Information Support For Organizational Development Structures

Master Thesis
to obtain the academic degree of
Diplom-Ingenieur
in the Master’s Program
Computer Science

Submitted by
Johann Tuder, BSc MSc

Submitted at
Institute for Software Systems Engineering

Supervisor
Prof. Dr. Alexander Egyed

Co-Supervisor
Dr. Christoph Mayr-Dorn

September 2018
Sworn Declaration

I hereby declare under oath that the submitted Master’s degree thesis has been written solely by me without any third-party assistance, information other than provided sources or aids have not been used and those used have been fully documented. Sources for literal, paraphrased and cited quotes have been accurately credited. The submitted document here present is identical to the electronically submitted text document.
# Contents

1 Introduction 1  
1.1 Problem and Goal 1  
1.1.1 Problem 1  
1.1.2 Goal 3  
1.2 Structure of Thesis 4  

2 Related Work 5  
2.1 Tools 5  
2.2 Development and collaboration 7  
2.3 Comparison of approaches 10  

3 Approach 11  

4 Design 13  
4.1 Architecture 13  
4.2 M1 - Issue resolved/closed date <= issue due date 13  
4.3 M2 - Due dates aren’t changed 14  
4.4 M3 - Fix version wasn’t changed 14  
4.5 M4 - Duration in approve state 14  
4.6 M5 - Re-open rate of issues closed shortly before their due date 15  
4.7 M6 - Assignees of an issue 15  
4.8 M7 - Commentators of an issue 16  
4.9 M9 - Issue re-assignment without any communication 16  
4.10 M11 - Issue re-assignments without status changes 16  
4.11 M12 - Duration between re-assignment and first status change 17  
4.12 M13 - Issue resolved date compared to code freeze date 17  
4.13 M14 - Issues moved to next milestone 18  
4.14 M15 - Issues of next milestone started in current milestone 18  

5 Implementation 19  
5.1 JIRAExtractor 19  
5.2 Analyzer Tool 21  

6 Evaluation 26  
6.1 M1 - Issue resolved/closed date <= issue due date 26  
6.2 M2 - Due dates aren’t changed 31  
6.3 M3 - Fix version wasn’t changed 35  
6.4 M4 - Duration in approve state 38  
6.5 M5 - Re-open rate of issues closed shortly before their due date 41  
6.6 M6 - Assignees of an issue 44  
6.7 M7 - Commentators of an issue 50  
6.8 M9 - Issue re-assignment without any communication 54  
6.9 M11 - Issue re-assignments without status changes 58  
6.10 M12 - Duration between re-assignment and first status change 61  
6.11 M13 - Issue resolved date compared to code freeze date 64  
6.12 M14 - Issues moved to next milestone 67  
6.13 M15 - Issues of next milestone started in current milestone 70
List of Figures

1. Architecture overview ............................................. 13
2. Initial analyzer view ............................................. 24
3. State transition of issue ......................................... 27
4. Test stage after finished sub issue ............................. 27
5. Issue compared to its parent-parent-issue .................. 28
6. Same occurrence of due date change in issue and sub issue 29
7. Late closed issue .................................................. 29
8. Fastly resolved issue that was closed late ................. 30
9. Database issue with heavy testing ............................. 30
10. Due date set after completion ................................. 31
11. Issue with multiple due date changes ...................... 32
12. APPMAP-1 with due date changes ............................ 32
13. Multiple changes without events in between ............. 33
14. Multiple changes without events in between ............. 33
15. Due date update after status change ....................... 34
16. Multiple changes without events in between ............. 34
17. Overview of changed issues .................................... 36
18. Timeline with pentagons ....................................... 36
19. Main project issue .............................................. 36
20. Cloned graphics issue ......................................... 37
21. Fix version changes without events in between .......... 37
22. Issue in approve state ......................................... 39
23. Issue passed to developer ..................................... 39
24. Issue reopened and resolved without approval .......... 39
25. Issue immediately approved .................................. 40
26. Issue with long approve state ............................... 40
27. Re-opened issue ............................................... 41
28. Re-opened issue with 143 days duration ................. 42
29. 5 times re-opened issue ..................................... 42
30. Issues with most inner changes ............................. 44
31. Issues with most outer changes ............................ 45
32. Graphics issues with most inner changes ............... 46
33. Quality assurance issues with most inner changes ..... 47
34. Development issues with most outer changes ........... 48
35. Development issue with "null" changes .................... 48
36. Most changed issues .......................................... 49
37. Issue with most intra department changes .............. 49
38. Most comments ............................................... 51
39. Commentators of issue ...................................... 51
40. Most comments and more independent ................. 52
41. Most comments, single user .................................. 52
42. Long comment ................................................. 53
43. Testing after long comment .................................. 53
44. Most re-assignments without communication .......... 54
45. Longest duration between re-assignments ............... 55
46. Re-assignments without status change ................... 55
47 Quality assurance issues with high average duration ........................................... 56
48 Most re-assignments with comments at the end .................................................. 56
49 Most re-assignments with comments during lifetime ......................................... 57
50 Most re-assignments with comments during lifetime ......................................... 57
51 Most re-assignments without status change ...................................................... 58
52 Low average duration ......................................................................................... 59
53 Low average duration and multiple changes on same date ............................... 60
54 Highest duration between assignment and status change .............................. 62
55 Multiple changes for one issue ......................................................................... 62
56 Changes with specific values .............................................................................. 63
57 Graphics issues with long duration .................................................................... 63
58 Highest differences to code freeze date ............................................................. 65
59 Maximum and minimum duration ..................................................................... 66
60 Issue resolved 554 days after code freeze ....................................................... 67
61 Milestone change ............................................................................................... 68
62 Resolved issue .................................................................................................... 68
63 MS moved without status changes .................................................................... 68
64 MS moved 5 times until first status change ...................................................... 69
65 Issues of P2 with highest distance to code freeze date ..................................... 71
66 P3 issues started in previous milestone ............................................................. 71
67 Issues for participants ......................................................................................... 73
### List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mapping of departments</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>Development of due date changes</td>
<td>34</td>
</tr>
<tr>
<td>3</td>
<td>Ratio of issues with a due date set</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>Ratio of issues that had to be changed</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>Development of fix version changes</td>
<td>38</td>
</tr>
<tr>
<td>6</td>
<td>Ratio of changed issues</td>
<td>38</td>
</tr>
<tr>
<td>7</td>
<td>Issue duration in approve state</td>
<td>41</td>
</tr>
<tr>
<td>8</td>
<td>Re-opened issues</td>
<td>43</td>
</tr>
<tr>
<td>9</td>
<td>Issue assignees</td>
<td>50</td>
</tr>
<tr>
<td>10</td>
<td>Comments of an issue</td>
<td>54</td>
</tr>
<tr>
<td>11</td>
<td>Comments of an issue</td>
<td>58</td>
</tr>
<tr>
<td>12</td>
<td>Top issues of M9 compared to M11</td>
<td>60</td>
</tr>
<tr>
<td>13</td>
<td>Comments of an issue</td>
<td>61</td>
</tr>
<tr>
<td>14</td>
<td>Average values for duration between assignment and status change</td>
<td>64</td>
</tr>
<tr>
<td>15</td>
<td>Average values for duration between assignment and status change</td>
<td>70</td>
</tr>
<tr>
<td>16</td>
<td>Most valuable metrics for participants</td>
<td>75</td>
</tr>
<tr>
<td>17</td>
<td>Tool rating by participants</td>
<td>77</td>
</tr>
<tr>
<td>18</td>
<td>Positive and negative criticism</td>
<td>77</td>
</tr>
<tr>
<td>19</td>
<td>Tool rating</td>
<td>84</td>
</tr>
</tbody>
</table>
Abstract

The purpose of this thesis was to analyze the quality of the organizational structure and issue handling process in the software development company bet-at-home.com. The problem was that projects often last longer than intended or mis-communication between employees happened and that such circumstances were not visible in a tool and therefore investigation was cumbersome. 13 metrics were defined and a tool was implemented to access the necessary information from JIRA and to analyze them according to the metrics. 4 projects were picked and compared, where the teams of the projects were differently organized and had different workflows. The results showed that vertically organized teams perform better during a project than horizontally organized teams. Which indicated that teams consisting of employees from different departments work more closely together when they are sitting next to each other and discuss problems/projects face-to-face.
1 Introduction

This section will give a short introduction to the master thesis. At first the general problem is discussed. Afterwards it is shown which problems at bet-at-home.com exists and what the goal of the thesis should be. At the end of the section a short overview of the thesis’ structure is given.

1.1 Problem and Goal

1.1.1 Problem

Valetto et al. [1] say that a correlation between the structure of a software and the layout of the development organization working on it has been recognized. Further they mention an important step to improve productivity and quality is to be able to measure and understand how people are organized and how they interact with one another when they develop software [1]. Therefore it should be measured if there is a good fit between the coordination structure mandated by the technical work units, and the actual social organization. Cataldo et al. [2] suggested that high degrees of socio-technical congruence are beneficial for software development performance. According to Valetto et al. [1] the term ”socio-technical” indicates a connection between social and personal behavior and technical work.

The study of Sosa et al. [3] found a correlation between artifact dependencies and communication frequency among stakeholders involved with the specific artifacts. For Valetto et al. [1] socio-technical software networks were created by building ”work relationships”, which can be seen as work done by stakeholders on specific artifacts. They used data maintained in various software repositories. Additionally to that they suggest the usage of developers’ mailing lists to better identify relationships, but an approximation can still be achieved by using source management systems.

In the future work part it is suggested that more defects could occur where the congruence is low. As an example for this thesis the company bet-at-home.com will be used to analyze the mentioned points [1].
Bet-at-home.com is a software development company that is interested in improving its workflows and software development process. Within completed major projects the current process is evaluated and adapted to fit new organizational circumstances such as new teams.

The company is structured in ten departments. Those departments consist of 22 teams. The following teams are of interest for this thesis: software developers - frontend, software developers - backend, database, graphics, quality assurance and product & project management. These teams are heavily involved into the software development process.

The departments are physically separated within two company buildings. A project team usually consists of at least one member of each department of interest for this thesis. When the project is released for public access the team is responsible for maintenance. This means the team has to work together during the project lasts (initial project coordination, implementation, further support) although they are situated in different offices.

This organization style causes a lot of necessary meetings or informal communication channels within the team. This often results in undocumented decisions in any of the used tools. The company uses JIRA, Confluence, Skype, rocketChat and Outlook to keep track of projects. JIRA is used as ticketing system and also provides further information such as comments and lists of changed source code files. Thompson et al. state that issue repositories (e.g. JIRA) hold key information defining what the system under development will do (description), who will work on different parts of the system (assignee), what defects occur as the system is being built, and more (e.g. resolution status, comments, reviewers) [4]. Muller and Fritz say that Confluence is used for documenting implementation details, meeting protocols and personal notes. It can be compared to a Wiki and is very useful to overcome the often time-consuming interaction with other stakeholders [5]. Wikis are used to manage a lot of different information artifacts that are used in a lot of different activities [5]. In the case of bet-at-home.com this means for example meeting protocols, technical specifications, system documentation, project diaries and so on. Skype, rocketChat and Outlook are used for work- and non-work-related communication. For Giuffrida and Dittrich instant Messaging (e.g. Skype), Internet forums, Blogs,
Microblogs, Wikis, Social Networks and Social Bookmarking are also known as Social Software [6].

De Souza et al. state that since software development is a collaborative activity in which experts from different domains work together [7] the company investigates if the current situation is the most efficient for handling projects and the daily business. For Panichella et al. teams somewhat mirror the software modularity and even if the number of teams across releases is quite stable, there is a continuous reorganization of teams during the evolution of a project [8]. Since a group of people always works together for a long period of time it is probably not the best way that they do not share the same office. Also [8] showed in their study that there seems to be a relationship between the testing- and the development-team. Further studies show that there is a deep unreliability of electronic traces (repositories, email conversations, meeting requests, specifications, document revisions, and organizational structure records) for bug reports, which were erroneous or misleading in seven of ten cases and incomplete and insufficient in every case [9]. These outcomes underline that informal communication seems to be irreplaceable for project work and that there is a possibility of positive influence on productivity and efficiency if the teams are organized in a new way.

1.1.2 Goal

The goal of the thesis is to provide a decision support for team restructuring. From a general view it should be possible to analyze the treatment of issues. Mainly development responsibles should use the tool to see where problems occur. They have to distinguish if problems result from e.g. workflow flaws, project descriptions or requirements. Furthermore multiple projects may be compared so that it can be measured if issue treatment is influenced by organizational changes after a project. Also developers should profit by analyzing their issues and adapt their own workflows.

For bet-at-home.com it is important to compare several issues and projects. It should be analyzed if issues are e.g. blocked by a specific department. With the solution it should be possible to see if specific patterns occur for the same issues and to analyze what the reason of the flaw is. With the analysis it should be possible to re-arrange teams within the company so that departments work more efficient and projects can be finished faster.
and with higher quality. Therefore it has to be evaluated if the current team organization and issue ticket handling is efficient.

