A Systematic Review of Software Robustness

Ali Shahrokni & Robert Feldt

Department of Computer Science & Engineering, Chalmers University of Technology, 412 96 Gothenburg, Sweden

Abstract

Context: With the increased use of software for running key functions in modern society it is of utmost importance to understand software robustness and how to support it. Although there have been many contributions to the field there is a lack of a coherent and summary view.

Objective: To address this issue, we have conducted a literature review in the field of robustness.

Method: This review has been conducted by following guidelines for systematic literature reviews. Systematic reviews are used to find and classify all existing and available literature in a certain field.

Results: From 9193 initial papers found in three well-known research databases, the 144 relevant papers were extracted through a multi-step filtering process with independent validation in each step. These papers were then further analyzed and categorized based on their development phase, domain, research, contribution and evaluation type. The results indicate that most existing results on software robustness focus on verification and validation of Commercial of the shelf (COTS) or operating systems or propose design solutions for robustness while there is a lack of results on how to elicit and specify robustness requirements. The research is typically solution proposals with little to no evaluation and when there is some evaluation it is primarily done with small, toy/academic example systems.

Conclusion: We conclude that there is a need for more software robustness research on real-world, industrial systems and on software development phases other than testing and design, in particular on requirements engineering.

Keywords:
Systematic review, Robustness, Software robustness

1. Introduction

As the importance and complexity of software systems increase, both software practitioners and researchers emphasize systematic and effective development methods. A key aspect of these methods is that they help maintain and increase the quality of the resulting software. Software quality is a multi-faceted concept and can be conceptualized and measured using many different quality attributes [22]. For critical software systems quality attributes dealing with dependability and reliability take center stage [13].

Email address: {ali.shahrokni, robert.feldt}@chalmers.se (Ali Shahrokni & Robert Feldt)

Robustness is one such important quality attribute which is defined by the IEEE standard glossary of software engineering terminology [1] as:

> The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.

In one of our previous studies [147] we identified gaps in the state of practice for specifying and assuring software quality. An important gap we identified was the lack of systematic ways to specify quality attributes such as robustness in many companies. One of our initial steps to address this gap was to search in the academic literature for ways to specify, improve and ensure robustness. Since our informal searches found only a limited number of results we here extend our method to a more systematic review of the existing literature on software robustness.

An obvious choice for this purpose was to follow the guidelines on conducting systematic literature reviews presented in [90]. A systematic review goes through all available literature regarding a specific research question or focus area and classifies the results following clear guidelines and steps. Systematic reviews consist of three main phases: planning, conducting, and reporting the review. Although this paper presents our findings from performing a systematic review, to present the results in a more clear and understandable manner, we decided to adopt some of the practices used for systematic mapping to visualize and classify the results of our review.

In this paper we discuss the results of a systematic literature review we performed on software robustness. The objectives of this study are to find the existing literature in the field and classify them according to their phase focus, system focus, and quality of studies. Quality of the studies was measured based on research contribution type (contribution facet), type of research and type, and strength of evaluation. Since the need to conduct this study was identified in an industrial project, an important factor for the studies were the type of evaluation performed, and whether they were done in an industrial setting. This measure is incorporated to evaluate the usability and validity of the studies in the industrial context.

We continue the paper by presenting related works in Section 2. Section 3 presents the research methodology we used to achieve our objectives, together with the results from the planning and conducting phases of the review. Section 4 reports the results from performing the review. The results are presented in a structure based on our research questions. In Section 5 we discuss and analyze the results to identify gaps and strengths in the state of knowledge. Finally, Section 6 gives our conclusions about the presented results and what they imply.

2. Related Work

There are no previous reviews or systematic reviews on the subject of robustness. However, there is a review in the field of quality attributes that can be mentioned here. A short but well known review on non-functional (quality) requirements was made by Chung et al. [34]. This study discusses quality attribute from a general perspective. The paper presents different definitions of software quality and quality requirements. It continues by presenting different classifications of quality requirements, which leads us to the ISO 9126 standard for classifying quality requirements. ISO 9126 divides quality requirements into main categories of *functionality*, *realiability*, *usability*, *efficiency*, *maintainability*, and *portability*. The paper also discusses different methods for specifying quality requirements. An important artifact in this regards is the IEEE standard 830 about recommended practices for software requirements specification.

Another concept that needs to be discussed in this section is software robustness. IEEE standard defines robustness as [1]:

> The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.

However, robustness is sometimes considered a quality for achieving higher dependability in systems by non-academicers. Dependability is an ‘umbrella’, ‘integrative’ concept having multiple attributes [99]. Formally it and its basic sub-concepts are defined as [13]:

> The ability to deliver service that can justifiably be trusted in a software system.

Dependability is the integrate of these basic attributes: *availability* (readiness for correct service), *reliability* (continuity of correct service), *safety* (absence of catastrophic consequences on the user(s) and the environment), *confidentiality* (absence of unauthorized disclosure of information), *integrity* (absence of improper system state alterations), and *maintainability* (ability to undergo repairs and modifications) [13].

Avizienis et al. defines robustness as dependability with respect to erroneous input [13]. However, robustness is not considered a main attribute of dependability, but is characterized as a secondary and specializing attribute [13]:

> An example of specializing secondary attribute is *robustness*, i.e. dependability with respect to external faults, that characterizes a system reaction to a specific class of faults.

Thus, it seems that robustness can either be seen as a specializing attribute within the more general concept of dependability, or it can be seen as an extension to the concept of dependability to the situation of invalid input or stressful environment.

Common to several uses of the term robustness is that a system should show ‘acceptable’ behavior in spite of exceptional or unforeseen operating conditions[50]. One task of requirements on robustness can be to specify different levels of acceptable behavior of the system. This is related to graceful degradation of a system, which means that the system can deliver parts of its originally intended behavior or function despite erroneous operating conditions.

3. Research Methodology

In this section we describe the methodology of a systematic review based on guidelines from [90]. We also discuss the research questions, results from the planning and conducting phases of the review, and the threats to validity.

Systematic reviews were first presented to software engineering by Kitchenham [89] and have since gained an increasing popularity for conducting literature reviews in the field [155, 5, 82, 47, 48, 92]. The goal of a systematic review is “to identify, evaluate, interpret all available research relevant to a particular research question, or topic area, or phenomenon of interest. Individual studies contributing to a systematic review are called *primary studies*; a systematic review is a form of a *secondary study*” [89].

Systematic reviews are conducted to:

- summarize the existing evidence about a topic
- identify gaps in current research

- provide background to position new research activities

This review is based on the guidelines specified in [21, 90, 89]. Parts of the structure we used were inspired by other systematic reviews [49, 106].

As discussed in [90], during the planning phase, the researchers identify the objectives of the review and develop a review protocol that specifies every step in detail. Then, they use the protocol to conduct each step of the review. The steps here are: to identify and select primary studies in the subject, extract data from the studies, assess the quality of the studies, and synthesize data based on the included studies. The last step is reporting where the researchers write and evaluate the report and draw conclusions based on the results.

As discussed in the introduction, we identified the need to conduct a systematic review on the basis of the results from [147] and the need of industry to have a better understanding of the concept of robustness. Furthermore, most academic works about robustness focus on limited parts and implications of the definition provided by the IEEE definition. This fact increases the need for a systematic review to build a larger perspective over the area and categorize the existing studies.

The next step is to identify the research questions. We needed to classify the results based on the focus area of the studies. We also needed to consider the system focus, and assess the quality of research and the contribution based on the type of research, contribution facet, and evaluation of the study. These research questions are defined in Section 3.1.

As another part of the review protocol we needed to decide which databases to search, what search terms to use, what types of studies to include, and decide how to classify and report the results. ISI Web of Knowledge, Compendex and IEEE Xplore digital libraries were chosen. We also used the ACM digital library to control the completeness of our selected studies.

We used the following search string within keywords, title and abstract to find results in the databases mentioned above:

((robust OR robustness) AND software)

The search term was chosen to be very broad in order to cover the most relevant results. The intention was to use more or fewer words depending on the number of hits, which proved to be unnecessary since the number of hits was in a typical range for large systematic reviews.

To select relevant primary studies from the whole set of identified studies the exclusion was made by one researcher. However, to minimize the bias the review protocol required us to use a second person in each step to classify 20% of the studies in every selection step. The control studies were to be selected randomly. The results from the primary and control selections were then to be compared and in case of difference the issue needed to be addressed. The results from this step are discussed in Section 3.5.1.

3.1. Research Questions

This paper identifies the state of knowledge and the gaps in the knowledge in Software robustness. Robustness is the main topic in our ongoing industrial project described in [147]. The goal of the project is to identify and improve the state of art and practice of software robustness in different phases of software development. To answer these questions we have specified the following research questions:

RQ1 Phase focus: Which phase of the software development process is the main focus of the existing studies?

RQ2 System focus: What kind of systems do these studies mainly focus on?

RQ3 Contribution/research quality: What is the quality of the research and contributions in each study:

1. **Research type:** What kind of a research study is this?
2. **Contribution facet:** What is the form of the main contribution?
3. **Evaluation:** What kind of evaluation is made to evaluate the proposed contribution?

We answered these questions by conducting this systematic literature review and following the review protocol described earlier in the paper. The results and conduct of the review are discussed in the following sections.

3.2. Sources of Information

To have the broadest set of papers possible, we searched the most popular literature databases in the field. These are databases used often by the researchers in the fielduse. Here is the list of the digital databases searched in our study:

1. ISI Web of Knowledge (ISI)[1]
2. Compendex (CPX)[2]
3. IEEE Xplore (IEEE)[3]
4. ACM Digital Library (ACM)[4]: this was searched partially to validate completeness of our selected results.

These databases cover a majority of journal and conference papers published in the field of software engineering. Technical reports and other documents with no peer review assessment were excluded from the study due to difficulty of assessing the results and to the large volume of results obtained if these were to be included.

After performing the search, we removed and resolved the overlaps and duplicates from different sources manually. The first three databases were fully searched while only 10% of results from ACM digital library ranked as most relevant, by the database search engine, were searched to check the completeness of the set of results. If the results here would show a significant number of relevant results had not already been included, the search would be expanded.

3.3. Search Criteria

To search in the databases the following search criterion was used on title, abstract and keywords fields:

((robust OR robustness) AND software)

It is a common practice and guideline to set a start date for systematic reviews [48, 155, 5]. We set 1990 as the start date since the formal definition of robustness in software engineering introduced by the IEEE standard glossary of Software Engineering [1] was introduced in that year. Another reason for selecting that date is that the important work in the field published

[1] www.isiknowledge.com

[2] www.engineeringvillage2.com

[3] ieeexplore.ieee.org

[4] portal.acm.org

before this date, such as [76, 23], have been referenced and further developed and used in studies published after 1990. Another criteria for the included studies was that only papers in English were included.

Depending on the functionality provided by the databases search engines, we excluded the results outside the field of software and computers before the studies were further processed. This was due to the wide use of the term *robustness* in other unrelated fields such as mechanics and agriculture. Since the papers that are related to both software and one of these fields are also sorted under the software category, excluding these results would not exclude any results of interest for this study.

Applying these criteria on the three initial databases gave 9193 non-unique results. Due to the large number of excluded papers based on titles, the results were exported to our reference management program after the title exclusion step. Therefore, the number of unique initial results is not known to us and we can only present the accumulative number of hits from the three databases.

Search results in the databases were all sorted according to relevance of papers. According to our observations, this criterion proved to be very accurate for our study since the number of included results dropped very fast when they were classified as less relevant by the databases.

After sorting the results from the previous search engines, the ACM digital library was searched and 3658 papers were found using the same search terms. In the top 350 most relevant papers, no new papers with a relevant title were found that were not already included in our final database. This suggested that we have already covered the most relevant results in the field.

3.4. Study Selection

For selection of studies we developed exclusion/inclusion criteria as described in this section. The initial hits were filtered and excluded in several steps which will be explained below.

Exclusion criteria based on title:

- The paper discusses a different field than software, i.e. the title does not suggest that there is any focus on software in that study.
- Short papers under 5 pages were excluded.
- Robustness is used as an adjective to describe something unrelated to software and it is clear from the title that software robustness is not discussed in this paper. e.g. “Robust Resource Management for meta computers” where robust is used as an adjective to describe resource management rather than the software.

