Supporting Software Decision Meetings: Heatmaps for Visualising Test and Code Measurements

Robert Feldt, Miroslaw Staron
Computer Science and Engineering
Chalmers & University of Gothenburg, Sweden
Email: robert.feldt@chalmers.se, miroslaw.staron@gu.se

Erika Hult, Thomas Liljegren
RUAG Space
Gothenburg, Sweden
Email: (erika.hult, thomas.liljegren)@ruag.com

***Abstract*—To achieve software quality it is critical to quickly understand the current test status, its changes over time as well as its relation to source code changes. However, even if this information is available in test logs and code repositories it is seldomly put to good use in supporting decision processes in software development. The amount of information is often large, is time consuming to extract and hard to monitor. This case study shows how visualisation and correlation between software measurements can support improvement discussions. In particular, simple heatmaps were found to be effective to visualize and monitor changes and identify recurring patterns in the development of a space-bourn, embedded control system. Statistical analysis quantified the correlation between different sources of development data and heatmaps then effectively focused the attention of stakeholders to importants parts of the system. Here the visual analysis was focused on post-project, historical data but we discuss how early identification based on dynamic data analysis could support more effective analysis, planning and execution of quality assurance. Based on our findings we state requirements on such an online, visual analysis system and present a prototype implementation that can help software measurements better support value-based decisions in software development.**

I. Introduction

To control the stability and quality of software products during their development it is important with effective monitoring of both test outcomes, test progress as well as source code changes. Several previous studies have shown the importance of including change metrics in the prediction of faults and fault proneness [1], [2]. These code changes have a potential of directing test efforts or pointing developers to places in source code which might be risky [3] - risks which might lead to increased costs/effort. But the information also need to flow in the opposite direction; test information need to be connected back to the source code to support fault localization, debugging and improvement efforts [4], [5].

Identifying source code changes which might impact integration processes in software development projects is usually done a posteriori during post-mortem analyses or as a response to problems with quality, functionality or project schedule. Changes in source code are a normal phenomenon in software engineering - as the source code is expected to grow and improve - so it is hard to filter out the normal changes from the unexpected change patterns like defect-fixes. However, if complemented with the visualization of results of testing, these change patterns can be used as early warning system for quality risks. Combining multiple data streams, visualising them and allowing interaction with the visualisations has also been found helpful in ongoing research into visual analytics [6].

For software quality we have found three elements to be important: source code with its changes, test progress with the outcome of executing test cases, and changes in test cases themselves [7], [8]. The last element needs to be considered as software engineering practices allow updates of test cases during the development process due to such aspects as defects in test code or updated requirements. Therefore discussions should investigate all three sources of data. Preferably these three elements should be visualized together to allow parallel analyses and visual coupling of causes and effects.

Previous research in this area has focused on small-scale experiements, often not in actual industrial projects. In this paper we present a case study from RUAG Space AB in Gothenburg, Sweden with the goal to *identify early warning indicators which can inform stakeholders about potential problems with unexpected costs*. The results of the case study showed that a main cost and time driver is integration efforts and defects related to integration of software with the hardware platform. By using heatmaps we were able to show which source code elements have potentially the most significant impact on the integration process. They also found patterns in defect occurence which relates to inefficiencies in the testing processes. The heatmaps have been found to be very effective to focus discussions on the most relevant elements of source code and test progress without loosing the overall picture.

This paper is structured as follows. Section 2 presents the related work in the area of early warning systems for software development, heatmaps and integration/test planning. Section 3 presents the concepts of code churns and heatmaps. Section 4 presents the design of our case study and Sectoin 5 presents the results from it. Section 6 discusses threats to the validity of our results and Section 7 presents the conclusions from our work.

II. Related work

Ball and Nagappan [2] have shown that large change bursts and relative code churns (described in section III-A) are good predictors of reliability of large software products (evaluated on two software products: Microsoft Windows Vista and

Eclipse). Although the results were obtained by studying large products, we decided to adopt this method and evaluate it for small products at RUAG Space in the domain of embedded software.

Our intention was to build something similar to CRANE toolset [9] where data from a number of sources was used to make decisions about proposed fixes to the post-release defects in Microsoft Windows Vista operating system. Providing a similar system for pre-release defects with the focus on integration problems and cost control would allow teams to quickly identify places in the code which should be tested more rigorously. Our goal was in line with the work of Buse and Zimmermann [10] who intend to create a tool for supporting software engineers in daily decision making - based on a survey of 110 developers from Microsoft.

