Improving Software Quality and Reliability Through Analysing Sets of System Test Defects

Vincent Sinclair
Bell Labs Software and Systems Reliability
Dublin, Ireland
e-mail: vincent.sinclair@nokia-bell-labs.com

***Abstract* — Telecommunications networks support many critical services, leading users to demand very high levels of quality and reliability from these networks. The quality and reliability of these services is mainly dependent on the network's software. At the same time, competition is driving the demand for new software features in short delivery cycles. There are many challenges to delivering high quality, highly reliable software in these short cycles. Overcoming these challenges requires fast feedback to the development processes to minimize the number of escaped defects. This fast feedback can be achieved through the systematic analysis of system test defects. This method contrasts with the typical practice of analysing customer found defects. This improved method analyses each system test defect as it is fixed and stores this data. A set of defect data is then analysed to identify the most common defect type and where they are injected. This enables teams to focus improvement efforts on their largest source of defects. By automating this method, teams can continually fine tune their development processes to minimise the number of escaped defects. This results in a steady improvement over time in the quality and reliability of the software.**

Keywords - software reliability; software quality; availability; defect analysis; continuous improvement.

I. INTRODUCTION

Today's communications networks enable critical services, such as e-health, video doctors, mobile banking and remote security. Given the importance of such services, customers are demanding high reliability from their network providers to ensure these services are available anywhere and at any time. Future networks will support driverless cars and robotic surgery, requiring even higher levels of reliability. The quality and reliability of these services is highly dependent on the quality and reliability of the underlying communications software. This software is very complex and hence intrinsically prone to failure [1]. The challenge for communications network suppliers is to deliver these complex software systems with high quality and high reliability, while at the same time delivering new functionality in short delivery cycles.

This paper describes a new method for the analysis of software defects to enable teams to quickly learn from escaped defects. Section II describes the challenges. Sections III and IV describe the solution and its automation. Section V outlines the proposed future evolution of the system.

II. KEY CHALLENGES

Network software suppliers face several challenges to delivering high quality, high reliability software. Large development organizations typically work in complex, multi-site, multi-time zone and multi-language teams. This environment raises many challenges to close communication and collaboration, a key enabler of high quality and high reliability software. Large teams usually have a very wide range of knowledge and skill levels, from highly experienced engineers to junior engineers. This results in teams with dissimilar defect patterns and hence different improvement priorities. Each team needs to drive its own improvement priorities. A common top down approach across different teams is not as effective. The development processes, tools, organisational structures, as well as roles and responsibilities regularly change, disrupting development activities.

The software solution is typically a combination of application, platform, third party and open source software, leading to very complex software interactions. This can lead to unforeseen quality and reliability challenges. Time pressures on an already stretched team leaves very limited time to implement improvements. Driven by end users, network operators are demanding faster deliveries of new features and functionality. This leaves less time for testing out defects at system or solution level testing.

The above challenges to large-scale development lead to defects escaping to system test and customers. Our challenge is how to quickly learn from these escaped defects. Review of current defect analysis methods shows that they tend to focus on technical aspects of the individual defects. The resulting actions focus on preventing the same defect from escaping in future through the addition of test cases. Previous studies on root cause analysis tend to focus on identifying the types of defect but not on where they should have been detected [4]. By collecting and analysing characteristic data on system test defects, we can identify the most common defect type and where they could have been detected. From this, we can identify the optimum improvement action(s) to give the largest reduction in escaped defects.

III. SOLUTION

We will outline current analysis methods, compare these with our method and describe the key advantages of our improved method. Particular focus is put on the ability of the improved method to provide fast feedback to development.

Copyright (c) IARIA, 2019. ISBN: 978-1-61208-701-6

A. Current method

The typical approach to root cause analysis of software defects is to focus on customer found defects. This results in relatively slow feedback to development. The analysis tends to focus on individual defects, with the resulting improvement focusing on the technical cause of the defect. Where the analysis looks at processes, it tends to focus on the superficial cause of the defect rather than the fundamental cause(s) of the defect and how the defect escaped [2][5].