Exclusion criteria based on abstract:

- If robustness is used in another meaning or the paper just claims that the software, method or contribution they have made is robust without any focus on how and why they have made it robust, then the paper is excluded.
- If the paper has a completely different focus than software robustness or software quality. If this has not been detected during the title phase, then it is excluded in the abstract review phase.

For instance, in some articles robustness describes the proposed method or a part of the development process rather than the software itself. These cases were usually detected and excluded during title or abstract phase. A common example was robust watermarks within software security that appeared very often but was out of the scope of our study although it had the word robust and was in the field of software.

Exclusion criteria based on full text were the same as criteria based on abstract. In some cases the product is claimed to be robust and judging whether there is any focus on software robustness is not possible from the abstract. These cases were handled on the full text level and studies were only included if the process of achieved robustness was presented. However, if robustness was claimed with no elaboration on how and why it was achieved, the paper was excluded from the study on full-text level.

3.5. Data Extraction and Synthesis

As mentioned in Section 3.3 papers that were clearly not related to the field of software robustness, based on the title, were excluded. In total 753 studies were included. Any study that has mentioned robustness and was in the field of software in a meaningful way was included after the first selection phase. After the results from each search were completed, the papers that had passed through the filter were moved to the final database. The cumulative number of unique results after each search in the final database was 282, 561 and 601 which indicates the high number of duplicates between the databases. These numbers are presented in Table 1[5].

Table 1: Initial search results for three databases and a control database

Database	Search date	Total hits	Title filtering	Cumul. unique
ISI	2010-12-31	2264	282	282
CPX	2010-12-31	3976	371	561
IEEE	2010-12-31	2953	148	601
ACM	2010-12-31	3645	-	601
Total	2010-12-31	9193	801	601

In the second phase of data extraction, the abstract of the 601 included studies was reviewed. In this phase, 16 more papers were excluded based on title. These papers had passed through the title exclusion although they should not have. Another 20 papers were excluded for being short papers and 280 papers were excluded based on the abstracts.

The third phase of exclusion was done on full-text level and 134 more studies were excluded, which left 144 studies for the next phase of the review. Statistics on how and when the studies were excluded can be seen in Figure 1.

The final 144 selected papers were then categorized based on the research questions. The categories used for each research question are presented in Table 2. Some of the categories are trivial, for the more complicated facets we were partly inspired by other systematic reviews and mappings.

The *phase focus* facet is divided based on the most common phases of software development. However, we considered design and implementation as the same category since many studies with implementation guidelines and contribution had a main focus on design.

[5] From the 3645 results found in the ACM digital library, the top 350 after ranking by relevance (approximately 10%) were analyzed and compared to the results from the previous databases. No new relevant studies that were not already included could be found. This step was conducted to ensure the completeness of the search.

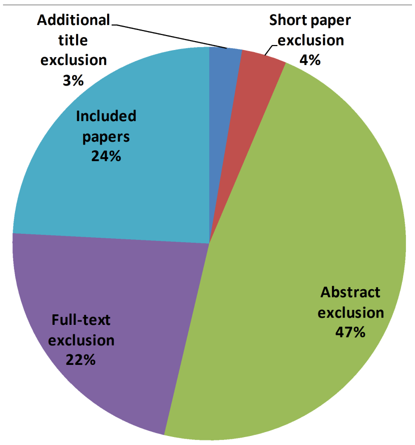

Figure 1: Statistics on data selection in different phases on inclusion/exclusion

The categories in the *system focus* facet were developed as the studies were classified. The category *general* includes the studies that do not have a certain system as focus and are applicable to different types of systems.

The *research type* categories were inspired by [130]. Here, we have combined the categories *Validation Research* and *Solution Proposal* since the distinction is often not easy. We also replaced *Opinion Papers* with *call for research*, *Investigation*.

The *contribution facet* was also partly inspired by [130]. Here, we have introduced the categories *review* for studies that review a certain concept or part of the field. We also added *framework* to replace *process* since most of the studies with a process focus also introduce a tool, method or model to help adopting the process and have a more generic view of the software robustness field. The last category added was *evaluation*. This category includes papers that further evaluate an already published concept and do not introduce any new concepts or solutions.

In the last facet, *Evaluation*, we categorized the studies based on the type of evaluation they have provided for their results. The *Academic lab/toy* category includes studies where the results have been evaluated on small systems developed as case studies for that specific project. Studies in the *large academic* category are also evaluated in an academic setting but on larger and already existing projects that are not specifically developed for a case study and are used more broadly. The *open source* category includes studies evaluated on existing open source systems. The *small industrial* category consists of small case studies developed for evaluation of that specific study in an industrial setting. Finally, studies with *large industrial* evaluation include studies with evaluation on a larger and existing industrial project.

3.5.1. Selection and Extraction Validity

Validity control I

From the 601 papers that had passed the initial title filtering, 60 (10%) were randomly selected and reviewed by author 2. This control was done on abstract level and the goal was to either accept or reject based on the abstract. Full text filtering, classification and categorization of the papers were left to the second validity control.

Table 2: Categories for the research questions

Phase Focus	Requirement, design & implementation, evaluation, analysis, verification and validation (V&V), general
System focus	Web application, distributed and network, real-time/safety critical, COTS, Operating systems, embedded systems, general
Research type	Philosophical paper, solution proposal, evaluation paper, experience report, review, investigation, call for research
Contribution fac.	Tool, method, framework evaluation, metrics, model, review
Evaluation	Academic lab/toy, large academic, open source systems, small industrial, industrial, no evaluation

In these 60 papers there were six deviations between the judgments of the two authors. After studying these deviations two of the excluded papers in the initial selection which were included by the control classification were judged to be included. In the other four cases the initial selection was judged to be correct. The two new papers were eventually excluded in the full text filtering step.

This control suggests that by studying 10% of the results, 10% irregularity was detected. Therefore, the studies were reviewed again by author 1 to detect possible exclusions that could be included or reexamined. The studies that could have been excluded by mistake were reviewed again and if the initial judgment was not correct they were included.

In the next step, another 10% of the studies were screened by author 2 which resulted in no inconsistencies. However, there were five studies that were accepted by author 2 at abstract level that were rejected by author 1 in the full-text level. This means that these studies were accepted by author 1 on abstract level as well.

Validity control II

To check the validity of categorization of results another control was performed. From the 144 accepted papers, 28 papers were randomly selected. Author 2 categorized the selected papers without any communication and access to the rankings of author 1. In this case, there were six deviations on ranking of evaluation of studies. After analyzing the deviations the reason was found to be using different definitions of the categories. Author 1 had ranked studies working on commercial systems in an academic context as *large academic* while author 2 had ranked them as *industrial*. This was discussed and clarified between the two authors and the studies in those categories were once again screened based on the definitions given in this paper. Two other deviations were observed on the phase focus of the studies. Author 1 had ranked them in the category *design* while author 2 had them in *analysis*. This also was found to be due to different definitions of the phase analysis. Clarifying our definitions of this category as presented in the paper, the authors agreed that the initial classification was correct. In three other cases author 2 had ranked the studies in more than one category, but on examining those papers the category selected by author 1 proved to be the main contribution.

3.6. Threats to Validity

The most important threat with this result is the possibility of using other words than robustness by some studies. However, since robustness is a commonly used term, the possibility of any significant contribution not mentioning the word in the title or abstract is minimal. The already large scope of the study would not allow us to expand this study with more search terms. Furthermore, all the results known to us during the years we have worked in the field were included in the final set of studies.

A limitation in this study is that some methods that are mainly used in studies focusing on concepts such as security and defensive programming, which can also be used for robustness assurance and design are not included. This is a common problem for many systematic reviews that deal with broad but important concepts such as robustness. In this study, we decided not to include the papers which are not directly related to robustness to limit the already large scope of the paper. However, future work could also consider techniques that indirectly or only partly has effects on software robustness.

Another threat that needs to be considered when conducting a systematic literature reviews is the possibility of bias in selection. This concern was addressed as described in Section 3.5.1. The selection phase was repeated by author 2 in two phases . The results show a satisfying validity of the screening process.

The last validity threat was the possibility of existence of interesting studies in other fora. However, the databases searched cover the area of software engineering well and we have no reason to think that this does not apply for software robustness. Although the ACM digital library was not fully searched, the search in this database provided no new papers in the top 10% most relevant results. Relevance has been a very accurate measure in all the other databases and we assume that this rule applies to ACM digital library too. This control raises the level of confidence on the coverage of the most important results.

4. Results and Analysis

In this section we present the results of the systematic review. The subsections are structured based on the research questions mentioned in Section 3.1. Section 4.1 presents our findings regarding *RQ1*. This section provides a more detailed overview of the main studies found by this review sorted after the focus area. Section 4.2 gives statistics and overview of the system focus of the included studies to answer *RQ2.* Section 4.3 discusses the quality of the research, contribution and evaluation in the selected studies to answer *RQ3* and its sub-questions. The results in this section are in the form of statistics. The quality of each single study is thereby not presented here. The goal is to give an overview of the quality of studies in the field and identify gaps and weaknesses in the quality of the results.

4.1. Phase Focus of Studies

In this section, we present an overview of available literature in the field of software robustness based on the phase of software development in focus. The different phases represented in this study are analysis, requirements, design & implementation, verification & validation, evaluation and general. While the first four categories are well-established phases of software development, more theoretical papers that evaluate a hypothesis unrelated to a specific phase of software engineering are categorized in the evaluation focus. The *general* category consists of studies with clear contributions that do not have a certain phase focus and are general for the field of software engineering or are valid for all phases of software development.

In each subsection, results with strong contributions are discussed in more detail and an overview of the discussed area is presented. The list of all papers in each subsection can be found in Table 3.

Table 3: Focus of the studies

Focus	Papers	
Verification & Validation	[15] [18] [25] [26] [29] [30] [33] [35] [36] [37] [44] [45] [51] [53] [57] [56] [61] [62] [65] [139] [79] [80] [81] [83] [84] [85] [95] [94] [127] [93] [97] [100] [102] [107] [109] [112] [113] [117] [118] [119] [120] [122] [123] [125] [128] [129] [131] [137] [138] [140] [142] [143] [145] [148] [150] [152] [158] [157] [161] [163] [162] [167] [169] [104] [103] [59] [96] [101]	68
Design & Implementation	[2] [3] [4] [7] [10] [14] [24] [32] [40] [41] [42] [43] [52] [54] [55] [58] [60] [67] [70] [110] [71] [72] [73] [74] [160] [75] [78] [87] [88] [98] [108] [111] [115] [121] [132] [133] [134] [135] [136] [141] [144] [149] [151] [154] [156] [165] [166] [168] [46] [64] [153]	51
Analysis	[6] [9] [17] [27] [28] [31] [63] [69] [105] [126] [164]	11
Evaluation	[8] [12] [16] [19] [39] [86] [114] [159]	8
General	[20] [68] [38]	3
Requirements	[66] [77] [146]	3

An interesting point in this table is the lack of studies in requirements engineering and maintenance. The main areas of *requirements*,*analysis*, *design & implementation*, and *verification & validation*, are discussed in more detail in separate sections. The papers in general and the evaluation category are discussed in section 4.1.5.

4.1.1. Requirements

According to our findings, the extent of published studies on software robustness requirements is very limited. No major study providing methods or tools on how to create requirements to ensure robustness has been found in this study.

In an industrial experience report, Heimdahl and Czerny [66] discuss the importance of completeness in requirements engineering to achieve reliability and robustness, and provide analysis on completeness of a large avionics system. Another contribution with the same focus is [77] that discusses completeness of requirements as the main factor to achieve robustness in a software system. In this study, Jaffe et al. suggests that for a system to be robust there should always be an expected behavior to leave every failure state, even if that results in degrading the functionality of the system. For this purpose, they propose a model that is evaluated in a large academic setting.

Another study with a certain focus on the requirements but main focus on testing is conducted by Nebut et al. [123]. They introduce a technique "with a light declarative formalism to express the mutual dependencies between the use cases, in terms of pre/post conditions (kind of contracts). From these enhanced requirements, a labeled transition system is built to capture all the possible valid sequences of use-cases from an initial configuration" [123]. Here, the goal is to improve the robustness by considered all the possible traces and sequences in the application.

Studies [66, 77] also propose that completeness of robustness requirements is important but there are no further studies that present a systematic way of ensuring the completeness. In general, there are very few studies regarding robustness requirements and there is a gap of knowledge that needed to be addressed. In an attempt to bridge this gap we conducted a study to identify different types of robustness issues [146]. This study presents a framework called ROAST which categorizes these robustness issues into patterns that can be used in the process of requirements elicitation. Furthermore, ROAST suggests different abstraction levels for specification of robustness requirements.

