JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS
J. Softw. Evol. and Proc. ; **00**:1–28
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr

Defect Analysis in Mission-critical Software Systems: a Detailed Investigation

Gabriella Carrozza[§], Roberto Pietrantuono[*], Stefano Russo[‡,†]

SUMMARY

The practice of defect analysis is recognized as an essential task for software process measurement, yet its effective application in the industrial development of large-scale software systems raises several challenges. We report the results of a study conducted at SELEX ES – a large system integrator leader in the market of software-intensive mission-critical systems. The article describes the defect analysis approach that we tailored to evaluate the software development process with respect to the quality of produced software and its relation with the required effort. Three key phases of the process were addressed, regarding the software implementation, the testing phase, and the pre-release defect fixing activity, over a set of six *Computer Software Configuration Items* (CSCIs) developed from 2009 to 2012 for the naval and maritime domain product line.
The analysis highlighted efficiency bottlenecks in each of the monitored phases, providing company engineers with insights about room for process improvement. The implemented approach, the observed phenomena, and the inferred conclusions are of support to practitioners coping with systems, development models, and industrial environments similar to the considered one.

Received . . .

KEY WORDS: Defect analysis, Process Measurement, Mission-critical systems, Industrial Study, Empirical Study, Process Evaluation

1. INTRODUCTION

Data about defects experienced during the software lifecycle are a valuable source of information for product and process quality assessment and improvement. Elaborate methods exist that use defect data (such as the type, the trigger, the injection and detection phase, the impact) for tracking the quality of development artefacts and of process activities, revealing inefficiencies, and supporting process improvement. Examples are the Orthogonal Defect Classification (ODC) [1], and the HP classification [2]. Nonetheless, the effective introduction of defect analysis in the industrial development of large software-intensive systems is heavily conditioned by the context, which finally dictates the objectives and the constraints of the analysis. There is, in fact, a trade-off between the target of the analysis, its potential outcomes, its extensiveness to several process aspects, and the cost required to implement it. For instance, a simple, widely used, method to track and measure the testing process is through software reliability growth models (SRGMs); they are easy to implement, because the defect detection time is the only required input, but, at the same time, they provide

[§]SESM s.c.ar.l., A Finmeccanica Company, Via Tiburtina 1238, 00131 Roma, Italy - gcarrozza@sesm.it
[‡]DIETI, Università degli Studi di Napoli Federico II, Via Claudio 21, 80125, Napoli, Italy - stefano.russo@unina.it
[†]Critiware spin off, Incubatore Incipit, Complesso Univ. di Monte S. Angelo, Via Cinthia, 80126, Napoli, Italy
[*]Correspondence to: DIETI, Università degli Studi di Napoli Federico II, Via Claudio 21, 80125, Napoli, Italy - roberto.pietrantuono@unina.it

limited insights into the process. Oppositely, implementing classification schemes as ODC or HP yields much information on single phases efficiency, but at higher expense and greater intrusiveness. Their application into real industrial settings can be difficult [3], [4], because of start-up costs (e.g., training, process changes), of required customizations (e.g., [5]), of non-immediate visible gain, and of reluctance of people to change their routine job.

This article presents a study of defect analysis in the domain of mission-critical software systems aimed at evaluating the software development process with respect to the quality of produced software and its relation with the effort required to attain that quality. With the objective of identifying software process improvements, a collaboration was started in 2010 among SELEX ES, SESM and the Federico II University. SELEX ES is a Finmeccanica company producing mission- and safety-critical systems in several domains: Avionics, Aerospace, Air Traffic Control, Homeland Security and Vessel Traffic Management. SESM acts as research centre supporting SELEX ES in innovating software quality processes. The systems they produce strongly and increasingly rely on software to satisfy a high number of diverse needs. They are typical large-scale and software-intensive products (with thousands of requirements and millions of lines of code), subject to stringent dependability requirements. Their development involves so a large number of people and teams that a lack of control over the process might easily result in low-quality products and cost or time overruns. In these critical domains, low quality is likely to lead to huge maintenance costs, exacerbated by the wide geographical distribution of systems. Thus, the company software engineering unit - one of the biggest units, accounting for about 1,000 employees in several countries - is focusing on the improvement of product and process quality.

The objective of the study is to assess the *quality-effort* trade-off of the current development process of SELEX ES. We started with an analysis of the existing process, wherein three key requirements clearly come out for a successful assessment and improvement strategy: low intrusiveness, low impact on current practices, low setting and execution cost. This limited any attempt to customize existing approaches (e.g., ODC-based, HP-based) in a top-down fashion. We implemented a bottom-up strategy, in which: the object of the evaluation (i.e., *which aspects of the process* are to evaluate), the type of information to gather or infer (i.e., by *what data*), the way to gather or infer them (i.e., *by what metrics* and measurement procedures), and the way to use it (i.e., *how to interpret and use results*) come out from the analysis of the context. The result merges various types of analyses useful for both product and process evaluation while keeping the impact on current practices low.

The study consists of a set of *Computer Software Configuration Items* (CSCIs) developed from 2009 to 2012 for the naval and maritime domain product line. We addressed research questions on *effectiveness* and *efficiency* in terms of quality and effort of the following process phases: CSCI *implementation*, CSCI *testing*, CSCI defect *fixing*. Results highlight process flaws impacting production efficiency and their potential root causes. Specifically, we derived: *i)* estimates of CSCIs quality; *ii)* effectiveness and efficiency measures of monitored process phases; *iii)* indications on sources of measures variability across process instances; *iv)* data about the impact of suppliers on quality indicators, pointing out the best and the worst contributor to process quality. Output data are supporting SELEX ES engineers to implement proper improvement actions. Besides specific findings, the analysis exposed patterns of typical problems that we conjecture may occur within similar contexts. These are used to frame hypotheses for future research.

In the following, we first introduce the development process of SELEX ES. Section 3 describes the followed method. Section 4 reports results of the analysis, while Section 5 discusses the findings. Section 6 surveys the related work, and Section 7 concludes the paper.

2. BACKGROUND

2.1. Development Process

The typical software process in the mentioned critical domains is based on some customization of the V-Model, described by the MIL-498 standard [6] (the SELEX ES version is in Figure 1).

DOI: 10.1002/smr

This model foresees a strong component-based approach, induced by the size and complexity of systems under stringent time-to-market and quality constraints. Each component, known as *Computer Software Configuration Item* (**CSCI**), is actually a product by itself, highly decoupled from the others, autonomously deliverable and deployable.

CSCIs are the basic units of development. They are usually implemented by supplier companies (and in few cases by internal teams). Thus, an important distinctive aspect of this model is the participation of suppliers. In our study, we refer to such a case, in which CSCIs are implemented by external supplier companies. Supplier's implementation teams take care of coding and of internal pre-release testing of the assigned CSCI. After coding, a CSCI undergoes the qualification testing stage. Detected defects are notified to the implementation team, which fixes them and iteratively provides new releases. Thus, a supplier is in charge of both the implementation and the corrective maintenance (i.e., defect fixing) tasks. Upon CSCI release, an integration testing stage is entered. System and acceptance testing are performed before final product release.

The main development phases and artefacts are:

- *System requirement analysis and specification*, that produce a System and Subsystems Specification (**SSS**) document, with requirements specified for a number of subsystems; it is complemented by the Interface Requirements Specification (**IRS**) with the system interfaces and data model;
- *System design*, which, starting from the SSS, produces a System and Subsystems Design Description (**SSDD**) document, containing the high-level architecture of the envisaged solution, and the allocation of requirements to subsystems;
- *Software requirements analysis and specification*, which, along with the architectural design, produce Software Requirements Specification (**SRS**) documents for each identified CSCI. Each SRS is complemented by an Interface Control Document (**ICD**) specifying the CSCI interfaces and the related data model;
- *CSCI design*, which produces a Software Design Description (**SDD**) reporting the internal design of a CSCI, and the allocation of software requirements to its internal subcomponents. An SDD document is produced for each CSCI. Each SDD is accompanied by an Interface Design Document (**IDD**) that specifies the CSCI internal interfaces and exchanged data.
- *Coding and Fixing*, where the CSCI source code from the SDD is produced, using the continuous feedback from CSCI qualification testing to iteratively fix detected defects before

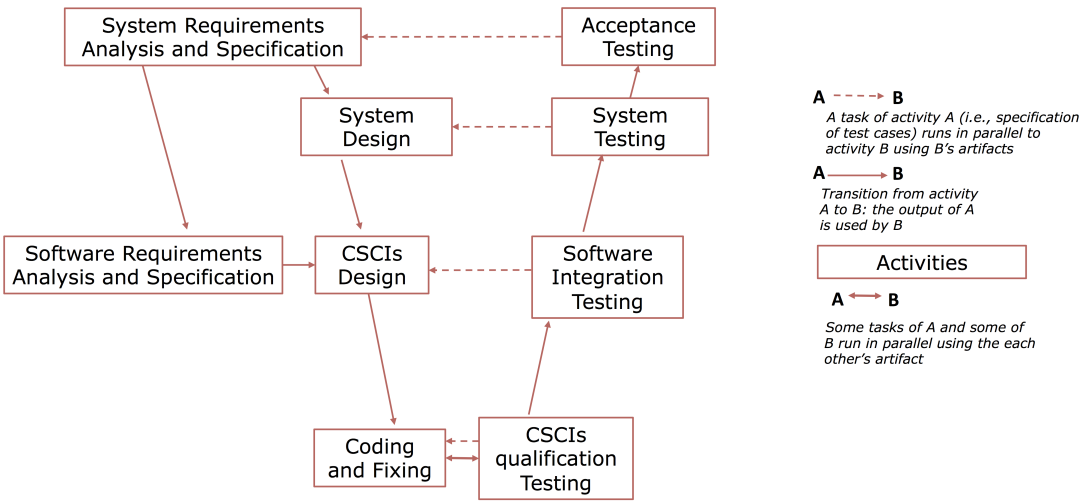


Figure 1. The V-Model in the considered industrial setting [7]

DOI: 10.1002/smr

releasing the CSCI to the integration testing stage. CSCI size typically ranges from several tens to hundreds of KLoC.

Testing phases and artefacts include:

- *CSCI Unit (or qualification) testing*, specifying the software test plan (**STP**) and description (**STD**) documents, running tests, and producing a Software Test Report (**STR**);
- *Software integration testing*, specifying the Software Integration Test Description (**SITD**) document, running tests, and producing the corresponding report (**SITR**);
- *System testing and acceptance testing*, specifying the Acceptance Test Plan (**ATP**), running tests, and producing, respectively, the *in-Factory* and *on-Site Acceptance Test* report (FAT and SAT report), according to the location where tests are performed.

2.2. Defect Lifecycle

The CSCI-based process entails a shared management of defects between supplier and testing teams. Defects are characterized through *issues* managed by an issue tracker. The issue tracker adopted by SELEX ES for the considered CSCIs is *Mantis BT*[†]. Not all the issues are defects, as they represent a generic problem to be managed - for our purpose, we consider only defects, filtering out feature requests and duplicates. The *Mantis BT* instance is configured to display the following fields for a reported issue:

- **ID**: a unique identifier.
- **Summary**: a brief text identifying the problem.
- **Description**: a textual description of the problem manifestation.
- **How to repeat**: the set of steps to reproduce the anomaly.
- **Status**, that can be: *New*, *Assigned*, *Feedback*, *Acknowledged*, *Confirmed*, *Resolved*, *Closed*.
- **Resolution**, indicating the fixing process status: *Open*, *Fixed*, *Reopened*, *Unable to reproduce*, *Not fixable*, *Duplicate*, *No change required*, *Won't fix*.
- **Priority**: *Immediate*, *Urgent*, *High*, *Normal*, *Low*, *None*.
- **Severity**, whose descending criticality levels are: *Block*, *Crash*, *Major*, *Minor*, *Tweak*, *Text*, *Trivial*, *Feature*.
- **Reproducibility**: an indication about the complexity of the issue triggering conditions (*Not always reproducible*, *Always reproducible*).
- **Release**: version in which the issue has been revealed.
- **Timestamp**, associated with the issue "status" variation.

The workflow defined in *Mantis BT* is reported in Figure 2. It shows the issue management and the communication among involved actors, which in our case are: the CSCI *implementation team*, the CSCI *manager* responsible for it, and the CSCI *testing team*. In the sequence, the testing team member submits a detected issue setting the status to *New*. The system notifies the CSCI manager, who redirects the issue to the implementation team, changing the status to *Assigned*. Implementer can ask for further clarification if the description is not clear, changing the status to *Feedback*. He changes the status to *Acknowledged* when has the necessary information to proceed with correction; if no action is required (e.g., it is not a defect), the status is set to *Resolved*; otherwise, it is set to *Confirmed* and the fixing process begins. When fixing is completed, the status is set to *Resolved*, with the implementation team and manager being notified. The manager verifies that the issue is actually fixed, and sets the status to *Closed*.

