Social Testing: A Framework to support adoption of continuous delivery by Small Medium Enterprises.

Jonathan Dunne
Hamilton Institute
Maynooth University
Email: jonathan.dunne.2015@mumail.com

David Malone
Hamilton Institute
Maynooth University
Email: david.malone@nuim.ie

Jason Flood
Institute of Technology
Blanchardstown
Email: jason.flood@itb.ie

***Abstract*—Continuous delivery (CD) represents a challenge for software test teams, because of the continuous introduction of new features and feedback from customers. We consider testing in a framework where users are encouraged to report defects through social or other incentive schemes. Using an enterprise dataset, we address the question of which types of defects can best be found in the field, allowing in-house test resources to be refocused. Validation of these touch points ultimately interweaves both customer and business needs. The proposed framework is one which can help small to medium software businesses, which typically have limited resources to test and release software via CD.**

I. Introduction

Small to medium enterprises (SME's) are the backbone of the European economy representing 79% of all employment with an annual turnover in excess of 440 billion Euro [1]. The European customer is maturing technologically and now demands more from their interaction with products and services. This has placed an additional challenge on the SME to provide products and services in a rapid and connected way. Nine out of ten SME's in Europe have less than ten employees [1] which makes it difficult for an SME to find the necessary additional capacity to cater for the new European customer. CD is seen as one approach that can be readily adopted by the SME to help reduce the software delivery life cycle. CD promotes faster delivery of software components, features and fixes [2]. With an accelerated delivery of product/service improvements, SME's want to keep pace with large enterprise solution providers. In the race to provide solutions in a dynamic agile way, large enterprises have the resources to exploit CD. These same enterprises can also leverage fully mature software test teams to ensure a succession of stable releases for the consumer and reduce the risk of subsequent brand damage due to releasing a poor quality product or service. The adoption of CD is non-trivial. Recent work has been conducted to outline the key challenges faced by software companies. These include: development of features rather than components and the development of an automated test harness to support testing [3]. An SME cannot compete at this level.

In this paper, we describe a framework, which the SME can leverage to best utilise their limited in-house test resources while utilising their greatest test asset: the customer. The core idea of this framework is for the in-house test teams to focus on high value test areas, while incentivising the customer to find low impact field defects. This paper contains a study of software defect data from a large enterprise dataset. Through this study of customer defect data we show which types of defects the customer is useful at finding. By leveraging the customer's skill at finding certain categories of defects, we argue that incentivising the customer to find a particular class of defect, can aid the SME to deliver higher quality software by diverting in-house test resources to high value areas such as performance and systems testing.

For cloud-based systems where multi-tenancy is employed, defect data could be shared socially between customers to better understand component usage patterns and fault prone functionality. The rest of the paper is structured in five Sections; Section II gives some description of study background and related works. Section III describes the enterprise dataset. Section IV discusses the analysis and method and it is followed by section V which explains the result. Finally, the conclusion and future work is described in Section VI.

II. Background and related research

A. Continuous Delivery

CD is an approach to software development that allows software companies to develop, test and release software in short, discrete delivery cycles. Releasing software with a low number of changes allows the rapid validation and release of a software product. CD Employs two methodologies; continuous test automation (CA) — the practice of employing an automated test script to validate delivered code and continuous integration (CI) — the practice of merging developer streams into a consolidated mainline, which allows software to be developed and tested to a high standard (due to the low level of code churn), and facilitates a swift release cycle. CD is used as part of a new wave of development, test, deployment and release strategies for Cloud based software services. Key evangelists for CD include Facebook, Google and Netflix [4].

B. Bug Bounty Programs

A bug bounty program is a scheme whereby software companies offer a reward to users that find defects within their software. The benefit to software companies is that it incentivises users to find defects (typically security vulnerabilities) before they are exploited by the general user base [5]. The *bugcrowd* website contains a list of current bug bounties

offered by software companies. Currently 116 companies are listed as having some form of reward and/or gift system for user found vulnerabilities [6]. Bug bounty schemes are not limited to start-up companies or open source projects. Some high profile software companies which participate in bug bounty schemes include Facebook, Google and Microsoft.

Recently there have been a number of famous bug bounties. For example Donald Knuth a computer scientist and creator of the TeX computer system [7], devised a bug bounty program (Knuth reward checks) where the reward doubled every year in value to a maximum of $327.68 in the form of a cashier's check. A second well-known bounty is related to D.J. Bernstein who is a cryptologist and programmer of qmail [8]. In 1997 he offered $500 to the first individual who could publish details of security exploits within his latest release of qmail. To date no one has found any vulnerability.