4.1.2. Analysis

Another major phase of software development is analysis. The identified studies in this category focus on how robustness issues can be predicted and prevented early in the development process.

For robustness analysis of web-based systems at an early stage of software development, Calori et al. [28] propose a framework consisting of five steps:

1. Rank the severity of failure scenarios.
2. Capture system behavioral aspects and identify interface objects.
3. Examine potential failures and possible causes and effects.
4. Create a model of the system failure states with probability and severity for nodes and edges.
5. Evaluate the nodes by entering evidence about the state of a variable.

This approach can be used to compare the severities and probability of occurrence of failure scenarios. "The most critical failures can be detected and targeted for prioritized remedial actions. Furthermore, the influence of a preventive action on the system being developed can be estimated. This can represent a powerful tool for design trade-off decisions." [28]

In a study with a more formal approach, [6] presents an abstract interpretation of the LUSTRE language to study propagation of errors and provides an analysis method for checking robustness of LUSTRE functions.

In [98], Kulkarni and Tripathi study robustness in context-aware software. The paper presents a forward recovery model for robustness issues encountered when using their framework for developing context-aware software. They categorize the major failures into service discovery and reconfiguration, service-level binding, service-level exceptions, and context invalidation failures.

On a more theoretical note, Laranjeiro et al. assess the effectiveness of using text classification algorithms such as Support Vector Machines, Large Linear Classification and K-nearest neighbor for identifying robustness issues in web service responses [101]. The results suggest that large linear classification has a high precision in classifying robustness problems.

In another theoretical paper, Cheng-Ying and Yan-Sheng [31] claim that organizing exceptions into hierarchies can be of great benefit to construct robust code. In this study they create a model for exception analysis to improve robustness of the system.

Another method for robustness analysis is introduced by [27], which presents RATF, a method for combining robustness analysis and technology forecasting.

Finally, Groot [63] proposes an approach called *degradation studies* for analysis of how system output degrades as a function of degrading system input such as incomplete or incorrect inputs.

4.1.3. Design & Architecture

After verification and validation, with 51 included studies, design and implementation is the largest focus group of the primary studies. One of the most discussed focus areas for design and architecture is wrappers (encapsulation), which is used to mask and prevent the propagation of robustness issues. Due to the large number of studies with focus on wrappers, these studies are discussed in a separate subsection. Furthermore, some of the other contributions in the field of robustness design and implementation are discussed in this section.

For automatically generating interface properties from high-level robustness rules, Acharya et al. [3, 4] propose a framework. They argue that related interfaces have "similar structural elements (such as function parameters, return values on success/failure, and error flags), when specified at a certain abstraction level. The generic rules are translated into concrete properties by a property engine that queries the specification database for interface information and the pattern database for source-code level, programming-language specific information." [3]

In a series of studies Huhns et al. [70, 110, 69, 134, 71, 72] claim that redundancy increases robustness of software. They "describe a methodology based on active, cooperative, and persistent software components, i.e. agents, and show how the methodology produces robust and reusable software."

Papers [41, 42] present a programming language for robust software systems, *Bristlecone*. Bristlecone applications have two components: high-level organization description that is used for recovery from an error to a consistent state and specifies how the applications conceptual operations interact, and a low-level operational description that specifies the sequence of instructions that comprise an individual conceptual operation.

In another study, Hui and Lui [73] argue that to ensure the function of critical services their operation can use but should not be dependent on less critical components. Otherwise, a minor fault can propagate "along complex and implicit dependency chains and bring down the system" [73]. In a theoretical attempt to address this issue, they present dependency algebra, a framework for dependency management in real-time systems. The framework allows comparison of different designs from the perspective of robustness.

Adding error-handling code and design is another popular way to increase the robustness of software. Keane and Ellman have "implemented a high-level language and runtime environment that allow failure-handling strategies to be incorporated into legacy Fortran and C analysis programs while preserving their computational integrity" [87]. Moreover, in a series of studies, Issarny and Benatre [160, 74, 75] investigate the design and implementation of exception handling support for architecture-based development environments.

On the other hand, Gabriel and Goldman [58] argue that achieving robustness by adding explicit exception handlers and error detection code to the program decreases the maintainability and increases the complexity of the system. Instead Gabriel proposes developing context-aware programs to solve the robustness problem. The main robustness issues in context-aware systems are [98]:

1. Failures in context-driven reconfigurations due to inaccessibility of services or services not being in the correct state for performing the reconfiguration actions.
2. Object-binding failures

3. Service-level operational failures
4. Context invalidations due to lack of privilege or unmet condition

Kulkarni and Tripathi [98] present a recovery model for context-aware applications which "consists of mechanisms for asynchronous event handling and synchronous exception handling. Events represent occurrence of a particular state (normal or failure) related to a context-aware application. Exceptions represent a subclass of events that arise synchronously during the execution of an action within a role operation or a reaction." [98].

Another study conducted by Hameed et al. [64] proposes an aspect-oriented approach to separate error discovery and handling from the core functionality. This can limit the cost and time for introducing changes in the system and make the robustness and quality testing of a system cheaper in the long run.

In a theoretical study, Ambriola and Gervasi [10] identify several different quality attributes such as robustness and investigate the effect of factors such as separation, abstraction, composition, replication and resource sharing in architecture on these quality attributes. The result is that replication has a positive effect on robustness and resource sharing has a negative effect.

Another aspect of software robustness is stability against unexpected events in the execution environment. Choi [32] introduces a kernel resource protector which shields the kernel from faults generated by modules. Observing the relationship between modules and resource objects, the protector can detect and resolve misuses of kernel resources by modules.

Wrapper

Wrapping or encapsulating external modules or services is a common design method used to improve robustness. The method aims at filtering the inputs and avoid the propagation of errors in the system.

Schmidt [144] presents the *Wrapper Facade* pattern to encapsulate low-level functions and data structures with object-oriented class interfaces. "Common examples of the Wrapper Facade pattern are class libraries like MFC, ACE, and AWT that encapsulate native OS C APIs, such as sockets, pthreads, or GUI functions" [144]. Schmidt proposes a cohesive approach for wrapping classes in the same pattern. He proposes the following activities for creating cohesive classes:

- Coalesce multiple individual functions into class methods
- Select the level of indirection
- Determine where to handle platform-specific variation

The aim for this encapsulation is partly to avoid robustness problems to happen or propagate.

Multilayer systems are another focus area for using encapsulation to achieve higher robustness. Lu et al. [108] investigate robustness in multi-layered software architectures such as AUTOSAR (AUTomotive Open System ARchitecture) which is the new architecture standard in the automotive industry. Lu et al. maintain that in such systems to minimize the error propagation from one layer to the next can increase the severity of the error. They argue that the quality of these systems relies not only on the correctness of underlying services but also on multilevel properties. To evaluate this argument they develop a "software as a set of wrappers checking multilevel properties at runtime. The wrappers check the correctness of the application that depends on the behavior of the middleware (communication channels between application components) and OS functions (task management and scheduling) despite accidental and design faults that could impair both the control flow and the data flow of the application. They also trigger recovery actions." [108]

Another area of focus for robustness design is programming libraries. Most common libraries are designed for reuse and focus on flexibility, neglecting robustness requirements [55]. Frick et al. [55, 54] investigate the trade-off between flexibility and robustness. They present an object-oriented library of algorithms and data structures with focus on robustness and flexibility called KARLA. In a similar study, De Vale and Koopman [43] present a safe/fast I/O library with higher safety and robustness than standard I/O libraries which has the same performance. The robustness of this library is evaluated using Ballista robustness testing tool.

In another attempt to increase the robustness of C libraries, Fetzer and Zhen [52] present the HEALERS system. Using header files and manual pages, HEALERS automatically generates a wrapper for each global function that performs argument checking before invoking C library functions. HEALERS has also been used for robustness and security hardening of COTS in [154]

Other interesting results using wrappers to increase robustness can be found in [78, 24, 40, 141, 149, 7, 46].

4.1.4. Verification and Validation

With 68 studies, robustness verification and validation (V&V) is the largest focus group in software robustness phases. The main technique used in this category is testing. This section is divided into three subsections *robustness benchmarks*, *fault injection* and *automated robustness testing*. Automated robustness testing tools almost exclusively use fault injection methods. However, the studies in that category have their main focus on the automated tool, unlike others that use less sophisticated automation mechanics for fault injections and focus more on presenting the technique they use.

Robustness benchmarks

As mentioned above, an important discussion in robustness testing is the different methods to benchmark the level of robustness in a system.

CRASH is the most commonly used metrics for robustness failures as presented in [95]. For grading the severity of robustness vulnerabilities, CRASH uses 5 levels:

C Catastrophic (OS crashes/multiple tasks affected)

R Restart (task/process hangs, requiring restart)

A Abort (task/process aborts, e.g. segmentation violation)

S Silent (no error code returned when one should be)

H Hindering (incorrect error code returned)

In another study, Dingman [44] uses the returned messages from an aerospace system under robustness test to create measurements and benchmarks for robustness failures. The results are very similar to CRASH.

Siewiorek et al. [152] present another benchmark which classifies robustness failure reasons into four categories: omissions, timing, value or state of Response, and crash. Another study providing robustness benchmarks is [26], which presents an experience report of developing a benchmark to measure system robustness. These metrics are used to evaluate robustness of C libraries. Robustness failures were mostly found in the following parts of the libraries: task handling, numeric manipulations, I0 and System protection mechanisms. A CRASH-like benchmark was also presented for failure classification in this study.

On a different level, Mukherjee and Siewiorek [122] divide existing robustness benchmarks into the following categories:

- Crashme: random data against system interfaces.
- Modular: Regard the system as isolated modules and check the robustness of each module.
- Hierarchical: decompose the software based on features and test robustness of each feature.

They also propose a hierarchical benchmark based on features and test it on C++ applications [122].

Some other robustness benchmarks included in this review can be found in [85, 117].

Fault injection

Software fault injection is the primary method used for robustness testing. Fault injection techniques are classified into the following categories by [169]:

- Software implemented fault injection where faults are injected by means of software.
- Scan-chain (flip-flopping ICs) implemented fault injection where faults are injected to physical system using scan-chains.
- Pin level fault injection where faults are injected to the pins of an IC.
- Fault injection by external disturbance where heavy ion radiation and power disturbance are used for injecting faults by external disturbances.
- Fault injection in emulated systems where faults are injected to an emulated model of the system.

Software robustness testing using fault injection uses exclusively the first and in some cases the last technique.

Voas et al. [164] present an automated software analysis environment called Pisces Safety Net (PSN) which is a part of Whitebox Software Analysis Toolkit that injects faults against hardware and software. Instead of analyzing correctness, this approach examines output behavior and aims to analyze and avoid dangerous and life-threatening outcomes. The goal of PSN is to identify weaknesses in software with catastrophic consequences and locate weaknesses in code. Input to PSN is a program, an operational profile, and a description of unacceptable outcomes. Based on that, PSN returns locations in code with potential weaknesses. PSN requires some manual setups for what part of the code to perform fault injection on and what is an unacceptable output.

Java applications have also been the subject of robustness fault injection tests. Zamli et al. [169] presents SFIT, a fault injection tool to assess the dependability of Java COTS, while Olah and Majzik [126] have developed an Eclipse-based fault injection framework that provides a model-based approach and a graphical user interface to specify both the fault injection experiments and the run-time monitoring of the results for Java applications.

Robustness testing of network protocols is another area of focus. Tsanchi et al. [158] test telecommunication systems' fault tolerance by injecting software faults into the service manager and observing the behavior in the fault manager.

In another study, Chuanming [33] uses a formal test specification language, TTCN, to describe test cases for robustness testing of network protocols. TTCN is also used in [140] to create a robustness test framework consisting of two phases: (1) Creating *increased specification* by considering hazards in the specification model (2) A method to generate robustness test cases in TTCN-3 from the *increased specification* provided in (1).

Fault injection has also been used to test and evaluate robustness of web applications. In [100] an online tool called Wsrbench is presented for evaluation of web services. Another study that focuses on this issue is [145], which provides a framework for testing robustness of web services. For white-box coverage testing of error recovery code in Java web services, Fu et al. [57, 56] use compiler-directed fault injection.