[†]Mantis BT is available at: *http://www.mantisbt.org/*

DOI: 10.1002/smr

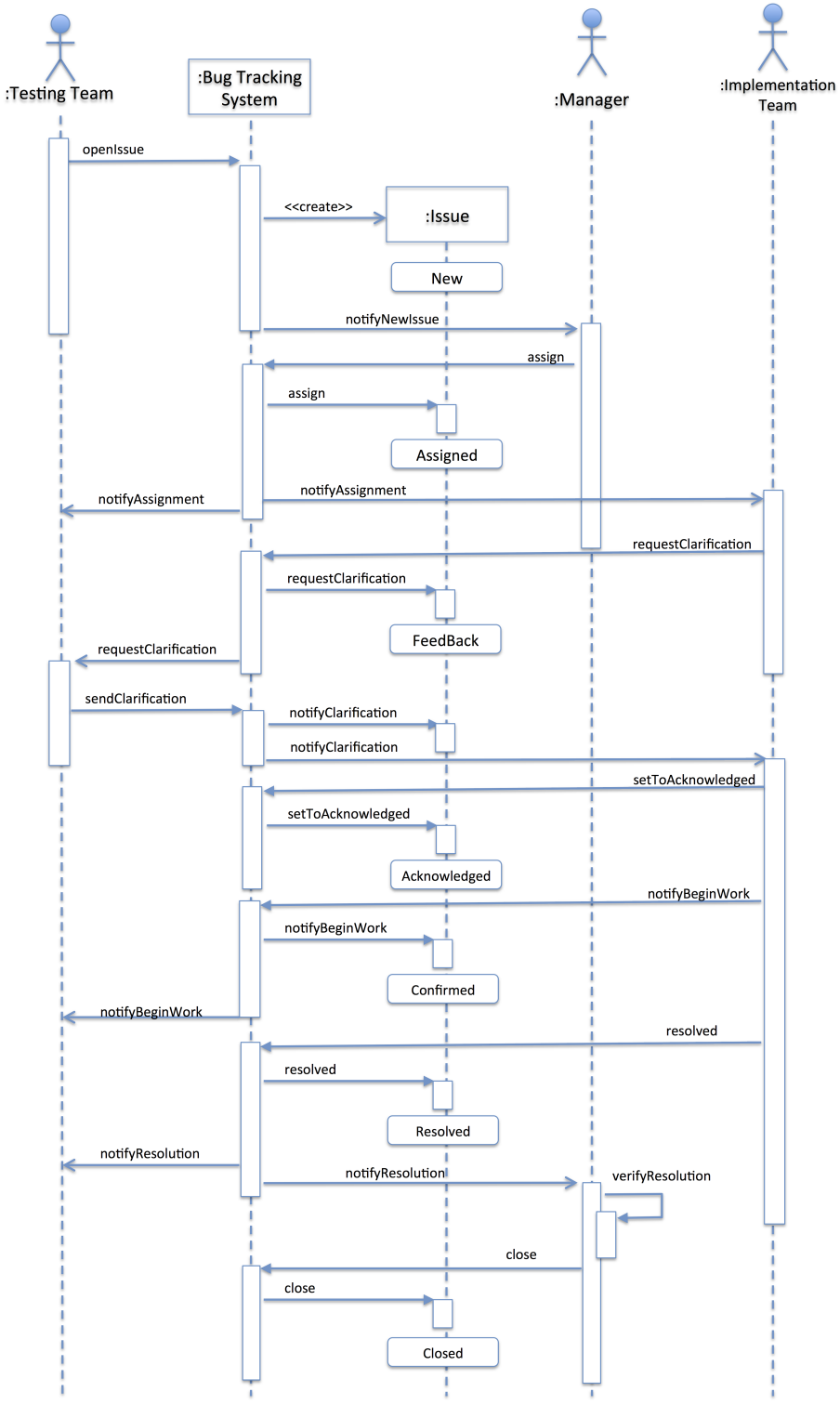


Figure 2. UML Sequence diagram describing the actions upon defect detection. Modified from *Mantis* issue workflow at: *http://www.mantisbt.org/forums/viewtopic.php?f=2&t=12188*, verified on August 8, 2014

DOI: 10.1002/smr

3. ANALYSIS

3.1. Overview

The objective of the study is to evaluate the development process to produce CSCIs, with respect to the *quality vs. effort* balance[‡]. Hence, we consider the CSCI as the basic instance of the process to evaluate. The stated high-level objective is to be achieved with a minimal impact on current development practices. Figure 3 represents the steps we followed to this aim.

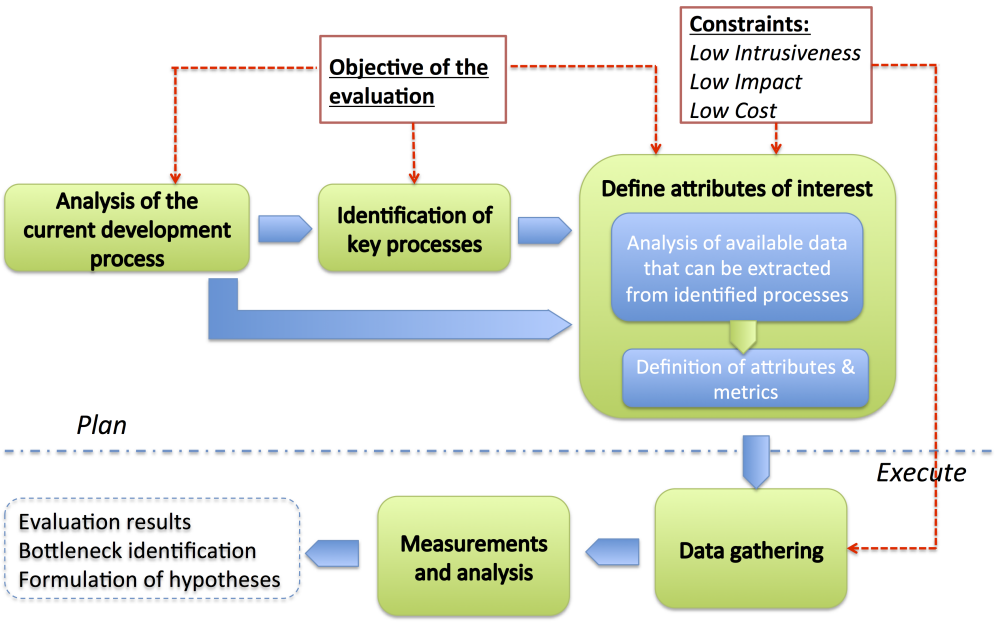


Figure 3. Process Overview

We started from the analysis of the development process as currently implemented in SELEX ES. The development model described in the previous Section tells that, in order to focus on CSCIs evaluation, the starting point is the CSCIs design specification given to suppliers for implementation. Considering the objective of the evaluation and the CSCI lifecycle, the next step has been the identification of the key processes to evaluate, along with a tailoring of the high-level objective for each of the identified process. For each process, we then looked at the effectiveness and efficiency attributes able to fulfil the evaluation objective. In this step, the important constraint of the low impact on current existing practices was considered. This is a requirement that we set together with SELEX ES engineers, in order to get as much information as possible but minimizing the risk of failure and the cost that a change in the current practices could entail. From analysing the context, we recognized, on one hand, the strong need of controlling the quality of what produced (low quality of a maritime traffic control system means huge maintenance cost requiring expensive on-site interventions), while, on the other hand, we recognized the difficulties entailed by the features of the systems developed (in terms of size, complexity, quality requirements), and consequently by the process to build them. Systems are large-scale and software-intensive products with thousands of requirements and millions of lines of code with strict dependability requirements. One of its key features of is the strong decoupling among developers and the involvement of external suppliers. This high modularity and strong component-based approach definitely bring advantages in terms of

[‡]In this context, quality is intended as level of defectiveness of software, while effort indicates the amount of resources spent for an activity on that software. Both defectiveness and effort can be measured in several ways, as showed in the metrics definition Section.

DOI: 10.1002/smr

reuse and implementation cost, and are likely the best way to build these systems, but they also entail numerous people and teams being involved, with different skills, working environments, coding rules and styles, development processes, communication patterns, data tracking and management practices, terminologies, tools, and internal quality standards. This high heterogeneity made it difficult to satisfy the low-intrusiveness, low-impact, and low-cost constraint: we agreed with SELEX and SESM engineers to implement a "black-box" approach, in which there is no need of knowing internal details on *how* the activities of the analysed processes are carried out (e.g., which coding rules, standards, or patterns, which tests generation criteria or tests execution environment, which code analysis or debugging tools, technologies adopted, organizational aspects, and so on), but the needed information is inferred only from what these activities produced, namely form the available defect data, and few other details (e.g., required implementation and testing effort). This allowed us not requiring any process change or any additional effort to developers (e.g., to re-classify defects according to a predefined scheme as could be the case with ODC, HP schemes), avoiding expensive training, terminology alignment, imposition to suppliers, and other adaptation activities. The minimal requirement (already met by the existing process) was the usage of a defect tracking tool and the sharing of defect repositories between the CSCI developers and the CSCI integrator, namely SELEX ES. From inspecting the bug repositories, we identified the common defect attributes gathered by suppliers and based our analysis on them. These include the characteristics of the defects specified in the previous Section. In the "data gathering" step, we queried the bug repositories to collect the needed data. Finally, by analysing the results of measurements as explained in the following, we yielded a picture of what and who contributed more to quality and effort factors, and of possible effectiveness and efficiency bottlenecks. Collected data and observed patterns also allowed formulating some hypotheses on phenomena needing further investigation, which SELEX engineers are currently working on. In the following, the outlined steps are detailed.

3.2. Identification of processes

Referring to the development model phases, we jointly determined three key processes of interest from the quality and effort perspective: the CSCI *implementation* process, the CSCI *testing* process, and the *defect fixing* (i.e., pre-release corrective maintenance) process (hereafter simply *implementation, testing, fixing*). Implementation and fixing involve several different groups, with a high degree of heterogeneity and a lack of direct control by the integrator; therefore, they are potentially subject to a high variability in terms of quality and effort. Testing is monitored because of its impact on what provided by suppliers: the quality of the delivered CSCI, as well as the release time and cost, depend heavily on the work of the testing team. The object of the evaluation is therefore the bottom part of the lifecycle in Figure 1, consisting of the iterative cycle where artefacts are implemented, tested, and undergo corrective maintenance actions. Actors involved are the CSCI supplier (for implementation and fixing) and the integrator's testing teams responsible for CSCI qualification testing. For these processes, defect data are available through the bug tracker shared between the testing team and suppliers. With these processes identified, the objective is re-stated into three research questions:

- **RQ1:** What are the *effectiveness* and the *efficiency* of the implementation process?
  As we focus on quality evaluation, with effectiveness we mean the quality – i.e., (un-)defectiveness – of what produced, while with efficiency we mean the quality with respect to the effort to attain it.
- **RQ2:** What are the *effectiveness* and *efficiency* of the testing process?
  The task of tester is to expose failures, i.e., the defects manifestation. Hence, testing effectiveness is the level of defectiveness exposed by testers, and efficiency relates this to the effort required.
- **RQ3:** What are the *effectiveness*, the *efficiency*, and the *internal quality* of the fixing process?
  These are intended, respectively, as: the extent of defectiveness reduction performed by the fixing team; the latter related to the effort required; the quality attributes of the fixing process, taken as indirect guarantee of a correct fixing, such as: the continuity of the fixing action over

DOI: 10.1002/smr

time, the homogeneity of the fixing actions across defects, the distribution across priority, severity, reproducibility categories.

These questions are answered through the analysis of defect data from two perspectives: *i)* analysis of the average effectiveness, efficiency, and internal quality measures for CSCIs; *ii)* analysis of the variability of these measures across CSCIs. The former provides synthetic indications on the overall process. The latter indicates the degree of control over the monitored process (with high variability indicating low control) pointing out the most critical CSCIs.

3.3. Metrics Definition

The following metrics are defined to target the outlined questions. The first set, called *basic metrics*, is meant to characterize the *defectiveness*, used as indicator of poor quality. For CSCI j, we have:

- *Defects*$_j$: number of opened defects;
- *EstDefects*$_j$ and *EstResiduals*$_j$ = *EstDefects*$_j$ - *Defects*$_j$; *number of estimated defects*, and *number of estimated residual defects*, obtained by the cumulative defect count curve. *EstDefects*$_j$ is the fundamental metric to measure the expected quality. To obtain it, we used software reliability growth models (SRGM) built on the defect count curve. SRGMs are a widely-used class of models to fit inter-failure times from test data, in order to estimate the next time to failure based on the observed trend. They are used to predict the number of (residual) defects, and the testing time needed to detect them (useful for scheduling the best time to release), to assess or predict reliability, and to allocate testing effort to system's components as well (e.g., [8], [9], [10], [11]). These models have been proven to work well even when the assumptions they ground on are partially violated [12]. SRGMs are characterized by a mean-value function (MVF) representing the cumulative number of expected defects detected at time t with respect to the testing effort (e.g., in terms of number of testing days). Different MVFs give rise to different SRGMs, as the ones adopted here. We used a subset of the most common class of non-homogeneous Poisson process (NHPP) models: *Exponential* [13], *S-Shaped* [14], *Weibull* [15], *Log Logistic* [16], *Log Normal* [17], *Truncated Logistic* [18], *Truncated Extreme-Value Max* [19], and *Truncated Extreme-Value Min* [19]. Since there is no model that fits all the situations, we applied all of them to each set of CSCI data, and chose the best one based on their Akaike Information Criteria (AIC) value, as in [20], [21].
- *Defects*$_{P_{i,j}}$: defects per priority class P_i; *Defects*$_{S_{i,j}}$: defects per severity class S_i; *Defects*$_{R_{i,j}}$: defects per reproducibility class R_i. These metrics are for a finer-grain analysis.

