UNIVERSITY OF GOTHENBURG

Introducing Requirement Stability Metrics for Test Case Success Prediction in RUAG Space AB

FARNAZ TAHERI
NGUYEN AN DUONG

Master of Software Engineering and Management Thesis

Report No. 2010:063
ISSN: 1651-4769

University of Gothenburg
Department of Applied Information Technology
Gothenburg, Sweden, May 2010

Introducing Requirement Stability Metrics for Test Case Success Prediction in RUAG Space AB

Nguyen An Duong
Department of Applied IT
Chalmers| University of Gothenburg
Gothenburg, Sweden
Gusnguyan@ituniv.se

Farnaz Taheri
Department of Applied IT
Chalmers| University of Gothenburg
Gothenburg, Sweden
FTaheri@student.gu.se

ABSTRACT

Context: In every software development method, requirement gathering and analysis phase plays the most important role. Stability of requirements potentially makes an impact on the success of later phases in a software project, including the success of test cases. Nevertheless, this impact is not well investigated in either theory or industry. Furthermore, the application of software metrics can improve the quality of software and efficiency of software development processes since metrics can help in controlling and making predictions in software development projects.

Objective: In this thesis, we intend to introduce requirements stability metrics for test case success prediction in the context of integration and verification unit of RUAG Space AB, Sweden.

Method: The research is done by conducting a case study that includes reviewing the related work, defining a set of requirement stability metrics, developing an automated tool for the data collection on a daily basis, and performing empirical evaluations on validity and usefulness of the introduced metrics in an industrial context.

Results: The research outcomes present that the proposed requirement stability metrics can be useful for stakeholders after making minor changes in their definitions and the metrics can be applied to integration and verification processes in RUAG Space AB.

Conclusions: However, more time is required for data collection to expand the thesis work and to conclude whether the proposed metrics can be used as predictors for test cases successes in RUAG Space AB and other companies. The remaining work can be pursued in future research work.

Keywords

Requirement Stability, Metrics, Test Cases, Success, Prediction, Measurement Systems, Integration, Verification.

1. INTRODUCTION

Requirements play a driving role during the product creation. The requirements are captured in the beginning of the project to conclude what exactly needs to be developed. According to Brooks [1], the toughest part in building a software system is to decide precisely what needs to be developed. Furthermore, the poor requirement gathering and analysis may affect negatively at a later stage [2, 3]. Moreover, predicting potential results of the later phases from early time of software development can obviously help the project team to better deal with the risks of project rescheduling and resulting in a low-quality product [4, 5, 6]. One of the noticeable aspects of requirement gathering and analysis phase is measuring the stability of requirements. The word 'stability' is defined as 'when something is not likely to move or change' [7], and in the case of requirements it is easier to define the in-stability of them. Hence, requirements instability can be understood as 'how often the requirements change'. The problem of measuring requirements stability has been research from several theoretical angles, but little has been done in terms of empirical validation of requirements stability metrics. Requirement stability has also been studied in other ways of approach such as requirement volatility [4] and requirement instability [5].

1.1 Problem Statement

The overall goal of this thesis project was to improve the quality of integration and verification process in RUAG Space AB. Such a need was raised by the stakeholders in integration and verification (I&V) department to have a better control over the test processes. In particular, they wanted to know in advance whether their test cases will be successful when executed during the integration stage of their processes. Our role was to evaluate whether the requirement stability metrics can help as potential predictors in anticipating the success of test cases.

In general, three research questions were addressed in our work:

- *Which requirements stability metrics are important for RUAG Space AB?*
- *How to integrate these metrics within the integration and verification process in RUAG Space AB?*
- *Can requirements stability metrics be used as a predictor for test case execution success?*

To address these questions an initial set of requirements metrics was proposed and implemented. After that, an automated tool was developed to collect metrics' values on a daily basis and evaluate them on a weekly basis with stakeholders at the company. The evaluation was done through interviews with stakeholders who are responsible for the integration and verification processes, including requirement engineers, verification engineers, and managers. The goal of the evaluation was to find the answers to the first and the last research questions.

This thesis is organized into 6 chapters. Chapter 2 discusses related work to the thesis topic. In chapter 3, we describe the industrial context where the thesis research was conducted. Chapter 4 presents the design steps of the conducted case study in

the company. Analysis and research results are thereafter discussed in chapter 5. Finally, the conclusions are presented in chapter 6.

2. RELATED WORK

In this chapter, we discuss the related work of the thesis. We have presented a list of existing requirement metrics which are potentially suitable for RUAG Space AB. We have also given a short review of the similar research which is related to requirement stability metrics and software quality.

In the research field of software measurement, there are a number of contributions for requirement metrics [8, 9, 10, 11]. However, the requirement metrics found in these contributions do not have a precisely described measurement method [12], which makes it hard to reuse these metrics or even implement them in the first place. Various studies in requirement engineering and software measurement have been undertaken. As a result, we gathered a list of requirement metrics in Appendix A. In the list, requirements metrics are detailed with metric name, description, purpose, and measurement method. Due to the lack of detailed procedures for collecting these metrics, we decided to investigate and implement a set of metrics which are used in another company in the region, but are not yet published.

Javed, Maqsood, and Durrani [13] in their research on the impact of requirements instability have proposed a metric named 'defects versus requirement changes' to examine the impact of requirements change on software defect throughout the software development life cycle and find out the root cause of defects. The metric is calculated per software release, and categorized into pre-released and post-released ones. Requirement changes and defects caused by them are counted using the project documents, such as function specifications, change requests, project schedule, etc. The metric is validated using a case study which is performed in numerous software projects in e-commerce field. Despite the fact that research carried out some interesting conclusions, it did not exactly define what should be considered as a requirement change. In addition, counting the requirement changes in project documents seems have to be done in a manual process, which may lead to inaccuracy.

In another approach of using metrics for identifying the requirement risks, Wyaat et al. [14] in NASA has introduced a set of measures on content of requirement documents and individual requirement specifications. The introduced metrics are: imperatives, incompleteness, option, weak phrases, continuances, directives, and lines of text. The authors focused on assessment of requirement document structure and quality of requirement specifications using a language based approach. Word counting is the adopted method to calculate the metrics value. The advantage of this approach is that the metrics calculation can be automated. However, the drawback is that the metrics are language dependent and in many situations, it would be difficult to exactly assess the semantics of requirement specifications.

Ambriola and Gervasi [15] introduce two metrics for requirement measurement: stability and efficiency. They suggest that requirements are developed into two steps: writing and polishing. Requirement stability is defined as variation of information volume contained in requirement specifications over time. Information volume is measured and transformed using Fourier Transformation [16], from which peaks and frequencies are observed in order to classify the requirement stability into different classes. This metric may show how smoothly the requirements are developed. On the other hand, requirement efficiency is used to measure the efficiency of requirement analysis process and estimate efficiency of further iterations in similar conditions. Therefore, these two metrics are helpful for the assessment of requirement analysis process rather than for predicting the risks in the later phases.

Loconsole [17] has used GQM model [18] in her research on measuring requirements management to find out a list of requirement metrics according to particular need. The introduced metrics target the requirements changes, which may be applicable in our thesis context.

Lam, Loomes, Shankararaman [19] has also proposed a set of metrics for managing the requirement change and action planning. The metrics are about variances on time, budget, as well as the quality before and after the requirement changes. This set of requirement metrics is probably useful for project management and planning.

3. INDUSTRIAL CONTEXT

In this chapter, we briefly describe the industrial context of this thesis work. The work was conducted in RUAG Space AB Sweden which is a leading software and hardware vendor of ESA (European Space Agency). We investigated the integration and verification processes of a sample project at RUAG Space AB, which aims to send an explorer robot to Mars, to perform the case study. The following chapters present the state of applying software metrics in RUAG Space, the integration and verification processes and the supporting tools.

