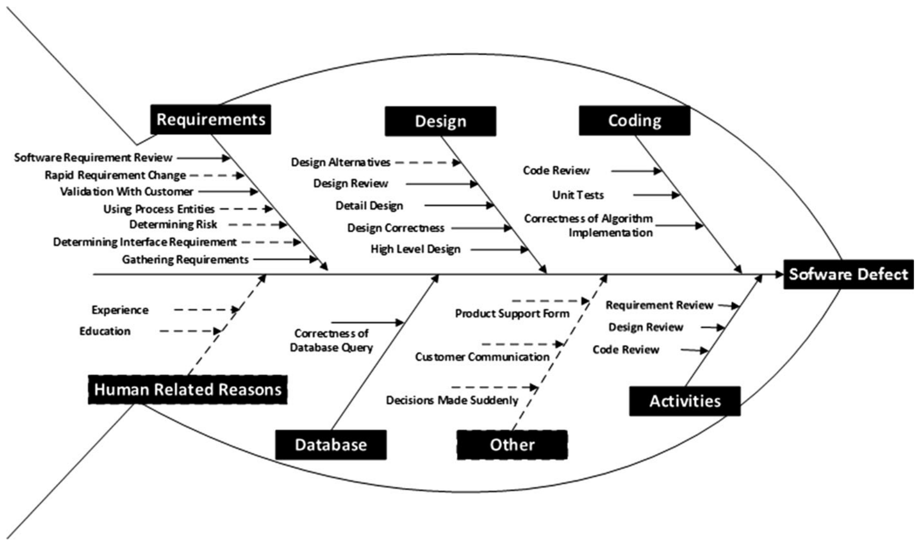


Fig. 8 Fishbone diagrams for the root causes of the defects

together and grouped on the Fishbone diagram. Figure 8 shows the combined causes that are determined by applying the ODC technique and the process enactment analysis. While the causes identified by the ODC technique are represented by solid lines in the figure, the causes identified by the process enactment analysis are represented by dashed lines. These items together represented the causes that originated the software defects in Project X and when addressed, created the opportunity for improvement.

4.7 Offer suggestions for process improvement

Using the Fishbone diagram enables developing suggestions to improve the process. This activity takes less time since the weak points in the process have already been identified. Considering the root causes of the defects, the following suggestions were developed in the case study:

1. Requirement-related risks should be determined in the Requirement Engineering process.
2. There should be a continuous communication with customers at the stage of determining the interface requirements.
3. The requirements should be constantly verified by customers.
4. Software requirements should be reviewed on a regular basis.
5. Rapid requirement changes are not recommended for sub-releases; they should be planned for the main release.
6. Alternative designs should be considered and the best one should be chosen in a rapid manner.
7. The selected design should be modeled in detail.
8. The detailed design should be reviewed on a regular basis.
9. Code reviews should be undertaken on a regular basis.
10. The effectiveness of unit tests should be increased in such a way that they can test all cases.

Table 4 Action items for process improvement and their evaluation by selected project members

Improvement suggestion	Mode of scores (in scale 1–5)
1. Determine risks in the Requirement Engineering process	5
2. *Frequently communicate with the customer while determining interface requirements*	5
3. Regularly validate requirements with the customer	5
4. Regularly review software requirements	4
5. Plan handling of urgent requirements changes in the main releases rather than in the sub-releases	4
6. *Allocate sufficient time to decide on the best design based on many alternatives*	5
7. *Model the selected design in detail*	5
8. Regularly review the detailed design	4
9. Hold code reviews regularly	5
10. Increase the efficiency and coverage of unit tests	5
11. Improve the integration of SQL scripts into the development environment, and increase the efficiency of unit tests for database inquiries	4

11. Since defects in database queries can result in critical errors on the real platform, the management of SQL scripts should be improved using various tools and the database queries should be tested through unit tests.

