IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 23, 2019, accepted March 30, 2019, date of publication April 3, 2019, date of current version April 17, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2909093

Information Flow in Software Testing – An Interview Study With Embedded Software Engineering Practitioners

PER ERIK STRANDBERG[1,2], EDUARD PAUL ENOIU[2], WASIF AFZAL[2], DANIEL SUNDMARK[2], AND ROBERT FELDT[3]

[1]Westermo Network Technologies AB, 721 30 Västerås, Sweden
[2]Mälardalen University, 721 23 Västerås, Sweden
[3]Chalmers University of Technology, 412 96 Gothenburg, Sweden

Corresponding author: Per Erik Strandberg (per.strandberg@westermo.se)

This work was sponsored in part by the Knowledge Foundation under Grant 20150277 (ITS ESS-H), Grant 20160139 (TestMine), Grant 20130085 (TOCSYC), and Grant 20130258 (Volvo Chair), in part by the Swedish Innovation Agency (MegaM@Rt2), Electronic Component Systems for European Leadership (Joint Undertaking under Grant 737494), and in part by the Westermo Network Technologies AB.

ABSTRACT **Background:** In order to make informed decisions, software engineering practitioners need information from testing. However, with the trend of increased automation, there is exponential growth and increased distribution of this information. This paper aims at exploring the information flow in software testing in the domain of embedded systems. **Method:** Data was collected through semi-structured interviews of twelve experienced practitioners with an average work experience of more than fourteen years working at five organizations in the embedded software industry in Sweden. Seventeen hours of audio recordings were transcribed and anonymized into 130 pages of text that was analyzed by means of thematic analysis. **Results:** The flow of information in software testing can be represented as feedback loops involving stakeholders, software artifacts, test equipment, and test results. The six themes that affect the information flow are how organizations conduct testing and trouble shooting, communication, processes, technology, artifacts, and the organization of the company. Seven main challenges for the flow of information in software testing are comprehending the objectives and details of testing; root cause identification; poor feedback; postponed testing; poor artifacts and traceability; poor tools and test infrastructure; and distances. Finally, five proposed approaches for enhancing the flow are: close collaboration between roles; fast feedback; custom test report automation; test results visualization; and the use of suitable tools and frameworks. **Conclusions:** The results indicate that there are many opportunities to improve the flow of information in software testing: a first mitigation step is to better understand the challenges and approaches. Future work is needed to realize this in practice, for example, on how to shorten feedback cycles between roles, as well as how to enhance exploration and visualization of test results.

INDEX TERMS Embedded software development, information flow, interview study, software testing.

I. INTRODUCTION

Software testing includes not only the creation, execution and evaluation of test cases, but also the process of communicating on test cases and their results between human beings, or between humans and machines. This includes face-to-face discussions, writing standards-compliant test reports, reading system logs, as well as making decisions based on test results visualizations. There is a broad industrial interest in the topic of test communication: Industry standards such as ISO/IEC/IEEE 29119-3 [27] prescribe formats for test reports and types of other test documentation. Many agile practices are also prescribing formats for test reporting, such as daily stand-up meetings. In addition, the International Software Testing Qualifications Board (ISTQB) syllabus on test automation prescribe approaches to logging and storing test results [5].

Information flow can be defined by a distributed system made up of agents and the behavioral and structural rules and relationships between them. Information flow is important

The associate editor coordinating the review of this manuscript and approving it for publication was Michele Magno.

when a synergy between humans and software systems is required for a work flow [16]. In the daily work of software developers and testers, there is a need to make many small decisions on what to implement, correct or test next, but also decisions on a project level, e.g. if the software has been tested enough, has a good quality, or can be released anytime soon. Oftentimes, test results are needed to make these decisions, and this information can be seen as flowing between humans and/or software systems.

In the embedded systems domain, software failures in communication equipment can lead to isolation of nodes in a vehicle or a plant, in turn leading to delays, loss of productivity, or even loss of life in extreme cases. The software in embedded systems needs to be of high quality, and software testing is the standard method for detecting shortcomings in quality. An important aspect of testing embedded systems is to conduct some of testing on real hardware [6], [12], [52] and a common approach to support this is to provide a test environment with devices. Cordemans et al. [12] found that, because software uploads to target are slow, a highly iterative testing process such as test-driven development, can be hard to implement, so some testing could be run on the host rather than the target. Furthermore, testing on real hardware opens up for questions on how the test environment has changed since a function was tested last, or how the changed software best reaches the test system.

In this context, an overwhelming majority of software testing conducted in industry is manual – from test creation and execution to evaluation of the test results. Kasurinen et al. found it as high as 90% in a study in Finland in 2010, a practitioner focused report from 2015 by Capgemini, Sogeti and HP, found that only 28% of test cases are automated [28], [44]. Software testing can represent between 30% to 80% of the development cost. Automation can reduce this cost, and also improve time to market [51]. However, challenges for test automation include both lack of time for testing, as well as low availability of the test environment [51]. As automation improves at an organization, the practice of continuous integration becomes relevant. A recent literature study on continuous practices identified that, as the frequency of integrations increase, there is an exponential growth of information used in software development [43]. The same study also identified lack of awareness and transparency in this process as an important challenge.

In addition to testing, requirements engineering is an important part of embedded system development. Requirements represents a major cost driver when developing embedded systems [17], [31]. To ensure functional safety, some embedded systems are developed according to standards such as IEC 61508 or ISO 26262 [24], [26]. The processes and people used for requirements engineering vary widely depending on the application domain and the organization developing and using requirements during development. In many companies, people in different roles (e.g., developers, testers, experts, architects) are involved in this iterative process and have overlapping goals, but also use different perspectives [14].

Standard software engineering tools may facilitate the process of testing, e.g. (i) requirements and traceability management tools such as IBM Doors [1], (ii) Bug/Issue tracking tools such as Mantis Bug Tracker [3], and (iii) build and test results dashboard tools such as Jenkins [2]. However, the flow of information in software testing cannot be seen as a tooling issue alone. Some studies have found that important challenges in software development are related to people and not tools [13], [30]. While studies presenting such problems and solutions exist in the wide scope of software engineering, it is unknown to what extent these tools are contributing to improving the information flow in practice.

A. RESEARCH QUESTIONS

There is a lack of evidence on how software testing can be analyzed from an information flow perspective. The body of knowledge on the flow information in software testing is limited, in particular with regards to industrial evidence. In this paper we explore the information flow and the factors influencing it by interviewing experienced industrial practitioners and analyzing interview transcripts with thematic analysis. The practitioners were sampled from five different organizations developing embedded software in order to get a more diverse set of results.

The following research questions (RQs) were studied:

- **RQ1:** What is the information flow in software testing?
- **RQ2:** What are the key aspects influencing the information flow in software testing?
- **RQ3:** What are the challenges affecting the information flow in software testing?
- **RQ4:** What are the approaches for improving the information flow in software testing?

B. MAIN FINDINGS

Software testing produces information that is non-trivial to manage and communicate. The flow of information in software testing is built up of a number of feedback loops with an origin in software development activities. Geographical, social, cultural, and temporal distances can have an impact on the loops (see section III-B). Six key factors at the organizations developing embedded systems affect the flow of information in software testing: how they conduct testing and trouble shooting, communication, processes, technology, artifacts, as well as how it is organized (see section III-C). There are seven main challenges for the flow of information in software testing: comprehending the objectives and details of testing, root cause identification, poor feedback, postponed testing, poor artifacts and traceability, poor tools and test infrastructure, and distances (see section III-D). Finally, we identified five good approaches for enhancing the flow of information in software testing: close collaboration between roles, fast feedback, custom test report automation, test results visualization, and the use of suitable tools and frameworks (see section III-E).

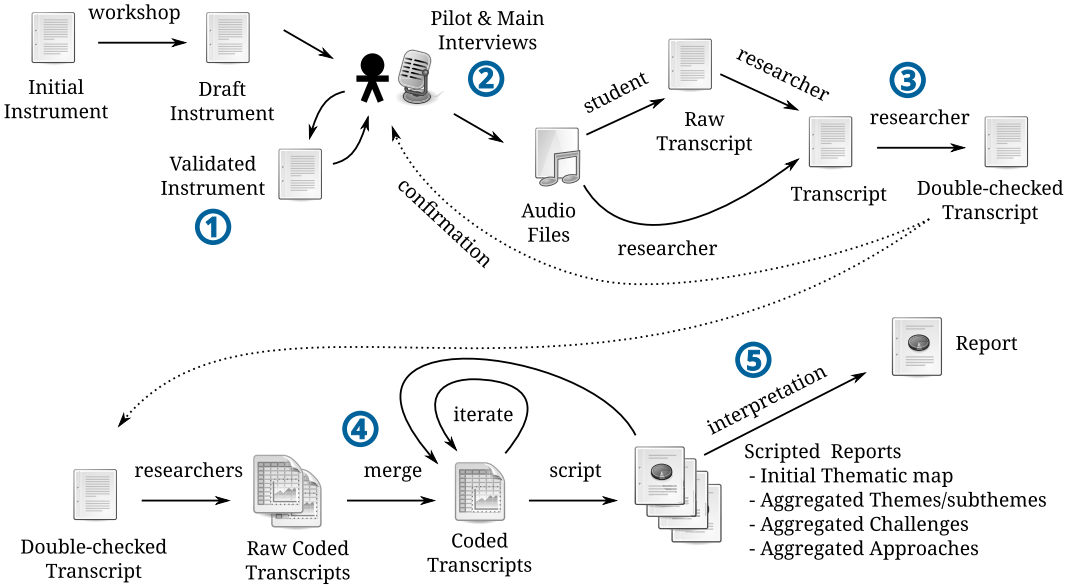

FIGURE 1. Overview of the method followed in this interview study: (1) preparing and validating the instrument, (2) conducting interviews, (3) transcribing interviews, and (4) qualitative analysis with thematic analysis that includes scripting of raw reports which were (5) interpreted when reporting the results.

II. METHOD

This study was conducted by doing face-to-faceå semi-structured interviews, following Linåker et al. [32] as the primary interview guidelines. The interviews were transcribed and then analyzed using Braun and Clarke's guidelines for thematic analysis [11]. The sections below explain the details of our method while Figure 1 gives an overview of it.

A. PREPARATION AND PLANNING

1) ETHICAL AND CONFIDENTIALITY CONCERNS

Based on the ethical guidelines proposed by the Swedish Research Council and the Centre of Research Ethics and Bioethics at Uppsala University [50] we took several steps to ensure we fulfilled ethical responsibilities as researchers. In particular, we designed a process for anonymizing and keeping the obtained data safe, keeping strict restrictions in space and time on the raw interview files, and anonymized the transcripts such that aggregated claims could not be traced back to individual organizations or interviewees. Further, we ensured that informed consent was obtained for all participants.