To describe the goal more precisely the GQM (Goal-Question-Metric) approach is used and described in chapter 3.

1.2 Structure of Thesis

In chapter 2 the related work is presented and will give a more detailed overview of the problem.

The following chapter 3 describes the Goal-Question-Metric approach.

In chapter 4 a short architectural overview of the tool is given. Furthermore the metrics of are explained in more detail.

Afterwards the implementation of the tool is shown in chapter 5.

In chapter 6 the tool is used to analyze 4 projects in terms of the metrics. Also the results of a questionnaire, taken by development responsibles at bet-at-home.com, about the tool is shown and the results are discussed.

In the last chapter 7 the thesis is summarized and an outlook to future work is given.
2 Related Work

2.1 Tools

Bertram et al. mention several important points about issue trackers and the software development process in general. Issue tracking itself is a social process and the tracker serves as a key knowledge repository, communication and collaboration hub [10]. The tracker shows the progress of issues from their initial creation to a final closed state and along with these various annotations such as the title, due date and a history of ownership and discussion are stored [10]. In their work they asked different stakeholders about their usage with issue trackers. They found that issues were frequently assigned between project managers and developers to clarify priority and direction, furthermore QA assigned bugs directly to specific (lead) developers [10]. Alongside the information to implement new features or fixing bugs it is very common to discuss decisions or solutions. The conversations are communicated across different channels and are afterwards stored alongside the bug, this can be challenging but proves to be crucial to provide the team the necessary tools to coordinate work [10]. Additionally Bertram et al. found that reports were often generated directly from the issue tracker or leveraged the data stored within it [10].

Also Reis et al. state that an issue tracker helps software teams to manage their workload via a reliable, shared to-do-list that is used by numerous stakeholders through the whole lifecycle and furthermore serves as an archive of completed work [11]. Shrivastava and Date [12] argue that maintaining valuable documentation may improve distributed software development and that tools like issue trackers help in maintaining documentation. They further state that there must be a commitment to provide the team tools to maximize communication and to give them time to optimize around them [12]. Martin Fowler [13] argues that becomes more important with separated teams since face-to-face communication is reduced, also there is more need for active collaboration tools like wikis and issue trackers. However they also pose new challenges: developers have to deal with a large number of reports and often those are not productive [14].

Luijten et al. implemented a tool to generate three different views that enable assessment of an issue handling process [15]. They had a high-level (Issue Churn View), a quantitative (Issue Risk Profiles) and a detailed lifecycle (Issue Lifecycle View) view [15]. The Issue
Churn View show issue handling on a monthly basis and they wanted to check if the backlog of the analyzed projects grows, since this could indicate that issue solving is too slow or bug reports have a low quality [15]. The Issue Risk Profiles quantify the speed of issue resolution, which is defined as that time where an issue is in an open state (new or assigned, but not closed or resolved) [15]. The Issue Lifecycle View offers a scatter plot of issues versus modification dates, where blue dots signal a submitted comment and yellow dots point at other events (e.g. reassignment) [15]. They concluded that such visualizations offer a quick way of comparing the defect solving efficiency and that those instruments can be used alongside others to assess the issue handling efficiency at a high abstraction level and in detail [15]. Kim and Whitehead found out that issues with long bug-fixing time also contain more bugs than other files [16]. Knab et al. [17] visualize the duration of a process step (submitted, in_analysis, in_resolution, in_evaluation) with a pie chart and they also add a state transition view for problem reports.

Sarma et al. developed Tesseract, a socio-technical dependency browser that enables exploration of relationships between artifacts, developers, bugs and communications [18]. They say that artifacts, developers and tasks are intrinsically bound together in a software project [18]. They suppose that developers who are modifying interdependent code modules and are not communicating to each other may indicate future integration problems [18]. Gutwin et al. [19] argue that for developers it is often difficult to remain up-to-date when communication is split on between email, chat and issue tracker, which often causes duplication of information and situations where developers missed important parts of the information.

Dal Sassc and Lanza [20] implemented in*Bug, a web-based software visual analytics platform. Starting with the main premise that repositories of bug tracking systems are a valuable source of information they implemented different panels for viewing issues [20]. The ”bug lifetime panel” shows duration (as a horizontal stacked bar chart) and status of bugs [20]. Fine grained bug views describe changes in a bugs properties, like opening of a bug, a status change, the addition of other information, assignment to another user and the fix of a bug [20]. As they say these events are the basic building blocks of a bug and visualizing them is the key for its understanding [20]. D’Ambros et al. [21] state that a well-designed visualization is crucial to cope with the data that has to be analyzed. Therefore they present two different visualization techniques: System Radiography and...
Bug Watch [21]. System Radiography happens at system level and provides indications about which parts of a system are affected by what kind of bugs at which point in time [21]. Whereas Bug Watch shows information of a specific bug and helps to understand various phases that it traversed [21]. Also it should show that the criticality of a bug is not only dependent on its severity and priority but also on its life cycle, which means that frequently opened bugs indicate deeper problems than expected [21]. Halversion et al. [22] describe change tracking systems as repositories for textual descriptions. They have a large number of attributes like who the changes was filed by, who is assigned to work on it, the current state, comments and much more [22]. Further they determined three problematic patterns of change management [22]:

- **Ping Pong Patterns**
  Recurrent loops (e.g. repeatedly resolving and reopening or reassigning) are potentially indicating deeper problems.

- **What’s Falling through the Cracks and Why: Prioritizing and Managing Work**
  Occurs when a defect goes too long without resolution.

- **Bug Reporting at the Wrong Level**
  This is mostly indicated through if the Ping Pong Pattern occurs. Than a manager should look at the change request for the level of detail.

Weiss et al. [23] implemented an approach that automatically predicts the effort in person-hours for new issues. For calculation existing issue reports that are most similar to the new one are used, to find those text similarity techniques based on the title and description of issues are used [23]. Their framework estimated effort for bugs very efficient and was only one hour off [23].

### 2.2 Development and collaboration

Perry et al. report on two experiments to discover how developers spend their time. For them it is important to approach all relevant contexts (technological, social and organizational) to understand and significantly improve development processes [24]. In the first experiment developers had to fill out a modified time card reporting their activities. The second experiment used direct observation to validate the use of the time diaries in
the first experiment. Besides their experiments they divided the working time into race and elapsed time. Where race time expands to elapsed time when there are interruptions, blocking and waiting periods [24]. A major finding is that the development phase primarily consists of coding, but roughly half the time is also occupied by non-coding tasks [24]. The ratio of elapsed to race time is roughly 2.5, which means that developers worked on a specific development task only 40 percent of the time [24]. LaToza et al. found that developers at Microsoft also reported that they are spending nearly half of their time with fixing bugs [25]. Perry et al. further revealed that developers are mostly working on two projects at once so that they can deal with with blocking or waiting phases [24]. Furthermore a developer spent about 75 minutes per day with unplanned informal communication [24]. However the most ubiquitous form of contact was in-person visits, which occurred approximately two to three times as often as other media [24]. The main results of the experiments were that software development is not an isolated activity and that progress is often impeded for a variety of reasons (reassignment to a higher priority task, waiting for resources, context switching to maximize individual throughput) [24].

Sandusky and Gasser stated that some of their research suggests that distributed teams cannot succeed without the access to richer modes of interaction (face-to-face interactions) [26]. Their study addressed software problem management which is part of traditional and F/OSS (free/open-source-software) development processes [26]. The analyzed bug reports indicate that software problem management involves interrelated and nested subprocesses like negotiation, which is seen as basic social process [26]. Strauss defines negotiation as a basic social process to address uncertainty and complexity, coordinate activity, and support distributed collective sensemaking [27]. In Sandusky and Gassers study implicit negotiation occurs, which is defined as an implicit agreement on an original statement if a specific response is missing [27]. One of the analyzed bug reports existed for 29 days in the system and four persons participated in the negotiation process with 7 contributions during the first 9 days [26]. Sandusky and Gasser state that it is possible that the individuals were communicating using other media (e.g. chat, email, face-to-face, etc.) [26]. Negotiation traces in bug reports are important, because they help to correctly analyze the evolution of interpretations, specifications, decisions and rationales [26]. Further information management activities are as important as problem solving activities because effective resource allocation (e.g. programmer time) depends on it [26].
However Crowston argues that coordination mechanisms within organizations will vary, this includes e.g. how developers are assigned to specific bug reports [28]. Herbsleb and Grinter found that there are coordination problems in globally-distributed projects within traditional software development organizations, because it limited the chance of informal contact and spontaneous support when it is needed [29]. In another study Herbsleb found evidence that cross-site work suffered significant delays when compared to same-site work [30]. It was observed that barriers to team coordination were lack of unplanned contact, knowing the right person to contact regarding an issue, cost of contact initiation, effective communication and lack of trust [29]. Larsson examined that it is important for globally distributed teams to use negotiation to construct common ground and to have a shared understanding [31]. In the study it is stated that it is critical for distributed activity to use rich media or periodic face-to-face interactions [31]. Martin Fowler states that teams should be separated by functionality and not by activity and that development works best with close communication and an open culture [13].

Nagappan et al. [32] investigated the organizational structure of a company on the quality of its software. They state that software engineering is a complex activity that involves interactions between, people, processes and tools to develop a product [32]. Conway defines a law which explains that organizations that design systems produce systems that are copies of the organizations communication structures [33]. Parnas goes into the same direction by arguing that a software module is more a responsibility assignment than a subprogram and that organizational structure is very important in the software industry [34]. The study of Nagappan et al. used a GQM (Goal-Question-Metric) approach. The defined the following eight organizational measures/assumptions [32]:

- The more people who touch the code the lower the quality.
- A large loss of team members affects the knowledge retention and thus quality.
- The more edits to components the higher the instability and lower the quality.
- The lower level is the ownership the better is the quality.
- The more cohesive are the contributors (organizationally) the higher is the quality.
- The more cohesive is the contributions (edits) the higher is the quality.
- The more the diffused contribution to a binary the lower is the quality.
• The more diffused the different organizations contributing code, the lower is the quality.

By using step-wise regression and principal component analysis Nagappan et al. observed that all these assumptions contribute towards explaining the variance in post-release failures of Windows Vista [32]. They concluded that their organizational measures predicted failure-proneness with significant precision, recall and sensitivity and that their model performed better than traditional prediction models (e.g. code churn, code complexity, code coverage, code dependencies) [32].

2.3 Comparison of approaches

The data used in the thesis differs from most related works since it is not open sourced. All project data is industrial data provided by a single company (bet-at-home.com) and is not publicly shared. The used data is provided by JIRA as issue tracker and Perforce as source code management system. The used data was selected by project size and people involved in them. Each project endured over several months and all but one of them was is completed. Workers of the departments project management, development, graphics, quality assurance and database were involved in the projects. In difference to the related works a goal-question-metric approach is used to analyze the data. The metrics were designed together with the company and reflect important topics that have to be investigated. The goal of this thesis is to create a tool that is able to analyze data extracted from JIRA and Perforce. The tool has to display specific values where possible and should show an issue lifetime on a timeline with all relevant events. This approach is also used in the related work since it provides a good and quick overview of data and possible flaws.
3 Approach

The following GQM-template is used to address the goal:

<table>
<thead>
<tr>
<th>Field</th>
<th>Aspect</th>
</tr>
</thead>
<tbody>
<tr>
<td>object of study</td>
<td>team structure, issue ticket usage</td>
</tr>
<tr>
<td>purpose</td>
<td>evaluate</td>
</tr>
<tr>
<td>focus</td>
<td>efficiency, coordination effort</td>
</tr>
<tr>
<td>stakeholder</td>
<td>developer, team leads</td>
</tr>
<tr>
<td>context factors</td>
<td>distributed teams</td>
</tr>
</tbody>
</table>

The template is expressed as:

The purpose of this study is to evaluate the current team structure and issue ticket usage on efficiency and coordination effort from the point of view of software developers and team leads in the context of distributed teams.

According to GQM questions have to be found and the answers (metrics) to them have to lead to the goal. For the defined purpose following questions and metrics are used:

Q1: What is an efficient team?

M1: issue resolved/closed date <= issue due date

M2: due dates aren’t changed

M3: fix version isn’t changed to a later version

M4: duration of an issue in an approve-state

M5: re-open rate of issues closed shortly before their due-date

Q2: Which coordination structures already exist?

M6: set consisting of assignees that overlaps with set of commentators/reviewers of an issue

M7: set consisting of commentators that overlaps with set of assignees/reviewers of an issue

M9: number of issue (re-)assignments without any communication
Q3: How efficient is coordination?

M9: see above

M11: number of issue re-assignments without status changes

M12: duration between issue re-assignment and first status change

Q4: How efficient is project planning?

M13: issue resolved-date compared to code-freeze date of current milestone

M14: number of issues that are moved to the next milestone

M15: number of issues of next milestone started in current milestone

There are multiple other (more specific) questions that should be answered for bet-at-home.com. The mentioned questions build a summary of the specific ones and will provide useful data to help re-organizing the teams and identifying problems with the issue handling processes on a snapshot base.
4  Design

In this section a short overview of the architecture is given and the metrics are precisely described.

