Social dogfood: A Framework to minimise Cloud field defects through crowd sourced testing.

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

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

***Abstract*—Delivering software for the Cloud represents a challenge for both micro teams and Small Medium Enterprises (SMEs), in part due to the rapid release methods adopted and the numerous ways in which software defects can be detected. We study field defect detection rates in a framework where these rates are used to refocus in-house test resources. Using an enterprise dataset, we address the question of what types of defects are found in the field and how soon after a system goes live defects are detected. Our framework can aid both micro teams and SMEs to minimise the number of defects found in the field by maximising internal usage through 'Dogfood' programs and by leveraging crowdsourced test methodologies.**

I. Introduction

Cloud computing is seen as a way for an SMEs to compete with larger companies, in terms of the rapid delivery of software and services. This is due in part to the 'always on' nature of Cloud computing. European SMEs are beginning to see the green shoots of economic recovery, in 2015 SME economic growth grew by 5.6% [1]. Additionally SME employment growth grew by 1.5% [1]. As the economic recovery continues SMEs are looking to maximise their potential.

However both micro-teams and SMEs face continuing challenges when adopting both Cloud computing and continuous delivery as their hosting and software delivery mechanisms respectively. A recent study has outlined the issues facing SMEs in adopting industry standards in relation to software development and delivery [2]. Furthermore almost all SMEs (93%) employ less than 10 people [1], therefore for this study, we analyse the factors that may impede the rapid delivery of high quality software for teams with low levels of resourcing.

At regular intervals throughout the past 20 years software quality commentators have discussed the need for companies to extensively use the software they develop prior to release to the customer. This practice is known as "Eating your own dogfood" [3]. A recent article in Forbes by the CEO of Lua (a startup company) impressed the need for companies to continuously use their software prior to release [4]. A leaked memo from the CEO of Yahoo lamented the fact that only 25% of employees were willing to use Yahoo mail as their corporate email reader [5].

In this paper we propose a framework that both micro teams and SMEs can use to deliver high quality software to the Cloud, while utilising their limited resource cohort. The core idea of this framework is for software test teams to amalgamate based on the types of defects found coupled with regular internal usage of their own software and additional crowdsourced test methods. For micro teams with a limited pool of test resources, leveraging a crowdsourced team of test resources can aid defect detection.

This paper contains research conducted on a large enterprise dataset of both in-house and field defects. Through study of this data we investigate whether there is a) an overlap between the types of defects that in-house test teams find and b) is there an overlap between the types of defects a customer finds in the field compared to that of in-house teams. Using the results of this study for our framework, a crowd-sourced dogfood program and an in-house test team alignment can be used to reduce the number of defects found in the field.

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 provides analysis and methodology. It is followed by section V that explains the result. Finally, the conclusion and future work is described in section VI.

II. Background and related research

A. *Eating your own dogfood*

Eating your own dogfood is a term given to the internal usage of a software product prior to release to the customer. The idea is that regular internal usage will improve overall software quality.

Warren Harrison [6] the then editor in chief of IEEE Software, mentions that Microsoft were one of the first companies to aggressively adopt the practice of "Eating your own dogfood" when developing their Windows platform in the early 1990's. Harrison also discusses the pros and cons of adopting a dog food approach to internal testing.

Adam Moskowitz [7] discussed the idea of dog fooding in the magazine ";login:". In the realm of system administration, Moskowitz provides some practical examples of how increased internal testing can help improve shell scripting and tooling.

Schmidt and Varian [8], in a Newsweek article, outlined ten rules, which they believe will drive success within Google over the next quarter of a century. They attribute the success of Gmail to the fact that it was extensively tested by the majority of Google employees over a several month period.

Prlić and Procter [9] in a Public Library of Science computer biology journal, also outline ten rules from the open development of scientific software. Rule three mentions how software in development should be useful as an end product and not simply to demonstrate a solution. In other words the software should be consumable by customers with a broad range of backgrounds rather than a specific cohort.

Jackson and Winn [10] conducted research in the field of research data management platforms. In building a large complex platform with many API endpoints, they cite internal usage of the in-development platform coupled with adoption of agile practices such as 'Continuous Integration' [11] as key methods in defect detection.