These suggestions were transformed into action items and shared with a number of people from the development team. Five people including project team leaders and members of the quality team evaluated these items in order to verify the feasibility of the implementation and provide a certain level of confidence on the action items. The following ordinal scale was used for the evaluation: 1: Strongly disagree, 2: Disagree, 3: Neutral, 4: Agree, and 5: Strongly agree. Since the number of people participating in the evaluation was low, the mode of the responses was calculated. Table 4 shows the action items and the mode of the values obtained from the evaluation of them. Mode values show that there is a high-level agreement (with the scores of 4 or 5 meaning Agree or Strongly Agree) on the action items identified.

Most of the action items were immediately taken into consideration and implemented to improve the Project X's development process. The suggestions 2, 6, and 7 (written in italics in Table 4) were eliminated at that moment due to not being feasible in relation to project schedule. The results achieved after the process improvement in comparison with the results of the initial state are discussed in the next section.

5 Results and discussion

The root causes of defects were analyzed by following the causal analysis method, and action items were implemented in Project X. The improved process was taken into execution, and defect data were collected from the Module D that was developed in versions V7 through V12 of the software. This section provides the results of process improvements from the viewpoints of development performance and product quality.

5.1 Improvements in software development process performance (RQ-1 & RQ-2)

Following the process improvement, the ODC technique was applied prospectively by classifying the defects according to their attributes by the developers and then extracting the specified metric values by the first author. The distributions of the defects with respect to the attributes of defect trigger and defect origin in comparison with the ones before process improvement are presented in Fig. 9. The distributions with respect to defect trigger in Fig. 9a demonstrate the improvements in the performance of the V&V processes, while the distributions with respect to defect origin in Fig. 9b indicate the improvements in the performance of the development processes.

According to Fig. 9a, the requirement and design reviews were undertaken more systematically after the improvement. The defects originating from the requirement and design phases were identified by these triggers, and majority of the defects were removed before the system test. A lower percentage of customer change requests indicated that the defect triggers after the improvement were effectively applied. According to Fig. 9b, the defects originating from the Requirement and Design phases were reduced. The main reason for this was the timely performances of the requirement and design reviews. Figure 9b also shows that the distribution of the defects by the defect origin shifted through the later stages of the development, which decreased the development costs. The defects originating from the coding phase having the highest percentage after the improvement could be corrected more easily and prevented by the improvement of unit tests and code reviews.

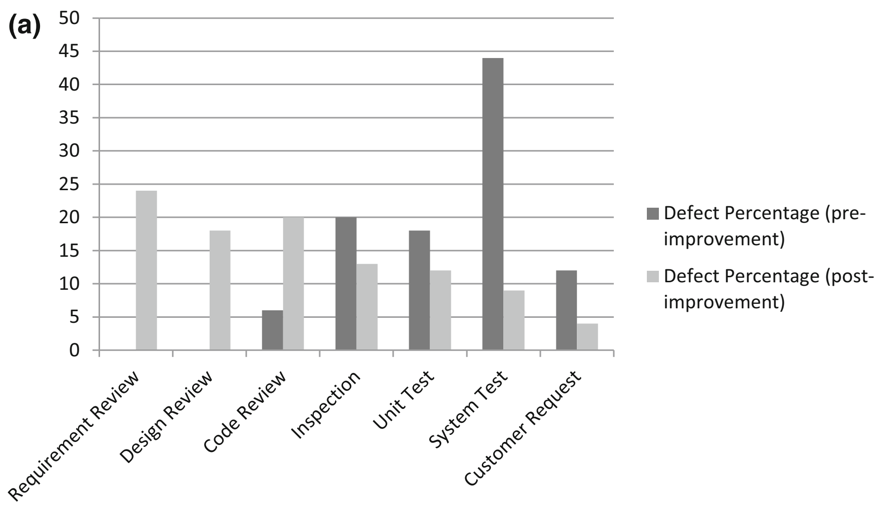
5.2 Improvements in software product quality (RQ-1 & RQ-2)

The distributions of the defects with respect to the attribute of defect type in comparison with the ones before process improvement are presented in Fig. 10.

According to Fig. 10, there was a decrease in the number of defects in Function and Interface types. The number of Database Query defects was also reduced after the SQL scripts were moved to the Integrated Development Environment, and instantaneous checks and continuous unit tests were conducted. Further improvement was required for the Assignment, Checking, and Algorithm defect types.