To characterize the *Effort* spent into an activity we use the measure of *man-weeks*. Depending on the target process, it refers either to the implementation or to the testing effort. We distinguish *ImplEffort*$_j$ and *TestEffort*$_j$, respectively, while we use *EFFORT*$_j$ to refer to one of both when there is no ambiguity. For fixing, the effort is measured by the total time required to fix all the defects. In fact, while the implementation effort is a piece of information shared between the supplier and SELEX ES, which commissions the implementation, the effort for fixing, e.g., in terms of man-months, was kept reserved by suppliers. Hence, we use the time to fix as approximation.

3.3.1. Implementation Metrics

- **Effectiveness**. *Estimated expected quality*, *EstQuality*: $EQ_j = \frac{Size_j}{EstDefects_j}$, where $Size_j$ is the code size measured as total number of thousands source lines of code (*KLoC*). The higher the value, the higher the expected CSCI quality. Note that this is the inverse of the *estimated expected defect density* after the implementation, defined as: *EstDensity*: $ED_j = \frac{EstDefects_j}{Size_j}$.
- **Efficiency**. *Quality-aware Productivity*: $QP_j = \frac{Size_j}{ImplEffort_j} \cdot \frac{1}{ED_j+1}$; this is derived from the the traditional form of *productivity* as size over effort, $P_j = \frac{Size_j}{ImplEffort_j}$, adjusted by a *quality factor* represented by the second term, varying between 0 and 1. The metric is defined because the plain productivity only accounts for how much has been produced, without any regard to

DOI: 10.1002/smr

quality (which we are interested in). For instance, spending 10 man-weeks for one KLoC containing 5 defects is not the same as 10 man-weeks per KLoC with 8 defects. Thus, we penalize the productivity as defect density increases, ideally achieving 0 under infinite defects, while we preserve the plain productivity under a perfect zero-defects product. The higher the value, the higher the actual productivity is, accounting for both the raw productivity and the quality of what produced.

3.3.2. Testing Metrics

Metrics to assess the defect detection phase are based on the number of found defects (*Defects*$_j$), on the estimate of expected defects, *EstDefects*$_j$, and on the testing effort. Unlike implementation, where CSCIs are at the same stage (namely, CSCI delivered to the testing team), the testing process of each CSCI may be at different stages at the time of the inquiry. Hence, the number of found defects is only a partial indication, as at early testing stage more defects are exposed. We consider:

- **Effectiveness**. *Defect detection state*, $DetState_j\% = \frac{Defects_j}{EstDefects_j} \cdot$ *100*, measured as the percentage of defects found over all the defects expected. It measures the current state of the testing process (its "maturity") with respect to the expectation as estimated by the SRGM.
- (Absolute) **Efficiency**. *Defect detection rate*, $DetRate_j = \frac{Defects_j}{TestEffort_j}$, measured as number of found defects over man-weeks of testing that have been spent.
- *Percentage detection efficiency* = $DE_j\% = \frac{DetState_j\%}{TestEffort_j} = \frac{DetRate_j}{EstDefects_j} \cdot 100$. It measures the *defect detection state* relatively to the testing effort spent so far, thus indicating *what percentage of defects, with respect to the expected total, has been found per man-week of testing*. It also corresponds to the normalized detection rate over expected defects.

Both detection rate and *DE%* indicate the actual efficiency of testing, but they do not allow a fair comparison of CSCIs, because the number of defects does not vary linearly with testing effort. Hence, two detection rates or *DE%* values at different stages are not comparable to each other (in the beginning, the process is expected to be more efficient, as it exposes more failures per time unit). For a better comparison, we use a "relative" efficiency measure: given the same detection state, which testing process exhibited the best (normalized) rate. For instance, considering a maturity of 90%, we compare the efficiency of testing teams in achieving that level. We can define the *number of man-weeks to detect the x% of total estimated defects* (*TestEffort*$_{x\%}$). This represents a relative inefficiency measure (i.e., the higher the value, the lower the efficiency). We define its normalization over the number of estimated defects (to account for differences among CSCIs) as:

- *Relative detection effort*: $RDE_{x\%,j} = \frac{TestEffort_{x\%}}{EstDefects_j}$

3.3.3. Fixing Metrics

- **Effectiveness**: *%Closed*$_j$ = *#Closed*$_j$*/#Opened*$_j$ defects for CSCI j, indicating how many defects are closed with respect to the opened ones.
- **Efficiency**: we consider the *FixRate* = *1/*$TTFix$, where $TTFix$ is the average *time to fix* a defect. We consider both the mean and the median time to fix, i.e., $TTFix_{\mu,Med}$, obtaining: *FixRate*$_{\mu,Med}$. The $TTFix$ is calculated in days as the time the resolution status is set to *fixed* minus the time when it is set to *open*. This excludes the time during which the manager verifies the issue and then closes it, to better approximate the actual time in which the developer works on the bug fix.

As for the **Internal Fixing Process Quality**, the following metrics provide insights on how the fix process is conducted. They are based on the *time to fix* evolution and distribution, and are therefore better understood by a visual analysis of graphs, as showed in the next Section.

Process continuity:

Consider NOpen$_j$(t) and NClosed$_j$(t) being two time series representing, respectively, the number of cumulative defects that are opened and the number of cumulative defects that are closed over

DOI: 10.1002/smr

time, for CSCI j (time is discrete and, for these two series, the time unit is the day). Let us derive, from them, the following two time series:
FixDiff$_j$(t) = *NOpen(t)*$_j$*(t) - NClose(t)*$_j$*(t)*: difference between opened and closed defects at a given time t (with time unit being the day).
FixTrend$_j$(t′) = number of defects closed in a time unit. For this metric, the time unit is the week (hence $t' = t/7$), since the day granularity hides relevant variations. More formally, *FixTrend*$_j$*(t′)* = 0 if $t' = 0$; *FixTrend*$_j$*(t′)* = *NClosed*$_j$*(t=t′* $\cdot$ *7)* - *NClosed*$_j$*(t=(t′*-1) $\cdot$ *7)* if $t' > 0$. Values of both these series are ideally desired to not change suddenly with time, since this would indicate that the defect closing trend is approximately the same as the opening one, and that there are no abrupt increases in the defect closing trend. Both aspects indicate a continuity in the process, intended as smoothness of the fixing action over time, which is a desirable feature to increase the chance of having good fixes [22]. We use their basic statistics as metrics to synthetically capture their characteristics:

- *FixDiff*$_{MIN_j}$, *FixDiff*$_{MAX_j}$, *FixDiff*$_{MEAN_j}$, *FixDiff*$_{STD_j}$, which are, respectively, the minimum, maximum, mean, and standard deviation, of the values of the *FixDiff*$_j$*(t)* time series.
- *FixTrend*$_{MIN_j}$, *FixTrend*$_{MAX_j}$, *FixTrend*$_{MEAN_j}$, *FixTrend*$_{STD_j}$, namely, the minimum, maximum, mean, and standard deviation of the *FixTrend*$_j$*(t′)* series.

Process homogeneity:
A homogeneous fixing process means that most of defects are resolved within a *TTFix* close to the average, and that most of the *TTFix* variance is due to many defects with short *TTFix*. The undesirable situation is to have a non-negligible number of defects over all the spectrum of the *TTFix*, highly distant from the mean. This characteristic is captured by the empirical *TTFix* distribution [22], on which we measure:

- *Kurtosis*$_j$, indicating the peakedness of the distribution. High Kurtosis is good, denoting that most of variance is due to few peaks. These peaks are of course desired in the left side of the distribution (i.e., many defects with a short *TTFix*); this is captured by skewness:
- *Skew*$_j$: a positive skew is desired, denoting a right-tailed distribution.

Fix per Type of Defect:
Finally, for a finer grain analysis, we consider the *TTFix* per defect category, in order to see if there are differences in fixing defects of different categories (e.g., high severity with respect to low severity): *TTFixP*$_{i,j}$: average time to fix a defect per priority class i; this indicates if the debugging team conforms to the indication of reporters about which bug is more urgent to fix;
TTFixS$_{i,j}$: average time to fix a defect per severity class i; this indicates if the level of severity if a bug is correlated with the time to fix it;
TTFixR$_{i,j}$: average time to fix a defect per reproducibility class i; it indicates if defects marked as *not always reproducible* are actually more difficult to fix or not, and if the indication is useful to prioritize defect fixing. Table I summarizes all the presented metrics.

3.4. Data Collection

The process evaluation approach is applied to a set of 6 CSCI developed in the context of the homeland security domain (specifically in the *naval and maritime* systems domain), in charge of managing the port, maritime, and coastal surveillance. In order to find a suitable way to assess the quality of such products and of the corresponding process, we started the mentioned collaboration among the University "Federico II", SELEX ES and SESM at the beginning of 2010 up to august 2012. Thus, the selection of CSCIs that we have used as pilot project was on a temporal basis: we considered all the CSCIs under development in that period, hence data collected range from October 2009 to August 2012. The features of CSCIs are summarized in Table II - names of CSCIs are anonymized as $C1$, $C2$, ..., $C6$, for confidentiality reason. All the CSCIs are self-consistent products, and their design and development is completely independent form the specific system they will be integrated in. A CSCI may be even reused across several domains, e.g., air traffic control. The considered CSCIs have a large range of variation in terms of size (ranging from 22 KLoC to

DOI: 10.1002/smr

Table I. List of Metrics for CSCI j

Name	Description	Unit of Measure
Basic Metrics		
Defects	Level of defectiveness detected by testing	*Defect*
EstDefects	Expected level of total defectiveness	*Defect*
EstResiduals	Expected level of residual defectiveness	*Defect*
$Defects_{P_i,S_i,\ R_i}$	Level of defectiveness distinguished per class of priority, severity, or reproducibility	*Defect*
$ImplEffort_j$, $TestEffort_j$	Effort for implementation, and for detection	*Man-week*
Implementation Metrics		
EQ	Estimated expected quality	*KLoC/Defect*
ED	Estimated expected defect density	*Defect/KLoC*
P	Productivity as quantity over effort	*KLoC/Man-week*
QP	Quality-aware Productivity	*KLoC/Man-week*
Detection Metrics		
DetState %	Defect detection state as opened defects with respect to the expected total	*Percentage (%)*
DetRate	Defect detection rate	*Defect/man-week*
DE %	Percentage detection efficiency	*Percentage (%)/man-week*
$TestEffort_{x\%}$	Test effort to detect the $x\%$ of expected defects	*Man-week*
$RDE_{x\%}$	(Relative) detection effort to detect the $x\%$ of expected defects	*Man-week/Defect*
Fixing Metrics		
%Closed	Level of defectiveness fixed	*Percentage (%)*
$FixRate_{\mu,Med}$	Fix rate	Day^{-1}
$TTFix_{\mu,Med}$	Average time to fix	*Day*
$FixDiff_{MIN}$, $FixDiff_{MAX}$ $FixDiff_{MEAN}$, $FixDiff_{STD}$	Minimum, maximum, mean, and standard deviation of the *FixDiff* time series	*Defect*
$FixTrend_{MIN}$, $FixTrend_{MAX}$ $FixTrend_{MEAN}$, $FixTrend_{STD}$	Minimum, maximum, mean, and standard deviation of the *FixTrend* time series	*Defect/week*
Kurtosis	Kurtosis of the TTFix distribution (coefficient of kurtosis)	-
Skew	Skewness of the TTFix distribution (coefficient of skewness)	-
$TTFixP_i$, S_i, R_i	TTFix per priority, severity, reproducibility class	*Day*

Table II. Features of the analysed CSCIs

CSCI	KLoC	Implementation Effort (man-weeks)	Description
C1	39.059	44	CSCI assuring interoperability between operative-strategic and tactical systems for Network-Centric Operations support during expeditionary warfare operations.
C2	55.154	24	CSCI in charge of managing anomalies, alarms and smart agents associated with maritime tracks
C3	22.208	40	CSCI in charge of managing presentation layer components
C4	59.535	48	CSCI managing standard-compliant messages and related standard communication protocols
C5	343.719	144	CSCI responsible for validation of messages of application-level protocols, and their correct sending and receiving
C6	34.700	42	CSCI managing the representation and publication of messages through a shared message channel model

343 KLoC), and of implementation effort (ranging from 24 to 144 man-weeks). They have different missions and offer several functionalities, ranging from low-level drivers to Web applications and GUIs.

The CSCI C1 has the primary goal of assuring interoperability during expeditionary warfare operations among different systems. The involved systems are the "operative-strategic system" and the "tactical system for the support of network-centric operations", which use different protocols and heterogeneous channels; the CSCI assures the correct communication among them. C2 is a CSCI responsible for controlling the maritime traffic. It manages, and presents to the user, anomalies

DOI: 10.1002/smr

and alarms that might be associated with maritime tracks. C3 is a GWT framework to manage components for the presentation layer. C4 is a CSCI in charge of managing and presenting messages set up according to various different standards and which utilize different communication protocols. C5 is a CSCI for verification and validation of messages belonging to application-level protocols. It is in charge of managing the correct representation of the messages, and the correct sending and receiving of messages according to a given protocol. Finally, C6 is in charge of managing the representation and correct publication and retrieval of messages implemented through shared models for message exchange.