In our previous research we identified that Agile prediction of trends in defect inflows is appreciated by managers in software development [8]. However, the predictions which yielded best results in terms of accuracy were based on the existence of defects - in other words assumed that there are defects to start with. In this research we intend to use source code changes and test case changes profiles as means of forecasting defects/integration problems. Examples of this kind approach for large systems could be found in [11], [12].

In our previous work we have investigated the use of early warning systems at Ericsson with the purpose of identifying, monitoring and warning about potential bottlenecks in large Agile+Lean software development projects [13]. The early warning system at Ericsson was aimed at monitoring the work flow in software development program and notify the program management of building queues and limitations of the overall capacity of the system. The most relevant aspect was the use of heterogeneous metrics related only through process flows. This concept was also used in this research: the collected metrics are related to each other through process, not through mathematical correlations.

Buse and Zimmerman [14] conducted a survey with 110 designers, architects and project managers at Microsoft with the goal to explore information needs for software analytics at a large software development company. Among a number of insights regarding the need to simulate/predict software development outcomes, the survey showed that metrics like code clones, bug and failure anlyses and code changes are among the top information needs.

This strand of research is part of a larger trend where ideas and tools from *Visual Analytics* is applied on software and its related data. An overview of recent work can be found in Reniers et al [6]. Visual Analytics (VA) integrates data collection and statistical analysis with visualisation and interaction to support sense and decision making. The majority of existing Software VA (SVA) papers focus on source code and studies changes either at the file or project level. There is a lack of results that combine source code with test related data for support of more complex decision making processes.

III. Theoretical background

In order to address our research problem we combined two concepts – code churning from Microsoft Research and heatmaps as a means of visualization of large quantities of three dimensional data.

A. Code churn

In our work we use the concepts of code churn and change bursts as the metrics for identifying patterns in source code. These concepts have been defined and evaluated by Microsoft Research [2]. *Code churns* are defined as source code which was added, changed or deleted between two versions of software unit over time. In their original work Nagappan and Ball [2] have identified that relative code churns (i.e. code churns normalized by the size of software unit) are very good predictors of software reliability. Those parts of Microsoft Windows Vista and Eclipse platform which exposed large change bursts (i.e. frequent and large code changes) were found to carry a lot of risk to reliability.

In Microsoft's case the main predictor was *change burst*, which is a set of consecutive changes of a specified magnitude. This concept had two parameters: *code churn* and *number of consecutive changes*, whose values were found empirically.

As code churns were found to be good predictors of reliability, we use this concept as a predictor of problems with integration as unreliable software is more likely to expose problems under testing than the reliable software.

B. Heatmaps

The origins of this data visualization format can be traced back to the 19th century as described by Wilkinson [15]. Heatmaps are most suited for visualizing large sets of data with at least three dimensions. Given two dimensions of the data mapped to the rows and columns of a square graphics canvas each cell is then coloured according to the size of an indicator or metric or to show the strength of a statistical effect. This can help show clusters or patterns in the data over time or in different areas of the data. An example heat map is presented in Figure 1 which shows code churns for one project by mapping each file to a separate row, time (here at the resolution of weeks) to columns and the cell colour set according to the churn for that file at the given week.

Voinea et al used a heat map to visualize code changes in software projects on both file and project level [16]. For example, at the file level, each line of code is given a coloured, horizontal line indicating its change status for each version of the file. This creates a colour map as the versions progress to the right and lines of code down on the visual 2D map. Different visualisation choices both when it comes to the colours involved but also the scaling used made it more or less effective to analyse different types of source code evolution [16].

At the project level, Voinea et al plotted each unique file on separate horizontal lines and used colours to map, for example, authors involved in changes. The heatmap visualizations were also augmented with software metrics or linked

to code fragments shown along the axis of the heatmaps themselves. The implemented tool also supported interaction with the visualization to filter based on time or entity values and to select new metrics to correlate with. In a few, small case studies the tool was shown to support 'a quick assessment of the important activities and artifacts produced during development' and could also help detect larger architectural changes when multiple files was added or changed at the same time [16]. The users of the tool involved in the case studies were surprised by the ease with which they could identify the major events and contributors without any prior knowledge of the investigated source code.

