Visual patterns in issue tracking data

Knab, P ; Pinzger, M ; Gall, H C

Abstract: Software development teams gather valuable data about features and bugs in issue tracking systems. This information can be used to measure and improve the efficiency and effectiveness of the development process. In this paper we present an approach that harnesses the extraordinary capability of the human brain to detect visual patterns. We specify generic visual process patterns that can be found in issue tracking data. With these patterns we can analyze information about effort estimation, and the length, and sequence of problem resolution activities. In an industrial case study we apply our interactive tool to identify instances of these patterns and discuss our observations. Our approach was validated through extensive discussions with multiple project managers and developers, as well as feedback from the project review board.

DOI: https://doi.org/10.1007/978-3-642-14347-2_20

Visual patterns in issue tracking data

Knab, P; Pinzger, M; Gall, H C

Visual patterns in issue tracking data

Abstract

Software development teams gather valuable data about features and bugs in issue tracking systems. This information can be used to measure and improve the efficiency and effectiveness of the development process. In this paper we present an approach that harnesses the extraordinary capability of the human brain to detect visual patterns. We specify generic visual process patterns that can be found in issue tracking data. With these patterns we can analyze information about effort estimation, and the length, and sequence of problem resolution activities. In an industrial case study we apply our interactive tool to identify instances of these patterns and discuss our observations. Our approach was validated through extensive discussions with multiple project managers and developers, as well as feedback from the project review board.

Visual Patterns in Issue Tracking Data

Patrick Knab[1], Martin Pinzger[2], and Harald C. Gall[1]

[1] Department of Informatics, University of Zurich, Switzerland
knab@ifi.uzh.ch, gall@ifi.uzh.ch
[2] Department of Software Technology, Delft University of Technology, The Netherlands
m.pinzger@tudelft.nl

Abstract. Software development teams gather valuable data about features and bugs in issue tracking systems. This information can be used to measure and improve the efficiency and effectiveness of the development process. In this paper we present an approach that harnesses the extraordinary capability of the human brain to detect visual patterns. We specify generic visual process patterns that can be found in issue tracking data. With these patterns we can analyze information about effort estimation, and the length, and sequence of problem resolution activities. In an industrial case study we apply our interactive tool to identify instances of these patterns and discuss our observations. Our approach was validated through extensive discussions with multiple project managers and developers, as well as feedback from the project review board.

1 Introduction

Most software development organizations have issue tracking systems in place where various information about the software development process is stored. Also with all the information requirements from CMMI and ISO certifications, there is a huge amount of data available for investigation. Applied correctly, the insights from such an investigation help to improve the development process.

However the data gathered from software projects have some properties that makes it hard to get meaningful results from traditional statistical analyses. The same properties are also deterrent to more advanced data mining approaches.

One of the biggest problem for statistical approaches is the severe right skewed distribution of the data gathered from software projects: Using problem reports (PRs) as a starting point we usually see a huge amount of problems that are fixed in a short amount of time, take only little effort, and affect only a small amount of source code modules. Calculating averages, comparing distributions, etc. does not yield the required insights. In the end, what does it mean if we can improve the average resolution time from 1 hour to 50 minutes, or even lower. It is very unlikely that the accuracy of the entered measures, *e.g.*, resolution time or effort, was that accurate to begin with. But on the other hand, finding the reasons why certain problems took more than 100 hours to be resolved, and present common patterns for these outliers, can help a manager to significantly improve the process.

Another problem with all the data gathered in software project repositories is quality. If the data is not used and discussed on a regular basis inside the project team, the accuracy of the human entered data is at best questionable. Entering accurate information is time consuming and does not yield direct results, *i.e.*, no new features get implemented. It is therefore difficult to motivate developers to dedicate enough time to this administrative tasks.

Given the difficulties stated above, we developed a visual approach that is strong in outlier detection, supports the discovery of common patterns, and helps in promoting discussions among the participating stakeholders regarding process development metrics. It also provides facilities to detect flaws in the entered data. The approach is based on the Micro/Macro Reading idea [1] and provides a combination of Overview and Details-on-demand where the big picture as well as detailed problem report (PR) information are displayed at the same time. This is a very important property of our approach and helpful for visual pattern detection.