All the CSCIs have been designed by SELEX ES teams, provided to different companies for implementation, integrated and tested by SELEX ES supported by SESM. In particular, the latter supports SELEX ES for software verification and validation (V&V) activities of these CSCIs. Defects management is performed through the shared *Mantis* bug tracker, from which we extracted data. The resulting set of defects is of **1,296** opened issues, of which **950** (i.e., the **73.3**%) were closed. We included in the analysis all the issues except the ones marked as *duplicate* and request of new features.

4. RESULTS

This Section reports the result of the analysis of each monitored process. Table III reports the basic metrics, as derived by the cumulative defect count graph (Figures 4(a)-4(f)) by means of the above-mentioned SRGMs. The estimation of the expected number of total defects (*EstDefects*) provided by SRGMs is in the second row. In fact, the different types of SRGMs can be described by their mean value function appearing in the form $m(t) = aF(t)$, where a is the expected number of total defects (corresponding to *EstDefects*), and $F(t)$ is a distribution function, which can take several forms depending on the fault detection process. Parameters of SRGM (both a and parameters of $F(t)$) are estimated by adopting the expectation-maximization (EM) algorithm as described in [20]. In the analysed cases, the best fitting models, according to the mentioned AIC value, have been the *truncated logistic* ($m(t) = a \cdot \frac{(1-exp(-t/\kappa)}{(1+exp(-(t-\lambda)/\kappa)}$) in four out of six cases, and the *truncated extreme-value min* ($m(t) = a \cdot (1 - \frac{1-exp(-exp(\frac{-(t-\lambda)}{\kappa}))}{1-exp(-exp(\lambda/\kappa))}$) in the remaining 2 cases. The adjusted coefficients of determination are $\bar{R}^2 > 0.96$ in all the cases but C4, where it is $\bar{R}^2 \approx 0.93$. From graphs, a remarkable difference is between C4 and all the other CSCIs, having the former still a high detection rate in the final part of the curve, that causes an estimate of residual defects significantly higher than the others (this is better investigated in the testing process analysis). Overall, according to the estimate, there are still 1,476 - 1,296 = 180 defects to detect, with pronounced differences among CSCIs (the standard deviation of residuals is very high due to C4 and C6).

Table III. Values of Basic Metrics for each CSCI

Basic Metrics	C1	C2	C3	C4	C5	C6	Mean.	Tot	Standard Deviation
$Defects_j$	206	166	283	152	177	312	**216**	**1,296**	**66.21**
$EstDefects_j$	216	168	288	287	179	338	**246**	**1,476**	**68.38**
$EstResiduals_j$	10	2	5	135	2	26	**30**	**180**	**52.21**

4.1. Implementation Process

Both the trend of the graphs and the number of detected defects reflect more the different stages of testing than the quality of the implementation. This is better described by the expected defectiveness as estimated by SRGMs. Table IV reports the implementation metric values. The expected quality is in the average 0.469 KLoC per defect (corresponding to a defect density of 6.1 defects per KLoC). It is not as high as desired, but it should be considered that the value refers to defectiveness at the end of the implementation process, not as usually taken (i.e., accounting for number of post-release defects); the CSCIs must still undergo testing and pre-release corrective maintenance. As such, the

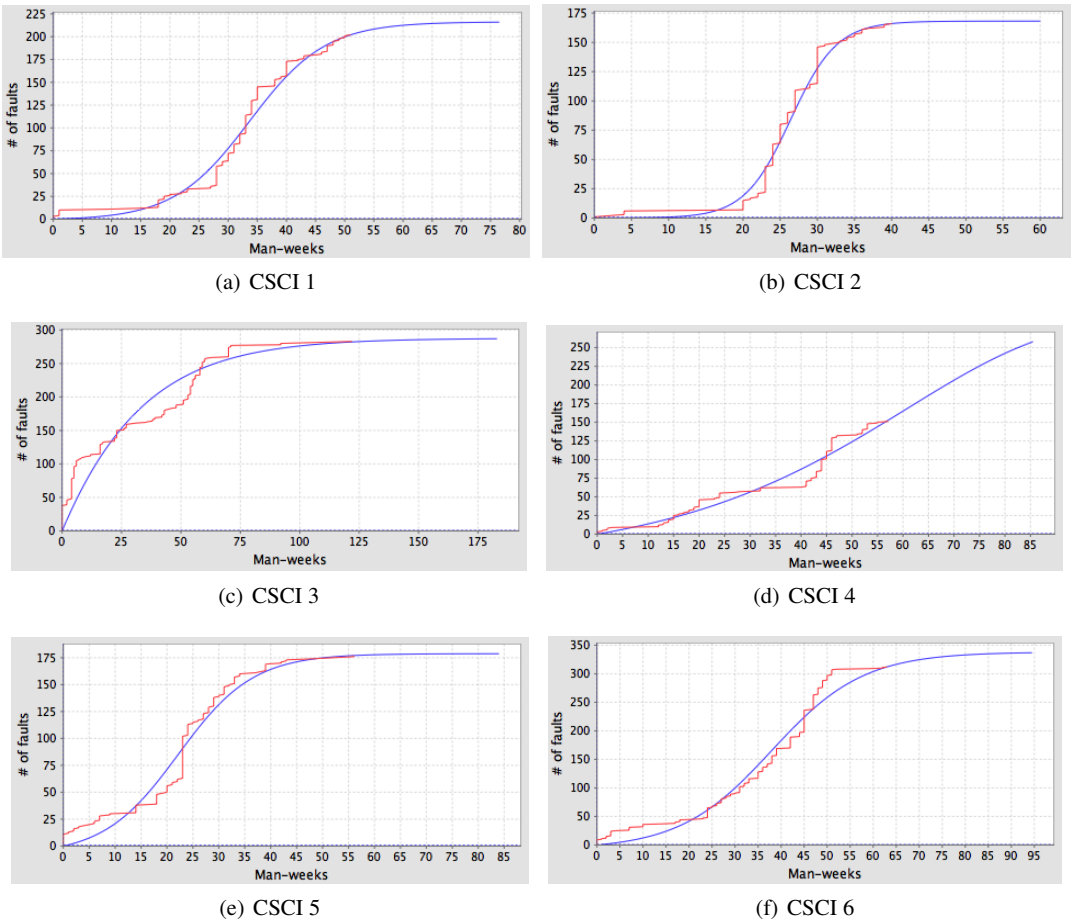
DOI: 10.1002/smr

measure is to evaluate just the implementation stage, not the CSCI as finally released. The average

Table IV. Values of Implementation Metrics for each CSCI

	C1	C2	C3	C4	C5	C6	Mean	Standard Deviation
ED_j	5.530	3.046	12.968	4.820	0.520	9.567	**6.104**	**4.535**
EQ_j	0.181	0.328	0.077	0.207	1.920	0.103	**0.469**	**0.716**
P_j	0.887	2.298	0.555	1.240	2.38	0.826	**1.365**	**0.787**
QP_j	0.136	0.568	0.039	0.213	1.569	0.077	**0.434**	**0.588**

productivity is 1.36 KLoC per man-week. The last row reports the quality-aware productivity, indicating the productivity adjusted by the quality factor. In the average, this factor causes a reduction, with respect to plain productivity, of almost 70%, leading to a quality-aware productivity of 0.434. Focusing on single values, we notice some cases significantly different from the mean, which cause a high standard deviation (compared to the mean) especially in the estimated expected quality (EQ) and quality-aware productivity (QP) values. We observe two extreme situations that determine most of the variability: C5 with the highest expected quality and the highest productivity, and C3, with the opposite result. Correspondingly, these CSCIs also experience the best (C5) and the worst (C3) quality-aware productivity, as well as the smallest (C5) and the biggest (C3) reduction from productivity to quality-aware productivity (of about 34% and 92%, respectively). C6 indicators report a bad performance too, very close to C3.



(a) CSCI 1 (b) CSCI 2

(c) CSCI 3 (d) CSCI 4

(e) CSCI 5 (f) CSCI 6

Figure 4. Cumulative number of detected defects for each CSCI

DOI: 10.1002/smr

It is interesting to observe that the supplier producing more (C5's supplier) is also the one producing with higher quality, and the opposite is also true for C3. This suggests that there may be a systematic reason for the observed differences, and the results concordance between the expected quality and quality-aware productivity is not by chance. From the implementation quality point of view, engineers are called to investigate the single processes implemented by the best and the worst case, trying to foster the replication of best practices across all the suppliers, with the double objective of: improving the mean values of expected quality and quality-aware productivity, and having more homogeneous results (i.e., lowering the standard deviation). Further indications come from the following fine-grain analysis and the final discussion targeting the processes altogether.

4.2. Testing Process

Table V reports the testing process analysis results. The *detection state* represents the current *effectiveness* of testing, while the detection rate (*DetRate*) and percentage detection efficiency ($DE\%$) value are the current *efficiency* measures. In the average, testers have detected about 3.7 defects per man-week of effort, detecting the 89.43% of the total expected defects. The standard deviation (STD) values of all the metrics (compared to their mean) is relatively low compared to implementation. This is explained by considering that testing is carried out entirely in SELEX ES, with lower heterogeneity with respect to implementation teams.

Table V. Values of Testing Metrics for each CSCI. Bold texts are the main effectiveness and efficiency metrics

	C1	**C2**	**C3**	**C4**	**C5**	**C6**	**Mean**	**STD**
$TestEffort_j$	52	40	123	57	56	63	*65.16*	*31.93*
DetState$_j\%$	**95.37%**	**98.81%**	**98.26%**	**52.96%**	**98.88%**	**92.31%**	**89.43%**	**18.05%**
DetRate$_j$	**3.96**	**5.19**	**2.26**	**2.98**	**3.05**	**4.73**	**3.69**	**1.03**
DE$_j\%$	**1.83%**	**2.47%**	**0.80%**	**0.93%**	**1.77%**	**1.47%**	**1.54%**	**0.62%**
$TestEffort_{50\%,j}$	33	26	22	53	23	39	*32.67*	*11.87*
$RDE_{50\%,j}$	3.27	3.23	6.54	2.70	3.89	4.33	*3.99*	*1.37*
$TestEffort_{90\%,j}$	49	34	61	85	37	52	*53*	*18.55*
$RDE_{90\%,j}$	3.97	4.45	4.25	3.04	4.35	5.85	*4.32*	*0.91*

Looking at CSCI values, data highlight that all of them but C4 have detected over 90% of expected defects, denoting that testing is in an advanced state; C4 is instead at 53%. To figure out if the low value is due to bad testing or to the testing process being at its early stage, we have to turn out to efficiency metrics. We notice that the C4 detection rate is almost the worst one, although we would expect a higher rate than the others. Typically, at later stages of testing, the number of defects found in a time unit gets decreasing, meaning that few defects are remaining and the work of tester is much harder (as a lot of effort is needed to detect further defects). Instead, C4 has approximately the same rate as C5, which has however detected almost all the defects (i.e., 98.8%). It means that the effort devoted to C4 testing has been spent badly so far. Contrarily, C2 and C5 have a very high detection rate if we consider that they are close to the end of testing (over 98%).

For a clearer picture, the fourth row reports the detection efficiency metric. In the average, the detection process has revealed the 1.54% of total expected defects per man-week. Focusing on the single values, we note that, for C2, the percentage detection efficiency (*DE%*) is 2.47% (best value), while C3 and C4 have been the most problematic ones. At a closer look, while C3 has achieved roughly the same state as C2 (98%), C4 has detected only the 53% of total defects, as discussed earlier. Since the detection rates do not increase linearly with testing effort, the similar *DE%* values of C3 and C4 do not indicate a similar situation, because they are at very different stages; C4 has more serious problems.

The percentage detection efficiency, *DE%*, provides the efficiency at the current time; therefore, for a fair comparison, we can use the *relative* detection effort metrics. We consider the *relative detection effort* at two maturity levels: 50% and 90 % of total defects as reference detection state, representing early and late testing stages ($RDE_{50}\%$ and $RDE_{90}\%$). The former metric is based on the actual man-weeks employed for each CSCI to detect 50% of defects; however, for the latter

DOI: 10.1002/smr

metric ($RDE_{90}\%$) there is one case in which the percentage of detected defects does not achieve the 90% yet, which is C4 (it is at 53%). Hence, for C4, we use the prediction of the man-weeks required to achieve the 90% of defects, obtained by the corresponding SRGM on the base of data collected up to 53%. Despite the goodness of the prediction (the goodness-of-fit measure of the SRGM for C4 is $\bar{R}^2 \approx 0.93$ over 152 data points), this is a source of uncertainty that can in principle change the result. On the other hand, the C4 value of $RDE_{90}\%$ is consistent with that of $RDE_{50}\%$ (it is again the worst one) and with all the effectiveness and efficiency metrics discussed above: the value just confirms that C4 testing team exhibited the lowest detection power.