2) INSTRUMENT

To plan and keep track of the work, a survey plan was written according to the guidelines by Linåker et al. [32]. As the work on the survey progressed, the survey plan was also used as a research diary. A raw survey instrument with 40 questions was created first. In a workshop amongst the authors, we refined it and organized it into initial groups of questions (topics). The topics were ordered to avoid sensitive topics (e.g. asking about misunderstandings and challenges) early on. The start of each interview was planned with a warm-up where we would explain the interview and transcription process. The first topic in the instrument focused on the interviewee (e.g., education background, work experience) while the last topic was related to challenges. In order to understand the flow of information and the workflow at the interviewed organizations, we used generic questions to probe this, before asking specifically about information flow in software testing. We purposely held three pilot interviews. Since we made very few major changes to the instrument after the pilot interviews, two of the three pilot interviews were used in the final study. The third interview did not target the context of embedded system development and therefore was not further used. The final instrument, (1) in Figure 1, covered eight topics, 31 questions and 43 sub-questions, [49]. The questions were open-ended, and most of them had sub-questions to serve as guidelines for the researcher conducting the interview.

B. INTERVIEWS AND TRANSCRIPTION

1) ROLES

We recruited a convenience sample of individuals affiliated with organizations in the embedded system domain. Using a stratified design to ensure experience and specialization diversity, we selected individuals from each of the following groups: developers, testers and managers. We later added the category 'other' for very experienced interviewees that did not fit into any of these three roles. The interviewees were selected from a diverse set of organizations, again by using a convenience sample. We recruited individuals that the authors knew themselves or through one or more contact persons in a given organization, or by searching for suitable individuals in corporate address books.

2) INTERVIEWS

Interviews were conducted face-to-face, and recorded with an off-line digital voice recorder. The interviewees were given a lot of freedom to express their thoughts and explain topics not covered by the instrument. The majority of the interviews were conducted by two researchers where one was the ''driver'' and the other researchers made clarifying questions, kept track of time and made sure no question was forgotten. We got a total of about 17 hours of audio material, (2) in Figure 1. The audio material was not stored on servers or emailed.

3) TRANSCRIPTION

The interviews were transcribed verbatim into 130 dense A4 pages of text, step (3) in Figure 1. However, the interviews that were conducted in Swedish also involved translation to English in the transcription process. Transcription was performed by hand in a text editor, either by the first author or by students having signed an NDA, while listening to the audio file in a media player running at a low speed. During the transcription we ensured anonymization of the transcript. Personal details, names of individuals and products, etc., were all anonymized. Anonymized words were replaced with neutral terms in pointy brackets, such as: <code change>, <vehicle> or <city>. All transcripts were read at least once as a double check, sometimes this involved improvements in transcription and anonymization. Finally, we asked the interviewees if they wanted to review the transcript, and those who wanted to were sent their transcribed interview for confirmation.

C. THEMATIC DATA ANALYSIS

There are many approaches to qualitative data analysis; we decided to follow thematic analysis as described by Braun and Clarke [11]. We saw this approach as comprehensible, suitable for the type of data we had, and it allowed for one sentence of the transcript to be coded as belonging to several themes. The terminology used in thematic analysis partially overlaps with the one in grounded theory (e.g., [47]) and content analysis (e.g., [22]). Important concepts are:

- **Code**: These are ''[lables that] allows the data to be thought about in new and different ways'' [22].
- **Subtheme**: ''Sub-themes are essentially themes-within-a-theme'' [11].
- **Theme**: A theme is a high-level abstraction of, or a ''patterned response or meaning within,'' the data set [11].
- **Theoretical saturation**: is the ''point at which a theory's components are well supported and new data is no longer triggering revisions or reinterpretations of the theory'' [47].

According to Braun and Clarke [11], there are a number of choices one must consider when doing thematic analysis. One choice to make is to define a theme. For us, themes are high level abstractions of the data. Another choice concerns the coding style. As opposed to having a detailed description of parts of the data, we did a broad and rich coding of the interview data. For the actual thematic analysis, we did an inductive or data-driven analysis, aiming at processing the data without trying to make it fit into an existing theoretical framework. We did not look for anything beyond what participants said, making our approach semantic. We were not interested in how meaning and experience are social constructs, instead we aimed at an essentialist/realist approach.

1) THE CODING PROCEDURE

Text coding, step (4) in Figure 1, started with all five authors coding one and the same interview to build consensus on the procedure. During a full day co-located workshop, we discussed the method for the qualitative analysis. We made a number of decisions: (i) to use the Braun and Clarke method, (ii) codes should preferably have four words or less – themes and sub-themes should preferably have fewer, and (iii) as not all sentences in the transcript were on the topic of information flow in software testing, we left the decision to each individual researcher whether to code, or not code, each fragment of an interview. Each remaining interview was coded independently by two authors in different combinations in order to encourage diversity in the coding. Next the first author merged the two codings of each interview into one. Below is a coding example of an interview excerpt:

- **Raw Transcript**: ''[. . .] the test setup fails quite often, there's so much hardware, so like a timeout on a link. . . that will probably generate, you know, like an error.''
- **Code**: (i) test setup fails often, (ii) test environment instability.
- **Subtheme**: (i) test environment, (ii) root cause analysis.
- **Theme**: (i) testing and troubleshooting, (ii) technology, (iii) challenges.

Completed interviews were added into an on-line spreadsheet. A list of themes, subthemes and codes were used to get consistency in the merged interviews. This sheet grew to almost 10 000 rows of interview text, and was later exported as a comma-separated-values (CSV) text file to support partial automated analysis through scripting. About 17.9% of the rows of the spreadsheet were coded. Next followed a period of iterations where the sets of themes and subthemes were reduced. For example, many subthemes on the lines of ''manager tester communication'' and ''developer tester collaboration'' were merged into the more general subtheme ''collaboration between roles''. This period ended with a workshop where the number of themes was reduced to 6. After a few iterations, the initial 357 subthemes were reduced to 86. There were about 1550 unique codes.

Regarding the theoretical saturation we found that the overwhelming majority of the subthemes (84 of 86, or 97.7%) appear in two or more interviews, only two subthemes occur in one, and nine occur in all twelve interviews. According to Guest et al., [23], the discovery of new subthemes is slow after twelve interviews, and according to statistical formulas by Galvin, [19], we should have a 95% confidence level to

have identified subthemes present in 22% or more of the interviewees.

To support the thematic analysis we implemented Python scripts performing raw analyses on the spreadsheet (between step (4) and (5) in Figure 1). The script summarized demographic information, ranked theme and subtheme combinations, extracted interview fragments from all themes and subtheme combinations, and summarized challenges and approaches. The output of this script was a PDF document that quickly grew to almost 2000 pages. Example of how a scripted extraction of interview snippets By iterating over these reports and generating ideas for challenges, a candidate list of twelve challenges was initially generated. This list was reviewed along with reading the raw data. Iterating over this list during a workshop, we arrived at a final list of seven main challenges. Using a similar approach, we arrived at five perceived good approaches from the interviewees. All this resulted in the final report.

III. RESULTS

Given the importance of context in empirical studies [40], we start this section with an overview of the organizations and interviewees. The central parts of this section covers the results of the thematic analysis, and thus the answers to our research questions.

A. ORGANIZATIONS AND INTERVIEWEES

All five organizations develop embedded systems. Three of the organizations work in the domain of safety-critical vehicular systems, one of which do this as a consulting service. The remaining two produce communication equipment. The organizations vary greatly in size and geographic distribution. The largest has more than 50 000 employees over many sites in and outside of Sweden. Two have between 10 000 and 50 000 employees over a few sites in Sweden, Europe, and also in Asia. One has between 100 and 500 employees in a few offices in the same building as well as a second office about an hour by car away from the first. The smallest has between 1 and 5 employees and does consulting at a few different sites. All organization's products and services target an international market. The vehicle organizations develop software according to the V-model, and at least parts of their product development adheres to safety standards. The communication equipment organizations are agile, one follows Scrum, the other Kanban. Both communication organizations have invested heavily in test automation. One of the vehicle companies is undergoing a transition to test automation and has started doing nightly builds and some of their projects have partial automated testing. The second vehicle organization has automation for some test levels, in some projects. The final vehicle organization has very mixed usage of automation. The communication organizations make extensive use of FLOSS (free, libre and open source software) tools for source code management, automatic builds, continuous integration, collaborative code reviews and so on.

We interviewed twelve practitioners, three women and nine men, that we grouped as three developers, four testers, three managers and two 'others'. This division is not trivial as an interviewee was for example a developer specialist, while another was developer of an automated test framework. All of the interviewed women were testers. The interviewees have between 2 and 35 years of experience with an average of 14.4 years. Eight had at least 10 years of work experience. Typical activities for the **developers** were to write, analyze and implement requirements. They also did design, and one lead a group for process-improvements. They could also have an architect role and no longer write code. Three of the **testers** were involved in work on test frameworks, either in a leading role (with responsibilities such as coaching and planning), or with implementation. Testers mentioned that they participate in meetings with test leaders or projects, write test specifications, write test reports, perform manual testing and implement test automation focusing on the test framework part so that the test cases can use the framework. The **project manager** interviewees included a line manager, a project manager and a program manager. They were responsible for teams from 10 to more than 100 members. Their activities were on following up on and leading work on a software release, making sure deadlines are reached, and some had responsibilities on making sure that people are being recruited, developed and are feeling well. Of the **others**, the first had a high-level architect role that moved between development teams. The second was a specialist in functional safety working with safety assessment and requirements, safety management, and education. Functional safety, architecture and requirements are not all in the domain of one given role in an organization, the responsibilities of the different roles had a considerable overlap, in particular with respect to requirements engineering that was the responsibility of several roles.

B. RQ1: WHAT IS THE INFORMATION FLOW IN SOFTWARE TESTING?

In this section we describe the testing process from the perspective of the overall information flow diagram described in Figure 2. This is based on a synthesis of the information provided by the interviewees and represents an abstract and inclusive view of the testing processes used within the studied organizations. In particular, if a test-related information flow has been expressed by any of the interviewees, it is captured in some form in the diagram. This does not necessarily mean that any particular information flow in the diagram is present in all studied organizations. Numbers within parenthesis refer to process steps indicated in the diagram.

1) DEVELOPER TESTING

Software is created, enhanced or corrected by a team of software developers (shown as (1) in Figure 2). Modified software is often tested locally first at the developers desk (2) – either by running unit level or integration level tests, often without target devices, or only a part of a device.

