2010 Third International Conference on Software Testing, Verification and Validation

Towards Fully Automated Test Management for Large Complex Systems

Sigrid Eldh[1,3], Joachim Brandt[2], Mark Street[2],
Hans Hansson[3] and Sasikumar Punnekkat[3]
Ericsson AB[1] Sweden, Ericsson Ltd[2] UK, Mälardalens University[3] Sweden
sigrid.eldh@ericsson.com, joachim.brandt@ericsson.com mark.street@ericsson.com
Hans.Hansson@mdh.se, Sasikumar.Punnekkat@mdh.se

Abstract—**Development of large and complex software intensive systems with continuous builds typically generates large volumes of information with complex patterns and relations. Systematic and automated approaches are needed for efficient handling of such large quantities of data in a comprehensible way.**

In this paper we present an approach and tool enabling autonomous behavior in an automated test management tool to gain efficiency in concurrent software development and test. By capturing the required quality criteria in the test specifications and automating the test execution, test management can potentially be performed to a great extent without manual intervention.

This work contributes towards a more autonomous behavior within a distributed remote test strategy based on metrics for decision making in automated testing. These metrics optimize management of fault corrections and retest, giving consideration to the impact of the identified weaknesses, such as fault-prone areas in software.

Keywords- test management system; large complex systems; automation; efficiency; industrial system;

I. INTRODUCTION

To increase efficiency, companies such as Ericsson work towards "automation everywhere", which means that all aspects of the software development – including the testing process – are targets for automation.

Today, it is not uncommon to have systems with more than 1 000 requirements, 100 000 test cases, and 1 000 software components, sub-systems and other software entities. These systems are developed in a distributed manner at multiple sites worldwide, include third party software, are frequently updated, and need to be tested at multiple integration levels. For a single site this could result in more than 50 000 test cases to execute for each iteration/release.

Even in a perfect test execution situation, every test cannot be executed on a daily basis, due to the complexity of the systems under test and execution time of the test cases. The complexity in terms of large sizes and internal dependencies of industrial systems are impacting all aspects of software development and test, something which research is rarely targeting.

To handle the complexity of the test effort an efficient and scalable test management is required. Current test management requires a lot of manual handling due to the diversity and lack of integration between tools for handling requirements, configuration management (system builds), test specification, test execution, test environments, failure handling, quality and progress reporting, etc. Total integration is not a viable solution, due cost and complexity, as well as its inability to incorporate legacy and emerging solutions.

This paper presents an autonomous test management framework with the following key features:

- A loosely-coupled integration of diverse tools, based on automated extraction and synthesis of a set of measurements.
- Using test specifications as release criteria, the system becomes ready for release when all tests have been successfully performed.
- Automated traceability of failures to software components, achieved by enforcing test specifications to represent system entities.

This provides cost efficiency through reduction of manual intervention and improved quality assessment in the context of complex industrial systems, since historic quality data can be taken into account. Using this framework, the role of test managers during test execution can essentially be eliminated. This has contributing to substantial savings for Ericsson over the past seven years in use.

Test management systems have been around for many years, with several commercial off-the-shelf (COTS) tools available. One of the motivations of this work is the problems associated with these, as e.g. outlined in [24]. At least 16 open source systems are available, and within Ericsson more than 20 different test management systems have been developed, considering different needs of automation. One of these systems aimed at tight integration of the involved tools for requirement, configuration, review, software and test management, as well as failure handling, all combined in a large relational database.

Our experiences from working with the above system, as well as with a variety of more loosely coupled test management systems, are that loosely coupled systems are less susceptible to performance problems and are much

978-0-7695-3990-4/10 $26.00 © 2010 IEEE
DOI 10.1109/ICST.2010.58

better in handling legacy systems. Other reasons for our lack of success with a tightly coupled solution, is that test automation is typically achieved by integrating a series of tools, and an integrated solution makes transfer to new tools costly. The flexibility of loosely coupled systems also increases their longevity, increasing the chances of being able to use the same test management system throughout the lifetime of the system.

Outline: In section II we describe an overview of test management systems in general and in section III the limitations of these types of systems. We present in section IV our solution, the system used in this study in section V, followed by a discussion in section VI, related work in section VII, and conclusion in section VIII.

II. OVERVIEW OF TEST MANAGEMENT SYSTEMS

Test execution and test management need to cater for the dynamics in the software build process. This means that for large complex industrial software systems the measurements collected throughout the test process are essential for the continuous quality assessments. Test management in the execution phase assumes that test cases are created, and that extensive regression testing is required during the entire project, taking historic test results into account.

The historic data can be as recent as yesterday, or originate from substantially older versions of the system. A positive consequence of automation is that it becomes feasible to mine and collect new measurements, which give new insights to the efficiency of software development and test, as well as highlighting potential flaws in a chosen approach. Figure 1 shows a screen shot from a test management tool, presenting the fraction of test cases executed for a sequence of builds. This is a typical picture of a parallel development and test process, showing how configuration management is combined with monitoring of test status.

