

Towards A Software Failure Cost Impact Model for the Customer



An Analysis of an Open Source Product

Ralf Gitzel
ABB Corporate Research
Center
Ladenburg, Germany
Ralf.Gitzel@de.abb.com

Simone Krug
University of Mannheim,
Germany
Information Systems III
skrug@wifo.uni-mannheim.de

Manuel Brhel
University of Mannheim,
Germany
Information Systems III
mbrhel@wifo.uni-mannheim.de

ABSTRACT

While the financial consequences of software errors on the developer's side have been explored extensively, the costs arising for the end user have been largely neglected. One reason is the difficulty of linking errors in the code with emerging failure behavior of the software. The problem becomes even more difficult when trying to predict failure probabilities based on models or code metrics. In this paper we take a first step towards a cost prediction model by exploring the possibilities of modeling the financial consequences of already identified software failures. Firefox, a well-known open source software, is used as a test subject. Historically identified failures are modeled using fault trees. To identify costs, usage profiles are employed to depict the interaction with the system. The presented approach demonstrates the possibility to model failure cost for an organization using a specific software by establishing a relationship between user behavior, software failures, and costs. As future work, an extension with software error prediction techniques as well as an empirical validation of the model is aspired.

Categories and Subject Descriptors

D.2.4 [Software Engineering]: Software/Program Verification; D.2.9 [Software Engineering]: Management—*Cost estimation*

General Terms

Open Source Software, Software Failure Costs, Reliability

1. INTRODUCTION

Errors made during the programming of software often lead to faulty behavior for end users. These perceived failures typically cause a number of direct and indirect financial consequences. While the effort and financial effect for the developers have been explored quite extensively [37], [40], [15],

the cost impact for the end user has received little attention so far. Researchers have been observing failure costs for a longer time, but always from the software manufacturer's perspective. Thus, costs for the end user play a menial role and are usually translated into warranty and service costs associated with product failure [24]. There are graphic textbook examples like the ARIANE 5 software failure [9], but the everyday business impact of poor software quality on the economy is rarely addressed by scientists. However, they should not be neglected, as cumulated over a longer period of time they will certainly have a significant impact. Note that the impact of an error is not seen from the developer's perspective here, e.g. in terms of the time needed to fix a bug. Expecting a user to do so would require access to the source code and the necessary expertise. This is rarely the case, even the users' influence on developers to resolve the fault is usually very limited. Therefore, our work does not focus on fixing a bug but on dealing with its impact on the end user.

While it is already difficult to statistically predict the number of defects contained in an application and the time needed to fix them, end user cost prediction adds a new challenge [32], [43]. In addition to failure probabilities, the impact on the work of the end user must be determined. For this purpose, more details of the code's behavior have to be modeled than for a prediction of code maintenance costs. In this paper, we take a first step towards a metric-based prediction of end user cost. Our approach is to perform a failure analysis on an existing software product using a database of known errors. The intention is to find out how well our chosen technique can be used to assess the cost of existing failures before applying predictive algorithms, which is our long-term goal. In order to establish a direct relationship between errors and costs, our main contribution in this work is to analyze the connection between software failures and costs for the end user.

The paper starts with a brief literature overview in Sect. 2. Sect. 3 elaborates our approach in detail, showing the relationship between user behavior, errors, and costs. Sect. 4 is the core analysis where we apply our approach to a fictional organization using the Web browser Firefox. The paper concludes with a discussion of the lessons learned and future work.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PROMISE2010, Sep 12-13, 2010, Timisoara, Romania
Copyright 2010 ACM ISBN 978-1-4503-0404-7...\$10.00.

2. RELATED WORK

While reliability is one of the most important factors in engineering disciplines, regarding software it does not really live up to our expectations [25]. Software reliability can be defined as the continuity of correct service [5] or the absence of events leading to incorrect service. Musa and Okumoto define it as the probability of a failure-free operation of a computer program in a specified environment for a specified time [29]. In contrast to this, other authors, e.g. in [2], describe software reliability as the end user's view of software quality, which cannot be directly measured. Concerning reliability, we will adopt Musa's approach as this is the one best applicable for future simulation.

2.1 Software Failure Classification

Avizienis et al. define a failure as an event that occurs when a system's delivered service deviates from the correct service of the system [5]. A failure is caused by the activation of one or more faults, i.e. software defects, persistent the software's source code. Those faults are the manifestation of errors committed by software developers [1]. As the term bug is not used consistently throughout the community, we define its meaning as synonymously to fault. It is important to notice that while an error does not necessarily induce a failure, e.g. due to fault tolerance mechanisms, every software failure may be traced back to one or more errors [31].