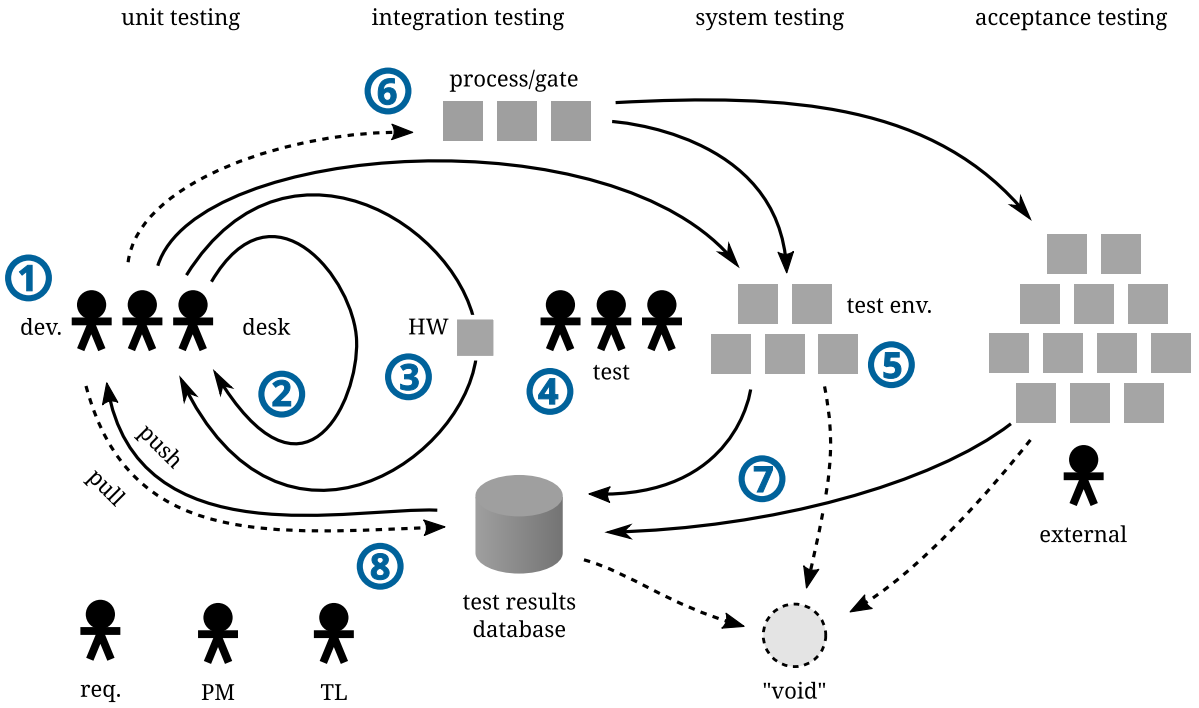

FIGURE 2. The overall information flow diagram for a typical software testing process, as described by our interviewees. The development team (1) produces new software, or software with changes. This is tested in a short and fast loop locally at the developers desk (2), or using some special hardware in a slower feedback loop (3). If the software is tested in a test environment (5) then there might be gates (6) slowing this process down and the need for a dedicated test team (4). Testing produces test results in a test results database (7). Sometimes the test results are pushed (8) back to the developers, sometimes they need to pull to get access to it (again: 8), and oftentimes the test results is never used ("void"). Peripheral, but important, roles are requirements engineers (req.), project managers (PM), and test leads (TL).

This is a fast feedback loop, initiated by the developer, when the developer wants to run it, and with the developer as the recipient of test results.

The software is also commonly tested in target devices (3). Sometimes this is done at the desk of the developers, or in a lab setting at some distance from the developers desk. When done locally, the test setup is typically smaller, perhaps containing one unit or subunit of an embedded system, and not a full system. This is a slower feedback loop than (2), but it is still initiated by the developer and with that developer as the recipient of the test results.

Among the studied organizations, some use simulators that could be used at a developers desk, allowing up to 90% of the system-level tests cases to be executed locally. One organization has test systems built up of network topologies at the desk of most developers, thus allowing them to run sophisticated test cases locally.

2) TESTING BY DEDICATED TESTERS

Software testing is typically divided in levels, where pieces of source code at a low level are tested using unit tests, with the purpose of testing one part of the code in isolation. When units are being tested together we typically talk about integration testing, with the purpose of verifying for example interfaces between units. A next step is system testing, where the whole system is being tested. Here the focus of the testing is to verify that the system as a whole can accomplish its task. During acceptance testing customers or external tester are often involved to validate the software with respect to user needs, typically in a customer setting. When going up in test level more hardware is needed, and greater feedback loops are required. In Figure 2 this is illustrated with feedback loops of increasing size when going to the right of the diagram.

All organizations that were part of this study perform some form of integration- or system-level testing in a dedicated test environment (at (5) in Figure 2), typically in one or several test labs. The test environment can be built up in racks, or on wagons or walls in dedicated rooms. Peripheral hardware to support testing in these labs include voltage generators, signal generators, vehicle displays, or other subsystems. In addition to undertaking manual or automated testing in these labs, dedicated test teams (4) are sometimes responsible for the test environment as well as for the test automation framework.

In order for the testing to commence, the software modified by the development team (1) must reach the test environment (5). The time required for this transition (6) varies considerably among our studied organizations. The slower cases typically involve checkpoints in the testing or development process. One organization did not allow code to reach the main repository, hence blocking it from testing, until the code

change had been reviewed – a process that could take days depending on availability of colleagues. For safety critical software, or when manual testing is done, it is common to bundle many code changes into one test session – because the test sessions require much resources such as manual test effort. This means that testing of a code change will be put on hold until a sufficient number of other code changes have been done. In such cases, the feedback loop can be delayed for months.

In the faster cases, testing typically involves some form of automation that enables test execution without human involvement. Alternatively, testing may be supported by a nightly build solution that compiles the new software, deploys it to the test system, and allows the test team to start testing as soon as they come into the office at the start of business on the work days. One of the studied organizations did automated system level testing where code changes were automatically tested nightly. Here the role of the test team was more focused on developing and maintaining the test framework and the test environment.

3) COMMUNICATION OF TEST RESULTS

Whenever testing is performed, it inherently produces test results (step (7) in Figure 2). In case of a failure, it is common for testers to have a face-to-face dialogue with the developers. This dialogue may lead to the failure being understood and resolved directly, or to the issue being filed in an issue tracker. Some organizations said that they both had a face-to-face conversation and filed an issue. One tester said that a considerable amount of effort (i.e., several hours of work) was sometimes needed before filing the issue – this was done out of fear of filing a false negative issue (i.e., issues that are not related to the software under test).

All vehicle organizations used requirements management tools to store requirements, test cases and the traceability between them. These tools could also store test results in the form of test records – typically an entry in a database saying pass or fail, with a trace to the version of the software being used. From this tool reports could be exported for humans to read. The communication equipment organizations instead used web-based portals where test results could be explored. One of them used plots of trends to visualize aspects of the testing – such as trends over time, or over test system. The other used a portal with many tabs to present results. Both used links to rapidly provide log files from the test framework to give developers the details they needed. Further, one of the vehicle organizations had the habit of gathering the team around visualizations exported from a requirements management tool. However, it should be noted that most of the data generated during testing is never used or looked at, in particular when tests pass. One reason for this is the rapid pace at which new, replacing, test results are generated.

Often a test lead will write a test report and make a high level investigation of all produced results. In those cases where safety-critical software was being developed, a formal test report was always written. For the interviewee in the 'other' category that works with functional safety, the test report was tremendously important and these reports had to strictly follow the domain specific safety standard. In more agile contexts, the role of the test report was questioned. In particular, when a test report was being produced the information in it was typically late, in the sense that it was not useful internally as the results had already reached the developers through other channels. However, several of the interviewees mentioned that the test report could be important for customers.

From our observations, test results are reaching developers (step (8) in Figure 2) when information is pushed back. This happens by allocating an issue directly to a developer (i.e., using an automated system producing notifications) or when a tester is reaching out to the developer. Another way for the test result to reach developers is for the information to be pulled by curiosity, or discipline, in the daily work of a developer. The pulled information is often extracted from an interactive environment (i.e., a portal or a requirement management system used for showing an overview of the information as well as visualizations). The result of a failed test could reach a developer weeks or months after the developer performed the code change that triggered the test execution.

As indicated in Figure 2, test leads, project managers and requirements engineers are important for the software testing process. Their interactions are diverse and occur with many roles in an organization through many contact interfaces. In other cases, customers or other external parties perform testing and have access to the full hardware system, but their test result feedback can be very slow.

C. RQ2: WHAT ARE THE KEY ASPECTS INFLUENCING THE INFORMATION FLOW IN SOFTWARE TESTING?

The thematic analysis of the interview transcripts identified six themes. The themes and their most important subthemes are summarized in Table 1. Many subthemes are important to more than one theme, such as collaboration between roles that is important to both communication and organization. The themes are:

1) **Testing and troubleshooting:** A prerequisite for using and communicating information about test results is testing, and this theme covers how the organizations conduct testing.
2) **Communication:** The flow of information in software testing is directly related to communication, and an important theme in our interviews was of course the ways in which the interviewees communicated with different stakeholders.
3) **Processes:** The theme of processes has an important impact on the flow of information: different development models prescribe different communication patterns, and enforced reviews turn out to be possible bottlenecks for the information flow.

TABLE 1. Overview of the identified themes and the most important subthemes.

Theme	Important subthemes
Testing and troubleshooting	Automated testing, root cause analysis, test environment, test execution, coverage, test selection, duration of test case, test levels, feedback, manual testing, ...
Communication	Collaboration between roles, visual reporting, automated reporting, management of changes, reporting, test reporting aspects, test results database, feedback, distance between teams, ...
Processes	Development model, management of changes, reviews, test levels, ...
Technology	Test environment, complexity, test levels, management of changes, tool support, automated reporting, simulation, test execution, traceability, ...
Artifacts	Adding/removing test case, test specifications, traceability, test design, requirements, safety development, specifications, ...
Organization	Collaboration between roles, team structure, distance between teams, development model, staffing aspects, test levels, misunderstandings, ...

4) **Technology:** In order to test complex embedded systems in the first place, a sophisticated test environment is needed. Another technological aspect of the test-related information flow is how it can be enhanced with tools.
5) **Artifacts:** Requirements, issues, test cases, and also coverage are important aspects of a software development process, and how an organization handles these are of importance to the information flow.
6) **Organization:** Two important organizational aspects that have an impact on the flow of information are team structure and the distances between stakeholders.

In the coming sections we present the themes and the most important subthemes (with the latter boldfaced in the text).

1) THEME 1: TESTING AND TROUBLESHOOTING

Testing and troubleshooting covers aspects of performing testing, regardless of level or degree of automation. We also consider the environment in which the testing is performed, as well as troubleshooting. This was the most important theme in terms of how frequently it appeared in the interviews.

Test execution in the studied organizations encompasses **manual testing** as well as **automated testing**. All studied organizations do some form of **manual testing**. There seems to be three main approaches for this; the most traditional approach is typically done by a tester in a test lab by following a step-by-step test specification, while pushing buttons or setting signals using control panels and interfaces. Organizations that perform testing like this typically store the results as test records in a protocol, a database, or both. The second approach is risk-based exploratory testing, where a risk analysis has been completed to identify the test scope, and manual testing is performed to cover different risks. At one organization this type of testing was conducted, but not really sanctioned. The third approach is manual testing where experience guides the testing, but there is no formal planning, no test specification, and sometimes no artifact produced by testing.

The two organizations producing communication equipment have target hardware infrastructure for **automated testing** in place. One of them use continuous integration, the other runs nightly testing. The vehicle organizations have some test automation in place, but primarily do manual testing. However, even if testing is automated, running a full test suite may still be time consuming: "we have old projects that are fully automatized, or to a very big part. And, if you run them 24/7 then it takes 3 weeks [...]" It should be noted that some organizations that are strong in test automation see automation as a way to free up time and resources for manual testing where it is most needed, as opposed to doing broad manual regression testing.