Tests - Test Results - Reports - Admin - Result Import - Users - Log Files - Logout

1:18:49 2009 **Report:** Build Statistical Summary
Document: All **Test Case:**
Variant: All **Test State:** All **Qualifier:** All
Rig: All **Priority:** All **Review State:** All **Flags:** All

Type:	Number of Tests:	Percent/Total:	Number of Tests not Run:	Percent/T
	44	10.65%	369	89.35%
	20	4.84%	393	95.16%
	32	7.75%	381	92.25%
	42	10.17%	371	89.83%
	21	5.08%	392	94.92%
	10	2.42%	403	97.58%
	65	15.74%	348	84.26%
	37	8.96%	376	91.04%
	66	15.98%	347	84.02%
	23	5.57%	390	94.43%
	39	9.44%	374	90.56%

Figure 1: Overview of progress (number of builds and % of test cases executed)

The test management process typically consists of a test preparation phase (Figure 2) and a test execution phase (detailed in Figure 3). In Figure 2 clouds denote manual work. In Figure 3 all tasks contain manual work in a COTS tool, whereas in our tool writing of failure report (by choice) as well as analyze (debug) and correct the fault, are the only tasks that still contain manual work.

The test preparation phase is often the same in different test management systems, although with variations in how data is stored, e.g. the test specification can either be a separate textual document or an entry in the database/test management system.

The test specifications might be traceable to the requirements, and implicitly, the test case may be indirectly traceable (usually through the ID or naming convention) to either a certain test aspect or to a requirement. Some test management systems can only provide this traceability if the requirements are in the same system. An alternative is that the relations between tests and the requirements, and thus traceability, is provided in the requirement management system, requiring the testers to enter information in more than one system. It is clear that in neither of these systems there is a direct relation or possibility to trace test cases to the actual software being tested.

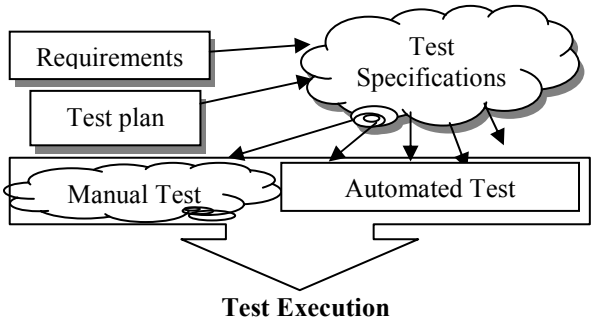

Figure 2. The test preparation phase

Basic data that are possible to record, edit, view and store in any test management system are the following:

- Dates, creators´ information which includes *version handling and change information*
- *Test cases*, including: Test ID/name/slogan, priority, and type. More elaborate systems allow a step by step description similar to the test instruction/test procedure assuming a manual or key-word driven test. Test Management systems do not contain test execution capabilities, which are left to the test automation system.
- *Scheduling* of tests (planning) which means assigning testers and other resources to one or more test cases, with qualifying information such as time, date and by whom.
- *Status of the test execution* (captured from the automated tool, imported from the automated system or manually recorded) such as passed, failed or attempted test cases. Variants of these status fields exist [10]. For failed test cases, often textual fields that

can be used for recording Failure Reports name or ID (e.g. failure ID from a failure tracking system).
- *Test environment/test bed* information, which is either a link to a document, a slogan/ID or a textual field.
- A number of *reports*, which provide possibilities to view the stored data in different ways, presented in graphical form with trend analysis.

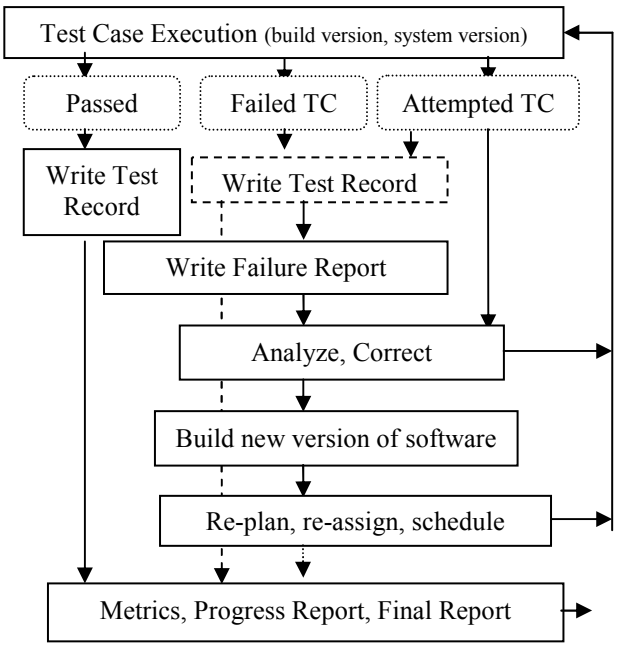

Figure 3. Activities within test execution

Depending on the test management system, different philosophies exist on what and how much is stored in the system. In some systems, only test cases and their status are stored, and others store the entire process [9]. This means that either a certain number of documents are to be linked or need to be directly written in the system. Examples of data that should exist can be found in IEEE Std. 829 -1998 [12] or [13], i.e. test plan, test specification, test procedure (often implemented as test case), test record, test reports, which all would be textual.