In [2] we already presented the interactive features for data exploration. In this paper we present and discuss generic visual patterns found in effort measurement and problem life-cycle data. We also apply our visualization technique to an industrial case study conducted in the EUREKA/ITEA[3] project SERIOUS[4].

Discussing the results of our analysis with a group of project managers from our industrial partner, managers and researchers from the other partners, as well as the ITEA review board, which also consist of members from the industry, we have strong anecdotal evidence that our approach stimulates discussion, is easy to understand, and provides valuable insights into the data buried in software project repositories.

The remainder of this paper is organized as follows: in the next section we describe the data of our case study. In Section 3 we show the relevant elements of our visualization approach. In Section 4 we present typical patterns found in project data, and in Section 5 we present our case study and discuss the detected patterns. In Section 6 we present related work and then conclude in Section 7.

2 Industrial Data Set

In this section, we describe the data from our case study in more detail and establish the background for the next section where we present our visualization building blocks.

Out of the many attributes associated with a problem report we focus mainly on the following four:

- *estimatedEffort*: The estimated total effort in person hours to fix the problem
- *actualEffort*: The actual total effort in person hours

[3] http://www.itea2.org/

[4] Software Evolution, Refactoring, Improvement of Operational & Usable Systems

– *analyzer*: The person responsible to analyze the problem and estimate the effort
– *priority*: The priority assigned to the problem, possible values are: low, medium, high, and top

The *estimatedEffort* is an estimate done by the *analyzer* who can be either a specially assigned person or the problem report owner herself. The *actualEffort* is the total effort actually used to fix the problem. It includes the analysis, the resolution, as well as the evaluation effort.

These attributes are relevant to improve the planning and resource allocation. If we know which PR needs the most effort, we can allocate more resources to the resolution of this PR. Here *estimatedEffort* comes into play, and also the difference between estimated and actual effort, because the quality of the estimates decides how well the effort prediction works. Based on priority more or less resources might be allocated to certain PRs.

In addition to the problem report attributes, we also extracted life-cycle data from the log files. The log files contain all changes to all the PR fields including status changes, *e.g.*, a PR is changed from *submitted* to *in_analysis*. All the PR (problem report) life-cycle states are shown in Figure 1. The states connected by thick solid arrows constitute the regular path a problem report travels from submitted to concluded. The other states (gray in the figure) are possible but less frequent. States can also be skipped or reached repeatedly, indicated by the dashed arrows.

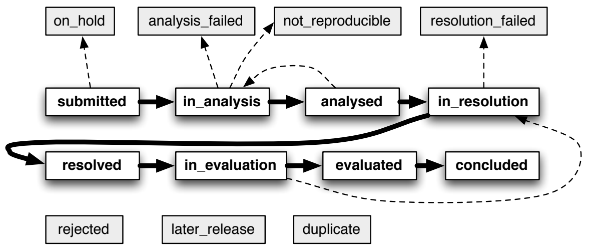

Fig. 1. Life-cycle States of Problem Reports

In an analysis, the transitions and the durations in the life-cylce of a PR allows to answer various questions concerning the relation of effort and time spent in the various phases, *e.g.*,*in_analysis* or *in_resolution*. Also the relation between priority and, for example, the duration of the *submitted* phase can give us interesting insights. In the case of a high priority PR we would expect a short *submitted* phase, since work was started as soon as possible.

3 Visualization Building Blocks

We present three relevant views of our visualization approach and the generic patterns that can be observed.

For the visualization of effort measures, we use **Polymetric Views** [3]. In Figure 2 the basic concepts of our visualization are shown: the width of the boxes is determined by the value of the *estimatedEffort* and the height by the value of the *actualEffort*. The effort measure is the sum of all efforts that were exerted to resolve the issue described by the problem report.

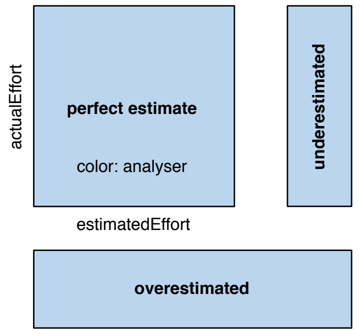

Fig. 2. Polymetric Effort Shape Patterns

With this mapping we can get a quick and effective overview over the quality of estimates: balanced estimates (square boxes) constitute problems that were estimated accurately with respect to the actual resolution effort needed; underestimated (boxes that are thin and tall), and overestimated (boxes that are short and broad) PRs can also be spotted easily.