Mandeville distinguishes between internal and external failures [26]. While internal failures occur under the developer's control, external failures arise in front of the user. They are also called field defects [23], post-release defects [31], or field failures [38]. We limit our model to the term failure, meaning those relevant for the end user. In addition to the definition given above, a failure can also be defined from the view of the user. Myers [30] stated that a software failure occurs when the software does not fulfill the reasonable expectations of the end user. Note that, following this definition, a failure is occurring when software exhibits conformance to its requirements specified by the developer, but this behavior deviates from the user's expectations.

An example we found in the Firefox Bugzilla database illustrates this well: When uninstalling a profile, bug # 270 705 deletes all files in the same folder, i.e. not only the profile's files, rather than the complete content of the particular folder. The developers claimed that this is indeed not a bug but an advanced feature as profile handling is usually only utilized by very experienced users. However, the damage can have a significant monetary impact if data is lost. We therefore focused on failures leading to the most important consequences for the user. We identified data loss and the breakdown of the program as most critical, as these entail significant effort on the side of the user. The most common data loss within Firefox is the loss of bookmarks. Next to crashing, freezing of the software leads to similar effects, as the user is no longer able to use it. The main failure scenarios for Firefox in this work are thus the loss of bookmarks, the programs crash, and the freeze of Firefox.

2.2 Software Defect Prediction

As for the prediction of software faults, a large amount of literature has been published. Historically, most approaches drew on static code metrics such as those introduced in the

object-oriented metrics suite by Chidamber and Kemerer [8], which has been validated by several authors, e.g. in [6]. Despite the criticism passed on static code metrics, they are still popular and widely used [12]. Gyimothy et al. showed that they are also suited to detect fault proneness of the source code in open source projects [16]. Other authors include metrics such as code churn [14], or information about process quality in software development, e.g. the number of developers editing the code [44]. Data is frequently analyzed with the help of regression models [33], or by employing various machine learning techniques, e.g. Support Vector Machines [10]. Machine learning techniques are also useful to predict the existence of bugs in software changes [22].

An exhaustive discussion of defect prediction models and metrics is beyond the scope of this paper. With the plethora of methods published, the assessment and comparison of defect prediction models is crucial and has gained much attention within the community recently [20], [27]. A study published lately by Arisholm, Briand, and Johannessen compared different modeling techniques for defect prediction, showing that an assessment is highly dependent on the evaluation criteria employed. A major feature of this study is the use of a surrogate cost-effectiveness measure to compare fault prediction models. The cost-effectiveness proved to be less dependent on the modeling technique applied than on the metrics used [4]. As the proven cost-effectiveness of fault prediction methods may presumably lead to an increased use of those methods by developing companies, this could be an important contribution to lessen the economic impact of software failure on end users.

2.3 Reliability in Open Source Software

As the popularity of Open Source Software (OSS) is rising during the last years, so do the discussions about whether its reliability can compete with the reliability of closed source software. There is strong evidence for a superior reliability of OSS as bugs are tracked down and fixed by the community faster due to the exposed code [36], [28], [34], [39]. Zhou and Davis [45] suggest that the reliability of OSS exhibits a similar growth pattern as close source software. Hence, as the software matures, bug arrival rates should stabilize at a low level after a certain development time, which can be considered as a point for adoption. Contradictory to these findings, the project analyzed in our study (Mozilla Firefox) appears to have an unsteady bug arrival rate despite being a mature software product that has been in development for years. We may explain this with the high number of new features that is introduced with every major version change, coming along with new bugs. An interesting point is the influence of testing on OSS quality. Li et al. [23] analyzed the characteristics of bugs in OSS, namely the root cause of a bug, its impact and the software component it occurred in. Their experience that many bugs are based on trivial causes as e.g. NULL pointer dereferences or uninitialized memory reads matches the observations in our analysis.

Our analysis is largely based on the Firefox bug repository in Bugzilla. Anvik, Hiew, and Murphy [3] analyzed the quality of information that is stored in publicly available bug repositories. Given the high number of bugs that are reported daily, he found that most reported bugs are not relevant for further investigation. Mostly they are either duplicates of

an existing report, do not describe an actual bug, or cannot be reproduced. Some reports describe behavior that will not be changed, as it is not regarded to be a fault by the developers. Duplicates present the largest percentage of all bugs.