Note that heatmaps is but one of the many possible visualisation techniques that can be employed for large data sets. Alternatives such as table lenses, hierarchically bundled edges and treemaps can provide even more information in a limited space and are included in recent SVA tools [6]. However, in our initial talks with company staff heatmaps were intuitively and quickly understood while more complex graphical solutions would require more explanation. This study thus focuses on the use of simpler and more traditional visualization tools.

IV. Design of the Case Study

The case study designed to address our research question was conducted in close cooperation with managers and engineers at RUAG Space AB in Gothenburg, Sweden. This business unit of a larger, multi-national corporation develops hardware and software for various space systems, e.g. satellites. Below we describe the industrial context in more detail and present the different phases of the case study.

A. Context

The context of this case study is RUAG Space AB (RUAG) a company based in Gothenburg, Sweden. RUAG was formerly known as SAAB Space AB but was acquired by the Swiss company RUAG Space in 2008. They have a long experience in the design, development and delivery of both hardware and software for computer and data handling products for space applications. The main product areas are data management systems, fault-tolerant computers and processor products, payload control computers, and small mass memories.

The software developed by RUAG for these computers is in the range from small boot software to full application software, but the main focus is on embedded, real-time software closely integrated with custom hardware also developed in-house. The software development process used is based on the ECSS standards, mixed with an integration driven development approach. In previous research collaborations with RUAG we have studied how to improve software development and make quality assurance more efficient within the quite strict process constraints of the ECSS standard [7], [17].

RUAG employs in total 360 people, of which about 35 work in the software unit. Typically up to five projects are developed simultaneously in varying team sizes of up to 10 people per team. The software is developed mainly in C but with some low-level parts written in assembler. The development process being used is incremental with between 12 to 25 increments per project. Initial increments are focused on base functionality while later increments integrates functionality into larger subsystems and sets of features. Since the hardware is commonly developed in parallel with the software the project planning, and in particular the distribution of functionality over the increments, is heavily influenced by when different hardware components will be available and ready for testing.

B. Case Study Phases

We designed the case study to follow a collaborative research approach, proven successful in our other research projects [18]. The process assumed that researchers from academia collaborated closely with practitioners in defining the problem, proposing/evaluating solutions and analyzing results. In detail, the different phases of the case study were:

1) **Workshop** with practitioners to understand the problem. The workshop resulted in defining the problem to be a need for predicting test effort based on source code, effort-related project data and test progress.
2) **One-on-one interviews** with a project manager and a technical leader to understand the software development and project monitoring practices at the company. The interviews established a common platform to start analyzing numerical data from the following sources: financial/effort reporting databases, test rigs, and source code repository.
3) **Data extraction and document analysis** from past project. In order to perform correlation analyses and to be able to understand cause-effect relationships we analyzed a past project. The project was representative for the projects at the company in terms of size and complexity of the problem, source code, or test base. The software development process followed by the past project was the standard, integration-driven process with multiple increments.
4) **Data visualization and analyses**. We visualized the data from source code repositories and test rigs using heatmaps, whereas we used standard line-plots for the financial data. The practitioners at the company assisted the researchers in understanding and filtering the data to avoid finding random dependencies in the data. The analyses should lead to identifying indicators that can foretell problems with late and large test effort.
5) **Workshop** with practitioners to evaluate indicators and identify the "best" way to visualize them. The discussions in the workshop showed that source code changes have an impact on test progress and that test progress and integration effort are significantly correlated. The discussions also showed that heatmaps are an efficient way of communicating the information within the company.
6) **Prototype online analysis system**. The evaluated indicators and visualization methods were used in a new project by two master students (of software engineering)

whose task was to build a measurement system for monitoring and controlling of software development projects [19].

The workshops were held at multiple times throughout this project and involved between 3-5 engineers having different roles throughout the company, from managers and test leaders to supporting roles. Interviews were held with two researchers present and one interviewee. The interviews were semi-structured where the researchers had a set of pre-defined topics or questions that were probed but where further detailing and question were flexibly adapted based on the role and experience of the interviewee and their previous answers. The researchers also had continuous contact with the integration lead while sitting at the company. The whole project team had continuous steering group meetings throughout the project.