To visualize the relative duration of process steps we use a **Pie Chart Visualization**. In Figure 3 we show a single pie with the mapping to the four process steps: *submitted, in_analysis, in_resolution, in_evaluation*. The size (*i.e.*, the area) of the pie is mapped to the total time from the creation of the PR until it was closed.

Finally our **Phase View Visualization** in Figure 4 is concerned with the process life-cycle sequence. This view depicts the sequence of the process steps, and allows an investigator to spot cycles and other exceptional situations.

4 Visual Patterns

In this section we present some generic visual patterns that we found in the course of our investigations. In the next section we will then show the actual instances of these patterns in the data of our industrial case study. The first part of this section presents the patterns for the **Polymetric Views**, followed by the patterns for our **Pie Chart View**, as well as our **Phase View** in the second part.

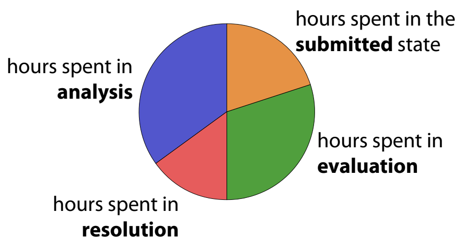

Fig. 3. Pie Chart Visualization Showing Process Step Lengths

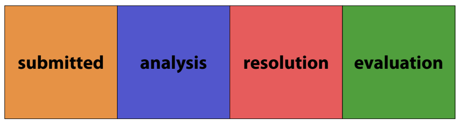

Fig. 4. Phase View Visualization to Show Process Step Sequences

4.1 Effort Estimation Patterns

For all effort estimation patterns that we present in this paper, we mapped the color of the boxes to the analyzer. With this mapping we can assess the behavior of individual analyzers.

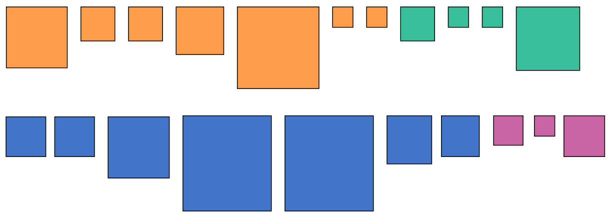

Fig. 5. Perfect Effort Estimates

In Figure 5 we see the pattern that we strive for. All the estimates of the four analyzers are perfect squares, *i.e.*, estimated and actual effort are the same. The problem with such a pattern, should it occur in actual data, is, that it is, most certainly, to good to be true. Therefore if one sees only or mostly squares this would be a strong indication that something with the data entry is wrong. If the picture looks like in Figure 6 the perfect estimates of the analyzer that was mapped to the blue boxes stand out. A manager should investigate the reason for this unlikely pattern.

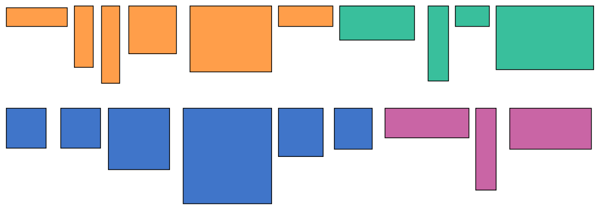

Fig. 6. Cheater

Another pattern that indicates a flaw in the data entry is shown in Figure 7. In picture d) *always the same*, all the boxes have the same width which means, that the entered estimate was always the same. This is again a strong indication that the effort estimation was not done carefully or something else went wrong. Again, the manager responsible for the estimation process should talk to the analyzer and improve the situation.