4.1  Architecture

The architecture consists of three parts:

![Architecture overview](image)

The JIRAExtractor exports a given project with all its sub-issues from JIRA and transforms each single issue into a JSON document. During this transformation the extractor tries to map a worker to a department. Therefore it looks up a dictionary with all already processed workers, if a worker is not in the dictionary the user is prompted to enter the workers department. Afterwards the worker is added to the dictionary and the JSON is sent to CouchDB. In CouchDB each project is represented as a single database with all issues in it. The Analyzer loads all available databases (i.e. projects) from CouchDB. The user can choose between the projects and the analyzer shows the available information and calculates the metrics.

4.2  M1 - Issue resolved/closed date $\leq$ issue due date

The metric shows if issues were closed or resolved before they were due. The resolved date is set e.g. if a developer finishes implementation. The closed date is set when e.g. a tester finished testing of bug-fix. In the ideal case every issue should be closed before its due-date. This would indicate perfect project planning and a good workflow through the development process. However if issue are closed too early it could also show that planning was not done right and developers were faster than expected, which could also lead to problems (e.g. not having enough workload). The metric is important to determine where a flaw appears and if there a patterns for specific issues.
For this metric the relevant fields of an issue are its closed-, resolved- and due-date. If the values are available the difference between the closed- or resolved-date and the due-date are calculated. A negative value of the difference indicates that an issue was closed or resolved before its due-date, a positive value shows the opposite.

4.3 M2 - Due dates aren’t changed

The metric shows if the due dates of issues were changed. Due dates should be set at the beginning of a project. So that project managers and developers can arrange the workload and know when which part of the project will be finished. If due date change indicates that something went wrong during project planning or a developer (or employee of an other department) had something more important to do. However it could also indicate that the effort estimation was wrong and therefore it had to be changed.

For the metric every changelog entry of type ”duedate” is important. If such an entry exists it indicates that the issues due date was changed. Every occurrence of a change is shown and it is calculated how many of the projects issues had their due date changed.

4.4 M3 - Fix version wasn’t changed

The metric basically shows the same information as M2 but for changelog entries of type ”fix version”. A fix version is set when e.g. an issue is finished and can be release or when an issue is assigned to a milestone. This means that fix version changes show if issues were moved between milestones during planning or implementation but also shows if already finished issues are moved to other fix versions. Therefore it can indicate problems in project planning and also the release workflow.

4.5 M4 - Duration in approve state

The metric shows how long a single issue was in the approve state. Some issues have to be approved by teamleads until someone can start working on an issue. Usually such an approval process does not take very long and with the metric it should be clarified if this is the case for all issues. If it is not the case the responsible people have to find out why it lasted longer, since this step of the process stops blocks other people from handling the issue.
For the metric the changelog is analyzed and if an status entry is found it is checked if the status was changed to ”Approval necessary”. If that is the case it is looked for the next status entry and the difference between both ChangedAt-dates is calculated.

### 4.6 M5 - Re-open rate of issues closed shortly before their due date

The metric shows all issues that were re-opened and have a due date set. It is analyzed if re-opens occur often. This would indicated that developers are not working properly or that requirements were not specified understandable or something was missing. Also it can be analyzed if re-opens occur more often closer to the due date, which would indicate that the due date is a stress factor and leads to lower quality during development.

For the metric the changelog is analyzed for status changes where the value was changed to ”Open”, ”Open Again” or ”Reopened”. If such an entry exists the ChangedAt-date is taken and the difference to the due date is calculated. A positive result indicates that the issue was re-opened after the due date, the negative result that is was re-opened before the due date.

### 4.7 M6 - Assignees of an issue

The metric shows all issues where the assignee was changed. Some changes are part of the workflow. For example a project manager creates an issue and assigns it to a developer. After the implementation the issue is assigned to a tester. This means that some changes are predicted and the metric should show if there occur more changes than the predicted ones. A high number of changes could indicate that there are communication problems between departments or that an issue was not implemented correctly. It could also show that within a department an issue is passed e.g. from developer to another developer which should not be the case. Therefore the metric is split up between intra and cross department changes.

For the metric entries in the changelog with the type ”assignee” are analyzed. Since the department is prepended to the username it can be extracted from the FromValue and ToValue property of the entry. The result shows for every issue the number of intra and cross department changes and the sum of all changes. It also shows the average of the
changes.

4.8 M7 - Commentators of an issue

The metric shows every issue of a project and who submitted a comment on an issue. It is implied that for complex issues comments should be made. It should indicate if users are working with the comment feature or if discussion, assumptions and decisions are stored outside of JIRA. Since the metric also shows the publisher of a comment it can be determined how many issues are made by tools and if they are useful or not. Usually issues with cross department concerns should show a higher amount of comments so that decisions or states can be communicated.

For the metric the length of every comment is counted as number of characters within the comment. Additionally to every issue the commentators and the amount of comments of each are shown. Also the maximum, minimum and average length of comments per issue are displayed for further analysis.

4.9 M9 - Issue re-assignment without any communication

The metric shows every issue that was re-assigned twice without a comment in between. An issue is re-assigned when e.g. a developer finished the implementation and a tester has to start testing the feature. This is a normal workflow and often does not require any additional communication. However it may happen that an issue re-assignment is not intended by the workflow but still occurs. If such an incident happens it should be commented.

The metric uses the changelog entries of type "assignee". If two entries were found it is checked if a comment was submitted between the two changes. If there was no comment a counter is incremented, otherwise it is looked for the next assignee changes. The metric shows the sum of changes without a comment in between and the duration between two changes in days.

4.10 M11 - Issue re-assignments without status changes

The metric shows nearly the same information as M9 however it is checked if between two re-assignments a status update happened. Usually an issue is re-assigned to another
user if the status of it also changed, e.g. the developer finishes implementation and sets the status to "Resolved", assigns it to a tester which changes the status to "In Testing". If issues are re-assigned without a status change something could be wrong either in the workflow or how users handle issues.

The metric uses the same changelog entries as M9. Additionally the "status" entry changelog type is used to find status changes between re-assignments.

4.11 M12 - Duration between re-assignment and first status change

The metric shows the timespan between a re-assignment and the first status change after it. The general assumption is that an issue is re-assigned so that another user can start the work on an issue. With the metric flaws should be found, where and why a particular timespan is too big. A long timespan could indicate that the user that got the issue assigned probably has enough work and that issues should be handled by other users. The metric uses the changelog entries of type "assignee" and "status". First it is looked for an "assignee" entry and if it was found a "status" entry should be found. If both entries exists the duration between both is calculated. The metric also shows the average duration between the two events.

4.12 M13 - Issue resolved date compared to code freeze date

The metric shows the difference between the resolve date of an issue and a code freeze date that is defined by a user. Usually issues should be resolved before code freeze, because during code freeze no new features are implemented for the current milestone and the time to release is for bug fixing and testing. If issues are resolved after code freeze the problem has to be found, since it can indicate work overload and wrong project planning or normal bug fixing.

The metric uses the resolved date of each issue and a code freeze date, which is an input field and has to be provided from the user. Afterwards the metric shows the timespan between both dates in days. A positive number indicates that an issue was resolved before the code freeze, a negative number the opposite. Additionally the average, maximum and minimum timespans are displayed.
4.13 M14 - Issues moved to next milestone

The metric is nearly the same as M3. However it only displays those issues that were moved to a following milestone. If project planning is done right no issue should be moved and everything should be finished in the milestone it was planned for. It can happen that issues are moved and then it has to be analyzed why this was the case. It could also be the case that a developer had too much work and therefore the issue had to be moved. With the metric it should be easier to find out which issues had such a problem and improve the situation.

The metric analyzes the changelog entries of type "fix version" and compares the From-Value and ToValue of a single entry. If the ToValue is greater than the FromValue it is counted as moved to a following milestone. The metric displays every single move to a following milestone and calculates the ratio of moved issues in all issues of a project.

4.14 M15 - Issues of next milestone started in current milestone

The metric calculates the timespan between the first progress of an issue to the start date of the a milestone. In this case it should show which and how many issues of a project were started too early. This indicates that some users do not have enough issues to work on or that project planning was not accurate enough. Which means that at the end of the milestone users were already implementing for another milestone and they could have brought more features into the current one.

The metric calculates the timespan from a date which is provided by the user to the first occurrence of a changelog entry of type status with a value of "In Progress" or "In Development". A positive timespan shows that the issue was started before the official milestone start, a negative number shows the opposite. Additionally the average, maximum and minimum timespan values are displayed.
5 Implementation

The tool was implemented to only interact with JIRA, since it is used in the company. It is split into two parts, the JIRAExtractor and the Analyzer. The extractor connects to JIRA and exports a project which is afterwards used by the analyzer to calculate the metrics and display the timelines.

5.1 JIRAExtractor

The extractor is written in C# and connects to the live JIRA system. It exports a project with all sub-issues and posts the files into a CouchDB database. This mechanism is used so that during calculation the live JIRA system does not have to be used. Furthermore the extractor accesses Perforce to get information about submitted files for a specific issue. It uses the following libraries to handle the information:

- Atlassian.SDK (handling of JIRA issues)
- Perforce.Api.Net (retrieving filenames from Perforce)
- RestSharp (connecting to JIRA, Perforce and CouchDB)
- Newtonsoft.Json (converting the issue into JSON)

To run the extractor the following information is needed:

- Username/password for JIRA and Perforce
- URL to connect to JIRA
- Key of a seed issue

The first step is to setup the database in CouchDB with the name of the seed issue. The Rest client is generated and connects to JIRA with the provided information. Afterwards the extraction starts with the seed issue.

The Atlassian.SDK library is used to access every information that is stored in an issue. The information is transformed into a new object that looks like the following code snippet:

```csharp
class BAHIssue
{
    public string Key { get; }
```
public string CurrentAssignee { get; set; }
public string Reporter { get; set; }
public List<string> Watchers { get; set; }
public List<string> Reviewers { get; set; }
public List<Comment> Comments { get; set; }
public DateTime? CreatedAt { get; set; }
public DateTime? ResolvedAt { get; set; }
public DateTime? LastUpdatedAt { get; set; }
public DateTime? DueDate { get; set; }
public string ResolutionName { get; set; }
public string IssueType { get; set; }
public string CurrentStatus { get; set; }
public List<string> Components { get; set; }
public string AffectsVersion { get; set; }
public List<string> FixVersions { get; set; }
public List<string> ParentIssue { get; set; }
public List<string> SubIssues { get; set; }
public List<string> BlocksIssues { get; set; }
public List<string> IsBlockedByIssues { get; set; }
public List<string> RelatesToIssues { get; set; }
public List<ChangelogEntry> Changelog { get; set; }
public List<Commit> Commits { get; set; }

To get all issues of a project the "Part"-relation of JIRA is used. If an issue has parts or is a part of another issue it is also checked and exported into CouchDB. When all information from JIRA for a single issue is extracted Perforce is contacted to get a list of committed files for this issue.

Afterwards the workers username has to be anonymized and mapped to a department. First the username is taken and checked against a dictionary that contains all already processed usernames. If it was found the values of the dictionary for the anonymized name and department are used. If the user could not be found the anonymized name is created by using the GetHashCode() method provided by the framework. For the department the extractor prompts the user to add the department name of the worker. Then these values are stored in the dictionary. After processing the entry for the user "tujo" in the dictionary looks like the following:

"tujo": {
The departments are mapped in the following way:

<table>
<thead>
<tr>
<th>Department</th>
<th>Mapped short name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development Web</td>
<td>devW</td>
</tr>
<tr>
<td>Development Client</td>
<td>devC</td>
</tr>
<tr>
<td>Database</td>
<td>db</td>
</tr>
<tr>
<td>Graphics</td>
<td>gfx</td>
</tr>
<tr>
<td>Product &amp; Project Management</td>
<td>ppm</td>
</tr>
<tr>
<td>Quality Assurance</td>
<td>qa</td>
</tr>
<tr>
<td>Marketing</td>
<td>mkt</td>
</tr>
<tr>
<td>IT Administration</td>
<td>adm</td>
</tr>
<tr>
<td>Controlling</td>
<td>ctrl</td>
</tr>
<tr>
<td>CIO</td>
<td>cio</td>
</tr>
</tbody>
</table>

Table 1: Mapping of departments

There is also one additional department called "tool" however this is not a department. It shows that the change was made by a tool like JIRA itself (e.g. created through a workflow) or any self-developed JIRA extension.

After completing this task the BAHIssue object is written to CouchDB. This is done by transforming the object into JSON with the Newtonsoft.Json library.