3.1 Applying the Requirement Stability Metrics

In RUAG Space AB, there were no requirement metrics collected during the integration work. There is no standard set of metrics since the stakeholders are not really expected to collect any metrics at the integration stage.

Metrics of source code are calculated during the code inspection activities and that kind of information is needed mostly for process improvement rather than for the use in the current project. For an instance, the time spent on redesign can be a useful measurement as a lesson for other projects rather than to be used in the current project. The company also collects certain cost related metrics. However this is done after the completion of the integration stage.

It was important for the stakeholders in RUAG Space AB to have a better control over the integration process. In particular, they wanted to know earlier if their test cases will be successful and whether the requirements stability metrics can be helpful to predict it. The information they had about the requirement stability is what they think rather than relying on the exact numbers. To apply a better approach, their requirement was to use the requirement stability metrics for the integration and verification processes to find out a potential prediction model for success of the test cases.

The requirement stability metrics should have a little impact on the way of working, especially when it comes to the integration. The integration is a fairly new way of working for stakeholders,

and they are still trying it out. They are not ready for the detailed measurements since the process is not yet fully understood. During the integration, activities are split into smaller steps that take short time and software engineers quickly switch between tasks. Adding complicated measurements may mean adding time-consuming tasks to each of these activities and significantly increasing the time spent on them. This in turn may add too much to the cost of integration that should be avoided when integrating the requirement stability metrics within the integration and verification processes.

3.2 Integration and Verification (I&V) Process

In this section, the current project is taken as an example to describe integration and verification processes in RUAG Space AB.

At the beginning of the project, a group of project members created the anatomy of the product. The anatomy consists of several modules that are the building blocks of the product. Then the development, integration and verification orders are chosen for the product. Integration is a framework in which development and verification activities are carried out, the structure and the order for the integration tasks are set. The project includes the requirements and design activities in integration steps which define what should be ready and when to synchronize between different parts of the project. After integration, the validation activities take place. It should not start before the requirements get frozen and remain stable in the project. Changes may appear, but the requirements should not change much.

There is an integration plan that divides the project into the different modules. Implementation of modules is done according to the integration plan. While different modules have different integration times, each module has three integration milestones which are named T0, T1, and T2. The time between these integration milestones varies depending on how big are the modules to be integrated. At T0, all the requirements should be specified and frozen. It means that changes in requirements before the T0 are acceptable but when the T0 is passed, changes are undesired. When the project is between T0 and T1, changes are unwanted because at this stage the developer team begins with coding the software and the test team starts the implementation of test cases according to the requirement specifications.

At T2, hardware integration is initiated. All software and hardware tests and implementations should be completed before T2. At T2, everything needs to be ready since the integration is done at this point. Normally, the interval between each integration step is a couple of weeks and it varies depending on how big the modules are.

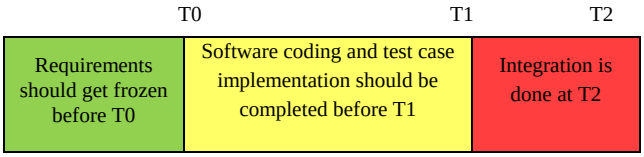

Figure 1. Integration step milestones

Although the people in integration and verification unit of RUAG Space AB have a lot of expertise in development, test, and verification, but they are quite new in the process of integrating modules. The integration processes in the project are quite new and not yet completely adapted into the system.

3.3 Supporting Tools

In RUAG Space AB, Telelogic DOORS software is used for management of requirements. The requirements are organized into modules, which can be used to categorize the requirements into different groups. There are links among requirement modules which allow forming the requirement hierarchy system. The hierarchy system can be seen in the Figure 2.

In the current project, there are five categories of requirements in DOORS:

- OBC: Requirements From Customer
- ERD: Equipment Description Requirements
- SSS: Software System Specification
- SRS: Software Requirement Specification
- TSPC: Test Case Specification

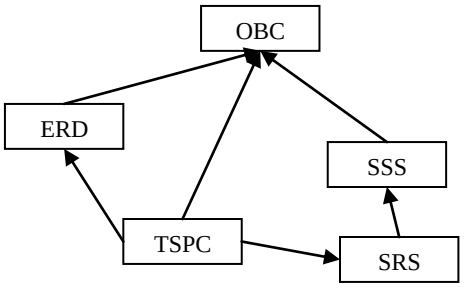

Figure 2. Requirement hierarchy

Furthermore, the project team uses MS Excel files to keep the track of the integration plan as well as the verification status file.

4. RESEARCH METHODOLOGY

The overall goal behind our thesis project was improving the quality in the I&V unit of RUAG Space AB by proposing a measurement system that can measure the stability of requirements before the start of the integration phase. We addressed the following research questions in our study:

- *Which requirements stability metrics are important for RUAG Space AB?*
- *How to integrate these metrics within integration and verification process in RUAG Space AB?*
- *Can requirements stability metrics be used as a predictor for test case execution success?*

To address the questions we chose the case study research methodology as it was applicable to our thesis. We had the opportunity to study the ongoing processes in the unit of integration and verification at a company, in particular making interviews with stakeholders was possible for us. Getting the access to the DOORS requirement management system, test cases specifications, and test result log files were given by the studied unit to us. Furthermore, with conducting a case study, it was possible for us to decide in advance what we want to investigate, how to design the case study and how to plan to collect the required data to support it.

Case studies are very suitable for industrial evaluation of the software engineering methods and tools. They sample from the variables representing the typical situations [20]. The level of control over variables is more limited than the level of control in experiments. [20] It is a preferred technique in situations where there is no need to have a strict control over the variables of study [21]. In our project, the only variables we controlled were the requirement stability metrics that we changed many times in order to reach to the best set of metrics that fitted the needs of integration and verification unit. However, we did not have any control over the other variables in studied unit such as the verification and integration activities as they were totally managed by the stakeholders in RUAG Space AB. With this limited control that we had over the project variables, the case study was considered as most suitable research method comparing to the formal experiment method that needs more freedom to have a control over project variables.

4.1 Case Study Design

The study at RUAG Space AB was conducted between March and May 2010. In order to address the research questions, we divided our case study design into four steps: literature review, introducing initial requirement stability metrics, data collection, and making interviews with stakeholders to validate the metrics and evaluate the findings. The following figure shows the steps taken in the case study, they are discussed in more detail in the following chapters.

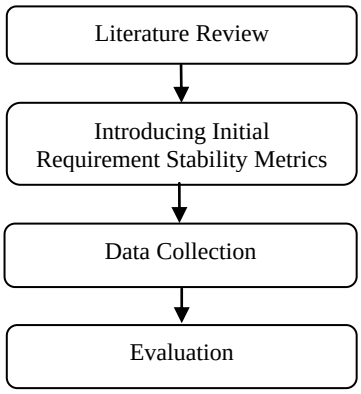

Figure 3. Design of Case Study

4.2 Literature Review

In order to find the related work done in the area of requirement stability in correlation with test case execution success we explored the databases of digital libraries, such as:

- IEEE,
- ACM,
- Science Direct,
- Springer-Links,
- Google scholar search engine, and
- DBLP Computer Science Bibliography.

Following are the keywords used to search through the databases (we combined them with AND and OR to make more extensive searches):

- Requirement,
- Stability/instability/volatility,
- Metrics,
- Measurement,
- Management,
- Test case,
- Test execution, and
- Failure/success.

In total, we found 35 papers that we had to filter in order to find the most relevant ones to our study. To achieve this, we read the abstract, introduction and conclusion parts of the papers. When a paper was found relevant, we read it completely to investigate it more. The overall goal of the literature study was to increase our knowledge in the area of requirement stability, searching for related work, finding out whether any research has been done in the area of introducing requirement stability metrics and applying them as predictors for test case success. We also looked for a list of requirement-related metrics along with their definitions, calculation methods and applications in papers, our findings are listed in Appendix A.

The result of the literature review revealed that although various studies have been done on requirement stability area, we found no research specifically focuses on investigating whether requirement stability metrics can be applied as predictors for test case execution success.