Hooimeijer and Weimer [18] introduce a model to measure bug report quality using the time until a bug is resolved as an indicator. In contrast to this, Bettenburg et al. [7] propose a prediction model based on the feedback from developers to appraise the quality of a bug report. They describe the remarkable mismatch between the information needed by OSS developers to fix a bug and the one that is provided by bug reporters. They also emphasize the need for well-structured bug reports, which are essential for both developers and researchers that use those reports for their work. Weiss et al. [42] presented an approach for an automated prediction of the time needed to fix a reported bug. However this kind of effort prediction only affects the developer's perspective, disregarding the concerns of the end user.

3. MODELING END USER FAILURE COSTS

In order to estimate the cost of software failure for the end user, we propose the following approach:

1. Identify cost drivers
2. Model failure occurrence by effect
3. Model user behavior
4. Calculate expected cost for the user

After a presentation of each step in this section, we will demonstrate their application in the subsequent chapter.

3.1 Identify Cost Drivers

The impact of a failure can only be described in terms of the user's ability to perform the task intended. Software failures may have negligible cost impact on the end user, as an alternative way to accomplish the task may be readily available. In other cases, failures can entail either significant repair costs for fixing the fault or search costs for finding an alternative. Also, in certain situations resolving the consequences of a software failure may be linked to costs, for example if replacing a tool destroyed by the malfunctioning of a machine controlled by software is necessary. These costs are evident; however it is insufficient to only observe costs which can be derived directly from failures. In an economic context, labor time lost is an essential cost factor. We argue that the various immaterial consequences of a software fault, as e.g. data loss, user dissatisfaction or cognitive load, affect the user's productivity, thus more time is needed to fulfill the task intended. Therefore, time misspent by staff due to software failures has to be taken into consideration as a primary cost driver. The time lost depends on different factors, such as the usage patterns and the capability of the user to cope with the failure. The task as well as personal preferences constitute the usage pattern in terms of interaction with the system. Different usage patterns may cause significantly different software behavior [43]. As faults are triggered by system-user interaction, the usage pattern influences the occurrence of failures. Since personal preferences

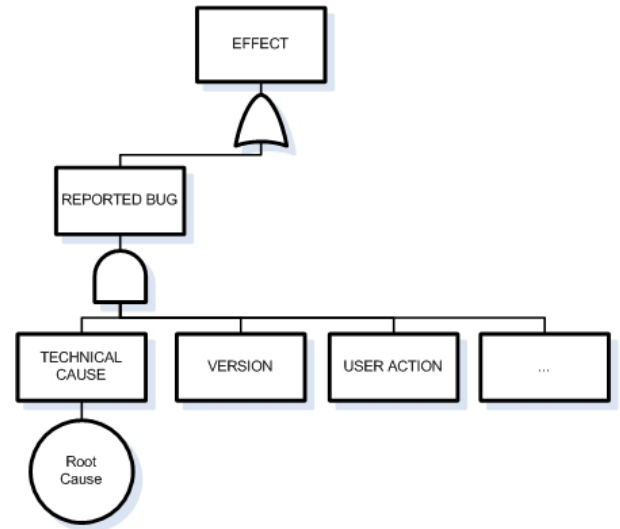


Figure 1: General Fault Tree Structure

and capabilities are linked to the individual while being independent of the task, we will not consider them but take into account the task only.

3.2 Model Failure Occurrence by Effect

In complex systems, such as most software nowadays, combinations even of minor errors can lead to far-reaching system failures [17]. This interconnection can be visualized by fault trees [41] which are a well-developed technique for analyzing the reliability and the performance of computer systems [35]. Fault trees allow breaking down complex systems with regard to the hierarchy of failure influences [21]. The symbolism is quite intuitive. Events, represented as rectangles, are connected by logical gates. The AND gate is semicircular-shaped, reminding of the logical notation. The OR gate is similar but with a rounded line at the bottom. Events which are not further developed have a diamond shape and those who have no antecedents at all are round.

Although constructing fault trees requires manual work, we consider them a valuable tool as they offer an intuitive graphical view of the different factors that have to concur in order to trigger a software failure. Going beyond the mere existence of faults in the software, we can model the failure probability by attaching probabilities to those events. The fault tree notation is also well suited to operationalize software behavior for future simulation, as it is possible to map fault trees including event probabilities to simulation software quite easily.