V. Results and Analysis

The results and analysis presented in this section are structured into three parts - initial feasability analysis and refinement of industrial requirements, data visualisations and analysis, and the design of a prototype early warning system. The first part corresponds to the explorative part of our study (i.e. phases 1-3 of the case study as described in IV-B) while the latter parts describes the detailed analysis (phases 4-5) and the prototype system (phase 6) developed at the company. The latter part also discuss the impact on the company's operation.

A. Initial analysis and data extraction

In the first two steps of the research process presented in section IV-B we narrowed the research problem to mining multiple data sources to build an early warning system to forecast large integration and test costs at the company. The workshops resulted in narrowing the problem and project focus to the most important areas of software development. In discussions staff agreed that:

- the major cost driver in late software development phases is the integration effort which is frequently high,
- the major integration effort driver is the fact that there are problems with failing test cases and the need to fix defects before progressing with the test and thus integration process, and
- the potential reason for failing test cases could be late access to hardware and the resulting late changes to the source code of the product.

Since the software and hardware is developed in parallel by two different departments at the company the integration activities are necessarily in focus. Furthermore, for dependability reasons the software development standards in use in the space industry requires having a separate test team [7]. Even though the departments have done considerable unit level testing and also tested the integration of their respective components internally, full system integration when running the system tests is complex enough that many problems will not surface until this time. To address this situation the systems are developed in a sequence of iterations where subsets of the full software functionality are integrated on a special hardware test rig. The test rig is connected to a computer running the test controller that, in turn, executes the system tests selected for the current iteration. The test rig also runs regression tests from previous increments.

In order to understand if the current notions of the development staff and managers about the problems reflected the actual problems we investigated different, existing data sources. Having a data-centric approach was essential, in particular given the particulars of the space industry/domain and the objective focus of both developers and managers. It was deemed unrealistic to rely on manually updated data or subjective judgement; a focus on existing and raw data was essential to get acceptance and create understanding.

Five main sources of data was identified: a requirements tool (containing requirements, high-level test specifications and traceability links between them), source code repository, test code repository, in-house economical tool (containing project/task planning and effort data), test rig (with test logs of all test executions). Since the requirements and their possible churn was thought to be of less importance for the integration problems at hand, and since the requirements tool was not so simple to connect to for data extraction purposes the present study did not focus on it. In contrast, the source and test code repositories used standard, albeit different, repository tools from which history logs could more easily be extracted. In particular, this was true for the source code which used ClearCase[1]. Extraction scripts was created to extract relevant data from the commit logs. The test rig is a custom, in-house developed Windows PC application that logs all testing activities in a custom text file format. Scripts were developed to automatically copy the test log files and extract information about the date, time, test case and detailed information about any assertions that failed, or not, during the execution of individual tests. From the economical tool the effort spent on different development activities on a weekly basis could be extracted. This was the only data with a subjective element to it since it is ultimately based on individual time reports from each staff member. Furthermore, the development activities are quite coarse-grained with typically 3-5 different activities per independent team over the course of a project.

B. Data visualisations and analysis

After having extracted data about source code changes/churn, testing and test failures, as well as project effort per week, the subsequent two phases of the case study analysed and visualised the data. Many different metrics and combination of metrics was considered as well as a multitude of visualisation techniques. Statistical analysis of different sorts were also applied during this process. Results were presented in workshops in which the most revealing and understandable visualisations and correlations were further discussed and compared. Below we describe the main results of these steps in more detail.

[1]In later projects, RUAG have transitioned to using Subversion for both source and test code which would make future data extraction even simpler and more consistent

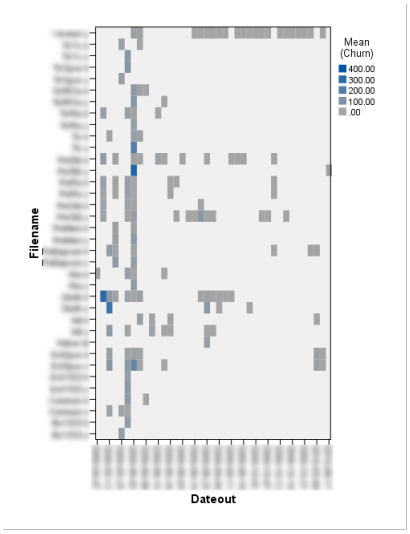

Fig. 1. Heatmap showing source code changes (code churns) per week for a subset of the project files and duration.