4.3 Initial Requirement Stability Metrics

As there was no requirement stability metric collection in RUAG Space AB we proposed five requirement stability metrics based on an unpublished research at another company and validate them in a later phase: 'Number of Requirements per Test Case', 'Number of Requirement Changes per Test Case', 'Number of Requirement Changes in Last Seven Days per Test Case', 'Number of New Requirements per Test Case', and 'Number of Not-Established Requirements per Test Case'.

In order to better understand the metrics related to requirement changes, first we describe what the definition of 'change' in requirements stability context is, and in the next chapters we introduce the detailed definition of metrics.

As discussed in Chapter 3, in the company the requirements are managed in DOORS. Requirements are defined by the three most important attributes:

- Requirement Text (specification): This attribute contains requirement specifications in the text format most often. It can also contain OLE objects which supplement to the requirement specifications.
- Requirement Status: This attribute is used to mark stability status of the requirements. There are two values for this attribute: unstable and established. Whenever a requirement's status is set to established, it is ready for implementation.
- Links: A requirement can refer to other requirements or test case specifications. If the requirement refers to a test specification, we assume it is directly linked to a test case.

On the other hand, if a requirement refers to another requirement which is directly linked to a test case, we assume the requirement is indirectly linked to a test case.

According to the company's process, the requirement status attribute is manually set by requirement engineers but setting it is not a mandatory action. Therefore, we do not count change of this attribute as a real change.

Consequently, we define that a requirement is considered as being changed if and only if the 'Requirement Text' attribute is different from its previous value. Due to the technical limitations, differences in embedded OLE objects are not counted.

The followings are definitions of the five initial requirement stability metrics with their pseudo code to calculate them.

4.3.1 Number of Requirements per Test Case Metric

This metric is the total number of requirements which are directly or indirectly linked to a test case. All the requirements in different categories and different hierarchical levels are counted in this metric. The metric is calculated for each test case.

```
1.  test_item = findTestItemInTSPC(TestcaseID)
2.  stack = create empty stack
3.  for each item in linksIn(test_item)
4.    push item into stack
5.  end
6.  metric_value = 0
7.  while stack is not empty
8.    pop_item = pop from stack
9.    metric_value++
10.   for each item in linksIn(pop_item)
11.     push item into stack
12.   end
13. end
```

To calculate this metric, all the test items in 'Test specification' module (TSPC) are traced to find a specific test case. Thereafter, all the requirements linked to this test case are recursively traced using a stack in pseudo code to count the number of all linked requirements.

4.3.2 Number of Requirement Changes per Test Case Metric

This metric is the total number of changes that have been made to requirements which are directly or indirectly linked to a specific test case. All the requirements in different categories and different hierarchical levels are counted. The metric is calculated for each test case. The calculation algorithm is as follows:

```
1.  test_item = findTestItemInTPSC(TestcaseID)
2.  stack = create empty stack
3.  for each item in linksIn(test_item)
4.    push item into stack
5.  end
6.  metric_value = 0
7.  while stack is not empty
8.    pop_item = pop from stack
9.    for each change in changesList(pop_item)
10.     metric_value++
11.   end
12.   for each item in linksIn(pop_item)
13.     push item into stack
14.   end
15. end
```

Similar to the previous algorithm, all the requirements linked to a specific test case are recursively visited using a stack. For each visited test case, the script checks the changes list, which is daily updated by comparing two latest versions of the requirements. The metric value is sum of all the changes found.

4.3.3 Number of Requirement Changes in Last Seven Days per Test Case Metric

Calculation of this metrics is similar to 'Number of Requirement Changes per Test Case' metric, the only difference is that the changes occurred in the last seven days are counted. The seven days time frame is used for capturing the weekly aspects of software projects. This metric is thought to prevent this problem and show only the recent changes occurred in requirements of a specific test case. This metric is calculated for each test case using the following algorithm:

```
1.  test_item = findTestItemInTPSC(TestcaseID)
2.  stack = create empty stack
3.  for each item in linksIn(test_item)
4.    push item into stack
5.  end
6.  metric_value = 0
7.  while stack is not empty
8.    pop_item = pop from stack
9.    for each change in changeList(pop_item)
10.     if change is in last 7 days
11.       metric_value++
12.     end
13.   for each item in linksIn(pop_item)
14.     push item into stack
15.   end
16. end
```

This algorithm is almost the same as the previous one. The only difference is that only the changes which were made in the last seven day are counted in calculating the metric value.

4.3.4 Number of New Requirements per Test Case Metric

This metric is the total number of requirements which are newly added to the project and linked to a specific test case. All the

requirements in different categories and different hierarchical levels are counted. The metric is calculated for each test case.

```
1.  test_item = findTestItemInTPSC(TestcaseID)
2.  metric_value = 0
3.  for each item in addedList
4.    parent = linkOut(item)
5.    while parent is not null
6.      parent = linkOut(parent)
7.    end
8.    if parent = test_item
9.      metric_value++
10. End
```

The given test case is searched in the 'Test Specification' (TSPC) module in DOORS. Then the script looks at the added requirements list, which is updated every day. For each requirement, the script checks whether it is linked to the given test case. If so, 1 is added to the metric value.

4.3.5 Number of Not-Established Requirements per Test Case Metric

This metrics is calculated by counting the number of 'unstable' requirements in all categories and hierarchical levels in DOORS which are directly or indirectly linked to a specific test case. The metric is calculated for each test case.

```
1.  test_item = findTestItemInTPSC(TestcaseID)
2.  stack = create empty stack
3.  for each item in linksIn(test_item)
4.    push item into stack
5.  end
6.  metric_value = 0
7.  while stack is not empty
8.    pop_item = pop from stack
9.    if statusOf(pop_item) = "unstable"
10.     metric_value++
11.   for each item in linksIn(pop_item)
12.     push item into stack
13.   end
14. end
```

All the linked requirements are recursively traced using a stack. For each requirement, the script checks whether the status attribute is set to 'unstable'. If so, the metric value is increased by 1.

4.4 Data Collection

As discussed earlier, in our research, we needed to collect the metrics data on a daily basis to evaluate their validity. Therefore, developing a measurement tool with the ability of gathering metric data was one of the most important tasks in the project.

In a research project that was conducted in Ericsson, Staron et al. [22, 23] introduced a framework for developing a measurement system and its industrial evaluation. The framework implemented the ISO/IEC 15939 standard [12] for the software measurement. It is based on pre-prepared MS Excel templates to be flexible and independent of information sources. The idea of this framework allows the measurement system to be able to work with the different data sources without changing its internal structure. New measures can be quickly implemented but do not impact the other measures. In addition, the framework allows the measurement system to do any further statistics easily since requirements data are kept in MS Excel sheets and do not require connecting to data source again. Therefore, we decided to apply the approach introduced in this standard to develop our measurement tool. This framework allowed us to implement our measurement tool in a standardized manner. However, we customized the framework implementation to be the best fit in the RUAG Space AB context.

Firstly, by investigating the industrial context, we characterized the following requirements which the measurement system should fulfill:

- The ability to extract the requirements data which is required to calculate the five initial metrics described in the previous sections.
- The ability to run automatically on a nightly basis without user intervention and generate accurate results.
- The ability to work transparently and make no changes to the other current systems at RUAG Space AB.
- The ability to keep the history of metric values to draw the trend diagrams over the time.

Secondly, the information flow is defined according to the Figure 4. The measurement system takes requirements as input data and produces metric values and trend diagrams as output.

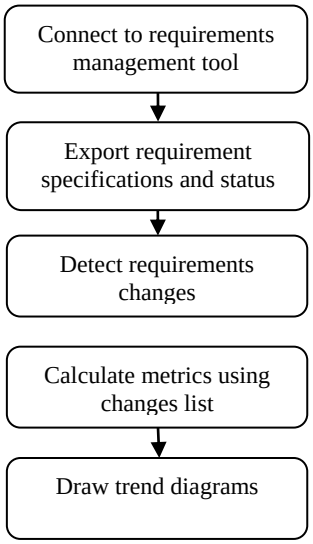

Figure 4. Measurement tool information flow

The whole scenario of measurement system including the export of requirement data into the internal MS Excel files, finding

requirement changes, generating trend diagrams and gadget file are shown in the Figure 5.

