Testing Practices of Software in Safety Critical Systems: Industrial Survey

Mohamad Kassab
Engineering Division, Pennsylvania State University, Malvern, PA, U.S.A.

Keywords: Safety-critical Systems, Software Testing, Software Professionals, Industrial Practices.

Abstract: The software becomes increasingly a core integrated part of the safety-critical systems. Unfortunately, little contemporary data exists to document the actual practices used by software professionals for software testing and quality assurance activities for software in safety-critical systems. To remedy the deficiency of lack of data, we conducted a comprehensive survey of software professionals to attempt to discover these practices. In this paper we report on our findings from this survey on the state of practice of testing software for safety-critical systems in respect to three areas: 1) The integration of the testing activities within the software development life cycle; 2) Testing methods and techniques; 3) Testing metrics and defects management. We also provide some comparison with testing software for non-safety-critical systems.

1 INTRODUCTION

A safety-critical system is a system whose malfunction may result in death or serious injury to people, loss or damage to property or environmental harm. Engineers have developed safety-critical systems by relying on conservative best practices, standards (e.g. MIL-STD-882E: System Safety (DoD, 2012), ISO 26262, Road vehicles-Functional safety (ISO, 2011), NASA-STD-8719.13C: Software Safety Standard (NASA, 2013)) and a culture where safety considerations are integral to all aspects of an organization (Feiler et al., 2013).

The software becomes increasingly a core integrated part of the safety-critical systems. The industry cost for the software of current-generation aircraft has reached an $8 billion (Redman et al., 2010). The avionics system in the F-22 Raptor consists of about 1.7 million lines of software code (Charette, 2009) as 80% of its functionality is achieved by software which compromised 30% of engineering and manufacturing development costs. Software in cars is only going to grow in both amount and complexity. It is estimated that cars will require 200 million to 300 million lines of software code in the near future (Charette, 2009).

While a software in isolation cannot do physical harm, a software in the context of a system and an embedding environment could be vulnerable (Naik and Tripathy, 2011). For example, a software module in a database is not hazardous by itself, but if a radiation therapy machine delivers fatal doses to patients because of a software error then it is not a safe software (Leveson and Turner, 1993). Software is considered safety-critical if it controls or monitors hazardous or safety-critical hardware or software. Such software usually resides on remote, embedded, and/or real-time systems (NASA, 2013).

The organizations developing safety-critical software systems should have a clear testing strategy that defines the methods for identifying, tracking, evaluating and eliminating hazards associated with a system. Despite best build-then-test practices, system-level faults due to software have increasingly dominated the rework effort for faults discovered during system integration and acceptance testing. Several studies of safety-critical systems show that 80% of all errors are not discovered until system integration or later. The rework effort to correct a problem in later phases can be as high as 300-1000 times the cost of in-phase correction (Feiler et al., 2013).

In order to trigger any favorable change in this state of practice, a serious effort is required in predicting the trends, learning the stakeholder mindsets, and pinpointing the problem areas in software testing. Unfortunately, little contemporary data exists to document the actual practices used by software professionals for software testing and quality assurance (QA) activities for safety-critical systems. This is partly because the data are commercially sensitive, and partly because the data is not always collected systemati-

Kassab, M.
Testing Practices of Software in Safety Critical Systems: Industrial Survey.
DOI: 10.5220/0006797003590367
In *Proceedings of the 20th International Conference on Enterprise Information Systems (ICEIS 2018)*, pages 359-367
ISBN: 978-989-758-298-1

cally (McDermid and Kelly, 2006).

We conducted a comprehensive survey of software professionals to attempt to discover these practices. Surveys of software industry professionals are an effective way to determine the current trends in software engineering processes. Survey responses can also help others to understand the relationship between area such as software quality and testing (Ng et al., 2004). A carefully constructed survey has the potential to: 1) remedy the deficiency of lack of data and 2) to identify the software testing best practices, which can then be disseminated. Based on these two objectives, We designed a survey study on the current software testing state of practice. While an initial view from the survey results was presented in (Kassab et al., 2017), in this paper we provide a different pragmatic view to report on the state of practice for testing software in safety-critical systems.

The rest of the paper is organized as follows: Section 2 discusses related work while Section 3 describes the survey's design and conduct. In Section 4 we provide general statistics regarding the participants, their organizations and the reported projects. Section 5 provides our findings on the state of practice of testing software for safety-critical systems and provides some comparison with testing software for non-safety-critical systems. Section 6 discusses the limitation to validity. Finally, the conclusions are presented in Section 7.