B. Crowdsourced testing

Crowdsourcing is the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call [12].

Nebling et al. [13] presents a study of Crowdstudy, a toolkit for crowdsourced testing of web pages. By crowdsourcing numerous individuals, data was collected on how users habits differed when engaging with a web site.

Vukovic [14] conduced a study of crowdsourcing services for the Cloud. While Amazon's Mechanical Turk and Innocentive appear to have the most supported features, most of the frameworks fall short in facilitating the dynamic formation of globally distributed teams.

Liu et al. [15] conducted a study into the use of crowdsourcing for usability testing compared to traditional face-to-face methods. Their study found the quality of results from crowdsourcing were not as good as those face-to-face testing. However crowdsourcing still represents value for design/development teams with limited time and money.

Zogaj et al. [16] present a case study with a crowdsourcing company who specialise in outsourcing software testing to specific groups. Their research found that there were three key challenges: managing the process, managing the crowd and managing the technology. By using an intermediary to manage all aspects of the process from procurement of individuals, monitoring of test progress to addressing technology skill gaps ensured a smooth end to end process.

C. Other related studies

Riungu et al. [17] performed research into the challenges Cloud computing presents to software testing. One concern raised was the human effort to test software with 24/7 availability. Automation aside, they mentioned the need for some level of manual testing to be conducted round the clock—an idea not easily implemented by SMEs.

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 [18]. The *bugcrowd* website contains a list of current bug bounties offered by software companies. Currently 329 companies are listed as having some form of reward and/or gift system for user found vulnerabilities [19]. 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.

III. Data set

Field defect studies have been shown to provide a useful way to infer gaps within in-house testing [20] [21]. Such studies, in conjunction with our framework, can be adopted by micro teams and SMEs to determine common failure types.

The study presented in this paper examines approximately 2000 defects (both field and in-house) from a large cloud based real-time collaboration system. The data was collected over a 22-month period (Jan '15 – Oct '16) and is comprised of three main components: Instant Messaging, Web Conferencing and an interactive audio and video component.

There are four in-house test teams whose function is to find software defects; Function test, Performance test, Security and System test. Defects are also found by Development, DevOps, Support, Accessibility and well as other general users. Field detects are found by either the customer or any of the in-house teams just mentioned. Defects are typically of one of four types: Functional, Performance, Security and System. Defect severity is categorised as either: Critical, Major or Normal. 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. Each release takes place on a Saturday.

Product development follows a continuous delivery (CD) model whereby small amounts of functionality are typically released to the public approximately every four to six weeks. For each defect we have access to the full report, but we particularly focus on the defect severity, defect type and found by whom. 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 [22].

- 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 mimics 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 either customers or internal teams using the software product post-release.

This study aims to answer the following questions. First, what group is most likely to find either an in-house or field defect based on defect severity and test type? Second, what is the field defect discovery rate during the first fourteen days of a release? In order to answer these questions our study is divided into the following two subsections: defect discovery probability by team and field defect discovery rates.

A. *Defect discovery probability by team*

A question to software companies is, what types of defect are found both internally and externally. Similarly, what are the most common severity types found in-house and in the field? By analysing the types of defects found both internally and externally, a map can be built to determine which teams are most effective at finding a particular class of defect. Likewise for defects found in the field a similar map can be drawn to determine what types of defects a customer is good at finding. Given the limited resources of both micro teams and SMEs, an intersection of both maps could be used by internal test organisations to a) realign their test organisation to discover more defects and b) pivot internal test practices to more customer based usage patterns.

B. *14/28 days later – defect discovery rates*

Studying when defects are found in the field gives us knowledge of how reliable software is. In the field of system reliability, there is the idea that failures (defects) may follow a specific pattern which can be represented in the form of a ‘bathtub curve’ [23] [24]. The patterns can be summarised as follows: 1) Decreasing failure rate (early failures), 2) Constant failure rate (random failures) and 3) Increasing failure rate (wear-out failures). Of interest is to understand if field defects failures found within the first twenty eight days conform to these characteristics. Of additional interest is to understand what types of defect (and their associated severity) occur within the first two weeks of a release.