Several studies concerning robustness testing of Commercial-Off-the Shelf (COTS) were found in the review. The interest in using COTS is rapidly increasing due to development of more complex systems and the amount of money and time that can be saved using third-party software. However, there is a need to evaluate the robustness and reliability of these systems before they can be integrated into the solution. Barbosa et al. [15] present a methodology for evaluating robustness and dependability of COTS using fault injection techniques, while Sarbu et al. [142] provides a robustness testing method for testing operating system drivers (COTS OS extensions) using their operational state model.

Finally, Johansson et al. [81] study the impact that varying the time for error injection has on evaluation of software robustness. Using "call blocks (i.e. a distinct sequence of calls made to the driver), the trigger for injection can be used to guide injections into different system states, corresponding to the operations carried out."

Some other studies using fault injection techniques for robustness testing are available in [25, 137, 138, 120, 124]

Automated robustness testing

Automated robustness testing tools are common contributions in the area of software robustness. The main idea of these tools is to use stress testing or fuzz testing to try to crash a system and assess its robustness. Several fault injection studies mentioned earlier use this principle as well, but since their main focus has been on the use of fault injection and not on the resulting tool, they were discussed in the previous part.

The most well-known contribution in robustness testing is the Ballista project. In [93] Koopman describes the Ballista robustness testing tool. Ballista uses random and extreme values of different parameter types against system interfaces to test their robustness and stability against random and extreme values. The results of Ballista testing on operating systems identify significant numbers of robustness failures in many well known operating systems [94, 97, 148]. Ballista-like testing might not be a substitute for other testing activities but it can serve to check the overall quality of software at a low cost due to its scalability and automation. Furthermore, it can estimate the extent of potential problems.

Several studies have used Ballista and the CRASH metrics to evaluate robustness of different kinds of systems. In [127] different operating systems are tested and compared using Ballista and CRASH. Invalid file pointers, NULL file pointers, Invalid buffer pointers, NULL buffer pointers, MININT integers, and MAXINT integers are the most detected robustness problems. Fernsler and Koopman [51] use Ballista exception handling to evaluate robustness of the high-level architecture of run-time infrastructure (RTI) which is a distributed simulation system providing robust exception handling. In another study, Jiantao et al. [80] extend Ballista to test the exception handling robustness of C++ ORB client-side application interfaces. They also provide a simple probing method for eliminating simple cases of robustness failures.

JCrasher [37] is another Ballista-like fuzz testing tool specialized for Java applications. "JCrasher offers several novelties: it transitively analyzes methods, determines the size of each tested methods parameter space, and selects parameter combinations and therefore test cases at random, taking into account the time allocated for testing; it defines heuristics for determining whether a Java exception should be considered as a program bug or whether the JCrasher supplied inputs have violated the codes preconditions" [37].

In other studies, Ghosh et al. [62, 60, 61, 143] wrap "executable application software with an instrumentation layer that can capture, record, perturb, and question all interactions with the operating system. The wrapper is used to return error codes and exceptions from calls to operating system functions. The effect of the failure from the OS call is then assessed. If the system crashes, it is non-robust" [62].

Belli et al. propose a model-based approach to robustness testing [18]. The models consist of event sequence graph and decision tables which are later tweaked by the testing application in order to generate robustness test cases.

Some other automated robustness testing tools identified in this review are presented in [167, 45, 53, 118, 119, 104, 103].

4.1.5. Other Work

There are some other major contributions that could not be classified in any of the above sections. Therefore, we will discuss them separately here.

In a theoretical study, De Vale and Koopman [39] argue that software developers identify two main reasons why software systems are not made robust: performance and practicality. They claim however that, by using automated wrappers and robustness testing techniques many of these problems can be solved. Furthermore, in a case study they claim that Maxion's hypothesis that "developers without specific training on the topic might not fully grasp exceptional conditions seems to hold" [39]. Thereby, they suggest that training developers to use robustness improvement techniques is another effective way of increasing robustness of a software.

Another theoretical paper about robustness is [68] where Henzinger identifies two challenges in embedded systems design: predictability and robustness. In this paper robustness is regarded as a form of continuity since in a robust system the reaction changes slightly if the environment changes slightly, and its execution properties change slightly if the platform changes slightly. This theory is used to create a model of how a robust system should behave and how it can be tested.

In another study, Maxion [114] divides the reason for programs failures into two main categories: logic errors in the code, and exception failures. Exception failures can account for up to 2/3 of system crashes. Then he goes on to test the hypothesis that robustness for exception failures can be improved through the use of dependability cases. "Dependability cases, derived from safety cases, comprise a methodology based on structured taxonomies and memory aids for helping software designers think about and improve exception-handling coverage" [114].

Some other important contributions are made by Nebut et al. [123] who present a method that generates robustness tests using requirement contracts, Mendes et al. [116] who propose a method to benchmark the robustness of web servers, and Luo et al. [109] who have developed a method for robustness test generation and execution using input syntax and interaction scenarios.

4.2. System Focus

Table 4 shows the primary studies categorized based on their main system focus. The main categories found were commercial-off-the-shelves (COTS), distributed & network systems, embedded systems, operating systems, real time & safety critical systems and web applications. There were some results that focused on other types of systems than the ones mentioned above. These are listed as *other*. There also exists a category *general* which includes studies that do not have any special kind of system in focus and their results can be applied to many different contexts and systems.

Table 4: System focus of the studies

System focus	Papers	#
General	[2] [4] [10] [17] [18] [19] [24] [26] [27] [29] [31] [37] [39] [41] [42] [43] [52] [54] [55] [58] [69] [70] [71] [75] [79] [80] [83] [84] [86] [87] [88] [109] [111] [113] [114] [120] [121] [122] [123] [126] [131] [132] [38] [134] [135] [136] [140] [141] [144] [151] [152] [156] [164] [166] [168] [104] [103] [59] [64] [146]	60
Other	[9] [40] [63] [110] [72] [74] [16] [25] [45] [78] [160] [98] [117] [158] [159] [167] [96] [153]	18
Operating System	[4] [8] [32] [53] [61] [81] [85] [127] [94] [93] [95] [107] [118] [119] [125] [143] [148]	17
Web Application	[28] [57] [56] [65] [139] [100] [102] [112] [145] [162] [163] [46] [101] [116]	13
COTS	[15] [35] [36] [60] [62] [97] [108] [128] [129] [142] [154] [165] [169]	13
Real-Time / Safety Critical	[20] [30] [44] [66] [67] [73] [77] [115] [133] [157] [161]	11
Embedded System	[6] [14] [68] [105] [137] [138]	6
Distributed & Network	[7] [12] [33] [51] [149] [150]	6

COTS, operating systems and web applications are the categories with most contributions. In the case of COTS the majority of studies focus on finding design or verification methods to ensure robustness of a system using COTS. This matter was discussed in Section 4.1 when the focus area of each study was presented. The same applies to web applications. However, regarding operating systems the main focus is to evaluate and test the robustness of different parts of them, mainly using fuzz testing techniques. Ballista and its extensions are commonly used tools for this purpose.

4.3. Quality of Research/Contribution

This section discusses the quality of the primary studies based on their research type and contribution. The quality is ranked based on several criteria here. The type of research is discussed in 4.3.1. Another criterion is the contribution facet (the type of contribution) which is

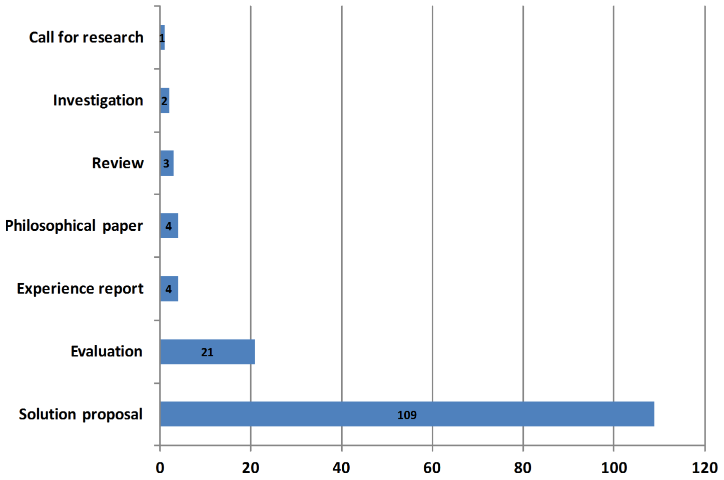

Figure 2: Research type

presented in Section 4.3.2. The last criterion is the type of evaluation performed to evaluate the contributions. This criterion is discussed in Section 4.3.3.

4.3.1. Research Type

Figure 2 shows the statistics on the type of the study in the selected primary studies. Many of the studies conduct several types of research. Therefore, the main contribution of the study was considered for categorization.

Most selected studies had some kind of evaluation which will be discussed in Section 4.3.3. However, evaluation as research type below refers to studies that do one of the following:

1. Evaluate robustness of a system using an existing method or tool
2. Evaluate an already existing method by applying it to different systems

Studies in the review category are the ones that are secondary studies reviewing a field to answer a specific research question. Solution proposals include studies that provide any new contributions or extend already existing ones.

As seen in Figure 2, the absolute majority of the research involves solution proposal or evaluation. The results suggest that there is a need for more overview studies like this one to coordinate and summarize the existing studies in the field.

It is notable that there are three review studies included in this review. The first review is presented in [38] and gives the state of art for development of real-time software where a minor focus is robustness. The second review [151] provides an overview of the use of formal methods in the developing robust and reliable safety-critical systems. The last review [151] compares different techniques for handling incomplete data in databases. One of the criteria for the comparison is robustness. This overview was given to show that studies similar to the one presented in this paper have not been previously done.

Our experience suggests that despite several existing results, the industry usually has problems adopting these results. One way to solve that problem is to perform overview studies like this one to present the academic results to the industry. The other way is to try to understand the

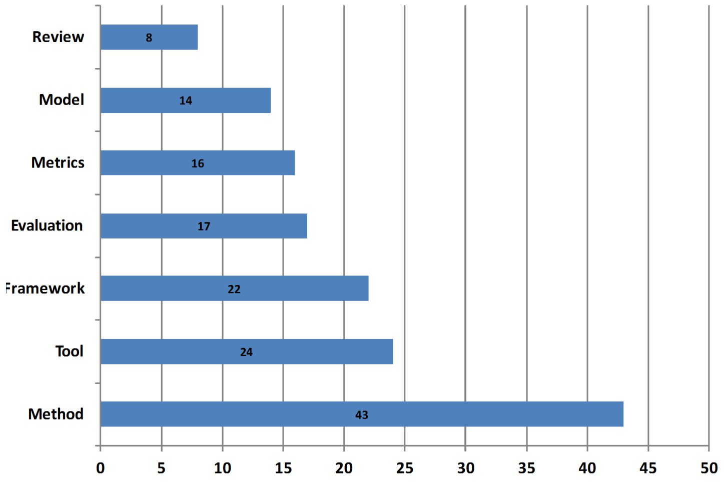

Figure 3: Contribution facet

problems of the industry and find their reasons behind this. This was done in our previous study presented in [147].

4.3.2. Contribution facet

Figure 3 shows a categorization of the studies based on their contributions. Similar to the issue discussed about the research type, most papers had more than one type of contribution. This was addressed in the same way as discussed in Section 4.3.1.

Evaluation had the same definition as the one presented in Section 4.3.1. The reason why there are not the same number of studies of the type evaluation in contribution and research type is that in some cases, although the study focused on evaluating a system or method, the main contribution was a type of metrics, method or tool which was considered more important than the evaluation results itself.

The majority of contributions were in the form of frameworks, methods or models. A framework is a detailed method which has a wide purpose and focuses on several research questions or areas. However, a method usually has a more specific goal and a narrow research question or purpose. A model is different from both the contribution facets mentioned above in the sense that it gives an abstract classification or model of a topic and problem rather than a specific and tangible way of solving a specific problem.

Alongside providing a model, framework, evaluation or method, many studies provided a tool for evaluating their concept. These studies were not classified in the tool category as contribution facet. Only the studies where the tool was the major topic are classified in this category.

Metrics is another type of contribution facet that provides guidelines for how to measure different aspects of robustness.

Figure 3 suggests there is a relatively even distribution in the contribution facets of the studies found. However, there number of reviews are much smaller and there is no systematic review with a more general focus, which further motivates the need for the current study.