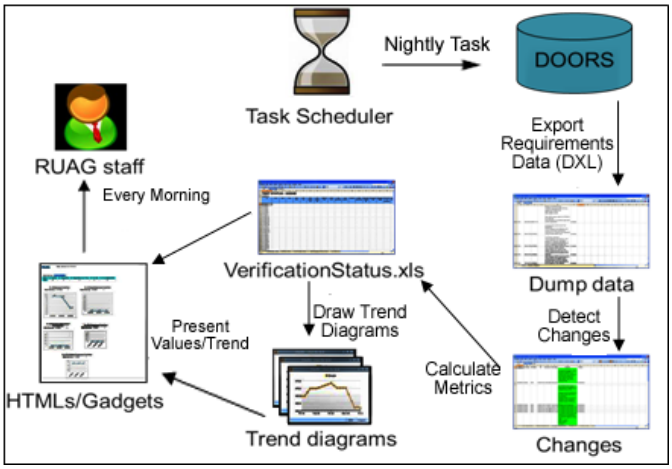

Figure 5. Measurement system architecture

The system is developed using MS Excel VBScript and DXL (DOORS Extension Language)– a scripting language which is designed for DOORS manipulation and interaction [24].

The Windows Task scheduler is used as the bootstrap loader of the measurement tool. It triggers a DXL scripts running in DOORS batch-mode to export requirements data into an MS Excel file. When the exporting task finishes, the script will activate the VBScript codes in MS Excel files.
The measurement system uses three different Excel files:

- ReqsDump.xls: This file contains raw requirements data exported from the DOORS.
- ReqChanges.xls: This file contains all the requirement changes gathered from the beginning of data collection process. The changes are categorized into three different worksheets: new requirements, deleted requirements, and changed requirements. To identify the requirements changes, this Excel file keeps two last versions of raw requirements data. When new requirement data are exported into ReqChanges.xls, these two versions are updated accordingly, and then the changes are detected by a simple comparison between data of two versions.
- VerificationStatus.xls: This file is the extension to the RUAG Space AB verification status file. As a supplement to the status columns for each test case, we added five more columns, in which the test case metrics values are stored. This file also contains history of metrics values in five separated worksheets. The trend diagrams are generated by running a macro in this file. Thereafter, they are exported in format of .jpg files to be used in representing the trend charts in gadget HTML.

We used a HTML gadget file to present our system, which embeds the metrics values, trend diagrams for the individual test cases, as well as the statistics to show the most noticeable test cases.

4.5 Evaluation

We held six interviews with four stakeholders from integration and verification unit in order to evaluate the validity of proposed metrics and investigate whether they can be used as predictors for success of test cases. The interviewed people have the following roles in I&V unit:

- Software validation manager: The responsibility for software validation implies issuing the software validation documents (software validation plan, software validation test specification, and software validation test report), developing the TASW (Test Application Software) and validation test scripts. The person also controls if the software development process is followed for the software testing. The person has been software validation manager for four years at RUAG Space AB.
- Integration responsible: This role is quite new at RUAG Space AB and it is the first time that the integration responsible works in this role. The person is responsible for making the integration plan updated every day according to the latest changes in the project. Furthermore, the person has been the technical team leader (object manger) for both hardware and software for ten years.
- Software requirement and design responsible: This person is responsible for writing and maintaining the software requirement documents. He also implies defining the overall software structure and specifying the module interfaces. This employee has worked with software development for about 30 years and has been the formal software requirements responsible in various projects for about 25 years.
- Verification object manager: This role usually implies acting as the verification object manager for data handling projects. The main task is to manage the 'Provide Test Equipment' development process. This includes the overall responsibility for all the tasks included in the 'Provide Test Equipment' process, and keeping the time schedule and cost budget. The second major responsibility is to establish the test specification and other critical test documents. In some projects the verification object manager also has the role of responsible for software validation. This employee has worked as verification object manager in six major data handling projects over the last 10 years and has been with the company for almost 15 years.

The data obtained during the interviews helped us to get a better understanding of how the requirements change and why the test cases fail. It also served us to improve the next interviews, making them more efficient and to the point.

The followings are brief summary of the interviews:

- *Interview with software validation manager:* We asked the software validation responsible to check the gadget file daily and notify us whether any unexpected changes happened in the requirements linked to a specific test case. In particular, the questions asked were intended to increase our knowledge about how requirements, test cases and test execution are managed in the RUAG Space project, as well as to capture

the unexpected changes, find out the reasons of requirement instability, and getting feedback to the developed measurement system and proposed metrics.

- *Interview with integration responsible:* The interview with the integration responsible focused on understanding which parameters are considered when he updates the integration plan daily every day and whether the proposed metrics and data presentation in the gadget file can be helpful for him in updating the plan according to the daily requirement changes.
- *Interview with software requirement and design responsible:* The questions asked to software requirement and design responsible were intended to find out the interdependency between requirements of different levels, how changes in upper level requirements affect the lower level requirements, the reasons of changes occurring in requirement, getting suggestions to improve the measurement systems and metrics.
- *Interview with verification object manager:* In particular, in our questions we focused on finding the reason for test case failures, the effect of requirement instability on failures, getting the feedback about proposed metrics and asking for suggestions to add new metrics into our system.

In the next chapter you can find the results of evaluation and analysis activities on metrics.

5. RESULTS AND ANALYSIS

The results of the thesis are derived from the interviews we held with people involved in the current project in order to evaluate how efficient the proposed requirement stability metrics are and whether they can be used as predictors for test execution success in integration and verification activities of RUAG Space AB. The evaluations for each of the initially proposed metrics, modifications, and introduced new metrics are included in this chapter.

5.1 Evaluating 'Number of Requirements per Test Case' Metric

According to the opinion of the interviewees, the metric 'Number of Requirements per Test Case' is an interesting one as it was integrated into the gadget file and the stakeholders could observe the changing trend in the number of requirements easily. Although in terms of importance this metric had the least importance among the metrics as it represents the increase and decrease in number of requirements in gadget file it was considered helpful to the stakeholders. According to interviewees' opinions, the most useful metric is the 'Number of Requirement Changes in Last Seven Days per Test Case' since it informs about the most recent changes occurred in test cases during the last one week and can be utilized in the integration processes.

Furthermore, the stakeholders suggested us to change the trend chart of this metric in a way that they could see the T0 milestone line. It is important for them to see whether the requirement additions or deletions still happen after the T0 integration milestone. In fact, all requirements should get frozen before T0 as the test case implementation and software coding phase starts in the next step.

Change in number of requirements causes a delay in integration process. The test cases are dependent on the requirement specifications and should get updated whenever the requirement additions or deletions occur in DOORS.

After updating the test cases, they are reviewed by the requirements engineers to check for correct understanding of the requirements. Some requirements are un-testable, or difficult to test with the given descriptions, therefore the requests may be made for changing them. Once the reviews are performed, the test cases are updated and the coding of the tests cases begins. The test case writer follows up with the implementer and reviews the test code to check that the test code is correctly implemented. The test code is reviewed, and the code review is the only thing that is on paper. The code is copied on MS Word documents and reviewed. The found problems are written down as comments. At this time the software should be completely module tested and code reviewed. At the end there is an official run of the tests, the test log files are saved and reviewed and the results are saved. All the log files should be reviewed and they are available for the customer to review. All these tasks should be fulfilled to integrate a new requirement into the system which imposes a delay in the whole project if it occurs after T0 milestone.