C. Other related studies

A number of studies have been conducted on customer reported defects. None of the software studied were developed using a CD release model.

Brooks and Robinson [9] performed a study on customer reported GUI defects found on two industrial software systems. Their work focused on the impact, location and resolution times of customer defects. Their study compared these factors from both in-house and customer defects. They found that in-house testers and customers found the same types of defects. 60% of the total defects found were in the GUI while the remaining 40% were application defects. Finally, that customers had to wait 170 days for defects to be fixed.

Moritz [10] conducted a study of customer defects raised within a large industrial telecommunications software component. Her work focused on analysis of customer defects found over a 10-year period with a goal of understanding how to improve future test quality. She reviewed whether defect regression, test phase, new functionality, load testing and environment were factors in a customer defect being raised. Her study found first, that the in-house system test environments and test use cases did not accurately match customer configurations or usage conditions. Second, that regression testing was inadequate, tests plans typically focused on new features, which left test exposures within legacy components. Finally, existing test methods were not suitable in finding customer defects.

Gittens et al. [11] studied the efficiency of in house software testing by investigating the scope and coverage of system and regression testing. They examined a number of factors, such as number of defects found in-house, by the customer and code coverage. Firstly with test coverage in the range of 71 – 80% fewer customer defects are found. Secondly that in-house tests coverage does not always overlap with customer usage areas. Thus there is a gap between in-house and customer product usage. Finally that greater in-house test coverage does not automatically translate into fewer customer defects found. The authors demonstrated that test coverage needs to be specifically targeted to reduce field defects.

Musa [12] developed a technique for Software Reliability Engineered Testing (SRET), which was implemented on the *Fone Follower* project at AT&T. Musa used a SRET method to classify defects found into four levels of severity based on their impact to the end user. Defect severity rates from prior regression testing were then used to guide future test coverage.

Sullivan and Chillarege [13] compared the types of customer defects found in Database Systems (DBS) and Operating Systems (OS). Their study looked at a number of factors including; error type, trigger and defect type. They had a number of key findings. Firstly they found that legacy DBS and OS had a similar number of high severity defects. Secondly, that newer DBS had a higher rate of high severity defects.

Adams [14] conducted a study of customer defects from nine products over a five-year period. He found that customer defects were typically discovered shortly after the product was released. He surmised that these defects would have taken many person months to find had they been tested on a single machine. He concluded that these customer defects would have been very difficult to find using existing test methods.

Existing research focuses on the impact of customer defects within a waterfall or agile development model. Additionally research also focuses on the challenges from the development side by adoption of CD. This paper will further the body of knowledge by adding data on the challenges faced by test teams as part of CD development and how field defect data can be used to optimise test coverage for SME's.

III. Data set

Defect studies have been shown to provide an effective way to highlight customer usage patterns of software. Defect studies can also aid businesses align their test coverage more towards customer based use cases.

The study presented in this paper examines approximately 1400 field defects from a large enterprise, cloud based system. The data was collected over a 12-month period (Jan - Dec) and is comprised of four main components: E-mail, Collaboration, Social and Business Support System (BSS). The systems have been deployed within three data centres and are used by customers globally. The software is developed in Java and runs on Linux. Product development follows a CD model whereby small amounts of functionality are released to the public on a monthly basis. For each defect we have access to the full defect report, but we particularly focus on the defect impact, defect component, data centre location and defect type. The following terminology will now be defined to provide clear context. These definitions are given in the glossary of IEEE Standards Collection in Software Engineering [15].

- Functional Testing: Testing which is focused on the specified functional requirements and does not verify the interactions of system functions.
- System Testing: Testing conducted on a complete integrated system to evaluate the system's compliance with its specified requirements. System test, unlike Functional testing, validates end-to-end system operations within the wider environmental context. Therefore system testing

should be conducted on an environment, which closely mimic's customer behaviour.
- Performance Testing: In software engineering Performance testing is performed to determine how a system performs in terms of responsiveness and stability under a particular workload.
- Field defects: Refers to all defects found by customers using the software product post-release.

This study aims to answer a number of questions. First, How do field defects impact the customers overall user experience? Second, what components are likely to yield field defects? Third, what data centres are likely to yield more field defects? Finally what types of defects do customers typically find? In order to answer these four questions, this study is broken down into the following attributes: defect impact, defect component, data centre location and defect type.