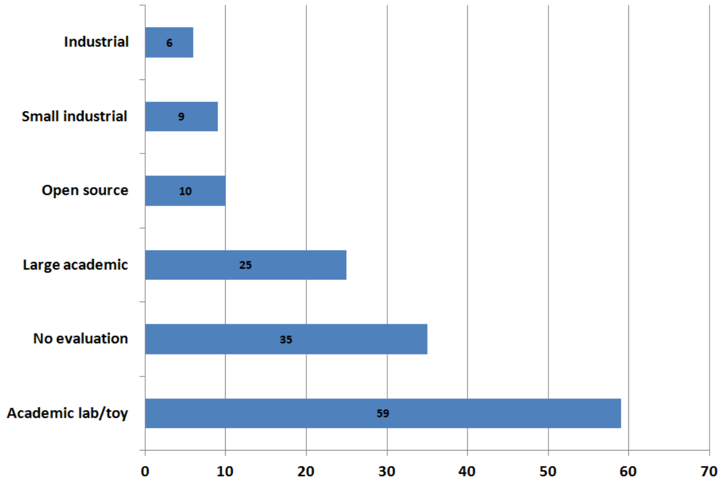

Figure 4: Type of evaluation

4.3.3. Evaluation

One of the important metrics for measuring the strength of academic results is their evaluations. Figure 4 gives statistics on how the primary studies found in this review were evaluated.

Academic lab/toy refers to studies where for the purpose of evaluation a small program has been developed or when a case study on a small commercial system was performed. *Academic OSS* (Open Source System) refers to the studies where the evaluation was done using an open source system. The results of these studies are usually more reliable than the previous category. *Large academic* evaluations refer to the studies where the evaluation is done on large commercial systems or a large number of small systems. These systems can be large commercial products, but if there is no indication of performing action research in an industrial context, the studies were classified in this category. These studies are usually reliable and large enough for proving a hypothesis.

Small industrial evaluation refers to studies where the industrial context is mentioned but the evaluation is done on one or two small projects. The *industry* category includes studies performed in an industrial context which include large projects or more than two small projects. The results of these studies are also strong and typically on the same level as large academic results.

As seen in Figure 4, 65% of the studies either lack evaluation or have a small academic evaluation. From the total of 144 studies, 13% are evaluated in either a small industrial project or a large open source system. The results are considered medium-strong from the evaluation point of view. The remaining 21% of the studies are evaluated either in larger industrial contexts or in large academic projects which typically work on commercial systems. Furthermore, 86% of the studies with strong evaluation focus on verification and validation. These numbers suggest that there are very few overall results in software robustness, especially in areas other than verification and validation.

5. Discussion

This systematic review gives an overview of the field of software robustness. According to the results, we can conclude that the research contributions in some areas of software robustness are very limited. The main gap we identified was the lack of studies on elicitation and specification of robustness requirements. Verification and validation in the form of testing is the largest focus area, followed by design and architecture solutions to improve robustness. Fault injection, automated robustness testing tool, and random interface testing are the main practices used for robustness testing. None of the studies focused on other verification and validation activities than testing.

Almost all the studies focus on robustness issues caused by invalid inputs, and they ignore other aspects of robustness included in the IEEE definition. More complex aspects of robustness that we discuss in [146], such as time out, interrupts, unexpected events, and stressful execution environment are rarely considered in these studies.

Robustness focuses on the states and events that should not happen in a system, rather than how the system should function in ordinary cases. It is cumbersome or even impossible in many systems to create a complete specification of all the possible events and states. Therefore, most academic and industrial projects neglect robustness requirement elicitation and specification. This neglect in many cases results in unawareness of the potential robustness risks among the developers and testers and decreases the overall robustness of the system. As mentioned, it can be uneconomical and even impossible for companies to create a complete requirement specification, which considers all the robustness risks. Nevertheless, the companies should be aware of these risks in a systematic manner, and consider specifying the most crucial risks with the largest potential negative impact.

Most identified design and architecture studies focus on interface wrappers that encapsulate external component interfaces from the rest of the system. This method is mainly used when working with COTS or third-party applications and services. Wrappers filter the input and output data from the external modules. Another popular design method to achieve robustness is graceful degradation. Since the developers are not always able to predict or intercept robustness issues it can be necessary to degrade the functionality of the system in a controlled manner.

The majority of the published studies on robustness focus on verification and validation of the system in the presence of input with faulty value. Most methods and tools introduced in this regard generate random test cases based on a simple model of the system. Although this method can discover robustness risks in the system, the high level of randomness and lack of structure and traceability with the requirements in most of these studies prevent us from guaranteeing the complete robustness of the system. The reason for this is that in order to prove the robustness of the system a more structured approach considering every possible state of the system should be taken. Randomly testing some parts of the systems can make us more comfortable with the robustness of the system but does not necessarily eliminate all potential robustness risks.

The automated robustness testing methods can only be viewed as complementary to other types of testing. It can not replace unit testing, system testing, overall functionality testing or even testing of other quality attributes. The reason for the popularity of the automated testing methods is the fact that they are to a large extent automated and do not require a large development and testing effort. Although this might be enough for smaller systems, for more complex and safety-critical systems or systems with requirement on high safety and availability a more systematic method is required to ensure or improve robustness.

We can draw many interesting conclusions based on the statistics provided in this paper.

Other than the majority of the paper with a general system focus, there are studies that specifically focus on web applications, COTS and operating systems. Since robustness is an especially important attribute for embedded systems, more studies with this specific system focus can be valuable to the practitioners.

Looking at the quality of studies, the conclusion is that many studies introduce new ideas and solutions to problems regarding software robustness but fail to evaluate their contributions properly and show their validity in larger contexts. Furthermore, given our findings in [147], many of the results are not usable for the industrial projects and remain pure academic contributions. One reason for this is lack of industrial validation and the fact that studies remain in a lab or academic context and never take the step to be evaluated in an industrial setting. Another reason is that academic results tend to be context-specific and hard to generalize. Therefore, they cannot be applied to many industrial situations. The strongest results with strong evaluation found in this review are focused on testing of large systems such as operative systems. These tests are mostly randomly generated test cases based on the structure of the system under test. We recommend the use of statistical and evidence-based methods as described in [11, 91] for design and evaluation of future studies in the field. This will provide more scientific, and repeatable results which are more useful for the industry.

6. Conclusion

This paper presents a review of the state of knowledge in the field of software robustness based on a systematic literature review. In total, we analyzed 9193 primary studies from the three well-known, scientific, digital libraries: ISI Web of Knowledge, IEEE Xplore and Engineering Village (Compendex & Inspec). Another 350 most relevant results were browsed from ACM digital library to ensure the completeness of our search.

A total of 601 papers were chosen based on primary title exclusion. After another title exclusion and abstract and full-text exclusion phases, 144 studies were selected. Based on the research questions, each study was classified based on development phase focus, system focus and quality of the research and evaluation.

The results indicate that in the field of software robustness there are many studies on robustness testing of COTS and operating systems, but very few studies about requirement elicitation and specification of robustness. Fault injection and automated testing tools based on fault injection are the main areas for contributions on robustness testing. The main contributions for improving robustness on the design and architecture level, the second largest area of contributions, focus on the use of wrappers and encapsulation of existing software components. Another finding was that most studies focus on a very narrow definition of robustness. Most studies only consider the invalid input aspect of robustness and neglect other more complex aspects like time outs, interrupts and robustness problems related to the execution environment of the software.

The quality of the studies included in this review varied. In total, 65% of the papers have weak or no evaluation, while only 21% of the contributions are strongly evaluated in large academic or industrial contexts. Therefore, there is a clear need to conduct stronger research in the areas where there is a gap of knowledge or where the existing solutions are not evaluated enough to be useful in industrial contexts.

Finally, we think that there is more research needed on eliciting and specifying robustness requirements. Stronger evaluation, especially industrial evaluation, of the studies is also strongly recommended in the future. Another issue that needs to be addressed is to consider more types of issue that can lead to robustness problems. Today, most of the studies focus on robustness in presence of input with faulty value but areas such as robustness in presence of input with unexpected timing or in presence of stressful environmental conditions has not been research as actively.

7. References

[1] , 1990. IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990.

[2] Abie, H., Savola, R. M., Dattani, I., November 2009. Robust, secure, self-adaptive and resilient messaging middleware for business critical systems. In: 2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns (ComputationWorld 2009). IEEE, Piscataway, NJ, USA, pp. 153–160.

[3] Acharya, M., Sharma, T., Xu, J., Tao, X., September 2006. Effective generation of interface robustness properties for static analysis. In: Proceedings of the 21st IEEE International Conference on Automated Software Engineering. IEEE Computer Society, Los Alamitos, CA, USA, p. 4.

[4] Acharya, M., Tao, X., Jun, X., November 2006. Mining interface specifications for generating checkable robustness properties. In: 2006 17th IEEE International Symposium on Software Reliability Engineering. IEEE Computer Society, Los Alamitos, CA, USA, p. 10.

[5] Afzal, W., Torkar, R., Feldt, R., 2009. A systematic review of search-based testing for non-functional system properties. Information and Software Technology 51 (6), 957 – 976.

[6] Ait-Ameur, Y., Bel, G., Boniol, F., Pairault, S., Wiels, V., 2003. Robustness analysis of avionics embedded systems. In: 2003 ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'03), 11-13 June 2003. Vol. 38 of SIGPLAN Not. (USA). ACM, USA, pp. 123–132.

[7] Al-Khanjari, Z. A., Woodward, M. R., Kutti, N. S., Ramadhan, H., Shibab, K., 2003. Masking errors through software robustness. In: International Conference on Internet Computing - IC'03, 23-26 June 2003. Vol. 2 of International Conference on Internet Computing - IC'03. CSREA Press, USA, USA, pp. 809–817.

[8] Albinet, A., Arlat, J., Fabre, J.-C., jun. 2004. Characterization of the impact of faulty drivers on the robustness of the linux kernel. Dependable Systems and Networks, 2004 International Conference on, 867 – 876.

[9] Allen, J., 2005. Towards robust agent-based dialogue systems. In: 2005 IEEE Workshop on Automatic Speech Recognition and Understanding, 27 Nov.-1 Dec. 2005. IEEE, Piscataway, NJ, USA, p. 4.

[10] Ambriola, V., Gervasi, V., February 1998. Representing structural requirements in software architecture. In: Proceedings of IFIP TC2 WG2.4 Working Conference on Systems Implementation 2000: Languages, Methods and Tools. Chapman &amp; Hall, London, UK, pp. 114–127.

[11] Arcuri, A., Briand, L., 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceeding of the 33rd international conference on Software engineering. ACM, pp. 1–10.

[12] Arunchandar, V., Memon, A. M., 2004. Aspire: automated systematic protocol implementation robustness evaluation. In: Proceedings of the 2004 Australian Software Engineering Conference, 13-16 April 2004. IEEE Computer Society, Los Alamitos, CA, USA, pp. 241–250.

[13] Avizienis, A., Laprie, J., Randell, B., 2001. Fundamental Concepts of Dependability. Tech. Rep. 1145, University of Newcastle.

[14] Bak, S., Chivukula, D., Adekunle, O., Sun, M., Caccamo, M., Sha, L., apr. 2009. The system-level simplex architecture for improved real-time embedded system safety. Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE, 99–107.

[15] Barbosa, R., Silva, N., Duraes, J., Madeira, H., 2007. Verification and Validation of (Real Time) COTS Products using Fault Injection Techniques. In: The 6th International IEEE Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems, 2007. ICCBSS '07. pp. 233–242.

[16] Baudry, B., Le Traon, Y., Jezequel, J. M., 2001. Robustness and diagnosability of OO systems designed by contracts. In: Proceedings of the 7th International Software Metrics Symposium. METRICS 2001, 4-6 April 2001. IEEE Computer Society, Los Alamitos, CA, USA, pp. 272–284.

[17] Belcastro, C., Chang, B.-C., 2002. Uncertainty modeling for robustness analysis of failure detection and accommodation systems. Proceedings of the 2002 American Control Conference 6, 4776–4782.

[18] Belli, F., Hollmann, A., Wong, W. E., 2010. Towards scalable robustness testing. In: Secure Software Integration and Reliability Improvement (SSIRI), 2010 4th International Conference on. pp. 208–216.