The metric 'Number of Requirements per Test Case' is helpful to find out how many requirement changes occur for each test case in DOORS. The Figure 6 shows a sample trend chart for this metric. In the next figure you can see the same chart when the T0 line is added into the diagram.

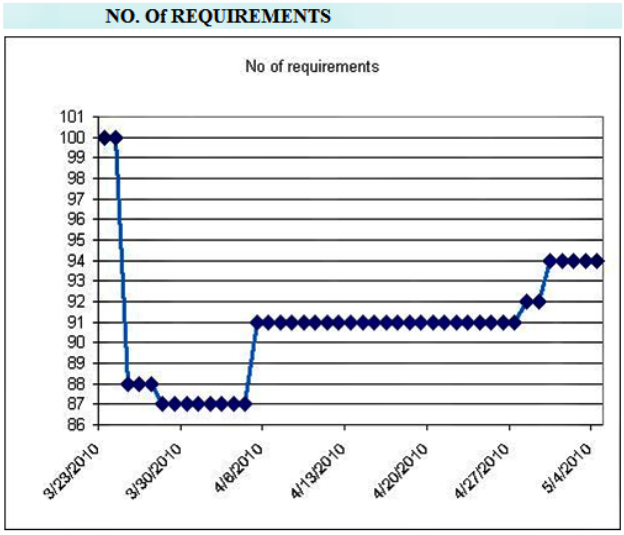

Figure 6. Number of Requirements per Test Case Trend Chart

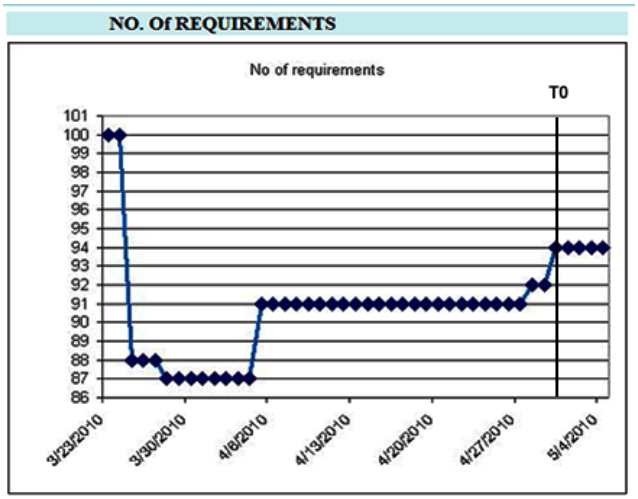

Figure 7. Number of Requirements per Test Case Trend Chart with T0 integration milestone

As it can be observed in the Figure 7 the T0 milestone appeared quite late in the studied project, which means that we did not have a good opportunity, due to the time limitations, to evaluate the requirements stability metrics after this milestone.

5.2 Evaluating 'Number of New Requirements per Test Case' Metric

During the period of time we observed the gadget charts, we realized that no changes occurred in the value of the metric 'Number of New Requirements per Test Case' as it always represented a flat line of zero. This can be seen in Figure 8 which is a snapshot from the new requirement trend chart.

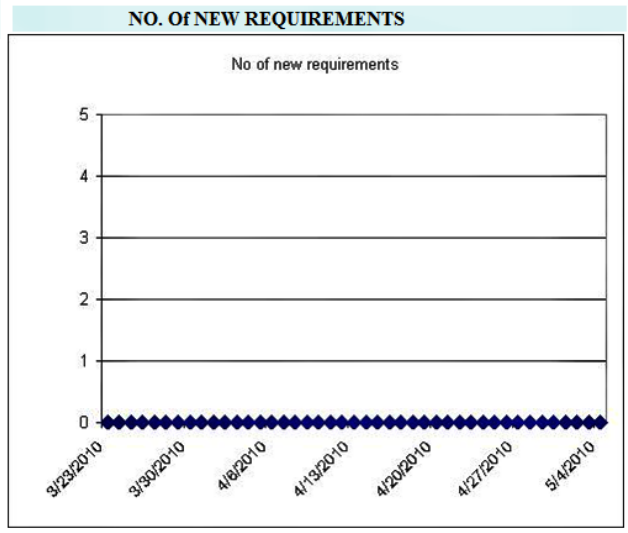

Figure 8. Number of new requirement trend chart

The software validation manager claimed that the new requirements added into DOORS during the period we collected data. However, we it was not reflected into our charts. To investigate more about why this happened, we examined the history of newly added requirements in our Excel databases. It was revealed that the new requirements are added into DOORS but as there are no links provided from them to the test cases it was not possible to trace them back to the test cases and that was why for this metric we always received a value of zero. According to the talk with the software validation manager, this problem could happen because of the requirement engineers have not yet finished with the specification of new requirements and it takes more time to be completed and linked with the test cases.

Therefore, to solve this problem we decided to modify the definition of 'Number of New Requirements per Test Case' metric in a way that regardless of having link with test cases we can see how many requirements are added into system. Hence, this metric replaced with a new metric of 'Number of New Requirements' that is calculated per project, rather than per test case.

5.3 Evaluating 'Number of Not-Established Requirements per Test Case' Metric

The status of a requirement is set by default to 'unstable' when a requirement is added into DOORS. This fact can be seen in the Figure 9 which is a snapshot from the gadget file. In this chart, the number of requirements has increased from 27 to 29 for a specific test case and the same increasing pattern can be followed in the Figure 10 that represents the chart for the number of unstable requirements for the same test case.

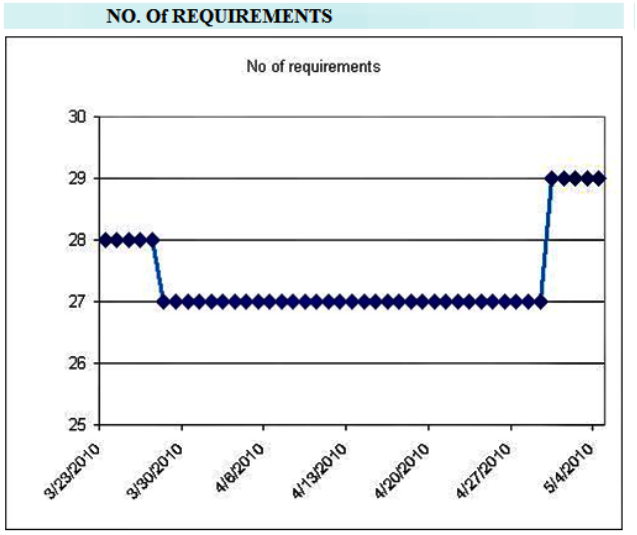

Figure 9. Number of Requirements per Test Case Trend Chart

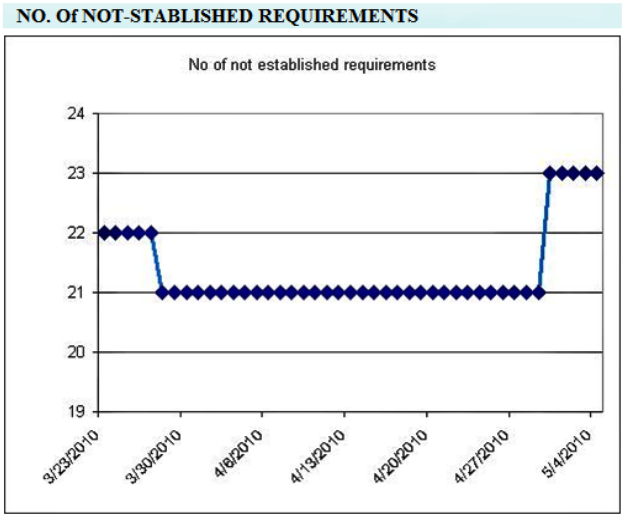

Figure 10. Number of Not-Established Requirements per Test Case Trend Chart

On the other hand, the number of requirement changes chart, as shown in the Figure 11, reveals a flat line of zero which means that no changes occurred when new requirements were added into DOORS. Thus, by default the status of requirements is set to 'unstable' in the beginning when they are added into the system.