5.2 Analyzer Tool

The analyzer is a simple HTML page that uses following libraries to handle the data:

- Bootstrap (display of result tables and lists)
- vis.js (timeline visualization)
- jQuery (DOM manipulations)
- moment.js (date handling)
• pouchdb (connection to and handling of CouchDB)

When the analyzer is started it automatically connects to CouchDB and loads all available databases. A single database consists of all issues that belong to the project and those are stored in JSON format. The following snippet shows an exemplary issue in JSON:

```json
{
    "Key": "WWW-11347",
    "CurrentAssignee": "devw-1890641447",
    "Reporter": "devw-41536697",
    "Watchers": [
        "devw-41536697",
        "qa_828054671"
    ],
    "Reviewers": [],
    "CreatedAt": "2016-02-12T08:43:09+01:00",
    "ResolvedAt": "2016-03-08T09:57:28+01:00",
    "LastUpdatedAt": "2017-02-22T15:18:15+01:00",
    "DueDate": null,
    "ResolutionName": "Fixed",
    "IssueType": "Task",
    "CurrentStatus": "Closed",
    "Components": [
        "Sportsbook",
        "Webservices"
    ],
    "AffectsVersion": "3.43.0.10",
    "FixVersions": [
        "MS 3.43.0",
        "3.43.0.31"
    ],
    "ParentIssue": "WWW-11393",
    "SubIssues": [],
    "BlocksIssues": [
        "APPMAP-128"
    ],
    "IsBlockedByIssues": [],
    "RelatesToIssues": [
        "REQ-217"
    ]
}
```
"Changelog": [
  {
    "ChangedBy": "devw_822371210",
    "ChangeLogType": "status",
    "ChangedAt": "2016-03-08T09:57:18.39+01:00",
    "FromValue": "Open",
    "ToValue": "In Development"
  },
  {
    "ChangedBy": "devw_822371210",
    "ChangeLogType": "status",
    "ChangedAt": "2016-03-08T09:57:28.367+01:00",
    "FromValue": "In Development",
    "ToValue": "Resolved"
  },
  {
    "ChangedBy": "devw_822371210",
    "ChangeLogType": "fix version",
    "ChangedAt": "2016-03-08T09:57:28.367+01:00",
    "FromValue": null,
    "ToValue": "3.43.0.31"
  },
  {
    "ChangedBy": "devw_822371210",
    "ChangeLogType": "assignee",
    "ChangedAt": "2016-03-08T09:57:45.987+01:00",
    "FromValue": "devw_822371210",
    "ToValue": "devw_–1890641447"
  },
  {
    "ChangedBy": "devw_–1890641447",
    "ChangeLogType": "status",
    "ChangedAt": "2016-03-15T11:17:31.187+01:00",
    "FromValue": "Resolved",
    "ToValue": "Closed"
  }
]
"Commits": [ Johann Tuder, MSc - 1255771
The documents of the first database are loaded and the initial view is displayed.
The red box in the top left corner shows the dropdown input that allows the user to pick between different projects. Each project is stored in a single database.

The yellow box below influences the timelines. If a user adds date values and presses the "Set Dates" button the timelines are shrinked or expanded to the given values. A press on the "Reset to default Dates" button adjust all timelines to the default values.

The blue box below allows a user to switch between single metrics. If an option input is clicked the metric is calculated and a result table is shown above the timelines.

The gray box below allows to filter for issues in a project. If the checkbox beside the issue key is checked the timeline of the issues is displayed.

The purple box in the middle shows all activated timelines. A single timeline shows all events that occurred during the issues lifetime. By hovering a single event additional information is displayed.

The brown box on the right top shows all symbols that can occur on a timeline. If a checkbox is unchecked the event is removed from all timelines.

All metrics are calculated on the fly. Which means that if the user selects a metric all values are instantly calculated and displayed by using JavaScript. The algorithm of each metric is stored in a single js-file (e.g. metric 1 -> M1.js). Further every metric is a class and has to provide a calculate, getDataTable and getSummaryDataTable method. The methods are used for the following scenarios:

- calculate - triggers calculation for all issues of the project
- getDataTable - returns the whole result table with values for each issue
- getSummaryDataTable - returns the summary table (e.g. with average, maximum and minimum values)
6 Evaluation

In this section four different datasets are compared. Three of the projects were held by the same team. The fourth project is evaluated to make better comparisons to the other projects and was developed by another team within the company. In the set of the projects a major organizational change took place. During the first project (in the following called "P1") the team was organized in a horizontal manner. This means that each department (development, quality assurance, project management, database, graphics) was separated from the other and located in its own office. Starting with the second project (in the following called "P2") the team was mixed and a responsible from the project managers, graphics and quality assurance department moved into the developers’ office. This setup was also chosen for their third project (in the following called "P3"). For P3 it is also important that during data examination the project was finished for about 66%. For some metrics this means that 736 issues will be used as whole set, other metrics will use the full 1,115 issues because project state is not relevant. This won’t be mentioned in following sections and the numbers are adapted accordingly. The fourth project (called "P4") was also developed with a horizontally set up team.

6.1 M1 - Issue resolved/closed date $\leq$ issue due date

For every project it is the case that some of the issues do not have due date set. So it is not possible to calculate values for those issues. However the important issues, e.g. bigger parts of the project or issues that are necessary for a successful outcome, have a due date.

**P1**

In P1 one issue was resolved and closed 101 days after its due date. The issue is an organizational issue. The state change was directly from "Open" to "Closed". The JSON file in the database additionally shows that the issue type is "New Feature" and that it was closed with the resolution "Won’t fix". Furthermore the issue always had the same assignee and the due date was not changed at any time.

However the ticket has multiple sub-issues but it is not blocking any other issue. A look at a part of the sub-issues shows that they were moved to the "Backlog" milestone. Some of them also had the same state transition from "Open" to "Closed". From
these observations it is most likely that the issue addressed a feature set that was not mandatory for a release and was moved.

Figure 3: State transition of issue

Another issue was resolved 154 days before its due date, but it was closed 100 days later. The timeline shows that the due date was changed. On July 7th it was updated from March 18th to August 1st. This means that the change of the due date was already after the initial date. It is also the case that the issue was closed before the due date was changed. The issue has three sub issues and two of them were closed later than their parent. Also development on those two were started after the parents closed date. Usually this should not be the case. A look in the database clears things up, since one issue was a bug that could not be reproduced and the other was an improvement issue. The third sub issue was really a task of the parent issue and was fixed on time. A comparison of the timelines reveals that when the task was finished the parent issue immediately went into the testing stage.

Figure 4: Test stage after finished sub issue

The next issue was closed 112 days before its due date. The issue itself looks normal. However its due date was changed three times after it was resolved. A look in the database reveals that the issue was for the product management department and handled the textual translations for several languages.

On average issues were closed 5.8 days after and 9.2 days resolved before their respective due date.
P2

Within P2 is one issue that was resolved 203 days after its due date. A look at the timeline reveals that the issue was created after its due date. The issue naming and the assignee changes on the timeline indicate that it is a testing issue. This perspective leads towards the interpretation that the due date was incorrect since creation. If this is the case someone has to consider about the meaning and importance of due dates in the respective department. Since there is a possibility to change it afterwards. A deeper look at the parent issue shows that it does not have a due date set, however the next parent issue has a due date set which is also before the date of the first issue. In this case the due date seems to be set by accident.

![Figure 5: Issue compared to its parent-parent-issue](image)

One issue was closed and resolved 193 days before its due date. However the timeline reveals that the due date was set for the first time 7 days before its value. The analysis of a part of the sibling issues shows that this is not the general case. The analyzed part does not have a due date set. This implies that the issue has an elevated meaning, so the sub issue is analyzed. The timeline immediately shows that the sub issue has the same occurrence. There the due date was set to the same date as its parent and also 7 days before the effective day.
Another issue was closed 213 days after its due date. It was also resolved 3 days late. The due date was changed once one day after the first set date.

However further investigation of sub issues or other issues it relates to did not reveal more insights.

On average issues were closed 21.9 and resolved 4.2 days after their due dates.

P3

In P3 the biggest gap of an issue between its due date and the resolved date is 19 days after the due date. The timeline shows that the issue had a short lifetime, because it was created 6 days before its due date. Seven issues were resolved 191 days before their due date. All of those seven were graphic related issues and were created, resolved and closed at the same date. However none of the seven had a commit but was closed with the resolution ”Fixed”. Here it should be investigated, why this ticket was closed without having something submitted. A look at the ”Reporter” field of the issue reveals, that it was created automatically by a tool. An assumption is that the creation of these issues was superfluous, because everything already existed.

The biggest gap of 48 days between due and closed date was found in a database issue. A look at the timeline reveals, that this gap should not really exist.
The issue was created and resolved on the same day and the due date is set 8 days later. However it was closed 48 days later and nothing happened since it was resolved. This leads to the assumption that it was simply forgotten to close the issue.

On average issues were closed 102.1 and resolved 102.2 days before their due dates.

P4

P4 contains a database issue that has the maximum gap between due and resolved date and also between due and closed date. The issue was resolved 70 and closed 90 days after the due date. The timeline shows that the real work on the issue started after its due date. And then its status transitions were mostly produced by the quality assurance department.

Questionable is that the issue often went from "Resolved" to "In Testing". Which should only occur after the developer finished the work. The testers should not set an tested issue into a "Resolved" state, when they are going to open it again.

Another issue was resolved and closed 14 days before its due date. The timeline shows that the workflow of creation and closing was completed at the same day. However it shows something more interesting:
The due date, November 12th, was set on December 12th and the issue was created and closed October 29th. Usually due dates are set when issues are created or the work on them has started. A comparison to other issues where the due dates were changed revealed, that this is the only occurrence of such a behavior.

On average issues were closed 15.6 and resolved 8.2 days after their due date.

Comparison

Comparing the average values of the data sets P2 performs poorly. However of the top 25 issues where the resolve date was after the due date are 21 testing issues. Whereas in P1 only 2 testing issues are in the top 25. The case is similar if a look is taken on the resolve date compared to the due date. In P2 are 11 in the top 25 and in P1 0. Furthermore it seems that to P3 the statistics have drastically improved. When P4 is taken as a reference project it is relatively obvious that P1 and P2 were performing ”normal”. These values lead to the following questions:

- Did testers start later in P2 than in P1?
- Does the due date only apply for the developers?
- Were testing issues treated in a different way for P2 than for P1?
- Was P3 too pessimistically planned?
- Did the benefits of the organizational change show up in P3?

6.2 M2 - Due dates aren’t changed

During projects the work packages get a due date so that shipment of a feature or product can be guaranteed. These due dates should not be changed since this means that the product will most likely be finished later and customers have to wait. However it can happen that something happens that was not planned beforehand and those due dates
have to be adapted. In this metric it is analyzed how often and how many due dates are changed.

**P1**

During P1 149 of 1,017 issues (about 14.6%) had their due date changed. However the dataset shows 296 changes, which means that some of the issues had their due date changed more than once, like it is shown in the picture below:

The issue APPMAP-1 had its due date changed for five times. The first time the due date was changed the day after its creation for three times:

![Figure 12: APPMAP-1 with due date changes](image)

The image shows that it was always moved for another month. Which leads to the assumption that product management was still in a planning phase and could not clearly state when it had to be finished. At the end it was set to July 11th. The second occurrence of the star-shape at the timeline is on July 7th and shows that it the due date was postponed again for a month and the new date is August 1st. The last shape is already at August 3rd and sets the final due date for August 5th.

However the issue was already closed on July 8th and the last due date change seems su-
perfluous. This indicates that there is a workflow part happening, that is not represented in JIRA.

**P2**

In P2 240 of 2,686 issues (about 8.9%) had changed due dates. Also in this dataset some issues were changed more than once, because the dataset contains 562 changes. This behavior is the same like for P1. Compared to P1 the relative amount of changed issues is lower in this subset. The issue BAH-550 was changed 23 times, which is the highest number in this dataset. The issues due date was changed very often without any status change or other event between, which is visible at the timeline:

![Figure 13: Multiple changes without events in between](image)

The following snapshot of the result table shows that only the due date was mostly adapted for only a few days:

![Figure 14: Multiple changes without events in between](image)

**P3**

In P3 67 of 1,115 issues (about 6%) had their due dates changes. Compared to the two projects before the relative amount of issues decreased. There were only 4 issues that were changed more than once and they were updated only 2 times. For example the issue BAH-551 had its due date set when it was assigned to a specific milestone. The
only change occurred when the requirements were reviewed and it was started with the specification:

![Figure 15: Due date update after status change](image)

**P4**

In P4 only 19 of 939 issues (about 2%) had their due dates changed. This represents the lowest number of all projects. Also these issues were not changed often, there is only one that was updated 4 times. All others were changed less often. Also this issue was mostly edited without any other events in between:

![Figure 16: Multiple changes without events in between](image)

**Comparison**

During the three projects (P1, P2 and P3) the relative amount of changed issues decreased constantly.

<table>
<thead>
<tr>
<th>Project</th>
<th>Issues changed</th>
<th>Delta to previous project</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>14.6%</td>
<td>-</td>
</tr>
<tr>
<td>P2</td>
<td>8.9%</td>
<td>-5.7%</td>
</tr>
<tr>
<td>P3</td>
<td>6.0%</td>
<td>-2.9%</td>
</tr>
</tbody>
</table>

Table 2: Development of due date changes

This progress underlines the positive effects of the new development team organization. The values of P4 are better (only 2% of changed due dates) compared to the others. However less issues of P4 had a due date set compared to the others.
In P4 only 2.3% of all issues had a due date set. This means that the value is only that low because very few tickets had a due date that could be changed. Next it is compared how many of the issues with a due date had to be updated.