An evaluation of Fig. 10 together with Fig. 9a indicates that the most of the Function and Interface defects were discovered in the requirement, design, and code reviews after the improvement, and severe defects were prevented from occurring in the software in the later stages. Function and Interface defects that originated from the Requirement and Design phases were discovered during these reviews, which also increased the product quality. Table 5 presents the comparative distribution of the defect types with respect to defect triggers (in percentages by columns).

The distributions of the defect densities through the versions of the software product were also identified in the case study. The defect density was calculated by dividing the number of defects detected in a product version by the software length in thousand of lines of code (KLOC). Figure 11 shows the cumulative values of these densities along the six consecutive product versions prior to and after implementing the process improvements. According to the figure, the values of the defect densities in the versions V7 through V12 were slightly less than the half of the values of the defect densities in the versions V1 through V6, which shows the significant effect of process improvements on the product



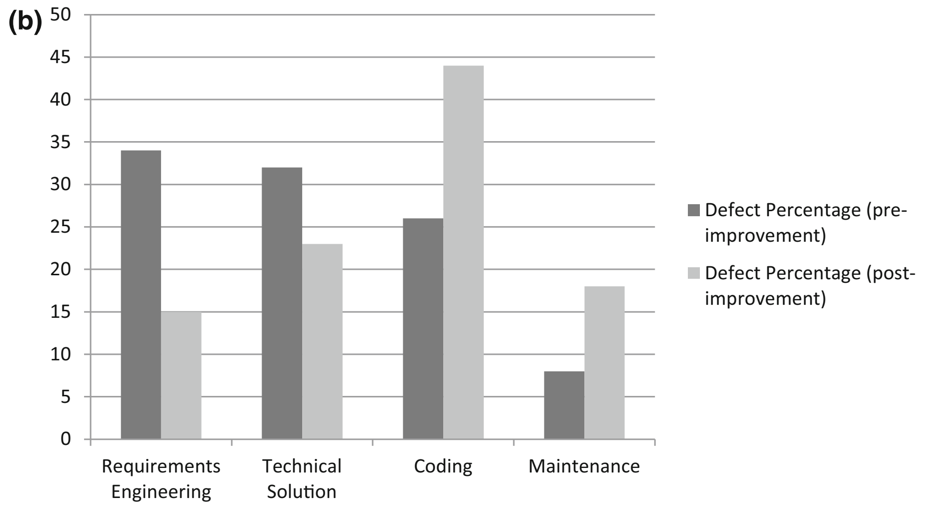


Fig. 9 Comparative distributions of defects with respect to defect triggers and defect origins

quality. The trends of the cumulative defect densities through the versions were similar before and after process improvements, the latter showing a better descending slope. Still, the slopes were far from approaching a horizontal line since the product development was in progress and the project was not close to its end. In other words, the development of new product versions was introducing new defects through the iterations of the software development process.

5.3 Cost of root-cause analysis (RQ-3)

In the initial defect analysis, the first author spent five minutes per defect to identify the attributes for the selected 300 defects, resulting in a total effort of 25 person-hours. The