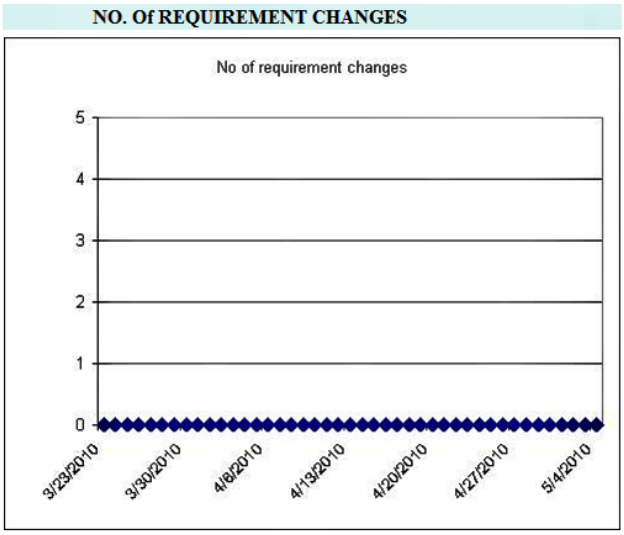

Figure 11. Number of Changed Requirements per Test Case Trend Chart

The status attribute is handled manually by the requirement engineers. Setting the value of this attribute is not a mandatory action. It is considered as an extra effort to set this attribute when adding new requirements into DOORS. In addition, it may happen that people forget to change the status of a requirement to 'established' although in real it occurs and the requirement is established. Therefore, we did not count the change of this attribute as a meaningful one and the metric we based on this attribute 'Number of Not-Established Requirements per Test Case' was not a reliable metric for stakeholders.

We suggest it to avoid establishing any metric on this attribute unless it is managed in a controlled way. Finally, as for our evaluation the metric 'Number of Not-Established Requirements per Test Case' was removed from our list of metrics and the company decided to be more stringent on setting the attribute "established" at T0 that this was not the case before our research.

5.4 Evaluating 'Number of Requirement Changes per Test Case' Metric

There are various reasons for changing the specifications of the requirements, which are described as follows:

- Some changes in the requirements are demands from the customers. RUAG Space AB sends the specification of requirements in the form of documents to the customers to be reviewed and confirmed. If the customers are not satisfied with the specifications, they request changes in the requirements. This type of change is considered as an external change into the system.
- The internal reviews on requirements specifications are done in RUAG Space AB by requirement engineers to ensure that the customer needs are fulfilled in them. It is possible that some changes occur at this stage.
- Some changes occur internally by people from different departments of RUAG Space AB (e.g. system department, software design department, hardware department) into system. When people work on the design and implementation phase they change the requirements so often, that it affects the system. This kind of change is considered as internal as is a demand from inside the company. Additionally, misunderstanding between the different departments of RUAG Space AB in the definition of the requirements can be another reason for changing the requirements. Also, it may be considered that there is not enough input for the tests so testers may need adding more details into the requirements.
- Most often, no change occurs in the requirements after the execution of test cases. However, sometimes, after analyzing the result of the test execution some requirements should be updated. This type of change does not occur often and is not a main reason for changing the requirements.
- Sometimes the hardware does not work properly as expected. Therefore, they should change the related requirements accordingly.
- Existing bugs in the requirements and working on incomplete requirements cause changes in the requirement specifications. If bugs are found after T0, they cause a delay in the integration plan as the requirements should be fixed before the start of the integration process.
- It may be found out that there is not enough input for the tests so people add more details into the requirements.
- There is interdependency between requirements of different levels. For example, if any change occurs in the SSS level requirements most likely the SRS requirements should get updated accordingly. It is likely that changes in the SSS and SRS affect the test cases. The test cases linked to these requirements should be updated as well since they must cover all aspects of the requirements. There is a direct link from TSPC (TC Specification) to the SSS, SRS, ERD, and OBC Spec requirements so it is highly possible that requirement changes demand changes in TSPC as well.

Regardless of the reason for changing the requirements, the metric 'Number of Requirement Changes per Test Case' captures all of the changes occurring in the 'requirement text' attribute of the requirements linked to a test case. The verification object manager (verification and validation) believes that this metric is useful for the system as it reflects how stable the requirements related to a specific test case are. He found it likely that there is a correlation between this metric and failure of test cases as after changing the requirements the test cases should be updated as well and if they are not updated most probably the test case execution will face failure. This, in turn, prolongs the process of integrating test cases into the system. Sometimes the test team does not realize when a new requirement is added to DOORS.

A change is critical into system when it passes the T0 integration milestone as every requirement should be frozen after that. It was very important for the stakeholders to know whether any changes occur after T0, and if it happens how many requirements have been changed. The current chart for the metric 'Number of Requirement Changes per Test Case' does not show T0 line but as a future effort we should implement it in our gadget file.

Furthermore, the verification object manager suggested that we implement the existing metrics per integration step instead of per test case as there are a few requirements in DOORS that do not have any link to the test cases so their changes cannot be reflected in our metrics.

To investigate the relation between the failed test cases and the stability of requirements, we looked for the test cases in test status log file, which failed due to the problems in the integration stage. The status of these test cases is reported as NOK (not Ok) in this file. Then we checked their situations in the gadget file to see their change trend in the requirement stability metrics. The result showed that out of 17 failed test cases reported in the verification status, 13 test cases experienced a stable situation in their requirements and had flat lines in their charts. This fact revealed that besides volatility of requirements there were other reasons for failure of the test cases at the integration time. After having discussed the matter with the stakeholders, we summarize the reasons for the failure as follows:

- The failure can occur because of a crash in the software.
- Another reason is failure of hardware in lab.
- Existing bugs in the test case code that is the most common reason for failure.
- A change can also occur in the specification of tests, as test cases are updated by test team until they are fixed and stable. The change can happen because of the new functionalities added to the system that are not specified enough or maybe the person working on them is not enough experienced and therefore change them several times until they are fixed.

Although there were different origins for test case failure, the verification object manager still believed that the requirement stability metrics could be helpful indicators to receive earlier warnings before the test cases are failed. To justify his claim, he pointed out that there is no systematic way to inform the test team about the change in requirements when they get updated by the requirement engineers. They inform the test team about updates only in an informal verbal way. It may even occur that they forget to report the changes. Additionally, sometimes people think that a requirement change does not require updates in the corresponding test cases so they just simply ignore them and do not inform the test team about updates. Thus, when a not-updated test case run on the system it is likely that it fails.

Similarly, the test team does not always realize when a new requirement is added to DOORS. Again, this is because of the fact that there is no controlled way to inform newly added requirements to the test team.

5.5 Evaluating 'Number of Requirement Changes in Last 7 Days per Test Case' Metric

The evaluation of this metric is the same as the last metric, 'Number of Requirement Changes per Test Case' as it has the same functionality. The only difference is that it shows the number of recent changes occurred in the last 7 days and its value is cumulative just for the data from one week. But, the 'Number of Requirement Changes per Test Case' metric counts all the changes occurred from the beginning time of data collection and its value is cumulative for the whole period of data collection.

This metric was considered as the most helpful one for the verification object manager as he can see the most recent changes in requirements.

According to the suggestions from stakeholders, the T0 line which is a milestone in the integration activities should be drawn in the trend charts related to the two last metrics. The current trend charts do not show this line.

6. CONCLUSIONS

In this thesis, we implemented and evaluated five requirement stability metrics applied to an industrial project in RUAG Space AB. A case study was conducted as our research methodology to evaluate the metrics.

We successfully developed a measurement system which collects the metrics data on a daily basis. This tool which implements ISO/IEC 15939 standard is able to run automatically without user intervention. The system was presented at RUAG Space AB and found very interesting for engineers and managers. They believed that the tool is very useful to integrate in their daily work.