Troubleshooting, or **root cause analysis**, refers to the process of finding out more about a failure: Is it an issue with the software being developed? Is this function not working on this hardware platform? Are there glitches in the test environment? Are peripheral hardware such as load generators not working as expected? Is the simulator not modeling reality with sufficient accuracy? The most common approach to pinpoint the location of an issue is by **collaboration between roles**. One manager mentions that a team of four roles might be needed to pinpoint where an issue is: "But if there's a more complex functionality problem, then... the tester brings down the developer, and if needed the requirement engineer also. Then they sit together and analyze [... We also] have a test tool team, responsible for the environment. And [sometimes] the test tool responsible engineer is coming also [...]".

2) THEME 2: COMMUNICATION

In the context of this study, the concept of *communication* may include numerous activities and aspects, including oral or written communication between colleagues, how reports are prepared and read, and also how test results are stored. Many interviewees expressed frustration over the test results **feedback** loop duration, regardless of its length. Waiting two months or two hours for test results was perceived as waiting for too long in both cases. Developers may have moved on to new tasks and they were annoyed when receiving negative results. In the cases where the feedback loop is slow because of process steps such as code reviews, some respondents indicated that being pushy towards your colleagues may speed up the loop. Slow feedback can have an impact on quality – a developer waiting for feedback on a code change, may break the implementation without realizing it. Automated testing, nightly testing and continuous

integration facilitate more rapid feedback, but this information may sometimes have troubles reaching the intended recipient: ''. . . we actually do have a large problem with communication [. . .] So I think that's, the main problem actually is to get that feedback, from, the [continuous integration] system, and from our users, to the developer. . . '' A common scenario was to get no feedback, sometimes expressed as ''no news is good news''. However, the interviewees could not know if there were no news because something had not been tested yet, or there were no news because everything was tested and passed.

The most frequently discussed subtheme in this entire study was **collaboration between roles**. Notably, walking over to a colleague to discuss test results is a standard approach in all organizations. This face-to-face communication occurs between most roles. Understanding issues is not always easy, and several interviewees emphasized that it is very important that they are written in such a way that they are understood. Collaboration seems to help locating issues faster, and is thus of great help during root cause analysis. There is often an informal communication between roles before filing an issue. According to one project manager, a tester finding a problem should first talk to a developer to get the details needed, and then file an issue. This way the developer will understand the issue once it reaches him or her. A consequence is also that distance is an important factor.

Visual reporting (i.e., using visualizations to support or replace reporting) was used at most organizations. It was suggested that visualizations supports communication. For example, organizations could use a practice where they gather development teams around a visualization of test results. A common visualization displayed the number of found, open or corrected issues. One vehicle organization used plots showing coverage in terms of system requirements, broken down to show requirements without test cases, as well as requirements with untested, failing, and passed test cases respectively. Most organizations used color codes to separate passing from failing results, here red means fail, and pass could be green or blue. The color yellow or orange was sometimes used – it could mean test not executed, or failure due to shortcomings in the test environment. Aspects with lacking visualization support included non-functional characteristics (such as memory usage), test environment details such as cable connection endpoints in the test system, and various aggregated views such as test results only relevant to a specific functional area, project or customer.

Both communication equipment organizations utilized web-based portals to allow interactive visual exploration of test results. These could display different aspects of testing, provide links to log files, and produce plots to reveal patterns in the test outcomes. In the test results exploration portals, the ability to filter results was deemed important, as well as being able to look at several log files at the same time in some way, and also to show coverage and coverage gaps. However, a test results exploration portal requires time and effort for development and maintenance. A prerequisite for visualizing trends is a **test results database** (also discussed in the theme on artifacts).

3) THEME 3: PROCESSES

The two main **development models** adhered to by the studied organizations are the V-model and/or agile development (according to scrum or kanban). The development of safety critical systems is under rigorous control of standards, and we repeatedly heard that it is important to harmonize the development model with the safety standard to avoid extra work. The testing of a project will get time pressure when the project runs late – in the words of a project manager: ''[T]esters are at the end [. . .] So the money is out, time is out, everything is out.'' A consequent **behavioral aspect** was having to work under stress. Generally, stress was felt to create a bad atmosphere, and lead to reduced test effort and lower quality. It was also suggested that stress may lead to lack of understanding and appreciation of what testers do. One tester said that his colleagues expect him to make tests pass, as opposed to perform testing.

Earlier in the process, **reviews** are typically part of the chain of events that comes after code change, before the updated code reaches the test environment. Code reviews were sometimes required by the process. These reviews could be a bottleneck, limiting the progress of a code change, in particular when other activities had priority over reviewing. In Figure 2 this is indicated with a slow track at (6). For development of safety critical software reviews not only of code, but of all artifacts, are important, and has to be carried out in rigorous ways by an independent person. A notable aspect of reviews is the review team composition. One interviewee mentioned that their organization had taken action against reviews where some colleagues made sure that ''unwanted'' people would not come to a review, in order to make sure the code review would produce a desired outcome. Another interviewee, a tester, wanted to be part of requirements reviews, in order to improve the quality of the requirements: ''[I] said that 'I'd really want to be a part of reviews and [. . .] help you write better requirements.' [But] I never get [invited].''

As visualized in Figure 2, there can be a *push* or a *pull* of test results in the final stage of the feedback loop. In comparison, a pushing system, such an issue tracker sending an email, or sending a text message was preferred over having to request test results. Regarding **decisions and decision support**, experience-based decisions are common. When an internal release is about to be made, the decisions are often informal. Notably, a formal test report was not of central importance to the release process, primarily since such reports were typically late.

4) THEME 4: TECHNOLOGY

Tools are used in the creation, execution and handling of artifacts and are used by developers, test engineers and project managers to coordinate and communicate test results. Thus, interviewees saw **tool support** as an important area

supporting the flow of information and it was mentioned both as an area of improvement, and as an important aspect enhancing the use of test results. Some interviewees gave examples of when tools are helpful for communicating information about testing: tools for traceability, for pushing notifications (such as sending an email when an issue in an issue tracker has been resolved), or for following code changes through the stages in the build and test phase. It is also common to use tools for exporting test results from one format to another for further analysis. However, interviewees felt that too many tools were needed and that tool interoperability was low. Another technological aspect mentioned by interviewees was related to the use of **test environments**. This aspect refers to the equipment used for the execution of tests. Most of the studied organizations used several co-located test environments (e.g., test rigs with several embedded systems, or a software simulation environment on a PC) or globally distributed environments supporting different teams. Test environments occurred in the interviews as a major factor impacting software testing. The interviewees perceived that these environments require a significant effort for configuration. In one case, the interviewee was frustrated with the test environment's instability and unavailability. When these issues do not occur, the availability of a test environment (e.g, developers testing locally using their own machines) speeds up the feedback cycles as well as reduces the number of involved persons. Test environments typically use **simulators** to simulate the environment around the system and its dynamic behavior (e.g., signal and traffic generators). Further, simulators were used when destructive tests are needed to not damage the hardware. One organization used a software test environment to support testing performed by developers at their own desks. These simulators are developed and maintained by a separate team focusing on simulators and test environments. Interviewees mentioned that testing on the target hardware typically results in difficulties in using test automation, which results in a slower feedback loop that might postpone testing: ''And, people don't tend to want to write [...] component tests [...] Because they say [...] 'my component is so special that it must run on hardware' [...]. Which might be true but still, that means that, you need to roll this, component, far in the development chain, until you actually have this particular type of hardware.''

Complexity is a major technological aspect influencing the flow of information, and affects the system under test – including the differences in the supported hardware targets, requirements on forwards or backwards compatibility with respect to both software and hardware combinations. Our interviewees also mentioned that the sheer number of communication protocols and their combinations, system configurations and code dependencies can directly influence software testing. We found evidence that complexity is rarely taken into account when testing and this can cause negative emotions and impact team productivity.

5) THEME 5: ARTIFACTS

Test artifacts of many types are developed in most project phases by different stakeholders, and are used to provide information and identify future actions. These test artifacts (e.g., test strategies, test reports, test cases) and their use or communication are discussed in this section. Organizations deal with **test selection** as a way to select a subset of the available test cases for execution. One interviewee mentioned that test selection seems to be easier when focusing on safety functions than when dealing with regular software, mainly because of the hazard analysis: ''So you can sort of limit the set of, the test space you need to explore, because you know that only a few of these are connected to this hazard [...]'' It seems that good traceability simplifies the impact analysis when changes occur to requirements. The organizations using requirements management tools frequently mentioned that selecting tests for execution was supported through the use of these tools. On the other hand, test selection for covering corrected issues is reported to be more difficult to perform. A common practice is to focus on ''testing the delta'', focusing a limited test effort on a bounded set of changes. One interviewee mentioned that this practice needs special attention when applied to the safety-critical domain: ''This is always a concern, when they go into a maintenance phase, or start working with <issues>, and only test the delta [...] To know that this has no impact on anything else.'' In the **development of safety systems**, impact analysis, code reviews and precise chains of evidence are needed for even the most minor code changes. One experienced interviewee in the domain of safety development mentioned that one of the most important aspects of safety development is to have a development process that is in harmony with the one prescribed by safety standards. Thus, companies in the safety-critical domain use formal standard-compliant test reports documenting as much artifact information as possible.

The lifetime of a test case from its creation to its retirement (i.e., the process of **adding/removing test cases**) may influence how test results are communicated. Several interviewees mentioned that test cases can be added in an *ad hoc* manner or in a structured strategy based on requirements or using risk analysis (e.g., hazard analysis and a fault tree analysis). In the organizations with good traceability between requirements and test cases, the retirement of a test case seems relatively easy; if a test case had its linked requirements removed then there is no reason for the existence of the test case and its results. In two of the studied companies, the interviewees instead mentioned that test cases are rarely removed and that their test scope grows in an uncontrolled way. Many interviewees mentioned that **grouping of test cases for particular areas** was a common way of assigning the responsibility of dealing with these test cases during their lifetime to a certain developer. There was also a need for area-tailored test reports.

Most of the studied companies use different types of **coverage** information, typically based on different artifacts:

requirements coverage, code coverage, risk coverage, hardware coverage, coverage of code changes, or coverage of functional areas just to name a few. Requirements coverage was a very important: " everything is based [on] requirements [...]. That's the starting point, that is the answer." **Requirements** and other types of **specifications** are in general important artifacts for generating information about what and how to perform testing. The two vehicular companies use requirements management tools to keep track of all artifacts created: requirements, test cases, test results and also the traceability information between these artifacts. Interviewees mentioned the use of a system for exporting of test reports containing coverage information, such as the requirements linked with passing test cases, failing test cases and the number of linked test cases created or executed. Respondents also mentioned that in developing safety-critical software code coverage was important, and also required, in order to manage safety certifications.