In the context of this approach, a fault tree has the general structure shown in Fig. 1. At the top is the effect, i.e. the failure impact of all faults in this group. On the next level, all reported bugs that can cause this effect are listed. Each of these faults occurs under a combination of circumstances that are listed on the third level. The circumstances must include a technical cause, i.e. a software defect, a software version, and a user interaction. Other conditions may be added as needed. The fault tree represents a combination of events that will lead to the top-level effect. To estimate

how often that situation occurs, user profiles are needed.

3.3 Model User Behavior

As stated in Sect. 3.1, user behavior is an essential aspect of failure cost prediction. Therefore our model includes multiple usage patterns reflecting the different tasks performed by user groups within the target organization. A user profile contains the features of a system and the purpose they are used for as well as the frequency of execution.

In our model, each user is represented by a bundle of actions that use a single functionality of the system. An action has a frequency of use per week associated with a typical length. We assume that different employees within an enterprise have similar usage patterns and hence one profile has to be created according to each job specification. Depending on the type of failure effect, a reaction time to the bug is estimated leading to time lost, which we identified as a primary cost driver in 3.1.

We also included the time to resolve the problem in the user profile, as we consider it dependable on the task. According to its complexity, the resolution time varies. Of course this also represents a simplification, as the recovery time should be allocated to a single failure instead of the task. Resolution time may decline after the user experiences a failure repeatedly due to improved coping strategies.

3.4 Calculate Expected Cost for the User

Costs can be categorized into the costs of conformance to requirements and the costs of noncompliance, the first covering the prevention and detection of defects, the second applying to failure-induced costs like fixing, reviewing, auditing, inspection, or testing [26]. Here, solely costs of noncompliance are of interest as long as they occur within the domain of the end user. As mentioned above, those costs are rarely addressed in literature.

We assess the cost of a failure in terms of the time lost to find a solution or a workaround to the failure's effect. Based on the user profiles and the fault trees, a cost estimation can be made. Assuming events in a fault tree are occurring independent of each other and that a user can only perform a single task at any given time, failure probabilities do not interfere.

The estimated time lost per week T_i due to crashes and halts can be calculated in the following way:

$$T_i = n_i \times t_i \times R_i \times P . \quad (1)$$

where

1. n_i : Expected frequency (per week)
2. t_i : Length (min)
3. R_i : Estimated resolution time (min)
4. P : Estimated failure probability


5. i : Indicator of the user profile concerned

Using T_i , it is possible to determine the estimated failure-induced costs per week C :

$$C = C^r + \sum_{i=1}^N (T_i \times C_i^m) . \quad (2)$$

where

1. C^r : Repair costs
2. C_i^m : Staff expenditures (per week)

Repair costs cover all costs concerning the recovery of the system to return to a productive state, excluding the time lost by the user. We intentionally chose a high level of abstraction here, as those costs are closely linked to the system in focus. They might be precisely recordable in the manufacturing industry, including e.g. material costs or expert consultation, but remain intangible in the context of the service industry. Also these costs are not always directly bound to a single failure, as they might occur only once for multiple issues. One could argue that repair costs tie into the user profile. As we try to minimize the use of assumptions, this is out of our scope. 

4. EXAMPLE: THE WEB BROWSER FIREFOX

In this analysis, we look at a fictional small office that uses Firefox as a Web browser. We perform the steps described in the previous section to assess the costs caused by software failures based on our model. We are aware that a Web browser may not seem to be a source of significant cost unlike specialized software applied in an industrial environment. Still, there are two major motivations for choosing this software instead of a more complex alternative: First, Firefox is well-known and no lengthy explanation of its features is needed for this paper. As the failures described below may also have been encountered by the reader, the examples are easy to follow. Second, the bug database is quite extensive and publicly available, so our findings can be reproduced by the reader.

4.1 Identify Cost Drivers in Firefox

As a first step, bugs occurring in an end user release are extracted from the database. It may be mentioned here that the majority of bugs is detected and subsequently solved during the development process. Out of the more than 78,000 bugs in the Firefox Bugzilla (as of October 2009), only a minority of ca. 7.5% appeared in versions that were actually intended for the end user. Considering our aim to quantify the cost impact for the end user, we limit our choice to those bug reports.

As mentioned above, failures in software are not always critical. Some failures even occur unnoticed and have a negligible effect on the software's functionality. Thus, only bugs

marked as blocker or critical are extracted from the data set. We further limited our sample to bug reports marked with the resolution *fixed* and the status *resolved*, *verified*, and *closed*. All others represent work in progress and can still emerge to be a duplicate, not reproducible, or will intentionally not be fixed. This limited our sample to 198 bug reports which were subsequently analyzed manually. To be suitable for a detailed analysis, bug reports have to provide a comprehensible description, ideally including the steps to reproduce the bug, and detailed development history information. Table 1 shows the result of this elimination process, grouping the example bugs by their effect.