2 RELATED WORK

There are few works available involving surveys of software professionals with respect to testing (e.g. (Haberl et al., 2011), (Ng et al., 2004), (Causevic et al., 2010), (Turkish-Testing-Board, 2014), (ISTQB, 2014), (Kasurinen et al., 2010), (Kanij et al., 2011), (Knauss et al., 2016)). For example, in (Knauss et al., 2016), the authors presented an investigation the state-of-the-art and future trends of testing critical cyber-physical systems on the example of active safety systems for vehicles. The results from conducting four focus groups with Swedish industrial partners show that while the main testing processes and scenarios are supported, there is a clear need to enable testing of more complex scenarios in realistic settings, as well as increasing the degree of automating therefor to achieve better repeatability and a more effective test resource usage. Indeed, many other survey results indicate that the percent of automated testing is low in industry (Causevic et al., 2010), (Rafi et al., 2012), (Lee et al., 2012). Supporting this fact are other studies that indicate tool adoption is also low (Ng et al., 2004), (Garousi and Varma, 2010). While current survey studies show that organizations do not make effective use of testing tools (Grindal et al., 2006), Garousi et al. (Garousi and Varma, 2010) found that automated testing has increased since 2004.

Causevics survey results (Causevic et al., 2010) indicated that the use of open source vs. proprietary testing tools depended on whether or not they were unit testing or performing higher level system testing. Although Causevic (Causevic et al., 2010) found that writing test cases before writing code is mostly not considered a current practice, our survey results showed that these approaches are becoming established in practice.

Other surveys with focus on the cost and productivity aspects of testing also exist. For example, Kasurinen et al. (Kasurinen et al., 2010) examined the cost factor for testing, finding that testing is often a much underestimated part of the project. These researchers found, however, that more effective testing process may reduce testing time which is often underestimated.

Another area of testing research is how to build an effective testing team. Kanij, Merkel and Grundy (Kanij et al., 2011) conducted a survey of software practitioners to determine the importance of factors in building testing teams. The results suggest that experience in software testing is more important than a team members interpersonal skills. The results also suggest the desire for the testing team to be built with members having diverse work experience (Kanij et al., 2011).

On the other hands, The U.S. Army has recognized that qualifying the airworthiness of rotorcraft has increasingly become infeasible with current software test practices trying to achieve full code coverage due to increased software size and interaction complexity (Boydston and Lewis, 2009). Compliance with standards and practices specific to the certification of safety-critical software systems such as DO-178B and C, SAE ARP 4754, and SAE ARP 4761 (Johnson et al., 1998), (International, 1996) becomes essential instrument besides testing to improve the quality and ensure the safety of the software. The Software Engineering Institute (SEI) published in 2013 a white paper presenting an improvement strategy comprising four pillars of an integrate-then-build practice that lead to improved quality through early defect discovery and incremental end-to-end validation and verification (Feiler et al., 2013). The four pillars are: 1. Capture of mission and safety-criticality requirements in analyzable form; 2. Virtual integration of the physical system, hardware platform, and software architectures through consistent analyzable

architecture models; 3. Static analysis techniques applied to the models and actual system implementation to complement testing; and 4. Incremental assurance of justified confidence through consistent end-to-end evidence throughout the development life cycle.

There are also studies to investigate the sources of errors in safety-critical systems. For example, studies of safety-critical software-reliant systems developed show that 70% of the errors are introduced in the requirements (35%) and architecture design phases (35%) (Planning, 2002), (Galin, 2004), (Dabney, 2003). A NASA study traced down requirements errors and found that missing requirements accounts for 33% of the all requirements errors comparing to 24% for incorrect requirements, 21% for incomplete requirements, 6% for and ambiguous, 5% to inconsistency, and 6% to over-specified requirements (Hayes, 2003).

3 SURVEY DESIGN AND CONDUCT

We designed a web-based survey using QuestionPro tool (www.QuestionPro.com). The survey questions were designed after a careful review to similar other conducted survey studies (Haberl et al., 2011), (Ng et al., 2004), (Causevic et al., 2010), (Turkish-Testing-Board, 2014), (ISTQB, 2014). To allow a valid comparison with the other conducted surveys, we also included selected questions from these surveys into ours. In total, the survey consisted of 40 questions arranged into six sections related to: project information, integrating software testing activities within the software development life cycle; software testing methodologies and techniques; testing metrics and defects management, organizations information and participants professional information. A summary of our survey questions is available in https://goo.gl/kGBLhq. In this summary we also highlight which questions are shared with the other surveys mentioned above.