From P1 to P2 the rate of changed issues became higher, but the value dropped significantly until P3. In P4 nearly every issue had to be updated.

Comparing the values of table 3 to table 4 the positive development from P1 to P3 remains. However the first positive impression of P4 is inverted by the high number of changed issues. Under these circumstances it is obvious that a simple comparison like in table 3 is not sufficient.

### 6.3 M3 - Fix version wasn’t changed

In the case of the master thesis a fix version represents a milestone number to which an issue belongs. The metric shows how many issues had their fix version changed and how often it was updated.

#### P1

248 of 1,017 issues had their fix version changed (about 24%). Some of them were updated more than once, like in section 6.2. The most often changed issue was APPMAP-282...
with 4 changes:

<table>
<thead>
<tr>
<th>Issue</th>
<th>FromValue</th>
<th>ToValue</th>
<th>ChangedAt</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPMAP-261</td>
<td>MS 1.0.2</td>
<td>MS 1.0.3</td>
<td>2016-04-11</td>
</tr>
<tr>
<td>APPMAP-262</td>
<td>MS 1.0.2</td>
<td>MS 1.0.3</td>
<td>2016-04-11</td>
</tr>
<tr>
<td>APPMAP-277</td>
<td>MS 1.0.1</td>
<td>MS 1.0.2</td>
<td>2016-04-14</td>
</tr>
<tr>
<td>APPMAP-282</td>
<td>MS 1.0.1</td>
<td>MS 1.0.2</td>
<td>2016-04-14</td>
</tr>
<tr>
<td>APPMAP-282</td>
<td>1.0.2.28</td>
<td>MS 1.0.2</td>
<td>2016-04-14</td>
</tr>
<tr>
<td>APPMAP-282</td>
<td>MS 1.0.2</td>
<td>MS 1.0.3</td>
<td>2016-04-25</td>
</tr>
<tr>
<td>APPMAP-282</td>
<td>1.0.2.37</td>
<td>MS 1.0.3</td>
<td>2016-04-25</td>
</tr>
</tbody>
</table>

Figure 17: Overview of changed issues

The timeline shows 3 pentagons:

Figure 18: Timeline with pentagons

The pentagons and table show that fix version changes happened twice on one day. A value change from "null" to a specific value (like in the tooltip) indicates that the "to" value was newly set.

P2

In P2 740 of 2,686 issues had their fix version changed (about 28%). Like in P1 some of them had to be updated more than once. The timeline of the main project issue shows an interesting timeline:

Figure 19: Main project issue

The issue already existed for about 8 months until the first work was done. In the meantime the fix version was changed twice. At issue creation it was assigned to the "Backlog" milestone. Then it was assigned to a specific milestone and 5 months later it
was back again in “Backlog”. After the next change the work/implementation started, which is indicated by the due date and more common status changes.

**P3**

The fix version was updated for 145 of 1,115 issues (about 13%). Compared to P1 and P2 the value is lower and indicates a positive effect of the organizational change. In this project another workflow flaw occurs:

![Figure 20: Cloned graphics issue](image)

The fix version was changed to the same value it already was. The timeline or metrics table does not show more insight why this event occurred. The issues was cloned to another project within JIRA. This led to the occurrence even when it was not changed by a user. In a future version of the tool this problem has to be remembered so that it is not in the results.

**P4**

In P4 165 of 939 issues had their fix version changed (about 18%). The issue BETDB-1853 had to be changed 14 times. The updates often occurred without any other event between them:

![Figure 21: Fix version changes without events in between](image)

This indicates that the issue was not important to be finished quickly. Also development on the issue started after the 9th fix version change. Afterwards were still 5 updates necessary until the issue was finished. This implies that time planning was not good.
Comparison

From P1 to P2 the amount of updated issues increased, however it decreased below the value of P1 in P3:

<table>
<thead>
<tr>
<th>Project</th>
<th>Issues changed</th>
<th>Delta to previous project</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>24.4%</td>
<td>-</td>
</tr>
<tr>
<td>P2</td>
<td>27.6%</td>
<td>+3.2%</td>
</tr>
<tr>
<td>P3</td>
<td>13.0%</td>
<td>-14.6%</td>
</tr>
</tbody>
</table>

Table 5: Development of fix version changes

The decrease in P3 is an indicator that the organizational changes were beneficial for the project. However in P4 the amount of changed issues is 17.6%, which is 4.6% more than in P3. In the amount of updated issues is compared to the total amount of issues. Often issues are updated more than once.

<table>
<thead>
<tr>
<th>Project</th>
<th>Changed issues</th>
<th>Sum of all changes</th>
<th>Rate Sum Changes/Changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>248</td>
<td>380</td>
<td>1.5</td>
</tr>
<tr>
<td>P2</td>
<td>740</td>
<td>1,152</td>
<td>1.6</td>
</tr>
<tr>
<td>P3</td>
<td>145</td>
<td>161</td>
<td>1.1</td>
</tr>
<tr>
<td>P4</td>
<td>165</td>
<td>261</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Table 6: Ratio of changed issues

Table shows how often an issue is changed if it had to be updated. In P2 an issue had to be changed 1.6 times. The values for P1 and P4 are slightly better with 1.5 changes. In P3 the value reaches 1.1.

This means that planning was done better in P4 than in the other projects. In the relative amount of all changed issues is the lowest and table reveals that an issue was mostly updated only once.

6.4 M4 - Duration in approve state

In this case only database issues can be in the approve state. The workflow is the following:

1. A developer creates the SQL.
2. The issue status is set to "Ready for approval".

3. The developers teamlead approves the SQL and sets the status to "Approved".

4. A database engineer prepares patches and applies them to the database.

For this metric it is interesting to measure the time between item 2 and 3.

**P1**

In P1 were only 2 database issues that needed to be approved. One issue were 6 days in approve state and the other was approved immediately.

![Figure 22: Issue in approve state](image)

In the case of the issue above the approval process prevented a faster workflow, because the database team was able to close the issue on the same date. The issue had to be reopened later. It seems that something was wrong in the SQL and the issue was reassigned to the developer. The developer updated the SQL and passed it again to the database team. Here a major workflow flaw occurs, because a reopened issue does not have to be approved again.

![Figure 23: Issue passed to developer](image)

![Figure 24: Issue reopened and resolved without approval](image)

On average the issues were 3 days in approve state.
P2

In P2 database issues were handled as it is expected by the workflow. There were 8 issues. 2 of them were in approve state for 1 day. The others were approved immediately. This leads to an average of 0.3 days per issue.

P3

In P3 were also only 2 database issues and both were fixed on the same day.

Figure 25: Issue immediately approved

The immediate approval enabled the developers to resolve and close the issue on the next day. Nobody was stopped by the approval process.

P4

In P4 occurred 70 database issues with an average duration of 1.7 days. There were 14 issues that needed longer than a day for approval. The longest lasting issue needed 29 days until it was approved:

Figure 26: Issue with long approve state

In the 29 days no other event happened. When it was approved it was also immediately closed. This shows the weakness of the workflow, when such an issue is forgotten, it blocks other developers from doing their work.

Comparison

In the following table the average values for the duration in approve state are compared:
<table>
<thead>
<tr>
<th>Project</th>
<th>Database issues</th>
<th>Average duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>8</td>
<td>0.3</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>P4</td>
<td>70</td>
<td>1.7</td>
</tr>
</tbody>
</table>

Table 7: Issue duration in approve state

The average duration declined from P1 until P3. In P3 the optimum was reached, because all issues were approved on one day. In P4 nearly 2 days were needed to approve an issue. Usually approval should not last longer than 1 day. In P1 and P4 were issues that needed longer than 1 day. These have to be checked by the process responsible so that flaws can be removed and can not happen again.

6.5 M5 - Re-open rate of issues closed shortly before their due date

In metric 5 all issues that have a due date and were re-opened during their lifetime are analyzed. This is used to see if e.g. more re-opens occur closer to the due date.

P1

The set contains 34 re-opened issues, in sum they were re-opened 53 times. On average the 34 issues were re-opened 1.6 times. The following issue was re-opened 2 times:

![Re-opened issue](image)

Figure 27: Re-opened issue

All 2 re-open status changes happened on the issues creation date. The state transitions indicate a mis-usage of the workflow. If a issue is paused it should be set to "Suspended" and not to "Open".

In the total set of 1,017 issues the 34 re-opened make up 3.3%. The issue with the smallest duration between due date and re-open date was re-opened once. The average duration
of all re-opened issues between due date and re-open date are 8 days before the due date.

P2

Out of 2,686 issues 94 were re-opened (about 3.5%). In sum the 94 issues were re-opened 163 times, which is an average of 1.7 re-opens per issue. When the duration between re-open and due date is smaller, the sum of re-open state changes for particular issues is higher. For example one issue was re-opened 143 days before its due date. This was the only re-open process of the issue:

![Figure 28: Re-opened issue with 143 days duration](image)

The issue GFX-4950 was re-opened 5 times. The last re-opens occurred 3 days before its due date.

![Figure 29: 5 times re-opened issue](image)

This issue has the same problem as the issue from P1. It seems that the workflow was either mis-used or that the workflow does not provide a way to pause an issue. The average duration between re-open and due date are about 6 days before the due date.

P3

In the set of 736 issues 55 had to be re-opened (about 7.4%). The issues were re-opened 89 times, which are about 1.6 re-opens per issue. In this dataset it is not observable that issues were more often re-opened when the last re-open change was close to the due date. The average duration between re-open and due date is about 93 days before the
due date. The average value is that high, because the project was handled different to the two above. A lot of issues already had a due date from the beginning of the project. This was not the case for the other projects.

P4

In P4 only 4 of 939 issues were re-opened (about 0.4%). Compared to the other projects the value is very good. Concerning the info that only very few issues had a due date in this project (mentioned in section 6.2) this is not surprising. However the 4 issues were re-opened 7 times, i.e. that a single issue was re-opened 1.8 times. Furthermore the average duration between re-open and due date is about 11 days before the due date.

Comparison

From P1 to P3 the rate of re-opened issues increased:

<table>
<thead>
<tr>
<th>Project</th>
<th>Re-opened issues</th>
<th>Re-opens per issue</th>
<th>Average duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3.3%</td>
<td>1.6</td>
<td>8</td>
</tr>
<tr>
<td>P2</td>
<td>3.5%</td>
<td>1.7</td>
<td>6</td>
</tr>
<tr>
<td>P3</td>
<td>7.4%</td>
<td>1.6</td>
<td>93</td>
</tr>
<tr>
<td>P4</td>
<td>0.4%</td>
<td>1.8</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 8: Re-opened issues

An increase does not implicitly mean that the project was organized worse. The reasons for re-opened issues can be e.g. a bug found during testing or setting the issue state in a false way. Like described above it should not be the case that an issue state changes from "In Development" to "Open". Usually a paused issue should be set to "Suspended". The issues above demonstrate that the status options are not used as desired and as prescribed by the workflow.

The average value of re-opens per issue also increased from P1 to P2, but stayed the same for P3. The values for the 3 projects are all acceptable. The value of P4 should not be the case for any other project. Issues that are re-opened more than twice indicate that either the programmer made several mistakes during development, the requirements were not clearly formulated or the developer mis-used the status options for the workflow.

The average duration value for P3 is very good. The issues were re-opened 93 days before
their due date, which indicates that the work of the developers had high quality even when it was close to the due date. Small values would show that developers were stressed by the upcoming due date and more mistakes in their code. However the average values of all projects do not show that there is a problem with more failures when issues have to be resolved closely to their due date. Some single issues face the mentioned problem.

### 6.6 M6 - Assignees of an issue

With this metric the department changes of an issue are compared. It is investigated if the organizational changes led to lower or higher re-assignments among the departments. There are two different values defined for changes:

- inner department (e.g. from one web developer to another)
- outer department (e.g. from web developer to tester)

**P1**

In P1 are 3 issues with 9 inner department changes. The 3 issues are from the departments graphics, quality assurance and development web.

The image shows that one node can show multiple assignee changes which occur at the same day. The graphics issue is the only one which does not have an assignee change after resolving. This is a normal case since they are verified when the respective code is tested. The graphics and the testing issue are not passed to other departments.
However the development issue (WWWMOB-3405) is additionally passed between departments. The issue seems to be a special case since it was only in the beginning assigned to a developer. Afterwards it was moved among the project management and quality assurance departments or within them. This implies that there was not any developing activity necessary. A look in the database verifies this assumption because there was not a single file submitted to the code repository. Furthermore the component of it is ”Visualization/Frontend”.

The 3 issues with the highest number of outer changes are also displayed below.

![Figure 31: Issues with most outer changes](image)

The issue with the most outer changes (14) is REQ-221. This is a requirement issue and usually defines a part of work, often in sub-issues. In this case there are not any sub-issues. However it was assigned to multiple departments. It was created by project management, passed to development web and afterwards to graphics. Then all outer department transitions (except for one) are between project management and graphics. Which indicates that the issue was about a design decision. The missing of a code submission further supports this theory.