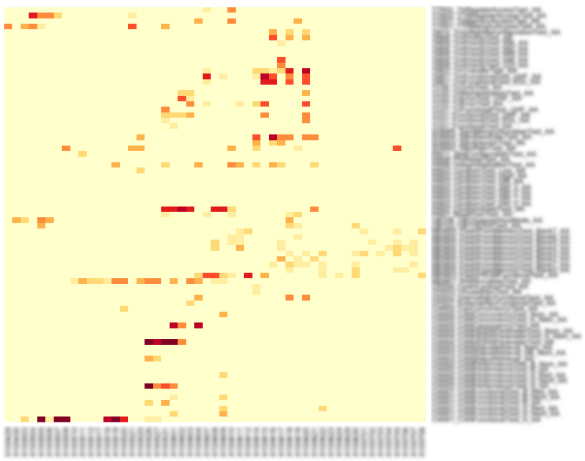

Fig. 2. Heatmap of the (log) number of test failures over time for different system test cases over a 2+ month period. Blurring for confidentiality reasons.

Heatmaps of the code churn per file over time lead to several important discussions and realizations. Figure 1 shows an example, here using a heatmap produced in the statistical tool SPSS. Here, rows in the heat map denote files in one source code package while columns denote the dates (here in one week per column resolution) when changes were made. The darker the color of a cell, the larger the code churn for that file package and week.

The code churn heatmap in Figure 1 exposes a number of interesting patterns:

- Vertical 'lines': a number of files have been changed during one date. This pattern indicates such events as baselining, reviews, deliveries, etc. in the project where a number of files are modified.
- Horizontal 'lines': one file was changed consecutively over a longer period of time. This pattern could indicate that intensive development was done over a longer period of time but could also indicate that there were continuous adaptations and changes to the file package. This pattern calls for checking whether this file was tested sufficiently.

Through the plotting of code churn heatmaps and workshop discussions around them we learned that a critical capacity of time-based heatmaps is that the timeline can be connected to major project events. Staff frequently had to 'translate' from the timeline in the heatmap to the time view of the project they had in their perception; even though the project events that were considered essential differed depending on the staff member and their roles key event like the start and end of the development increments and their review status was considered important by many.

We further learned that for a heatmap to be truly useful it needs to support different filtering and grouping of items to be mapped. The static visualisation tools that we used in steps 4-5 were too limited to dynamically allow more focused interaction with the data. Since the patterns that can be seen and thus analysed in the heatmap depends on the vertical grouping and granularity of source code files that are included inflexibility in choosing them can limit the patterns that are found.

Heatmaps were also effective in analysing patterns and trends in the test outcomes. Figure 2 shows a test outcome heatmap where the logarithm of the number of failed assertions per test case is the colour which is mapped for each test case (row) and day (column) where there was testing. This time the heatmap was produced in the statistical program R. The logarithm was essential to create a more realistic scale for the test outcomes of the investigated project. For one the test cases are typically long with long series of assertions that test multiple aspects of both the software and its integration with the hardware given a certain system set up. But these assertions are not independent, rather when one fail there are many common-mode causes that will cause many other assertions to also fail. However, we could not simply normalize this to the test case failing since we do not have good models or knowledge of these co-failing assertions. The logarithm was found to give a reasonable mapping of the actual failure behavior as perceived by testers, but the actual scaling and colour mapping of heatmaps might vary between projects or at least between companies, and thus warrants future investigation.

A number of patterns can be seen also in this test outcome heatmap, Figure 2. Since the mapped test cases are sorted into related sets of test packages rows that are close to each other also typically has a semantic or direct relation. However, even so there is variance between the behavior of individual test cases within different test packages. As an example, the low fourth of the rows all correspond to tests of a specific bus handling software. But while the bottom-most test case is executed early in the given time span there is a clear line at which other test cases in the same package start failing. In discussions this was found to relate to dependencies among test cases or to when specific hardware components were

available. For other test packages the failure behavior is more evenly spread out over time.