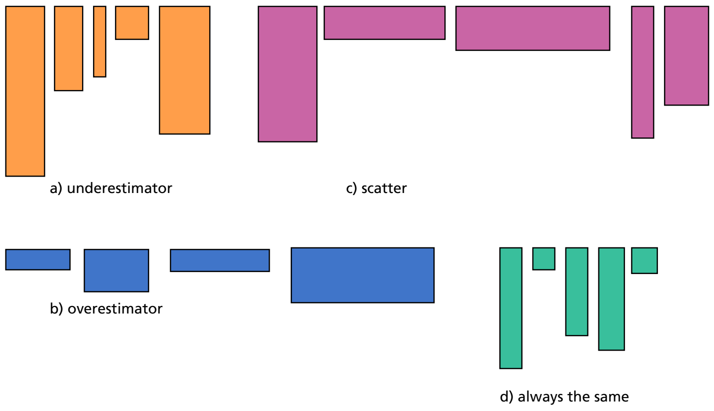

Fig. 7. Visual Effort Estimation Patterns

The patterns a) *underestimator* and b) *overestimator* provide probably the best information to improve the performance of certain analyzers. These analyzers show a strong tendency either towards underestimation or towards overestimation. With such a clear tendency it should be easy to correct their estimation behavior and improve the quality of the estimates. The pattern c) *scatter* shows a problem that is harder to correct since the effort estimations deviate in both directions, without a strong tendency that can easily be corrected. Therefore the only way to improve the quality of the estimates is to train the analyzer. Another explanation for the inaccurate estimates could also be that this particular analyzer just got the problems which are the hardest to analyze.

Similar considerations apply also to the patterns a) *underestimator* and b) *overestimator*. There are always multiple possible explanations for a weak estimation performance and it is necessary to investigate further, and especially discuss the initial findings with the affected engineers.

The primary goal of our visualization approach is not to asses the performance of analyzers but to help improve the software development process and find unwanted patterns that can then be corrected. If a correction is not possible then at least more knowledge about dependencies in the software process can be gained, which should than lead to improvements in future projects.

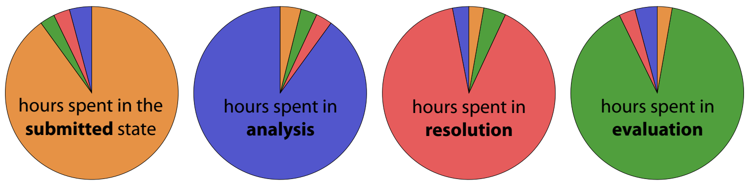
hours spent in the **submitted** state
hours spent in **analysis**
hours spent in **resolution**
hours spent in **evaluation**

Fig. 8. Life-Cycle Duration Patterns

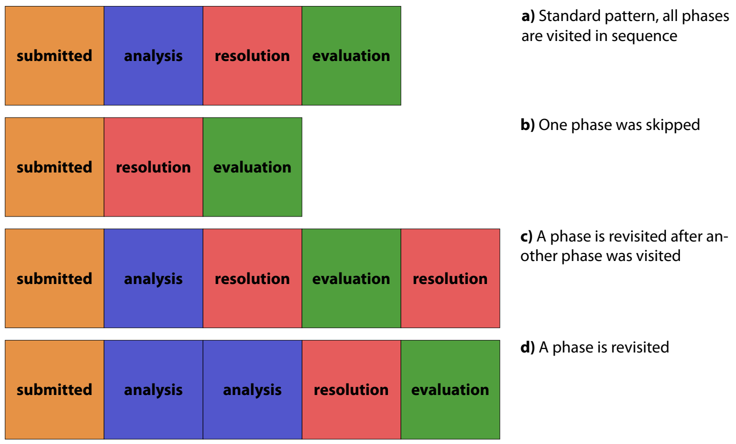
submitted
analysis
resolution
evaluation
a) Standard pattern, all phases are visited in sequence
submitted
resolution
evaluation
b) One phase was skipped
submitted
analysis
resolution
evaluation
resolution
c) A phase is revisited after another phase was visited
submitted
analysis
analysis
resolution
evaluation
d) A phase is revisited

Fig. 9. Life-Cycle Sequence Patterns

4.2 Process Life-Cycle Patterns

We present two visualizations for life-cycle patterns. One, the **Pie View** is mainly concerned with dominating process phases, *i.e.*, process phases, such as evaluation, that dominate the overall problem handling duration. In Figure 8 we depicted the possible domination patterns. The **Phase View** is then concerned with life-cycle step sequences. In Figure 9 we see the basic sequence patterns.