Finally the tools have built-in or indirect methods to integrate with other systems. For example, integration with a failure tracking system or with a test automation system is common, but more advanced test management system can also be integrated to support traceability with requirements management systems.

III. LIMITATIONS OF CURRENT TEST MANAGEMENT SYSTEMS

There is a series of problems with the current approach to test management that are listed and explained below:

- Information handling of test cases are manual [3, 5, 6, 7, 28, 29]
- The test cases are often written assuming manual execution or a tool within the same tool family [3, 5, 7, 28, 29]
- Lack of version control or communication with a configuration management tool (outside the "family of tools"). [3, 5, 6, 7, 28, 29]
- Collection of information from other systems(failure tracking, configuration management, test automation tools) is often manual [3, 5, 6, 7, 28, 29]
- Historic data (traceability to the software quality) between projects is not available [3, 5, 6, 7, 23, 28, 29]
- Changes in status of failures often requires manual imports [3, 5, 6, 7, 23, 28, 29]
- Scheduling, starting and management of regression testing (what test cases to select) is mostly a manual task [3, 5, 6, 7, 23, 28, 29]

Most of these systems deal with management and handling of *manual* test cases, their execution and outcome, and all information is entered into the system by hand.

If the test case is automated, it is not very clear what and how much of the managing information that is in the test execution automation system or in the management system. Automated test execution systems often have rudimentary handling and management of the test cases, and provide verdicts, often in the form of logs. These systems often fail to collate and present the result without substantial manual work or without manually exporting data to another system.

Historic data is not available without extensive data mining and without know-how of the software system structure in the different projects.

Handling of failure tracking in separate tools, e.g., [23], requires testers to constantly monitor when the correction is scheduled into a build. Though more modern failure tracking system can inform when a failure has been debugged and corrected, a lot of manual interventions are still required, both to make the new version with the corrections and to create information about the corrections included. This is tedious manual work. Not only must the re-assignment be done, but the actual planning of what test bed and what build to use etc. are manually selected and the build might be manually initiated. When the goal is a daily build process, this would beyond doubt require full time work to supervise, to make sure what corrections are submitted in the build, and what test can be executed as a consequence. Handling a large amount of automated test cases, the problem becomes unmanageable, and often solutions such as minimizing the number of test cases are adopted to make sure a decent turn-around time is met. Minimization of test suites seriously compromises the overall quality of the test.

Current test management tools lack in active and seamless integration with tools for system build, failure tracking, automated test execution, and report construction. Ericsson has attempted to create an "ultimate" tool, in which all involved systems are retained. Unfortunately this resulted in

heavy performance problems, and lack of scalability between different sized projects. This system also enforced a rigid (inflexible) development process. A similar though more open approach is adopted in the new Jazz platform by IBM/Rational [4], which is still under development.

IV. Our Test Management System Solution

Our Test Management System (named TMS) enables a new approach to automated test management. In TMS (shown in Figure 4) we introduce the following new elements compared to standard test management as shown in Figure 2: (1) we make explicit links between test specifications and elements of the system structure, (2) we use test specifications as release criteria, and (3) we make sure the manual test cases can be handled by the test management system, rather than being contained in separate documents. This enables control of execution through sending email to a tester when a specific manual test is to be executed or re-tested after a correction.

By defining the test specification as release criteria, and by letting test specification represent a part of the software (could both be aspects of the system, or a particular part of the software, e.g. a sub-system, component or similar), it becomes possible to use each test specification as a container for the test cases defining the particular release criteria for that particular part of the software. Each part of the software is uniquely identifiable through its structure, naming and hierarchy. The unique ID is denoting each source code, each executable, and furthermore each "level" of integration. The unique ID of the software in the system structure makes it possible to link software with the test specification. We create our test cases as an implementation of a test specification. The test specification is then manually related to the test object list, which is in its turn related to the software structure. Hence, traceability will automatically be obtained for the test case. The binding of the system structure to the test specification is indicated by the gray areas in Figure 4.

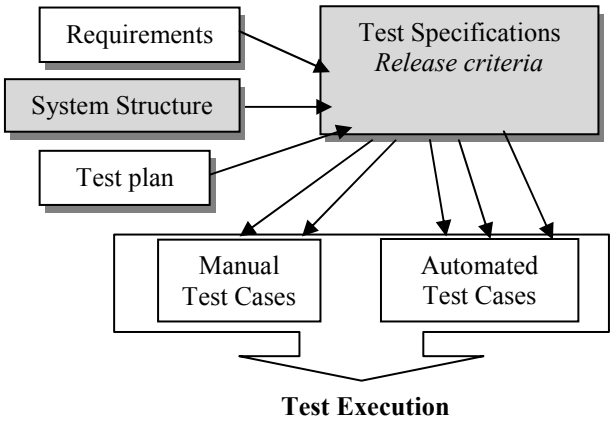

Figure 4. The test preparation phase in TMS

By tracing test result back not only to specific requirements, but also to the software system, it becomes possible to retrieve historic data on software quality. This gives a better overview of quality over time, compared to any other available test management system.