The interviews were held with the project members to validate the proposed metrics and find out whether any of them can be used as predictor of test case execution success in RUAG Space AB. During the interviews, the metrics and trend charts were presented to the members to make them realize how unstable the requirements behave in DOORS and to get their opinions as well. The stakeholders did not have the opportunity to observe the requirements changes before the implementation of our measurement system, so it was found very interesting to them that we could expose the information that were hidden for the stakeholders. The automated data collection, analysis and presentation were integrated into integration and verification processes and found very useful in the daily work at RUAG Space AB.

The evaluation of outcomes showed that among the introduced metrics, 'Number of Requirement Changes per Test Case' is the most probable candidate to be an indicator for success of test case execution while 'Number of Changes per Test Case in Last 7 Days' caught the attention of the verification object manager, who wanted to know what happened in the project recently. 'Number of Requirements per Test Case' can also be considered as an

indicator to show the complexity of the test cases. Meanwhile, the two other metrics 'Number of New Requirements per Test Case' and 'Number of Not-Established Requirements Per Test Case ' did not make so much sense due to the fact that the integration and verification process was not fully standardized in RUAG Space AB. In addition, a number of metrics were suggested as an improvement: 'Number of New Requirements per Project' and 'Number of Requirement Changes per Integration Step'.

Although the problem of measuring requirements stability has been researched from several theoretical angles, but little has been done in terms of empirical validation of requirements stability metrics. The impact of requirement instability on the success of test cases has not been investigated in industry. In our research, we could study the problem of requirement instability in relation with success of test cases. We implemented five requirement stability metrics and empirically validated them to reach to a final set of metrics applicable in the RUAG Space AB context.

The previous studies on requirement stability metrics lack providing a precise definition to the metrics that creates ambiguity in understanding the meaning and functionality of metrics. We resolved this issue by providing clear metric definitions. We wrote pseudo code algorithms for measurement of metrics to clear their definitions. The algorithms can be reused by others.

Only a few requirement stability metrics that were introduced in previous studies have a clear measurement method. However, we have developed a measurement system which conform a standardized framework and is able to capture the requirement data, run automatically, and can work transparently within RUAG Space AB's existing systems.

Comparing to the requirement stability metrics that are only helpful in the assessment of requirement analysis, project management and project planning processes, our implemented metrics provide a better control over the test process that may occur in later phases of software projects.

However, because of the time limitation, the thesis needs further work in the future. So far, we did not find enough evidence that requirement stability metrics can be helpful in anticipating the success of test cases. The project will only start the execution of test cases after the thesis is finished. Therefore, we are not able to perform a full empirical validation for the potential of the metrics to act as predictors. The implementation and validation of the new suggested metrics are also necessary to be done in the future.

To sum up, the thesis achieved useful results which can make contributions to the progress of research and application of software metrics in industry. We have set up a ground at the company for future work, which can be used for a longer period of time to collect the data and provide more advanced analyses about the requirements stability of test case execution success. As it was not possible to perform the full empirical validation of the metrics, we suggest RUAG Space AB to continue our work to extend the results in the future.

ACKNOWLEDGMENTS

We would like to thank RUAG Space AB for providing the possibility of doing this study and their support to us. We would also thank our supervisor, Dr. Miroslaw Staron, for suggesting us this project, giving support through whole project and providing helpful feedbacks to improve our thesis report.

7. REFERENCES

[1] Brooks, F. 1987. No Silver Bullet: Essence and Accidents of Software Engineering. IEEE Computer, Vol. 20, No. 4, April 1987, 10-19.

[2] Jayaswal, B. K., Patton, P. C. 2006. Design for Trustworthy Software: Tools, Techniques, and Methodology of Developing Robust Software, 1st Edition. (September 2006), Prentice Hall edition.

[3] Zowghi, D. 2002. A Study on the Impact of Requirements Volatility on Software Project Performance. Proceedings of Ninth Asia-Pacific SE Conference (APSEC' 2002), IEEE Computer Science.

[4] Taghi, M., Khoshgoftaar, N., Sundaresh, N. 2006. An empirical study of predicting software faults with case-based reasoning. (June 2006), Software Quality Control.

[5] Jiang, Y., Cukic, B., Ma, Y. 2008. Techniques for evaluating fault prediction models. (October 2008), Empirical Software Engineering.

[6] Arisholm, E., Briand, L. C., and Johannessen, E. B. 2010. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. Journal of Systems and Software 83, 1 (Jan. 2010), 2-17.

[7] Cambrige Advanced Learner's Dictionary. 2008. 3rd edition. Cambridge University Press, (April 2008).

[8] Henry, J. and Henry, S. 1993. Quantitative assessment of the software maintenance process and requirements volatility. In Proceedings of the 1993 ACM Conference on Computer Science (Indianapolis, Indiana, United States, February 16 - 18, 1993). CSC '93. ACM, New York, NY, 346-351.

[9] Li, L., Shu-guang, H., Er-shi, Q. 2008. On Software Requirement Metrics based on Six-Sigma. Advanced Management of Information for Globalized Enterprises, AMIGE 2008.

[10] Rosenberg, L.H., and Hyatt, L. 1996. Developing an Effective Metrics Program. In proceedings of the European Space Agency Software Assurance Symposium, the Netherlands, (March 1996).

[11] Malayia, Y.K., Denton, J. 1999. Requirements Volatility and Defect Density. 10th IEEE International Symposium on software reliability Engineering, Boca Raton, Florida, (Nov 1999), 28-39.

[12] ISO/IEC 15939:2007, Systems and software engineering - Measurement process: http://www.iso.org/iso/catalogue_detail.htm?csnumber=44344

[13] Javed, T., Maqsood, M. e., and Durrani, Q. S. 2004. A study to investigate the impact of requirements instability on software defects. SIGSOFT Software Engineering Notes 29, 3 (May. 2004), 1-7.

[14] Wyatt, V., DiStefano, J., Chapman, M., Aycoth, E. 2003. A Metrics Based Approach for Identifying Requirements Risks. Software Engineering Workshop, Annual IEEE/NASA Goddard, p. 23, 28th Annual NASA Goddard Software Engineering Workshop (SEW'03), 2003.

[15] Ambriola, V. and Gervasi, V. 2000. Process Metrics for Requirements Analysis. In Proceedings of the 7th European Workshop on Software Process Technology (February 21 - 25, 2000). R. Conradi, Ed. Lecture Notes In Computer Science, vol. 1780. Springer-Verlag, London, 90-95.

[16] Kammler, D. W. 2000. A First Course in Fourier Analysis, 1st Edition. Upper Saddle River, NJ: Prentice Hall.

[17] Loconsole A. 2001. Measuring the Requirements Management Key Process Area. Proceedings of ESCOM - European Software Control and Metrics Conference, London, UK (April 200).

[18] Solingen, R. V., Berghout, E. 1999. The Goal/Question/Metric Method. McGraw-Hill Education, US (January 1999).

[19] Lam, W., Loomes, M., and Shankararaman, V. 1999. Managing Requirements Change Using Metrics and Action Planning. In Proceedings of the Third European Conference on Software Maintenance and Reengineering (March 03 - 05, 1999). CSMR. IEEE Computer Society, Washington, DC, 122.

[20] Wohlin, C., Runeson, P., Höst, M. 1999. Experimentation in Software Engineering: An Introduction. Springer (December 1999).

[21] Fenton, N. E., Pfleeger, S.L. 1997. Software Metrics: A Rigorous & Practical Approach. PWS publishing company, Boston.

[22] Staron, M., Meding, W., and Nilsson, C. 2009. A framework for developing measurement systems and its industrial evaluation. Information and Software Technology. 51, 4 (Apr. 2009), 721-737.

[23] Staron, M. and Meding, W. 2009. Using Models to Develop Measurement Systems: A Method and Its Industrial Use. In Proceedings of the international Conferences on Software Process and Product Measurement (Amsterdam, The Netherlands, November 04 - 06, 2009). A. Abran, R. Braungarten, R. R. Dumke, J. J. Cuadrado-Gallego, and J. Brunekreef, Eds. Lecture Notes In Computer Science, vol. 5891. Springer-Verlag, Berlin, Heidelberg, 212-226.