C. *Limitations of dataset*

The dataset has a number of practical limitations, which are now discussed. While the defect tracking system allows for a granular categorisation system, whereby field defects can be mapped to a specific release. There were a number of field defects, which were mapped to an incorrect release. The authors used the defect creation date to determine which field defects belonged to which release.

The defect reports that form part of this study are from an enterprise Cloud system. As a result the analysis may not be relevant outside of these fields.

IV. Results

We now explore the results of our analysis.

A. *Defect discovery probability by team*

Fig. 1 shows a bubble plot of in-house defect detection probability grouped by team type and defect severity. The size of the bubbles are scaled relative to the probability. The greater the probability the larger the bubble diameter. We also show a

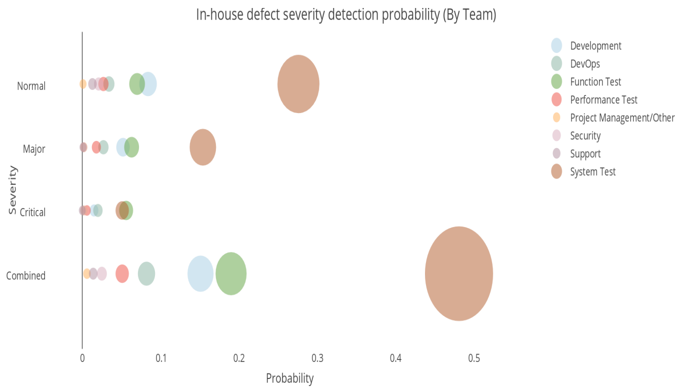

Fig. 1. In-house defect severity detection probability (By Team)

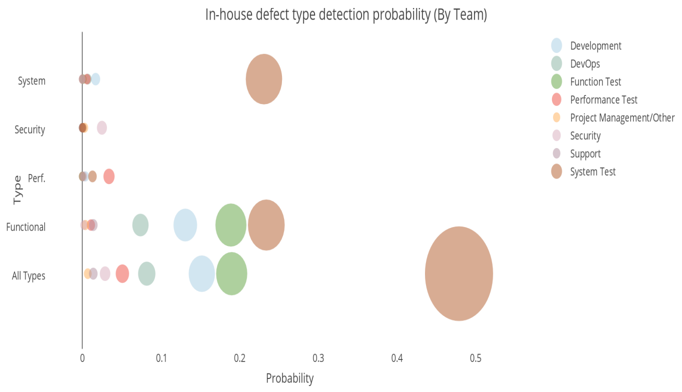

Fig. 2. In-house defect type detection probability (By Team)

probability by combined defect severity. The System test team have highest probability of finding a normal or major defect. System test are also most likely to find a defect of any severity. Function test are most likely to find critical defects.

Fig. 2 shows a bubble plot of in-house defect detection probability grouped by team type and defect type. We also show the probability of all defect types. The System test team have highest probability of finding a System, Functional and combined defect type. The Performance team are most likely to find a performance defect. Likewise the Security team are most adept at finding security defects.

Fig. 3 shows a bubble plot of field defect detection probability grouped by team type and defect severity. We also show a probability by combined defect severity. The customer is most likely to find a defect of any given severity in the field.

Fig. 4 shows a bubble plot of field defect detection probability grouped by team type and defect type. We also show a probability of all defect types. The Customer is the most likely group to find any a defect of any given severity in the field.

B. *14/28 days later – defect discovery rates*

Fig. 5 shows a line plot of percentage field defects raised during the first fourteen days of a release. The percentage values are an aggregate over the entire fifteen releases studied.

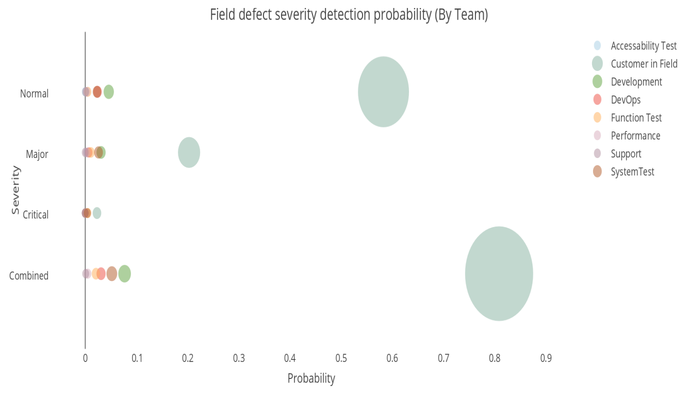

Fig. 3. Field defect type detection probability (By Team)