A. Defect Impact

A loss of functionality at either a system or client level is categorised as critical, major or minor. A critical defect can be defined as a defect where there is a loss of core functionality from either a server side component or from a client side perspective. A major defect can be defined as a defect where there is some loss of functionality but the loss is not system wide nor does the loss affect all end users. A minor severity defect can be defined as a defect with no loss of data, but some form of unexpected behaviour has occurred. Other ways in which the impact of the defect can be expressed is by the number of customers, who experience the same type of problem. Finally, it should be noted that defects of a similar type can vary in impact depending on whether they were raised as an in-house or field defect.

B. Defect Component

Understanding the location of field defects at a component level, gives an awareness of how customers use the product and more importantly what types of defects they are useful at finding. For example, in house test teams may design a set of tests, which will find a certain class of defect. Field defects can provide test teams with insight as to potential gaps in their coverage. Depending on the nature of these test gaps and the size of the test organisation, they may be difficult to close. For this study we categorised our software components as follows: e-mail, collaboration, social and BSS.

C. Data Centre Location

Understanding the location of field defects at a data centre level can highlight whether a specific data centre or high usage is a factor in the number of field defects raised. There are three data centres in our dataset: data centre A (High usage), data centre B (Low usage) and data centre C (Medium usage).

D. Defect Type

We consider three defect types: Functional, Performance and System. A functional issue may relate to behaviour observed directly by the customer, for example a component feature when used may either fail nor work entirely as expected. Performance defects fall into two main categories, client side and server side. For client side issues, an end user may experience an unresponsive or slow UI. Additionally a server side performance defect may be related to a sudden burst of user activity, which has undesirable performance impact for the entire system. System defects generally relate to a class of problem where either an end-to-end system workflow has failed. Or by virtue of having multiple concurrent users using the system at a given point in time has caused a feature or process to fail.

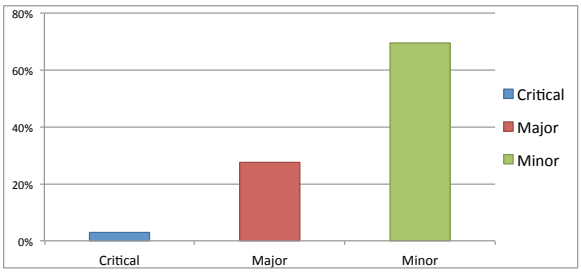

Fig. 1. % Field defects by severity

TABLE I

Severity	Critical	Major	Minor
% of Total	2.9%	27.5%	69.5%

E. Limitations of dataset

The dataset has a number of practical limitations, which are now discussed. Defect severity can vary depending on the support engineer filing the bug report or the customer logging the field defect. This subjectivity can lead to a different severity rating being assigned to the same type of defect.

While the field defect tracking application has a granular system to aid the classification by functional location, there are challenges in locating the parent area of a defect particularly when the defect displays errors in multiple subsystems. The authors reviewed the severity and the functional location of each defect. The goal was to ensure that each defect in terms of severity and categorisation remained constant.

The defects that form part of this study are from a large enterprise cloud system. The defects are applicable to the domain of email, collaboration, social and BSS.

IV. Results

We now explore the attributes of field defects observed.

A. Defect Impact

Fig.1 shows the percentage of the total defects broken down by severity. Minor defects are the most common with critical defects being the least common.

Field defects were classified by impact, which are shown graphically in Fig. 1 and textually in TABLE I. These show the percentage of all defects of each severity type. The majority of