The second issue has 13 outer changes (APPMAP-538). The timeline investigation shows a lot of changes between development web and quality assurance. If the status changes
on the timeline are checked it is obvious that there was implementation work done which
did not fulfill the quality standards or requirements, because the tester had to re-open
the issue 4 times. The issue was first resolved at 2016-06-05, the final resolve date was
at 2016-06-27. This means that there were 22 days of testing and re-implementing code
which could be avoided if tester and developer were sitting next to each other.
The same observations were made for the issue APPMAP-522 which has 11 outer depart-
ment changes. However there is a small difference observable on the timeline compared
to the before mentioned issue. On nearly every status change also a comment was made
in JIRA. This is not the case for the other issue. This indicates that here main commu-
nication was made through the commenting feature of JIRA and in the other case via
Skype.
At the end the 3 issues with the most combined changes are investigated. All 3 were
already mentioned in one of the cases above. APPMAP-538 has a total of 17 changes,
REQ-221 has 15 and WWWMOB-3405 has 13.
The average of all inner transitions is 0.9, outer transitions is 2.3 which sums up to 3.2
changes on average.

P2

In P1 has 1 issue 12 inner department changes and 3 follow-ups have 10. The issue with
the most inner changes belongs to the graphics department. Also one of the follow-ups
counts to them. It should be investigated why there are a lot of changes at the same day
between one department. This can be seen in the image below.

![Figure 32: Graphics issues with most inner changes](image-url)
The timeline shows that there are comments made on assignee changes. The comments are made from the people who were assignees at this particular change. This implies that changes are documented (as comment) in JIRA.

The other 2 issues belong to the quality assurance department for testing. It is very common that test issues are passed often within the department and there is mostly only one change per day. When there were multiple changes it was because of a requirement change or the issue went in review and was opened again.

![Figure 33: Quality assurance issues with most inner changes](image)

Next issues with the most outer changes are investigated. 2 of them have 13 and another 2 have 12 changes. All 4 have in common that they are for the development department.

The 3 issues in the image below were mainly passed between quality assurance and development web department. This represents a normal workflow when issues went from development to testing and back if something did not work as expected.
However the fourth issue was often passed between "null" and the development department. In this case the value null symbolizes that the issue was or is un-assigned. In this case also the database does not give a good explanation. The type is a bug issue that was fixed and closed properly but nothing was submitted into the code repository. A look into JIRA clears things up. In the description it is mentioned that some views and implementations of the last iteration have to be verified again by the developers and quality assurance.

Looking at the combined changes the GFX-5081 issue has the most with 20. The next are APPCASINO-439 with 16 and APPCASINO-579 with 15 changes. All of them were mentioned above in detail.

The average of all inner department changes is 1.0, outer transitions are 1.2 and those sum up to a combined average of 2.2.
P3

The most intra department changes in P3 were 8 and these occurred for 6 issues (e.g. APPSPORTS-28). The most cross department changes were 13 for 1 issue (APPSPORTS-324). The issue with the most cross department changes also had the most changes in total with 18.

Figure 36: Most changed issues

Both timelines show that assignees were often changed when also the status of the issue changed. This is the case when e.g. a developer finishes the implementation and passes the issue forward to the quality assurance department. However there are also changes without status updates.

On average an issue was re-assigned 1.9 times. 0.7 times within the same department and 1.2 time to another department.

P4

In P4 the issue BETDB-1605 had to be re-assigned within the same department for 20 times:

Figure 37: Issue with most intra department changes

It also has a lot of status changes and it had to be re-opened 5 times. This is not how a issue should be handled. This issue has also the most combined department changes.
Another issue with the most cross department changes is also a BETDB-issue. On average an issue had 2 intra department and 1.6 cross department changes. This means that a single issue changed its assignee for 3.6 times on average.

**Comparison**

P2 has a lower average value than P1 for outer department and combined changes. As shown in table 9 P3 decreased the average values for intra department changes in comparison to P2. The lower value indicates a possible better communication between team members. If a project manager or a tester can precisely describe a problem to a developer because they are sitting in the same room the issue does not have to be passed to each other.

<table>
<thead>
<tr>
<th>Project</th>
<th>Avg intra</th>
<th>Avg cross</th>
<th>Avg sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0.9</td>
<td>2.3</td>
<td>3.2</td>
</tr>
<tr>
<td>P2</td>
<td>1.0 (+0.1)</td>
<td>1.2 (-1.1)</td>
<td>2.2 (-1.0)</td>
</tr>
<tr>
<td>P3</td>
<td>0.7 (-0.3)</td>
<td>1.2 (0.0)</td>
<td>1.9 (-0.3)</td>
</tr>
<tr>
<td>P4</td>
<td>2.0</td>
<td>1.6</td>
<td>3.6</td>
</tr>
</tbody>
</table>

Table 9: Issue assignees

Also the similar value of average inner changes leads to the assumption that communication within a department is easier and does not need an issue ping-pong via JIRA. The values for P4 were higher, but are comparable to P1. This indicates that the old organization form resulted in a bigger overhead, because issues were passed to each other more often (inner and cross department wise).

**6.7 M7 - Commentators of an issue**

With the metric it is analyzed if the amount of comments have an influence on status changes or lifetime of an issue. Additionally the length of comments and number of commentators are analyzed.

**P1**

In P1 834 of the 1,017 issues were commented (about 82%). The average length of those comments is 93 characters. The comments were on average made by 1 user and on average
were 2 comments per issue.

The following issue has 25 comments, the most in the set:

![Figure 38: Most comments](image)

The timeline shows that every time a comment was made also the status changed. Only once this was not the case. The status changes most occurred because the issue had to be re-opened. This implies that requirements were probably not well written and therefore comments were necessary.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Set of Commentators (Comments per Commentator)</th>
<th>max length</th>
<th>min length</th>
<th>avg length</th>
<th>comment count</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRAN-16905</td>
<td>mdk_1586914184 (1) tool_155948924 (18) ppm_.185964776 (1) ppm_21028642 (5)</td>
<td>97</td>
<td>1</td>
<td>46</td>
<td>25</td>
</tr>
</tbody>
</table>

![Figure 39: Commentators of issue](image)

The "Set of Commentators" column reveals that 18 of the 25 comments (72%) were made by an external tool and not a person. This is the case for 6 issues in the top 10 of most commented issues. JIRA reveals why this occurs. The issue is for translations of a specific term into multiple languages. When translations are finished the tool combines them into one package that is applied to the database, so that the text can be used on the website. Because some of the translations were wrong or the requirement mis-leading the tool had to be started several times.

Also the minimum length displayed in the picture above is questionable, because a comment with only one character does not make sense. In JIRA it is observable that with the comment of a single character the workflow for the tool is triggered, which is not a good use of the comment-function.

P2

In P2 2,013 of 2,686 issues were commented (about 74.9%). The average length of a comment is 118 characters. The average of users and comments per issue is the same as for P1.

The most commented issue also as 25 comments. Compared to P1 the comments
appeared more often independent from other events:

![Figure 40: Most comments and more independent](image)

The number of status changes would indicate that something went wrong during development. This is not the case, since the issue was often suspended and went afterwards again into development. However it has to be clarified if these are too many status changes and what was the cause for the status updates. The issue was commented by 3 different developers of the quality assurance department, where one made 16, another 8 and the third 1 comment. In a way of communication the amount of comments should even out, i.e. if user 1 makes 10 comments, other user should also make 10 (+1) comments. Furthermore it is observable that comments were used when development started. The first comment was made 9 months after the issues creation. Starting from this point comments were regularly made.

**P3**

In P3 523 of 736 issues were commented (about 71.1%). On average a comment is 100 characters long. In difference to P1 and P2 the average commentators of an issue is only 1 user.

The most commented issue was again commented 25 times, but by a single user.

![Figure 41: Most comments, single user](image)

Despite of the first comment every other comment was 19 characters long. This indicates that some kind of workflow is represented within the comments. Furthermore every comment was created when a status was changed. Since this issue is from the quality assurance department, the comments show a different behavior.
assurance department this is the case, because in their workflow they have to state why they suspended a test. After analyzing the issues with the longest comments it was observed that the long comments were made before an issue was tested.

![Figure 42: Long comment](image)

![Figure 43: Testing after long comment](image)

Furthermore it was observed that most of the long comments were written by a user of the quality assurance department. This happens because they describe what is done for testing an issue or how they found a bug. There are also long comments from developers, they are mostly in conversations with a quality assurance user and describe how they fixed it or how they tested the solution for a bug.

**P4**

In P4 765 of 939 issues were commented (about 81.5%). The average length of a comment is 123 characters. The average commentators and comments per issue are the same as for P1 and P2. The most commented issue was commented 23 times and only 2 comments were not at the same time to another event. In comparison to the other projects no new observations could be made.

**Comparison**

The following table shows the values of all the projects:
<table>
<thead>
<tr>
<th>Project</th>
<th>Commented issues</th>
<th>Avg length</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>82.0%</td>
<td>93</td>
</tr>
<tr>
<td>P2</td>
<td>74.9%</td>
<td>118</td>
</tr>
<tr>
<td>P3</td>
<td>71.1%</td>
<td>100</td>
</tr>
<tr>
<td>P4</td>
<td>81.5%</td>
<td>123</td>
</tr>
</tbody>
</table>

Table 10: Comments of an issue

In table 10 no clear distinction between the projects can be made. For P3 it has to be remembered that the project is not fully finished and therefore the value tends to be lower than for the other projects.

In P1 and P3 it was shown that some departments use the comment function for workflow related steps. This biases the numbers and analysis heavily.

It could not be observed that a higher number of comments influenced the lifetime of an issue. However it was shown that comments are mainly written when the status of an issue changes.

### 6.8 M9 - Issue re-assignment without any communication

Usually an issue should be assigned to another person if the workflow is designed that way (e.g. an issue has to be tested and is assigned to quality assurance). Another option is that an issue is passed to a different user because of workload or pressure to e.g. fix a bug.

If the second option occurs a comment should be made why the ticket was re-assigned.

**P1**

In the set 644 of 1,017 issues (about 63.3%) were re-assigned without communication. On average issues were re-assigned 1.8 times and the maximum number of re-assignments without communication for one issue was 14.

![Figure 44: Most re-assignments without communication](image)

On the timeline every re-assignment occurred simultaneously to a status change. This
represents a normal workflow. The issue is a project management issue and contains the requirements for a specific feature. Therefore it should not be the case that it has that many re-assignments, especially it should not be re-assigned that often between the departments, because the workload for a specific department is handled in a sub-issue.

The next issue had the longest duration between 2 assignments without communication.

Also here the changes only occurred simultaneously to status changes. Like for the issue above it is a normal workflow, since the issue went into testing change and therefore the assignee had to be changed.

P2

1,408 of 2,686 issues (about 52.4%) were re-assigned in P2. On average issues were re-assigned once and the maximum number of re-assignments without communication per issue was 11.

The issue above had 11 re-assignments. Within the red-shape are assignments to other users without a status change or comment. The changes were always within the web development team. This is not representing the workflow. In this case it is not possible to tell why the assignee was changed, even when the issue is looked up in JIRA. This means that information got lost and the developers have to be asked why this happened.

The following figure shows that when the issues are ordered by their average duration between re-assignments without communication the issues from the quality assurance department make up 9 of the top 10:

Figure 45: Longest duration between re-assignments

Figure 46: Re-assignments without status change
For the exemplary taken issues it is always the case that the assignee was set to null. Afterwards it is assigned to a quality assurance employee and the work starts. Since test-plans are written at the start of a milestone or project this is a normal workflow behavior. The plans are written and are suspended until development finished and the feature can be tested. At this time a tester takes the issue and starts testing.

P3

In P3 450 of 736 issues (about 61.1%) were re-assigned without communication. On average an issue is re-assigned 1.4 times and the maximum of re-assignments per issue was 14.

In the figure above occur a lot of re-assignments. At the beginning the re-assignments were made without comment or status change. Only two time a status change happened. After the issue went into development on December 18th comments were made with every status and assignee change. The end of the timeline represents how the workflow is intended and also how status and assignee changes remain understandable for other departments and analysis.
658 of 939 issues (about 70.1%) were re-assigned. The average re-assignment rate per issue is 2. The issue with the maximum number of changes was changed 21 times.

Figure 49: Most re-assignments with comments during lifetime
The figure above shows a lot of re-assignments and only at 3 points the change was made without a comment. However it is the issue with the most changes without a comment between them. This is because on single days the issue was re-assigned multiple times to other users but not on every change a comment was made. This is illustrated in the next figure:

Figure 50: Most re-assignments with comments during lifetime
On April 28th were 3 comments made and the assigned was changed for 5 times. This means that 2 assignments were made without a comment. This happens on several points at the timeline and therefore the value for this issue is that high.

Comparison
The metric showed that the comment function is not used in that way that it can deliver a valid output about beneficial changes for team organization. It shows that often changes are made without communication or status change. This is a problem because there is...
some information hidden why this change was necessary. This would make long term
evaluation hard because users have to remember why this happened.