In combination with priority measures, the patterns provide information to improve the process and speed up the handling of high priority items. This can also be seen in the discussion of our case study in the next section.

5 Case Study

The case study is based on a five year multisite project in the consumer electronics domain. The issue tracking repository contains approx. 20'000 problem reports (PRs) that were handled by 368 distinct *analyzers*.

In Figure 10 we configured a view that groups and colorizes PRs according to the analyzer that did the estimation. Looking for patterns, we can see that there is a mix of estimation errors as well as some fairly well estimated PRs. There are instances of all the presented effort estimation patterns. We have highlighted an example for every pattern in Figure 10, but one can easily spot other occurrences for other analyzers.

Discussing the highlighted cases we see a), b), and c) where the main concern is to improve the quality of the estimates, but we have also the cases d) *always the same* and e) *cheater*. For d) and e) it might be advisable for a manager to talk to the corresponding analyzers and maybe also take the estimation performance from other projects into consideration. Since for d) the average actual effort is not that big, the impact on the overall project efficiency might be neglected. But for e) not only the perfect estimates are suspicious, but the actual effort measures are among the biggest in this display, and therefore the impact on the overall estimation quality is significant.

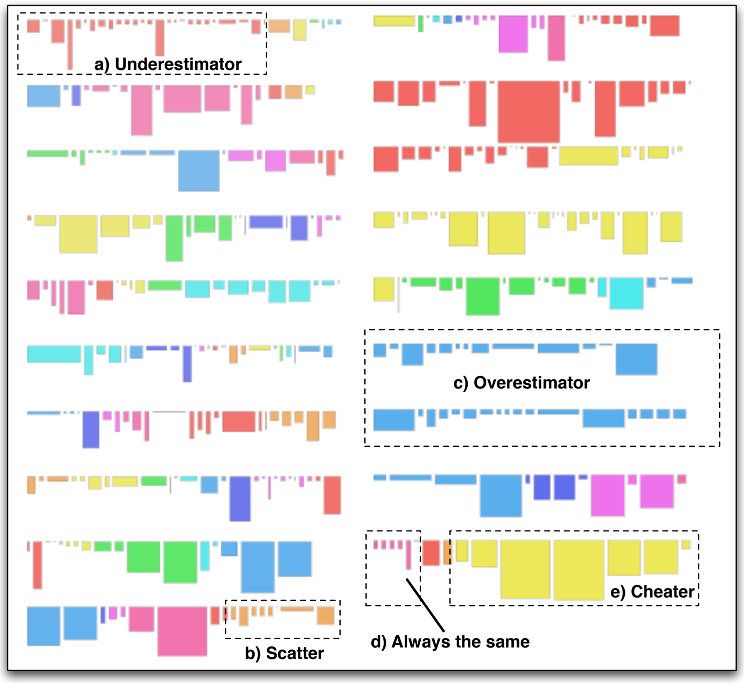

Fig. 10. Effort View on PRs Grouped and Colored According to the Analyzer

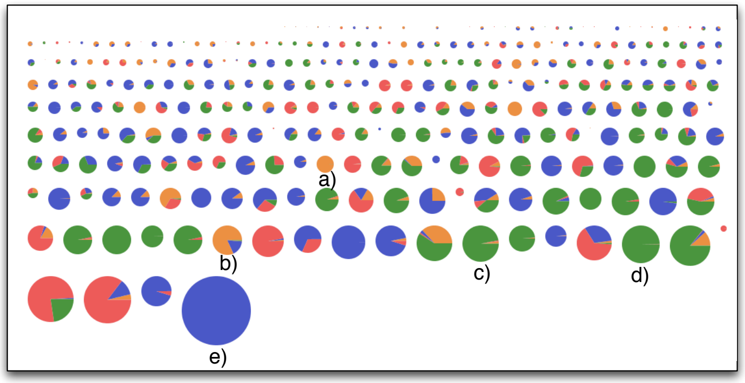

Fig. 11. Pie View for Top and High Priority Problems

In Figure 11 we show the pie view for all top and high priority problems. In this view one of the main advantages of our visualization approach is very obvious. By presenting the complete context, *i.e.*, all the problem reports for the selected priority we automatically focus on problems with the most impact, *i.e.*, the biggest ones. Considering that we are looking at high priority items, the problems labeled a) and b) stand out, because they stayed most of the time in the submitted state. Again looking at their surrounding, almost no other pie displays a similar pattern. It might therefore be worthwhile to take a closer look at a) and b). We selected c) and d) to represent the most dominant pattern in this visualization. There are a lot of problems that spent most of their time in evaluation, and therefore took a long time until they were closed. And finally e) stands out in two ways, it is the biggest pie and it is also totally blue. Consulting the detailed log file for this problem we see that after an initial analysis that yielded a "not repeatable" verdict, the report was not touched for a long time until it finally was rejected. This is rather surprising, since one would expect that, for high priority bugs, there is pressure to close problems as fast as possible.

