UNIVERSITY OF GOTHENBURG

VOLVO

VOLVO TECHNOLOGY

A case study on Software Testing Methods and Tools

A pre-study on software testing requirements of ISO/DIS 26262

Master of Science Thesis in Software Engineering and Management

Bharat Bhushan Konka

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, July 2011

A case study on Software Testing Methods and Tools
A pre-study on software testing requirements of ISO/DIS 26262

Bharat Bhushan Konka

Examiner: Miroslaw Staron
Supervisor: Gerardo Schneider

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover:
This research study was performed at VOLVO Technology

Department of Computer Science and Engineering
Göteborg, Sweden , July 2011

Acknowledgements:

This thesis would not have been possible without the support from various persons. First of all, I would like to thank Volvo Technology (VTEC) for providing opportunity to conduct this research study and also providing necessary resources for this research. I owe my deepest gratitude to supervisors at Volvo Technology, Marcy Kirk and Camilla Dahl. This thesis could not have been possible without their constant guidance and support.
I am grateful to my supervisor Dr. Gerardo Schneider from Department of Computer Science and Engineering, Chalmers University, for providing his crucial feedback and guidance during this research.
Moreover, I am very thankful to Dr. Dick Stenmark and lecturer Agneta Nilsson from IT University for providing their valuable guidance regarding research methods. I also thank Olof Bridal from Volvo Technology for providing guidance regarding ISO 26262.
Finally, it is a pleasure to thank all the interview respondents from Volvo Technology who participated in this research.

Bharat Bhushan Konka

Contents

Abstract

This paper presents the findings and analysis done on a case study conducted at Volvo Technology (VTEC, Göteborg, Sweden) for studying the software testing and verification methods, tools and practices used in automobile domain. This case study was conducted during 2011 (Feb - June). A total of 14 interviews were conducted with persons belonging to 2 different departments, 6 groups and 8 projects. This case study focuses on software testing methods and practices, activities performed with software testing tools and also software testing standards. Based on the outcomes of the case study the contemporary practices of software testing in automotive domain are presented and also some recommendations regarding best practices. This thesis also presents a pre-study on the forthcoming automobile standard ISO/DIS 26262.

1. Introduction

Software testing is defined as *a formal process in which a software unit, several integrated software units or an entire package are examined by running the programs on a computer. All the associated tests are performed according to approved test procedures on approved test cases* (Galin, 2004). Testing plays a central role in quality assurance activities of many organizations. Finding more efficient ways to perform more effective testing is a key challenge in testing. It is observed that an efficient testing practice is vital to the quality of the developed product and to reduce the overall development expenses (IEEE, 1990). Galin (2004) explains that software quality has a direct relationship with software testing; hence testing is an important phase of the software development life cycle. According to Perry (1995) about 24% of the overall software budget and 32% of project management budget is allocated for testing. Due to extra pressure to finish projects on time project managers are likely to reduce the testing activities (Galin, 2004). This can bring adverse effects on software quality, therefore to achieve benefit of software testing under limited resources, it becomes necessary to identify the best software testing practices and create a mapping between various existing software methods and tools. This can be achieved by analyzing current testing practices and identifying the improvement potential.

Today software is a crucial part of automobile development; it is predominantly used in functionalities such as driver assistance, vehicle dynamics control, and other active and passive safety systems. With more usage of software in mechatronics implementation, and increase in complexity of systems, the possible challenges arising due to systematic failures and random hardware failures are increased (ISO26262, 2009). Therefore standards are being developed to provide guidance to automotive based industries to minimize safety related risks to a tolerable level (ISO26262, 2009). The currently available standards such as IEC 61508 on functional safety are not completely dedicated towards the automotive domain based industries, and because there is possibility of interpreting IEC 61508 in different ways it is tough to achieve harmony in functional safety within different automotive domain based industries (Schwarz & Buechl, 2009). A new state of the art standard ISO 26262 is being developed in collaboration of 9 countries: Belgium, Canada, France, Germany, Italy, Japan, Sweden, United Kingdom and USA. It is planned to be released as International Standard (IS) in June 2011, the latest draft international standard (DIS) is available since June 2009 (Schwarz & Buechl, 2009). Since this new standard will be soon adopted by all the major automotive domain based industries, it becomes necessary to investigate how the introduction of ISO 26262 standard will affect the existing testing practices in automotive industries.

The scientific contribution of this thesis is the study of various software testing practices in a specific automotive company (Volvo Technology (VTEC)) and investigate how the introduction of the new safety standard ISO 26262 will affect the existing testing practices. Also to give

recommendations for improvement in the existing testing practices and make a pre-study over the ISO 26262 standard, providing suggestions to achieve synergism in terms of testing among different departments within the organization. The intended audiences of this thesis are persons involved in software testing activities, and in particular the testing, development teams, and project managers at VTEC.

This document will firstly present related work and background, then the research objectives, research method and setup for this case study, latter the analysis of the data obtained from interviews and literature review on ISO 26262, and finally discussion and recommendation for improvements based on the patterns emerging from the interview data.
Interview data is presented in three different sections addressing software testing methodologies, tools and standards. The data presented in ISO 26262 sections is divided in two different sections which address introduction of ISO 26262 and its ASIL levels and discussion over the applicability of the ASIL levels to the projects covered under this case study.

2. Related work & Background

There are studies conducted to determine the best testing practices, such as the one by Ram Chillarege, IBM (Chillarege, 1999) and Antonia Bertolino (Bertolino, 2007). These research studies present very significant amount of knowledge on good testing practices, however they are mostly based on theoretical aspects. This study will try to answer similar research questions with support of empirical data collected via an industrial case study. Also a survey was conducted in 2004 to study the software testing practices in Australia by Reed. K. et al. (2004), which provided good insights of software testing practices useful to design this research study. Another recently published research study by Sundmark et al (2010) presents results of an industrial survey on contemporary aspects of software testing using qualitative and quantitative methods. Their study gives crucial information about discrepancies observed between the current practices and the perceptions of respondents which could prove beneficial in shaping future research on software testing, however the explanations for these observed discrepancies were provided based on researchers assumptions, or in some cases the explanations were not clear. In this case study the observed patterns in perception of respondents will be presented and an explanation for the observed anomalies or discrepancies will be explained by using qualitative data.

There are few research work conducted recently to study the adoption of ISO 26262, such as by Schwarz Juergen et al. (2009), Krammer, M et al. (2010), Lemmer, K et al. (2010), Matheis Johannes et al. (2010); but these studies are not specific to software testing practices. In this thesis we study similar aspects but focusing on the software testing field.

This thesis has been carried out at Volvo Technology Corporation (VTEC), an automotive research and development organization within the Volvo group. VTEC holds the focus of research, innovation and development for Volvo Group, VTEC is located at various establishments of Volvo in Europe, North America and Asia. VTEC has different departments each specializing in a certain field of engineering. VTEC intends to strengthen its software development process; starting with an aim to formulate a common base software development process for all its departments. Therefore this thesis was proposed by VTEC's software testing & verification department with an aim to gain insights of testing practices being used in various departments of VTEC. The objectives of this thesis are presented in section 3.1 (research objectives).

3. Research Methodology

This section presents the research objectives and research questions of this work. Also provides a description of the methods used to perform this study and various factors influencing this research study.

3.1 Research Objectives

The following are the aims of this thesis:

1) Mapping test methods and tools used in different departments within VTEC.
2) Identifying the possibilities of improvement in the test practices used within VTEC.
3) Performing a pre-study on the forthcoming automobile industry standard ISO 26262.
4) Enabling knowledge transfer of the test practices used within different departments of VTEC, and also providing suggestions for achieving best results by synergism in terms of testing among various departments of VTEC.

3.2 Research Questions

By addressing the following research questions this thesis aims to achieve the research objectives mentioned above (section 3.1):

1. What are the different methods and practices being employed for software testing in automotive industries? And what is the co-relation of these methods and practices with the nature of the projects?
2. What software testing activities are performed with help of software testing tools in automotive domain, in particular at VTEC?
3. What are the software testing requirements mentioned in ISO 26262 that are applicable for software projects in VTEC and similar automotive industries?

3.3 Research Method

This thesis addresses the above mentioned research objectives and questions by performing an *interpretive case study* on similar lines of the method described by Walsham (1995). Qualitative methods are selected to conduct this study, because they provide much richer data for analysis and concept formation. As a matter of fact, the sample size for this case study is small and therefore any generalization of the results obtained from research based on only quantitative methods and tightly controlled deductive approach might lead to validity threats.

The overall strategy for this research study was:

1) To identify the scope of research by performing systematic literature review.
2) Collecting data via systematic literature review and semi-standardized interviews.
3) Analysis of the collected data based on grounded theory as explained in (Strauss & Corbin, 1998).

During the analysis phase open coding was performed on the data collected from interviews and literature review. The inductive approach was followed to categorize different sources and patterns emerging from the data.

Walsham (1995) explains the two different roles (outside observer, involved researcher) adopted by researchers during case studies. For this research a role of “involved researcher” was taken, as it allows to gain deeper insight of the research subject. However Walsham (1995) explains de-merits of this role as:

> *The involved researcher will be perceived as having a direct personal stake in various views and activities, and other personnel may be more guarded in their expressed interpretations as a consequence. In addition, unless participant observers or action researchers hide their research motives, which could be considered an unethical position (Mumford, 1985), they will still not be regarded as normal employees and thus not total insiders. A final problem with the role of involved researcher is the extreme difficulty of reporting the part one has played in the various matters under consideration. Self-reporting faces the twin dangers of over modesty and self-aggrandizement, and it is particularly difficult to steer a middle path between these two extremes.*

To address the above mentioned issues, the interview respondents were made aware of the motives of the interview and after conducting the interviews a summary of interview transcription was sent to the interview respondents to point out any sensitive data that should be excluded from the research, and also to inform about any miss-interpretations of their statements during the interview. Only after receiving the confirmation from the interview respondents the data was included in the research.

Literature review: As per the guidelines of Webster & Watson (2002), literature review was performed for domain analysis and to generate a theoretical background on testing practices and requirements of ISO 26262 for software testing. Also the recommendations for the improvements on the currently reported practices were given on the basis of literature review.

Interviews: This method was employed to get the qualitative insight of testing practices within the organization. The following are the main theme and sub-themes that were covered during interviews in order to address the research objectives:
Interview Theme: Software Testing Practices in Automotive domain.
Sub Themes:

1) Software testing methodologies and techniques:
   Methodology, stopping criteria, barriers, benefits, expected areas of improvements, generation of test cases, formal methods and software testing metrics
2) Software testing tools:
   Tools used, purpose of the tool, issues, automated tools, level of automation expected.
3) Software testing standards:
   Published or in-house standards that are being employed, and barriers in adopting the standards.

The interviews were designed as a *semi-standardized interviews* (Berg, 2009), because it will help in keeping the *focus of interview* (Steinar, 1996) towards the main theme of interview and still provide the flexibility to adapt the interview to the situations where there is a possibility to gain new insights. The interview respondents were assured about their confidentiality. The questions were framed in such a way that while conducting the interview it was possible to shift between questions belonging to different themes seamlessly. The interview questions were categorised according to the following categories: *open questions (O), probing questions (P), linking questions (L), guiding questions (G), hypothetical questions (H), forcing questions (F), and closed questions (C)*). After framing questions according to the above mentioned categories, the questionnaire was verified in order to check if all the sub-themes of interview were covered. The interview questionnaire is presented in appendix B.

Purposive sampling technique was employed for selecting interview samples and persons involved in software testing activities were contacted to participate in the interviews. More regarding the sampling strategy for interviewing is mentioned in section 3.4 (sampling strategy for interviews). During the interview sessions notes were taken, and all the interviews were recorded (interviewee's permission was taken before recording the interview session), and later the interview recording was transcribed according to *systematic filling systems* method as described by Berg (2009). The transcribed data was summarised and sent to the interviewee for approval. The analysis of the interview data was done in the following way: 1) Referring the interview notes and transcripts; 2) Identification of certain keywords which were frequently appearing; 3) Dividing the interview responses in terms of categories; 4) Observing patterns.

Tools employed for this research were: Cockos Reaper (recording interviews), Google Docs (for scheduling interviews) and Nvivo 9 (qualitative data analysis of collected data).