We drew our survey participants from multiple sources but primarily from a database of past graduate students in Software Engineering of the Penn State School of Graduate Professional Studies. The school caters primarily to working professionals. An email invitation (and subsequent reminder) was sent to these individuals. We also posted an invitation at related Linked-In professional testing and quality groups, to which the author belonged. Respondents were asked to base their responses on only one software project that they were either currently involved with or had taken part in during the past five years.

We collected survey data through two phases. The first phase was from January and June 2015. At the end of this phase, we conducted sessions to analyze the captured responses. We presented an initial view of the overall collected results from all projects by the end of this phase (Kassab et al., 2017). We also made a decision to go into a second phase to collect more responses. The second phase lasted from June to December 2016. Overall, of the 293 who viewed the survey; there were 195 who started taking the survey. Of these survey takers; there were 72 who completed the survey all the way to the end. The completion rate was 37% and the average time taken to complete the survey was 17 minutes. We also included the results of the partially completed responses, which have been analyzed following the standard statistical practices (Phillips and Sweeting, 1996), particularly revived in medical research (Rezvan et al., 2015), taking also into account the specificity of online questionnaires (including the lack of responses or the partial responses), as discussed in details in (Tsikriktsis, 2005). All responses were treated anonymously and only aggregate data were used - not the individual responses.

4 GENERAL STATISTICS REGARDING THE PARTICIPANTS AND PROJECTS CHARACTERISTICS

In order to make well-informed statements about the practice of software testing, it was essential to attract as many experienced participants as possible. Both objectives were achieved for this study. Eighteen different industries were represented. The reported professional experience represented in the survey was impressive with an average of 7.8 years of related IT/Software experience. The reported academic qualification indicated that 100% of survey participants have successfully completed a bachelors or equivalent, and 32% even hold a masters degree or doctorate. The survey responses captured a diverse mix of positions within the chosen projects. To view the complete survey results on participant and project characteristics in a graphical format, we suggest to the reader to view these charts at: https://goo.gl/xWHEhO

Since we aimed at classifying the projects based on their safety-criticality, the respondents were asked to specify the maximum loss or damage if the software being developed for the project failed (that is,

the delivered service no longer complies with the specifications). There were 62 responses reported a highly critical system where serious failure could involve loss of one or more lives. The majority of respondents (133 responses) reported the loss would be limited to essential funds, discretionary funds or comfort. In this paper, we analyze in depth those 62 responses for the safety-critical software and compare their reported practices with the non-safety-critical software projects. In general, the projects were distributed across different categories with bias towards database projects (22% of the projects). In case of the safety-critical software sample, there was an obvious bias towards embedded systems (42%). The overall projects sample showed also a distribution across a broad range of application domains with a mild bias towards applications in the Information Technology sector (15% of the projects) while in case of the of the safety-critical software sample, both the Aerospace and Defense domains dominated the distribution at 53% and 37% correspondingly. The majority of the projects were classified as new development (at 64% for the overall sample and 47% for the safety-critical software sample), while 14% of the overall projects were legacy system evolution (32% of the safety-critical software) and 14% of the overall sample were classified as enhancement projects (21% of the safety-critical software). The overall sample population presents companies located in different geographic regions (9 countries were represented in this survey). As far as the size of the participating companies is concerned, a representative sample can be determined. Almost 44% of the participants work in small companies (with 1- 100 full time employees). But also very large companies (with more than 1000 full time employees) are well represented at 28%. Regarding those software projects for safety-critical systems, 40% of the corresponding respondents worked in very large companies. It was surprising to find that the question enquiring about an independent QA department in the organization was affirmed by 56% from the safety-critical projects comparing to the higher figure of 67% from the non-safety-critical projects.

5 STATE OF PRACTICE OF TESTING SOFTWARE IN SAFETY CRITICAL SYSTEMS

In this section we report on our findings from this survey in respect to three areas related to software testing for safety-critical software: 1) The integration of the testing activities within the software development life cycle; 2) Testing methods and techniques; 3) Testing metrics and defects management.

5.1 Integrating Testing Activities Within Software Development Life Cycle

Since software development has changed increasingly into an engineering discipline, often involving widely distributed teams, the methods and frameworks used have evolved accordingly. The need for efficient and cost-effective software production has also reached software testing. Hence, we were motivated to investigate how is the software testing methodically implemented in organizations. Several studies of safety-critical systems show that 80% of all errors are not discovered until system integration or later (Feiler et al., 2013). The results from our survey provided an evidence that performing QA measures is concentrated on the late phases of software development. This was actually the pattern in both samples of safety-critical and non-safety-critical software. Only 29% of the participants from the safety-critical software agreed that they use QA measures in the Study & Concept phase. While the shares of those that use quality assurance in the Requirements Specifications and System Design phases are 59% for each. From the Implementation phase onwards, quality assurance practices increase significantly (Figure 1). We observed that quality activities for the safety-critical software were performed at a higher rate than for non-safety-critical software in every phase of development except for Implementation.