4.2 Model failure occurrence by effect

For the sake of clarity, bugs in Table 1 are already sorted by their impact's expected severity. We consider the top four failure categories to be the most relevant cost drivers, while the bugs on the lower ranks represent situations where performing an action does not work but can be replaced by an alternative user action that is not necessarily more complicated. For example, if a search engine does not work in the toolbar, it can be directly accessed via the URL field or the bookmarks. Since the costs of security leaks are a field beyond the scope of this paper, we focus on the examples of crash and halt. When crashed, Firefox takes a certain time to restart. Logging into lost sessions can be very time consuming. In some cases, even a reboot of the system might be necessary. The halt of the Web browser will additionally lead to several minutes waiting time, plus the time needed to terminate the process manually. Figures 2 and 3 illustrate the fault trees for these cases.

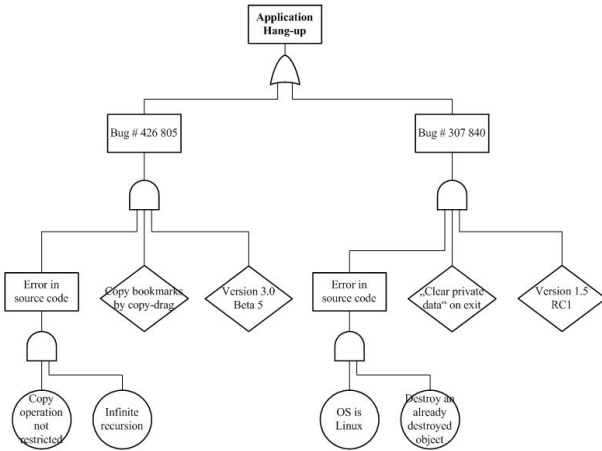


Figure 2: Fault Tree “Halt”

As shown in both fault trees, failures occur due to a combination of the user's behavior and defects in the source code, which are version-dependent. Every user of Firefox has experienced these effects; as demonstrated the occurrence is based on the complex interaction of different factors, while the source is unrecognizable to the user.

4.3 Model User Behaviors in Firefox

Using the Internet is a basis for many tasks in a company. In Europe, 93% of all companies with at least ten employees have Internet access, while 68% use the Internet to engage

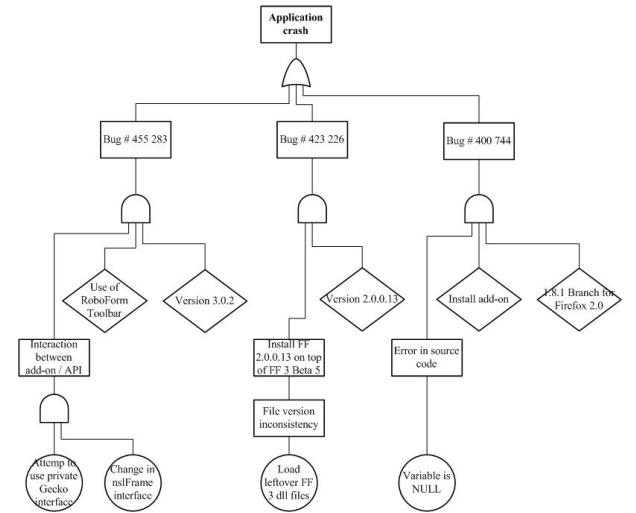


Figure 3: Fault Tree “Crash”

with public authorities [11]. Thus, Web browsers have to be used by employees to perform various work-related tasks. These tasks are typically activities such as gathering information, placing orders, comparing offers, etc. Web based applications are gaining more and more prevalence; this dependency can be expected to grow even further in the future. To do justice to the different user profiles, we selected representative departments of a fictive small office. Table 2 shows exemplary user profiles for procurement workers, secretaries and staff in the marketing department. A procurement worker will use Firefox to search the Web for the best offer matching a given purchase order. He will want to access information on products as well as order them online and even complete the transaction by using online banking. Typical online tasks of a secretary will be, amongst others, to handle hotel or meeting room reservations and order different products like office equipment. Marketing staff needs to be up to date on information concerning advertising and movements of competitors. Based on expert estimation we listed the frequency of a task per week as well as the length of use. The resolution time varies according to the complexity of the task. For some activities, e.g. online booking, substantial rework has to be done if Firefox crashes or halts and all data entered is lost. Simple activities like an online search can usually be recovered quickly.