<table>
<thead>
<tr>
<th>Project</th>
<th>Re-assignments</th>
<th>Avg re-assignments per issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>63.3%</td>
<td>1.8</td>
</tr>
<tr>
<td>P2</td>
<td>52.4%</td>
<td>1.0</td>
</tr>
<tr>
<td>P3</td>
<td>61.1%</td>
<td>1.4</td>
</tr>
<tr>
<td>P4</td>
<td>70.1%</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Table 11: Comments of an issue

Table 11 shows that from P4 to the other projects all values decreased. It can be care-
fully stated that organizational changes may have a positive effect on the number of
re-assignments. However it seems to have a lower impact on the number of issues that
have to be re-assigned during a project. This reason therefore might be the workflows
design.

6.9 M11 - Issue re-assignments without status changes

In the designed workflow an issue is re-assigned to another user when the status of the
issue changes. This is the case when e.g. a developer implemented a fix for a bug and
the fix has to be tested. Then the developer sets the status ”Resolved” and passes the
issue to a quality assurance engineer. The tester sets the status ”In Testing” and checks
if everything works as expected. The metric should show if this is the case or if issues are
re-assigned without any status change.

P1

In P1 458 of 1,017 issues (about 45.0%) were re-assigned without a status change. The
average number an issue was re-assigned is 0.7 and the maximum of re-assignments for
one issue is 7.

Figure 51: Most re-assignments without status change
The figure shows that only in the beginning were re-assignments on different days without status changes in-between. The other points on the axis mainly have a pair of status change and re-assignment. However at some points an issue was re-assigned multiple times and the state of the issue did not change on the same day. At some points of the workflow this is the expected behavior. When a developer finishes an issue it is passed to the quality assurances team-lead. The team-lead assigns the issue to one of the testers without altering the status of the issue.

**P2**

807 of 2,686 issues (about 30.0%) were re-assigned during P2. On average an issue was re-assigned 0.4 times and the maximum was 13 re-assignments for one issue. The most re-assigned issue was from the graphics department. The metric shows a main flaw of issue handling:

<table>
<thead>
<tr>
<th>Issue</th>
<th>Nr. of assignments</th>
<th>Max Duration (in days)</th>
<th>Min Duration</th>
<th>Avg Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFX-5081</td>
<td>13</td>
<td>6</td>
<td>0</td>
<td>1.1</td>
</tr>
</tbody>
</table>

Figure 52: Low average duration

The issue was re-assigned 13 times and the longest duration between two assignments without a status change was 6 days. However the average duration between re-assignments is only 1.1 day. This underlines the findings of P1 that multiple assignments are made on the same day without a status change in between.

**P3**

In P3 371 of 736 issues (about 50.4%) had to be re-assigned. On average an issue was re-assigned 0.9 times and the maximum is 12.
Figure 53: Low average duration and multiple changes on same date

The figure shows how the cases of P1 and P2, which are also present in P3, emerge. From the analysis of the provided data it is not possible to determine if this is intended by workflow or not.

P4

In P4 322 of 939 issues (34.3%) were re-assigned without status change. The average number of re-assignments is 0.7 and the maximum is 10.

The issue with the most changes is shown in figure 50. For this metric appear only two points for assignee changes that are independent from a status change at the same day. To see if there is a general overlap between this metric and metric 9 the top 10 issues of both are displayed:

<table>
<thead>
<tr>
<th>Rank</th>
<th>M9</th>
<th>M11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BETDB-1605</td>
<td>BETDB-1605</td>
</tr>
<tr>
<td>2</td>
<td>BETDB-1475</td>
<td>BETDB-1611</td>
</tr>
<tr>
<td>3</td>
<td>WWW-8381</td>
<td>WWWMOB-2249</td>
</tr>
<tr>
<td>4</td>
<td>WWW-8399</td>
<td>BETDB-1525</td>
</tr>
<tr>
<td>5</td>
<td>BETDB-1525</td>
<td>WWW-6062</td>
</tr>
<tr>
<td>6</td>
<td>BETDB-1611</td>
<td>WWW-8631</td>
</tr>
<tr>
<td>7</td>
<td>WWW-6062</td>
<td>WWWMOB-2243</td>
</tr>
<tr>
<td>8</td>
<td>WWWMOB-2243</td>
<td>WWWMOB-2244</td>
</tr>
<tr>
<td>9</td>
<td>BETCLNT-10619</td>
<td>WWWMOB-2245</td>
</tr>
<tr>
<td>10</td>
<td>BETDB-1688</td>
<td>WWWMOB-2250</td>
</tr>
</tbody>
</table>
Table 12: Top issues of M9 compared to M11

In P4 40% of the top 10 issues are overlapping in comparison to M9.

Comparison

The workflow is designed that there are some assignee changes when the issue is in the same state. However the projects show that for some issues those re-assignments occur more often than intended by the workflow. The transitions seen in figure 53 have to be investigated in more detail. It is not the case that an assignee is removed and the issue stays unassigned. This points towards an implicit workflow that is not represented in JIRA by setting a new state for the issue. The projects show that they all suffer the same problem that there are more re-assignments on a single day than status changes.

<table>
<thead>
<tr>
<th>Project</th>
<th>Re-assignments</th>
<th>Avg re-assignments per issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>45.0%</td>
<td>0.7</td>
</tr>
<tr>
<td>P2</td>
<td>30.0%</td>
<td>0.4</td>
</tr>
<tr>
<td>P3</td>
<td>50.4%</td>
<td>0.9</td>
</tr>
<tr>
<td>P4</td>
<td>34.3%</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Table 13: Comments of an issue

The table indicates that the organizational change did not have a positive influence on the re-assignment changes. However it is possible that the organizational change can not influence these numbers because there are "designed" assignee changes within the workflow. This means that for a part of the issues it is intended that they have a transition. The metric can support in long term evaluation to verify that the numbers do not increase and e.g. do not exceed a threshold.

6.10 M12 - Duration between re-assignment and first status change

An issue is assigned to another user if there has to be work done that the current user should not (because of the workflow) or can not (e.g. because of responsibilities) do. After a re-assignment the work should go on immediately in most cases or only few days later. With this metric it is analyzed how long an issue is inactive (i.e. without status change) after a re-assignment.
P1

In P1 occurred 2,105 re-assignments that were followed by a status change. Compared to the 1,017 issues in the dataset each issue was re-assigned and status updated about 2.1 times. The average duration between re-assignment and next status update is 2.8 days. The maximum value is 144 days.

The issue of the above figure was re-assigned at the beginning of its lifetime, but without status change in between. After the re-assignment the fix version was changed twice and afterwards the issue status immediately switched from "Open" to "Closed". This indicates that at project start the feature was meant to be necessary but later it was not needed any longer. Therefore it was simply closed and did not have other states. In CouchDB the resolution of the issue is "Won’t Fix" and supports the assumption.

P2

P2 had 4,018 changes for 2,686 issues (about 1.5 changes per issue). The average duration is 4.4 days and the maximum value is 282 days. Compared to P1 the changes per issue decreased but the average duration increased.

In the whole lifetime of the issue displayed in the figure happened 3 re-assignments. Only 2 contribute to the metric because they were followed by a status change. In the results table both changes with their durations can be seen:
At the first re-assignment the status was set from "In Development" to "Open". The JIRA workflow allowed this, but it should not be done. When an issue is paused it should be set to "Suspended" and a comment why it is suspended should be written.

**P3**

In P3 occurred 1,210 re-assignments for 736 issues (about 1.6 changes per issue). The average duration between re-assignment and status update is 1.9 days and the maximum value is 111.

In this project the top 10 issues with the longest duration between an assignment and status change consist of 8 graphics issues.

The figure shows that the 3 issues were handled in a similar way. About a month after their creation they were assigned to a user of the graphics department. Nearly 3 months after their assignment they were resolved by the graphics department. In this case it should be analyzed why there was such a gap. Had the graphics so much to do in the meantime? Where they issues blocked by any other issue or decision?
P4

P4 had 2,230 changes for 939 issues (about 2.4 changes per issue). The average duration between an assignment and a status change is 6.8 days and the maximum value is 204 days.

In this case the issue BETDB-1605 shown in figure 50 is analyzed again. In M6 the issue had a sum of 33 changes. In this metric the issue occurs 22 times. The sum of all durations is 23 days, the average is 1.05 days. However there were only 5 changes that at least had 1 day as duration. With the metric it is possible to easily see why 5 of 22 changes needed longer and the problem can be analyzed by team-leaders.

Comparison

In the following table the values of each project are compared:

<table>
<thead>
<tr>
<th>Project</th>
<th>Avg update per Issue</th>
<th>Avg duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>2.1</td>
<td>2.8</td>
</tr>
<tr>
<td>P2</td>
<td>1.5</td>
<td>4.4</td>
</tr>
<tr>
<td>P3</td>
<td>1.6</td>
<td>1.9</td>
</tr>
<tr>
<td>P4</td>
<td>2.4</td>
<td>6.8</td>
</tr>
</tbody>
</table>

Table 14: Average values for duration between assignment and status change

The table[14] shows a positive trend with the organizational change. In general the values for P1, P2 and P3 are lower than for P4. Especially the average duration of P4 shows that there was a problem in project handling and that probably some departments had too much work packages assigned and needed longer to complete or start with them. In P3 only the average re-assignments per issue where a little bit higher as for P2 (+0.1) but it has the lowest value in the average duration.

6.11 M13 - Issue resolved date compared to code freeze date

During a project all issues should be resolved before the milestones code freeze date. After code freeze only bugs should be solved, no new features are implemented. In an ideal case there should not be a single issue that is in development after code freeze. In this metric it is analyzed if there are any issues resolved after the code freeze and how long they stay unresolved.
**P1**

For P1 one issue was resolved 256 days before (WWWAPP-109) and another 41 days after (APPMAP-387) the code freeze.

![Figure 58: Highest differences to code freeze date](image)

On average issues were resolved 127.3 days before the code freeze. The figure above shows a problem that was mentioned in sections before. The issue APPMAP-387 was created and had a due date. It was immediately closed and nothing was implemented. This means that there were organizational flaws in the workflow. Either the issue became unnecessary over time or the initial creation was superfluous.

**P2**

P2 has a slightly better average value, issues were resolved 107.3 days before the code freeze. However the maximum value is 1035 days before code freeze. This happened because the issue was linked to the project long after its creation. The issue covers maintenance of a build job to activate statistics. Since it existed that long it was quickly to resolve after it was linked to the project. Another issue was closed 121 days after code freeze.
Figure 59: Maximum and minimum duration

In this case the issues with maximum and minimum duration are quality assurance issues. Because they are created early in the lifetime of a project and also have maintenance topics to cover, it is not surprising that they are often placed at the end or beginning of this metric.

**P3**

The values for P3 can not be taken as valid, since the project is still ongoing. However for the already completed issues the average is that they are resolved 133.8 days before code freeze. The maximum is 278 days and the minimum 123 days before code freeze. Compared to P2 no test issues occurred. A reason for this could be that testing has not started yet, because it has to be waited until more issues are resolved and the project can be tested for the first time.

**P4**

The values of P4 are very different to those of the prior projects. The average duration between code freeze and resolve date is 37.5 days before code freeze. The maximum value is 333 before and the minimum 554 days after code freeze. Like in P3 the border issues are development issues, but testing issues occur more often than in P3.
The figure shows that on code freeze (mid 2015) nothing was done with the issue. The developers began working with the issue nearly 1.5 years after code freeze. This implies that the issue was still linked to the project even when it was not relevant for the successful outcome of it.

Comparison

The average numbers of each project show that the issues are resolved before the code freeze (on average). At first the high averages of P1 to P3 suggest a better performance in comparison to P4. However it has to be discussed if project planning was not accurate enough. It is possible that project managers thought it will need a longer time to finish the project. This is also a problem in project organization. If a project is planned to last longer, less projects can be developed by the developers. Also the issues that are closed long after code freeze have to be analyzed. The issues indicated that there is a step necessary to unlink issues from a project, if they do not belong to the project any longer or are postponed to another project.

6.12 M14 - Issues moved to next milestone

Before a project is started it is defined when it should be released for public access and is assigned to a specific milestone. However it is possible that something went wrong during planning or development and a feature can not be delivered. Then the project or parts of it are moved to another milestone. In this metric it is analyzed how often and how many issues are moved to other milestones.

P1

Out of 1,017 issues 128 have been moved to another milestone (about 12.6%). These issues had their milestone updated 141 times, which is an average of 1.1 updates per issue.
The following figure shows the timeline for an issue that had its milestone changed twice:

![Milestone change](image1)

**Figure 61: Milestone change**

The timeline shows 3 pentagons that represent fix-version changes. A milestone is also a fix-version and is displayed in one of these tooltips. The second line of the tooltip means that the issue had a specific fix-version but had to be updated. This is the case when e.g. an issue is resolved and gets this specific fix-version and other parts of the project can not be finished on time. Then also the already finished parts have to be moved to a new milestone because they can not be released without the other part. This is the case for the timeline above. The mentioned status change is displayed below:

![Resolved issue](image2)

**Figure 62: Resolved issue**

**P2**

In P2 611 of 2,686 issues were updated (about 22.7%). The moved issues’ milestone were changed about 1.4 times per issue.