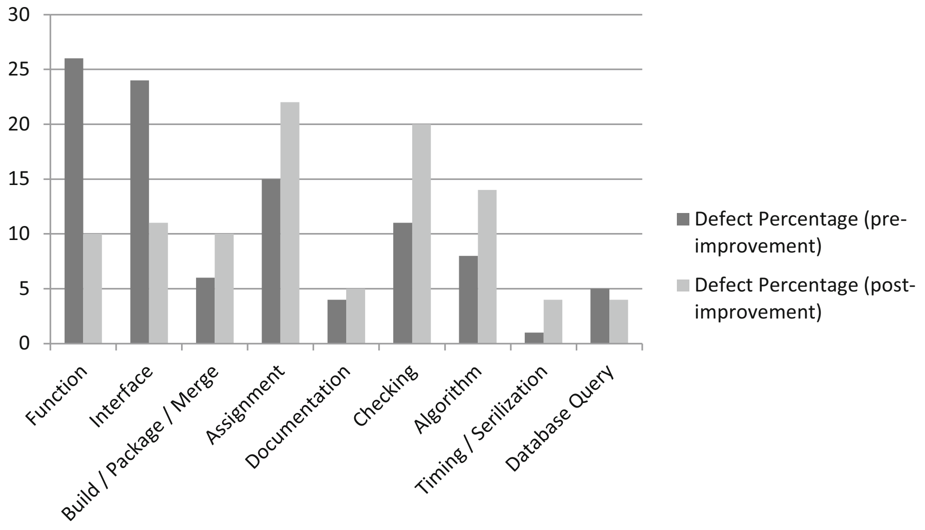


Fig. 10 Comparative distributions of defects with respect to defect types

assigned attributes were then reviewed by three experienced developers in six hours, leading to an additional effort of 18 person-hours. The developers retrospectively completed 17 Process Enactment Attribute Forms to collect process enactment data, and they claimed that the completion of a form took about three minutes. Thus, the total effort spent for the collection of the process enactment data was 51 person-minutes which is about a person-hour. Also, it took four person-hours to derive the root causes of the defects from the collected data and determine the process improvement suggestions. Therefore, the overall effort spent for the causal analysis by following the suggested method was 48 person-hours.

In the defect analysis performed after applying the improvement suggestions, it took one person-hour to derive the distributions since the defects attributes were already entered by the developers during development by the prospective application of the ODC technique.

6 Threats to validity

The validity of the case study is evaluated in terms of four tests, which are used commonly to establish the quality of the empirical research, as summarized by Yin (2013). These tests are construct validity, internal validity, external validity, and reliability (or conclusion validity).

The *construct validity* is related to identifying correct operational measures for the concepts being studied, and it requires developing a sufficiently operational set of measures and to avoid subjective judgments to collect the data. The construct validity of the case study is assured by the usage of the ODC technique as well as the predefined forms like PEAF and PSM. The defect attributes defined by the ODC are used to obtain the base measures in the analyses of defects attributes.

Table 5 Comparative distribution of the defect types with respect to defect triggers

Defect type versus defect trigger (% distributions of pre-imp and post-imp)	Function	Interface	Build/package/merge	Assignment	Documentation	Checking	Algorithm	Timing/serialization	Database query
Requirement Review (pre-imp)	0	0	0	0	0	0	0	0	0
Requirement Review (post-imp)	25	4	0	0	46	0	0	0	0
Design Review (pre-imp)	0	0	0	0	0	0	0	0	0
Design Review (post-imp)	24	22	2	0	42	0	0	0	0
Code Review (pre-imp)	11	7	3	12	0	18	22	2	0
Code Review (post-imp)	14	16	23	26	0	30	30	22	0
Inspection (pre-imp)	18	35	47	11	100	18	10	0	38
Inspection (post-imp)	13	22	27	26	12	22	22	46	42
Unit test (pre-imp)	9	6	0	12	0	20	22	0	12
Unit test (post-imp)	8	18	28	36	0	38	38	6	46
System test (pre-imp)	40	48	50	47	0	36	36	14	42
System test (post-imp)	10	14	20	8	0	8	9	14	7
Customer request (pre-imp)	22	4	0	18	0	8	10	84	8
Customer request (post-imp)	6	4	0	4	0	2	1	12	5