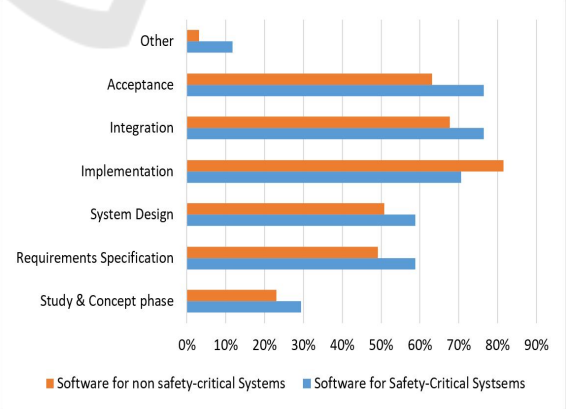

Figure 1: In which phases are Quality Assurance measures are / were applicable within this project?

When asked on the ”Testing as defined phase in the project development”; 85% of participants from the safety-critical software sample reported a level of

agreement (strongly agree or agree) that this was the actual practice in the project, and 92% reported an agreement that they personally prefer to have testing as a defined phase on its own. The corresponding numbers for the non-safety-critical software were 70% for the actual practices and 82% for the preference. If we considered a dissatisfaction in a question as the difference between the current and the preferred practice, then these numbers indicate a higher level of satisfaction in the safety-critical software on how the software testing activities are distinct from the rest of activities within a project. Planning the duration and the budget of the quality assurance activities together with other software development activities in a total package was the most common approach in both safety-critical software (44%) and the non-safety-critical software (37%) whenever the duration and budget were planned (See Figure 2). The safety-critical software sample showed a more maturity level in budget and duration planning as only 5% of the participants do not plan any explicit estimation for QA activities (comparing to 18% from the non-safety-critical sample) (see Figure 2).

About 33% from safety-critical software sample reported that the team is not doing a good job for estimating the size / effort of software testing activates (comparing to 30% from the non-safety-critical software); nevertheless, it was surprising to see a significant difference between the two samples when reporting on that they didn't have enough time to test the software before its deployment (66% for safety-critical and only 39% for non-safety-critical).

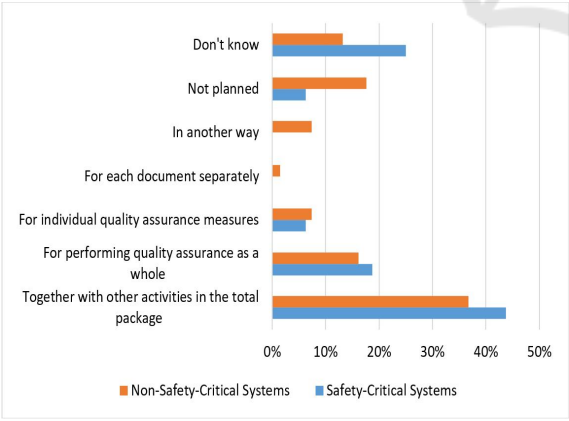

Figure 2: The quality assurance effort (budget and time) in your projects is planned ...

5.2 Testing Methods and Techniques

Respondents, who indicated that they perform testing activities were additionally asked questions regarding tools and techniques in use within their projects.

It is very common for organizations to have defined levels of testing. Those levels include unit, integration, system, acceptance and regression testing.

While the results in Figure 3 clearly show that system-level testing is the most common level of testing for safety-critical software (93% of participants reported applying system test for their projects), around 90% of participants reported that system testing was applied in order to test more than just one characteristic of the system with a clear focus on testing the functionality (Figure 4). Performance was the most tested quality attribute in the surveyed projects. This was the case for both samples: safety and non-safety-critical software. As one may expect, both regulatory and reliability testing were executed at a higher rate for safety-critical software in comparison to non-safety-critical (Figure 4).

Regression Testing is a level of software testing that seeks to uncover new software bugs, or regressions, in existing functional and non-functional areas of a system after changes such as enhancements, patches or configuration changes, have been made to them. We observed that Regression testing is executed more frequently for safety-critical software (75% of respondents) in comparison to non-safety-critical (49%) (Figure 4). Only 14% from the safety-critical software sample reported that regression testing was outsourced for their projects; and this number is close to the only 20% of participants from safety-critical software who personally preferred to outsource regression testing. This indicates a level of satisfaction on the current outsourcing practices of the regression testing activities.