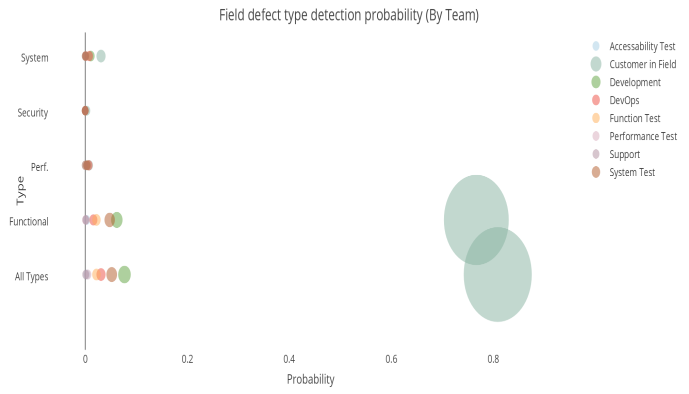

Fig. 4. Field defect type detection probability (By Team)

Additionally the percentages are calculated as follows: an aggregate daily rate divided by the total number of field defects found. Days five and seven saw the highest aggregate percentage of field defects raised, while days eight and nine saw the lowest aggregate percentage field defects detected.

Fig. 6 shows a bar plot with fitted curve of percentage field defects raised during the first four weeks of a release. The percentage values are an aggregate over the entire fifteen releases studied divided by the total number of field defects found in the first twenty eight days of a release. Week one saw the highest number of field defects raised with just over 31%, while weeks two and three had near identical rates with approximately 21%, while week four had an increased rate (over weeks two and three) of almost 27%.

V. Discussion

Section IV provided an outline of both in-house and field defects that were studied as part of our overall dataset, including defect detection by team and field defect detection in the field within the first fourteen days of a release. The following section provides deeper analysis and discussion of the results. In each section references will be made to each research question asked in section III.

A. Defect discovery probability by team

As mentioned in Section VI, we wanted to understand, for in-house defects, how defect detection was distributed across each internal test team by defect severity and type. Fig. 1 shows that the System test team was more likely to find both Normal (0.28) and Major (0.15) severity defects while the Function test team was slightly better at finding Critical defects (0.06). Overall the System test team were more likely to find a defect when we collapsed severity into a combined severity ranking (0.48).

Interestingly, for defect type Fig. 2 illustrates that, the System test team were most likely to find System and Functional defects (both 0.23), while the Performance and Security teams found the most performance and security related defects. Overall when all defects types are reduced to a single category the System test team are most likely to find a defect irrespective of type (0.48).

Fig. 3 shows that the customer is most likely to find a defect in the field, irrespective of severity (0.81). It's worth noting that for normal severity defects the customer has a probability of 0.58 while the next highest is the development team with a probability of 0.05.

Fig. 4 highlights that the customer is highly skilled in detecting functional defects (0.77) and has a low probability of finding other types of defects: Performance (0.01), Security (0) and System (0.03). For non functional defects, internal consumers (i.e. test teams) do find "field defects" as part of their daily usage of the software. Development (0.01 System defects) and DevOps (0.01 Performance and System defect types).

A number of interesting points are raised by analysis of both sets of defect data. Firstly that the *System test team have the highest probability of finding a defect in-house when severity or type are reduced to a single category*. What is surprising though is that while the *Performance, Security and System teams are most adept at finding homogeneous defect types*, the *Function test team is less likely to find a Functional defect (0.19) compared to System Test*. It is worth noting that the Development team are also quite skilled at flushing out functional defects too (0.13). Second that the Customer is more likely to find a field defect than internal users "Dogfooding" their own software. That said, the customer appears to find a very specific type of defect, a normal severity functional defect.