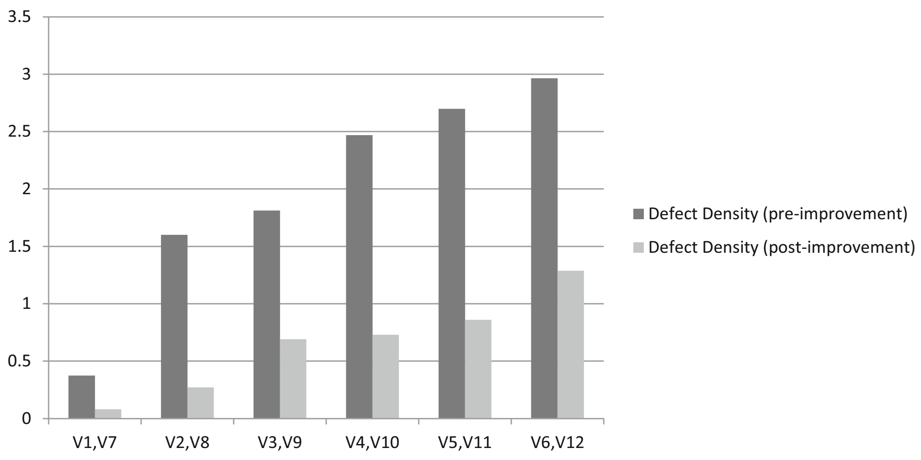


Fig. 11 Comparative distributions of defects densities through product versions

The *internal validity* is applicable for explanatory or causal studies only and not for descriptive or exploratory studies. It requires seeking to establish a causal relationship, whereby certain conditions are believed to lead to other conditions, as distinguished from spurious relationships. The internal validity of the case study is supported by the following factors; the selected modules were from the same project, the team members and the development infrastructure were the same, and the project's process had been defined. However, the selection of different product versions to compare the results, the complexity of the domain, the level of experience of the team members in the development, and the subjectivity that might have included in the identification of the defects attributes are the threats to the internal validity of the study. We should also note that the maturation of the overall software product throughout the releases in iterations is another threat to the validity of the results. The subject to the second analysis was a new Module D of the product (i.e., released by the versions 7–12), and not the modules B and C (i.e., released by the versions 1–6) as in the first analysis. Still, the overall software product and the environment that it was produced in (e.g., the components of the infrastructure, the experience of developers, and team cohesion) might have become more stable with every iteration in product development.

The *external validity* is about defining the domain to which a study's findings can be generalized. The case study was carried out on a defect set selected from a single project of a software institute for the specific purpose of process performance and product quality improvement. Further studies are needed to investigate the effectiveness of the improvement method in different contexts and to confirm the external validity of the results found by this research.

As the last test, the *reliability* is related to demonstrating that the operations of a study can be repeated, with the same results. Here the emphasis is on doing the same case over again, not on replicating the results of one case by doing another case study—which is vital to external validity. In our study, the method explained in Sect. 3 was used to conduct the causal analysis, and the requirements of the ODC technique were obeyed in the defect analyses conducted prior to and after the improvement. Therefore, the methodological replication of the case study is enabled, but with a certain degree of subjectivity that might have been included due to human factors.

7 Conclusion

Causal analysis of software defects creates opportunities for the improvement of the development process and product quality. Identifying the root causes and the improvement items enables the development of process awareness in the software teams regarding the practices of both technical and V&V processes. This might also contribute to developing a process-oriented culture within software organizations, which is already expected by the process improvement models like CMMI at higher maturity levels. Software teams and organizations that follow iterative development practices might especially benefit from this kind of analysis and improvement.

In this study, the root causes of the defects of critical importance were analyzed by following the causal analysis method that was suggested as specific to the study, and the action items for process improvement were identified and implemented. The distributions of the defect attributes provided by the ODC technique enabled us to perform a quantitative comparison between the initial and improved states of the development performance and product quality. As a result, process performance and product quality were found to have been positively affected due to earlier identification of defects and reduced cost of software quality.

The root causes of the defects were identified from the evaluation of the results of the ODC technique as well as from the analysis of process enactment data that showed the origin of the defects. After the improvements were made to eliminate the root causes of the defects and the improved processes were deployed to develop a new module, the comparative results showed that the distance between the injection and the detection of the defects was reduced; that is, the defects could be detected earlier the life cycle, thus reducing the development costs.

The method defined and followed for the causal analysis was effective and efficient in identifying the root causes and demonstrating the benefits of the process improvements. The utilization of the process enactment data in the defect causal analysis was supportive to the ODC technique, and enabled the identification of the root causes at a deeper level since it provided insight into the components and contextual factors of the processes. However, though the recording of the process enactment data required only little effort, convincing the developers to do so was not very easy. It was only after the results of the study were shared with the developers that they could see the importance of recording the process enactment data. We find the management's support and the feedback from the process improvement team crucial here.