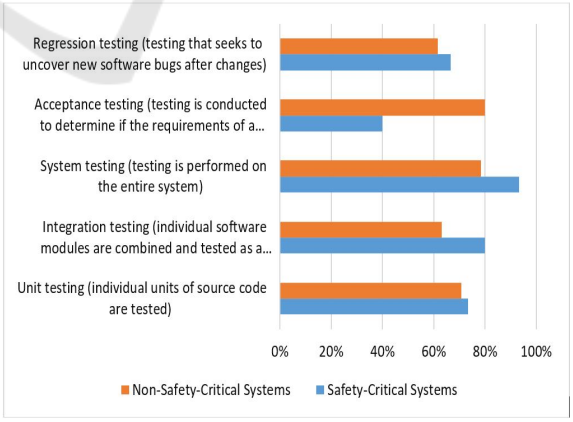

Figure 3: Which of the following levels of testing are/ were applied for this project? (Select all that apply).

The survey showed that in all of their dynamic testing activities, participants from both samples use principally Black-Box testing techniques (80% used this technique in safety-critical sample comparing to

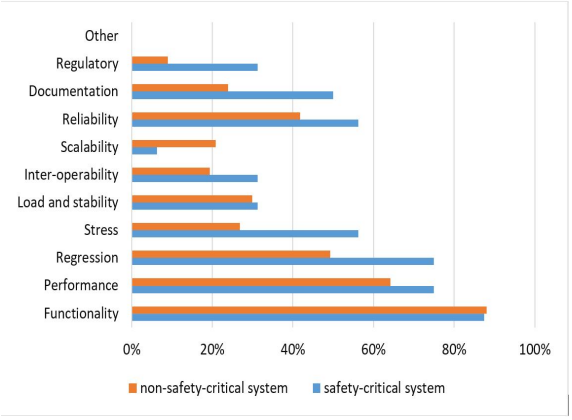

Figure 4: Which of the following types of system tests are / were executed for your project? (Select all that apply).

79% in the non-safety-critical systems sample). It is noticeable the difference between the two samples on using the structured-based techniques (white-box). Sixty percent of the safety-critical software reported using white box in comparison to 39% of the non-safety-critical software.

The results show that systematic test case design and test data definition for the non-safety-critical software are widespread. Even though 60% of the participants reported that they prefer to have the test cases written before writing the code; only 28% reported that this was the actual practice in their projects.

It was surprising to see that for the non-safety-critical software category of respondents, the current practice of writing test cases before writing code was more popular than for the safety-critical software category (reported only at 14%). However, while the non-safety-critical respondents seem quite willing to even improve the situation more, the safety-critical respondents show no interest as a group in changing towards a more test-driven development (preference was reported only at 20%). This is noteworthy considering the fact that empirical studies seem to ascribe test-driven developed code a high external code quality. For fairness sake, it is not trivial to see how such a practice would affect, and be affected by, other specific aspects of safety-critical system development, e.g., fulfillment of safety certification standards.

On the positive side, formal languages are used more often to describe test cases for the safety-critical software (57%) than in non-safety-critical software (43%). Most of the test cases for the non-safety-critical software were described freely in verbal or text-based forms (56%), while only 29% of the test cases from the safety-critical sample were described this way. The overall effectiveness of test cases for the safety-critical software is rated high since almost 60% reported that most or all of defects are found during test cases execution. This number was higher than the one from non-safety-critical software sample (43%). About 40% rate test case effectiveness in safety-critical software as medium (some defects are found) (Figure 5).

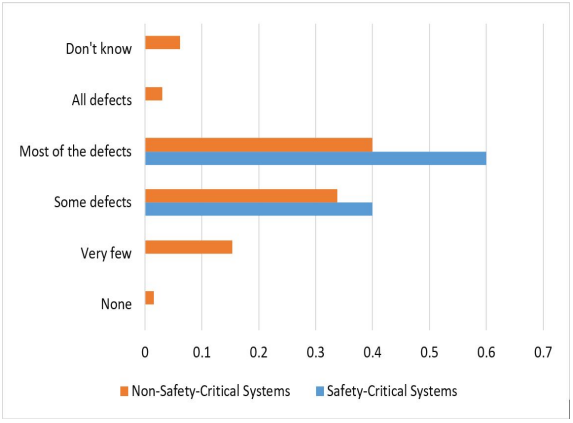

Figure 5: How many of the defects in this project are / were discovered by executing the test cases?

The expected test results from safety-critical software sample were available prior to test execution in 53% of all cases (in comparison to 70% for non-safety-critical sample). This survey result is a surprise. On the other hand, both samples were almost similar in comparing the expected results with the actual results manually (80% for safety sample and 82% for non-safety sample).