There is also a difference in the time evolution of test cases. While the non bottom-most test cases in the bottom test package fail heavily for a short time period and then remain stable and without subsequent failures, test cases just below the middle of the heatmap have a less distinct failure behavior that is spread out over long stretches of time. The latter test cases were related to software for new hardware components that were new to the company. In workshops the managers confirmed that they had been the cause of much concern and effort during the project. Thus it is not self-evident which patterns are clearly negative or not; what might initially look more 'benign' in that few failures are seen but spread out over long stretches of time might indicate more far-reaching and costly problems than intensive failure periods after which initial problems are then rectified and further failures avoided.

A major revelation from the discussions around test failure heatmaps was the apparent lack of traces of the increment-based development and integration approach. Essentially, even though earlier increments have been tested, reviewed and 'closed' some of their associated tests could still fail at later times. Even though this is to be expected since it is typical of any integration efforts it was unexpected that so few but time-limited clusters of failing test packages could be identified. In workshops it was decided that having such heatmaps during a project could provide this information and insight at a much earlier time and thus act as an early warning system of subsequent problems. However, again, the non-interactive and rather inflexible visualisation capabilities of R and SPSS would not be enough to create the kind of informed and focused discussions of the workshops without considerable manual efforts. Since these efforts would also be needed on a daily basis they were unlikely to be performed; a more dynamic and automated approach was needed.

For visualising the effort data, normal time series graphs was sufficient and heatmaps did not add any value. Probably this is due to the fact that the effort data contains a few summary categories, based on the main project activities. This is different for the code churn and testing data where the number of items to be mapped is typically large. Stacked graphs of the effort per activity over time was useful both in showing the total effort and its components.

In order to verify the staff notions and claims about correlation of failed test cases and integration effort we collected data from the financial/project management systems and test progress and outcomes. Plotting them on the same diagram resulted in figures like 3 where the dotted line is the integration effort (in person-hours) and the solid line is the number of failed test cases per week. The double line at the bottom is the number of new revisions in the code repository at the same time.

The Pearson correlation coefficient was 0.73 which supported that the experts' suggestions that test problems is a major factor in increasing integration efforts and costs. The diagram also shows that the three data series - effort, failing test cases and changes in the code - are related. The peak in week w8 shows that the integration was indeed costly at the same time as we can observe testing problems, which were also supported by working with the source code. The technical leader in the project and the manager responsible for that product could pinpoint which part of the product were problematic at that time and why - which we interpreted as a empirical validation of the representational property of our measures.

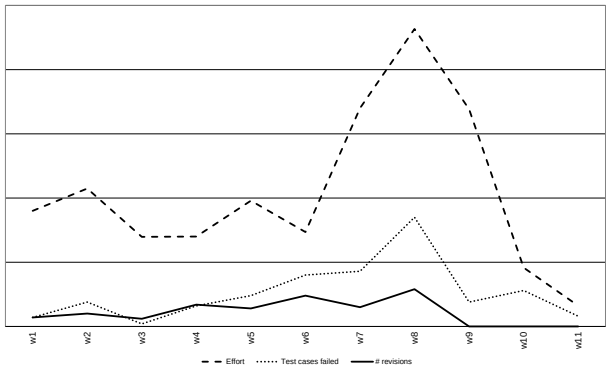

Fig. 3. Total project effort, number of test failures and number of revisions per week over the course of a project.

Exploring the data and plotting it on simple diagrams provided a solid basis to understand the underlying phenomena in software development processes at the company and helped to pinpoint problem areas. However, these visualizations did not provide the stakeholders with the ability to quickly react in the face of incoming problems. For example plotting the total number of all revisions per week was an indicator that this could be an underlying cost driver together with the failing test cases. However, in order to react, the stakeholders needed insight on which files were changed and which test cases were the most problematic. That knowledge was necessary for the stakeholders to make decisions abour prioritizing test effort, resource allocations or understanding the problematic parts of the product under development. Based on these findings we used heatmaps to provide stakeholder with the early warning systems.

C. Early warning system prototype

The results from the initial analysis and from the visualisations and correlation analysis lead to a number of conclusions.

- Heatmaps is an excellent way to visualize large data sets and identify complex and 'distant' connections.
- By combining heatmaps with group discussions a better understanding can be reached for high-level patterns in how development and testing is currently carried out.
- Complex dependencies makes it hard to close some integration steps which leads to delays in both the testing

and development processes.