3.4 Sampling Strategy for Interviews

This case study was focused towards the software testing practices at organization level, and during the case study interviews were conducted within different departments working on different projects and hence allowing to set the focus of this case study at department and project levels both. A draft version of the questionnaire was verified by the supervisors at the industry and university, and based on the feedback a final version of the questionnaire was prepared. Various persons who are involved in software testing practices at VTEC were contacted through email to request their participation in this case study. The preference for selecting samples for interviews was for members of software test teams, and persons involved in testing activities and then for test managers, software project managers and group managers.

To minimize the conclusion validity threat due to small sample, a purposive sampling approach was employed, i.e. the participants were selected because they represented a section of the target group. 30 persons from VTEC were contacted, and a total of 15 persons responded to participate in the interviews. That calculates to a 50% response rate, and given that few persons were out of station or unavailable this response rate can be considered relatively good. Discussions on the response rate of persons for this case study with research methods lecturers and supervisors at university, indicates that the majority of the groups might be using some structured approach for software testing, or at least have a positive mind set towards achieving improvements in their existing practices. Out of 15 respondents one of the respondents was involved in managerial aspects of a group that was majorly involved into mechanical engineering aspects of product development, and therefore the data obtained might not have much significance, hence interviews were conducted with the remaining 14 respondents.

Out of the 14 persons who were interviewed, 5 of them were developers who were also involved in testing activities, and 2 dedicated testers who were also test leaders, 2 managers, and 5 persons who perform multiple roles in the project such as project management, development, testing and requirements engineering. Attaining a good relevant sample was considered very important for this case study. Over 40% of the interviewed persons were involved in the managerial aspects of projects, and 50% of the persons were involved in software testing, and around 36% of persons were mainly developers but were also involved in software testing activities, or had experienced software testing in earlier projects. Over 70% of the interviewed persons had their educational background in computer science, and other respondents who had their educational background in other fields of engineering were working in the organization from many years and have significant

level of understanding of software engineering aspects and terminology. Therefore there are reasons to believe that results by interviewing the above described sample can be considered as "suggestive" towards the software testing practices and not as thoroughly conclusive.

4. Interview results and data analysis

This section depicts the results obtained from the interviews which are concerned with the interview themes mentioned in the research method section. This section is divided into 4 sub-sections, namely: case study setup description, software testing methodologies, software testing tools and software testing standards.

Note: The data presented in this section is extracted from qualitative semi-standardised interviews. After conducting interviews the responses of the interview respondents were grouped based on the project they belong to. The results in some sections present the overall aggregate of the responses of the team members. In case of some projects only 1-2 persons from the project team were interviewed, and since it cannot be said that all the persons involved in a project team have complete information of all aspects of the projects. Therefore statistics presented in these sections should not be taken as exact values, but rather as an indicator for the popularity or extent of usage or certain method, tool, or practice.

4.1 Case study setup description

The 14 interviewees belonged to 2 different departments and 6 different groups in VTEC, and each group was specialized in a certain domain area.
A total of 8 different projects were identified, 5(8) [1] (62.5 %) of these projects were regular product development projects with customers in private sector, and the other 3(8) (37.5 %) projects were research projects funded by government and non-commercial organizations.
4(8) (50%) of these projects were based on embedded software domain, 3(8) (37.5%) were regular software projects, and 1(8) project was based on both embedded and regular PC software domain.

Team sizes: 4(8) projects had team sizes of around 5 members, and 2(8) projects with team size of around 2 members, 1(8) projects with team size of 15 members and another project with 40-45 members in team. It was mainly the research projects that had relatively small team size.

None of the 8 projects involved development of products with high safety criticality on automobiles. 4 project-teams were involved in development of features with very low safety criticality, and the remaining 4 projects did not involve development of any safety critical features.

50% (4(8)) of the projects were old projects (continuing from past 5 years or more) and the remaining 50% were relatively new (which started around past one year). All the old projects were product development projects, and out of the 4 newly initiated projects 3 were research projects and one regular product development project.

Only 3(8) projects had dedicated software testing teams, and out of the remaining 5 teams 3 teams had external testing team or the testing was performed by their customer. It can be interpreted that software testing is also being employed as an "extrinsic" process. The remaining 2 teams had

[1] For reducing the amount of text in many places usage of "X out of Y " is replaced with " X(Y) " , for example:"5 projects out of 8 were using …. " is written as "5(8) projects were using …"

relatively small team and project size, and hence the same person was responsible for performing multiple roles concerned with the project which also included testing.

4.2 Software Testing Methodologies

This section presents the extent of adoption of software testing practices in the various projects that were involved in this case study and also discusses the possible factors affecting the adoption of these factor practices.

4.2.1 Methods and practices

This section presents the interview response for the questions concerned with software testing methods and practices.

The graph shown in figure 1 represents the response of the interview respondents, type of project vs. software testing approach. x- Axis of the graph represents methodologies employed (namely ad-hoc, structured, semi-structured), z-axis represents the type of projects (product development and research) and y-axis shows the number of responses.
5(8) project-teams were employing some sort of structured approach for software testing; script-based testing was more common practice, reported by 6 teams. 2(8) teams were following exploratory testing with session based test management, and these two teams were also using script-based software testing.

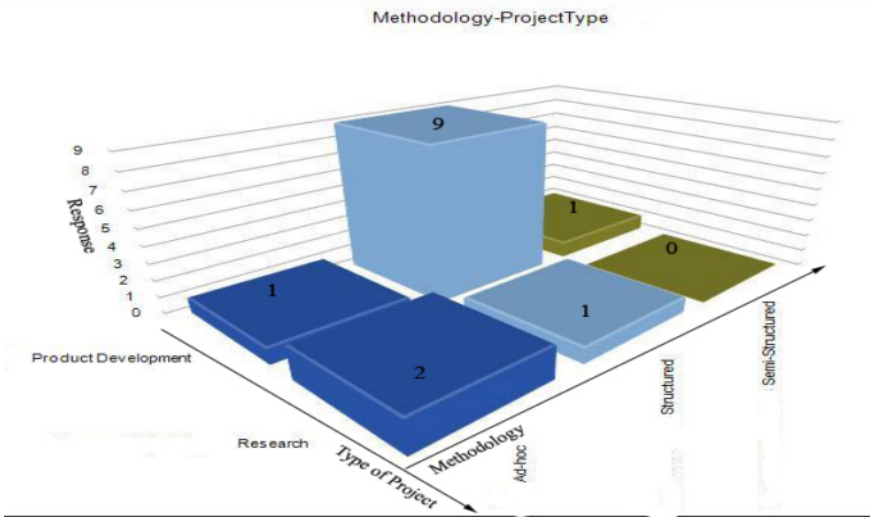


Figure 1: Response of persons from different project type towards methodology employed.

There were 3(8) teams with un-structured (ad-hoc) approach towards software testing. All the teams with ad-hoc approach for testing had projects that were recently started. Also 2 of them were research projects with small team size of 2 members in one project and 4 members in another.
It was also noticed that all the teams with ad-hoc testing approach did not had a dedicated testing team. 2 teams with ad-hoc testing approach had external software testing team, and another team had small team size, so the team members were responsible for performing multiple roles.

4 out of 5 teams working with product development projects had more structured approach towards software testing. It was also seen that most of the projects with structured testing approach had a

dedicated testing team. Also 4 out of 5 teams working with a structured software testing approach were also working on old projects with project age more than 5 years; this suggests that over time projects tend to move towards more mature software testing process.

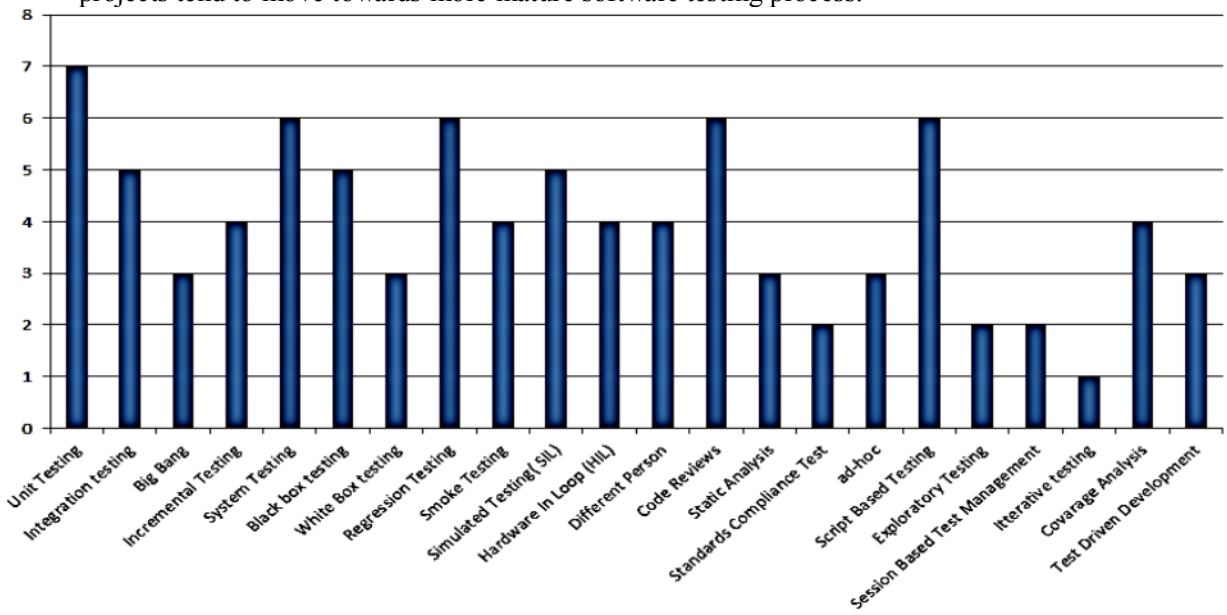


Figure 2: Response towards various software testing activities

The graph shown in figure 2 represents the popularity of the different testing methods and techniques among the various project teams in VTEC.

Unit testing was fairly popular practice as it was reported by 7(8) teams, and 5(8) teams reported usage of integration testing, 3(8) teams which did not perform integration testing either had an external testing team or testing was performed by the project–customer.
In case of two project teams the interviews respondents mentioned that the projects they are working on are in alpha stage (proof of concept stage) and requirements are not strictly defined and therefore software testing is not very vigorously applied.

It was noticed that incremental testing was the more popular type of integration testing technique (used by 4 teams) compared to the "big bang" testing (which received responses from 3 teams). One of the teams employed both big-bang and incremental testing approach. But the differences in responses were not sufficient to make any concrete statement on this type of preference.
System testing was fairly popular among the teams, reported by 6 teams. Black box testing was more widely used system testing technique (5(6) teams) compared to white box testing (which was employed by 3(6) teams.

Regression testing and Code reviews were quiet extensively used practices among project-teams at VTEC (reported by 6 teams). Respondents from 4 out of 6 projects in which the code reviewing techniques were employed mentioned that the reviews were performed by a different person other than the person one who wrote the code.

Smoke testing was reported to be used in 4 projects. Coverage analysis and test driven development received 4 and 3 responses each respectively and respondents from 2 teams mentioned usage of tests to check the code compliance with standards that they were employing.

4.2.2 Test planning activities

This section presents the interview response related to the questions regarding test planning activities. Figure 3 presents the data collected from interviews regarding test planning. Horizontal - axis of the graph shows the different project teams, and their responses are plotted against vertical-axis which represents the planning strategy.

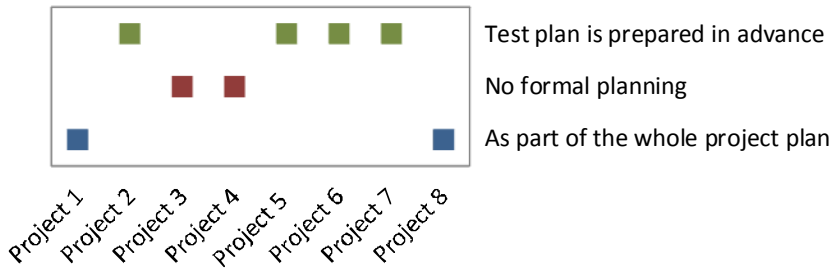


Figure 3: Software test planning

4(8) of the teams prepared software test plan in advance, before the software testing phase. 3 of the 4 teams with this strategy were involved in regular product development projects and dedicated testing team.

2(8) teams had their testing activities planned along with the planning of the entire project. Respondents from these projects mentioned that these plans are not very detailed. Detailed test plans are made during testing phase of the projects, and also these two teams did not had a dedicated testing team. One of these teams had an external test team and another team was working on a research project with a small team.