B. Learning cycle

Key to addressing the wide variety of challenges listed in section two is a learning cycle which provides fast feedback to the development teams through analysing system test defects. The approach must ensure that developers identify and record the fundamental cause of each defect at the organisational and process level. The system can then identify the most common defect type through analysing sets of defect data, highlighting which improvement will give the biggest reduction in escaped defects. The system must also measure the percentage of actual defect reduction to ensure the improvements implemented have been effective. Using this approach, teams can continually learn from their escaped defects. This learning cycle is outlined in Figure 1.

C. Improved method

The innovation is the real time classification and analysis of sets of system test defects. This method is built on the idea that sets of defects have small but definite patterns or signatures [3]. The steps in the method are:

- Classify each system test defect at the time the defect is fixed, when all of the information about the defect is fresh in the mind of the developer. The developer selects from drop down menus the type of defect and the development phase where it should have been detected. This data is recorded in the defect management tool and is mandatory to move the defect to the next phase of the defect life cycle.
- Select a specific set of defects for analysis. This could be at a team level, a component/sub-system level or at the level of a complete product.
- Extract the data from the defect management tool.
- Apply decision tree techniques to determine the most common defect type and in what phase of development they should have been detected.
- Based on the most common defect type and where they should be detected, quality experts perform a deep dive analysis to identify the fundamental changes that will reduce this specific defect type.
- Measure the impact of the improvement action(s) to quantify the reduction in the specific defect type.

D. Advantages of the method

Analysing system test defects provides much faster feedback to the development teams, compared to analysing customer found defects. Characterising each defect at the time it is fixed by a developer is easy and quick, as the developers have all the defect details fresh in their mind. Classifying the defect in the management system facilitates easy and accurate data labelling. Using drop down menus to classify each defect guides the developer towards the real root cause. Using drop down menus to classify the defects also provides data standardisation, facilitating easy and accurate clustering. By applying clustering techniques within a set of defects, teams can identify the most common type of defect within the set and the source of these defects. Empirical experience shows that a set of fifty defects provides sufficient data to identify the most common type of defect. A deep dive analysis on the most common defect type will identify fundamental changes to the organisation that will systematically prevent a whole class of defects escaping from development. These are typically fundamental improvements in communications and collaboration, changes in roles and responsibilities as well as improvements in processes, tools, templates and checklists. Each team can analyse their own defects to help them identify improvements relevant to their team. This devolves accountability for quality down to team level. Teams can also focus on improvements in specific areas of the software to strengthen weaker components. Automation enables the method to be applied regularly, with sustained defect reduction over time.

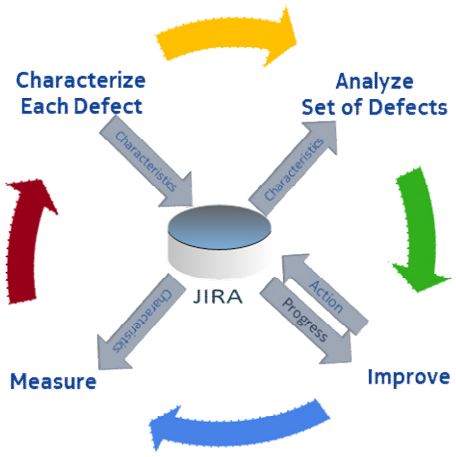

Figure 1. Learning Cycle.

E. Testing the method

Over a period of nine months, the improved method was tested with one product unit of seventy people, distributed across three continents and spanning systems engineering to system validation. The data processing and visualization were performed manually. The analysis identified the need for improvements in the areas of requirements gathering and communication, including detailed customer use cases, interface interoperability, corner cases, error cases and failure modes. For design, it identified improvements needed

Copyright (c) IARIA, 2019. ISBN: 978-1-61208-701-6

during design reviews, including communications tools and improved checklists to assure critical points were not missed. For the coding phase, it identified improvements in unit test, including additional rainy day/negative testing. At system validation, it identified the need for increased robustness testing as well as quality assurance of third-party software.

The parameter for evaluating the method is the number of escaped defects. Over the nine-month pilot period, these fundamental improvements resulted in a greater than 50% reduction in defects escaping to customers.

IV. AUTOMATING THE METHOD

A critical aspect related to the deployment of this method is the automation of the process, integrating the data collection, analysis and presentation of results into an organization's existing tool set.