A. Is TMS a Fully Automated Solution?

The test execution can be fully automated, though we claim that there is currently no business case for fully automated generation of failure reports. The reason being that if a test case does not pass, a manual review must be made at some point in the process anyhow, and if this is done early in the process, our experience is that a substantially smaller number of failure reports are written.

Our estimations indicate that each duplicate of a failure report consumes on average one hour, and in one project this could easily amount to several man years in extra costs if allowed. This manual handling of failure reports is labor intensive, and research contributions can be seen in e.g. automatic debugging [19], but have not yet reached industrial maturity.

For example many test cases can fail for the same reason, causing duplications of reports, and it is possible that a failure is already reported in another failure tracking system or for another project on the same software. Instead of analyzing a series of "duplicate" failures that might have occurred due to the latest change, and same source file, we perform a brief analysis of the failed test cases by the testers in relation to software involved, minimizing the number of failure reports. If the same failure occurs as the result of multiple test cases, then only a single failure report should be written. The relations are visible in Figure 5[1], showing that failures are associated to sets of test cases. This is saving considerable time for developers correcting the corresponding code fault. Consequently, when failure reports are manually connected to test cases, the system makes an automated association to trace a failure to a specific software unit identified (since the test specification is a direct trace to the software). This feature in the system supports faster identification of who is to debug and correct the problem. This is an essential time saver in globally distributed software development.

B. Test Case Scheduling

The total number of test cases developed for a particular product should be represented in the test management system. The database containing this information is very large, and similar to most test management systems. The active selection of what test cases are going to be used and updated for a particular project (software enhancement) is still a test manager task. Once it is decided that these test cases form the right scope for the project, and the test cases are assigned to a tester, the test management system takes over and scheduling is automated, meaning that if a test case

[1] Note that this is just an example in which actual content of individual test specification, test cases, and failures in are not supposed to be readable.

has failed it will be automatically re-scheduled when the software is corrected in a new build.

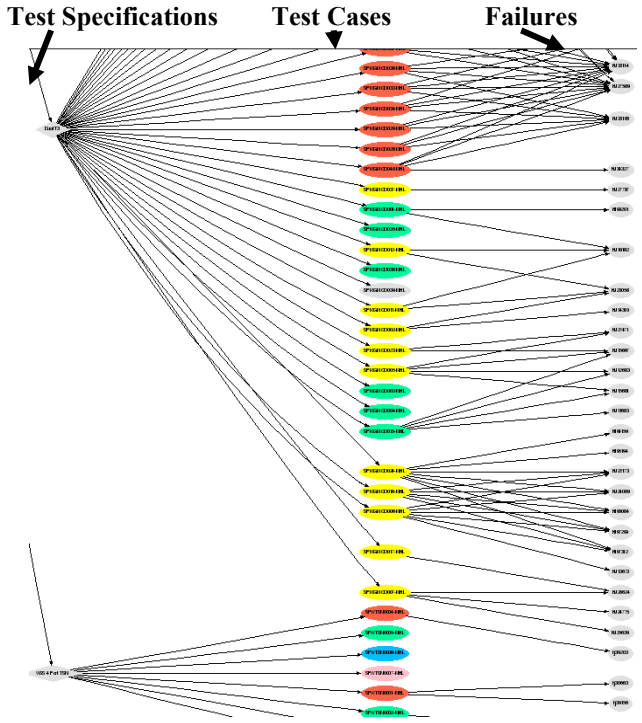

Figure 5. Hierarchical/traceability view: Relationship with failures, test case and system

Any management information needed will be retrieved by pre-defined queries to both the proprietary and commercial systems used. For example, different types of failure tracking systems are checked by the time of a request. Some data in the different databases are actively refreshed, and some need page-refreshment to be accurate. E.g., when a failure report has been "fixed" or cleared by the developer, TMS will first identify the status-change and then trigger the automated rescheduling of re-execution of all the test cases involved in that failure report. As a user you can set this regression testing at any frequency you prefer.

Our system can concurrently work with several different failure tracking systems. Traceability with respect to the initial requirements is provided by the system (see Figure 5). The test case can be tagged for regression test, for use in a particular version or branch, or for other purposes. The test cases are as usual in a test management system containing author, date, updates, priority and other necessary information.