When exploring test results, **traceability** between artifacts allows a developer to easily obtain more information on the result of a failed test by exploring other types of artifacts, such as revision numbers, log files, test case description, requirements, issues in an issue tracker, details on the test system used for executing the test, and the test case source code. One developer, emphasized the importance of traceability in the following way: "I think it's good that we can connect our issues to the requirements [but] we cannot [connect] our issues to the source code in a good way."

By **reporting** and **test reporting aspects** we both mean making an actual report, and reporting in general. For safety critical development a formal test report in accordance with the relevant safety standard was very important, other organizations saw little value in such test reports. A **test results database** is a good starting point for a report, and our interviewees argued that an automated summary would be excellent, and that most parts of a report should be automated. But in reality this is not the case. Typically a test leader writes a report, and rather few people find value in the test report. In addition: test reports age fast, act as a slow feedback loop, and instead talking face-to-face or using issues were two central ways to communicate around test results.

6) THEME 6: ORGANIZATION

We found that the **team structure** is an important factor impacting the flow of information in software testing. Team design choices usually involve a separation between the development and the test teams. It seems that development teams are typically responsible for unit-level testing in addition to programming and software development. On the other hand, test teams can be categorized into three different types: (i) a test framework team responsible for developing and testing the software used in the test framework, (ii) a team dealing with test equipment such as the test environment, test systems and simulators, and (iii) a traditional test team in which testers are usually performing manual testing. We found that most organizations had a clear separation between the test framework team and the development team(s), and that this separation was clearer in the vehicular organizations. The interviewees involved in safety development had an even stronger focus on clearly separating testing and development roles.

Our interviewees also mentioned **distance between teams** (or distribution of teams) as an important attribute affecting the flow of information in software testing. For example, the organizations had teams working together in different cities, different countries, and even in different continents. It seems that communication in large organizations between teams not located together (even if the distance is relatively short) negatively impacts testing. The interviewees mentioned that the organizational distance and the coordination between large teams can affect the flow of information in software testing. The communication links between team members are scattered and divided throughout the organization: " we often work pretty much independently, so you don't have so much in common with others. And then you don't have the natural group, that you probably should have [in the agile process]." Another interviewee noticed that the use of different development processes in other parts of the organization was confusing. Several interviewees mentioned that a way of overcoming this type of organizational distances is to use different tools and techniques such as filming a test session in the test environment.

Considering the **staffing aspects**, our findings indicate that under-staffing, staff circulation and scarce resources assigned for testing are heavily impacting the flow of information in software testing. Team composition with low-ability profiles and knowledge levels was considered an important negative aspect. In addition, it seems that some organizations are not having teams with enough testers while others are facing the opposite organizational side of having more testers than developers. Some respondents mentioned that many testers do not possess any meaningful education in software testing, and that in some cases developers are working as testers without much test-related knowledge.

Through thematic analysis, we have identified three relevant **misunderstandings** related to the organizational aspects of testing. First, the company background seem to have an impact on how software testing is understood and implemented in the organization: "It's a mechanical company to start with, so [managers], normally they don't understand software at all [...] they're just scared to death about software. It's always late, and not working, and you name it. And then when it comes to testing: 'Just do it [...] how difficult can it be?' " Secondly, interviewees faced the problem that managers do not properly understand software testing and its implications: "I said that 'I don't think we're testing [...] in a good enough way.' And they said that 'well, we are working with the system, we are not working with the source code and testing. That's part of [another department]' [...] people, on top level, they find that testing [...] is not part of the product..." And thirdly, test engineers feel that non-testers don't understand what their work is about: "I don't create

TABLE 2. Overview of challenges and recommendations for practitioners.

Challenge	Recommendations for Practitioners
1. Details of testing	Store all details from the test execution, allow searching, filtering and links between systems.
2. Root cause identification	Make recommendations for where to start fault finding when tests fail.
3. Poor feedback	Speed up feedback loops by testing without hardware when possible. Allow filtering of logs.
4. Postpone testing	Test without hardware when possible, allow informal/exploratory testing.
5. Poor artifacts, poor traceability	Specify requirements on a suitable level. Good traceability allows removal of test scope.
6. Poor tools, poor test infrastructure	When adding a new tool, take interoperability with existing tools into account.
7. Distances	Short distances improve information flow. Some distances can be mitigated with tools.

the green color [...it's] the output of everybody [...] from requirements and through my written specification [...]''

D. RQ3: WHAT ARE THE CHALLENGES AFFECTING THE INFORMATION FLOW IN SOFTWARE TESTING?

This section presents the findings related to seven challenges strongly related with the identified themes and sub-themes (also in Table 2).

1) CHALLENGE 1: COMPREHENDING THE OBJECTIVES AND THE TECHNICAL DETAILS OF A TEST RESULT

When test results are coming back to developers, there is a difficulty to access the particular information and technical details needed to understand this information, e.g. how the test results were created: ''I can extract information [...] but I have no clue about what products we [used], and what roles the different products have had [in the test case].'' It can also be difficult to identity the test steps of the test case, or the source code of the test scripts. Some participants identified that it is hard to find details on revisions of the test framework, test case, simulator software, and the software under test. Due to lack of possibility to extract these details, developers seem to be struggling in how to make the test results actionable. Other participants mentioned that reporting has a focus on what has been tested and leaves the recipient wondering in what order the tests were executed, or what features have not been tested.

Some organizations store the results from manual testing in a test results database, but only when the testing is performed by following manual step-by-step instructions. The use of exploratory testing at one of the organization was incorporated in the development process and they stored these results as part of the risk-based testing approach. All other organizations seem to lose the results from performing exploratory testing: ''[T]he logs for each delivery [are] stored. But not for the manual testing, of course. That's lost.''

2) CHALLENGE 2: UNDERSTANDING THE ORIGIN OF A FAILURE

In a complex test environment the root cause of a failure may be located in the software under test, the test framework, hardware components in the test environment, or be caused by shortcomings in the test case. This challenge is related to issues the participants are facing when understanding faults that could also originate from poor or untestable requirements or from problems in the simulators used. Many interviewees mention that this is a challenge, and some talk about strategies on how to pinpoint the location of an issue (e.g. use different exceptions for different root causes), and how hard it is to perform such actions. A project manager mentioned that this problem is worst in the early stages of software development. One of the developers we interviewed expressed that this challenge is primarily an issue for testers: ''You make these kind of checks. An experienced tester, have already done that before coming to me as an implementer.''

3) CHALLENGE 3: POOR FEEDBACK ON TEST RESULTS

Several interviewees highlighted that poor feedback occurs when the feedback loop is slow. ''When I think I have done a very good job and I have tested my stuff well and the tester comes and says 'I found faults,' I get a little bit angry because I have, I have that behind me [...] and then all of a sudden, months later, someone will show [up and say] 'No, you're not done.' '' In other organizations, a feedback loop requiring hours is also considered slow and seems to affect the communication of test results: ''[A few] hours holding up a test bed to get acceptance testing feedback [...] is too long. When it comes to having a developer keep track of his [code change], someone said 15 minutes [which] would be an acceptable time.''

In one case, poor feedback was related to having too little feedback information available: ''Sometimes you have to go back and see: Did this thing work ever before? What did the log look like? [...] That information gets [garbage collected] after about 1 month [...]''. Equally important, too much information in test reporting can also be a problem, in particular when the information is unstructured or when it cannot be furthered filtered in an overview. None of the interviewees discussed strategies on how to actually perform logging, and it seems that more logging is implemented when needed, in an ad-hoc manner.

Some organizations have implemented test results exploration solutions, but extending or maintaining these is not a priority. These solutions are considered immature and lack important features such as filtering. Thus, a number of participants saw shortcomings related to poor feedback intertwined with the lack of traceability between these tools and the use of several different views and perspectives for searching feedback information.

4) CHALLENGE 4: POSTPONING TESTING TO LATER DEVELOPMENT STAGES

A number of participants saw that postponing testing is a challenge to effective flow of information. If the testing process does not start early, it means that the communication of test results cannot start and the results cannot be transformed into actions. We identified four separate causes for postponing testing: (i) unclear test entry criteria, (ii) tight coupling between hardware and software, (iii) scheduling of testing after software development ends, and (iv) ''the crunch'' occurring when development runs late and testing time needs to be dramatically reduced.

One organization required code reviews prior to the start of testing, but colleagues were not always available for reviews and therefore the start of testing could be postponed. When it comes to the development of safety critical software, participants judged that testing is cumbersome to perform for each code change in isolation – this implies that testing is postponed until several major changes have been implemented. Another participant mentioned that a possible misconception in their organization is that testing can only be performed when the target hardware is available which is a mindset that delays the flow of information and feedback cycles for a certain time.

In practice projects are delayed, and when organizations experience tight deadlines, testers face time pressure: '' [what] you don't trifle with is the development time, but you trifle with test time.'' As a consequence there may be very poor information flow in the later stages of a project.

5) CHALLENGE 5: THE USE OF POOR ARTIFACTS AND TRACEABILITY

Several participants identified that the use of poor requirements specifications in software testing can lead to poor test artifacts: ''We have noticed in a number of instances that because the tester doesn't actually understand why this is happening [because of a lack of a functional description], they write poor test cases, that aren't really executable.'' Some interviewees recognized that too detailed specifications influence the use of artifacts in software testing negatively. In addition to poor requirements, another participant mentioned that test specifications can be of poor and insufficient quality, they are not properly created and not traceable from the test results. It seems that poor traceability can lead to excess work and lost opportunities for effective testing and the flow of information. As already mentioned on Theme 5 Artifacts, an additional aspect is the potentially uncontrolled growth of the test scope when new test cases are added over time. Many of these test cases are added while not taking into account any traceability and coverage analysis. Equally, some engineers are experiencing a certain fear of removing old test cases, which may contribute to the growth of the test scope.

6) CHALLENGE 6: USE OF IMMATURE TEST INFRASTRUCTURE AND TOOLS

Software testing relies on several layers of environments such as test automation frameworks or commercial testing tools. Testing generates large amounts of information, uses numerous software executions, and requires coordination and communication between test engineers, managers and developers. Tools are used in the creation of tests, test execution, test result handling, and test communication. Some study participants have recognized that several test infrastructure and tool issues seem to hinder their testing work. One participant recognized that the test environment instability is disturbing the execution of tests: ''When you reset the units, then it requires some time [...] When you reboot it [...] it has to stabilize, this takes some time [...] and this can disturb the next test.'' A related problem was the test environment saturation that often happened when test cases cannot be scheduled for execution: ''you don't always get testbeds [...] if someone else is testing something with higher priority then we don't get the feedback until later.''

Another participant raised the issue of testing in a simulator as a tool for rapid prototyping and development. Though most of the functionality can be simulated, some hardware and software features are not matching their actual behavior. One participant explained the problems related to simulators as follows: ''The simulators [...] often don't cover the entire reality, [and they] miss certain situations [...] so when you get a fault, and you correct it, then it still won't work in practice because the simulator does not match reality.''

In addition, participants felt that legacy tools that are not well known among, or maintainable by, the colleagues represent a challenge in the testing.