[19] Bennani, M., Menasce, D., may. 2004. Assessing the robustness of self-managing computer systems under highly variable workloads. Proceedings of the International Conference on Autonomic Computing, 62–69.

[20] Berztiss, A. T., 1994. Safety-critical software: a research agenda. International Journal of Software Engineering and Knowledge Engineering 4, 165–181.

[21] Biolchini, J., Mian, P., Natali, A., Travassos, G., 2005. Systematic review in software engineering. System Engineering and Computer Science Department COPPE/UFRJ, Technical Report ES 679 (05).

[22] Boehm, B., 1978. Characteristics of software quality. North-Holland.

[23] Boehm, B., Brown, J., Lipow, M., 1976. Quantitative evaluation of software quality. In: Proceedings of the 2nd international conference on Software engineering. IEEE Computer Society Press, pp. 592–605.

[24] Brito, P., de Lemos, R., Rubira, C. M. F., 2008. Verification of exception control flows and handlers based on architectural scenarios. In: High Assurance Systems Engineering Symposium, 2008. HASE 2008. 11th IEEE. pp. 177–186.

[25] Brito, P. H. S., de Lemos, R., Martins, E., Moraes, R., Rubira, C. M. F., 2009. Architectural-based validation of fault-tolerant software. In: The 4th Latin-American Symposium on Dependable Computing, 2009. LADC '09. pp. 103–110.

[26] Byung-Hoon, S., Hudak, J., Siewiorek, D., Segall, Z., 1992. Development of a benchmark to measure system robustness: experiences and lessons learned. In: Proceedings of the 3rd International Symposium on Software Reliability Engineering (Cat. No.92TH0486-1), 7-10 Oct. 1992. IEEE Computer Society, Los Alamitos, CA, USA, pp. 237–245.

[27] Calas, G., Boklund, A., Mankefors-Christiernin, S., 2006. A First Draft of RATF: A Method Combining Robustness Analysis and Technology Forecasting. In: Information Technology: New Generations, 2006. ITNG 2006. 3rd International Conference on. pp. 72–77.

[28] Calori, L. C., Stalhane, T., Ziemer, S., 2007. Robustness analysis using FMEA and BBN - Case study for a web-based application. In: WEBIST 2007: Proceedings of the 3rd International Conference on Web Information Systems and Technologies, Vol IT - INTERNET TECHNOLOGY.

[29] Chan, H. A., 2004. Accelerated stress testing for both hardware and software. In: Proceedings of the Annual Reliability and Maintainability Symposium, 26-29 Jan. 2004. Proceedings of the Annual Reliability and Maintainability Symposium. 2004 (IEEE Cat. No.04CH37506C). IEEE, Piscataway, NJ, USA, pp. 346–351.

[30] Chattopadhyay, J., 2006. Methodology to test the robustness of a fault tolerant system to meet realtime requirements. Journal of Aerospace Quality and Reliability 2 (Copyright 2009, The Institution of Engineering and Technology), 81–88.

[31] Cheng-Ying, M., Yan-Sheng, L., 2005. Improving the robustness and reliability of object-oriented programs through exception analysis and testing. In: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems, 2005. ICECCS 2005. pp. 432–439.

[32] Choi, J., 2007. Kernel aware module verification for robust reconfigurable operating system. Journal of Information Science and Engineering 23 (5), 1339–1347.

[33] Chuanming, J., Zhiliang, W., Xia, Y., Jianping, W., 2008. A formal approach to robustness testing of network protocol. In: Network and Parallel Computing. IFIP International Conference, NPC 2008. 18-20 Oct. 2008. Springer-Verlag, Berlin, Germany, pp. 24–37.

[34] Chung, L., do Prado Leite, J., 2009. On non-functional requirements in software engineering. In: Borgida, A., Chaudhri, V., Giorgini, P., Yu, E. (Eds.), Conceptual Modeling: Foundations and Applications. Vol. 5600 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 363–379.

[35] Costa, D., Madeira, H., 1999. Experimental assessment of COTS DBMS robustness under transient faults. In: Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing, 16-17 Dec. 1999. IEEE Computer Society, Los Alamitos, CA, USA, pp. 201–208.

[36] Costa, D., Rilho, T., Madeira, H., 2000. Joint evaluation of performance and robustness of a COTS DBMS through fault-injection. In: Proceedings of the International Conference on Dependable Systems and Networks (includes FTCS-30 30th Annual International Symposium on Fault-Tolerant Computing and DCCA-8), 25-28 June 2000. IEEE Computer Society, Los Alamitos, CA, USA, pp. 251–260.

[37] Csallner, C., Smaragdakis, Y., 2004. JCrasher: an automatic robustness tester for Java. Software-Practice & Experience 34 (11), 1025–1050.

[38] de la Puente, J. A., 1994. Real-time software development: a perspective. In: Proceedings of the 12th Triennial World Congress of the International Federation of Automatic Control, 18-23 July 1993. Pergamon, Oxford, UK, pp. 693–696.

[39] De Vale, J., Koopman, P., 2002. Robust software - no more excuses. In: Proceedings of the International Conference on Dependable Systems and Networks, 23-26 June 2002. IEEE Computer Society, Los Alamitos, CA, USA, pp. 145–154.

[40] Dehnert, J., 2002. Non-controllable choice robustness expressing the controllability of workflow processes. In: Proceedings of the Application and Theory of Petri Nets 2002. 23rd International Conference, ICATPN 2002. 24-30 June 2002. Vol. 2360 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, pp. 121–141.

[41] Demsky, B., Dash, A., 2008. Bristlecone: A language for robust software systems. In: ECOOP 2008 - Object-Oriented Programming. 22nd European Conference, 7-11 July 2008. ECOOP 2008 - Object-Oriented Programming. 22nd European Conference. Springer-Verlag, Berlin, Germany, pp. 490–515.

[42] Demsky, B., Sundaramurthy, S., 2010. Bristlecone: Language support for robust software applications. Software Engineering, IEEE Transactions on PP (99), 1.

[43] DeVale, J., Koopman, P., 2001. Performance evaluation of exception handling in i/o libraries. In: Proceedings of the International Conference on Dependable Systems and Networks, 1-4 July 2001. IEEE Computer Society, Los Alamitos, CA, USA, pp. 519–524.

[44] Dingman, C. P., Marshall, J., Siewiorek, D. P., 1995. Measuring robustness of a fault tolerant aerospace system. In: 25th International Symposium on Fault-Tolerant Computing. Digest of Papers, 27-30 June 1995. IEEE Computer Society, Los Alamitos, CA, USA, pp. 522–527.

[45] Dix, M., Hofmann, H. D., 2002. Automated software robustness testing - static and adaptive test case design methods. In: Proceedings of the 28th Euromicro Conference, 4-6 Sept. 2002. IEEE Computer Society, Los Alamitos, CA, USA, pp. 62–66.

[46] Dolev, S., Gersten, O., 2010. A framework for robust active super tier systems. International Journal on Software Tools for Technology Transfer 12 (Copyright 2011, The Institution of Engineering and Technology), 53–67.

[47] Dybå, T., Dingsoyr, T., 2008. Empirical studies of agile software development: A systematic review. Information and Software Technology 50 (9-10), 833–859.

[48] Dybå, T., Kampenes, V., Sjöberg, D., 2006. A systematic review of statistical power in software engineering experiments. Information and Software Technology 48 (8), 745–755.

[49] Engström, E., Runeson, P., Skoglund, M., 2010. A systematic review on regression test selection techniques. Information and Software Technology 52 (1), 14–30.

[50] Fernandez, J., Mounier, L., Pachon, C., 2005. A model-based approach for robustness testing. Testing of Communicating Systems, 333–348.

[51] Fernsler, K., Koopman, P., 1999. Robustness testing of a distributed simulation backplane. In: Proceedings of the 10th International Symposium on Software Reliability Engineering, 1-4 Nov. 1999. IEEE Computer Society, Los Alamitos, CA, USA, pp. 189–198.

[52] Fetzer, C., Zhen, X., 2002. An automated approach to increasing the robustness of c libraries. In: Proceedings of the International Conference on Dependable Systems and Networks, 23-26 June 2002. IEEE Computer Society, Los Alamitos, CA, USA, pp. 155–164.

[53] Forrester, J. E., Miller, B. P., 2000. An empirical study of the robustness of windows nt applications using random testing. In: Proceedings of the 4th USENIX Windows Systems Symposium, 3-4 Aug. 2000. USENIX Assoc, Berkeley, CA, USA, pp. 59–68.

[54] Frick, A., Goos, G., Neumann, R., Zimmermann, W., 2000. Construction of robust class hierarchies. Software - Practice and Experience 30 (Copyright 2000, IEE), 481–543.

[55] Frick, A., Zimmer, W., Zimmermann, W., 1995. On the design of reliable libraries. In: Proceedings of the 17th International Conference. TOOLS USA '95. Technology of Object-Oriented Systems, 1995. Prentice Hall, Englewood Cliffs, NJ, USA, pp. 13–23.

[56] Fu, C., Milanova, A., Ryder, B. G., Wonnacott, D. G., 2005. Robustness testing of java server applications. IEEE Transactions on Software Engineering 31 (Copyright 2005, IEE), 292–311.

[57] Fu, C., Ryder, B. G., Milanova, A., Wonnacott, D., 2004. Testing of java web services for robustness. In: ACM SIGSOFT International Symposium on Software Testing and Analysis - ISSTA 2004, 11-14 July 2004. Vol. 29 of Softw. Eng. Notes (USA). ACM, USA, pp. 23–34.

[58] Gabriel, R. P., Goldman, R., 2006. Conscientious software. Acm Sigplan Notices 41 (10), 433–450.

[59] Garousi, V., 2010. A genetic algorithm-based stress test requirements generator tool and its empirical evaluation. IEEE Transactions on Software Engineering 36 (Copyright 2010, The Institution of Engineering and Technology), 778–97.

[60] Ghosh, A. K., Schmid, M., 1999. An approach to testing cots software for robustness to operating system exceptions and errors. In: Proceedings of the 10th International Symposium on Software Reliability Engineering, 1999. pp. 166–174.

[61] Ghosh, A. K., Schmid, M., Hill, F., 1999. Wrapping windows NT software for robustness. In: Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing, 15-18 June 1999. IEEE Computer Society, Los Alamitos, CA, USA, pp. 344–347.

[62] Ghosh, A. K., Schmid, M., Shah, V., 1998. Testing the robustness of Windows NT software. In: Proceedings of the 9th International Symposium on Software Reliability Engineering, 4-7 Nov. 1998. IEEE Computer Society, Los Alamitos, CA, USA, pp. 231–235.

[63] Groot, P., Van Harmelen, F., Teije, A. T., 2000. Torture tests: a quantitative analysis for the robustness of knowledge-based systems. In: Proceedings of the 12th International Conference on Knowledge Management, 2-6 Oct. 2000. Vol. 1937 of Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, Germany, pp. 403–418.

[64] Hameed, K., Williams, R., Smith, J., 2010. Separation of fault tolerance and non-functional concerns: Aspect oriented patterns and evaluation. Journal of Software Engineering and Applications 3 (Copyright 2011, The Institution of Engineering and Technology), 303–11.

[65] Hanna, S., Munro, M., 2009. An approach for wsdl-based automated robustness testing of web services. Information Systems Development, 493–504.

[66] Heimdahl, M. P. E., Czerny, B. J., 2000. On the analysis needs when verifying state-based software requirements: an experience report. Science of Computer Programming 36 (1), 65–96.

[67] Henderson, M. I., Gill, K. F., 1996. Design of real-time concurrent software. Mechatronics 6 (Copyright 1996, IEE), 209–225.

[68] Henzinger, T. A., 2008. Two challenges in embedded systems design: predictability and robustness. Philosophical Transactions of the Royal Society London, Series A (Mathematical, Physical and Engineering Sciences) 366 (Copyright 2009, The Institution of Engineering and Technology), 3727–3736.

[69] Holderfield, V. T., Huhns, M. N., 2003. A foundational analysis of software robustness using redundant agent collaboration. In: Agent Technologies, Infrastructures, Tools, and Applications for E-Services. NODe 2002 Agent-Related Workshops. Revised Papers, 7-10 Oct. 2002. Vol. 2592. Springer-Verlag, Berlin, Germany, pp. 355–369.

[70] Huhns, M. N., 2001. Interaction-oriented software development. International Journal of Software Engineering and Knowledge Engineering 11 (3), 259–279.