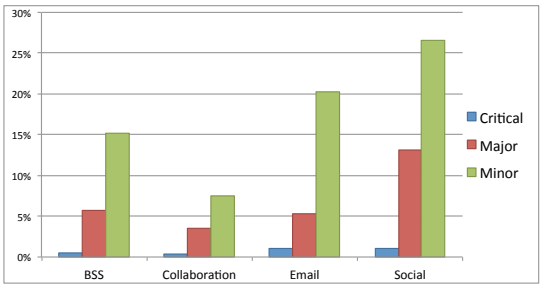

Fig. 2. % Field defects by component and severity

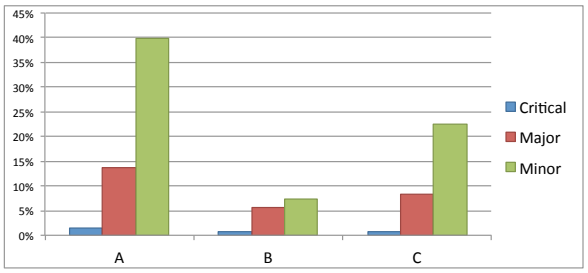

Fig. 3. % Field defects by data centre and severity

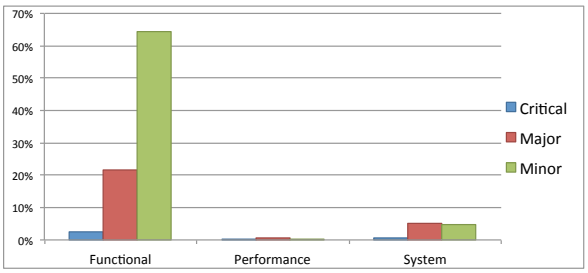

Fig. 4. % Field defects by test type and severity

TABLE II

Severity	Critical	Major	Minor	Total
BSS	0.5%	5.7%	15.2%	21.4%
Collaboration	0.4%	3.5%	7.5%	11.4%
Email	1.1%	5.2%	20.2%	26.5%
Social	1.0%	13.1%	26.5%	40.6%

defects found by the customer had a minor impact on their user experience (approximately 70%), while approximately 28% of users experienced a major severity defect and the remaining defects (just under 3%) were of a critical severity.

B. Defect Component

Fig. 2 shows the percentage of the total defects broken down by component and their severity. In each component minor defects are the most common with critical defects being the least common.

TABLE II. Shows the percentage of all field defects broken down by component and severity. The Social application contained the most defects (41%), Email (27%) and BSS (21%) had a broadly similar level of defects, and while the collaboration application had the least percentage number of defects found with 11%. The customer was most likely to find a minor severity defect irrespective of component used.

C. Data Centre Location

Fig. 3 shows the percentage of the total defects broken down by Data Centre and severity. In each data centre minor defects are the most common with critical defects being the least common. Previously it was noted that both centres A and C are high and medium usage, while data centre B is low usage. Given the level of field defects found in each data centre this supports the logical concept that higher usage leads to a greater number of defects.

TABLE III. Breaks down the Field defects by data centre and by severity. 55% of all field defects found were in data Centre A. data centre C recorded 31% of defects, while data Centre B recorded only 14% of defects.

D. Defect Type

Fig. 4 shows the percentage of the total defects broken down by testing type and severity. It was expected that minor severity would feature significantly, it's interesting to observe that the majority of functional defects are minor in nature. It was observed that for defects classified as System that the number of major and minor defects are practically the same. Finally it was noted that the customer did not find very many Performance defects. However of the defects that were found, slightly more were major than minor severity.

TABLE IV shows the percentage of field defects found according to their type. 89% of all customer issues reported were functional in nature, a further 10% of customer defects were related to end to end system issues, while only 1% of defects found were related to performance of either client interface or underlying server system.

V. Discussion

Section IV provided an outline of field defects that were studied as part of our overall dataset, including defect impact, defect component, data centre location and defect severity. The following section provides deeper analysis of the results. In

TABLE III

Data Centre	Critical	Major	Minor	Total
A	1.6%	13.6%	39.8%	55.0%
B	0.7%	5.6%	7.3%	13.6%
C	0.6%	8.3%	22.4%	31.3%

TABLE IV

Field Defect Type	Critical	Major	Minor	Total
Functional	2.4%	21.8%	64.3%	88.5%
Performance	0.1%	0.7%	0.4%	1.2%
System	0.5%	5.0%	4.7%	10.3%

each section references will be made to each research question asked in section III.

A. Defect Impact

To answer the question how do field defects impact the customers overall user experience, Fig. 1 and TABLE I, clearly show that the customer finds more minor defects than any other type, almost 70%. This means that the customer will come across some unexpected behaviour which does not result in data loss during their day to day product usage. Given that minor severity defects are found most often, the logical conclusion is that these types of defects are found by the customer exercising the most common component use cases. With the level of major defects found being 28%, this also shows that these defects were found as part of a typical customers day to day usage, albeit to a lesser degree than minor defects. Clearly as part of the customers typical use case, they did encounter some form of data loss or non-trivial unexpected behaviour. With approximately 3% of field defects being critical, this suggests that the likelihood of a customer experiencing a total component or some system wide failure is a rare event. That said customers, either by themselves or in conjunction with other users, were able to bring about behaviour, which impacted the wider system.