4.4 Cost Estimation in Firefox

The failure rate in Firefox depends on the version used, thus on the user's update preferences. On one hand, grace to auto-update, most users install a new Firefox version within three days, while on the other hand the maximum share of the latest version does not exceed 80% [13]. The failure probability can be based on code metrics and simulated user behavior to obtain a more precise estimation. Still, we employ an estimated global failure probability because it is sufficient at the moment for the purpose of demonstration.

Based on the work of Huang and Boehm [19], we estimated P to correspond to $\frac{1}{300}$ per execution hour. Based on the first formula presented in Sect. 3.3 it is now possible to calculate the estimated time lost per week T_i due to crashes

Table 1: Relevant Firefox Errors.

Category	Bugs in Category (Bugzilla IDs)
Crash	455283, 423226, 400744
Halt	426805, 307840
Security risk	339377, 341829
Loss of bookmarks	477739, 478258, 478258, 477739
Cannot save files	263956
Crash during installation	369221, 364710, 285283, 261734
Unable to bookmark URL	264031
Temporarily unable to access bookmarks	452469, 414715
Waste of disk space	271883
No live bookmark functionality	398398
Limited bookmark information loss	473120, 377500
Unable to search bookmarks	336488
Unable to access all tabs	475031
Focus of URL bar must be gained by click	333651
News search does not work	402508
Search engines do not work in the toolbar	342540, 341908
Limited bookmark usability	434749, 330929
Completely trivial bugs	342110, 308743, 423226, 340167, 268144, 475030, 474964 333651, 266983, 360572, 337625, 317060, 258088

Table 2: Example User Profiles

Activity	Frequency (per week)	Length (min.)	Recovery Time (min.)
<i>User: Procurement Worker</i>			
Research offers online (e.g. compare prices)	10	30	5
Visit online shops	20	15	5
Order online	1	15	15
Use online auctions	5	15	5
<i>User: Secretary</i>			
Search for information online	10	30	5
Online banking	2	15	15
Use online dictionary	10	3	3
Online booking	2	15	15
<i>User: Marketing</i>			
Search online (e.g. competitor's pricing)	10	30	5
Create online polls	1	30	30
Update website information	2	15	15

and halts for each of the different user profiles. For one employee in the procurement department this amounts to 12.0 min, for secretaries to 8.3 min, and for marketing personnel to 9.5 min. Utilizing formula two, we arrive at a cumulated amount of failure-induced costs of \$ 49.67 per week, considering an average expenditure of \$ 100 per hour for one employee. The repair cost is zero in our analysis, as there are no repair actions besides the actions of the user. This is only a small amount, which is not surprising considering the small number of employees in this example. Still, over a longer period of time, a significant amount can be derived even for small organizations.

A deeper analysis of software failure impact is required in order to refine these results. The described dependability on Firefox for certain business tasks might drastically increase in the future. As the vision of the Software-as-a-Service (SaaS) paradigm gains more and more importance, so will the Web browser as its main delivery gateway enable users to access software via the web to support core business processes.

4.5 Assumptions and Limitations

Due to the lack of awareness for the problem of software failure induced costs in enterprises, only limited empirical data is available. An analysis of OSS is well suited to demonstrate the presented approach, as the source code and historical failure data are open to the public. On the other hand, commercial software may vary from OSS in the time to fix a bug and the possibility to explore alternative solutions.

Also it has to be taken into consideration that cost drivers are very domain specific, i.e. differ widely from one industry to the other. Still, this only affects the repair costs as elaborated in Sect. 3.4. Time lost is a universal cost driver affecting every kind of business, so regarding only time lost as a cost driver allows a fast comparison between domains.

Ideally the usage profiles would be verified based on empirical data, implying the need to capture user behavior in total. To arrive at a detailed user profile, one would have to record all user actions by observation or technical means, which is questionable due to legal implications. A trade-off has to be made, as this procedure would seriously violate the user's privacy. Self-report is afflicted with intended and unintended bias limiting the validity of the study. The update behavior of an user could also be included in the usage profile in order to identify which faults are present in the software the user is running. Here one has to consider the risk of updates introducing new bugs [22]. Usage profiles could also be constructed in a more detailed way including a larger number of tasks and a break-down into subtasks etc. Both issues however would overly extend the complexity of the usage profile and is hence excluded at this stage of development of our approach.