The remaining two teams had no formal test plan prepared in advance of software testing phase, also these teams were following ad-hoc testing methods, one of these two teams has an external testing team and the other team is associated with a research project with a small team size of 1-2 persons.

Two of the teams that had test planning in advance were following session based test management along with exploratory testing, and one of these two project –team was employing Scrum methodology as their product development paradigm and the other team was also following agile methods.

4.2.3 Pre-requisites for testing

This section presents and discusses about the interview responses of respondents towards the questions regarding pre-requisites for software testing. Figure 4 presents the mapping of responses made by interviewees, all the responses are aggregated to represent the response of the project team that they belong to. There were 9 different pre-requisites for software testing identified from the interview responses, these pre-requisites are presented as vertical axis in figure 4 and the horizontal axis represents the 8 project-teams.

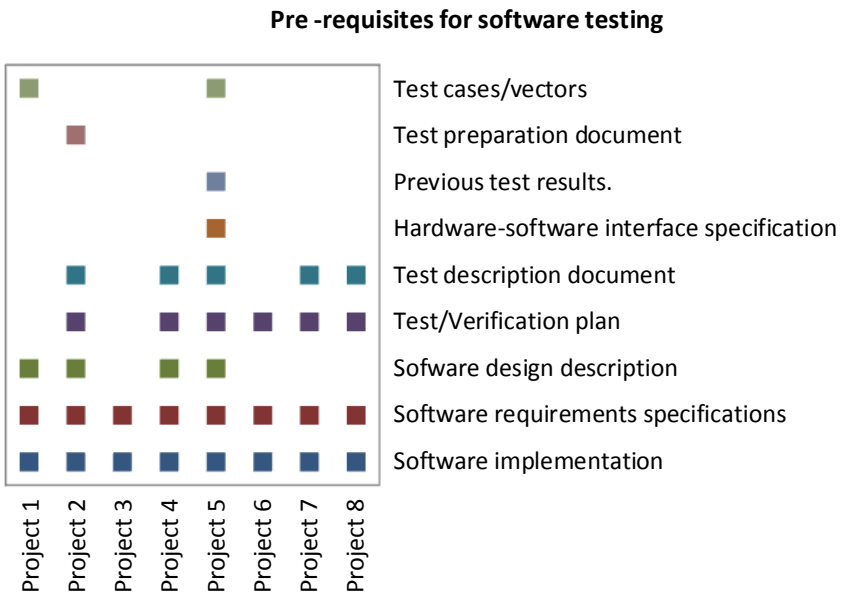


Figure 4: Pre –requisites for testing

Software requirements specification and *software unit implementation* were the main pre-requisites for every team; in case of 3 projects software unit models were used in place of software implementation.

6(8) teams had *software test/verification plan* as one of the pre-requisites for their testing. 1 of the 2 teams that did not mention this requirement did not had any structured testing approach and were also associated with a research project and had a team size of 1-2 members. The other team had external testing team, and a respondent from this team mentioned that planning for testing phase is done during the initial project planning phase and there is no explicit test /verification plan.
5(8) teams had *test description document* as pre-requisites. Out of the remaining 3(8) teams that did not had this as a pre-requisite; one team was employing exploratory testing, another team had external testing team and the remaining one team was using ad-hoc approach.

Software design description document was reported by respondents of 4 teams. 2(8) teams mentioned *test cases/vectors* as pre-requisites and other pre-requisites such as *previous test results* and *test preparation document* received one response each.

4.2.4 Training and guidelines

All the teams were reported to have some rudimentary guidelines regarding software testing; Teams share the guidelines for testing during team meetings, or mention them in the software verification plan of test-description document. Few teams had online-wiki where they can refer to the guidelines regarding software testing. In one of the cases a team had persons who provide training regarding software testing methods to other members of the team.

4.2.5 Test cases

This section presents the summary of responses made by interview respondents towards the questions regarding test case selection criteria. Figure 5 shows the response of different project

teams towards test case derivation methods. There were 6 different test case derivation methods identified from the interviews, mentioned on the vertical axis of the graph presented in figure 5. Horizontal axis of the graph represents the 8 different project teams in this case study.

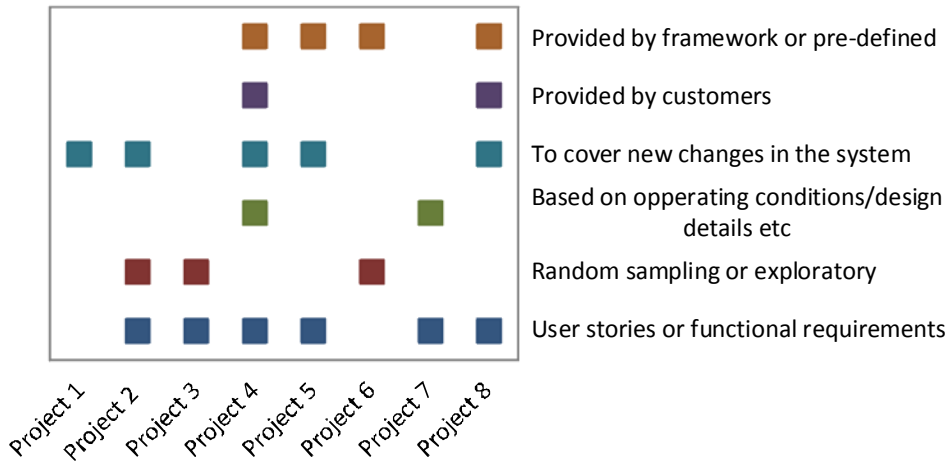


Figure 5: Test Case Sources

User stories or *functional requirements* were the most popular sources of test cases, reported by 6(8) teams. 5(8) teams generate test cases in order to cover the newly introduced changes to the system; this practice was more prevalent in the teams that were working on old (>5 yrs.) projects as they were implementing new features or improving old functionalities into the system.
Next popular source for test cases was pre-defined test cases or test cases provided by testing frameworks which was reported by 4(8) teams, but the actual number could be much higher as at least 6 teams were using script based testing methods.

3(8) teams used exploratory or *random sampling* (Galin, 2004) techniques for generating test cases. 2 out of these three teams has dedicated testing team, and were working on regular product development projects and the remaining one project-team was associated to a research project with relatively small team size (1-2 persons).

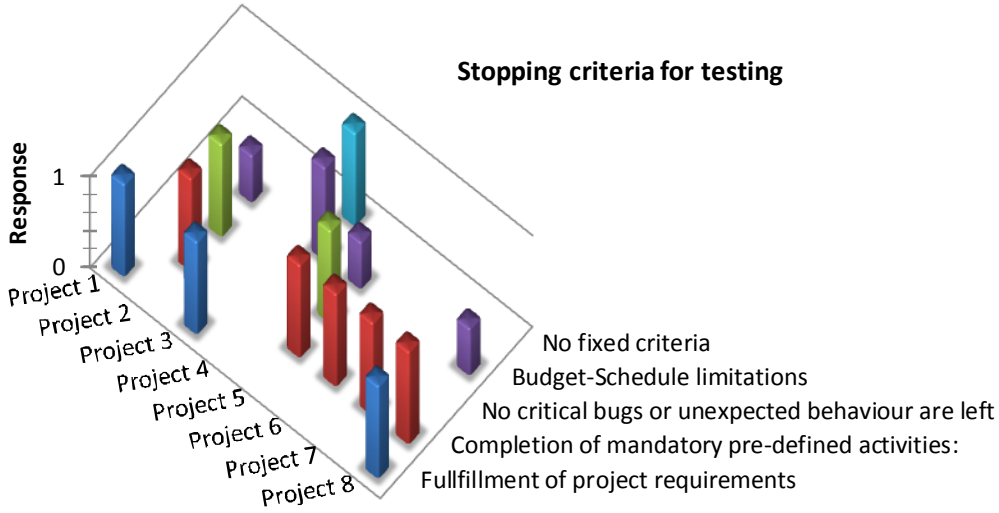
2 teams had their test cases based on the operating conditions or design details of the system. In case of 2(8) teams test cases are provided by the customer, for these two teams software testing is either performed by an external testing team or the customer.
Most of the test cases were based on black box testing approach and this co-relates with the data in section 4.2.1 which shows that teams had more preference towards black box testing.

4.2.6 Stopping criteria

This section discusses the results of the interview questions concerned with stopping criteria employed by teams to conclude their software testing. Figure 6 shows the responses of the persons belonging to different projects towards stopping criteria[2] for software testing. There were 5 stopping criteria identified from the interview responses, these criteria are mentioned on the right

[2] Stopping criteria are not mutually exclusive; therefore a project-team can have multiple stopping criteria for software testing.

side of the graph, and the bars represent the aggregated response of the interview respondents belonging to a specific project team.



	Project 1	Project 2	Project 3	Project 4	Project 5	Project 6	Project 7	Project 8
Fullfillment of project requirements	1		1					1
Completion of mandatory pre-defined activities:		1			1	1	1	1
No critical bugs or unexpected behaviour are left		1			1			
Budget-Schedule limitations		0,5		1	0,5			0,5
No fixed criteria				1				

Figure 6: Stopping criteria for testing in different projects, Response ("1" denotes "Yes" , "0.5" denotes "May be")

The table presented in figure 6 represents the data presented in the graph; response "1" indicates that the interview respondents in a project team mentioned usage of a stopping criteria in that particular row and "0.5" means that the stopping criteria was not employed on all occasions (rather in some rare occasions).

The most popular trend for stopping criteria for testing was *completion of mandatory pre-defined activities* such as:

1. Execution of all the test cases or test scripts,
2. Activities mentioned in verification plan,
3. Documentation of reports and results,
4. Completion of planned sessions (in case of teams with session based test management).

5 project teams had this kind of stopping criteria.

The next popular stopping criterion was *fulfilment of project requirements*, or when the product under development clears the criteria for acceptance test; 3 teams responded to this criteria. Among the teams that did not had this criteria 3 teams were following agile methods, and hence the requirements that are not achieved or have issues that were not addressed in the current iteration were addressed in next iteration (sprint). The remaining 2(5) teams that did not had these criteria; one team had external testing team, and another team was following ad-hoc methodology.

Only one team had budget and schedule limitations as a stopping criterion for software testing, respondents from 3 teams mentioned that *budget-schedule constraints* was a stopping criteria for them but on very rare occasions.

For two teams *absence of critical bugs or unexplained behaviour in the product* was one of the stopping criteria, and one team had no specific stopping criteria, but this team had an external testing team.

4.2.7 Testing environment

All the project teams showed satisfaction with their testing environment, and in 3 cases the respondents mentioned that their software testing environment is one of the strengths of their testing.

All the embedded system related projects were using CAN bus simulation tools for test environment simulation. In-order to reduce the testing cost, teams first performed the testing on a simulated environment and then tested on the target environment. 3 respondents mentioned that the CAN bus simulation environments were prone to timing errors, but there was nothing too critical.
The teams working with software development projects mentioned they use virtual machines to emulate the target environment.

4.2.8 Software testing metrics

There was relatively less response received towards questions regarding software testing metrics, and on few occasions some of the interviewees found the questions regarding software testing metrics ambiguous (but explanations of questions were provided on those occasions). This suggests that usage of metrics is not very common.

Figure 7 shows preference of different project-teams towards employment of metrics in software testing. Both usage and non-usage of metrics received equal response (50% each). However this data co-relates to the data in section 4.2.6 "Stopping Criteria", by comparing the data in both sections we can see that the project-teams that were employing software testing metrics also had more sophisticated stopping criteria for software testing.

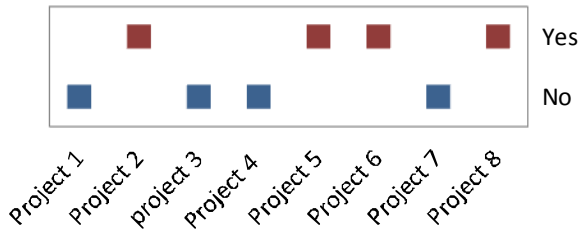


Figure 7: Usage of metrics in software testing

This data also speaks in favour of the point made by Reed et al. (2004) which says that

> "*Defining stopping criteria is never easy, especially without the usage criteria based on metrics. Also usage of stopping criteria based upon non-statistical methods can be considered as potentially risky.* "

After comparing with the data in Section 4.2.1 it can be observed that out of 4(8) project-teams that did not have software testing metrics, 3 were following ad-hoc testing methodology. All the 4

project-teams that reported usage of software testing metrics have structured approach towards software testing.