Given the nature of CD, it is important to point that new code and features are released frequently. Therefore by it's nature new field defects are raised on a continuous basis once new features are delivered. If a bug bounty program were introduced to incentivise field defect discovery, it would be interesting to determine the increase in the velocity of field defects from the introduction of such a scheme. Typically bug bounties have been the preserve of security defect discovery. By rolling out such a scheme for field defect discovery in general, it would be valuable to both the software developer: to focus on testing code paths which lead to higher severity issues, while incentivising the customer to uncover lower priority defects.

B. Defect Component

Examining defect component, gives an indication of where customers are likely to find defects within each component.

Fig. 2 and TABLE II highlight that, at a component level approximately 41% of the field defects found were in the social component. A further 27% found in the email component, with 21% in the BSS component with a final 11% found in the collaboration component. While defect yields may not map directly to application usage (Unfortunately, no application usage metrics were available), conditional probabilities were calculated to determine like likelihood of certain combinations of defect attributes being found. With minor field defects being raised most often, conditional probabilities for each component at minor impact level were calculated. It was noted that P(minor|email) had the highest probability with 0.762, P(minor|BSS) with 0.709, P(minor|collab) with 0.660 and P(minor|social) with 0.654. This tells us that the customer is more likely to find a minor impact field defect in the email component.

This may seem counter intuitive given that 41% of field defects were found in social. It is concluded that given the lower level of major impact defects found in email increases the likelihood of a minor impact defect being found. The logical conclusion is that the more a component is used the more defects are likely to be found by end users. It is proposed that for popular components (in our study email & social) that have continuous feature releases, by incentivising the discovery of lower impact defects by the customer, can help in-house testing refocus their efforts on the major severity testing across all test disciplines in their key feature/components areas.

C. Data Centre Location

Fig. 3 and TABLE III give an insight into field defect breakdown by data centre. As mentioned in section III, it is known that the level of usage varies from data centre to data centre, interestingly the customers of data centre A (High Usage) reported the highest number of field defects with 55% while data centre C (Medium Usage) and data centre B (Low Usage) had 31% and 14% defects raised respectively. Clearly there may be some form of correlation between concurrent user population and field defects raised.

Checking conditional probabilities for each data centre for both minor and major defects, as follows P(Minor|DC-A) and P(Major|DC-A) gives 0.724 and 0.248 respectively. These conditionals state that the customer is more likely to find a minor defect within data centre A. For data centre B the following conditionals were calculated; P(Minor|DC-B) and P(Major|DC-B), which gives 0.537 and 0.411 respectively. These conditionals tell a similar story to that of data centre A, that the customer is more likely to find a minor defect than a major one. It is conjectured that the customer use case on data centre B is different to that of the other two data centres. Further analysis should be employed by the in-house test teams, to ensure their test scripts cover the main customer use case in data centre B which generates major impact field defects. Finally checking P(Minor|DC-C), P(Major|DC-C) gives 0.714 and 0.265 respectively. These conditional probabilities are very similar to those of data centre A. Customers are almost three times as likely to encounter a minor impact defect on data centre C than that of a major impact field defect.

Overall it was found that that for high and medium usage data centres the likelihood of finding minor field defects was almost three times that of finding a major impact defect. For the low usage data centre the probability of finding a major impact field defect was broadly similar to that of finding a minor impact field defect. Finally the customer was less effective at finding high severity defects irrespective of the data centre used.

In the context of CD, the same code is released to each data centre; clearly customers are more likely to be impacted differently depending on data centre. Knowing the underlying

customer data centre use case is key. With knowledge of both data centre usage and field defect data, incentivisation schemes can be tailor-made according to data centre. One suggestion would be a bounty to target minor field defects on high usage data centres, while refocusing in-house resources to find more major impact defects prior to release.

D. Defect Type

Finally in order to understand which type of field defect that the customer typically finds, defect type was examined. This metric may be one of the most important in terms of this study. It helps underscore which class of defect the customer is proficient at finding.

Fig 4 and TABLE IV clearly indicate that functional defects are the most commonly uncovered by the customer with 89% of all defects found being functional. Also of significance is the severity of these defects with 64% being minor severity. Clearly functional defects typically present themselves in the form of user experience behaviour errors where the end user attempted an operation and the behaviour encountered was unexpected. It is also important to note that 10% of all issues were system errors, typically these manifest themselves as unexpected behaviour during active concurrent usage. One can infer that system errors are less common than functional ones. It may also be the case that system errors do not readily manifest themselves to the end user in the same way as functional defects.