Given the overlap in terms of detection of Functional defects there is an argument to merge both Function and System test teams into a single group. Based on the dataset we know that the System test team are skilled at finding functional defects, by merging teams there would be the added benefit of upskilling the functional team members to test and detect System defects. Furthermore, it makes sense to have a separate team to test both the Performance and Security areas of the product because this testing requires specialist knowledge which enables these test teams to focus on their core areas of expertise.

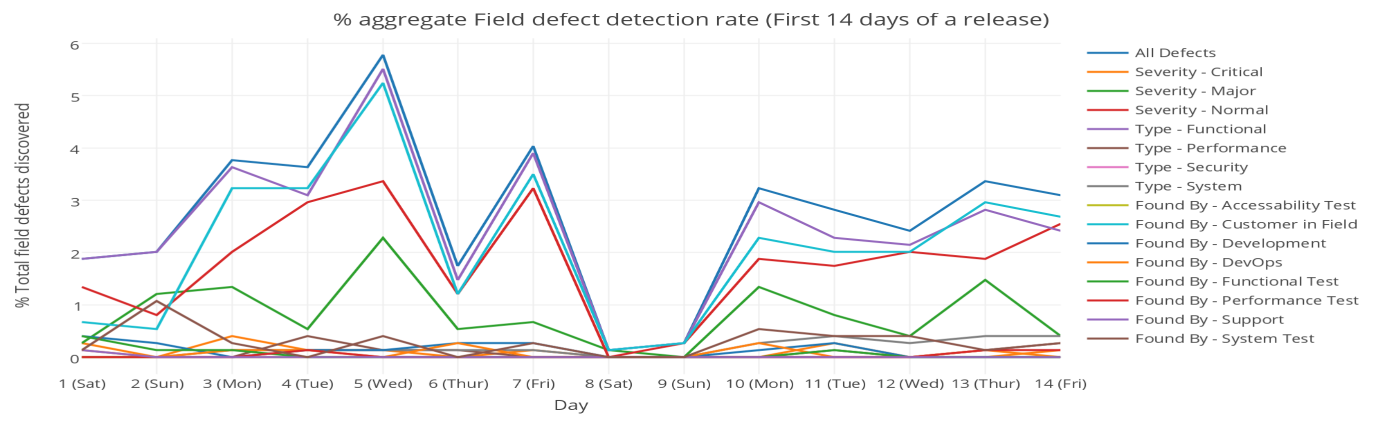

Fig. 5. % aggregate field defect detection rate (First 14 days of a release)

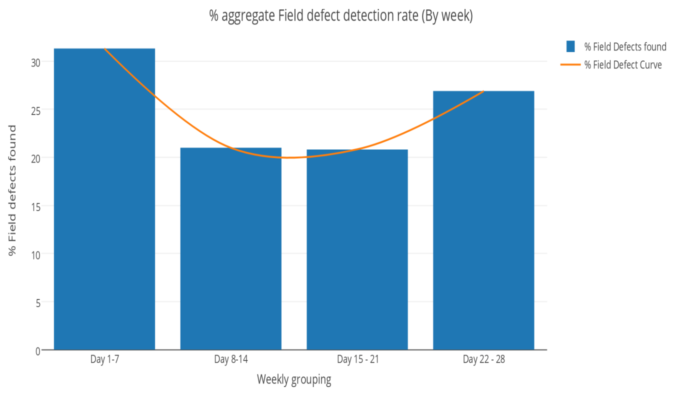

Fig. 6. % aggregate field defect detection rate (By week)

With the customer being so adept at flushing out normal severity functional defects, focus instead should be placed on determining what test paths should be included in future test cases to capture these classes of functional defects as part of an internal testing. However the customer should be incentivised in some manner for the number of lower severity defects they find. We discuss this subject matter in more detail in prior work [25].

Finally, irrespective of the dataset and the results tied directly to it, we suggest the following outcomes for our framework: a) test teams should be aligned based on the types of defects they find, b) in-house testing should prioritise test paths to areas of the product, which are both critical path and where the customer is least likely to find a defect and c) schemes to incentivise the customer to find low severity defects could be introduced.

B. 14/28 days later – defect discovery rates

Examining the field defect discovery rate during the first twenty eight days of release helps us to understand what types of defects a customer is likely to find.