- It is hard to know exactly which visualisations, metrics and analysis that support useful analysis and sensemaking; the number of alternatives and combinations is so large that several needs to be 'screened' before a top list can be produced.
- Code churn, test outcome data and project effort are essential and should be collected but requirements stability information might also provide additional analytic power
- A software improvement project like this is as much a learning and teaching opportunity for the company as it is a way to develop concrete tools and support; the sensemaking that visualizations plus discussions create is important to be able to change how employees actually work.
- The visualisations and support for analysis must be available during a project; post-mortem analysis is not enough since each project have unique characteristics and generic process improvements that fits all projects are unlikely to exist.

Based on these conclusions from the first steps of the case study it was decided that an Early Warning System for integration problems is feasible and that a prototype measurement system to enable this should be developed. Compared to the previous analyses and visualisations such a system would dynamically present a current view of a running project. This is critical not only to go from understanding of the problems to having a chance to solve them within a running project (i.e. the early warning), but also in that daily and wide-spread access to key project data visualisations would help create support among staff at large and thus make changes more easy to motivate and to perform. Furthermore, developing a framework to perform data collection and analysis dynamically, during a project would make it much more lightweight and cheaper to try new correlation analyses, metrics, as well as visualizations.

Even though requirements data was not included in the analyses it was investigated in parallel and also promise in predicting integration problems. Basically, even though requirements are fixed at an early stage, when there are requirements changes it might heavily affect integration testing and increase costs. It might also be a more direct indicator of later problems since it can be a common cause in both source code changes, test code changes as well as failing test cases. This further indicated that an online measurement system, that can handle multiple data streams from many and different sources, is needed.

Based on this, the last step of the case study was the design, installation and evaluation of an early warning system. The basic design is to have specific data probes that can be installed at each data source (such as the source and test code repositories, financial system, test management system, etc) that all report to a database server to which an analysis and visualisation system is connected. Figure 4 shows the prototype systems dashboard that integrates key, online project information in a single dashboard view for easy access during stand-up meetings.

VI. Validity of Results

The main threat to *internal validity* is the fact that parts of the investigated development projects were ongoing in parallel to our study. For the development of the prototype system this posed a threat that our study intervened with the proceeding of the project by providing the team with additional knowledge, thus confounding the results of the study. Even though this is a threat, it is inevitable in these types of case studies and affecting development practice is also part of the goal.

The main threat to the *construct validity* is the choice of metrics for source code and test progress. In this study we do not use metrics like code coverage or code complexity, which can lead to us having and giving an incomplete view on the source code or test processes. However, as the main metrics used were found to be correlated with the main focus of improvement efforts (cost/effort) we do not consider this an important threat; completeness would go against the reasons for focusing on the main causes and avoiding information overload. Our interviews and initial workshops also supported our claims that the metrics used in our study is relevant for the studied phenomena.

The fact that we only investigate one company is a threat to *external validity* and since we only study one project with a relatively small code base this is also a threat to *conclusion validity*.

VII. Conclusions

Ensuring a high quality of software is a complex task, especially as systems grow more complex and involve more people and more diverse technologies. With more mature tools at their disposal companies nowadays have many sources of data to measure and better understand their development strengths as well as areas where they have improvement potential. However, to really get information and insight from these multiple streams of data tools are needed to summarize and give overview as well as identify patterns and trends in the extracted data. This case study have focused on how visual analytics and simple statistical tools can be applied to better understand the integration and testing efforts in a company developing embedded, real-time control software.

Through a multi-step case study performed on site at the company we refined the most relevant improvement areas, identified data sources, created extraction scripts and then used different visualisations, graphs and statistical analyses to make sense of the data. In workshops with engineers and managers at the company the created visualisations were then further analysed and discussed. Throughout this process we saw that simple heatmap visualisations mapping key, but raw, data over time helped engineers and managers identify patterns and understand development and testing behavior. Often the insights created was in line with expectations but in other cases the identified patterns and trends was unexpected and indicated improvement areas.

In the present study the visual analytics was applied off-line, for historical project data. Based on the discussions in the workshops and the understanding the company wanted an

Fig. 4. Parts of prototype system dashboard with heatmaps integrating information about both source code changes and test outcomes.

online, dynamically updated visualisation system. In the final phase, we detailed requirements and developed a prototype such a system. The developed system is not a stand alone system which automatically warns when certain pre-specified limit values are reached, rather it makes the visual summaries and analysis available to daily meetings at the company and thus makes it possible for the discussions among project participants to happen during the project rather than post-project. This shows a lot of promise in supporting key decisions about where to focus testing and quality assurance efforts as well as when testing can be stopped. A key result is that these kinds of systems must work in unison with the people involved in development, and support the type of questions they have, to support value-based management and decisions in software development. Future work will focus on extending and adapting the analysis framework to and then conducting further evaluation in more companies and context.