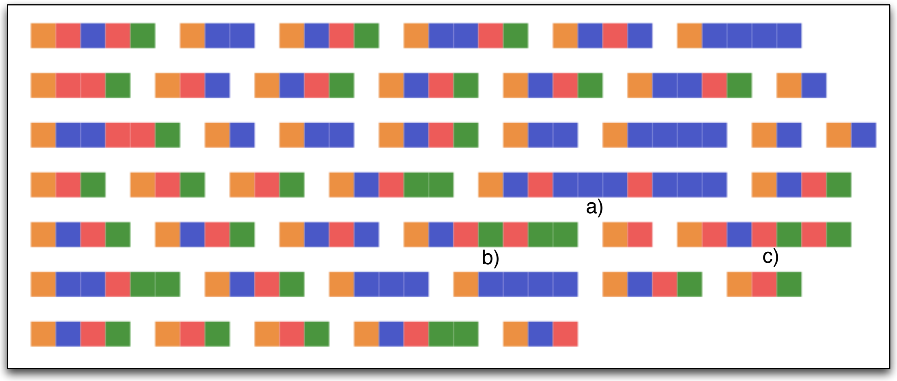

Fig. 12. Phase View for Top and High Priority Problems

In Figure 12, we select the biggest pies from Figure 11 and further analyze them in our Phase View visualization. We selected the problems labeled a), b), and c) because they look the most interesting. In the handling of problem a) a lot of shuffling between different analyzers has happened. From the log file we can extract that every blue block represents a change in the analyzer. There were in total seven analyzers involved and in the end the problem report was marked as a duplicate. The first resolution attempt for the problem labeled b) did not succeed and a second attempt was necessary. And for c) there were three attempts necessary. Another interesting fact that we can spot by looking at c) is, that the first resolution phase was not preceded by an analysis. This might be acceptable behavior or something that should be corrected. Looking at the other problems we can spot quite a few that are missing an initial analysis phase.

We can see from these examples, that by using this easy to understand visualizations, we have already gained insights into potential points for improvement. It is important to keep in mind, that one of the main goals of our approach is to improve awareness for hidden problems and stimulate discussion, and not so much to provide ready made action plans.

6 Related Work

D'Ambros *et al.* used data from a release history database (RHDB) [4] to visualize the life-cycle of bugs [5]. With the System Radiography and the Bug Watch View, they provided a tool to focus on time, activity and severity/priority aspects of bugs. The advantage of such a visualization is the possibility to switch easily between an overview picture and a detailed inspection of an individual bug.

Halverson *et al.* [6] devised a visualization that focuses on the social aspects of change requests and problem reports. Similar to our approach, they also visualize state changes and reveal problematic patterns such as multiple resolve/reopen cycles.

The main differences between these related approaches and ours is that the state changes or the bug activity, in our approach, is only one of many possibilities to look at the data. Also the relatively simple process model underlying Bugzilla, and the missing effort measures, result in a different kind of analysis.

In [7] Weiss *et al.* predict the effort a problem requires to get fixed by analyzing the problem description and search for similar problem reports. The average effort it took these similar problems is than used as a prediction. This relates to our approach in the way, that it uses only the attributes of the problem report without relying on additional information such as source code changes.