[71] Huhns, M. N., Holderfield, V. T., 2002. Robust software. IEEE Internet Computing 6 (Copyright 2002, IEE), 80–82.

[72] Huhns, M. N., Holderfield, V. T., Gutierrez, R. L. Z., 2003. Achieving software robustness via large-scale multiagent systems. In: Garcia, A., Lucena, C., Zambonelli, F., Omicini, A., Castro, J. (Eds.), Software Engineering for Large-Scale Multi-Agent Systems - Research Issues and Practical Applications. Vol. 2603 of Lecture Notes in Computer Science. pp. 199–215.

[73] Hui, D., Lui, S., 2006. Dependency algebra: a tool for designing robust real-time systems. In: Proceedings of RTSS. 26th IEEE International Real-Time Systems Symposium, 5-8 Dec. 2005. IEEE Computer Society, Los Alamitos, CA, USA, p. 11.

[74] Issarny, V., 1993. An exception-handling mechanism for parallel object-oriented programming: toward reusable, robust distributed software. Journal of Object-Oriented Programming 6 (Copyright 1993, IEE), 29–40.

[75] Issarny, V., Benatre, J. P., 2001. Architecture-based exception handling. In: Proceedings of the 34th Annual Hawaii International Conference on System Sciences, 2001. p. 10.

[76] Jaffe, M., Leveson, N., May 1989. Completeness, Robustness, And Safety In Real-time Software Requirements Specification. In: Proceedings of 11th International Conference on Software Engineering. pp. 302–311.

[77] Jaffe, M. S., Leveson, N. G., Heimdahl, M. P. E., Melhart, B. E., 1991. Software requirements analysis for real-time process-control systems. IEEE Transactions on Software Engineering 17 (Copyright 1991, IEE), 241–258.

[78] Jhumka, A., Hiller, M., Suri, N., 2002. An approach to specify and test component-based dependable software. In: Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering, 2002. pp. 211–220.

[79] Jiang, S., Yan, D., 2005. Approach to testing java exception handling mechanism quickly. Mini-Micro Systems 26 (Copyright 2006, IEE), 1854–1857.

[80] Jiantao, P., Koopman, P., Yennun, H., Gruber, R., Mimi Ling, J., 2001. Robustness testing and hardening of CORBA ORB implementations. In: Proceedings of the International Conference on Dependable Systems and Networks, 1-4 July 2001. IEEE Computer Society, Los Alamitos, CA, USA, pp. 141–150.

[81] Johansson, A., Suri, N., Murphy, B., 2007. On the impact of injection triggers for os robustness evaluation. In: Software Reliability, 2007. ISSRE '07. The 18th IEEE International Symposium on. pp. 127–126.

[82] Jorgensen, M., Shepperd, M., 2007. A systematic review of software development cost estimation studies. IEEE Transactions on Software Engineering 33 (1), 33–53.

[83] Jwo, J. S., Cheng, H. M., 2001. A robust framework for building java applications. In: Computer Science and Technology in New Century. International Academic Publishers LTD, pp. 506–510.

[84] Kaksonen, R., Laakso, M., Takanen, A., 2001. Software security assessment through specification mutations and fault injection. In: Steinmetz, R., Dittman, J., Steinebach, M. (Eds.), Communications and Multimedia Security Issues of the New Century. Vol. 64 of International Federation for Information Processing. Kluwer Academic Publishers, pp. 174–183.

[85] Kanoun, K., Crouzet, Y., Kalakech, A., Rugina, A. E., Rumeau, P., 2005. Benchmarking the dependability of windows and linux using postmark/spl trade/ workloads. In: Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on. pp. 10–20.

[86] Kashmirian, J., Padgham, L., 2000. Relative robustness: an empirical investigation of behaviour based and plan based paradigms as environmental conditions change. In: Foundations of Intelligent Systems. 12th International Symposium, ISMIS 200, 11-14 Oct. 2000. Vol. 1932 of Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin, Germany, pp. 205–215.

[87] Keane, J., Ellman, T., 1996. Knowledge-based re-engineering of legacy programs for robustness in automated design. In: Proceedings of the 11th Knowledge Based Software Engineering Conference, 25-28 Sept. 1996. IEEE Computer Society, Los Alamitos, CA, USA, pp. 104–113.

[88] Khedri, N., Rahgozar, M., Hashemi, M., 2006. A study on using n-pattern chains of design patterns based on software quality metrics. In: Ardil, C. (Ed.), Proceedings of World Academy of Science, Engineering and Technology. Vol. 14. WORLD ACAD SCI, ENG & TECH-WASET, pp. 354–359.

[89] Kitchenham, B., 2004. Procedures for performing systematic reviews. Keele, UK, Keele University 33.

[90] Kitchenham, B., Charters, S., 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering, Keele University. Tech. Rep. UK EBSE-2007-1, Software Engineering Group, School of Computer Science and Mathematics, Keele University, and Department of Computer Science, University of Durham.

[91] Kitchenham, B., Dyba, T., Jorgensen, M., 2004. Evidence-based software engineering. In: Proceedings of the 26th international conference on software engineering. IEEE Computer Society, pp. 273–281.

[92] Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., Linkman, S., 2009. Systematic literature reviews in software engineering-A systematic literature review. Information and Software Technology 51 (1), 7–15.

[93] Koopman, P., 1999. Toward a scalable method for quantifying aspects of fault tolerance, software assurance, and computer security. In: Proceedings of the Computer Security, Dependability, and Assurance: From Needs to Solutions, 7-9 July 1998 &amp; 11-13 November 1998. IEEE Computer Society, Los Alamitos, CA, USA, pp. 103–131.

[94] Koopman, P., DeVale, J., 2000. The exception handling effectiveness of posix operating systems. Software Engineering, IEEE Transactions on 26 (9), 837–848.

[95] Koopman, P., Sung, J., Dingman, C., Siewiorek, D., Marz, T., 1997. Comparing operating systems using robustness benchmarks. In: Proceedings of the 16th Symposium on Reliable Distributed Systems, 1997. pp. 72–79.

[96] Kovi, A., Micskei, Z., 2010. Robustness testing of standard specifications-based ha middleware. In: Proceedings of 2010 30th International Conference on Distributed Computing Systems Workshops (ICDCS 2010 Workshops), 21-25 June 2010. IEEE Computer Society, Los Alamitos, CA, USA, pp. 302–6.

[97] Kropp, N. P., Koopman, P. J., Siewiorek, D. P., 1998. Automated robustness testing of off-the-shelf software components. In: Proceedings of the 28th International Symposium on Fault Tolerant Computing, 23-25 June 1998. IEEE Computer Society, Los Alamitos, CA, USA, pp. 230–239.

[98] Kulkarni, D., Tripathi, A., 2010. A framework for programming robust context-aware applications. IEEE Transactions on Software Engineering 36 (Copyright 2010, The Institution of Engineering and Technology), 184–197.

[99] Laprie, J., Avizienis, A., Kopetz, H., 1992. Dependability: Basic concepts and terminology. Springer-Verlag New York, Inc. Secaucus, NJ, USA.

[100] Laranjeiro, N., Canelas, S., Vieira, M., 2008. wsrbench: an on-line tool for robustness benchmarking. In: 2008 IEEE International Conference on Services Computing (SCC), 7-11 July 2008. Vol. 2. IEEE, Piscataway, NJ, USA, pp. 187–194.

[101] Laranjeiro, N., Oliveira, R., Vieira, M., 2010. Applying text classification algorithms in web services robustness testing. In: Proceedings of 2010 29th IEEE International Symposium on Reliable Distributed Systems (SRDS), 31 Oct.-3 Nov. 2010. IEEE Computer Society, Los Alamitos, CA, USA, pp. 255–64.

[102] Laranjeiro, N., Vieira, M., 2009. Extending test-driven development for robust web services. In: Dependability, 2009. DEPEND '09. 2nd International Conference on. pp. 122–127.

[103] Lei, B., Li, X., Liu, Z., Morisset, C., Stolz, V., 2010. Robustness testing for software components. Science of Computer Programming 75 (Copyright 2011, The Institution of Engineering and Technology), 879–97.

[104] Lei, B., Liu, Z., Morisset, C., Xuandong, L., 2010. State based robustness testing for components. Electronic Notes in Theoretical Computer Science 260 (Copyright 2010, The Institution of Engineering and Technology), 173–88.

[105] Liggesmerer, P., Rothfelder, M., 1998. System safety improvement by automated software robustness evaluation. In: Proceedings of TCS98: 15th International Conference and Exposition on Testing Computer Software, 8-12 June 1998. ACM, New York, NY, USA, pp. 71–77.

[106] Lisboa, L., Garcia, V., Lucrédio, D., de Almeida, E., de Lemos Meira, S., de Mattos Fortes, R., 2010. A systematic review of domain analysis tools. Information and Software Technology 52 (1), 1–13.

[107] Liu, H., Jiang, J., 2006. A robustness testing platform for file system. High Technology Letters (English Language Edition) 12 (Copyright 2006, The Institution of Engineering and Technology), 23–27.

[108] Lu, C., Fabre, J. C., Killijian, M. O., 2009. Robustness of modular multi-layered software in the automotive domain: a wrapping-based approach. In: 2009 IEEE 14th International Conference on Emerging Technologies &amp; Factory Automation. ETFA 2009, 22-25 Sept. 2009. IEEE, Piscataway, NJ, USA, p. 8.

[109] Luo, X., Ji, W., Chao, L., 2009. TTCN-3 based robustness test generation and automation. In: Proceedings of the 2009 International Conference on Information Technology and Computer Science (ITCS 2009), 25-26 July 2009. Vol. 2. IEEE, Piscataway, NJ, USA, pp. 120–125.

[110] M. N. Huhns, 2000. Agent teams: building and implementing software. IEEE Internet Computing 4 (Copyright 2000, IEE), 93–95.

[111] Majumdar, R., Saha, I., 2009. Symbolic robustness analysis. In: Proceedings of the 2009 30th IEEE Real-Time

Systems Symposium (RTSS 2009), 1-4 Dec. 2009. IEEE, Piscataway, NJ, USA, pp. 355–363.

[112] Martin, E., Basu, S., Tao, X., 2007. WebSob: a tool for robustness testing of Web services. In: 29th International Conference on Software Engineering (ICSE'07 Companion), 20-26 May 2007. IEEE, Piscataway, NJ, USA, pp. 67–68.

[113] Maxion, R. A., deChambeau, A. L., 1995. Dependability at the user interface. In: The 25th International Symposium on Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers. pp. 528–535.

[114] Maxion, R. A., Olszewski, R. T., 1998. Improving software robustness with dependability cases. In: Proceedings of the 28th International Symposium on Fault Tolerant Computing, 23-25 June 1998. IEEE Computer Society, Los Alamitos, CA, USA, pp. 346–355.

[115] Mays, D., Leblanc, R. J., J., 2002. The CycleFree methodology: a simple approach to building reliable, robust, real-time systems. In: Proceedings of the 24th International Conference on Software Engineering. ICSE 2002, 19-25 May 2002. ACM, New York, NY, USA, pp. 567–575.

[116] Mendes, N., Duraes, J., Madeira, H., 2010. Evaluating and comparing the impact of software faults on web servers. In: Dependable Computing Conference (EDCC), 2010 European. pp. 33–42.

[117] Micskei, Z., Majzik, I., Tam, F., 2007. Comparing robustness of AIS-based middleware implementations. In: Service Availability. Proceedings of the 4th International Service Availability. 21-22 May 2007. Vol. 4526 of Lecture Notes in Computer Science. Springer, Berlin, Germany, pp. 20–30.

[118] Miller, B., Koski, D., Lee, C., Maganty, V., Murthy, R., Natarajan, A., Steidl, J., 1995. Fuzz revisited: A re-examination of the reliability of UNIX utilities and services. Tech. rep.

[119] Miller, B. P., Fredriksen, L., So, B., 1990. An empirical study of the reliability of unix utilities. Commun. ACM 33 (12), 32–44.

[120] Moraes, R., Barbosa, R., Duraes, J., Mendes, N., Martins, E., Madeira, H., 2006. Injection of faults at component interfaces and inside the component code: are they equivalent? In: Proceedings of the 6th European Dependable Computing Conference, 18-20 Oct. 2006. IEEE Computer Society, Los Alamitos, CA, USA, p. 10.