7) CHALLENGE 7: GEOGRAPHICAL, SOCIAL, CULTURAL AND TEMPORAL DISTANCES

The information flow is affected by geographical, social, cultural and temporal distances. These distances are fairly well known and researched in distributed software development [9]. The interviewees expressed concerns that factors related to these distances are making communication more difficult. For example, several participants observed that spoken language affects communication of test results: ''...so when we [say] the English term for something [...] we don't know what it's called in Swedish.'' Another interviewee was concerned about the increase in geographical distance between a development site and a testing site and how this affects the flow of information: ''Then you went 10 meters down the hallway, and then we take their help and then they find the error very fast. When it is <in other city>, it's very hard. Because often it's something that actually is visual [in the test environment].'' Another participant recognized the problem of increasing the organizational distance in terms of the fragmentation of one development team with one kanboard into several isolated teams.

With these increased distances come an increased need for the use of tools to bridge the distances. One participant

TABLE 3. Overview of approaches and their implications for practitioners.

Approach	Recommendations for Practitioners
1. Close collaboration between roles	Encourage collaboration, e.g. tester/req. engineer "pair programming" of requirements.
2. Fast feedback	Test without hardware when possible, allow informal/exploratory testing.
3. Custom test report automation	A test results database can enable automation of test reporting.
4. Test result visualization	Visualizations require a test results database, as well as resources for development and maintenance.
5. Tools and frameworks	Use suitable tools, make sure they can be integrated and/or support linking.

mentioned video-recording test sessions or using a tool to access a test system remotely. However, the use of tools can negatively impact the flow of information. The challenge is that tools may have compatibility distances, may not always work well, and manual traceability between tools is quite common. Sometimes tools are used in ways they are not designed for, such as using an issue tracker as a to-do list manager, because "it's easier for everyone to keep all the things in the same place instead of having fifteen different places to look [in]."

E. RQ4: WHAT ARE THE APPROACHES FOR IMPROVING THE INFORMATION FLOW IN SOFTWARE TESTING?

In this section we present good approaches for improving the information flow in software testing mentioned by the interviewees (summarized in Table 3). Some aspects of them are also discussed in relation to existing research in the discussion section.

1) APPROACH 1: CLOSE COLLABORATION AND COMMUNICATION BETWEEN DIFFERENT ROLES

Verbal communication is perceived as a good way to collaborate and communicate on test results, it also facilitates knowledge transfer between experienced and novice engineers. The participants saw face-to-face communication and close collaboration as central when creating a common understanding of the scope of testing. For one organization in particular, several participants mentioned an exceptional level of collaboration between roles: "[We] have work days when implementation, testing and requirements spend their day together [...] Just running tests and correcting quite fast" Also, in order to make the requirement specification less of an obstacle, one participant suggested that, even if this is not in reality a common occurrence, testers should physically join requirement reviews. Similarly, another participant emphasized that the communication between roles when understanding how requirements should be tested is amplified when done verbally between people sitting close to each other.

2) APPROACH 2: FAST FEEDBACK

The participants recognized that fast feedback is a good approach to improving the flow of information and communication. Having the ability to compile locally, having automated builds, using test automation frameworks, soft test environments, hardware emulators, and testing in a simulator are all methods to get fast feedback and deal with test results in a good and efficient manner. One participant mentioned: "We have this database and [get] data out. You start a script and it takes 10 minutes, and [...] then it's sent out. So it's working extremely well [...]" Also, the use of push notifications through email or text messages can speed up the feedback loops.

3) APPROACH 3: CUSTOM TEST REPORT AUTOMATION

We identified several approaches regarding the automatic creation of test reports and the use of custom reports tailored to different roles. Many participants expressed the view that there are benefits in creating custom reports focusing on specific functional aspects and filtering out the useful information from nightly testing in a summary report. The participants wanted to understand where an error was located and what factors to consider by examining the suggestions given in the test report. Also, in order to make the bug finding less ambiguous, one participant suggested that test reports should be interactive. For example, one can claim a failure for troubleshooting and trace the execution of a certain test to specific errors in the software or the test environment by using a user interface specifically designed for aggregating test results. One participant explained the use of interactive custom reports as follows: "...we have this aggregated view, there's a web-page that looks through the logs when it's executing, and it goes either yellow, green or red. And there's an excerpt, of the error message that triggered the fail. And then you can click on it and get to the logs." Another participant suggested that reports should be automatically generated into documents for a release, and this can be beneficial when used for safety certification.

4) APPROACH 4: TEST RESULT VISUALIZATION

Visualization of test results may serve as an important instrument in understanding the test execution by using different feature and characteristic trends over time as well as a general performance plot in form of a dashboard. One participant mentioned that even the use of colors for visualizing the results of testing can show where the errors manifest and how acute the problems are by representing the severity and the importance of the information using gradients of representative colors. For example, red is used for fail, yellow for inconclusive results. Another participant encouraged the use of an overview of the results that is used to examine in-depth each level of information needed to discover the location of

a fault. These visualizations were thought as a good basis for fostering the communication around diagrams and visual information instead of using textual results which can be difficult to understand.

5) APPROACH 5: USE OF TOOLS AND FRAMEWORKS FOR ENHANCING THE INFORMATION FLOW

The participants stated that for strengthening the flow of information, a better use of tools and frameworks is needed to improve the continuous exchange of information between different tools and people throughout the development process. One participant mentioned that the use of code collaboration tools for reviews and static analysis could help improve the flow of information, while another participant said that tools and artifacts should be linked and test results must be showed inside the IDE, for allowing developers to test in their own sandbox. In addition, recording signal values, and re-running test cases can help in enhancing the flow of information when the SUT is located in another location than the main team(s).

IV. DISCUSSION AND RELATED WORK

In this study we have explored the information flow in software testing by conducting and analyzing interviews with twelve experienced practitioners in the embedded software industry. While much of research and even practice of software testing is focused on improving the testing itself a main conclusion from our study is that large gains can be had by better using and communicating around the testing that already happens. In this section we put these results in context of related work.

The results of the thematic analysis show that software testing produces information that is non-trivial to manage and communicate. The flow of information in software testing is built up of a number of feedback loops initiated by software development activities. As illustrated in Figure 2 distances may have an impact on the loops. Six key factors at the organizations developing embedded systems affect the flow of information in software testing: how testing and trouble shooting is conducted, communication, processes, technology, artifacts, as well as how the companies are organized (see Table 1). There are seven main challenges for the flow of information in software testing: comprehending the objectives and details of testing, root cause identification, poor feedback, postponed testing, poor artifacts and traceability, poor tools and test infrastructure, and distances (see Table 2). Finally, we identified five good approaches for enhancing the flow of information in software testing: close collaboration between roles, fast feedback, custom test report automation, test results visualization, and the use of suitable tools and frameworks (see Table 3).

A. FINDINGS WITH RESPECT TO ESTABLISHED PRACTICES

There are well established practices in industrial software engineering and software testing, described by e.g. Sommerville's book on software engineering [45] and ISTQB syllabi [37]. Much of what we have found overlaps with or builds upon already well established practices. However, in addition to the already mentioned key findings, we have also discovered that: (i) Much of the test results where tests pass are never used. (ii) Processes such as reviews may slow down the feedback loops. (iii) The test results may be pushed back to developers, or pulled back by developers, or both. (iv) A test report written by a human is not always seen as an important document. (v) Testers may spend a significant amount of time investigating a failing test before filing an issue out of fear of reporting a false positive.

B. COMMUNICATION AND SOFTWARE TESTING

In what resembles a definition of communication for software engineering, Kraut and Streeter [29] write that: ''In software development, [coordination] means that different people working on a common project agree to a common definition of what they are building, share information and mesh their activities ...'' They found that certain coordination techniques were under- or over-utilized with respect to how valuable they were. Underutilized techniques were: discussion with peers, customer testing, design reviews, requirement reviews, co-organization, discussions with boss and group meetings. Overutilized techniques included: project documentation and error tracking. In the light of our results, one might say that the flow of information, the challenges and good approaches are not just relevant for testing, but for communication in software engineering in general. However, we argue that the contemporary industrial challenges with respect to software testing motivates studying this phenomenon specifically in the context of software testing of embedded systems. An overlap of testing practices and general communication practices is likely to be expected.

Communication frequently relies on the use of tools and technology. Olson and Olson [38] use the example of a conference phone and mention a type of failure in remote work: ''They adapted their behavior rather than fix the technology. On many occasions, the participants shouted because the volume [...] was set too low.'' They also noticed that new technology may make geographical distance smaller, causing cultural distances appear to be greater [38]. Bjarnason et al. [9] propose a theory for distances in software engineering, it ''explains how practices improve the communication within a project by impacting distances between people, activities and artefacts.'' They mention eight people-related and five artifact-connected distances. The first of the people-related distances are also mentioned by our interviewees: geographical, socio-cultural, temporal and organizational. As an example, the practice of cross-role collaboration has an impact on distances, for example on the temporal distance. From this perspective, several of the approaches we propose could be seen as related to this theory of distances.

Illes-Seifert and Paech [25] investigate the role of communication, documentation and experience during system testing. They found that ''most communication during the testing process occurs with the requirements engineer and the project manager followed by the developer.'' This is a type of

pattern we had expected to be able to find in our interview data, but we cannot confirm or reject their finding. However, we note that communication patterns like this one could be of importance to the practice of software testing.

C. COMMUNICATION IN AGILE ENVIRONMENTS

Two recent papers on agile communication practices discuss approaches and barriers for agile communication. The barriers for effective knowledge sharing were categorized as team factors, process factors and contextual factors, and include: distances, lack of social skills, stakeholder neglect of non-functional requirements, product owner lack of sharing client feedback, inadequate planning, insufficient documentation, as well as lack of high-quality collaboration techniques and processes [21]. Proposed approaches for enhanced communication in a global outsourced agile development included the use of good tools that can replace face-to-face communication when teams are distributed, as well as the use of tools for social networking and continuous integration. The authors also propose to establish trust and a one-team mentality to overcome cultural distances [15]. This is in line with our findings.

Continuous integration is an important agile practice. A recent study by Spieker et al. [46] highlights the need for speeding up continuous integration, and they describe a method ''with the goal to minimize the round-trip time between code commits and developer feedback on failed test cases.'' In a literature study of 69 papers on continuous practices Shahin et al. [43] identified that, as code integrations become more frequent, the amount of data such as test results will increase exponentially, therefore it is ''critical to collect and represent the information in a timely manner to help stakeholders to gain better and easier understanding and interpretation of the results…'' Also: ''[CI tools] produce huge amount of data that may not be easily utilized by stakeholders …'' Furthermore, they identified 20 challenges in continuous integration. Three of them were lack of awareness and transparency, more pressure and workload for team members, and skepticism and distrust on continuous practices.

We identified that a review process, and the practice of bundling several software changes before testing can both have a negative impact on the pace at which software is delivered from developers to the test environment. Two papers on continuous practices in the embedded systems domain has identified additional factors to this challenge: having a time-consuming or complicated delivery process, difficulties with configuration management of the test environment, and also human factors such as team structure, or not seeing value in delivering often, [33], [34].