Among the teams that were employing software testing metrics, most common type of testing metric was *bug counting*. 2 teams that used session based test management employed metrics related to test management. One of the teams reported that they were using metrics concerned with coverage analysis and cyclomatic complexities. During interviews, persons from the teams that were employing software testing metrics mentioned the need of improving the usage of metrics:

> "A*s it helps in performing the analysis of the testing, and helps to see what kind of testing is catching most of the bugs and also the typical amount of bugs found during different testing phases.* "

In the interview one of the persons from a research oriented project mentioned:

> "*Because of the nature of the test cases it is tough to employ metrics, in our testing we see the variation in the test results with respect to the previous release of the software if there is a large variation then we mark it as bug.*"

and few other persons mentioned that sometimes there is a need of visual examinations during the testing, these points reflects some scenarios in which respondents feel application of metrics is tricky.

4.2.9 Formal Methods

None of the projects reported any concrete usage of formal methods or specifications, and often interviewees found the questions regarding formal methods a bit ambiguous. These results were surprising considering the amount of research being done on formal methods.

An explanation for this could be, as it was discussed earlier in section 4.1; that none of the projects are associated with high safety critical features in vehicle. Therefore the usage of formal methods is minimized. Also one of the interview respondent mentioned that their team is employing test-driven-development approach for product development and this allows them to perform testing and refine the requirements and they are able to fulfil customer requirements quiet well, and hence they haven't given much consideration towards formal methods. Another respondent mentioned that because of employment of agile methods, and good communication with customer any misinterpretation of requirements it is quickly sorted out and also because they are not working with any high safety critical feature the need of employing formal methods was not felt.

One of the interview respondents mentioned that

> "*It could be good to employ them, after considering the time and cost of implementation*".

Another respondent mentioned that

> "*introduction of formal methods will be very useful considering the amount of time being spent on dealing with un-clear requirements, but it depends on nature of the projects and customer, it will be tough to employ formal methods.* "

Also there is not much proposal of usage of formal methods in software testing in ISO/DIS 26262 (please refer "Methods for unit testing (a)", presented in appendix A), even for high safety critical features the recommendation of formal methods is not high. Researchers working in the field of software engineering should address these points in order to make formal methods more favourable to automotive industry.

4.2.10 Impressions and expected improvements on methodologies employed

This section presents the impressions of interview respondents to the question "What are your impressions on the testing methodology that your team has employed? And what improvements do you expect?"

After observing various patterns emerging from the responses of this question; it is seen that *exploratory testing* was the most popular subject among the interview respondents (8(14)) even though only two teams were using this type of methodology yet this practice was fairly popular. Figure 8 shows the extent of different methodologies employed by different project teams.

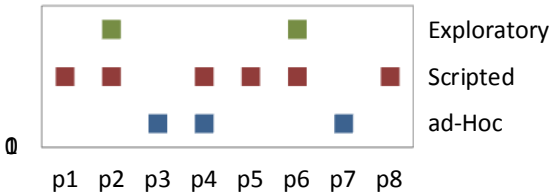


Figure 8: Testing Methodology employed in different projects

Most of the persons from the 4(6) teams that were working with purely script-based testing mentioned a need for introduction of the exploratory testing approach in order to complement their existing script-based testing approach. Also the there was a similarity observed in the responses of the persons regarding this subject. All the persons who mentioned the need for employment of exploratory testing mentioned that the script-based testing is restrictive, and it was difficult to see the bugs in any undocumented behaviour of system and "the human perspective" allows performing testing out of box.

The responses from the teams that were following exploratory testing approach indicated that they were fairly satisfied with their approach of working, and their responses showed good co-relations with the responses of the persons that mentioned the need for introduction of exploratory testing. One of the respondents mentioned that one of the main reasons for the introduction of exploratory testing was that the

> *"Old procedure (script based testing) was very much dependent on test cases and which intern were dependent on requirements, and if there was a change in requirements quite often, and it was tough to cope up with. Therefore exploratory testing accompanied with session based test management was introduced"*

A similar statement was made by another respondent who mentioned "*exploratory testing is very agile in nature and it is easy to adapt to changes*", another respondent from the team that employs exploratory testing also mentioned that:

> "*Exploratory testing is more intuitive in nature and keeps software tester always interested, and also helps in increasing the knowledge of system significantly*"

Another pattern that was observed was that at least one person from every project-team mentioned the need of introduction of more *automation*, or the barriers for introduction of automation (the barriers for automation are discussed in section 4.3.5). One of the respondents mentioned that

> "*Due to complexity of the system the human errors are bound to occur, hence introduction of more atomized procedures will help in having better results*"

The other responses mostly pointed towards introduction or achieving more automation of various activities such as unit testing, regression testing, report generation, test vectors generation and test results analysis. One of the respondents mentioned that

> "*If automation is introduced at system testing level then we can do more testing, and hence achieve better quality*"

It was also observed (5 responses) that there was a concern regarding the methodologies being employed. Respondents mentioned about the need for introduction of a common base process/methodology/practices as the results obtained from the existing methods are very much dependent on the experience of the persons performing testing, and therefore it requires more effort to enable knowledge transfer before the new tester starts testing. One of the respondents mentioned:

> "*It will be good to have some sort of common base methods, and the other more advanced methods could be used as an ad-on, this will enable easy switching of persons between different projects, for example: if there is a project that does not requires too many resources then persons from those projects can be easily moved into projects that need more resources*"

4 respondents mentioned that there is a need for introduction of more incremental testing approach, and were opposed to usage of big-bang approach for integration testing, one of the respondents mentioned that the big bang testing approach was followed in their project because of historical reasons, i.e.,

> "*When the project was started it wasn't as complex and large as it is now, and hence big-bang approach was employed and it still continues*"

Another respondent mentioned that in their project they use one automated test script to performing testing on all the requirements and due to growing complexity of the system it is becoming tough to achieve it. Also the big-bang approach takes a lot of effort to diagnose the root cause of the failure.

The next popular subject was test-driven development, 4 respondents mentioned that usage of test-driven development is one of the positive points in their testing practices. One of the respondents mentioned that employment of test driven development has helped in deriving more concrete requirements from the high level requirements in the initial phases of project. The following quoted text is a response from one of the interviewee:

> "*Test driven development has benefits such as it helps in ensuring that the application is written for testability, as the developers must consider how to test the application from the outset, rather than worrying about it later. It also ensures that tests for every feature will be written*"

Another pattern that was observed in the responses was requirement for dedicated testers. One of the respondents mentioned that due to lack of dedicated testers the test cases for a particular feature are written by the developer of that feature, and this introduces the risk of ignoring some aspects due to bias. A respondent from a project with small team (1-2 persons) mentioned that

> "*Because the developer and tester is same person on many occasions it introduces a risk of overlooking some potential fault areas due to bias; hence there is a need to introduce dedicated software testers in the team*"

Two other respondents had similar views, but in terms of code reviews.

The following are the points mentioned by individual respondents as positives in their practices:

1) Exploratory testing and session based test management
2) Short iterations
3) Unit testing
4) Informal code reviews

5) Usage of test-management tools.
6) Well planned and structured approach.
7) Good testing environment.
8) Modifiability of the testing system to cover new changes in system

The following are the points mentioned by individual respondents as expected improvements in testing practices:

1) Introduction of better documentation, such that old test cases can be accessed.
2) Emphasis on the testability of the system during development.
3) Providing better definition of different activities to be performed during different types of testing phases
4) Improvement in project planning with emphasis on testing phase.
5) More emphasis on unit testing
6) Allocation of more resources on testing.

4.3 Software Testing Tools

This section presents the responses made by interview respondents towards the questions regarding software testing tools and also provides some discussion on the trends observed in the interview responses. In this section firstly the activities performed by various project-teams with help tools are discussed, later requirements for any specialized tools and impression of interview respondents about the tools they are employing are presented, and finally the barriers these project-teams are facing to achieve more automation are discussed.

4.3.1 Activities performed by the tools

The graph show in figure 9 presents the response of interview respondents towards the questions regarding software testing tools and activities performed by them. The individual interview responses are aggregated and presented as project- team response.

There were a total of 12 distinct activities performed with tools were identified from the interviews, these activities are presented on the right hand side as vertical axis of the graph mentioned in figure 4, and the horizontal axis represents the 8 project-teams.
As expected from the data in section 4.2.1 unit testing was the most popular activity performed with automated tools, reported by 7(8) project-teams.

Software in loop (SIL) testing was reported to be performed by 6(8) teams, 4(8) teams reported usage of automated test rigs for hardware in loop (HIL) testing and all of these teams were associated with embedded systems projects.

Test-environment simulation was next popular activity this was practiced by 5(8) teams, simulation of CAN bus was most common it was reported by 4(8) teams. All of these 4 projects were associated to embedded systems projects, and the remaining one project-team was associated with regular software development product they used tools to simulate customer software environment.

5(8) project teams reported that they have been employing automated tools to generate documentations such as: test case, test design/description, test reports, and test results analysis. 4(8) teams reported usage of testing frameworks to execute test scripts, and all these 4 project teams had their own in-house tool to run the test-scripts. 4(8) teams reported usage of developer tools to

perform software debugging. 3 teams all working with old (~10yrs) projects were using test management tools.

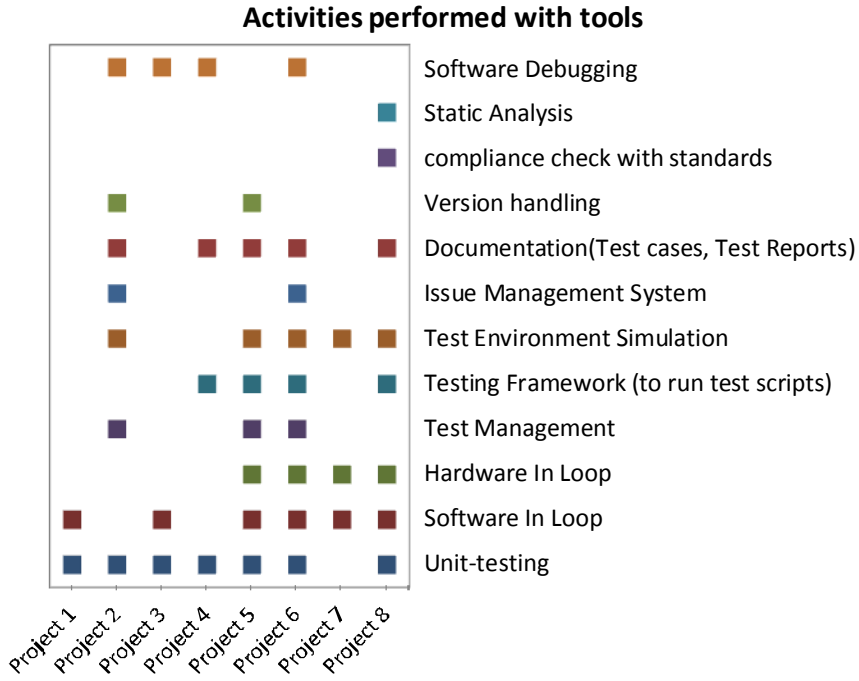


Figure 9: Activities performed with tools in different projects-teams

2(8) teams were using issue management systems and version handling systems. The tools for performing static analysis and tool for checking compliance with standards received response from one project-team each.

4.3.2 Requirements of specialized tools

On being asked "if there is a requirement of any specialized tool because of the domain of the project?" 6 teams responded yes (75%). There was no pattern observed in the responses for this question, every team mentioned requirement of a different type of tool. Respondents of one of the team working with embedded systems project mentioned the need for more tools for test environment simulation and tools to compare Simulink models. Another team mentioned need for tools to automate regression testing. Static and coverage analysis tools received one response each. Respondent from one of the teams which was working on an embedded system project mentioned:

> "*Because of the diversity and complexity of the system it is tough to find one tool that does everything, therefore most of the tools we have is for one specific purpose, and we develop our own in-house tools*"

4.3.3 Impressions on tools

Most of the teams (6(8), 75%) teams mentioned that their impression on the tool that they were using is satisfactory, one of the teams mentioned requirement for more support regarding problems with calibration on SIL testing tool. Another team mentioned need of more tools to perform unit and integration testing, and also improvement in software testing repository.