The method used in this case study was defined in accordance with the requirements of the CMMI CAR process area and was based on the principle of continuous improvement. In other words, the defects identified in the project can be continuously analyzed in cycles by using the method and new action items for further process improvements can be identified in each cycle. Software organizations that regularly record their defects and have well-structured (and not necessarily defined) processes in terms of input, activity, and output components may plan and implement similar implementations. A software application or add-on to the defect tracking system that enables the automatic collection of the enactment data at each phase of the development process might ease capturing and recording of the enactment data.

References

Andersen, B., & Fagerhaug, T. (2006). *Root cause analysis: simplified tools and techniques*. Milwaukee: ASQ Quality Press.

Aslan, D., Tarhan, A., & Demirörs, O. (2014). How process enactment data affects product defectiveness prediction—a case study. In R. Lee (Ed.), *Software engineering research, management and applications* (pp. 151–166). Heidelberg: Springer International Publishing. doi:10.1007/978-3-319-00948-3_10.

Bassin KA, Santhanam P (1997) Use of software triggers to evaluate software process effectiveness and capture customer usage profiles. Proceedings The Eighth International Symposium on Software Reliability Engineering—Case Studies 103–114. doi:10.1109/CSSRE.1997.637852

Bassin, K. A., Kratschmer, T., & Santhanam, P. (1998). Evaluating software development objectively. *IEEE Software, 15*(6), 66–74. doi:10.1109/52.730846.

Boehm, W. B. (1984). Software engineering economics. *IEEE Transactions on Software Engineering, SE-10*(1), 4–21. doi:10.1109/TSE.1984.5010193.

Bridge, N., & Miller, C. (1998). Orthogonal defect classification using defect data to improve software development. *Software Quality, 3*, 1997–1998.

Buglione, L., & Abran, A. (2006). Introducing root-cause analysis and orthogonal defect classification at lower CMMI maturity levels. In *Proceedings of International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA 2006)* (p. 29). Cadiz, Spain. Retrieved from http://www.gelog.etsmtl.ca/publications/pdf/1037.pdf

Butcher, M., Munro, H., & Kratschmer, T. (2002). Improving software testing via ODC: Three case studies. *IBM Systems Journal, 41*(1), 31–44. doi:10.1147/sj.411.0031.

Carleton, A., & Florac, W. (1999). *Measuring the software process: Statistical process control for software process improvement*. Boston: Addison-Wesley Professional.

Chillarege, R. (1996). Handbook of software reliability engineering. In M. R. Lyu (Ed.), (pp. 359–400). Hightstown, NJ, USA: McGraw-Hill, Inc. Retrieved from http://dl.acm.org/citation.cfm?id=239425.239453

Chillarege, R. (2013). Using ODC to diagnose an Agile enterprise application development project. *IEEE International Symposium on Software Reliability Engineering Workshops, (ISSREW 2013)*, p. 45. doi:10.1109/ISSREW.2013.6688862

Chillarege, R., Bhandari, I. S., Chaar, J. K., Halliday, M. J., Moebus, D. S., Ray, B. K., et al. (1992). Orthogonal defect classification—A concept for in-process measurements. *IEEE Transactions on Software Engineering, 18*(11), 943–956. doi:10.1109/32.177364.

Chillarege, R., & Prasad, K. R. (2002). s_Test and development process retrospective—a case study using ODC triggers. *Proceedings of International Conference on Dependable Systems and Networks,*. doi:10.1109/DSN.2002.1029012.

Dubey, A. (2012). Towards adopting ODC in automation application development projects. *Proceedings of the 5th India Software Engineering Conference (ISEC'12),*. doi:10.1145/2134254.2134282.