Data protection is a matter that organizations must take very seriously these days. It is therefore quite surprising that 33% of safety-critical sample use the original the production data (in comparison to 20% from non-safety-critical sample), and only 33% of the respondents comprehensively document the test data (Figure 6). In addition, majority (60%) of the survey participants stated that they do not explicitly distinguish between test case generation and the generation of associated test data. Only 40% of safety-critical reported on using a separate test system. Respondents from this sample utilized more often using the integration system for testing purposes (73%) while respondents from the non-safety-critical systems utilized more often using the development system (82%) (Figure 7).

5.3 Testing Metrics and Defects Management

The results on exit criteria to conclude testing activities indicated that the majority from both samples were in favor of concluding the test when ”all planned

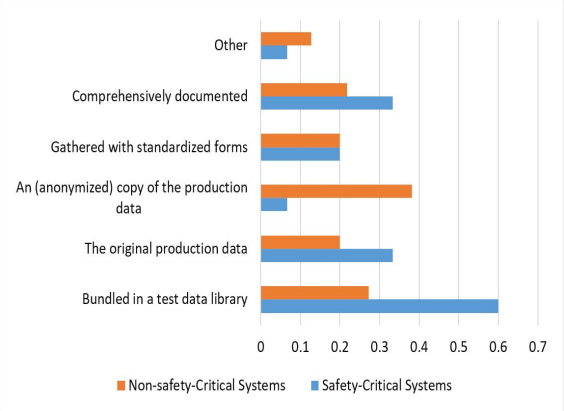

Figure 6: In this project, test data are / were ...

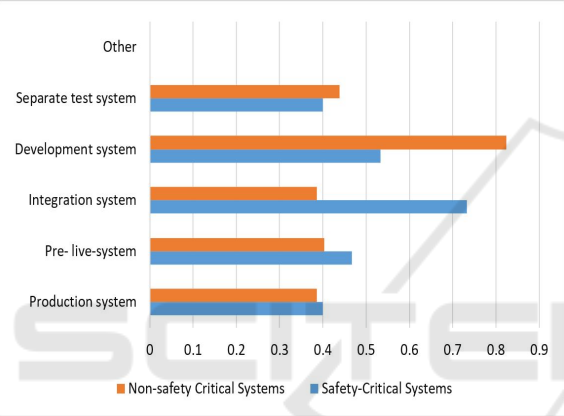

Figure 7: Which system environments are used for testing in your projects?

test activities were performed" and when "each requirement has been tested at least once" (Figure 8). Although a good test process maturity can be concluded from this, it is nonetheless 29% of the safety-critical participants close their test activities when the delivery time has been reached (42% for the non-safety-critical). This finding is an indication that even though the safety-critical sample showed a more mature practice; testing continues to be planned as a "buffer" in the project that will be sacrificed if there are delays from preceding project phases.

The most common cause in both samples for the discovered defects was related to requirements problems (omissions, changes, and errors) - 72% reported this cause. This is consistent with the findings from other studies we referred to in Section 2 on the sources of defects in the Software (Planning, 2002), (Galin, 2004), (Dabney, 2003), (Kassab, 2014), (Kassab, 2015). Design problems was the second most reported cause at 66%. JIRA was the most used tool to report the defects.

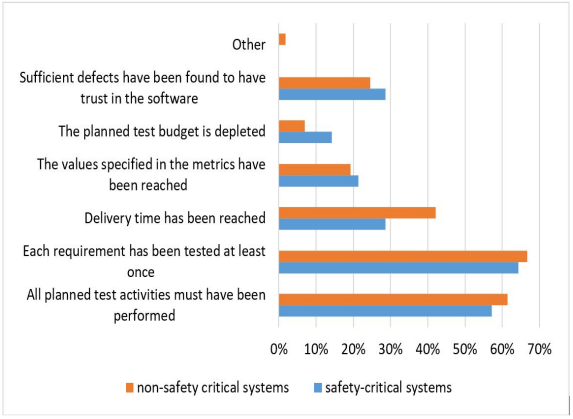

Figure 8: For test completion, the following exit criteria are used ...