Analysing the effectiveness and efficiency indicators altogether, we note that, contrarily to the expectation, the detection rate of all CSCIs except C3 increases with testing effort; namely, the *relative detection effort at 90%* values ($RDE_{90}\%$) are greater than the corresponding ones at 50%, and the variation decreases. This is, in general, unexpected, as we would like to expose most of defects as early as possible. At 50% of the work, C3 and C6 have the best detection rate. At 90%, C6 remains the best one, while C3 rate decreases, lower than C2 and C5, even though still high. After 90% (i.e., from 90% to the current state), comparing the detection rate (*DetRate*) with $RDE_{90\%}$ we observe that: *i)* C1 performance decreases very slightly; *ii)* C5 and C6 decrease considerably (more than 1 defect per man-week); *iii)* the biggest decrease is for C3, whose rate drops down of about 2 defects per man-week; *iv)* C2 rate, contrarily, keeps increasing. This means that the knee point, where defects detection starts decreasing, is after the 90% for C1, C4, C5, and C6, while it is between the 50% and 90% for C3. While this is a good behaviour for C3 *early* testing, the final actual efficiency depends on the release criteria: if it is at 90 % of the total estimated defects, C3 testing team did a good job (the rate is close to the average); if it is higher, for instance at 98%, C2 and C5 are better than C3, despite an early testing not as good as C3. Considering that all the CSCIs but C4 have overcome the 90%, and taking this threshold as optimal release criterion, the best testing process has been observed in C6, while the worst one is C4.

4.3. Fixing Process

We analyse the results of fixing effectiveness and efficiency. The sample size is of 950 defects, since it accounts only for closed defects. As overall indication, we look at the percentage of closed defects with respect to the opened ones, and the mean and median *fix rates* (with the corresponding *time to fix*). Closed defects are 73.3%, with a mean *fix rate* of 0.0323 defects per day (mean *time to fix* a defect of 30.92 days), and a median of 0.0841 defects per day (mean *time to fix* of 11.89 days). The difference between mean and median is high because of the strong non-normality of the distribution (the *Shapiro-Wilk* test *p-value* is:$< 10^{-5}$).

The high standard deviation of the time to fix (54.03) leads to investigatethe differences at CSCI level. Table VI reports the results per CSCI. C1 is clearly an outlier, having much fewer closed defects and lower fix rate. The best fix rate is for C2 (0.07 and 0.16 defects per day, corresponding to 12.65 and 5.93 days to close a defect, for mean and median indicators, respectively), even though the defects to close at the time of the inquiry are only 63.25% of opened ones. Looking at the trade-off between effectiveness and efficiency, we notice that C3 and C5 have good performance in terms of percentage of closed defects (both over the 90%) and of fix rate mean and median. Standing with these indications, engineers should target C1 to lower the variability among results and improve the overall average indicators. In the following, we look at the process continuity and homogeneity of each CSCI for a deeper understanding of effectiveness and efficiency measures.

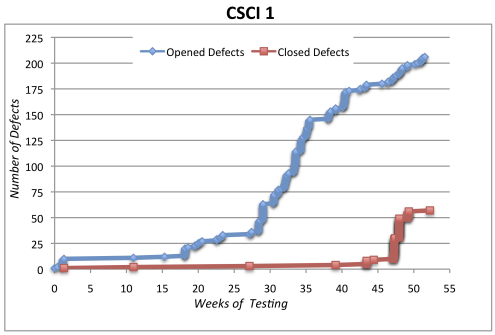
Process Continuity and Homogeneity

Process continuity is inferred from the time series of opened and closed cumulative defects (Figure 5(a)-5(f)). Table VII reports the continuity metrics of *closed-opened difference* (*FixDiff* metrics) and *closing trend* (*FixTrend* metrics). The former metrics indicate how the closing curve follows the opening one (i.e., suppliers promptly fix defects), the latter ones indicate if the increase is smooth or not. They provide insights on the evolution of corrective maintenance process over time.

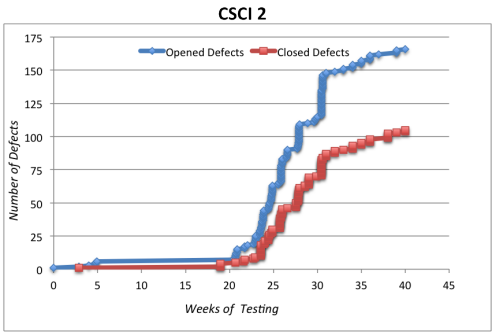
DOI: 10.1002/smr

Table VI. Time to Fix per CSCI

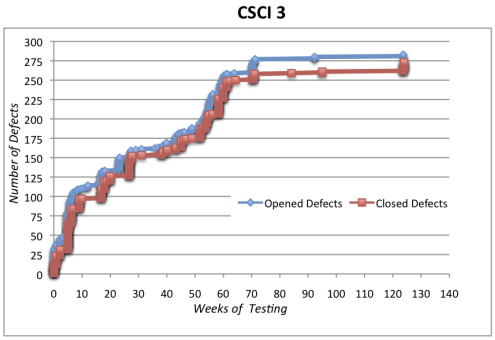
CSCI	*C1*	*C2*	*C3*	*C4*	*C5*	*C6*	(Inter-CSCI) Mean (STD)
Sample Size	57	105	273	129	160	226	*158.33 (79.62)*
% Closed	27.67%	63.25%	96.47%	84.87%	90.39%	72.43%	*72.51% (25.07%)*
FixRate$_{\mu}$ ($TTFix_{\mu}$)	0.0099 (100.34)	0.0790 (12.65)	0.0404 (24.74)	0.0183 (54.42)	0.0489 (20.43)	0.0427 (23.40)	*0.0399 (0.0244)* *39.33 (33.12)*
FixRate$_{Med}$ ($TTFix_{Med}$)	0,0095 (105.24)	0,1686 (5.93)	0,1563 (6.40)	0,0294 (34.00)	0,0901 (11.10)	0,0714 (14.00)	*0,0876 (0,0649)* *29.44 (38.53)*



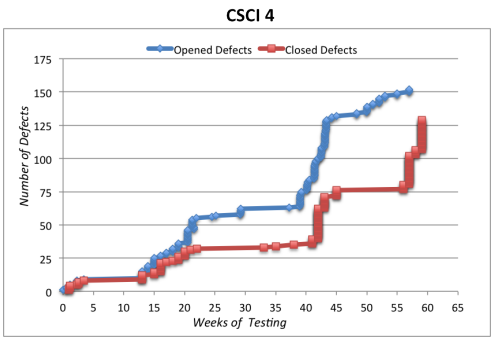
(a) Trend for CSCI 1



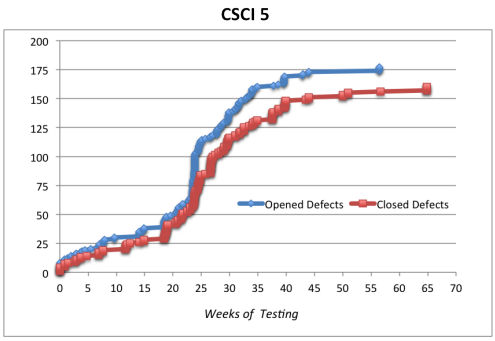
(b) Trend for CSCI 2



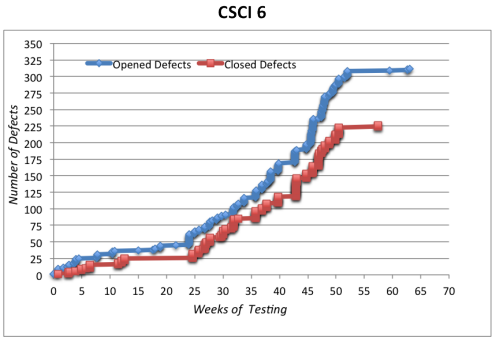
(c) Trend for CSCI 3



(d) Trend for CSCI 4



(e) Trend for CSCI 5



(f) Trend for CSCI 6

Figure 5. Fixing Process Continuity: Opening and Closing Curves

DOI: 10.1002/smr

Table VII. Values of Fix Process Continuity Metrics for each CSCI

	FixDiff $_j$ Metrics				***FixTrend*** $_j$ Metrics			
CSCI	*MIN*	*MAX*	*MEAN*	*STD*	*MIN*	*MAX*	*MEAN*	*STD*
CSCI 1	1	175	70.33	65.22	0	40	1.07	5.56
CSCI 2	1	61	14.12	17.06	0	35	3.28	7.06
CSCI 3	6	46	15.80	5.25	0	25	2.12	4.88
CSCI 4	1	73	27.53	20.79	0	24	2.52	5.73
CSCI 5	4	33	17.10	7.83	0	15	2.4	3.72
CSCI 6	1	86	40.23	27.61	0	22	3.53	5.09

We observe that:

- C1 has always a high number of non-closed (pending) defects (with a maximum of 175 around week 45). Its bad performance, already verified by the final low effectiveness, is stable along the entire period of observation. We see that the opening trend increases while the closing trend remains approximately constant (Figure 5(a)).
- For C6, despite both the curves increase quite smoothly, numbers highlight some problems, since, toward the end of testing, there is a maximum of 80 defects left open. Contrarily to C1, this has more likely been a temporary problem, as the number of pending defects increased only after week 40 (Figure 5(f)). Besides this effectiveness problem, the *closing trend* standard deviation (*FixTrend*$_{STD_j}$) is high compared to the mean, indicating a high variability in the fixing time, despite the graph appearance (the high number of defects make it appear as smooth).
- C2 has a good performance (the mean *closing-opening difference*, *FixDiff* $_{MEAN_j}$, is 14.12), although the final pending defects are 100%-63.25%=36.75%. A summary conclusion could be that the team fixes few defects but in a short time (the *time to fix* median is the lowest one, 5.93). Looking at the *closing trend*, the high standard deviation indicates an abrupt increase of fixing. This, in general, is not a desirable behaviour, as it might indicate that many fixes are performed altogether and can have a negative impact on the fixing process quality [22], with high likelihood of introducing new defects. However, only considering these values is inconclusive: if we look together at the good *closing-opening difference* mean (*FixDiff* $_{MEAN_j}$) and consider the trends of both curves, we see that the there is an abrupt increase of both opening and fixing graphs, and that the final moderate effectiveness is more likely due to the sudden increase of defect defection than to slow fixing. All the other indicators show that C2 fixing team is able to keep a very high *fix rate* and a good average effectiveness.
- In C4, the maximum *closing-opening difference* is *FixDiff* $_{MAX_j} = 73$ and the standard deviation is high. The corresponding graph shows a non-smooth increase of the opening and closing curves, and an irregular trend with respect to the other CSCIs. In this case, unlike C2, the high standard deviation of the *closing trend* (*FixTrend*$_{STD_j}$) along with the values of *FixDiff* $_j$ metrics confirm that the discontinuity is due to both detection and fixing actions. Both behaviours are not desirable and need investigation.

Further considerations on homogeneity are made looking at the *time to fix* distribution. Figure 6 reports the distribution over the entire dataset, which shows good properties (high positive *skewness*: $\gamma = 8.5872$, and high *kurtosis*: $\kappa = 97.4564$), with about the 80% of defects fixed within 50 days. There are some exceptional cases requiring more than 150 days.

Table VIII reports the kurtosis and skewness values of each CSCI, highlighting that:

- C2 and C3 are confirmed to be the best ones; the high kurtosis and skew, along with the low mean and median *time to fix*, indicate that most of defects are fixed in very short time.
- The worst ones, for these indicators, are C4 and C6. While for C4 this is in line with the discontinuity observed before and with the high *time to fix* (mean= 54 days; median = 34 days), the values for C6 call for deeper analysis. In fact, although the graph with so many points hides the variability, both the previous analysis and these indexes suggest that there are not

DOI: 10.1002/smr

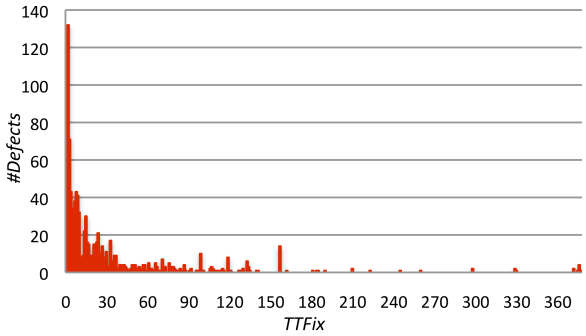


Figure 6. Distribution of the Time to Fix (in days) for all data

few peaks deviating from the mean. Overall mean and median are good, but the homogeneity of fixing actions should be improved for more consistent and controlled measures.

- C1 does not show the lowest kurtosis and skew values: this means that the bad results observed before are not due to some outliers that influence the mean, but most of the samples are around the high mean: the *time to fix* is systematically bad, around 100 days. The positive skew is explained by the initial phase of testing, where the opening and closing curves are close (due to the very few opened defects).

Table VIII. Time to Fix Distribution Kurtosis and Skewness

CSCI	C1	C2	C3	C4	C5	C6	Mean	Standard deviation
Kurtosis	66.8	100.7	78.7	18.1	58.2	10.4	**55.5**	**35.04**
Skew	6.9	9.5	8.0	4.0	6.6	3.1	**6.4**	**2.4**

In the following, we report a finer grain analysis, referred to the three processes, aimed to highlight aspects not captured by the previous metrics.

4.4. Fine-grain Analysis

We investigate the impact of *Priority*, *Severity*, and *Reproducibility* assigned by reporters during testing. Percentages over the entire dataset reveal that: the *Normal* priority is the most assigned level (55.94%); 25.77% of defects are marked as *High* priority, whereas the very high priority categories (that is: *Immediate* and *Urgent*) are used in 14.89% of cases. As for perceived severity: *Major* and *Minor* categories are used in 52.08% and 27.08% of cases, respectively; *Block* category is assigned in 9.34% of cases, which is not a negligible proportion. The remaining categories are used in less than 2% of cases, except the *Feature* category (4.40%). Table IX provides a combined view of the two features. Defects at high severity (such as the with category *Block* and *Crash*) have almost always an *Immediate, Urgent*, or *High* priority; thus the most critical defects are the ones at the right-bottom of the Table, which overall account for 474 defects (36.57%) considering *Major, Crash and Block* severity categories along with *High, Urgent, and Immediate* priority categories.