Other approaches, such as the one from Ko *et al.* [8] analyze the linguistic characteristics of problem report descriptions to improve the quality of these reports by providing better tool support for problem report entry. Focusing especially on the quality of bug reports, Hooimeijer *et al.* [9] assume that higher quality bug reports are adressed more quickly than those with lower quality. Based on this assumption they use various bug report attributes, *e.g.*, severity, readability, submitter reputation, etc., and predict if a bug report is adressed in a given amount of time.

Who should fix this bug [10] by Anvik *et al.* uses data mining techniques to suggest suitable developers to whom a bug should be assigned. The same authors describe in [11] the problems that arise when using open bug repositories, *e.g.*, duplicates and irrelevant reports. From our experience with an inhouse commercial bug repository we can generally observe similar problems, but less frequently, since the process of bug submission and handling is controlled more tightly.

Most of the presented additional information that researchers are extracting from SCM systems could be integrated in a future version of our approach by simple adding additional possibilities for coloring the polymetric views or adding new sort orders.

7 Conclusions

In this paper we presented an interactive approach to detect visual patterns in issue tracking data. Our approach allows us to detect outliers, flaws in the data entry process, and interesting properties of individual data points.

The visual nature of the approach enhances the awareness for the buried information and promotes discussion between the various stakeholders.

In an industrial case study, we applied our interactive tool to detect instances of the presented generic effort estimation, process step length, and process step sequence patterns. We demonstrated how one can detect relevant information in our easy to understand visualizations.

Discussion with project managers and other software development professionals provided strong anecdotal evidence that our approach is indeed capable to improve certain aspects of the software development process in the companies that participated in this innovation project. In all discussions the participants were especially impressed by the straightforward interpretation of the visualizations. This is a key point to promote the adaption of such a tool in the companies of the industrial partners.

Acknowledgments

This work was sponsored in part by the Eureka 2023 Programme under grants of the ITEA project if04032. It was also supported by the Swiss National Science Foundation under grants of the Project DiCoSA - Distributed Collaborative Software Analysis (SNF Project No. 118063).

References

1. Tufte, E.R.: Envisioning Information. Graphics Press (May 1990)
2. Knab, P., Pinzger, M., Fluri, B., Gall, H.C.: Interactive Views for Analyzing Problem Reports. In: ICSM '09 Proceedings of the 25th International Conference on Software Maintenance. (2009) 527–530
3. Lanza, M., Ducasse, S.: Polymetric views — a lightweight visual approach to reverse engineering. IEEE Transactions on Software Engineering **29**(9) (September 2003) 782–795
4. Fischer, M., Pinzger, M., Gall, H.: Populating a release history database from version control and bug tracking systems. In: Proceedings of the International Conference on Software Maintenance, Amsterdam, Netherlands, IEEE Computer Society Press (September 2003) 23–32
5. D'Ambros, M., Lanza, M., Pinzger, M.: "a bug's life" - visualizing a bug database. In: Proceedings of VISSOFT 2007 (4th IEEE International Workshop on Visualizing Software For Understanding and Analysis), IEEE CS Press (June 2007) 113–120
6. Halverson, C.A., Ellis, J.B., Danis, C., Kellogg, W.A.: Designing task visualizations to support the coordination of work in software development. In: CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work, New York, NY, USA, ACM (2006) 39–48
7. Weiss, C., Premraj, R., Zimmermann, T., Zeller, A.: How long will it take to fix this bug? In: MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories, Washington, DC, USA, IEEE Computer Society (2007) 1
8. Ko, A.J., Myers, B.A., Chau, D.H.: A linguistic analysis of how people describe software problems. In: VLHCC '06: Proceedings of the Visual Languages and Human-Centric Computing, Washington, DC, USA, IEEE Computer Society (2006) 127–134
9. Hooimeijer, P., Weimer, W.: Modeling bug report quality. In: ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, New York, NY, USA, ACM (2007) 34–43
10. Anvik, J., Hiew, L., Murphy, G.C.: Who should fix this bug? In: ICSE '06: Proceedings of the 28th international conference on Software engineering, New York, NY, USA, ACM (2006) 361–370
11. Anvik, J., Hiew, L., Murphy, G.C.: Coping with an open bug repository. In: eclipse '05: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange, New York, NY, USA, ACM (2005) 35–39