6 LIMITATION TO VALIDITY

We carefully examined our study for the possible types threats to validity described in (Campbell and Stanley, 2015), (Hyman, 1982) and (Wohlin et al., 2012). One possible internal threat to validity that we identified is related to the instrumentation. This is the effect caused by the artifacts (e.g. survey questions) if these are badly designed. Our survey questions were designed after a careful review to similar other conducted survey studies (Haberl et al., 2011), (Ng et al., 2004), (Causevic et al., 2010), (Turkish-Testing-Board, 2014), (ISTQB, 2014). In addition, we sent out the link to the survey to a number of researchers to collect their feedback before the data collection phase started. We addressed the feedback towards improving the quality of the questions. One received comment during the survey design assessment phase suggested providing an explanation next the possible answers for particular questions to reduce possible ambiguity. We implemented this suggestion when applicable. For example, in the question on what levels of testing were executed for the project, a clear definition was provided next to each of the possible answers a participant may select from (e.g. unit testing, integration testing, acceptance testing, etc.).

Another possible internal threat is related to the morality. This is the effect due to the different kinds of persons who drop out from the survey. We carefully examined the sample whom dropped out in regards of three participants characteristics: job role, education level and years of experiences. We observed that the drop out sample was representative to the total sample. We couldn't relate the dropout to a particular participants characteristic. In addition, we examined the results from the perceptions of managerial job roles and non-managerial roles. No significant

differences on the results came from these two samples.

A third aspect of internal validity we examined was related to history. This is related to different results may be obtained by the same sample at different times. To address this, we run our data collection phase into two rounds as explained in Section 3. We followed the same recruiting strategy in each round. If we break the results into two samples to correspond to the two rounds, we also observe no significant differences on the findings between the two samples.

On the external validity threats, we examined the possible threat related to interaction of selection and treatment. This is an effect of having a subject population, not representative of the population we want to generalize to. Judging from the participants characteristics we presented in Section 3, this was not the case. A second possible external validity threat may be related to the interaction of history and treatment. For example, if the survey is taken a few days after a big software-related crash, people tend to answer differently than few days before. To address this threat, the two data collection phases were spanned over relatively a long period (12 months).

Lastly, it is worth saying that all the studies like this require replications and confirmatory studies, especially in software engineering. To facilitate the replication of our study, we posted the survey questions from this survey through the link: https://goo.gl/kGBLhq. Researchers and practitioners are welcome to execute further analysis on the data. The author is available to offer the original text of the questionnaire to any scientist interested in replication.

7 CONCLUSION

In this study, we collected 195 partial or complete responses from software professionals from wide range of industries and backgrounds to analyze the software testing state of practice. In this paper, we provided a pragmatic view from the results to focus on the software in safety-critical systems. The survey results included a wide variety of raw information, but interpreting some of this information, we offer the following key findings:

- Performing quality assurance measures for software in safety-critical systems is concentrated on the late phases of software development.
- The safety-critical software sample showed a higher level of maturity (comparing to non-safety-critical sample) in maintaining ”testing” as a distinct phase.
- The overall effectiveness of test cases is perceived to be high for software in safety-critical software (60%) comparing to 43% for the non-safety-critical.
- Safety-critical software utilizes mostly formal languages to describe test cases.
- Regulatory and reliability testing are executed at a higher rate for safety-critical software (comparing to non-safety-critical sample). White-box testing techniques are also executed at a higher rate.
- Particular areas that require an attention for potential improvement include:
  - Systematic test case design and test data definition are not uniformly practiced.
  - Expected results need to be available sooner and prior to test execution.
  - The usage of production data need to be avoided.
  - There is a need to improve the size and effort estimation practice for testing activities.
  - Requirements problems continue to be the most common cause for the discovered defects.

We hope the survey and corresponding results stimulate research into prevailing software practices, but moreover, we intend these results to highlight the areas of software testing that need the attention of both the research community and the industry professional. Our own subsequent work will offer more detailed analysis of some of the survey results. We also plan to replicate the study in the near future to observe any potential changes in the landscape of testing practices for software in safety critical systems.

REFERENCES

Boydston, A. and Lewis, W. (2009). Qualification and reliability of complex electronic rotorcraft systems. In *Army Helicopter Society System Engineering Meeting*.

Campbell, D. T. and Stanley, J. C. (2015). *Experimental and quasi-experimental designs for research*. Ravenio Books.

Causevic, A., Sundmark, D., and Punnekkat, S. (2010). An industrial survey on contemporary aspects of software testing. In *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, pages 393–401. IEEE.

Charette, R. N. (2009). This car runs on code. *IEEE spectrum*, 46(3):3.

Dabney, J. (2003). Return on investment of independent verification and validation study preliminary phase 2b report. *Fairmont, WV: NASA IV&V Facility*.

DoD, U. (2012). Mil-std-882e, department of defense standard practice system safety. *US Department of Defense*.