D. TEST DESIGN TECHNIQUES

Well-defined test design techniques exist, e.g. [4] and [18], but in practice ''testers rely on their own experience more than on test design techniques when deciding on test data and test steps.'' [25]. Our results indicate that a pattern of unbalanced test techniques exist in the industry; organizations seem to overly rely on one or a few testing techniques, and they seem to over-depend on confirmatory and positive testing.

E. TEST RESULTS VISUALIZATION

A number of studies on visualizing test results exist. Just as Shahin et al. [43] proposed, Nilsson et al. [36] emphasize that summaries are needed, and Brandtner et al. [10] find that information may be spread out in several different systems. Both studies also propose visualization methods. In two recent literature studies [54], [55] the authors investigated issue tracking and highlighted that too long reports are bad, and that they need to be summarized, which is in line with what we found. We also found that testers may spend a significant amount of time before reporting an issue due to fear of filing a false negative. Opmanis et al. [39] considers transitions between different views of a test results visualization tool. Strandberg [48] suggested that visualizations may target managers, experts, testers and code maintainers, with the purpose of giving early warnings, suggestions for code improvements, planning support and support for making decisions. The results of this study emphasize a need for visualization and clear summaries as well as the generation of custom test reports.

Ralph [41] argues that many mistakes in software development have a root cause in cognitive biases, and that ''sociotechnical interventions'' such as planning poker might reduce the effects of these biases. We speculate that good practices such as custom automated reports and the use of visualizations can serve to de-bias decision making. Another aspect related to cognition is how visualizations are perceived. Zeileis et al. [53] discuss that when making statistical graphics the colors should ''not introduce optical illusions or systematic bias'' furthermore, the ''different areas of a plot should still be distinguishable when […] displayed on an LCD projector […] or when it is printed on a gray-scale printer, or when the person viewing the graphic is color-blind.'' Our study shows that test results visualizations created in the embedded system industry today seem to be implemented by engineers that are not skilled in how to make interfaces with these considerations in mind.

F. TESTING EMBEDDED SYSTEMS WITH AND WITHOUT HARDWARE

Because of timing issues and other non-functional aspects, the software testing of embedded systems has to be conducted on target hardware at some point in the development process [6], [12], [52]. This will have a significant impact on unit testing, if done on target, as software uploads will be a limiting factor. Cordemans et al. [12] identified strategies for combining unit level testing on both target and host, as well as proposed new strategies for test-driven development of embedded systems, in particular they discuss hardware mocking. In addition to testing on host and on target, the use of simulators and emulators are common and allow higher level testing without access to hardware. Becker et al. [7] developed XEMU which extends the

common QEMU hardware simulator [8] for mutation based testing of embedded systems. The importance of simulation was also emphasized by web-based survey with more than 1200 respondents as the most important design technique for future designs of embedded systems [35]. With respect to the flow of information in software testing, we could not only expect faster feedback loops with increased simulation, testing would also be able to start earlier in the development process. An earlier start of the testing could reduce the impact of the gate at step (6) in Figure 2. There is, however, a cost involved in developing and maintaining such tools.

G. IMPLICATIONS FOR INDUSTRIAL PRACTICE AND RESEARCH

Based on our findings, organizations developing embedded software may want to tailor their communication processes to improve the information flow and feedback loops. Tools and frameworks for test result reporting and communication should take these challenges and approaches into account. For example, when developing frameworks for automated test reporting one should try to address the challenges of understanding the objectives and details of a test result and how testing was conducted by providing all the needed instructions and details; one idea would be to provide a framework suggesting that a particular failure was due to errors in the test environment, simulators, test case itself, test framework, software or hardware under test. Successfully implementing this kind of suggestion system could save a lot of time in properly using and communicating test results.

Some of the mitigations to the challenges described are already known to industry. One example is the ISTQB released syllabus for certification of test automation engineers from 2016 [5]. For example, it covers a generic test automation architecture, as well as the potential risks when implementing and using test automation. We would like to highlight the chapter on reporting and metrics as these are topics that occurred frequently in our thematic analysis and results. This chapter recommends to: (i) measure benefits of automation (costs are easily seen), (ii) visualize results, (iii) create and store logs from both the system under test and the test framework, and (iv) generate reports after test sessions. This is an indication that this syllabus could be relevant to the information flow in software testing, both for an academic and an industrial perspective.

For research, the findings in this paper are important as they bring ''tacit knowledge'' from industrial practice into academia. By knowing that the information flow in software testing is not a trivial process, further research is made possible. We also provide a model for what we know about the information flow inner-workings in organizations developing real-world embedded software. We invite other researchers to add on, revisit in other contexts, and possibly revise or reject, this model of the overall flow of test information.

V. VALIDITY EVALUATION

Here we present a validity analysis based on the guidelines proposed by Runeson and Höst, [42].

A. CONSTRUCT VALIDITY

Research with good construct validity investigates the phenomenon that the researchers intended to study – a misunderstood question is an example of a threat to construct validity. The design of our instrument was based on research questions, and was thoroughly reviewed before use (by running pilot interviews and evaluating the instrument in workshops). We designed a process for managing and anonymizing the interview audio files and transcripts, and by presenting it to the interviewees we reduced the threat of evaluation apprehension, and reactivity, where interviewees alter their behavior in the presence of others. Information flow in software testing is an area of research that intertwines with other processes of software development, therefore, we had to ask interviewees a large number of questions, not necessarily focused on communication of test results and the information flow per se. This resulted in a large amount of data to analyze for understanding the full spectrum of software testing and sometimes it was difficult (in our analysis) to primarily focus on the flow.

B. INTERNAL VALIDITY

FOR a qualitative study as ours, internal validity translates to how credible the results and the analysis are. There was a non-random selection of the interviewees – so one could think of selection bias as a validity threat. However, since our focus is on embedded systems development, we had to select relevant companies and random selection was not an option. We were also careful in selecting a variety of roles to interview to reduce the threat of selection bias. This study is performed in the context of embedded software organizations in Sweden. Our general perception is that the overall communication style in Swedish embedded software organizations is informal and this might have impacted our analysis. Thus our results should be interpreted with keeping the context in mind. Finally, as with the majority of interview studies, the credibility of our results would have improved by conducting a much larger set of interviews, owing to availability of time and availability of industrial participants.

C. EXTERNAL VALIDITY

External validity refers to the generalizability of the study results. We interviewed more than one type of role, in multiple organizations. Our focus on maximizing diversity in organizations, roles and also in how coding of interviews was made (more than one researcher coded each interview), should serve to increase the external validity of the paper, as should our efforts to try to ensure theoretical saturation. Also, we followed the guidelines proposed by Petersen and Wohlin [40] and make a detailed report on the context (in section III-A) to support transfer of results in similar

contexts as ours. However, it should be noted that the external validity of an interview study is always limited and our results should be seen as a proposal for an explanatory model of information flow in embedded software testing, rather than conclusive evidence.

D. RELIABILITY

Studies with good reliability are independent of the researchers that performed them. We have been careful in describing the method used for conducting this study and also made the instrument available [49], making it easier for researchers and practitioners to understand the details of this study. This threat has also been addressed by involving multiple researchers and iterating over proposed results between researchers. That said, it is most likely inevitable for our preconceptions and cultural and professional backgrounds to have affected the data collection as well as the synthesis processes, and consequently also the results. We thus welcome complementary work on this topic in different settings and by authors with contrasting backgrounds.

VI. CONCLUSION

We have conducted an interview study of information flow in software testing of embedded systems in five organizations in Sweden. The results presented in this paper are based on semi-structured interviews with twelve practitioners with an average work experience of more than fourteen years, and thematic analysis to identify major themes, challenges and good approaches. Based on this analysis we describe the flow of information, and show how this communication process is built up of a number of feedback loops originating in software development activities, typically connecting test activities, test artifacts, test results databases and issue trackers with teams of developers, testers and managers (as described in Figure 2). These loops are negatively affected by increased geographical, social, cultural or temporal distances. We discovered themes that impact this flow: how organizations conduct testing and trouble shooting, communication, processes, technology, artifacts, and the organization of the company (summarized in Table 1).

We identified several challenges (Table 2) that need to be considered: comprehending the objectives and details of testing, root cause identification, poor feedback, postponed testing, poor artifacts and traceability, poor tools and test infrastructure, and distances. In addition, we highlight approaches (Table 3) positively influencing the flow of information. Close collaboration and communication between different roles was perceived as a good way to enhance the information flow. Fast feedback, custom report generation, and use of suitable tools were seen as critical approaches in dealing with test results in an efficient manner. Our study also indicates that visualization of test results can be used for fostering communication around testing.

Finally, our results show that more research on the flow of information and communication in testing is needed and that practitioners need to take it more clearly into account; it is not enough to improve testing itself unless its results are clearly communicated so they can be acted upon.

VII. FUTURE WORK

Future work could investigate the information flow in contexts other than embedded software organizations, use a larger sample, if possible in additional countries, in order to strengthen our understanding, identify more challenges and helpful approaches. In a recent literature study of literature studies, Garousi and Mäntylä, [20], found that there is a "need for secondary studies in the areas of test-environment development and setup, test results evaluation and reporting." Secondary studies on test reporting, as well as on communication within the context of software testing, could widen and extend the collective understanding of the flow of information in software testing. Finally, we would like to see implementation of tools for improving the information flow in software testing: from introduction of visualizations, to how to make test results communication feedback loops faster, as well as more general exploration tools of all aspects of test results. Perhaps simplifications when going from test verdict via root cause analysis and source code exploration to discussions with colleagues on possible fixes for issues.

REFERENCES

[1] *IBM Rational DOORS Family*. Accessed: Dec. 10, 2018. [Online]. Available: https://www.ibm.com/us-en/marketplace/rational-doors
[2] *Jenkins*. Accessed: Dec. 10, 2018. [Online]. Available: https://jenkins.io/
[3] *Mantis Bug Tracker*. Accessed: Dec. 10, 2018. [Online]. Available: https://mantisbt.org/
[4] P. Ammann and J. Offutt, *Introduction To Software Testing*. Cambridge, U.K.: Cambridge Univ. Press, 2016.
[5] B. Bakker *et al.*, "Certified tester advanced level syllabus test automation engineer," in *Proc. Int. Softw. Testing Qualifications Board (ISTQB)*, 2016, pp. 1–84.
[6] A. Banerjee, S. Chattopadhyay, and A. Roychoudhury, "On testing embedded software," *Adv. Comput.*, vol. 101, pp. 121–153, 2016.
[7] M. Becker, D. Baldin, C. Kuznik, M. M. Joy, T. Xie, and W. Mueller, "XEMU: An efficient QEMU based binary mutation testing framework for embedded software," in *Proc. 10th ACM Int. Conf. Embedded Softw.*, 2012, pp. 33–42.
[8] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proc. USENIX Annu. Tech. Conf., FREENIX Track*, vol. 41, 2005, p. 46.
[9] E. Bjarnason, K. Smolander, E. Engström, and P. Runeson, "A theory of distances in software engineering," *Inf. Softw. Technol.*, vol. 70, pp. 204–219, Feb. 2016.
[10] M. Brandtner, E. Giger, and H. Gall, "Supporting continuous integration by mashing-up software quality information," in *Proc. Softw. Evolution Week IEEE Conf. Softw. Maintenance, Reeng. Reverse Eng. (CSMR-WCRE)*, Feb. 2014, pp. 184–193.
[11] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Res. Psychol.*, vol. 3, no. 2, pp. 77–101, 2006.
[12] P. Cordemans, S. Van Landschoot, J. Boydens, and E. Steegmans, "Test-driven development as a reliable embedded software engineering practice," in *Embedded and Real Time System Development: A Software Engineering Perspective*. Berlin, Germany: Springer, 2014, pp. 91–130.
[13] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Commun. ACM*, vol. 31, no. 11, pp. 1268–1287, 1988.
[14] R. C. de Boer and H. van Vliet, "On the similarity between requirements and architecture," *J. Syst. Softw.*, vol. 82, no. 3, pp. 544–550, 2009.
[15] T. Dreesen, R. Linden, C. Meures, N. Schmidt, and C. Rosenkranz, "Beyond the border: A comparative literature review on communication practices for agile global outsourced software development projects," in *Proc. IEEE 49th Hawaii Int. Conf. Syst. Sci. (HICSS)*, Jan. 2016, pp. 4932–4941.