Acknowledgment

The authors thank RUAG Space AB and the Swedish Space Agency for the collaboration and financing the project. We also acknowledge the thesis work of Tobias Alette and Viktor Fritzon [19] which affected the present study.

References

[1] R. Bell, T. Ostrand, and E. Weyuker, "Does measuring code change improve fault prediction?" in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*. ACM, 2011, pp. 2–10.

[2] T. Ball and N. Nagappan, "Use of relative code churn measures to predict system defect density," in *27th International Conference on Software Engineering*, 2000, pp. 284–292.

[3] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, 2005.

[4] J. Jones, M. Harrold, and J. Stasko, "Visualization of test information to assist fault localization," in *Proceedings of the 24th international conference on Software engineering*. ACM, 2002, pp. 467–477.

[5] R. Wettel and M. Lanza, "Visual exploration of large-scale system evolution," in *Reverse Engineering, 2008. WCRE'08. 15th Working Conference on*. IEEE, 2008, pp. 219–228.

[6] D. Reniers, L. Voinea, O. Ersoy, and A. Telea, "The solid* toolset for software visual analytics of program structure and metrics comprehension: From research prototype to product," *Science of Computer Programming*, 2012.

[7] R. Feldt, R. Torkar, E. Ahmad, and B. Raza, "Challenges with software verification and validation activities in the space industry," in *Proceedings of the Int. Conf. on Software Testing*. IEEE, April 2010, pp. 225–234.

[8] M. Staron, W. Meding, and B. Söderqvist, "A method for forecasting defect backlog in large streamline software development projects and its industrial evaluation," *Information and Software Technology*, vol. 52, no. 10, pp. 1069–1079, 2010.

[9] J. Czerwonka, R. Das, N. Nagappan, A. Tarvo, and A. Teterev, "Crane: Failure prediction, change analysis and test prioritization in practice - experiences from windows," in *International Conference on Software Testing*, 2011, pp. 357–366.

[10] R. P. L. Buse and T. Zimmermann, "Information needs for software development analytics," *Microsoft Research Technical Report*, vol. MSR-TR-2011-8, 2011.

[11] T. M. Khoshgoftaar, B. Cukic, and N. Seliya, "Predicting fault-prone modules in embedded systems using analogy-based classification models." *International Journal of Software Engineering and Knowledge Engineering*, vol. 12, no. 2, pp. 201–222, 2002.

[12] T. M. Khoshgoftaar, B. B. Bhattacharyya, and G. D. Richardson, "Predicting software errors, during development, using nonlinear regression models: a comparative study," *IEEE Transactions on Reliability*, vol. 41, no. 3, pp. 390–395, 1992, 0018-9529.

[13] M. Staron and W. Meding, "A method for identifying and monitoring bottlenecks in lean software development projects," in *International Conference on Product Oriented Software Process Improvement (PROFES)*, 2011, p. n/a.

[14] R. P. Buse and T. Zimmermann, "Information needs for software development analytics," in *Proceedings of the 34th International Conference on Software Engineering*, June 2012.

[15] L. Wilkinson and M. Friendly, "The history of the cluster heat map," *The American Statistician*, vol. 63, 2009.

[16] L. Voinea, J. Lukkien, and A. Telea, "Visual assessment of software evolution," *Science of Computer Programming*, vol. 65, no. 3, pp. 222–248, 2007.

[17] E. Ahmad, B. Raza, R. Feldt, and T. Nordebäck, "Ecss standard compliant agile software development - an industrial case study," in *In Proceedings of the National Conference for Software Engineering (NSEC 2010)*, 2010.

[18] A. Sandberg, L. Pareto, and T. Arts, "Agile collaborative research: Action principles for industry-academia collaboration," *Software, IEEE*, vol. 28, no. 4, pp. 74–83, 2011.

[19] T. Alette and V. Fritzon, "Introducing product and process visualizations to support software development," Master's thesis, Chalmers University of Technology, Sweden, 2012.