Feiler, P., Goodenough, J., Gurfinkel, A., Weinstock, C., and Wrage, L. (2013). Four pillars for improving the quality of safety-critical software-reliant systems. Technical report, SOFTWARE ENGINEERING INSTITUTE.

Galin, D. (2004). *Software quality assurance: from theory to implementation*. Pearson Education India.

Garousi, V. and Varma, T. (2010). A replicated survey of software testing practices in the Canadian province of Alberta: What has changed from 2004 to 2009? *Journal of Systems and Software*, 83(11):2251–2262.

Grindal, M., Offutt, J., and Mellin, J. (2006). On the testing maturity of software producing organizations. In *Testing: Academic and Industrial Conference-Practice And Research Techniques*, pages 171–180. IEEE.

Haberl, P., Spillner, A., Vosseberg, K., and Winter, M. (2011). Survey 2011: Software test in practice. *Translation of Umfrage*.

Hayes, J. H. (2003). Building a requirement fault taxonomy: Experiences from a NASA verification and validation research project. In *14th International Symposium on Software Reliability Engineering.*, pages 49–59. IEEE.

Hyman, R. (1982). Quasi-experimentation: Design and analysis issues for field settings (book). *Journal of Personality Assessment*, 46(1):96–97.

International, S. (1996). *Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment*. SAE International.

ISO, I. (2011). 26262: Road vehicles-functional safety. *International Standard ISO/FDIS*, 26262.

ISTQB (2014). Istqb effectiveness survey 2013-14. http://www.istqb.org/documents/ISTQB_Effectiveness_Survey_2013_14.pdf.

Johnson, L. A. et al. (1998). Do-178b, software considerations in airborne systems and equipment certification. *Crosstalk, October*, 199.

Kanij, T., Merkel, R., and Grundy, J. (2011). A preliminary study on factors affecting software testing team performance. In *2011 International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 359–362. IEEE.

Kassab, M. (2014). An empirical study on the requirements engineering practices for agile software development. In *40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pages 254–261.

Kassab, M. (2015). The changing landscape of requirements engineering practices over the past decade. In *IEEE 5th International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 1–8.

Kassab, M., DeFranco, J. F., and Laplante, P. A. (2017). Software testing: The state of the practice. *IEEE Software*, 34(5):46–52.

Kasurinen, J., Taipale, O., and Smolander, K. (2010). Software test automation in practice: empirical observations. *Advances in Software Engineering*, 2010.

Knauss, A., Berger, C., and Eriksson, H. (2016). Towards state-of-the-art and future trends in testing of active safety systems. In *Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems*, pages 36–42. ACM.

Lee, J., Kang, S., and Lee, D. (2012). Survey on software testing practices. *IET software*, 6(3):275–282.

Leveson, N. G. and Turner, C. S. (1993). An investigation of the Therac-25 accidents. *Computer*, 26(7):18–41.

McDermid, J. and Kelly, T. (2006). Software in safety critical systems-achievement & prediction. *Nuclear Future*, 2(3):140.

Naik, K. and Tripathy, P. (2011). *Software testing and quality assurance: theory and practice*. John Wiley & Sons.

NASA (2013). Nasa-std 8719.13 software safety standard. https://standards.nasa.gov/standard/nasa/nasa-gb-871913.

Ng, S., Murnane, T., Reed, K., Grant, D., and Chen, T. (2004). A preliminary survey on software testing practices in Australia. In *Software Engineering Conference, 2004. Proceedings. 2004 Australian*, pages 116–125. IEEE.

Phillips, M. and Sweeting, T. (1996). Estimation for censored exponential data when the censoring times are subject to error. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 775–783.

Planning, S. (2002). The economic impacts of inadequate infrastructure for software testing.

Rafi, D. M., Moses, K. R. K., Petersen, K., and Mäntylä, M. V. (2012). Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In *Proceedings of the 7th International Workshop on Automation of Software Test*, pages 36–42. IEEE Press.

Redman, D., Ward, D., Chilenski, J., and Pollari, G. (2010). Virtual integration for improved system design. In *Analytic Virtual Integration of Cyber-Physical Systems Workshop (AVICPS)*, volume 52498, pages 57–64.

Rezvan, P. H., Lee, K. J., and Simpson, J. A. (2015). The rise of multiple imputation: a review of the reporting and implementation of the method in medical research. *BMC medical research methodology*, 15(1):30.

Tsikriktsis, N. (2005). A review of techniques for treating missing data in om survey research. *Journal of Operations Management*, 24(1):53–62.

Turkish-Testing-Board (2014). Software quality report 2014-2015 released by Turkish testing board. http://www.istqb.org/documents/TurkeySoftwareQualityReport_2014_2015.pdf.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.