The difference here is that the test execution will be performed by a diverse set of test tools, and that the test management system is able to handle the results in an integrated manner. The aim is to group the test cases according to how the tests relate to the structure/anatomy of the software system into software component or series of integration components. The hierarchy of the software system is clearly reflected in the different levels, as illustrated in Figure 5. By hierarchically organizing the test cases, it is easy to select which test cases should be used for executing a particular part of the system that has been changed. From an automation standpoint, TMS provides available test cases for a particular software product (part) and makes it easy to define and decide which test cases should be planned for execution and on what test bed/test environment.

C. Test Execution

Test Execution means that the test procedure or test cases are automated (implemented) in some script language to be executed with given data. Currently the test suites are driven from the test execution system, but with little effort it is possible to glue existing test harness systems with the test management.

Once the system is set up at the beginning of the project, the execution is fully automated. Still manual test cases might be assigned and executed in parallel with an automated suite. Invoking regression test (after a fix, or when initiating the system) is triggered by the TMS sending an email to a set of pre-assigned testers. Not all test cases are possible to automate (at a reasonable cost), since some tests contain elements of manual nature, e.g. to test the behavior when a cable is broken (simulated by a pulling it out manually).

This means that our automation can combine and integrate the test execution, the build system and the failure tracking system, and yet allow these systems to develop and mature at their own pace, to achieve the best execution. A test management system that combines supervision will provide a better overview of status, e.g. checking in which state a particular failure is, and have a convenient overview of what is tested in each version and in what build, in addition to provide system specific quality and historic data.

D. Instant Progress Reporting

The planning of test cases in TMS provides few new features compared to existing commercial software tools, since creating test cases, naming and describing them in a stepwise manner, and provide a verdict is not new. However, by deliberately making test specifications a representation of the system and requirements, the set of test cases will represent the system quality criteria and thus function as release criteria. This eliminates the need of monitoring during the execution phase, which substantially improves efficiency of this phase.

The test management system will in addition to planning, also support the manager when deciding which test cases should be run at a specific instant, and then it should collect the result in a way that makes the evaluation of the system clear. These common tasks are normally handled manually, as is the re-assignment to the next test execution of test cases that did not pass. Often test managers need to spend most of their time in following up problems, in particular related to the test cases that did not pass. This means ensuring that someone takes care of the problem, analyzes it, debugs and corrects the fault(s), and then makes sure that

the proper regression testing is done. Many systems will provide a clear pass or fail picture of the system at any instant, but there are seldom more information that improves on the quality assessment. A more modern test management system allows defining the status types at set up time. Combining historic and concurrent information of software components is an important feature of this system that contributes to a much more elaborate analysis of quality and regression tests. Not only should the particular test case which found the failure be re-tested, but also any other dependent test cases should be re-scheduled. How the dependencies of test cases are related to the software components is described and defined in the test specifications and test procedures. This feature is provided by the system and will in both a push and pull fashion get the test result from whatever test automation tool in use. The feature is one of the main contributors to the perceived savings and success of TMS. The hierarchy of the system and its test cases is clearly reflected – and visible – in the test management system.

The most common data used to describe progress and actual results for most Test Management systems are combining the accumulated number of test cases planned, attempted, passed or failed, as well as faults found and fixed. In addition Ericsson uses the following status labels:

- *Blocked Not Run*, which means the test case cannot be executed, since something (failure of another test case, code, equipment etc) is missing
- *Blocked Run*, which means that the test case was attempted and executed partially, but could not complete due to a blocking problem (special case of fail)
- *Concessed*, which means that the test case was granted a concession to be deferred to a later release due to irresolvable reason.
- *Not Possible*, a status describing test case description/implementation that was scoped in but not implemented satisfactorily to test execution
- *Correction (fix) Available*, which is a measure of corrections (that should fix a failed test case), as discussed with respect to fault-failure relations in [16].

The combined test case results for a certain project is also available at any time, which supports baseline, and release support. This means that at any given moment it is possible to enforce sign off-procedures, which is to be used for auditing purposes. This status is communicated by the use of icons and color in the system.

The trend analysis can be used to monitor progress at any time of the test execution, and gives information on how much work is needed for completion of the test tasks in the project. This is considered the most important task of a test management system. We have found that this data can give an accurate picture of the progress of the software development, but it can also be deceiving since the reporting is based on a non-disturbed environment, and contains accumulated data. In fact, for every new build or release there should be an individual curve, since each build is a "new" system, and if not a thorough dependency analysis (on how the software impacts the change) is done, one cannot make sure that because a test case passed the first time, the same test case is unaffected in the next new build version. The probability that the test case will pass the next time is probably high, but should not be taken for granted. Therefore it is important to have other metrics that give a better picture of the quality. Currently the Configuration Management system holds the data of what lines have been changed, but the information is insufficient, since every code change triggers a series of execution impacts, and can thus make test cases that have passed fail, even if they were not directly involved in the change. This is due to dynamic binding and dependencies in the system.