5. CONCLUSION

This work represents a first step towards a plausible cost model for software failure impact for the end user. We applied the presented approach to an example, utilizing the Web browser Firefox. After identifying the relevant errors and cost drivers for our analysis, we modeled the occurrence of failure effects in correlation with user behavior. In order to reduce complexity, we assumed a general failure proba-

bility. Based on these premises we calculated an estimation for user costs.

There are several lessons learned from our analysis. Bugs in Firefox leading to catastrophic impact are rare, even in the blocker category, and are usually eliminated prior to release. The remaining field failures have rather trivial impact. However, these minor effects accumulate over time and can become monetarily significant. We will therefore have to consider and classify low-level events to create a model that can also be applied under these conditions. Next to expected behavior, also uncommon user interaction will be incorporated into the next stage of the model development.

Our next steps include extending the presented approach with software error prediction techniques to enable a precise assessment of failure probabilities. Applying the model based on fault trees to a simulation environment will enable an empirical validation of the approach, comparing our predictions about the cost impact of field failures in business critical software to data collected in an industrial company. Simulation is certainly a suited instrument in order to incorporate dynamic aspects such as for example the change between versions. Thus our model will be beneficial for customers wanting to estimate the impact of software failures in their enterprise, as well as for software providers, enabling them to better anticipate their customers' needs.

6. REFERENCES

- [1] IEEE Standard Glossary of Software Engineering Terminology, Std 610.121990, 1990.
- [2] R. Amuthakkannan, S. M. Kannan, K. Vijayalakshmi, and V. Jayabalan. Managing change and reliability of distributed software system. *Int. J. Inf. Syst. Change Manage.*, 2:30–49, June 2007.
- [3] J. Anvik, L. Hiew, and G. C. Murphy. Coping with an open bug repository. In k, editor, *eclipse '05: Proceedings of the 2005 OOPSLA workshop on Eclipse Technology eXchange*, pages 35–39. ACM, 2005.
- [4] E. Arisholm, L. C. Briand, and E. B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, 83(1):2–17, Jan. 2010.
- [5] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secure Comput.*, 1(1):11–33, 2004.
- [6] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Software Eng.*, 22(10):751–761, Oct. 1996.
- [7] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 308–318. ACM, 2008.
- [8] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Software Eng.*, 20(6):476–493, June 1994.
- [9] M. Dowson. The ARIANE 5 software failure. *ACM SIGSOFT Software Engineering Notes*, 22(2):84, Mar.