[24] DOORS DXL Guidelines: https://www.ibm.com/developerworks/wikis/display/dxl/Home

APPENDIX A: LIST OF REQUIREMENT-RELATED METRICS

Metric	Object	Description	Used for	Calculation method
Defects versus requirement changes [13]	Project's release	Number of defects found due to requirement changes throughout SDLC	Examining the impact of RC on software defects throughout the SDLC and the root causes of those defects	1) Requirement changes are categorized into Pre-Release changes(before the system has been deployed) and Post-Release changes (after the system has been deployed) 2) Requirement changes are collected from FS, Change request, Project schedules, then assigned high/medium/low severity 3) Defects are counted in Defect Repository System which are linked to requirements and assigned severity 1 and 2. 4)Metric value is calculated by number of defects with particular severity which is caused by pre-release/post release requirement changes in particular with particular severity
Imperatives/Incompletes/Option/ Weak phrases/Continuances [14]	Individual requirement specification	1) Imperatives: Words and phrases that command that something must be provided 2) Incompletes: Indications of incomplete requirements 3)Option: Words that loosen the specification by giving the developer latitude 4) Weak phrases: Multiple interpretations or ambiguous terms 5) Continuances: Phrases that follow an imperative and introduce lower level specification requirements	Assessing the structure and quality of individual specification and the vocabulary used to state requirements to assist in identifying risks associated with poorly specified requirements that could impact the project	Search a requirement specification's text to count number of keywords and phrases identified as quality indicators. Details are: 1) Imperatives: "shall" 2) Incompletes: "TBD","TBR", "etc." 3) Option: "should" 4) Weak phrases: 5) Continuances:
Directives	Requirement specification document	Measure of references to figures, tables	Providing indications of requirements document quality but not individual requirement indications	Count number of figures and tables in the requirements specification document
Lines Of Text	Requirement specification document	Measure of physical lines of text	As above	Count number of physical lines (CR-LF) in the requirements specification document

Stability [15]	Requirement document	Measure how smooth the two phases of writing and polishing requirements are integrated	Predicting requirements document quality	1) Define: - F(t): Amount of information contained in the requirement at time t. Information volume can be counted as: + Functional score + Behavioral complexity + Static complexity + Document size - δF(t) = F(t)-F(t-1) 2) Use Fourier transform to analysis the frequency of δF(t) 3) Classify into 4 classes: - Low peaks on high frequencies - Low peaks on low frequencies - High peaks on low frequencies - High peaks on high frequencies
Efficiency [15] $W(a,b,F) = \sum_{t=a+1}^{b} \lvert \delta F(t) \rvert$ $E(a,b,F) = F(b) - F(a)$ $\varepsilon(a,b,F) = \frac{E(a,b,F)}{W(a,b,F)}$	Requirement document	Measure efficiency of the analysis process	Estimating the expected efficiency of further iterations of requirements process in similar conditions	1) Define: δF(t) as above where [a,b] is time interval ε(a,b,F) is the metric value
Number of initial requirement [17]	Project	Number of requirements at the beginning of project	Measuring level of stability of the requirements	Total number of requirement established before implementation
Number of final requirement [17]	Project	Number of requirements at the end of project	Measuring level of stability of the requirements	Total number of requirements after delivery
Number of changes per requirement [17]	Project	Number of changes made to the requirement	Measuring level of stability of the requirements	Total number of changes made to a particular requirement
Number of test cases per requirement [17]	Requirement	Number of test cases linked to the requirement	Measuring impact of requirement to testing	Total number of test cases to verify the requirement
Number of changes to requirements proposed [17]	Project	Number of changes made to requirements in Proposed status	Measuring manageability of requirement changes	Total number of changes made to all requirements with Proposed status
Number of changes to requirements open [17]	Project	Number of changes made to requirements in Open status	Measuring manageability of requirement changes	Total number of changes made to all requirements with Open status
Number of changes to requirements approved [17]	Project	Number of changes made to requirements in Approved status	Measuring manageability of requirement changes	Total number of changes made to all requirements with Approved status
Number of changes to requirements incorporated into base line [17]	Project	Number of changes made to requirements which are not base-lined	Measuring manageability of requirement changes	Total number of changes made to all requirements which are not base-lined

Number of changes to requirements rejected [17]	Project	Number of changes made to requirements in Rejected status	Measuring manageability of requirement changes	Total number of changes (?) made to all requirements with Rejected status
Number of requirement affected by a change [17]	Requirement	Impact of a change to all requirements	Measuring impact of a change	Total number of requirements affected when a change is implemented
Number of changes to requirements per unit of time [17]	Time	Total number of changes made to the requirement in a specific time unit (day, week, month)	Measuring if number of changes decrease with time	Total number of chances(?) made to all requirements in a given reporting period
Number of TBDs in requirement specifications [17]	Requirement	Number of requirements that contain "TBD", "To be done"	Measuring completeness of requirements	Total number of requirements whose specifications contain "TBD" or "To be done"
Number of TBDs per unit of time [17]	Unit of time	Number of requirements that contain "TBD", "To be done" in a specific unit of time (day, week, month)	Measuring if incompleteness of requirements decrease with time	Total number of requirements whose specifications contain "TBD" or "To be done" in a given reporting period
Number of requirements scheduled for each software build or release [17]	Build/Release	Size of release/build	Measuring how many requirements were scheduled for implementation	Total number of requirement taken to be implemented in the specific build/release
Number of base-lined requirements [17]	Project	Number of requirements which are base-lined	Measuring how many requirements were base-lined	Total number of all requirements which are base-lined
Change Effort [19]	Change	Effort to implement requirement change	Assisting maintenance project planning and the production of software maintenance contract	Equal to the effort (in person-hours or person-month) required to implement the requirement change
Change Cost [19]	Change	Cost to implement requirement change	Assisting maintenance project planning and the production of software maintenance contract	Equal to the actual cost of implementing the change
Delivery Time [19]	Change	Time to implement requirement change	Assisting maintenance project planning and the production of software maintenance contract	Equal to time (in hours or days, months) to implement the change

Quality Variance [19]	Change	Impact of implementation of requirement changes into quality	Protecting and maintaining specific quality aspects of a software product	Define quality levels of the system Measure quality level before and after implementation of the change Quality variance equal to the differences of the two measured levels
Budget Reduction [19]	Change	Monetary effect of a requirement change on the project budget	Managing software maintenance budget	Equal to difference of budget before and after implementation of change
Requirements Dependency [19]	Requirement	Dependencies that exist among requirements in project	Providing a loose measure of the coupling of the system	Equal to number of requirements that are dependent on a particular requirement
Change Density [19]	Requirement	Times the requirement changes	Distinguishing between stable and instable requirements	Equal to number of times a particular requirement or particular type of requirement has changed within a given reporting period
Requirements Addition/Modification/Removal [19]	Requirement	Different types of requirement change	Providing a measure of maturity of a system	Equal to number of requirements that added/modified/removed within a given reporting period
Error Rate [19]	Requirement	Error rate of implementing requirement change	Assessing how proficient an organization or team implement requirement changes	Equal to number of errors produced per requirement as result of implementation of requirement changes
Fix cost [19]	Requirement	Fix cost of implementing requirement change	Assessing how proficient an organization or team is implement requirement changes	Total cost (money/effort) to fix all errors produced per requirement as result of implementation of requirement changes
Acceptance Rate [19]	Project	Acceptance rate of implementation of requirement changes	Measuring customer satisfaction of implement requirement changes	% of requirement changes accepted by customer ar delivery time within a given reporting period
Timescale Variance/Budget variance [19]	Change	Delivery effectiveness of implementation requirement changes	Measuring how effective the organization is at estimating changes	Difference (both positive and negative) between actual and estimated delivery timescale/budget