Feiler, P. H., & Humphrey, W. S. (1993). Software process development and enactment: Concepts and definitions. *Software Process 1993 Continuous Software Process Improvement Second International Conference on the, Berlin, Germany*, pp. 28–40. doi:10.1109/SPCON.1993.236824

Freimut, B. (2001). *Developing and using defect classification schemes. Fraunhofer IESE*. Retrieved from http://en.scientificcommons.org/20203575

Gómez, O., Oktaba, H., Piattini, M., & García, F. (2006). A systematic review measurement in software engineering: State-of-the-art in measures. In *International Conference on Software and Data Technologies (ICSOFT 2006)* (pp. 165–176). doi:10.1007/978-3-540-70621-2

Grundy, J. C., Mugridge, W. B., & Hosking, J. G. (1997). Utilising past event histories in a process-centred software engineering environment. *Proceedings of the Australian Software Engineering Conference,*. doi:10.1109/ASWEC.1997.623764.

Gürgen, T., Tarhan, A., & Karagöz, N. A. (2014). An integrated infrastructure using process mining techniques for software process verification. In R. Perez-Castillo & M. Piattini (Eds.), *Uncovering essential software artifacts through business process archaeology* (pp. 364–382). Hershey: IGI Global. doi:10.4018/978-1-4666-4667-4.ch014.

Hommes, B. J. (2004). *The evaluation of business process modeling techniques*. Ph.D. thesis, Delft University of Technology, Delft.

Huang, L., Ng, V., Persing, I., Geng, R., Bai, X., & Tian, J. (2015). ac_AutoODC: Automated generation of orthogonal defect classifications. *Automated Software Engineering, 22*, 3–46. doi:10.1109/ASE.2011.6100086.

Humphrey, W. S. (1989). *Managing the software process*. Boston: Addison-Wesley Professional.

Huo, M., He, Z., & Jeffery, R. (2006). A systematic approach to process enactment analysis as input to software process improvement or tailoring. *Proceedings—Asia-Pacific Software Engineering Conference, APSEC*, pp. 401–408. doi:10.1109/APSEC.2006.14

IEEE. (1990). *IEEE standard glossary of software engineering terminology (IEEE Std 610.12-1990). Los Alamitos. CA: IEEE Computer Society* (Vol. 121990). Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:IEEE+Standard+Glossary+of+Software+Engineering+Terminology+(IEEE+Std+610.12-1990)#0

Ishikawa, K. (1986). *Guide to quality control* (2nd ed.). Clearwater: Quality Resources.

ISO. (2015). *ISO 9001: Quality management system-requirements*. Geneva: International Organization for Standardization.

Jalote, P., & Agrawal, N. (2005). Using defect analysis feedback for improving quality and productivity in iterative software development. In *Proceedings of 3rd International Conference on Information and Communications Technology (ITI 2005)—Enabling technologies for the new knowledge society*. (pp. 703–713). doi:10.1109/ITICT.2005.1609661

Jones, C. (2008). *Applied software measurement: Global analysis of productivity and quality* (3rd ed.). McGraw-Hill Osborne Media. Retrieved from http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0071502440

Kabbaj, M., Lbath, R., & Coulette, B. (2008). A deviation management system for handling software process enactment evolution. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5007 LNCS*, pp. 186–197. doi:10.1007/978-3-540-79588-9_17

Kalinowski, M., Card, D. N., & Travassos, G. H. (2012). Evidence-based guidelines to defect causal analysis. *IEEE Software, 29*(4), 16–18. doi:10.1109/MS.2012.72.

Kan, S. H. (2002). *Metrics and models in software quality engineering* (2nd ed.). Boston, MA: Addison-Wesley Longman Publishing Co., Inc.

Kanniappan, H. S., & Mishra, U. (2011). Orthogonal defects classification of observed defects in C-DAC Noida projects. In *Proceedings of Annual Seminar of C-DAC Noida Technologies (ASCNT-CDAC 2011)*. Noida, India.

Kuhrmann, M., Kalus, G., & Then, M. (2014). The process enactment tool framework—Transformation of software process models to prepare enactment. *Science of Computer Programming, 79*, 172–188. doi:10.1016/j.scico.2012.03.007.

Kumaresh, S., & Baskaran, R. (2010). Defect analysis and prevention for software process quality improvement. *International Journal of Computer Applications, 8*(7), 42–47. doi:10.5120/1218-1759.