4.3.4 Barriers in automation

The following are the summary of the points mentioned by the interview respondents about the barriers faced to employ more automation in their projects:

1) Changing requirements were one of the most common barriers. The following are the excerpts of the comments made by respondents which points towards this:

   "*Automation of testing will be fruitful only if it is used in projects where requirements don't change that frequently*" "*If you change anything in the systems code, then you need to make changes to the automated testing code too, in-order to cover those changes*"

2) Another person mentioned that because most of their tests require visual examination it is tough to incorporate automated testing in such scenario.

3) One of the test leaders mentioned that automated testing requires more training, and it is tough to find persons with good knowledge in automated testing due to its high requirements.

4) Budget and schedule limitations was another barrier reported by respondents for automated testing, many projects had small team size and therefore it is tough to allocate separate resources for enabling automation. The following is an excerpt of the response from one of the interview respondents:

   "*Writing automated tests takes time, and also we require time to select the tools for automation of these tests, and due to scheduling limitations it is tough to do*"

4.4 Software Testing Standards

This section presents the analysis of the results obtained from the questions concerned with usage of published standards in software testing and the barriers faced in employing these standards.

4.4.1 Standards being employed

Only 1(8) team was employing published standards, this team was following MISRA (Motor Industry Software Reliability Association). 3(8) teams were following their own in-house standards and remaining 4(8) teams were not following any standards.

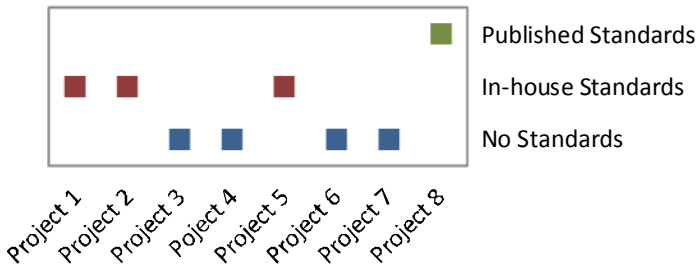


Figure 10: Software testing standards being employed

The graph shown in figure 10 shows the responses made by different project-teams regarding their employment of published and in-house standards.

All the project-teams that were following standards had a structured software testing methodology, and out of 4 teams that were not following any standards 3 teams were following ad-hoc testing.

4.4.2 Barriers in adopting standards

To get insights about this low response towards published standards, questions regarding barriers in terms of adoption of standards were asked. One of the respondents mentioned that

> "*in order to adopt a standards we have to mature our process and change the way of working, and it is always tough to do it, and it requires more effort*" another respondent mentioned that "*adoption of a new standard is a costly and time consuming process, and also it is not feasible to implement a standard for one only group within a big organization*"

One of the interview respondents from a project-team that was following MISRA mentioned that they had to modify or go around certain rules mentioned in MISRA C, as it is not possible to meet those rules due to some technical factors associated with product domain. Therefore they mark all the deviations that they have made in code with respect to the compliance requirements of MISRA and make a MISRA compliance report and submit it to the customer.

This low response towards the published standards and the above mentioned statements suggest that there is a need to introduce a significant level of improvement in the existing standards, mostly in terms of cost-effectiveness and effort required while adopting of the standards. Also it appears that previously published standards such as MISRA are deficient in covering some aspects.

5. ISO 26262

This section presents some aspects of ISO 26262 which are concerned with the scope of this research study (i.e. Software testing). This section describes risk level (ASIL) classification mentioned in ISO 26262 and later based on the interview data the applicability of risk level classification to the projects involved in this study is discussed.

Automobile safety standards were introduced in order to address the rising level of risk due to increase in software dependent features in vehicles. In 1994 MISRA (Motor Industry Safety and Reliability Association) standard was release and later in 2000, IEC 61508 was introduced but IEC 61508 was a more generalized standard and there was a possibility that different organizations can make different interpretations of it and therefore making it tough to achieve harmony among industries in automotive domain (Schwarz & Buechl, 2009). To address this ISO 26262 was introduced; this new standard is basically adaptation of IEC 61508 but more specific to automotive domain.

Notice: This section is written based on the ISO/FDIS 26262:2010(E), i.e. the Draft International Standard, ISO 26262 is not yet an international standard, and the contents of the ISO/DIS 26262 can be subjected to change in future. Therefore the intention of this paper is just to provide a pre-study on the available draft version ISO/DIS 26262.

5.1 ASIL levels

The main idea of ISO 26262 is to identify the level of risk within a vehicle feature, and define an acceptable level of risk (tolerance level, or a residual risk level) on safety requirements and then perform validation in order to see if the defined goals are achieved or not. The technical risk reduction levels in ISO 26262 are defined in terms of 4 different classifications, termed as ASIL (Automotive Safety Integrity Level), each ASIL level defines a specific level of safety requirement. ASIL can be defined based on severity, exposure and controllability of a fault.

*ASIL = Severity * Exposure * Controllability*
Severity, exposure and control of a risk according to ISO 26262 are classified as follows in table 2, table 3, and table 4 respectively; the information presented in these tables are presented from Rogger Rivett's work (Rivett, 2009).

Table 1: Severity Classification , (Rivett, 2009)

Severity: S0	S1	S2	S3
No injuries	Light and moderate injuries	Severe and life threatening (survival probable)	Fatal injuries (survival uncertain)

Table 2: Exposure Classification , (Rivett, 2009)

Exposure: E0	E1	E2	E3	E4
Incredibly low	Very low probability	Low Probability: <1% of average operation time	Medium probability: 1-10% of average operating time	High probability: > 10% of average operating time.

Table 3: Controlability classification , (Rivett, 2009)

Controllability: C0	C1	C2	C3
Controllable in general	Simply controllable, 99% or more of drivers or other traffic participants should be able to avoid this situation	Normally controllable 90% or more of drivers or other traffic participants should be able to avoid this situation	Difficult to control or Uncontrollable , <90% of drivers or other traffic participants will be able to avoid this situation

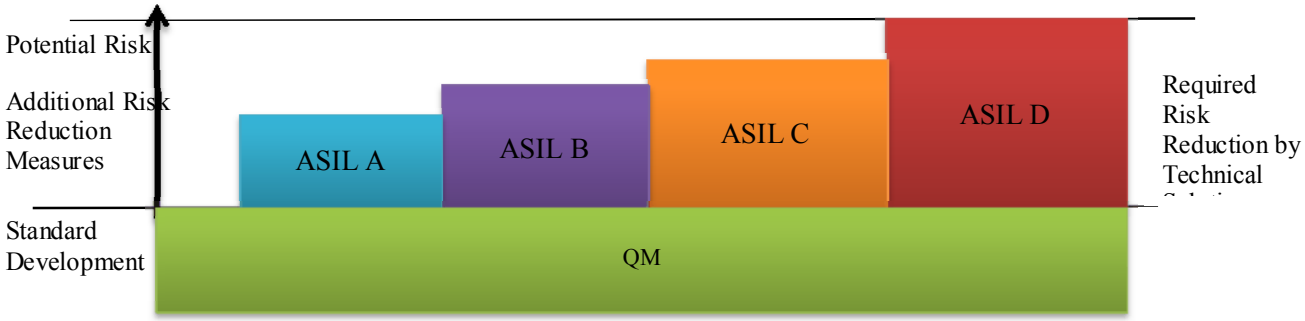


Figure 11: An illustration of ASIL Levels of ISO 26262 (From internal presentation on ISO-26262 at VTEC by Olof Bridal)

Figure 11 presents a visual explanation of ASIL levels of ISO 26262 based on the extent of risk reduction activities required to be performed.

ASILs describe necessary software design, development and quality assurance activities which when performed the risk level are reduced to a tolerant value. ASIL level D requires the maximum

risk reduction activities, and ASIL C requires less extent of risk reduction, and so on ASIL A requires the least extent of risk reduction activities. For ASIL QM there are no activities specified in ISO 26262, it is assumed that the risk involved in QM level functionalities is under tolerable value and hence standard development practices can be employed.

		Control		
Severity	**Exposure**	C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	ASIL A
	E4	QM	ASIL A	ASIL B
S2	E1	QM	QM	QM
	E2	QM	QM	ASIL A
	E3	QM	ASIL A	ASIL B
	E4	ASIL A	ASIL B	ASIL C
S3	E1	QM	QM	ASIL A
	E2	QM	ASIL A	ASIL B
	E3	ASIL A	ASIL B	ASIL C
	E4	ASIL B	ASIL C	ASIL D

Figure 12: ASIL level classification based on severity, exposure and control parameters (Rivett, 2009)

Figure 12 presents a classification of ASIL levels based on *severity, exposure and control* factors. All the faults with severity S0 are under QM (Quality management) ASIL, This category is defined for the risks that are too low to be considered. The following table presents much detailed explanation of categorization of ASIL levels.

5.2 ASIL level of projects at VTEC

As discussed earlier in section 4.1 none of the 8 projects that were involved in this case study were associated with any high safety critical feature. 6(8) project-teams were working with development of features with very low or no safety criticality. Other 2 (8) project-teams were working on projects that involve development of systems that are not part of vehicles. The scope of ISO 26262 (ISO26262, 2009) is confined to regular passenger vehicles (with mass up to 3500 kg), and is meant to address the possible hazards that could occur due to defective functioning of E/E systems that are installed in passenger vehicles and are associated to safety related functionalities. ISO 26262 does not addresses any specialized vehicles, also the physical hazards such as fire, electric shock, radiation, etc. are not addressed unless the hazard is caused by the malfunctioning of the E/E system installed in the vehicle directly.

Based on the scope defined in ISO 26262 the new standard is not applicable for the two projects which are associated to development of systems that are not meant to be installed in vehicles. Out of 6 projects that are associated to the development of systems that will be integrated in vehicles, 2 projects were concerned with intelligent transport systems which provide driver assistance functionalities, and remaining 4(6) projects were associated with in-vehicle systems that integrates with vehicle electronic architecture. Out of these 4 in-vehicle embedded systems project; 1 project was associated with functionalities for trucks, therefore ISO 26262 is not applicable for this project,

and another project is associated with systems that will be mainly deployed on trucks, hence ISO 26262 can be partially applicable.

However as word of mouth it is being said that in a span of 2-3 years ISO 26262 will be made generally applicable for all the projects concerned with development of features for road vehicle (at least in VTEC). Also it is difficult to introduce a standard for just some specific projects; therefore it is more likely that if ISO 26262 is introduced then it will be applicable for all the projects in VTEC.

To determine the ASIL level the interview respondents were asked questions regarding safety criticality of the features being developed in their project. Based on the interview response *severity* associated with the faults in these systems can be S0 (table1 in section 5.1) and S1 on rare occasion in one of the projects. Data regarding *exposure* of faults associated with these features is not available. Considering the nature of the features of the system being developed in these systems and the interview response; it can be said that even the maximum far-fetched value of *control* parameter associated with the faults in these projects can be C2 (table 3 section 5.1).
Based on the *risk graph* presented in figure 11 in sections 5.1; the overall ASIL level for the features being developed in these projects can be ASIL-QM, and in some rare far-fetched cases ASIL- A. The software testing requirements for different ASIL levels mentioned in ISO /DIS 26262 are presented in appendix 9A (Software Testing Requirements for different ASIL levels).

6. Discussion & Recommendations for improvements

This section presents the discussion on the most common software testing practices mentioned by the interviewees, and some recommendations for improvements in their current practices. This section does not address every aspect of software testing comprehensively but instead focuses mostly on the aspects that are identified in section 4.2.10 (impression and expected improvements on methodologies employed). Discussion presented in this section is based on the interview results and literature review.

1) **Combination of exploratory testing with script based testing, session based test management and metrics:**

Bach mentions that usage of exploratory testing will be beneficial or will fit correctly into certain situations. For sake of clarity the following is quoted from (Bach J. , 2003) to illustrate such situations:

- *"You need to provide rapid feedback on a new product or feature.*
- *You need to learn the product quickly.*
- *You have already tested using scripts, and seek to diversify the testing.*
- *You want to find the single most important bug in the shortest time.*
- *You want to check the work of another tester by doing a brief independent investigation.*
- *You want to investigate and isolate a particular defect.*
- *You want to investigate the status of a particular risk, in order to evaluate the need for scripted tests in that area.*
- *Improvising on scripted tests.*
- *Interpreting vague test instructions.*
- *Product analysis and test planning.*
- *Improving existing tests.*

- *Writing new test scripts.*
- *Regression testing based on old bug reports.*
- *Testing based on reading the user manual and checking each assertion. "*

In the same article, Bach also mentions that the advantages of exploratory testing is that often a tester is assigned to one set of components, such that the tester has an uninterrupted learning curve and the project could benefit a lot from this. One of the interview respondents from the team that uses exploratory testing mentioned a similar point related to this:

> "*Another advantage is that tester enjoys testing because tester does not need to run the same test cases again and again. This method is more intuitive in nature and people enjoy working this way some of the consultants here are working from long time. And I think this method is one of the reasons*".