There is seldom time to execute all tests in a regression suite. Based on available reports we have identified that some projects could not execute more than 40% of their test cases in a suite. A goal must be that at least once in each release, all test cases should pass, preferably in the final test suite. By comparing the accumulated view provided in the trend analysis, the percent of tests provides a new visibility, which provides an overview of the system regression testing can be reached. It should be clear that what is considered "100%" completed in Figure 1, is referring to the selected number of test cases, not the system. The numbers of selected test cases are still only a small sample of all possible test cases that could be created for the software in question. The number of builds in a system is an indication of the number of changes of the system.

E. Failure Tracking and Test Case relations

TMS allows different failure tracking systems to be used for the same project. This reflects a common work-habit that developers in the agile work-teams often record their failures differently than testers. The customers also have different ways to report anomalies. The different failures should be consolidated by relating them to one or many test cases. If the test case is hierarchically organized in containers reflecting the system components, this could give a strong indication on what part of the system needs more attention. One example of this is that if a particular fault is contributing to failing several test cases, it should be given priority.

This support is a necessity when there are thousands of failures to correct in a large complex system, where this still equates to a failure density of telecom grade (aka 10^{-6}) in relation to the code base. In this context it is not humanly possible to determine the priority in terms of failures relating to the development. Our system contributes to an improved sensitivity analysis of the failures and subsequent software improvement, due to better data.

V. THE TEST MANAGEMENT SYSTEM USED IN THIS STUDY

Test Management System (TMS) was developed by using open source products, including MySQL v. 5 [14], Apache web server v.2 [15], and Perl scripts. The first prototype was developed in 2002 during one year, and has been used and constantly improved since then. TMS is an example of a tool that has been created out of need, in a situation where no commercial tool has been possible to use or available to handle the diverse environment. By integration with the largest test harness execution system at Ericsson, it will support future generations in strong competition with a number of commercial tools.

Is TMS fully automated? Yes, in one sense, since all the tasks performed by the test manager in the execution phase can be removed. The entire test execution phase includes manual tester tasks, e.g. writing failure report, to minimize the administrative overhead in analyzing (see discussion below). The idea was that once the test cases were entered, the system would be available to all to give an instant overview of all aspects of the progress until completion through its web-interface, guarded in availability by user security settings. This goal is not farfetched, and few enhancements are needed to get there. We are still hesitant to automate the failure report writing, even if it was possible, since all available logs and data exist. We can easily assign the failed test cases to the correct design team, but believe that it is better to let testers write the reports. But even if this part of the process is manual, the rest of the process is automated, since when the debugging (currently also manual – but not a part of any test management system) and correction is done, the developer responsible will assign the correction to a build and change the status in the report.

Currently the test management system will monitor and make sure the correct test or tests for the specific fault is rescheduled and retested. This is visible in Figure 5, where it is easy to identify test cases that failed, and the current status of the correction. Figure 5 also shows that one test case can have two or more failures that must be corrected to pass the test case. The colors represent the status, e.g. if a passed test case have a failure associated with it but is now corrected and re-tested.

The TMS system contains a series of reports, not seen in normal test management systems, including:

- Test Statistical overview
- Hierarchical/traceability overview (Figure 5) – where you can visualize the relationship between the system/documents test cases and failures.
- Test State overview – which gives the test progress by Test Specification, thus software area or characteristic of the system.
- Failure (Fault) Statistical summary– which gives failures based on priority – where you can get a multitude of system versions, and also for all history of the product, where different states can be identified from the failure tracking system.
- Time Statistical Summary – which gives time for executing the test case. This is automatically recorded for the automated tests, and gives a feedback for planning purposes. This also gives an input on how to combine automated daily builds in different suites. It also shows, the total time used for running a certain test, and an average execution time for each test.
- Change Activity Report – presenting what a particular team has done recently, and what change has occurred in the project.

In our quest to automate the entire management, we want to find measurements that truly would give important insight in the actual process. Not only how progress is proceeding, but giving us a feedback on the quality evaluation – and also on where extra attention needs to be focused on. Important data that could easily be retrieved from the TMS system are:

- What test cases are associated with a certain part of the system, and traceability to either requirement or software parts
- What faults/failures that are known for a certain product – and the current status of all these
- How many faults a certain product or part of product have had over time and where
- How long a test execution takes (both for manual and execution), which make better planning of regression suites and manual testing possible
- Estimations on total time for test execution in the system
- What test cases located failures
- Time to fix a fault and time for retest e.g. turnaround time

These measures contribute to improve planning of both manual and automated tests, planning of order to correct failures – which is a very important area in large complex system, since more testing means more failures found. An advanced test suite finds more failures, since it has higher coverage.

VI. DISCUSSION

Our claim and hypothesis is that it is possible to automate many of the test management tasks, e.g. supervision of test cases being corrected and reassigning test cases to the different testers. This means that the work of the test manager could diminish almost completely in the test execution phase. The tester, once assigned to the test case, will make sure it executes to completion, analyze if a failure occurs, and report failures, and when corrected, automatically be reassigned to re-testing (if manual), or alerted if a failure in the automatic suite is occurring. The entire test suite will be automatically executed for each release. Automating the test case execution will in itself contribute to efficiency, but our focus on automation of the

entire test execution phase could save substantial additional time.