Performance defects ranked the lowest in overall defects found with only 1% of all problems being attributed to performance defects. This data suggests that either the performance of each component was adequately tested prior to release or that performance defects may be harder for the end user to measure and quantify once in the field.

Clearly from a customer's perspective they are more likely to find functional defects as these issues are found within the UI, however the customer finds a greater proportion of lower impact functional defects. Additionally the customer was less effective at finding high severity Performance and System field defects. From a CD/CI perspective, a balance needs to be struck in terms of the features being released and their likely defect type yield. For backend server features in-house test teams can focus almost exclusively on Systems and Performance testing. For features with rich functionality in-house test teams can focus on use cases, which are likely to yield critical and major defects with some additional minor impact areas.

In terms of bounties, for releases with high functional content software developers could award triple / double and single prizes for critical, major and minor impact field defects respectively with the knowledge that adequate testing was conducted in-house for critical and major use cases. Similarly for releases with high System and Performance and low functional features, proportional bounties may be awarded.

VI. Conclusion

Previous studies have shown that analysis of field defects is a valuable exercise. Additionally that bug bounty rewards provide an incentive to end users to improve software quality once in the field. The purpose of this study was to examine the role of the customer in the generation of field defects. It was found that the customer was quite adept at finding minor severity functional defects. The findings of this study support previous work particularly in the gaps between in-house software testing and general customer usage.

This work provides a more detailed study in relation to software developed using a CD model. Adoption of CD means continuous feature releases and continuous defects, however feature releases can be delivered in such a away to ensure that there is not a significant burden on in-house test teams.

In future SME's can assess their field defect data to understand the core gap areas in relation to defect impact, component, data centre and type. A specific test framework can then be built to allow SME's to focus on test areas, which are likely to yield high impact defects, and which may be difficult for an end user to discover. Furthermore by providing the customer with an incentivised scheme, in the form of bug bounties to find specific defects types will improve overall software quality through the iterative development and release process that is CD. In future work we shall assess the framework behaviour in relation to in-house defect deferral rates and the number of field defects raised.

References

[1] Annual report on European SME's. [Online]. Available: http://bit.ly/1PoGGVz

[2] J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson, 2010.

[3] H. H. Olsson, H. Alahyari, and J. Bosch, "Climbing the 'stairway to heaven'–a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software," in *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*. IEEE, 2012, pp. 392–399.

[4] Best examples of companies using continuous deployment. [Online]. Available: http://bit.ly/1OZQ5SP

[5] (2015) Bug bounty program. [Online]. Available: https://en.wikipedia.org/wiki/Bug_bounty_program

[6] The bug bounty list. [Online]. Available: https://bugcrowd.com/list-of-bug-bounty-programs

[7] D. E. Knuth. Homepage. [Online]. Available: http://www-cs-faculty.stanford.edu/~uno/

[8] D. J. Bernstein. homepage. [Online]. Available: http://cr.yp.to/djb.html

[9] P. Brooks, B. Robinson, and A. M. Memon, "An initial characterization of industrial graphical user interface systems," in *Software Testing Verification and Validation, 2009. ICST'09. International Conference on*. IEEE, 2009, pp. 11–20.

[10] E. Moritz, "Case study: how analysis of customer found defects can be used by system test to improve quality," in *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*. IEEE, 2009, pp. 123–129.

[11] M. Gittens, H. Lutfiyya, M. Bauer, D. Godwin, Y. W. Kim, and P. Gupta, "An empirical evaluation of system and regression testing," in *Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research*. IBM Press, 2002, p. 3.

[12] J. D. Musa, "Software reliability-engineered testing," *Computer*, vol. 29, no. 11, pp. 61–68, 1996.

[13] M. Sullivan and R. Chillarege, "A comparison of software defects in database management systems and operating systems," in *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*. IEEE, 1992, pp. 475–484.

[14] E. N. Adams, "Optimizing preventive service of software products," *IBM Journal of Research and Development*, vol. 28, no. 1, pp. 2–14, 1984.

[15] I. C. S. S. E. S. Committee and I.-S. S. Board, "IEEE recommended practice for software requirements specifications." Institute of Electrical and Electronics Engineers, 1998.