A similar point was mentioned by another respondent from the project team (which wants to employ exploratory testing):

> "*I think there are more possibilities to discover bugs effectively with a human perspective, we have automated tools running the same scripts and test cases all the time, so it is difficult to see bugs in un-documented part, with a human perspective you can think out of box, and perform testing*". Another respondents mentioned "*Main advantages are that you are testing outside the box if you follow a pre-documented procedure you are just testing the expected results of the requirements and in exploratory testing you can test anything that comes to your mind and it is more easy to detect failures, because you can come up with all sorts of unexpected testing so it is more effective*."

After observing repeating patterns of responses similar to that mentioned above, it can be said that exploratory testing could be one of the promising recommendations to be introduced in VTEC's testing practices.

However the respondents from the teams employing exploratory testing showed concern over the fact that exploratory testing depends a lot on the testers experience with the system, and this was one of the concerns from manager's point of view also. As it requires significant effort to enable knowledge transfer to the new tester in the team before the new tester could start performing testing effectively. Therefore it may have a significant effect on the project-team's testing if there is a change in team members.

Tinkham & Kanner (2003) mention that a testers experience depends upon the learning style of the tester, and hence learning style is one of the important variables of exploratory testing. Tinkham & Kanner (2003) also provides methods to determine the learning style of the tester and based on the learning style they have provided recommendations to perform exploratory testing in an effective way. Tinkham & Kanner (2003) provide a good insight on co-relation of learning styles and exploratory testing; their work can prove useful in optimizing exploratory testing based on testers learning style and experience.

Bach J. (2003) mentions that exploratory testing can very well compliment script based testing because of its ability to create and improve tests while executing them, also one of the main advantages of exploratory testing is its ability to provide an effective feedback, but also says that this very much depends upon the feedback loop, if this loop is slow or too long then exploratory testing loses its significance. In such circumstances it is advisable to fall back to script-based testing procedures, also in the conditions when testing is being performed upon the part which requires a

high degree of measurement, customer approval. Therefore a combination of both script-based testing and exploratory testing will let the project teams to get the best of both worlds.

Bach J. (2000, 2003, 2006), Lyndsay & Eeden (2003) mention session based testing as a way to compliment the benefits of exploratory testing, because session based testing makes exploratory testing auditable and measureable on a wider scale. Lyndsay & Eeden (2003) say that the session based test management method brings measure and control to an immature process by introducing documentation of test reports and test sessions, and division of work in sessions brings out ability to control the work. Also introduces ability to control the scope of testing by using test charters (which mention which requirements should be tested for a specified amount of time). Also one of the respondents from a team that follows session based test management mentioned:

> "*This testing process is quite agile, and it is easy to adapt to changes and it can be improved constantly*"

Introduction of session based test management can definitely help the teams that are following ad-hoc methods, as session based test management could introduce some structure in their testing procedures without completely altering their way of working. As it is tough to incorporate large changes in methodologies of working in short period of time, therefore session based testing becomes a good option to choose as it structures the testing process and still allows the testers to perform their testing in their own methods. Teams that are following ad-hoc methods have a good possibility of improve their procedure by introduction of combination of exploratory testing and session based test management.

Session based testing also introduces certain metrics (TBS), test execution & design (T), bug investigation & reporting (B), setup (S) (Bach J. , 2006). These metrics help in determining the effort put on test setup, design execution and bug investigation & reporting. Therefore allowing teams to keep track of resources and also helping in improve the future estimation concerned with the test planning.

However during this research there was some hardship faced to find any valid empirical data in the existing literature that could suggest that introduction of these recommended practices (exploratory testing, session based test management) is advantageous over other practices that teams are already using. A study by Itkonen & Rautiainen (2005) provides some support to the claim "with exploratory testing more defects can be found", but the data does not provides any strong support for this claim, as the study was conducted on relatively small sample and the data used was not adequate enough, despite this the study by Itkonen & Rautiainen (2005) shows that session based exploratory testing is advantageous compared to freestyle exploratory testing. As it provides some support for managing test coverage which is mentioned as the biggest shortcoming in exploratory testing. In the same article Itkonen & Rautiainen (2005) also mentions that with free style exploratory testing it is difficult to track progress of overall testing and also the progress of the tester, and it is also ambiguous to see how testing is performed, and by employing only exploratory testing the prevention of defects becomes tough (for example, script-based testing can be initiated during requirements phase and hence enabling early detection of defects).

On the other hand the qualitative data obtained in this case study suggests that both of the project teams that employ exploratory testing have shown fair amount of satisfaction in their testing practices and one of the respondents mentioned:

> "*We have a matrix to present the requirements, and we mention details such as in which session a particular requirement was tested. And which requirements are to be tested in the next session", " Before performing testing testers go through the*

requirements and communicate with developers regarding testing procedures on some specific functionalities, and also communicate with test manager or test leader to discuss any known issues with the testing ", "After tester finishes performing testing and submits the test-charter the test leader goes through the test charter report and then contacts the tester and then has a debriefing meeting and discusses it. For example things like how many requirements are fulfilled or covered in this charter? Is there anything more to test? Have we found any bugs that could affect other functionalities? After debriefing is done and then we mark the requirements that are tested in the matrix"

These responses suggest that session based exploratory testing provides some level of mapping test coverage, but still it is most of the decisions are based upon human interpretations, which could be seen as an issue of reliability at least when it comes to development of safety-critical features. This leads to the point made by Bach J. (2003) that says a combination of script-based testing and session based exploratory testing allows you to achieve best of both worlds.

Therefore this could be seen as a future prospect of research that could be conducted at VTEC. As there are project-teams at VTEC that have employed exploratory testing with session based test management and also the project-teams that majorly rely on the script-based testing, this provides a good setup to perform a comparative study on effectiveness and efficiency of testing practices. This could be achieved by help of introducing certain metrics that could help in determining the software testing process efficiency and effectiveness. The future research on this can lead to some concrete conclusions about effectiveness and efficiency of exploratory testing with session based test management.

2) Automation:

In order to address the requirement of introduction of more automation in software testing, it becomes consequently important to identify and address the challenges and barriers for automation in software testing. As mentioned in section 4.3.4 the barriers identified from the interview data were:

1) Changing requirements
2) Visual examination/human evaluation of system
3) Requirement of training/and experienced personal
4) Budget and schedule limitations

To address the issues due to changing requirements teams can identify the areas where the requirements are less likely to be subjected to change and introduce automation in those parts at least, and also try to keep these automated testing implementations as abstract as possible in order to incorporate changes in requirements. As discussed before in section 4 some teams have developed their own in-house automated tools to perform testing, this knowledge can be transferred to other teams and help them in achieving automation.

Xie (2006) identifies the main challenges associated with automated testing as:

1) Generate test input effectively
2) Running the test input
3) Verifying test executions.

These 3 challenges could be taken as main goals for designing an automated testing tool for software testing. Xie (2006) further describes that the tool should have subsystems to

1) Detect redundant test inputs/tests as these redundant tests don't contribute to effective bug detection.
2) A non-redundant test generator

3) Test selector, as it could be infeasible to inspect the output of all the tests, test-selector allows to narrow down the scope of tests
4) Test abstracter, as the concrete state-transitions of modules can be difficult to understand for developers.
5) Program-spectra comparator, to compare the outputs of the tool
6) Feedback loop to dynamically improve tools performance.

Shahamiri et al. (2009) discus about test oracle; test oracles are described as a reliable source of expected outputs; they are used to verify the test case results that are performed upon a software unit. Since test automations requires automated test execution and results verification, a good test oracle could play a key role in achieving effective software test automation. Shahamiri et al. (2009) further discuss about challenges in developing an efficient test oracle and other crucial insights on test oracle.

Since interview results depict that budget and scheduling limitations is one of the barriers for automation. Therefore even though these new concepts sound promising towards achievement of effective and efficient automated software testing; there is still a requirement to perform a careful cost-benefit analysis on these concepts.

As automated software testing itself is a vast fertile ground for active research; and due to scope and schedule limitations of this study further discussion on automation is not feasible. However this research provides the possible common barriers faced by project teams to implement automation in industrial environment which could be considered as suggestive towards the future research work and could be also helpful in making design decisions regarding automated testing.

3) Common base testing process:

Bertolino (2007) describes universal test theory as one of the long lasting dreams of software testing, a good amount of research efforts are being put in this direction in order to develop a common universal test theory, such that testers can refer to one coherent and rigorous framework and perform testing and understand the relative strengths and limitations of existing practices and select the one that suits their needs. As discussed before in section 4.2.10; there was a trend observed in interview response which pointed towards this requirement for establishment of a common base process for software testing. The issues mentioned by the interview respondents were: 1) current methods depend a lot on tester's experience, 2) lack of a concrete defined process for testing (not in all cases) 3) lot of effort is spent to transfer knowledge to new tester in team. Therefore introduction of a common base process will not only address the issues mentioned by interview respondents but also help in building a body of knowledge on software testing for the organization. Such that it can provide a mapping of the most effective combinations of methods, tools and practices associated with a certain well defined testing statement or goal, and also provide explanations of any underlying assumptions if involved.

Bertolino (2007) Mentions that in order to establish a good base testing theory, there is a need to analyse effectiveness of different testing methods and tools. Studies such as the ones conducted by Shahamiri et al. (2009), Kevrekidis et al. (2009), Basilli & Selby (1987), Itkonen & Rautiainen (2005), Frankl & Hamlet (1998) provide good comparative insights of various testing practices which can prove useful while making trade-off decisions and establishing good base testing practices. Bertolino (2007) mentions that while defining a base test theory; combination of different methods and tools should be used otherwise it could lead to *saturation effect* (Lyu, 1996).

Gaudel (2005), Bertolino (2007) and Bernat, Gaundel, & Merre (2007) discuss how formal specifications can be used to define some testing strategies, also called as *test hypotheses*. Gaudel (2005) describes some general schemes to make these test hypotheses. Test hypotheses provide a "black box" type testing strategy which can be unique for a finite set of test set, depending upon the scope of hypotheses. If the definition of hypothesis is strict then it is feasible to form an "exhaustive" test set. Therefore theoretically it is feasible to create test criteria for "exhaustive" testing, such that after completion of test run it can be concluded that the software module under testing complies with a criteria based on a certain hypothesis. Bertolino (2007) Mentions this could help in creating fault coverage models. These schemes can be used to create fault coverage models in order to address ISO 26262 requirements. Test hypotheses can be made based on software testing requirement of ISO 26262 and a set of these hypotheses could lead to creation of a fault coverage model and similar analogy could be used to approach other standards. On the other hand creation of test hypothesis depends a lot on *formal specifications*, and the results mentioned in section 4.2.8 depict that usage of formal methods and specifications is not popular at VTEC; therefore by keeping all the benefits of formal methods in mind it is recommended to introduce the usage of formal methods.

4) Incremental Testing:

Galin (2004) mentions that unless the program is very small and simple, applications of "big bang" testing approach can lead to severe problems, also to identify errors in big software modules is relatively difficult to perform. If "big-bang" testing is performed on a relatively large software module, it will require more resources and on the other hand effectiveness of this strategy is uncertain. Also if "big bang" testing approach is applied to a large number of software modules, the estimation of effort and other resources of testing will be relatively fuzzier (Galin, 2004). In section 4.2.10 it was discussed that the interview respondents mentioned the need for more incremental testing approach. On being asked "why big bang testing approach was adopted?" the respondents mentioned that

> "*It was because of historical reasons, the project was relatively small in size when this approach was adopted, but now it has grown much bigger in size and hence it is becoming a cumbersome task*".

Therefore it is recommended for the project teams to adopt best testing practices during the initial phases of project; otherwise it could be much more difficult to introduce any major change in latter phases.