[121] Moses, J., Jackson, K., 1991. Ensuring robustness and reliability of object oriented software using mascot 3. In: Proceedings of the 2nd International Conference Reliability and Robustness of Engineering Software II. 22-24 April 1991. Comput. Mech. Publications, Southampton, UK, pp. 19–34.

[122] Mukherjee, A., Siewiorek, D. P., 1997. Measuring software dependability by robustness benchmarking. Software Engineering, IEEE Transactions on 23 (6), 366–378.

[123] Nebut, C., Fleurey, F., Le Traon, Y., Jezequel, J. M., 2003. Requirements by contracts allow automated system testing. In: 14th International Symposium on Software Reliability Engineering, 17-20 Nov. 2003. IEEE Computer Society, Los Alamitos, CA, USA, pp. 85–96.

[124] Neishaburi, M. H., Daneshtalab, M., Kakoee, M. R., Safari, S., 2007. Improving Robustness of Real-Time Operating Systems (RTOS) Services Related to Soft-Errors. In: Computer Systems and Applications, 2007. AICCSA '07. IEEE/ACS International Conference on. pp. 528–534.

[125] Neishaburi, M. H., Kakoee, M. R., Daneshtalab, M., Safari, S., Navabi, Z., 2007. A hw/sw architecture to reduce the effects of soft-errors in real-time operating system services. In: Proceedings of the 2007 IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, 11-13 April 2007. IEEE, Piscataway, NJ, USA, pp. 247–250.

[126] Olah, J., Majzik, I., 2009. A model based framework for specifying and executing fault injection experiments. In: The 4th International Conference on Dependability of Computer Systems, 2009. DepCos-RELCOMEX '09. pp. 107–114.

[127] P. Koopman and J. DeVale, 1999. Comparing the robustness of posix operating systems. In: The 29th Annual International Symposium on Fault-Tolerant Computing, 1999. Digest of Papers. pp. 30–37.

[128] Pan, J., Koopman, P., Siewiorek, D., 1999. A dimensionality model approach to testing and improving software robustness. In: Proceedings of the 1999 IEEE AUTOTESTCON, 30 Aug.-2 Sept. 1999. IEEE, Piscataway, NJ, USA, pp. 493–501.

[129] Pardo, J., Campelo, J. C., Serrano, J. J., 2009. Non-intrusive tool for verifying COTS components in embedded systems. In: 2009 International Conference on Embedded Systems &amp; Applications. ESA 2009, 13-16 July 2009. CSREA Press, Las Vegas, NV, USA, pp. 274–279.

[130] Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M., 2008. Systematic mapping studies in software engineering. In: 12th International Conference on Evaluation and Assessment in Software Engineering. pp. 71–80.

[131] Popovic, M., Kovacevic, J., 2007. A statistical approach to model-based robustness testing. In: 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 2007. ECBS '07. pp. 485–494.

[132] Preston-Thomas, P., Paterson, R., 1991. A technique for improving software robustness to failure. In: ICC 91. International Conference on Communications Conference Record (Cat. No.91CH2984-3), 23-26 June 1991. IEEE, New York, NY, USA, pp. 1159–1163.

[133] Rajanikanth, K. N., Narahari, Y., Prasad, N. N. S. S. R. K., Rao, R. S., 2003. A robust and scalable architecture for

airborne radar simulation. In: IEEE TENCON 2003. Conference on Convergent Technologies for the Asia-Pacific Region, 15-17 Oct. 2003. Vol. 1. Allied Publishers Pvt. Ltd, New Delhi, India, pp. 173–177.

[134] Rajesh, T., Huhns, M. N., 2005. Multiagent reputation management to achieve robust software using redundancy. In: Intelligent Agent Technology, IEEE/WIC/ACM International Conference on. pp. 386–392.

[135] Robertson, P., Williams, B., 2006. Automatic recovery from software failure. Communications of the ACM 49 (Copyright 2006, The Institution of Engineering and Technology), 41–47.

[136] Robillard, M. P., Murphy, G. C., 1999. Analysing exception flow in javatm programs. In: ESEC/FSE'99. 7th European Software Engineering Conference. Held Jointly with 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering, 6-10 Sept. 1999. Vol. 24 of Softw. Eng. Notes (USA). ACM, USA, pp. 322–327.

[137] Rollet, A., Saad-Khorchef, F., 2007. A formal approach to test the robustness of embedded systems using behaviour. In: 2007 5th International Conference on Software Engineering Research, Management and Applications, 20-22 Aug. 2007. IEEE, Piscataway, NJ, USA, pp. 667–674.

[138] Ruiz, J. C., Pardo, J., Campelo, J. C., Gil, P., 2005. On-chip debugging-based fault emulation for robustness evaluation of embedded software components. In: Proceedings of the 11th Pacific Rim International Symposium on Dependable Computing, 2005. p. 8.

[139] S. Hanna and M. Munro, 2008. Fault-based web services testing. In: The 5th International Conference on Information Technology: New Generations, 2008. ITNG 2008. pp. 471–476.

[140] Saad-Khorchef, F., Rollet, A., Castanet, R., 2007. A framework and a tool for robustness testing of communicating software. In: Applied Computing 2007. The 22nd Annual ACM Symposium on Applied Computing, 11-15 March 2007. Vol. 2. ACM, New York, NY, USA, pp. 1461–1466.

[141] Santana, F. G., Gonzalez, J. M., Espino, J. M. S., Calero, J. C. R., 2001. Building robust applications by reusing non-robust legacy software. In: Reliable Software Technologies - Ada-Europe 2001. 6th Ada-Europe International Conference on Reliable Software Technologies, 14-18 May 2001. Vol. 2043 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, pp. 148–159.

[142] Sarbu, C., Johansson, A., Fraikin, F., Suri, N., 2006. Improving robustness testing of COTS OS extensions. In: Penkler, D., Reitenspiess, M., Tam, F. (Eds.), Service Availability. Vol. 4328 of Lecture Notes in Computer Science. Springer-Verlag Berlin, pp. 120–139.

[143] Schmid, M., Ghosh, A., Hill, F., 2000. Techniques for evaluating the robustness of windows nt software. In: Proceedings of DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Vol. 2. pp. 347–360.

[144] Schmidt, D. C., 1999. Wrapper facade: a structural pattern for encapsulated functions within classes. C++ Report 11 (Copyright 1999, IEE), 40–41.

[145] Seung Hak, K., Hyeon Soo, K., 2009. Robustness testing framework for web services composition. In: 2009 IEEE Asia-Pacific Services Computing Conference (APSCC 2009), 7-11 Dec. 2009. IEEE, Piscataway, NJ, USA, pp. 319–324.

[146] Shahrokni, A., Feldt, R., 2010. Towards a Framework for Specifying Software Robustness Requirements Based on Patterns. Requirements Engineering: Foundation for Software Quality, 79–84.

[147] Shahrokni, A., Feldt, R., Petterson, F., Bäck, A., 2009. Robustness verification challenges in automotive telematics software. In: SEKE. pp. 460–465.

[148] Shelton, C. P., Koopman, P., Devale, K., 2000. Robustness testing of the Microsoft Win32 API. In: Proceedings of the International Conference on Dependable Systems and Networks, 25-28 June 2000. IEEE Computer Society, Los Alamitos, CA, USA, pp. 261–270.

[149] Shin, M. E., 2005. Self-healing components in robust software architecture for concurrent and distributed systems. Science of Computer Programming 57 (1), 27–44.

[150] Shu, X., Sheng, L., Xiangrong, W., Lijun, D., 2003. Fault-oriented software robustness assessment for multicast protocols. In: Proceedings of the 2nd IEEE International Symposium on Network Computing and Applications. NCA 2003, 16-18 April 2003. Proceedings of the 2nd IEEE International Symposium on Network Computing and Applications. NCA 2003. IEEE Computer Society, Los Alamitos, CA, USA, pp. 223–230.

[151] Shyamasundar, R. K., 1994. Design of software for safety critical systems. Sadhana 19 (Copyright 1995, IEE), 941–969.

[152] Siewiorek, D. P., Hudak, J. J., Suh, B. H., Segal, Z., 1993. Development of a benchmark to measure system robustness. In: FTCS-23 The 23rd International Symposium on Fault-Tolerant Computing, 22-24 June 1993. IEEE Computer Society, Los Alamitos, CA, USA, pp. 88–97.

[153] Sloan, J., Kesler, D., Kumar, R., Rahimi, A., 2010. A numerical optimization-based methodology for application robustification: Transforming applications for error tolerance. In: 2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 28 June-1 July 2010. IEEE, Piscataway, NJ, USA, pp. 161–70.

[154] Susskraut, M., Fetzer, C., 2007. Robustness and security hardening of COTS software libraries. In: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), 25-28 June 2007. IEEE, Piscataway, NJ, USA, pp. 61–71.

[155] Svahnberg, M., Gorschek, T., Feldt, R., Torkar, R., Saleem, S. B., Shafique, M. U., 2010. A systematic review on strategic release planning models. Information and Software Technology 52 (3), 237–248.

[156] Tao, X., Acharya, M., Thummalapenta, S., Taneja, K., 2008. Improving software reliability and productivity via mining program source code. In: Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on. pp. 1–5.

[157] Tarhini, A., Rollet, A., Fouchal, H., 2004. A pragmatic approach for testing robustness on real-time component based systems. In: Book of Abstracts. ACS/IEEE International Conference on Computer Systems and Applications, 3-6 Jan. 2005. IEEE, Piscataway, NJ, USA, pp. 143–150.

[158] Tsanchi, L., Chi-Ming, C., Horgan, B., Lai, M. Y., Wang, S. Y., 1994. A software fault insertion testing methodology for improving the robustness of telecommunications systems. In: IEEE International Conference on Communications, 1994. ICC '94, SUPERCOMM/ICC '94, Conference Record, 'Serving Humanity Through Communications.'. pp. 1767–1771.

[159] Twala, B., Cartwright, M., Shepperd, M., 2005. Comparison of various methods for handling incomplete data in software engineering databases. In: 2005 International Symposium on Empirical Software Engineering, 17-18 Nov. 2005. IEEE, Piscataway, NJ, USA, p. 10.

[160] V. Issarny, 1992. Exception handling mechanism for parallel object-oriented programming. towards the design of reusable and robust distributed software. Tech. rep., Inst. Nat. Recherche Inf. Autom., Le Chesnay, France.

[161] Verde, L., Amato, F., Canzolino, P., 2000. A software tool for robustness analysis in plant parameters space (roban). Computer-Aided Control System Design, 2000. CACSD 2000. IEEE International Symposium on, 196–201.

[162] Vieira, M., Laranjeiro, N., Madeira, H., 2007. Assessing robustness of web-services infrastructures. In: Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP International Conference on. pp. 131–136.

[163] Vieira, M., Laranjeiro, N., Madeira, H., 2007. Benchmarking the robustness of web services. In: Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium on. pp. 322–329.

[164] Voas, E., Charron, F., McGraw, G., Miller, K., Friedman, M., 1997. Predicting how badly good software can behave. IEEE Software 14 (Copyright 1997, IEE), 73–83.

[165] Voas, J., Miller, K., 1996. Interface robustness for COTS-based systems. In: IEE Colloquium on COTS and Safety Critical Systems (Digest No.1997/103), 28 Jan. 1997. IEE, London, UK, p. 7.

[166] Waydo, S., Dunbar, W. B., Klavins, E., 2003. Toward feedback stabilization of faulty software systems: a case study. In: 42nd IEEE International Conference on Decision and Control, 9-12 Dec. 2003. Vol. 1. IEEE, Piscataway, NJ, USA, pp. 738–743.

[167] Wei Hoo, C., 2007. RPB in Software Testing. In: Computing in the Global Information Technology, 2007. ICCGI 2007. International Multi-Conference on. pp. 8–8.

[168] Yeh, C. H., Parhami, B., Varavrigos, E. A., Varvarigou, T. A., 2001. RACE: a software-based fault tolerance scheme for systematically transforming ordinary algorithms to robust algorithms. In: Proceedings of the 15th International Parallel and Distributed Processing Symposium. p. 6.

[169] Zamli, K. Z., Hassan, M. D. A., Isa, N. A. M., Azizan, S. N., 2006. An automated software fault injection tool for robustness assessment of java cots. In: 2006 International Conference on Computing & Informatics. ICOCI 2006, 6-8 June 2006. IEEE, Piscataway, NJ, USA, p. 6.