Histograms in Figure 7-8-9 show the defect count per priority, severity, and reproducibility for each CSCI. The priority attribute regards more the fixing process (it is an indication that should be used by the fixing team to prioritize defects) and will be investigated later on; whereas, for implementation and testing teams, we focus on severity and reproducibility. As for severity, we first conjecture whether the CSCI implementer impacts the observed number of high severity defects. To simplify the reasoning, we consider two classes of severity: *Low* severity, from *Trivial* to *Minor*, and *High* severity including *Major*, *crash*, and *Block* categories. We test the null hypothesis that: *the proportion of high severity defects is the same across all the CSCI*, which is rejected at $p < .05$ (the χ^2 test of independence is used for comparison among proportions). The pairwise comparisons (with *p-values* adjusted using the Benjamini-Hochberg procedure) reveal 8 out of the 15 significant

DOI: 10.1002/smr

Table IX. Defects Severity and Priority

Priority / Severity	*None*	*Low*	*Normal*	*High*	*Urgent*	*Immediate*	Total	Percentage
Feature	6	8	38	4	1	1	**57**	**4.40%**
Trivial	0	2	2	1	1	0	**6**	**0.46%**
Text	1	2	12	12	0	0	**27**	**2.08%**
Tweak	0	4	23	2	0	0	**29**	**2.24%**
Minor	1	13	305	21	7	3	**351**	**27.08%**
Major	0	6	331	270	47	21	**675**	**52.08%**
Crash	0	1	5	8	13	3	**30**	**2.31%**
Block	0	0	9	16	33	63	**121**	**9.34%**
Total	**8**	**36**	**725**	**334**	**102**	**91**	**1,296**	-
Percentage	**0.62%**	**2.78%**	**55.94%**	**25.77%**	**7.87%**	**7.02%**	-	**100%**

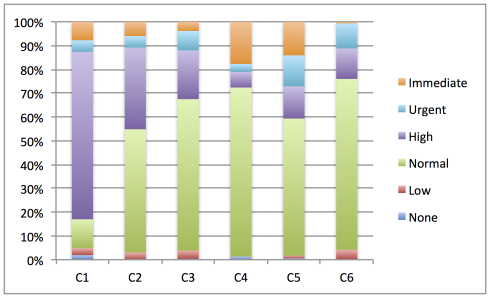


Figure 7. Defects percentage of *priority* categories per CSCI

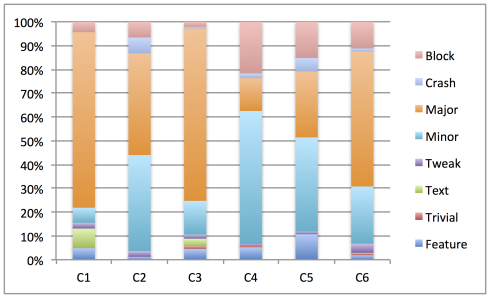


Figure 8. Defects percentage of *severity* categories per CSCI

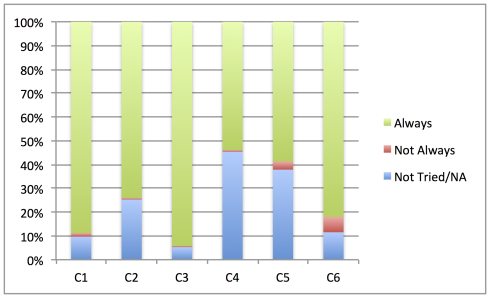


Figure 9. Defects percentage of *reproducibility* categories per CSCI

pairwise differences, the biggest one being between C1 (with the biggest high-severity defects proportion) and C4 (with the smallest) – *p-value* =.032. The implementation metrics in Table IV suggested that the most critical CSCI is C3; at the same time, this result suggests also putting some more focus on C1, since, even though it does not exhibit a high defect density as C3 (and as C6), it introduced the most severe defects. On the other hand, we also test whether severity, in turn, is related to testing performance metrics. We observe that there is no significant correlation between proportions of high severity defects and any of the testing metrics (coefficients are never over 0.6, and in no case statistically significant at $\alpha < .05$). If severity were related to testing metrics, then the evaluation of the testing teams could change: if a team receives a CSCI with more high-severe defects its worse performance is partly justified. However, we cannot state, from observing data, that differences in testing are due to defect severity.

Reproducibility (Figure 9) is a relevant attribute especially for the testing process. It indicates if testers experienced a situation in which he has not been able to reproduce the defect. In 78% of the cases, defects have been marked as *Always reproducible*, in 19% as *Not tried or Not Available* and in the remaining 3% as *Not always reproducible*. There is a high percentage of *Not tried or*

DOI: 10.1002/smr

Not Available cases, in which reporters did not care about this aspect or were not able to establish this value[§]. Considering that, in the literature, this type of defects are reported with percentages around 20-40%, even in critical contexts [23], [24], the low percentage can be justified by the scarce focus on establishing if a defect is always reproducible or not. Although the testing phase is not the final one, and more bugs of this type are expected to surface in later stages, our conjecture is confirmed by the high percentage of *Not tried or Not Available* cases. Considering their high impact at operational time [25], [26] this aspect should be improved. On single CSCIs, we observed again no significant correlation among the proportion of *Not Always Reproducible* defects and any of the testing metrics. Thus, differences among CSCI are likely not due to reproducibility.

We investigate the impact of *Priority* and *Severity* on the time to fix. Table X shows the median *time to fix* per priority for each CSCI (the corresponding count of defects is in Figure 7). The distributions of datasets are all non-normal (the *Shapiro-Wilk* test gives *p-value* < 0.0001 in all cases) - thus we use medians. We split data into *high* priority (including: *Urgent*, *Immediate*, *High* categories) and *low* priority (*Normal*, *Low*, *None* categories), and investigate the null hypothesis that *the time to fix for high and low priority is sampled from the same distribution*. The hypothesis is rejected with a confidence of 95% ($\alpha < .05$) in two cases, namely in C2 and C6. The hypothesis test adopted depends on the properties of data. In all the cases data are non-normal: when they are homoscedastic, we apply the Wilcoxon Mann-Withney U test. As it is sensitive to highly unequal variances [27], [28], [29], when data are heteroscedastic we transform data in ranks and use the *t-test* on ranks, with the version of *t-test* dependent on new variances on ranks: the classical *t-test* if variances are equal, the *Welch's t-test* otherwise [29], [28], [30]. Again, *p-values* are adjusted by the Benjamini-Hochberg procedure for multiple comparison protection.

From results, there is not a uniform indication: high priority defects take much longer than low priority ones in C2, while the opposite happens in C6. In the other CSCIs, the difference is not statistically relevant at 95%, but the same behaviour as C2 is observed in C1, C4 and C5. If we assume that the expected behaviour is the high priority defects being solved in shorter time, implementers of these CSCIs seem to neglect the priority indication as set by the testing team. Contrarily, C6 solves correctly the high priority defects earlier (as well as C3). This raises a warning about the way priorities are assigned or managed. Engineers are called to better investigate if this is due to: inappropriate assignments of priorities, to priorities being neglected by suppliers, or to the actual fixing difficulty of bugs at higher priorities. In order to gain deeper understanding of this, we analyse the difference, in the time to fix, between defects at different levels of severity. In fact, if we assume severity as an indirect indication of fixing difficulty, we can verify if differences are due more to the actual difficulty of fixing rather than to developers neglecting priorities.

Table X. Priority vs. Time to Fix Medians for each CSCI.

CSCI	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**
High	107.18	12.96	5.31	69.17	13.08	12.20
Low	86.88	1.02	7.22	30.98	11.07	15.26
Normality *Shapiro-Wilk test*	**<0.0001**	**<0.0001**	**<0.0001**	**<0.0001**	**<0.0001**	**<0.0001**
Unequal variance *Levene's test*	**0.0105**	0.0546	**0.0039**	**0.0007**	0.8153	**0.0164**
High vs. Low test *(adj. p-value)*	0.7675	**0.0004**	0.1817	0.9323	0.4436	**0.0415**

Table XI shows the *time to fix* medians for *high* and *low* severity. Like for priority, the tested hypothesis is whether *the time to fix for high and low severity is sampled from the same distribution*. In all the cases but C6, the high severity defects take longer than low severity ones. Among these, the significant differences rejecting the hypothesis are in C2 and C6, which have again opposite

[§]It should be observed that the *Always reproducible* defects are overestimated, since, in principle, the tester cannot state that a bug is "always reproducible" (he could have not tried enough); contrarily if tester experiences one case in which defect is not reproduced he can mark it as *Not always reproducible* with certainty

DOI: 10.1002/smr

results. The reason behind this should be investigated, to figure out if C6 team decided to neglect low severity defects and fix them later, or the severity indication by the testing team is inappropriate.

Observing the entire dataset, we noticed that: *i)* except for C3, there are very similar differences in the time to fix between *high vs low* priority and *high vs low* severity; *ii)* the 89.94% of defects (Table IX) at high priorities (i.e., *Urgent, Immediate, High* categories) are also at high severity (i.e.,*Major, Crash, Block* categories). These remarks are in favor of the conjecture that developers are not neglecting priorities, but that high priorities defects are actually the most severe ones and the most difficult to fix; hence, the behaviour about priority observed in C2 (and C1, C4, C5 too, but with *p-value* $> .05$) can be partially justified. For such a behaviour, it should be decided, at policy level, if it is acceptable or not, namely if high severity defects are accepted to take longer even though marked with a high priority. On the other hand, although this seems the most likely hypothesis to investigate, we also notice that the differences are significant at 95% only for two out of six components – therefore data are not sufficient to statistically support the provided explanation. Besides the mentioned cases of C2 and C6, the insignificant difference in all the others confirms that severity, as priority, did not make the difference among CSCIs in terms of fixing effectiveness and efficiency.

Table XI. Severity vs. Time to Fix Medians for each CSCI

CSCI	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**
High	108.53	7.36	6.95	69.09	12.97	13.93
Low	82.97	0.97	3.22	23.60	11.07	17.74
Normality *Shapiro-Wilk*	**<0.0001**	**<0.0001**	**<0.0001**	**<0.0001**	**<0.0001**	**<0.0001**
Unequal variance *Levene's test*	**<0.0001**	0.5760	**<0.0001**	**<0.0001**	0.37	**0.0009**
High vs. Low test *(adj. p-value)*	0.8220	**0.0051**	0.7917	0.8640	0.3721	**0.0267**

5. DISCUSSION

5.1. Global results

Results of the study focused on the implementation, detection, and fixing evaluated globally and at CSCI-level. Besides the presented results, we summarize some further global findings (F_i) that may be useful in contexts resembling the one outlined here.

Regarding the first research question, we observed that the implementation phase effectiveness, measured as expected quality, is, in the average, 0.469 KLoC per defect corresponding to a defect density of 6.104 defects per KLoC, while the efficiency, measured as productivity and its adjusted value (i.e., quality-aware productivity), turned out to be 1.365 and 0.434 KLoC per man-week, respectively. About variability across CSCIs, we observed that:

F1: in this type of process, even with requirements specification and design phases under the control of the same entity, the effectiveness and efficiency of the implementation given to suppliers are highly variable, with coefficient of variation (CoV) of metrics greater than 50%: we had $CoV\%$ = 74.3%, 152.66%, 57.69%, and 135.48% for, respectively, expected defect density, expected quality, productivity, and quality-aware productivity. This indicates much margin for improvement by just levelling the performance of the single CSCIs around the same values, reducing the heterogeneity. The adoption of stricter coding rules to impose to supplier is an action supposed to improve this aspect. More specific actions can be taken by considering results at CSCI-level (see the next Section).

As for the second research question, the effectiveness of the testing process in terms of defect detection state turned out to be of 89.43% in the average, whereas the testing efficiency is of 3.69 defects per man-week, corresponding to 1.54% of total expected defects detected per man-week. Relatively to the predefined levels of 50% and 90% of total expected defects, the average number

DOI: 10.1002/smr

of man-weeks required to achieve those levels is 32.67 and 53. The normalized values, i.e., man-weeks per defect to achieve those levels, are 3.99 and 4.32, respectively. Coefficient of variations are in this case much lower than implementation and fixing processes; they are: 20.18%, 27.79%, and 40.25%, 34.33%, and 21.06% for, respectively, the detection state, detection rate, percentage detection efficiency, and the two relative detection effort values, at 50% and 90%. In this case, there is a smaller variability with respect to the implementation process, reflecting the fact that the detection of all the CSCIs is done internally (with roughly the same process, even though by different team). There is however margin for improvement too. Additionally, a common pattern that should be improved is the low detection rates experienced at early testing stages, namely:
F2: in testing (externally supplied) components, the detection rate in the first 50% of testing time is lower than the rate at 90% . We observed this in most CSCIs; a possible explanation is related to the complexity of CSCIs that causes a slow start-up by testers in learning its behaviour and in setting up the test scaffolding.