Li, Z. B., Hou, X. M., Yu, L., Du, Z. P., & Xu, B. (2011). Analysis of software process effectiveness based on orthogonal defect classification. *Procedia Environmental Sciences (PART A), 10*, 765–770. doi:10.1016/j.proenv.2011.09.124.

Poncin, W., Serebrenik, A., & Brand, M. Van Den. (2011). Process mining software repositories. *2011 15th European Conference on Software Maintenance and Reengineering*, pp. 5–14. doi:10.1109/CSMR.2011.5

Rubin, V. A., Günther, C. W., Van Der Aalst, W. M. P., Kindler, E., Van Dongen, B. F., & Schäfer, W. (2007). Process mining framework for software processes. In *Software process dynamics and agility* (Vol. 4470, pp. 169–181). doi:10.1007/978-3-540-72426-1_15

Rubin, V. A., Mitsyuk, A. A., Lomazova, I. A., & van der Aalst, W. M. P. (2014). Process mining can be applied to software too! *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement—ESEM'14*, pp. 1–8. doi:10.1145/2652524.2652583

SEI. (2010). *CMMI for development, Version 1.3*

Shenvi, A. A. (2009). Defect prevention with orthogonal defect classification. In *Proceedings of the 2nd India Software Engineering Conference* (pp. 83–88). New York, NY, USA: ACM. doi:10.1145/1506216.1506232

Si, Q., & Yan, G. (2013). Research on quality assurance method based on software defect analysis. In *Proceedings of the 26th Conference of Spacecraft TT&C Technology in China* (Vol. 187, pp. 371–378). doi:10.1007/978-3-642-33663-8

Söylemez, M., & Tarhan, A. (2013). Using process enactment data analysis to support orthogonal defect classification for software process improvement. In *Proceedings of International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA 2013)* (pp. 120–125). doi:10.1109/IWSM-Mensura.2013.27

Tarhan, A., & Demirors, O. (2011). Investigating the effect of variations in the test development process: A case from a safety-critical system. *Software Quality Journal, 19*(4), 615–642.

Tarhan, A., & Demirors, O. (2012). Apply quantitative management now. *IEEE Software, 29*(3), 77–85. doi:10.1109/MS.2011.91.

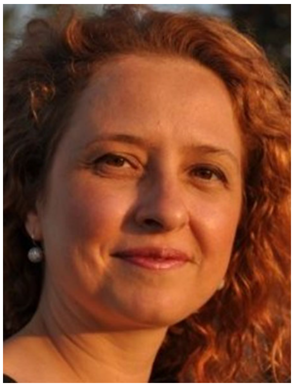
Van der Aalst, W. M. P., & Weijters, A. J. M. M. (2004). Process mining: A research agenda. *Computers in Industry, 53*(3), 231–244. doi:10.1016/j.compind.2003.10.001.

Wagner, S. (2008). Defect classification and defect types revisited. In *Proceedings of the 2008 Workshop on Defects in Large Software Systems* (pp. 39–40). New York, NY, USA: ACM. doi:10.1145/1390817.1390829

Wolf, A. L., & Rosenblum, D. S. (1993). A study in software process data capture and analysis. In *Proceedings of the Second International Conference on the Software Process—Continuous Software Process Improvement*. doi:10.1109/SPCON.1993.236817

Yin, R. K. (2013). *Case study research: Design and methods* (5th ed.). Thousand Oaks: Sage Publications.

Mehmet Söylemez works as a senior researcher and software engineer at Software Technologies Research Institute in Ankara since 2010. He completed his B.Sc. and M.Sc. in Computer Engineering Department of Hacettepe University and pursues his Ph.D. with a focus on process management in the same department.



Ayca Tarhan works as a researcher and practitioner in the area of software engineering for fifteen years. She is experienced in model-based assessment and improvement of software processes. She pursues her studies with a focus on software quality, software development methodologies, software measurement, business processes, and process management. She completed her Ph.D. in Information Systems at Middle East Technical University and currently works as an Assistant Professor in Computer Engineering Department of Hacettepe University, Ankara.