We have built and used a tool in real production, and tried it over seven years to evaluate its sustainability to carter for old and new tool adaptations. In practice, a large number of tools are used to cater for different aspects of the test process in large scale, complex software development within Ericsson.

Our results indicate that not only does efficiency by automating the test execution, but also by automating support for test management beyond the normal test management systems, which makes the test execution phase seamless. Our aim in this paper is to clarify some distinguishing features of our system, and also to present a more elaborate view of future test management and test, which we envisage to be more autonomous in nature, compared to contemporary test management.

The strength of our test management system lays in its open interaction with failure handling, build management systems, and several different test automation and execution systems, which enables new types of automated information processing. During the setup of the test plan and test specification, traceability is provided through the test specifications that explicitly relate the requirements to the corresponding software system parts. This is enabled by active polling through queries; the result of the query (the log) gets selected, aggregated and displayed according to the presentation required in our test management system web interface.

This contributes to a new and unique overview of the system being developed. The data can then be combined into reports containing, e.g. execution times, failure traceability and test case dependencies, which adds new information and insight into the product development.

Development of the TMS system as well as its evolution over the past few years has given us a series of insights. We have questioned the practice where current test management systems require many manual tasks. Our insights can be summarized in the following statements:

- Creating test cases based on test specifications that are defined (linked) to the system components enables traceability. This provides the novel feature of being able to assess the quality of individual software components over time in many different versions and configurations, instead of only for a specific project.
- Considering a set of passed test cases as release criteria makes the test manager role redundant in the test execution, since all tasks of are handled by the test management system.
- We can automate the test execution process, including the handling of build information, and failure tracking and to combine these different tools and separate processes into one seamless flow of loosely coupled separate systems.
- Even if it is possible to automate failure reports, we advise against it. If one fault makes many test cases fail, this will result in many duplicated failure reports, which is inefficient.
- It is possible to automatically build, reschedule and automatically re-test corrected code, without any human intervention. This enables daily build and an extremely quick turn-around time.

By our contribution several manual tasks in the test execution phase are not needed in our proposed test management system, such as handling progress, handing out tasks, dealing with failure reports, re-planning of test cases for regression test, and identifying software with poor quality.

The remaining tasks can be part of the basic project management. Progress and trend information is available instantly in the system. The system is ready for release when all test cases are passed or remaining failure cases have been handled. Test execution planning of the regression suites improves when execution times are available for both manual and automated test cases.

Tracking execution times will be instrumental in planning the daily build regression suites, to make sure the test suite can be completed within the time limits. Our system contributes to remove some of the administrative overhead, where one important aspect is enabling a faster turn-around time (the time between failure discovery, until it is corrected and back for retesting).

VII. Related Work

There are several commercial off-the-shelf (COTS) tools such as [3, 4, 5, 6, 23, 28, 29] available. One of the motivations of this work is the problems associated with these, as e.g. outlined in [24]. There are at least 16 open source systems available with various features and qualities identified [7], and within Ericsson more than 20 different test management systems have been developed, considering different needs of automation.

There are a web-based system to support a process similar to ours [1] and a related evaluation of test automation management [2]. None of these solutions cater for all aspects in our solution, especially the handling of a multitude of different systems. The area of test management tools is surprisingly scarce in research papers, as vivid as it is in usage, especially during the latest decade, e.g. [10]. The common view is that test management is either a result of automating the test process [9] or a consequence of test execution automation [11]. Defining the set of test cases as quality criteria up front, i.e. by letting successful completion of all test cases in the set be the release criteria is an old idea [22], and defining test specifications as the bases for release is not new. Organizing the test specification in a way that reflects requirements is the most common approach, supported by standards [12, 13] that yield traceability [25], Connecting test cases to source file is also part of the regression paradigm in [27]. The notion of grouping the test specifications in direct relation not only to requirements, but also to the different software entities is new. Traceability is

always possible to set up manually or automatically. Technically this is not a problem, but creating the active relations is seldom done. Instead most organizations define this to be the prime work of the test manager. The new system Jazz [23] attempts an approach similar to ours, by requiring all other systems not already within the framework to add an *eclipse* plug-in. This solution might be a bit costly for some legacy tools. The use of the *eclipse* framework [26] is an improvement, since it is enabling tool integrations, which provides a partial solution to our problem.

Currently there is little support for test design in any test management system. Only some of the test design techniques are supported by tools for test case generation, e.g., evolutionary/genetic programming (GP) [17] systems and state transition/model based testing (MBT) [18] systems. This is today entirely an engineering task, even if formal approaches can solve specific task in such systems. Even a very small software component in our system will be at least around 200 000 lines of code, implying that the use of only GP or MBT as the main source of test design techniques is not sufficient for a quality test [16].