The third research question was about the fixing process. In this case, the percentage of closed defects is of 73.3%, with a rate of 0.084 defects per day taking the median, corresponding to an average time to fix of 11.89 days. The time to fix distribution shows good properties in terms of skew and peakedness, but it exhibits high variability (like the implementation). *CoV*%s are: 34.57%, 61.15%, 84.21%, 74.08%, 130.87% for, respectively, the percentage of closed defects, the *fix rate* and *time to fix* taking the mean, the *fix rate* and *time to fix* taking the median. This reflects the fact that:
F3: the variability of implementation and fixing process is greater than the variability of the testing process, presumably because they are carried out by (different) suppliers, while testing is done internally.

Further indications come from the fine-grain analysis. As for severity, we see that:
F4: there are significant differences among CSCIs in terms of high-severe defects, but we cannot state that severity has an impact on the performance of the testing team. Regarding reproducibility, the low number of *not-always reproducible* defects, compared to the literature, would induce to formulate the hypothesis that hard-to-reproduce defects are fewer in mission-critical systems; however, for what said in Section 4.4, this is more likely due to an inaccurate classification:
F5: despite its importance, especially in these contexts, testers overlook the reproducibility classification, not focusing properly on establishing if a defect is always reproducible or not.

Finally, priority and severity analysis *vs time to fix* reveals, besides what mentioned, some contradiction with respect to the expectation:
F6: the time to fix for high priority and high severity defects is not significantly lower than the low priority and low severity ones. In the case of severity, this is not so relevant; indeed, a defect with high impact is not necessarily more difficult to fix. In the case of priorities, managers should investigate if priority levels are set by the testers shallowly, or if suppliers neglect them. In both cases, precise guidelines and instructions are needed to make this attribute useful.

5.2. CSCI results

In all the cases, *CoV*s tell that there is high room for improvement by a better control of heterogeneity. Hereafter, we summarize the bottlenecks at CSCI- (hence at *supplier*-) level:

- C1 implementation indicators highlight productivity and quality indicators rated as fourth in the ranking. At the same time, the severity analysis denoted this CSCI as the one with highest percentage of most severe defects. Thus, a better work is required to C1's implementer to reduce high severity defects (e.g., by conducting the internal testing more focused on high impact functionalities). The same supplier is in charge of maintenance: the C1 fixing team showed by far the worst performance. All the indicators pinpoint problems in fixing effectiveness, efficiency, and internal quality. Instead, the SELEX testing team for this CSCI showed performance in the average (metrics are in the middle of the ranking - some improvements are required for the *relative detection effort at 90%*, $RDE_{90\%}$).
- C2 has a very good implementation and fixing process. In fact, the only metric rated in the fifth position is the fixing effectiveness, but, as discussed, a deeper analysis showed that this

DOI: 10.1002/smr

Table XII. Ranking per CSCI with respect to the most relevant metrics

CSCI	C1	C2	C3	C4	C5	C6
Implementation Metrics						
EQ	4^{th}	2^{nd}	6^{th}	3^{rd}	1^{st}	5^{th}
P	4^{th}	2^{nd}	6^{th}	3^{rd}	1^{st}	5^{th}
QP	4^{th}	2^{nd}	6^{th}	3^{rd}	1^{st}	5^{th}
Detection Metrics						
State	4^{th}	2^{nd}	3^{rd}	6^{th}	1^{st}	5^{th}
Rate	3^{rd}	1^{st}	6^{th}	5^{th}	4^{th}	2^{nd}
DE%	2^{nd}	1^{st}	6^{th}	5^{th}	3^{rd}	4^{th}
$RDE_{50\%}$	4^{th}	5^{th}	1^{st}	6^{th}	3^{rd}	2^{nd}
$RDE_{90\%}$	5^{th}	2^{nd}	4^{th}	6^{th}	3^{rd}	1^{st}
Fixing Metrics						
%Closed	6^{th}	5^{th}	1^{st}	3^{rd}	2^{nd}	4^{th}
$FixRate_{Med}$	6^{th}	1^{st}	2^{nd}	5^{th}	3^{rd}	4^{th}

metric has been influenced by the abrupt increase of defect detection. Testing is good in terms of effectiveness and efficiency; however, the *relative detection effort at 50%* (*RDE* at 50%) is low, while it increases at 90%, and keeps increasing at 98%. Since the overall rate is the best one, the suggestion is to anticipate this ability, improving the defect detection in the early stage.

- C3 has the worst implementation as for quality and productivity; the overall detection rate and testing efficiency are also unsatisfactory. A positive note regards the detection trend, which decreases over time (at 50%, at 90% and at 98%). The process is continuous and regular, but the detection rate is too slow. From the fixing point of view, C3 is the best one as number of pending defects, and the second one as time to fix. C3 supplier provides a bad implementation partially "compensated" by a good fixing team.
- C4 has a close-to-average implementation, with the lowest percentage of high severity defects, but a non-satisfactory fixing. The latter should improve both the *fix rate* and the continuity and homogeneity of the process. The testing team is still at early stage. The detection is at 53% of the total estimated defects, and, in this 53%, all the detection rate indicators are bad. Testing and fixing for C4 need substantial improvement.
- C5 supplier is the best one in terms of productivity and defect density; at the same time, fixing is very satisfactory. Testing is almost at its final stage (98.88% of total defects), and the rate indicators are in the average. As for the continuity of testing, an increase from $RDE_{50\%}$ to $RDE_{90\%}$ of the rate indicates the need to improve the early detection, like C2, whereas at the end the rate correctly goes toward the saturation.
- C6 implementation is close to the worst one; the defectiveness introduced is also reflected by the good job of the testing team (whose process achieved the 92% of defects, which is good although is the fifth one, and the best rate at 90%). The detection rate is also correctly decreasing from 90% to 92% , even though not from 50% to 90%. Fixing is rated as fourth; we showed that some problems are present in terms of continuity of the process that leads to many pending defects. Stricter rules for a more continuous fixing should be imposed to this supplier too.

Both global and CSCI-level remarks are being used by SELEX engineers to implement improvement policies internally and toward each CSCI supplier as well.

5.3. Threats to Validity

In this subsection, we discuss the potential threats to validity, using the scheme by Wohlin et *al.* [31].

- **Construct validity**: the computation of the estimated number of total defects (*EstDefects*), used for constructing implementation and testing metrics, is based on SRGMs. These models rely on a set of assumptions (e.g., independent inter-failure times, equal probability to find a failure across time units, immediate repair, no change to the code during testing), which are easily violated in the practice. However, SRGMs have been demonstrated to provide good

DOI: 10.1002/smr

results in terms of prediction accuracy, even when these assumptions are partially violated [12] (especially in the presence of numerous data points as in our case), and are therefore regularly used in the practice since more than 30 years. Additionally, to mitigate the threat, we relied on a set of eight SRGMs, of which the best one is selected in terms of AIC, resulting in high $\bar{R}^2$ goodness-of-fit measures.

In the implementation process analysis, we opted for a structural metric, namely the number of KLoC, to measure the size of the software. Although functional size measures could also be adopted for the same type of analysis, the number of KLoC is judged reliable enough in this context, since the chosen CSCIs are written in the same language and all CSCIs stem from the same design process used by SELEX ES design teams (e.g., they adopt the same internal design guidelines and rules).

As for the testing process, the *relative detection effort* metric using the 90% of expected defects ($RDE_{90\%}$), when computed on C4, is based on a prediction of effort rather than on the actual effort as in the rest of CSCIs. This is because the percentage of detected defects for C4 did not achieve the 90% yet, but it is at 53%. We therefore used the predicted effort required for C4 testing to achieve the 90% of total defects, by using its SRGM, rather than the actual effort. The uncertainty associated with the prediction might bias the evaluation of C4. However, while we suggested care to SELEX ES engineers in interpreting the value of $RDE_{90\%}$ for C4, there are several reasons reducing its potential impact on the overall evaluation of C4 testing process: *i)* both testing effectiveness and efficiency metrics pinpoint the bad performance of C4; *ii)* the same metric evaluated at 50% (where defect data are available for C4 as for the other CSCIs) confirms the worst performance of C4 like the $RDE_{90\%}$ metric; *iii)* the prediction is carried out through a model with a goodness-of-fit measure of $\bar{R}^2 \approx 0.93$ obtained over 152 data points, i.e., the 53% of expected total ones. Overall, consistency of metrics along with the goodness-of-fit of the model for C4, allow us to confidently confirm the results discussed for C4.

Finally, in the fixing process, the time to fix is calculated as the time when the resolution status is set to fixed minus the time when it is set to opened. Although we exclude the time in which correction is verified (which may be highly impacting [32]), this may be not the exact amount of time in which the developer actually works on the bug fix. This might mislead the judgement about the root cause of the inefficiency of suppliers' fixing process. If this is the case, what changes is the action that suppliers would implement to get better times to fix: instead of improving the fixing activity in itself, it would simply impose, internally, to use the bug tracking system correctly in the maintenance process so that values would represent the actual fixing time.

- **Internal validity**: some considerations derived from priority, severity, and reproducibility are under the assumption that detection teams within SELEX ES use the same criteria to judge and assign these attributes to defects, and, hence, the bias due to different people doing this task is minimized. More generally, results could be not representative of the context if people involved in the evaluated processes were aware of the project; however, we assured that neither developers, nor testers, nor suppliers, as well as project managers of these CSCIs were aware of the study until the end of data collection; hence results are free from potential psychological bias.

  A further treat is that we are considering only the discovered defects; there are possibly others with a different distribution that can change the result. Additional threats to internal validity include the correctness of scripts for data collection process as well as of implementation of the fitting algorithm (namely, the EM algorithm [20]) for SRGMs construction done by authors.
- **External validity**: results rely on six CSCIs analysed within one industrial context in the field of large-scale mission-critical systems. Results observed on a single case are not statistically generalizable; rather, reported findings, supported by data, serve as basis to develop testable hypotheses for contexts (namely, type of systems and development models) similar to the considered one, which researchers can verify by replicating our analysis in similar industrial

DOI: 10.1002/smr

settings. We believe that, since analyses on proprietary industrial systems are scarce in the literature, the reported results provide an important contribution despite the limitations in terms of external validity.

- **Conclusion validity**: conclusions about the effectiveness and efficiency of analysed processes are drawn from the considered CSCIs. Due to the information available about the process (hence, to metrics derived from it) and to the limited number of CSCIs, we have based most of the analyses on average and standard deviation figures, limiting the application of statistical tests to fine-grain analysis on severity, priority, and reproducibility. More CSCIs and further insights in the process could entail stronger conclusions in terms of statistical validity.

These threats have been presented to SELEX engineers as possible causes of bias; they should be had in mind before drawing conclusions based on the observed results.

5.4. Remarks on the implemented evaluation method

We hereafter briefly discuss about the potential benefits and the drawbacks we found in applying the described evaluation method. The key for success has been definitely the very low effort required to SELEX ES engineers, which has been a driving criterion from the beginning. As experienced by many researchers in software engineering, it is often the case that industry is, with reason, anchored to their consolidated processes to build this kind of (slowly changing) systems. Instead of trying of changing their processes from the top, we experienced that starting from their daily work and from exploiting the information available "for free" is by far more useful. This, in fact, produces immediate and visible feedbacks, which more easily expose the need for implementing improvement actions. In terms of effort, one engineer from SESM, during the project, acted as interface between the academic and industry part, interpreting the SELEX management needs and procuring the needed data. Other engineers, working on the specific CSCI, were occasionally asked for small technical issues to facilitate the data gathering. Compared with more rigid defect analysis solutions, this approach required no training, no process change, no customization, and no modification to the way people are used to work routinely.

Moreover, since basic information is required to obtain the presented evaluations, the reusability of the approach is expected to be high. Defect tracking and analysis is indeed a practice recommended by the most important software process standards [33], [34], [35], and the attributes chosen for the analysis are very basic ones (e.g., opening and closing time, defect severity, priority). These are supposed to be collected by most of companies, at least in the context of mission-critical systems. Reusing the approach requires therefore a preliminary analysis of the context, in order to infer process phases, lifecycle model features, and defects data available; once data are collected, the same analyses reported in this paper are easily applicable to other contexts, provided that such basic attributes are collected. Finally, from the academic point of view, such an approach allowed setting up a concrete experience in an industrial scenario, which is often the missing link to convince people that certain changes are possible and worthwhile.

On the other hand, there are limitations of the adopted strategy. Being it based on a black box approach, we had a partial view of the process internals; consequently, deeper analyses were not possible. For instance, we could not perform evaluations based on: defect type, defect trigger, fault-slip-through analysis about phases where defects were injected, detected, and corrected, source code-defects relationship, and many others. Additionally, while we rely on existing data, the other side of the coin is that the applicability is fully dependent on which information is collected. With an ODC-like approach, this is established a priori at the expense of starting collecting data from scratch. A suitable strategy to profitably apply defect analysis in industry could be to start from applying a lightweight and flexible approach like ours at the beginning, providing immediate evidence, and then turning to more complex methods once people are convinced of their potential usefulness.

DOI: 10.1002/smr

6. RELATED WORK

Defect analysis is widely used in software quality assurance and process improvement. The mentioned schemes of HP classification and ODC have been implemented with success into several contexts (e.g., [36], [37], [38], [39], [40] [41]). The board characterization they provide is useful especially for companies willing to setup a defect tracking process from scratch, and yields a rich feedback to the process. On the other hand, several difficulties have been experienced, for training, start-up, and customizations or adaption to existing processes [5], [3], [4], [42], [43], [44]. These difficulties are exacerbated as the software development process within the organization is hard to change (e.g., it involves several independent groups, is geographically distributed, unstructured, etc.), and the type of documents and considered defects differ from traditional ones.