On the other hand incremental testing is usually subjected on a relatively small software module at one instance, therefore it becomes relatively easier to achieve a higher rate of error detection and the root-cause identification is much simpler compared with testing a relatively large size of software in "big-bang" fashion and also it requires less resources. With incremental testing a higher percentage of errors can be detected and fixed at earlier stages, which helps in screening the errors from migrating to the later much complex stages.

Besides these advantages incremental testing also has limitations such as 1) it requires preparation of stubs and drivers in order to perform unit and integration testing, also 2) integration testing requires many operations to be performed over same software sub unit. However after considering the requirements for software testing mentioned in ISO 26262, the latter disadvantage can be considered as a beneficial factor, and besides that ISO 26262 requirements to a higher degree make adoption of incremental testing a compulsion(at least to the projects to which it is applicable).

5) Test Driven Development:

Respondents from 3 project teams mentioned the usage of test-driven development approach and at least one person from these teams mentioned that this as a positive feature of their testing approach. Furthermore Maximilien & Williams (2003), Artem, Abrahamsson, & Ihme (2009) provide empirical case studies conducted at IBM and Nokia, which support the statement that test-driven development leads to higher quality. Therefore it is consequently evident that employment of test-driven development would induce positive effects on software development. Adoption of test-driven development could also prove helpful in development of software code based on the requirements of ISO 26262.

6) Dedicated Testers:

As mentioned before in section 4.2.10 that there was a pattern observed in the interview responses; which pointed towards requirement of dedicated software testing team. Considering that ISO 26262 will be a mandatory standard for projects associated with passenger car sub systems, it is consequential that testing activities has to be tweaked in order to meet ISO 26262 requirements. Teams that have external testing team should effectively communicate the requirements of ISO 26262 to their external testing team.

7) Usage of more advanced software testing models:

From the data in section 4.1 it can be derived that at least 50% of the projects are based on embedded systems. Due to special requirements of embedded systems such as: 1) inconsistency with the run-time environment, 2) hard-ware dependency 3) strict deadlines, embedded systems are required to be tested on target environment even if the software passes the testing campaign on host environment. This makes software testing more expensive for embedded systems. Therefore there is a requirement to adopt software testing models that are more explicit to embedded systems.
Zheng & Qian (2009) Mentions that the problems with V-model are that 1) the issues with requirement analysis phase are detected in acceptance testing phase 2) and the strict deadlines for embedded systems demand good reliable performance; which in turn is dependent upon system architecture, and flaws in architecture are detected in system testing phase.
Zheng & Qian (2009) introduce a comprehensive software testing model for embedded systems domain that emphasizes on introduce software testing as early as in requirements phase; this can help in attaining further refinement over requirements (which is mentioned as one of the issues by interview respondents). Also problems with performance can be detected in software design phase. This model also helps in making effective decisions for distributing testing activities over host and target environments and hence indirectly inducing better decisions regarding expenses of embedded software testing.

7. Conclusion

This thesis presents the findings and analyses from the case study conducted at Volvo technology (VTEC), interviews were conducted with 14 persons belonging to 8 different project teams at VTEC. As this case study was conducted in one organization the results and analysis cannot be generalised to complete automobile domain, but however the results and discussion can be suggestive towards the general practices regarding software testing in automobile industries, and these findings also reflect some trends in software testing in general.

This research also discusses some aspects of software testing that could be improved by inclusion of certain practice. This research study provides data that supports some existing beliefs in software

testing, also presents some notable findings which could prove helpful in shaping future research work in software testing. As the research was more qualitative in nature; it not only indicates the popularity or extent of usage of various testing practices but also reflects the reasons of the observed trends.
During this research one of the issues that were faced was lack of sufficient literature on ISO 26262 relevant to software engineering; hence as future work it is suggested to conducted studies to investigate the software engineering aspects of ISO 26262, which could be helpful in designing software testing process models to effectively cover requirements of ISO 26262.

This study also points towards the need of conducting further research to see the validity of advantages exploratory testing with support of empirical data. Also there is a need to perform similar studies on a wider scale in order to establish a concrete understanding of software testing practices in automobile domain which will be helpful in creating synergism among automotive industries.

Bibliography

1. Artem, M., Abrahamsson, P., & Ihme, T. (2009). Long-Term Effects of Test-Driven Development A case study. In: *Agile Processes in Software Engineering and Extreme Programming,10th International Conference, XP 2009,. 31*, pp. 13-22. Pula, Sardinia, Italy: Springer.
2. Bach, J. (2000, November). Session based test management. *Software testing and quality engineering magzine(11/2000)*,(http://www.satisfice.com/articles/sbtm.pdf).
3. Bach, J. (2003). Exploratory Testing Explained, The Test Practitioner 2002, (http://www.satisfice.com/articles/et-article.pdf).
4. Bach, J. (2006). *How to manage and measure exploratory testing.* Quardev Inc., (http://www.quardev.com/content/whitepapers/how_measure_exploratory_testing.pdf).
5. Basilli, V., & Selby, R. (1987). Comparing the effectiveness of software testing strategies. *IEEE Trans. Software Eng., 13*(12), 1278-1296.
6. Berg, B. L. (2009). *Qualitative Research Methods for the Social Sciences (7th International Edition)* (7th ed.). Boston: Pearson Education.
7. Bernat, G., Gaundel, M. C., & Merre, B. (2007). Software testing based on formal specifications: a theory and tool. In:*Testing Techniques in Software Engineering, Second Pernambuco Summer School on Software Engineering. 6153*, pp. 215-242. Recife: Springer.
8. Bertolino, A. (2007). Software Testing Research: Achievements Challenges Dreams. In: *International Conference on Software Engineering, ISCE 2007,* (pp. 85-103). Minneapolis: IEEE.
9. Causevic, A., Sundmark, D., & Punnekkat, S. (2010). An Industrial Survey on Contemporary Aspects of Software Testing. In: *Third International Conference on Software Testing, Verification and Validation* (pp. 393-401). Paris: IEEE Computer Society.
10. Chillarege, R. (1999). *Software Testing Best Practices.* Tehcnical Report RC2145, IBM.
11. Frankl, P. G., & Hamlet, R. G. (1998). Evaluating Testing Methods by Delivered Reliability. *IEEE Trans. Software Eng., 24*(8), pp. 586-601.
12. Galin, D. (2004). *Software Quality Assurance: From theory to implementation.* Pearson Education Ltd.

13. Gaudel, M. C. (2005). Formal Methods and testing: Hypothesis, and correctness Approximations. In: *FM 2005: Formal Methods, International Symposium of Formal. 3582*, pp. 2-8. Newcastle: Springer.
14. IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology. IEEE Std.
15. ISO/IEC/IEEE-24765. (2010). International Standard - Systems and software engineering-Vocabulary. ISO, IEC, IEEE.
16. ISO26262. (2009). ISO/DIS 26262 (2009) Road vehicles – Functional safety –Part 1-10, Standard under development.
17. Itkonen, J., & Rautiainen, K. (2005). Exploratory Testing: A multiple Case Study. In: *International Symposium on Empirical Software Engineering* (pp. 84-93). IEEE.
18. Kevrekidis, K., Albers, S., Sommemans, J. P., & Stallmon, M. G. (2009). Software Complexity and Testing Effectiveness: An Empirical Study. In: *Reliability and Maintainability Symposium, 2009. RAMS 2009. Annual* (pp. 539-543). Fort Worth, TX: IEEE.
19. Krammer, M., Marko, N., Armengaud, E., Geyer, D., & Griessnig, G. (2010 , September). Improving methods and processes for the development of safety-critical automotive embedded systems. In: *Proceedings of 15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2010* (pp. 1-4,13-16). Bilbao, Spain: IEEE.
20. Lemmer, K., Köhler, S., Köster, F., Jost, H., Hahn, A., Häusler, S., et al. (2010, October). Supporting qualification: Safety standard compliant process planning and monitoring. In: *IEEE Symposium on Product Compliance Engineering)ISPCE 2010* (pp. 1-6, 18-20). IEEE.
21. Lyndsay, J., & Eeden, N. V. (2003). *Adventures in session based testing , Technical Report* . Workroom Productions Ltd. (http://www.workroom-productions.com/papers/AiSBTv1.2.pdf)
22. Lyu, M. (1996). *Handbook of software reliability engineering* (Vol. 7). New York: McGraw-Hill.
23. Matheis, J., K.D., M.-G., M., H., M., H., & N., A. (2010). Failure mode and effect analysis based on electric and electronic architectures of Vehicles to support the safety lifecycle ISO/DIS 26262. In: *21st IEEE International Symposium on Rapid System Prototyping (RSP), 2010* (pp. 1-7). Fairfax, VA: IEEE.
24. Maximilien, E. M., & Williams, L. (2003). Assessing Test-Driven Development at IBM. *25th International Conference on Software Engineering, 2003. Proceedings.* (pp. 564 - 569). IEEE.
25. Perry, W. (1995). *Effective Methods for Software Testing.* New York: John Wiley & Sons.
26. Reed, K., Ng, S., Murnane, T., Grant, D., & Chen, T. (2004). Preliminary survey on software testing practices in Australia. In: *Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'04)* (pp. 116-127). Melbourne, Australia: IEEE Computer Society.
27. Rivett, R. S. (2009, December 03). Hazard Identifications and classifications: ISO 26262-The application of IEC 61508 to automotive sector. In: *5th IET Seminar on SIL Determination, 2009* (pp. 1-24). London: IEEE.
28. Schwarz, J., & Buechl, J. (2009). *preparing the future for functional safety of automotive E/E systems,*. Enhanced Safety of Vehicles, Paper number 09-0445, US Department of Transportation.
29. Shahamiri, S. R., Wan Mohd Nasir, W. K., Mohd-Hashim, S. Z. (2009). A Comparative Study on Automated Software Test Oracle Methods. In: *Fourth International Conference on Software Engineering Advances, ICSEA '09* (pp. 140 - 145). IEEE.

30. Steinar, K. (1996). *Interviews: an Introduction to qualitative research Interviewing.* Sage publications, Inc.
31. Strauss, A., & Corbin, J. (1998). *Basics of Qualitative Research: Techniques and procedure for developing Grounded Theory.* Sage, Thousand Oaks.
32. Tinkham, A., & Kanner, C. (2003). Learning Styles and Exploratory Testing. In: *Pacific Northwest Software Quality Conference (PNSQC) 2003.*
33. Walsham, G. (1995). Interpretive case studies in IS research: nature and method. *,Vol 4*, (pp. 74–81), *European Journal of Information Systems*.
34. Webster, J., & Watson, R. (2002). Analyzing the past to prepare for the future: Writing a literature review, *26*(2), (pp. 13-23), *MIS Quarterly*.
35. Xie, T. (2006). Improving effectiveness of automated software testing in absence of specifications. In: *22nd IEEE International Conference on Software Maintenance. 22*,(pp. 355-359). Philadelphia, Pennsylvania, USA: IEEE Computer Society.
36. Zheng, C., & Qian, H.-m. (2009). A Embedded Software Testing Process Model. In: *International Conference on Computational Intelligence and Software Engineering, 2009. CiSE 2009.* (pp. 1-5). Wuhan: IEEE.

Appendices

A. ISO 26262 Software testing requirements

This section presents the requirements for software testing activities, and also to what level these activities are required on a particular ASIL level.

Unit Testing:

Methods for unit testing (a):

1. Walk through
2. Inspection
3. Semi-formal verification
4. Formal verification
5. Control analysis
6. Data analysis
7. Static analysis
8. Semantic analysis

Walk through is highly recommended for ASIL level A and recommended for ASIL b and not recommended for ASIL C, D.
Inspection is recommended at all levels and highly recommended for ASIL B, C, and D.
Control/Data/Static/Semantic analyses are recommended for all ASIL levels, and for ASIL B, C and D it is highly recommended equally for all ASIL levels, except for semantic analysis which is equally recommended for all ASIL levels.
Semi-formal verification is recommended for all ASIL levels and highly recommended for ASIL C & D.
Formal verification is recommended only for C and D ASIL levels.

Methods for software unit testing (b):

1. Requirements-based tests
2. Interface test
3. Fault injection test
4. Model-code comparisons test

Requirements-based tests and *interface tests* are highly recommended for all the ASIL levels.
Fault injection test and *resource usage test* is recommended for all ASIL levels and highly recommended for ASIL D.
Model-code comparisons tests if applicable are recommended at all ASIL levels and highly recommended for ASIL C, D.