VIII. CONCLUSIONS

This paper demonstrates the importance of continuous data capture of test results in relation to traceability and failure detection, as a result of an automated test process. This is an important step towards a fully automated test management, through the following main features:

- A loosely-coupled integration of diverse tools, based on automatic extraction and synthesis of a set of measurements.
- Using test specifications as release criteria, i.e. the system will be ready for release when all tests have been successfully performed.
- Automatic traceability of failures to software components, achieved by enforcing test specifications to represent system entities.

By the development and use of a test management system (TMS) that can integrate a wide variety of external systems and thus automated the entire execution, correction and re-testing, we have demonstrated the feasibility and value of automation of test management. We believe we have contributed towards an autonomous solution, and thus minimized the work for test managers in the execution phase.

ACKNOWLEDGMENT

We would like to thank Ericsson AB and Ericsson Ltd for funding our work and for allowing us to publish these results. The Knowledge Foundation is acknowledged for funding this work through the SAVE-IT program.

REFERENCES

[1] Giruado, G., Tonella, P. and Basili, V. (ed). "Designing and Conducting an Empirical Study on Test Management Automation", Empirical Software Engineering, 8, Kluwer Academic Publishers, p. 59–81, 2003
[2] Gao, J. Z., Itaru, F. and Touoshima, Y. "Information Technology and Management" 3, 85–112, Kluwer Academic Publishers 2002
[3] HP Test Director https://h10078.www1.hp.com/ cda/ hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-127-24%5E1131_4000_100__
[4] IBM Rational(R) Quality Manager http://www-01.ibm.com/software/awdtools/rqm/standard/
[5] T-plan see http://www.t-plan.co.uk/
[6] Imbus TestBench, see http://www.imbus.de/products/ imbus-testbench/
[7] See http://www.opensourcetesting.org/testmgt.php/
[8] Rothermel, G., Harrold, M. J. "Analyzing regression test selection techniques" IEEE Trans. on Software Engineering, V. 22, Issue 8, pp. 529-551, Aug. 1996
[9] Pocatilu, P. "Automated Software Testing Process", Economy Informatics, no. 1, p. 97-99, 2002
[10] Eickelmann, N.S. and Richardson, D.J. "An Evaluation of Software Test Environment Architectures", IEEE Proc. of ICSE-18, 1996
[11] Berner, S. Weber, R. and Keller, R.K. "Observations and Lessons Learned from Automated Testing", IEEE Proc. of ICSE'05, 2005
[12] IEEE Std. for Software Test Documentation 829-1998
[13] IEEE Standard for Software and Systems Test Documentation 829, 2008
[14] MySQL v.5.1 see: http://www.mysql.com/
[15] Apache see: http://www.apache.org/
[16] Eldh, S., Punnekkat, S., Hansson, H., Jönsson., P: "Component Testing is Not Enough - A Study of Software Faults in Telecom Middleware", *Proc. 19th IFIP* Int. Conf. TESTCOM/FATES, Springer, 2007
[17] Wegener, J. Sthamer, H., Jones, B.F. and Eyres, D.E., "Testing real-time systems using genetic algorithms", *Journal of Soft. Quality*, Springer, Vol. 6 (2), June 1997
[18] Dalal, S.R.; Jain, A.; Karunanithi, N.; Leaton, J.M.; Lott, C.M.; Patton, G.C.; Horowitz, B.M: "Model-based Testing in Practice", Proc. of ICSE, IEEE 1999
[19] Agrawal, H. "Towards automatic debugging of computer programs", Thesis, Purdu University, 1991
[20] Failure Tracking tools see http://www.testingfaqs.org/t-track.html
[21] Auguston, M., Jeffery, C., Underwood, S., "A Framework for automatic debugging", Proc. of 17'th Int. conf ASE, IEEE, 2002
[22] Bruno, M., Canfora, G., Di Penta, M., Esposito, G., and Mazza, V., "Using Test Cases as Contract to Ensure Service Compliance across Releases", Springer Verlag, LNCS Vol. 3826, pp.87-100, 2005
[23] IBM Jazz Technological Platform, see http://www-01.ibm.com/software/rational/jazz/
[24] Basili, V.R., Boehm, B.,"COTS-Based Systems Top 10 list", IEEE Computer, Vol. 34, Issue 5, 2001
[25] Nebut, C., Fleury, F., Le Traon, Y., Jezequel, J.-M.: "Automatic test generation: a use case driven approach", IEEE Trans. On Soft. Engin. Vol 32, Issue 3, 2006
[26] eclipse, see http://www.eclipse.org/
[27] Wikstrand, G., Feldt, R., Gorantla, J., Zhe, W., and C. White. "Dynamic regression test selection based on a file cache an industrial evaluation". ICST, 2009
[28] QAtraq Professional, see http://www.testmanagement.com/
[29] SilkCentral Test Manager, Borland, see http://www.borland.com/us/products/silk/silkcentral_test/index.htm