[16] C. Durugbo, A. Tiwari, and J. R. Alcock, "Modelling information flow for organisations: A review of approaches and future challenges," *Int. J. Inf. Manage.*, vol. 33, no. 3, pp. 597–610, 2013.
[17] C. Ebert and C. Jones, "Embedded software: Facts, figures, and future," *Computer*, vol. 42, no. 4, pp. 42–52, 2009.
[18] S. Eldh, "On test design," Ph.D. dissertation, School Innov., Des. Eng., Mälardalens Univ., Västerås, Sweden, 2011.
[19] R. Galvin, "How many interviews are enough? Do qualitative interviews in building energy consumption research produce reliable knowledge?" *J. Building Eng.*, vol. 1, pp. 2–12, Mar. 2015.
[20] V. Garousi and M. V. Mäntylä, "A systematic literature review of literature reviews in software testing," *Inf. Softw. Technol.*, vol. 80, pp. 195–216, Dec. 2016.
[21] S. Ghobadi and L. Mathiassen, "Perceived barriers to effective knowledge sharing in agile software teams," *Inf. Syst. J.*, vol. 26, pp. 95–125, Mar. 2016.
[22] U. H. Graneheim and B. Lundman, "Qualitative content analysis in nursing research: Concepts, procedures and measures to achieve trustworthiness," *Nurse Edu. Today*, vol. 24, no. 2, pp. 105–112, 2004.
[23] G. Guest, A. Bunce, and L. Johnson, "How many interviews are enough?: An experiment with data saturation and variability," *Field methods*, vol. 18, no. 1, pp. 59–82, 2006.
[24] *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems*, Standard IEC 61508-1:2010, 2010.
[25] T. Illes-Seifert and B. Paech, "On the role of communication, documentation and experience during system testing—An interview study," in *Proc. PRIMIUM*, 2008, pp. 1–15.
[26] *Road Vehicles—Functional Safety—Part 1: Vocabulary*, Standard ISO 26262-1:2011, International Organization for Standardization, 2011.
[27] *ISO/IEC/IEEE Software and Systems Engineering—Software Testing—Part 3: Test Documentation*, ISO/IEC/IEEE Standard 29119-3:2013, International Organization for Standardization, International Electrotechnical Commission, Institute of Electrical and Electronics Engineers, 2013.
[28] J. Kasurinen, O. Taipale, and K. Smolander, "Software test automation in practice: Empirical observations," *Adv. Softw. Eng.*, vol. 2010, Nov. 2010, Art. no. 620836.
[29] R. E. Kraut and L. A. Streeter, "Coordination in software development," *Commun. ACM*, vol. 38, no. 3, pp. 69–82, 1995.
[30] P. Lenberg, R. Feldt, and L. G. Wallgren, "Behavioral software engineering: A definition and systematic literature review," *J. Syst. Softw.*, vol. 107, pp. 15–37, Sep. 2015.
[31] P. Liggesmeyer and M. Trapp, "Trends in embedded software engineering," *IEEE Softw.*, vol. 26, no. 3, pp. 19–25, May/Jun. 2009.
[32] J. Linåker, S. M. Sulaman, M. Höst, and R. M. de Mello, "Guidelines for conducting surveys in software engineering," Dept. Comput. Sci., Lund Univ., Lund, Sweden, 2015. [Online]. Available: http://portal.research.lu.se/portal/files/6062997/5463412.pdf
[33] L. E. Lwakatare *et al.*, "Towards devops in the embedded systems domain: Why is it so hard?" in *Proc. IEEE 49th Hawaii Int. Conf. Syst. Sci. (HICSS)*, Jan. 2016, pp. 5437–5446.
[34] T. Mårtensson and P. Hammarström, and J. Bosch, "Continuous integration is not about build systems," in *Proc. IEEE 43rd Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Aug./Sep. 2017, pp. 1–9.
[35] C. Maxfield. (2017). *2017 Embedded Markets Study*. [Online]. Available: Eetimes/embedded.com
[36] A. Nilsson, J. Bosch, and C. Berger, "Visualizing testing activities to support continuous integration: A multiple case study," in *Proc. Int. Conf. Agile Softw. Develop.* Cham, Switzerland: Springer, 2014, pp. 171–186.
[37] K. Olsen *et al.*, "Certified tester foundation level syllabus," in *Proc. Int. Softw.Testing Qualifications Board (ISTQB)*, 2018, pp. 1–96.
[38] G. M. Olson and J. S. Olson, "Distance matters," *Hum.-Comput. Interact.*, vol. 15, nos. 2–3, pp. 139–178, 2000.
[39] R. Opmanis, P. Kikusts, and M. Opmanis, "Visualization of large-scale application testing results," *Baltic J. Mod. Comput.*, vol. 4, no. 1, p. 34, 2016.
[40] K. Petersen and C. Wohlin, "Context in industrial software engineering research," in *Proc. 3rd Int. Symp. Empirical Softw. Eng. Meas.*, 2009, pp. 401–404.
[41] P. Ralph, "Toward a theory of debiasing software development," in *Research in Systems Analysis and Design: Models and Methods*. Berlin, Germany: Springer, 2011, pp. 92–105.
[42] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Eng.*, vol. 14, no. 2, pp. 131–164, 2009.
[43] M. Shahin, M. A. Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. 5, pp. 3909–3943, Mar. 2017.
[44] *World Quality Report 2014–2015*, Sogeti, Capgemini, and HP, Paris, France, 2015.
[45] I. Sommerville, *Software Engineering*. London, U.K.: Pearson, 2015.
[46] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proc. 26th Int. Symp. Softw. Test. Anal. (ISSTA)*, Santa Barbara, CA, USA, Jul. 2017, pp. 12–22.
[47] K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: A critical review and guidelines," in *Proc. ACM 38th Int. Conf. Softw. Eng.*, 2016, pp. 120–131.
[48] P. E. Strandberg, "Software test data visualization with heatmaps," Mälardalen Real-Time Research Centre (MRTC), Mälardalens Univ., Västerås, Sweden, Tech. Rep. MDH-MRTC-318/2017-1-SE, 2017.
[49] P. E. Strandberg, E. P. Enoiu, W. Afzal, D. Sundmark, and R. Feldt, "Instrument from: Test results communication—An interview study in the embedded software industry," Mar. 2018. doi: 10.5281/zenodo.1189562.
[50] *Forskningsetiska principer inom humanistisk-samhällsvetenskaplig forskning*, Swedish Res. Council, Stockholm, Sweden, 1996.
[51] K. Wiklund, S. Eldh, D. Sundmark, and K. Lundqvist, "Impediments for software test automation: A systematic literature review," *Softw. Test., Verification Rel.*, vol. 27, no. 8, 2017, Art. no. e1639.
[52] W. H. Wolf, "Hardware-software co-design of embedded systems," *Proc. IEEE*, vol. 82, no. 7, pp. 967–989, Jul. 1994.
[53] A. Zeileis, K. Hornik, and P. Murrell, "Escaping RGBland: Selecting colors for statistical graphics," *Comput. Statist. Data Anal.*, vol. 53, no. 9, pp. 3259–3270, 2009.
[54] J. Zhang, X. Wang, D. Hao, B. Xie, L. Zhang, and H. Mei, "A survey on bug-report analysis," *Sci. China Inf. Sci.*, vol. 58, no. 2, pp. 1–24, 2015.
[55] T. Zhang, H. Jiang, X. Luo, and A. T. S. Chan, "A literature review of research in bug resolution: Tasks, challenges and future directions," *Comput. J.*, vol. 59, no. 5, pp. 741–773, 2016.

PER ERIK STRANDBERG received degrees in applied mathematics and bioinformatics from Linköping University, in 2004 and 2005, respectively, and the Licentiate of Engineering degree from Mälardalen University, Västerås, Sweden, in 2018, where he is currently pursuing the Ph.D. degree in the topic of software test automation of networked embedded systems. He has more than a decade of practical experience of working with software development, software testing, and requirements engineering. He is employed as the Test Lead and an industrial doctoral student with the Westermo Network Technologies AB, Västerås.

EDUARD PAUL ENOIU received the Engineer's degree from the Polytechnic University of Bucharest, in 2009, and the Ph.D. degree from Mälardalen University, Västerås, Sweden, in 2016, where he is currently a Researcher, primarily affiliated with the Software Testing Laboratory and the Formal Modelling and Analysis Groups, Department of Networked and Embedded Systems. His research interests include software engineering and empirical research.

WASIF AFZAL received the Ph.D. degree in software engineering from the Blekinge Institute of Technology. He is currently an Associate Professor with the Software Testing Laboratory, Mälardalen University. His research interests include software testing, empirical software engineering, and decision-support tools for software verification and validation.

DANIEL SUNDMARK is currently a Professor of computer science with Mälardalen University, Västerås, Sweden, and the Leader of the Software Testing Laboratory Research Group. Since 2001, he has been focusing on research on testing, debugging and monitoring, primarily of embedded systems. His current projects, undertaken in close collaboration with industry partners, focus on engineering of embedded software and systems, particularly focusing on embedded system architecture and software and system testing.

ROBERT FELDT is currently a Professor of software engineering with the Chalmers University of Technology, Gothenburg, where he is also a part of the Software Engineering Division, Department of Computer Science and Engineering. He is also a part-time Professor of software engineering with the Blekinge Institute of Technology, Karlskrona, Sweden. He has broad research interests, but focuses on software testing, requirements engineering, psychological and social aspects, as well as agile development methods/practices. He was one of the pioneers in the search-based software engineering field. He has a general interest in applying artificial intelligence and machine learning and has tried to get more research focused on human aspects – behavioral software engineering. Since 2017, he has been the Co-Editor-in-Chief of the *Empirical Software Engineering (EMSE) Journal*.