Test case deriving methods for unit testing:

1. Analysis of requirements
2. Generation and analysis of equivalence classes
3. Analysis of boundary values (applicable only for interfaces)
4. Error guessing

Analysis of requirements is highly recommended method at all ASIL levels
Generation and analysis of equivalence classes and *Analysis of boundary values* methods are recommended for ASIL levels and highly recommended for ASIL B, C & D.

Error guessing is not highly recommended for any ASIL level; however it is equally applicable at all ASIL levels.

Structural coverage metrics at software unit level:

1. Statement coverage
2. Branch coverage
3. Modified condition / Decision coverage

State coverage metrics are recommended at all ASIL levels and highly recommended for ASIL A, B.
Branch coverage metrics are recommended for ASIL B, C & D.
Modified condition/Decision coverage metrics are recommended for all levels and highly recommended for ASIL D.

Pre-requisites for unit testing:

- Hardware-software interface specification
- Software verification plan
- Safety plan
- Software unit design specification
- Software unit implementation
- Software verification report
- Tool application guidelines
- Guidelines for the application of methods
- Software tool qualification report

Integration Testing:

Methods for software Integration testing:

1. Requirements-based tests
2. Interface test
3. Fault injection test
4. Model-code comparisons test

Requirements-based tests and *interface tests* are highly recommended for all the ASIL levels.
Fault injection test and *model-code comparisons tests* is recommended for all ASIL levels and highly recommended for ASIL C, D.
Resource usage test if applicable are recommended at all ASIL levels and highly recommended for ASIL D.

Test case deriving methods for unit testing:

1. Analysis of requirements
2. Generation and analysis of equivalence classes
3. Analysis of boundary values (applicable only for interfaces)
4. Error guessing

Analysis of requirements is highly recommended method at all ASIL levels
Generation and analysis of equivalence classes and *Analysis of boundary values* methods are recommended for ASIL levels and highly recommended for ASIL B, C & D.
Error guessing is not highly recommended for any ASIL level; however it is equally applicable at all ASIL levels.

Structural coverage metrics at software unit level:

1. Functional coverage
2. Call coverage

Functional and *call coverage metrics* are mentioned as recommended methods to measure structural coverage of software at architectural level and are highly recommended for ASIL C and D.

Prerequisites for integration testing: (the prerequisites mentioned below should be conforming to ISO 26262 guidelines)

- Hardware-software interface specification
- Software architectural design specification
- Safety plan
- Software unit implementation
- Software verification plan
- Software verification specification
- Software verification report
- Tool application guidelines
- Qualified software components
- Guidelines for the application of methods
- Software tool qualification report

Test Environment:

1. Hardware in loop
2. ECU network environments
3. Vehicles

Hardware in loop environment testing is recommended are all ASIL levels and highly recommended for ASIL C and D.

Testing on *ECU network environments* and the *vehicles* is highly recommended for all ASIL levels.

B. Interview Questionnaire.

Abbreviations used: *open questions (O), probing questions (P), linking questions (L), guiding questions (G), Hypothetical questions (H), forcing questions (F), and closed questions (C)*

Main Theme: Software Testing Practices in Automotive Domain
Sub-Themes:

1) **Software Testing Methodologies & Techniques:** Methodology, Stopping Criteria, Barriers, Benefits, expected areas Improvements, generation of test cases, formal methods, Software Testing Metrics.
   - i) Can you tell us about the project you are working on? (Probing questions on: domain, functionality, scope, etc.) (O)
   - ii) What are the pre – requisites for the type of testing you perform, what are the things you require to start testing (Design, Specifications, Plans, guidelines, etc.)? (O)
   - iii) Can you tell us about the purpose of testing, for example if you are performing testing on a software component/unit, what are characteristics of that component you check by performing testing? (H)
   - iv) Which software testing method are you using to check whether the software component satisfies the criteria's that you mentioned in earlier question? (L)
   - v) Reason behind the selection of this/these methodology/ies (Probing questions)
   - vi) How good is the support for the methodology you have adopted (In terms of available tools, Guidelines, etc)? (F)
   - vii) Do you receive any guidelines for application of testing methods that you use (Or do you require them)? (F)
   - viii) What do you think can be further improved in the methodology/ies you are using? (O)
   - ix) Do you think the specificity of your domain would requires more specialized testing/verification techniques, methodologies and Tools ? (C)
   - x) How do you plan/generate test cases (SWTD)? (L)
   - xi) What are the requirements for the test cases that you create? (L)
   - xii) Do you use any metrics to indicate the completion of test cases? (G) If yes, which one? (P)
   - xiii) Are you using any other metrics for other purposes? If yes, can you tell us about them?
   - xiv) Can you tell us about the testing environment that you use? (O)
   - xv) How do you compare the test environment with the target environment? How closely are you able to emulate the target/customer environment during testing? (L)
   - xvi) What are the outcomes of the testing that you perform, what documents do you produce as test results? (SWTR)(F)
   - xvii) Can you tell us about the test planning activities? (O)
   - xviii) Do you prepare plan (SWTP) for the tests in advance, Preparing test plan before performing testing? (C)
   - xix) Do you employ formal methods (Explain, if required) while testing some specific components? (C) If yes, which methods? (P) And can you also tell us about the nature of those software components that are subjected to formal methods?(P)

2) **Software Testing Tools:** Tools used, Purpose of the Tool, Issues, Automated Tools, level of automation expected.

i) Which testing tools are you using to support the testing methodologies you are working with?(G)
ii) Do you receive any guidelines for usage of the tools that you use (Or do you require them)?(F)
iii) What are the activities that you are able to perform with these tools (probing questions to determine tools used for: Test case Generation, Test planning and Management, code coverage analysis, and check the other requirements for ISO 26262)?
iv) Are you satisfied with the effectiveness/efficiency of the tools that you are using? (O)
v) Do you expect any improvement in these tools? If yes, please mention (O).

3) **Software Testing Standards:** Standards being employed, if any in-house standards are employed, barriers in adopting the new standards.
i) Are you following any explicit standards for your testing activities? (G)
ii) Was the transition to new standard smooth? (P)
iii) Have you performed any modification to the standard to suit your needs? If yes, please mention? (L)
iv) Are there any barriers that you have faced while adopting this standard? (C) If yes, please mention ?(P)

4) **MISC:**
I) Can you suggest us any questions regarding software testing, that you think we have missed, or can you give us suggestions to improve this interview questionnaire? (O)

C. Glossary

1 Ad-Hoc testing:

Testing carried out using no recognized test case design technique. Ad hoc testing is usually performed by component developers and is not documented. The tests do not have to be repeatable. (BCS)

2 Big bang testing:

To test the software in its entirety, once the completed package is available (Galin, 2004), Pg. 182

3 Black box (functionality) testing:

Testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions. (IEEE, 1990)

4 Code reviews:

Code review is systematic examination (often as peer review) of computer source code. It is intended to find and fix mistakes overlooked in the initial development phase, improving both the overall quality of software and the developers' skills. Reviews are done in various forms such as pair programming, informal walkthroughs, and formal inspections. (Kolawa & Huizinga 2007)

5 Coverage Analysis:

Is a one of the white box testing methods, coverage analysis is used to test the structure of code. Functional coverage, statement coverage, decision coverage, condition coverage are the sub-criteria involved in coverage analysis.

6 Control/ Data flow analysis:

A white box testing method which is used to perform analysis on the sequence in which operations are performed during the execution of a computer program (ISO/IEC/IEEE-24765, 2010)

7 DIS: Draft International Standard

8 Exploratory testing:

Exploratory testing is any testing to the extent that the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests. (Bach J. , 2003)

9 Formal Specification:

1. A specification that is used to prove mathematically the validity of an implementation or to derive mathematically the implementation (ISO/IEC 2382-20:1990) Information technology — Vocabulary — Part 20: System development.20.01.04. 2. A specification written in a formal notation, often for use in proof of correctness. 3. A specification written and approved in accordance with established standards

10 Hardware –in-loop test:

Hardware-in-the-loop (HIL) simulation is a technique that is used in the development and test of complex real-time embedded systems (Wiki).

11 Incremental testing:

To test the software piecemeal, in modules, as they are completed (unit tests); then to test groups of tested modules integrated with newly completed modules (integration tests). This process continues until all the package modules have been tested. Once this phase is completed, the entire package is tested as a whole (system test). This testing strategy is usually termed "incremental testing".

12 Inspection
1. A visual examination of a software product to detect and identify software anomalies, including errors and deviations from standards and specifications. IEEE Std 1028-2008 IEEE Standard for Software Reviews and Audits.3.3. 2. A static analysis technique that relies on visual examination of development products to detect errors, violations of development standards, and other problems 3. [Technique] Examining or measuring to verify whether an activity, component, product, result, or service conforms to specified requirements. A Guide to the Project Management Body of Knowledge (PMBOK® Guide) — Fourth Edition

13 Interface test:

Testing conducted to evaluate whether systems or components pass data and control correctly to one another (ISO/IEC/IEEE-24765, 2010).

14 Iterative testing:

Testing done in projects associated with agile based development paradigm.

15 Regression testing:

Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements 2. testing required to determine that a change to a system component has not adversely affected functionality, reliability or performance and has not introduced additional defects. ISO/IEC 90003:2004, Software engineering— Guidelines for the application of ISO 9001:2000 to computer software.3.11. 3. functional testing that follows modification and maintenance

16 Script based testing:

A testing scenario which depends on pre-defined test scripts that execute a certain set of pre-defined test cases.

17 Session based test management:

Session-based testing is a technique for managing and controlling unscripted tests. It is not a test generation strategy, and while it sets a framework around unscripted testing, it is not a systematic approach whose goal is precise control and scope. Rather, it is a technique that builds on the strengths of unscripted testing - speed, flexibility and range - and by allowing it to be controlled, enables it to become a powerful part of an overall test strategy. (Lyndsay & Eeden, 2003)

18 System testing:

Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. IEEE Std 829-2008 IEEE Standard for Software and System Test Documentation.3.1.37

19 Software design description:

A representation of software created to facilitate analysis, planning, implementation, and decision-making. IEEE Std 1012-2004 IEEE Standard for Software Verification and Validation.3.1.28; IEEE Std 1016-1998 IEEE Recommended Practice for Software Design Descriptions.3.4

20 Static analysis:

The process of evaluating a system or component based on its form, structure, content, or documentation. (ISO/IEC/IEEE-24765, 2010)

21 Test cases/vectors:

1. A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. IEEE Std 1012-2004 IEEE Standard for Software Verification and Validation.3.1.31. 2. Documentation specifying inputs, predicted results, and a set of execution conditions for a test item. IEEE Std 1012-2004 IEEE Standard for Software Verification and Validation.3.1.31

22 Test description document:

A document specifying the details of the test approach for a software feature or combination of software features and identifying the associated tests (ISO/IEC/IEEE-24765, 2010)

23 Test report:

1. A document that describes the conduct and results of the testing carried out for a system or component. Syn: test summary report. cf. test case specification, test incident report, test item transmittal report, test log, test plan, test procedure (ISO/IEC/IEEE-24765, 2010)

24 Test/verification plan:

A document describing the scope, approach, resources, and schedule of intended test activities. IEEE Std 1012-2004 IEEE Standard for Software Verification and Validation.3.1.33. 2. a document that describes the technical and management approach to be followed for testing a system or component. IEEE Std 1012-2004 IEEE Standard for Software Verification and Validation.3.1.33. 3. a plan that establishes detailed requirements, criteria, general methodology, responsibilities, and general planning for test and evaluation of a system. ISO/IEC 2382-20:1990, Information technology — Vocabulary — Part 20: System development.20.06.09

25 Test Environment:

Hardware and software configuration necessary to conduct the test case, ISO/IEC 25051:2006, Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Requirements for quality of Commercial Off-The-Shelf (COTS) software product and instructions for testing.4.8

26 Testing frameworks:

A software tool that accepts as input source code, test criteria, specifications, or data structure definitions; uses these inputs to generate test input data; and, sometimes, determines expected results. (ISO/IEC/IEEE-24765, 2010)

27 White box (structural) testing:

Testing that takes into account the internal mechanism of a system or component.

28 Walk-through:
a static analysis technique in which a designer or programmer leads members of the development team and other interested parties through a segment of documentation or code, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems (ISO/IEC/IEEE-24765, 2010).