A. *Advantages*

Automation of the method makes it fast and easy for teams to regularly analyse their own defects at any time. They can, for example, analyse the defects from the past four weeks to identify the most common defect type occurring today. This fast feedback enables teams to regularly fine tune their development and testing processes.

B. *Web application*

A cloud-based Web application was developed, which connects to the company's defect management systems. The application allows users to select a specific set of defects, extracts the defect classification data, performs data pre-processing and data analytics and finally visualises the results.

A typical defect pattern is shown in Figure 2. For this set of defects, the most common defect type is coding error. The next most common defect types are requirement gaps and high-level design gaps. From this graph, it is evident where the team should focus their improvement efforts.

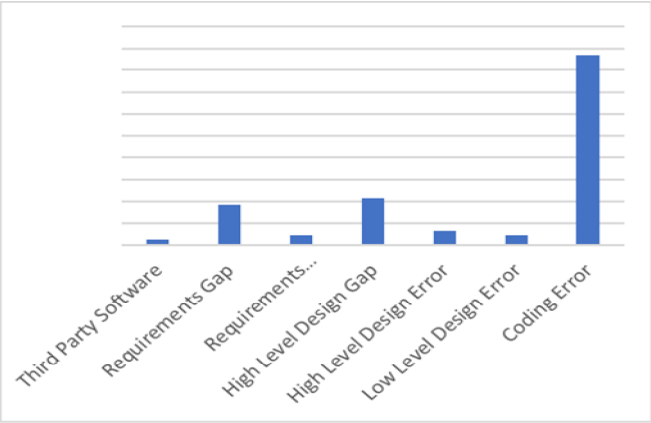

Figure 2. Defect Types – Set of 200 defects.

C. *Scaling the adoption of the method*

The application has been made available to all teams through the Nokia corporate cloud, enabling a wide variety of teams to test the usability of the application. This will also encourage an even stronger culture of learning from sets of defects to identify the most common defect type and trigger actions to prevent similar defects in future. Automation will also encourage a mindset of regular improvements to continually fine tune the development activity.

V. EXTENDING THE APPLICATION

A number of additional capabilities are planned for future releases.

Based on multiple factors, a weighting engine will quantify the impact of each defect. For example, a defect which causes a network outage or affects live traffic has more impact than a defect that does not affect live traffic. These weighted values will be used to rank the improvement priorities. A recommendations engine will use data analytics and machine learning techniques to automatically select the optimum improvement actions from a knowledge base of known effective solutions for specific defect type/phase combinations. This knowledge base will evolve over time based on the evaluation of the effectiveness of specific improvement actions. The system will include automated tracking of defect trends over time to measure the impact of specific improvement actions.

VI. CONCLUSION

This method has been shown to be effective in reducing escaped defects, resulting in improved software quality and reliability. A key enabler to widespread deployment of the improved method is the Web application, which enables teams to regularly analyse their own escaped software defects. Future releases will build on the current system to add further machine learning and artificial intelligence techniques to automatically recommend the most effective improvement actions. Machine learning techniques will also be used to optimise over time the knowledge base of known solutions for specific defect types. Results will be presented in a future paper.

REFERENCES

[1] R. I. Cook, "'How complex systems fail," Cognitive Technologies Laboratory, University of Chicago IL [Online]. Available:https://www.researchgate.net/publication/228797158_How_complex_systems_fail. [Accessed: Jan. 30, 2019].

[2] A. C. Edmondson, "'Strategies for learning from failure," Harv.Bus.Rev., vol. 89, no. 4, pp. 48-55, April 2011.

[3] M. Syed, "'Black Box Thinking: Why Most People Never Learn from Their Mistakes--But Some Do", November 2015.

[4] Timo O.A. Lehtinen, "What Are Problem Causes of Software Projects?, International Symposium on Empirical Software Engineering and Measurement, September 2011.

[5] Harsh Lal, "Root cause analysis of software bugs using machine learning techniques", International Conference on Cloud Computing, Data Science and Engineering, January 2017.

Copyright (c) IARIA, 2019. ISBN: 978-1-61208-701-6