Defect analysis for process improvement has been used in [45], where authors apply it as feedback to improve quality and productivity in an iterative development model. In [46], authors classify and analyse about 12,000 defects taken from bug tracking of three companies, showing that most of defects, 65.5%, are functional (computation or logic); results are supposed to support software process improvement. Other works focus on specific aspects of the defect analysis cycle; in [47] authors remark the importance of an efficient defect reporting on the testing process, demonstrating that improving reporting decreases the percentage of invalid reports from 26 to 19.53%. Several studies analysed the fixing process of defects. The study in [48] reports an analysis on 1500 defects revealed in 5 years on an IBM middleware, classifying defects per topic and developer expertise, showing that the time to fix is impacted from these two factors. Zhang et *al.* [49] found some factors influencing the time lag between the defect assignment to a developer and the actual starting of the fixing action, through a case-study on 3 open source software applications. They found that the assigned severity (unlike our case), the bug description, and the number of methods and changes in the code as impacting factors.

The work in [32] reports a study specifically focused on finding bottlenecks in the issue management process, through a case-study of the Apache web-server and the Firefox browser. The main cause of inefficiency in that case is the time lag in which the correction is verified aimed at confirming the correct resolution. We, in fact, excluded this time from the *TTFix* computation. Authors in [22] analyse the performance of the fixing process in nine open source projects, focusing on continuity and homogeneity, which we also borrowed for our analysis. Mockus et *al.* [50] conducted an analysis on two case studies, Apache and Mozilla, aimed at evaluating several aspects of the open source development. This study is close to ours with respect to the extensiveness of the analysis, even though focused on different questions, on a different class of systems, and on different goals: they evaluate aspects related to the implementation (e.g., the development community largeness, the community contribution and coordination, the introduced defectiveness), and to the fixing (time to fix problems, and priority analysis) in open source settings.

These works differ in several aspects from ours, the most important ones being: *i)* the processes evaluated, wherein most of the surveyed works focuses on one specific aspect (e.g., defect type, defect fixing, defect reporting); *ii)* the procedure adopted for the evaluation; *iii)* the object of the evaluation, that in our case is oriented toward component-based processes, with possibly external actors involved, and toward the class of mission-critical large-scale industrial systems with their own peculiarities (e.g., high cost of a defect, very high maintenance cost, complex and heterogeneous development process, closed source code, etc.).

7. CONCLUSION

We investigated software process evaluation through defect analysis in a large system integration company. We applied a defect analysis procedure conceived to avoid introducing heavy cost and intrusive inspections into the current process. Results provided a set of insights on the overall development efficiency, on each of the monitored activities, and on potential causes of inefficiencies, as critical phases, critical teams on a specific component, or unsatisfactory suppliers.

DOI: 10.1002/smr

Findings are of support to practitioners dealing with medium and large-scale systems developed according to a component-based approach, having the same problems entailed by this type of development model. The "lightweight" procedure can also be borrowed for similar analyses, especially by those researchers involved into industrial environments. Additionally, individual results on implementation, detection, and fixing can be compared to other contexts where it makes sense to see pros and cons of such a development model. We expect that future research will address such a kind of process evaluation, considering several aspects together, including issues and challenges that come out from industrial contexts to go toward actionable evaluation frameworks.

8. ACKNOWLEDGEMENT

This work has been supported by MIUR under project SVEVIA (PON02_00485_3487758) of the public-private laboratory COSMIC (PON02_00669). The work of Dr. Pietrantuono is supported by the project Embedded Systems in Critical Domains (CUP B25B09000100007) in the framework of POR Campania FSE 2007-2013.

REFERENCES

1. Chillarege R, Bhandari I, Chaar J, Halliday M, Moebus D, Ray B, Wong M. Orthogonal defect classification – a concept for in-process measurements. *IEEE Transactions on Software Engineering* 1992; **18**(11):943–956.
2. Grady R. *Practical Software Metrics For Project Management and Process Improvement.* Hewlett-Packard Professional Books, 1992.
3. Mellegård N, Staron M, Törner F. A light-weight defect classification scheme for embedded automotive software and its initial evaluation. *Proc. 23rd IEEE International Symposium on Software Reliability Engineering (ISSRE),*, 2012.
4. Wagner S. Defect classification and defect types revisited. *Proc. 2008 workshop on Defects in large software systems*, 2008; 39–40.
5. Freimut B, Denger C, Ketterer M. An industrial case study of implementing and validating defect classification for process improvement and quality management. *Proc. 11th IEEE Int. Symposium on Software Metrics*, 2005.
6. MIL-STD-498, Overview and Tailoring Guidebook, US Dept. of Defense 1996.
7. Carrozza G, Faella M, Fucci F, Pietrantuono R, Russo S. Engineering air traffic control systems with a model-driven approach. *IEEE Software* 2013; **30**(3):42–48, doi:http://doi.ieeecomputersociety.org/10.1109/MS.2013.20.
8. Huang CY, Lyu M. Optimal release time for software systems considering cost, testing-effort, and test efficiency. *Reliability, IEEE Transactions on* Dec 2005; **54**(4):583–591, doi:10.1109/TR.2005.859230.
9. Pachauri B, Kumar A, Dhar J. Modeling optimal release policy under fuzzy paradigm in imperfect debugging environment. *Inf. Softw. Technol.* Nov 2013; **55**(11):1974–1980.
10. Pietrantuono R, Russo S, Trivedi K. Software reliability and testing time allocation: An architecture-based approach. *Software Engineering, IEEE Transactions on* May 2010; **36**(3):323–337, doi:10.1109/TSE.2010.6.
11. Carrozza G, Pietrantuono R, Russo S. Dynamic test planning: a study in an industrial context. *International Journal on Software Tools for Technology Transfer* 2014; :1–15doi:10.1007/s10009-014-0319-0. URL http://dx.doi.org/10.1007/s10009-014-0319-0.
12. Almering V, Genuchten MV, Cloudt G, Sonnemans P. Using software reliability growth models in practice. *IEEE Software* 2007; **24**(6):82–88.
13. Goel AL, Okumoto K. Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability* 1979; **R-28**(3):206–211.
14. Yamada S, Ohba M, Osaki S. S-shaped reliability growth modeling for software error detection. *IEEE Transactions on Reliability* 1983; **R-32**(5).
15. Goel AL. Software reliability models: Assumptions, limitations and applicability. *IEEE Transactions on Software Engineering* 1985; **SE-11**(12):1411–1423.
16. Gokhale S, Trivedi K. Log-logistic software reliability growth model. *Proc. 3rd Int. High-Assurance Systems Engineering Symposium (HASE)*, 1998; 34–41.
17. Mullen R. The lognormal distribution of software failure rates: application to software reliability growth modeling. *Proceedings of the The Ninth International Symposium on Software Reliability Engineering*, 1998; 134–142.
18. Okamura H, Dohi T, Osaki S. Em algorithms for logistic software reliability models. *Proc. 22nd IASTED Int. Conference on Software Engineering*, 2004; 263–268.
19. Ohishi K, Okamura H, Dohi T. Gompertz software reliability model: Estimation algorithm and empirical validation. *Journal of Systems and Software* 2009; **82**(3).
20. Okamura H, Watanabe Y, Dohi T. An iterative scheme for maximum likelihood estimation in software reliability modeling. *Proc. 14th Int. Symp. on Software Reliability Engineering (ISSRE)*, 2003; 246–256.
21. Cinque M, Gaiani C, De Stradis D, Pecchia A, Pietrantuono R, Russo S. On the impact of debugging on software reliability growth analysis: A case study. *Computational Science and Its Applications – ICCSA 2014*, *Lecture Notes in Computer Science*, vol. 8583, Murgante B, Misra S, Rocha A, Torre C, Rocha J, Falcão M, Taniar D, Apduhan

B, Gervasi O (eds.). Springer International Publishing, 2014; 461–475, doi:10.1007/978-3-319-09156-3_33. URL http://dx.doi.org/10.1007/978-3-319-09156-3_33.
22. Francalanci C, Merlo F. Empirical analysis of the bug fixing process in open source projects. 2008. *Open Source Development, Communities and Quality* 2008; **275**:187–196.
23. Chillarege R. Understanding Bohr-Mandel bugs through ODC triggers and a case study with empirical estimations of their field proportion. *Proc. 3rd Int. Workshop on Software Aging and Rejuvenation (WoSAR)*, 2011; 7–13.
24. Grottke M, Nikora A, Trivedi K. An empirical investigation of fault types in space mission system software. *Proc. Int. Conference on Dependable Systems and Networks (DSN)*, 2010; 447–456.
25. Carrozza G, Cotroneo D, Natella R, Pietrantuono R. Analysis and prediction of mandelbugs in an industrial software system. *Proc. 6th Int. Conference on Software Testing, Verification and Validation (ICST)*, 2013; 262–271.
26. Cavezza DG, Pietrantuono R, Russo S, Alonso J, Trivedi KS. Reproducibility of environment-dependent software failures: An experience report. *To appear in: The 25th IEEE International Symposium on Software Reliability Engineering*, 2014.
27. Ruxton GD. The unequal variance t-test is an underused alternative to student's t-test and the mann–whitney u test. *Behavioral Ecology* July/August 2006; **17**(4):688–690.
28. Neuhauser M. Two-sample tests when variances are unequal. *Animal Behaviour* 2002; **63**:823–5.
29. Zimmerman D, Zumbo B. Rank transformations and the power of the student t test and welch t test for non-normal populations with unequal variances. *Canadian Journal of Experimental Psychology/Revue canadienne de psychologie expérimentale* Sep 1993; **47**(3):523–539.
30. Conover WJ, Iman RL. Rank transformations as a bridge between parametric and nonparametric statistics. *The American Statistician* 1981; **35**(3):124–129.
31. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers: Norwell, MA, USA, 2000.
32. Ihara A, Ohira M, Matsumoto K. An analysis method for improving a bug modification process in open source software development. *Proc. of the joint Int. and Annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, 2009; 135–144.
33. ISO/IEC 15504:2012 - Information technology – Process assessment 2012.
34. ISO/IEC 12207-2008. Systems and software engineering - Software life cycle processes 2008.
35. CMMI Product Team. CMMI for Development, Version 1.2, CMMI-DEV, V1.2, CMU/SEI-2006-TR-008, ESC-TR-2006-008. Pittsburgh: Software Engineering Institute. 2006.
36. Butcher M, Munro H, Kratschmer T. Improving software testing via ODC: Three case studies. *IBM Systems Journal* 2002; **41**(1).
37. Mullen R, Hsiao D. Orthogonal Defect Classificiation at Cisco. *Proc. ASM Conference*, 2002.
38. Chillarege R, Chillarege KRPR, Chillarege KRPR, Prasad KR. Test and Development Process Retrospective - a Case Study using ODC Triggers. *Proc. Int. Performance and Dependability Symposium*, 2002.
39. Bridge N. Orthogonal defect classification using defect data to improve software development. *Software Quality* 1998; **3**.
40. Cotroneo D, Pietrantuono R, Russo S. Testing techniques selection based on odc fault types and software metrics. *Journal of Systems and Software* 2013; **86**(6):1613 – 1637, doi:http://dx.doi.org/10.1016/j.jss.2013.02.020.
41. Grady RB. *Successful Software Process Improvement*. Prentice Hall, 1997.
42. Li J, Stalhane T, Conradi R, Kristiansen JMW. Enhancing defect tracking systems to facilitate software quality improvement. *IEEE Software* 2012; **2**(29):59–66.
43. Dubey A. Towards adopting odc in automation application development projects. *Proc. 5th India Software Engineering Conference*, 2012; 153–156.
44. Wagner S, Jurjens J, Koller C, Trischberger P. Comparing bug finding tools with reviews and tests. *Proc. 17th Int. Conference on Testing of Communicating Systems*, vol. 3502, Springer, 2005; 40–55.
45. Jalote P, Agrawal N. Using defect analysis feedback for improving quality and productivity in iterative software development. *Proc. ITI 3rd Int. Conference on Information and Communications Technology. Enabling Technologies for the New Knowledge Society*, 2005; 703–713.
46. Raninen A, Toroi T, Vainio H, Ahonen JJ. Defect data analysis as input for software process improvement. *Proc. 13th Int. Conference on Product-Focused Software Process Improvement*, *Lecture Notes in Computer Science*, vol. 7343, Springer, 2012; 3–16.
47. Wang D, Wang Q, Yang Y, Li Q, Wang H, Yuan F. Is it really a defect? an empirical study on measuring and improving the process of software defect reporting. *Proc. Int. Symposium on Empirical Software Engineering and Measurement*, 2011; 434–443.
48. Nguyen TT, Nguyen T, Duesterwald E, Klinger T, Santhanam P. Inferring developer expertise through defect analysis. *Proc. 34th Int. Conference on Software Engineering (ICSE)*, 2012; 1297–1300.
49. Zhang F, Khomh F, Zou Y, Hassan A. An empirical study on factors impacting bug fixing time. *Proc. 19th Working Conference on Reverse Engineering (WCRE)*, 2012.
50. Mockus A, Fielding RT, Herbsleb JD. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodolology* 2002; **11**(3):309–346.