- 1997.
- [10] K. O. Elish and M. O. Elish. Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, 81(5):649 – 660, 2008. Software Process and Product Measurement.
- [11] Eurostat. Data in focus 48/2008 - ICT usage by enterprises, 2008.
- [12] J. Ferzund, S. N. Ahsan, and F. Wotawa. Analysing bug prediction capabilities of static code metrics in open source software. In *IWSM/Metrikon/Mensura'08: Proceedings of the International Conferences on Software Process and Product Measurement*, volume 5338 of *Lecture Notes in Computer Science*, pages 331–343. Springer, 2008.
- [13] S. Frei, T. Dübendorfer, and B. Plattner. Firefox (in) security update dynamics exposed. *SIGCOMM Comput. Commun. Rev.*, 39(1):16–22, 2009.
- [14] T. L. Graves, A. F. Karr, J. Marron, and H. Siy. Predicting fault incidence using software change history. *IEEE Trans. Software Eng.*, 26(7):653–661, July 2000.
- [15] W. J. Gutjahr. Optimal test distributions for software failure cost estimation. *IEEE Trans. Software Eng.*, 21(3):219–228, Mar. 1995.
- [16] T. Gyimóthy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Trans. Software Eng.*, 31(10):897–910, Oct. 2005.
- [17] G. J. Holzmann. Conquering complexity. *IEEE Computer*, 40(12):111–113, Dec. 2007.
- [18] P. Hooimeijer and W. Weimer. Modeling bug report quality. In *ASE'07: 22nd International Conference on Automated Software Engineering*, pages 34–43, Nov. 2007.
- [19] L. Huang and B. W. Boehm. How much software quality investment is enough: A value-based approach. *IEEE Software*, 23(5):88–95, 2006.
- [20] Y. Jiang, B. Cukic, , and Y. Ma. Techniques for evaluating fault prediction models. *Empirical Software Engineering*, 13(5):561–595, 2008.
- [21] B. Kaiser, P. Liggesmeyer, and O. Mäkel. A new component concept for fault trees. In *SCS'03: Proceedings of the 8th Australian workshop on Safety critical systems and software*, volume 33, pages 37–46. Australian Computer Society, 2003.
- [22] S. Kim, E. J. W. Jr., and Y. Zhang. Classifying software changes: Clean or buggy? *IEEE Trans. Software Eng.*, 34(2):181–196, Mar. /Apr. 2008.
- [23] P. L. Li, J. Herbsleb, M. Shaw, and B. Robinson. Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB inc. In *ICSE'06: Proceedings of the 28th international conference on Software engineering*, pages 413–422. ACM, 2006.
- [24] C. E. Love, R. Guo, and K. H. Irwin. Acceptable quality level versus zero-defects: some empirical evidence. *Computers & Operations Research*, 22(4):403–417, Apr. 1995.
- [25] M. R. Lyu. Software reliability engineering: A roadmap. In *FOSE'07: 2007 Future of Software Engineering*, pages 153–170, May 2007.
- [26] W. A. Mandeville. Software costs of quality. *IEEE J. Sel. Areas Commun.*, 8(2):315–318, Feb. 1990.
- [27] T. Mende, R. Koschke, and M. Leszak. Evaluating defect prediction models for a large evolving software system. In *CSMR '09: Proceedings of the 2009 European Conference on Software Maintenance and Reengineering*, pages 247–250. IEEE Computer Society, 2009.
- [28] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Softw. Eng. Meth.*, 11(3):309–346, July 2002.
- [29] J. D. Musa and K. Okumoto. A logarithmic poisson execution time model for software reliability measurement. In *ICSE'84: Proceedings of the 7th international conference on Software engineering*, pages 230–238. IEEE Computer Society, Mar. 1984.
- [30] J. G. Myers. *Software reliability: Principles and practices*. Wiley, New York, 1976.
- [31] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *ICSE'06: Proceedings of the 28th international conference on Software engineering*, pages 452–461. ACM, 2006.
- [32] S. Ntafos and V. Pocciun-Benson. Improved testing using failure cost and intensity profiles. In *ASSET'00: Proceedings of the 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, pages 143–147. IEEE Computer Society, 2000.
- [33] T. J. Ostrand, E. J. Weyuker, and R. M. Bell. Predicting the location and number of faults in large software systems. *IEEE Trans. Software Eng.*, 31(4):340–355, Apr. 2005.
- [34] J. W. Paulson, G. Succi, and A. Eberlein. An empirical study of open-source and closed-source software products. *IEEE Trans. Software Eng.*, 30(4):246–256, Apr. 2004.
- [35] C. V. Ramamoorthy, G. S. Ho, and Y. W. Han. Fault tree analysis of computer systems. In *AFIPS'77: Proceedings of the June 13-16, 1977, national computer conference*, pages 13–17. ACM, 1977.
- [36] E. S. Raymond. *The Cathedral & the Bazaar*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, Oct. 1999.
- [37] A. Schiffauerova and V. Thomson. A review of research on cost of quality models and best practices. *International Journal of Quality & Reliability Management*, 23(6):647–669, 2006.
- [38] M. S. Sherriff, S. S. Heckman, J. M. Lake, and L. A. Williams. Using groupings of static analysis alerts to identify files likely to contain field failures. In *ESEC/SIGSOFT FSE companion '07: Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 565–568. ACM, 2007.
- [39] D. Spinellis, G. Gousios, V. Karakoidas, P. Louridas, P. J. Adams, I. Samoladas, and I. Stamelos. Evaluating the quality of open source software. *Electronic Notes in Theoretical Computer Science*, 233:5–28, 2009. Proceedings of the International Workshop on Software Quality and Maintainability (SQM 2008).

- [40] S. Wagner. Towards software quality economics for defect-detection techniques. In *29th Annual IEEE/NASA Software Engineering Workshop*, pages 265–274, 2005.
- [41] H. A. Watson. Launch control safety study, technical study. Technical report, Bell Telephone Laboratories, Murray Hill, NJ, USA, 1961.
- [42] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *MSR'07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 1. IEEE Computer Society, May 2007.
- [43] E. J. Weyuker. Difficulties measuring software risk in an industrial environment. In *DSN'01: Proceedings of the 2001 International Conference on Dependable Systems and Networks*, pages 15–24. IEEE Computer Society, July 2001.
- [44] E. J. Weyuker, T. J. Ostrand, and R. M. Bell. Using developer information as a factor for fault prediction. In *PROMISE '07: Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, pages 8–14. IEEE Computer Society, 2007.
- [45] Y. Zhou and J. Davis. Open source software reliability model: an empirical approach. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–6, July 2005.