Fig 5. shows the percentage of defects found the first fourteen days of a release. Looking at the 'All Defects' line initially we can clearly see that field defect detection peaks at day five. Approximately 6% of all field defects (raised in the first twenty eight days of a release) were found on the fifth day of a release. The second highest daily rate appears on day seven with 4%. Also of note is that, on days eight and nine, we observed the least number of field defects raised (0.13% & 0.27% respectively). This may be attributed to both of these days falling on a weekend. That said the first two days of a release also fall on weekend and see a higher rate of field defects raised (Approximately 2% on both days). Finally as part of the second working week of a release the rate rises to approximately 3% and remains steady at this rate for the remainder of the second week.

Another point of interest is evident in Fig. 5. If we consider Normal & Major severity, Functional type and Customer found field defects we can see how close these lines mirror the overall rate line. This confirms our intuition that these categories of field defect are highly correlated as part of the overall data set. Our reason for this belief is that, these four categories of defect contribute a significant proportion to the overall dataset. No formal regression analysis has been conducted to confirm our intuition.

Fig 6. Presents a bar plot with fitted spline curve. This plot illustrates the percentage of field defects raised weekly during the first twenty eight days of a release. Clearly we can see that the highest percentage of field defects detected within the first month of a release are found in the first week of a release. 31% of all field defects found in the first month of a release are found during the first seven days. Also of note is the rate drop during weeks two and three of release, approximately 21% for both weeks. Finally it is worth noting that the rate increases to approximately 27% during week four.

Looking through the lens of analysis from both a fortnightly and monthly perspective, we can clearly see that highest percentage of field defects are found within the first seven days of a release (specifically day five). There may be a host of reasons why the customer finds so many defects in such early stage of a release: Poor customer usecase profiling, lack of test automation, lack of test resources. Irrespective of the root causes it is worth noting that the majority of field defects the customer uncovers are low severity and functional. Therefore a targeted set of crowdsourced tests would be useful in flushing our additional defects prior to public release.

In Section III we mentioned the idea of software reliability and how the 'bathub curve' is used (at a high level) to illustrate the three stages of system reliability. We observed in Fig 6. the weekly percentage rate of field defect discovery. We noted the high initial rate in week one, the drop in weeks two and three and finally the increased rate in week four. This behaviour can be mapped directly to the early, random and wear-out (i.e. failure that occur due to sustained usage) failures, which characterise reliability. Targeted survival analysis of field defects found in the first month of a release is required to determine whether field defect do indeed contain early, random and wear-out attributes. We temper this finding with the observation that there is an baseline of at least 20% failure rate on any given week.

In future SMEs and micro teams can adopt a two week crowdsourced–dogfood test program prior to general release. By leveraging the power of both the internal work force and a crowdsourced test cohort for a fixed duration, companies can uncover many 'field' defects prior to general release. Certainly this crowdsourced–dogfood program will uncover many defects with early and random characteristics. Based on the analysis of in-house test data, these types of defect are difficult to uncover as part of internal testing. For defects with wear-out attributes by adopting a framework of survival analysis, teams can determine if there are specific components which wear out more quickly than others. With this knowledge development teams can adopt remediation plans to ensure their components are more robust to wear-our failures.

Putting the findings drawn directly from our dataset to one side, we can add the following proposals to our framework: a) a period of crowdsourced-dogfood testing should be conducted for a two week period prior to release, b) the crowdsourced team should contain individuals from a variety of backgrounds.

VI. Conclusion

The purpose of this study was to examine in-house and field defects for a Cloud based collaboration application. We found that Performance and Security defects are specialist focus areas which are best handled by specific teams. We also found a significant overlap in the area of Functional defects found by the System test team. Additionally the customer is skilled at finding specific defects types (Normal Functional) in our data set.

Furthermore we found that field defect detection occurs most often in the first seven days of a release and that over a monthly period field defect detection rates mirror those seen in other fields of reliability engineering.

In future SMEs can align in-house test teams to overlapping test areas to help focus test effort. Additionally for field defects of high severity re-focusing of test plans to high impact areas can mitigate their detection in the field. A reward/gamification framework can be introduced to customers who find low severity defects in the field. Likewise a co-ordinated system of crowdsourced dogfood testing can be implemented prior to release, to reduce the number of defects found in the first two weeks of release.

In subsequent work we shall assess our framework in conjunction with each feature component, to understand which components are most likely to break once released into the field. This future work can aid micro teams and SMEs to continuously pivot to further reduce defect detection rates from customers within the field.

References

[1] P. Muller, S. Devnani, J. Julius, D. Gagliardi, C. Marzocchi, R. Ramlogan, and D. Cox. (2016) Annual report on European SMEs 2015/2016. [Online]. Available: http://bit.ly/2kBHgGF

[2] O. T. Pusatli and S. Misra, "A discussion on assuring software quality in small and medium software enterprises: An empirical investigation," *Tehnički vjesnik*, vol. 18, no. 3, pp. 447–452, 2011.

[3] (2016) Eating your own dog food. [Online]. Available: http://bit.ly/2kHVTHP

[4] (2014) Not eating your own dog food? You probably should be. [Online]. Available: http://bit.ly/2k64eWH

[5] (2013) Clinging to outlook. [Online]. Available: http://bit.ly/19dZ40P

[6] W. Harrison, "Eating your own dog food," *IEEE Software*, vol. 23, no. 3, pp. 5–7, 2006.

[7] A. Moskowitz, "Eat your own dog food," *; login:: the magazine of USENIX & SAGE*, vol. 28, no. 5, pp. 18–19, 2003.

[8] E. Schmidt and H. Varian, "Google: ten golden rules," *Newsweek http://www. msnbc. msn. eom/id/10296177/site/newsweek/print/1/displaymode/1098*, 2005.

[9] A. Prlić and J. B. Procter, "Ten simple rules for the open development of scientific software," *PLoS Comput Biol*, vol. 8, no. 12, p. e1002802, 2012.

[10] N. Jackson, J. Winn *et al.*, "Eating your own dog food," 2012.

[11] (2017) Continuous integration. [Online]. Available: http://bit.ly/1Ty3ZfV

[12] (2017) Crowdsourcing: A defintion. [Online]. Available: http://bit.ly/QwOkEh

[13] M. Nebeling, M. Speicher, M. Grossniklaus, and M. C. Norrie, "Crowdsourced web site evaluation with crowdstudy," in *International Conference on Web Engineering*. Springer, 2012, pp. 494–497.

[14] M. Vukovic, "Crowdsourcing for enterprises," in *Services-I, 2009 World Conference on*. IEEE, 2009, pp. 686–692.

[15] D. Liu, R. G. Bias, M. Lease, and R. Kuipers, "Crowdsourcing for usability testing," *Proceedings of the American Society for Information Science and Technology*, vol. 49, no. 1, pp. 1–10, 2012.

[16] S. Zogaj, U. Bretschneider, and J. M. Leimeister, "Managing crowdsourced software testing: a case study based insight on the challenges of a crowdsourcing intermediary," *Journal of Business Economics*, vol. 84, no. 3, pp. 375–405, 2014.

[17] L. M. Riungu, O. Taipale, and K. Smolander, "Research issues for software testing in the cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 557–564.

[18] (2017) Bug bounty program. [Online]. Available: http://bit.ly/2jTEaw8

[19] (2017) The bug bounty list. [Online]. Available: http://bit.ly/1orp03T

[20] M. Sullivan and R. Chillarege, "Software defects and their impact on system availability: A study of field failures in operating systems," in *FTCS*, vol. 21, 1991, pp. 2–9.

[21] B. Boehm and V. R. Basili, "Software defect reduction top 10 list," *Foundations of empirical software engineering: the legacy of Victor R. Basili*, vol. 426, p. 37, 2005.

[22] 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.

[23] G.-A. Klutke, P. C. Kiessler, and M. A. Wortman, "A critical look at the bathtub curve," *IEEE Transactions on Reliability*, vol. 52, no. 1, pp. 125–129, 2003.

[24] (2016) Bathtub curve. [Online]. Available: http://bit.ly/2kGzD50

[25] J. Dunne, D. Malone, and J. Flood, "Social testing: A framework to support adoption of continuous delivery by small medium enterprises," in *2015 Second International Conference on Computer Science, Computer Engineering, and Social Media (CSCESM)*, Sept 2015, pp. 49–54.