![MS moved without status changes](image3)

**Figure 63: MS moved without status changes**

The figure above reveals a problem that should be analyzed and discussed with the responsible developer or project manager. The first time the milestone was updated 4 days after the issues creation. Until the second update 3 weeks later nothing changed despite the assignee. Then due dates were updated twice and at the end development started after the due date. Analyzing this timeline the following questions occur:
1. Why was the MS updated shortly after issue creation? Was there a communication between developer and project manager that could have avoided this change, because it would have been better planned?

2. Why did nothing happen until the second change? Had the developer too much to do?

3. Why was development started after the due date?

These problems have to be analyzed by a team responsible (e.g. team-lead or group-lead) so that this flaw can be avoided in future projects.

P3

In P3 56 of 736 issues had their milestone changed at least once (about 7.6%). These 56 issues were changed 59 times which are 1.1 changes per issue.

In comparison to P1 and P2 no further insights are gained from P3.

P4

P4 had 112 of 939 issues changed (about 11.9%). On average an issue was changed 1.3 times.

The following figure shows the same flaw like P2 but with an even longer timespan until work on the issue started:

![Figure 64: MS moved 5 times until first status change](image)

Comparison

First the results of each project are compared:
<table>
<thead>
<tr>
<th>Project</th>
<th>Ratio of updated issues</th>
<th>Updates per issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>12.6%</td>
<td>1.1</td>
</tr>
<tr>
<td>P2</td>
<td>22.7%</td>
<td>1.4</td>
</tr>
<tr>
<td>P3</td>
<td>7.6%</td>
<td>1.1</td>
</tr>
<tr>
<td>P4</td>
<td>11.9%</td>
<td>1.3</td>
</tr>
</tbody>
</table>

Table 15: Average values for duration between assignment and status change

The ratio of updated issues is the lowest for P3. This indicates that the organizational change and different handling of the project positively influenced the ratio of issues that needed its milestones updated. For all projects the number of updates per issue is low. This is good because it shows that most of the project is correctly planned and an adaption is rarely needed. For a small number of issues occur problems that have to be analyzed, which is shown for P3. Also it has to be investigated if a milestone of an issue is changed because its sub-issues or parent-issue was updated.

6.13 M15 - Issues of next milestone started in current milestone

If projects are planned correctly it should be the case that developers have enough work and issues until the day of code freeze. However it happens that issues of the following milestone are already started in the prior milestone and indicate that developers need more issues for a milestone. With this metric it is investigated if this problem occurs for the projects and if there are a lot of issues affected.

P2 in P1

In this section it is checked if any issues of P2 were already started before the code freeze date of P1. On average issues of P2 were started 211 days after code freeze of P1. One issue was started 707 days before (TAW-731) and the other 448 days after (WWW-15501) code freeze.
Figure 65: Issues of P2 with highest distance to code freeze date

The issue that started 448 days after code freeze is a testing issue. The data shows that issues that were started late consisted of testing, developing and graphics issues. However a lot of testing issues were started early like TAW-731. An analysis of these issues suggests that they are not project specific. They are mainly for maintenance of the testing environment and most probably were not planned as a part of the project.

P3 in P2

On average issues of P3 were started 95 days after the code freeze of P2. One issue was started 150 days after and another 56 days before the code freeze. However it has to be remembered that P3 is still ongoing and not finished. The following figure shows that milestone planning for P2 was better than in other projects:

<table>
<thead>
<tr>
<th>Issue</th>
<th>First Progress</th>
<th>Delta To Next MS</th>
</tr>
</thead>
<tbody>
<tr>
<td>WWWAPP-230</td>
<td>2017-07-03</td>
<td>56</td>
</tr>
<tr>
<td>WWWAPP-244</td>
<td>2017-08-21</td>
<td>7</td>
</tr>
<tr>
<td>BETDWH-667</td>
<td>2017-08-23</td>
<td>5</td>
</tr>
<tr>
<td>WWW-14951</td>
<td>2017-08-23</td>
<td>5</td>
</tr>
<tr>
<td>JIRA-196</td>
<td>2017-08-24</td>
<td>4</td>
</tr>
<tr>
<td>JIRA-197</td>
<td>2017-08-24</td>
<td>4</td>
</tr>
<tr>
<td>WWWAPP-253</td>
<td>2017-08-25</td>
<td>3</td>
</tr>
<tr>
<td>BETCLNT-13124</td>
<td>2017-08-28</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 66: P3 issues started in previous milestone

Only 8 issues were started before or at code freeze of the previous project. This means that during P2 everybody had enough work until the end of the milestone and only those
issues had to be started earlier.

**P4 in prior milestone**

On average issues were started 62 days before the code freeze of the prior milestone. An issue was started 392 days before and another 637 days after the code freeze. As already shown in figure 64 the main problem of P4 is that its milestone was updated very often. Which means that first issues of the project were implemented very early (over 1 year before). Other issues that were started late existed since the beginning of the project and also had their milestones multiple times updated.

**Comparison**

The numbers from above show that project organization became better over time. P4 issues were started over a year in advance, whereas P2 started 211 days early and P3 only 56 days. However there are still problems. Often issues are in a project that does not seem to be there. However they are all related and are mostly negatively influencing the numbers. This suggests that issues are assigned to projects so that they are implemented for a specific milestone but have nothing or not very much to do with it.

### 6.14 Tool Evaluation Questionnaire

The tool was evaluated with stakeholders that should use it. In this case the stakeholders are team- and group-leads that can influence the organizational structure of the development team and also assign work packages to developers. A questionnaire was used and passed to 2 team-leads and 2 group-leads. Group-leads are responsible for the developers in their group and assign work packages to them. Whereas team-leads are responsible for the group-leads and assign them projects and want frequent updates about project progress and workload.

The questionnaire is shown in appendix A. It is split into two parts:

- Demographics
- Tool Evaluation
6.14.1 Demographics

At the beginning of this section it was mentioned that the tool was shown to 2 team- and 2 group-leads. The job title for the team-leads is "Teamlead WebDev". One group-leads job title is "Grouplead Architecture & Performance" the others is "Grouplead App". The group-leads are in their position since 0.5 and 1 year, the team-leads 1 and 10 years. In the software development business the group-leads are for 12 and 5 years, the team-leads for 6 and 17 years. For one group-lead it is the first company working for. For the other group-lead and one of the team-leads it is their second company, for the other team-lead it is the third.

6.14.2 Tool Evaluation

For tool evaluation 3 issues were presented and the participants had to find flaws or anomalies either via the timelines or metrics. The timelines of the chosen issues are shown in the figure below:

![Figure 67: Issues for participants](image)

Since the questionnaire was held as an interview the answers for the first two questions were mixed. The following answers were given to the questions:

- Can you spot any problem that occurred during the issues lifetimes?
- Can you identify high coordination efforts?

The participants were instructed to locate problems either via the lifetime or a metric. The
issues BETDB-1475 and BAH-71 were chosen, because they show problems in organization and workflow. The issue WWW-7370 represents a normal issue without anomalies.

**BETDB-1475**

On general it was positively stated that the database team reacted very well on changes and immediately tackled them so that big delays could be prevented. However it was criticized that status changes appear too close to each other. This leads to the point that every participant mentioned: The workflow for handling database issues is badly designed. When something is bad in a SQL query nearly the whole workflow has to be executed again, except for additional approval steps. This was already mentioned by one of the participants, because it is not clearly defined what should be done in the approval process. Furthermore approval does not seem to work if issues are re-opened that often. Another assumption from the participants is that requirements were not written good enough. This assumption is based on the fact that preparation (at the beginning of the lifetime) lasted one month and this is not the usual case. This is a problem because other teams have to wait. Also the issue were often re-assigned within the database team and it was also once re-assigned to a developer. At last it was mentioned that there are a lot of comments for the issue, however they result from a tool that automatically writes comments when the SQL is generated.

**BAH-71**

For this issue the main problems were found in milestone and status handling and the lifetime of project issues. All participants stated that the lifetime is very long and it has to be asked if it was planned that long at the beginning. Furthermore they mentioned that the issue was moved a lot to other milestones without any work in between and this indicates that issue planning is not efficient. Also status for issues are not handled correctly. Some are superfluous, e.g. "QA Test" which indicates that the whole project is in testing. However sub-issues are already tested before and the status does not make sense at project level. This problem is supported by the event on March 31st, where status changed from "In Progress" to "Internal Review" to "External Review" and then to "Software redesign". A participant mentioned that it is not possible to review specifications of a whole project within a single day. Furthermore a project of this size should not stay 1 month in the design-stage, this is usually done faster. At the other end it seems that the whole project was tested in only 3 days which also cannot be the "real" case. It was
mentioned that the end of the timeline (beginning in May) represents the usual workflow.

**WWW-7370**

All participants mentioned that this is the usual workflow of a developer issue. This is the ideal case and it was probably a simple bug fix.

The problems for these specific issues were mainly found by timeline investigation. When the participants wanted to analyze the whole project they used the metrics. When a result of a metric was displayed they used timelines to analyzed the outliers (e.g. issues with min- or max-value).

For the first two questions it can be summarized that metrics are used for quickly analyzing a project and finding anomalies with each metric. If issues should be investigated in more detail the participants always used the timelines.

The next question asked the participants to name their 3 most valuable metrics (GL = group-lead, TL = team-lead).

<table>
<thead>
<tr>
<th>Metric</th>
<th>GL1</th>
<th>GL2</th>
<th>TL1</th>
<th>TL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>M3</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>M5</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M6</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M11</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>M12</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 16: Most valuable metrics for participants

As shown in the table for every participant metric 1 ("Issue resolved/closed date <= issue due date") is part of the most valuable metrics. For the participants the metric is good for comparing projects and can answer the following questions:

- How long is an issue at a specific department?
- How was the project planned?

Metric 3 ("Fix version wasn’t changed") was mentioned by 3 participants. It can indicate if there is too much work to do and can answer:

- Why issues were moved?
• How many issues are moved to backlog or come from backlog?

• Was something planned for an unrealistic time-frame?

Metric 5 ("Re-open rate of issues closed shortly before their due date") is good for finding out why issues were re-opened and if there is a pattern for the re-open changes.

Metric 6 ("Assignees of an issue") is good for finding out if requirements were not formulated in a good way or if it maybe was unclear who is responsible for something.

Metric 11 ("Issue re-assignments without status changes") shows problems with long-term issues and that in some cases workflows have to be changed.

Metric 12 ("Duration between re-assignment and first status change") helps a participant to identify faulty poorly planned projects.

The next part of the questionnaire was to rate the tool on a scale of 1 (not good) to 5 (very good). The following points (based on [35]) were to rate:

• Responsiveness (performance good with a lot of data)

• Forgiveness (allows exploration)

• Intuitivity (easy to navigate)

• Icons & Symbols understandable

• Information overload (enough or too much information shown)

• Low learning curve (easy to understand)

• Match to the real world (words...)

• Flexibility/Efficiency of use (for expert and novice user)

The average rating for the tool is shown in the table below (raw data of the rating is shown in appendix 8):
<table>
<thead>
<tr>
<th>Measure</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsiveness</td>
<td>4.25</td>
</tr>
<tr>
<td>Forgiveness</td>
<td>4</td>
</tr>
<tr>
<td>Intuitivity</td>
<td>4</td>
</tr>
<tr>
<td>Icons &amp; Symbols understandable</td>
<td>4</td>
</tr>
<tr>
<td>Information overload</td>
<td>4.5</td>
</tr>
<tr>
<td>Low learning curve</td>
<td>4</td>
</tr>
<tr>
<td>Match to the real world</td>
<td>3.25</td>
</tr>
<tr>
<td>Flexibility/Efficiency of use</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 17: Tool rating by participants

Overall the tool was rated as good by the participants. After rating the tool they were asked for positive and negative criticism:

<table>
<thead>
<tr>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>• trends</td>
<td>• misleading naming of M1</td>
</tr>
<tr>
<td>• lessons learned</td>
<td>• process knowledge needed</td>
</tr>
<tr>
<td>• what’s (not-)efficient</td>
<td>• JIRA knowledge needed for interpretation</td>
</tr>
<tr>
<td>• combined metrics and timelines</td>
<td>• M2 also counts issues without due-dates</td>
</tr>
<tr>
<td>• timeline to find patterns</td>
<td>• descriptions for metrics missing</td>
</tr>
<tr>
<td>• find flaws in short time</td>
<td>• show title of chosen metric more prominent</td>
</tr>
<tr>
<td>• comparing projects is easy</td>
<td>• better to choose date and afterwards metric</td>
</tr>
<tr>
<td>• simple design</td>
<td>• icon legend should be beside timelines</td>
</tr>
<tr>
<td>• find flaws in workflow</td>
<td>• list of tickets needs headline</td>
</tr>
<tr>
<td>• good to see wrong processes</td>
<td>• tooltips hard to read</td>
</tr>
<tr>
<td>• visualization</td>
<td></td>
</tr>
<tr>
<td>• good for improvements</td>
<td></td>
</tr>
<tr>
<td>• easy overview</td>
<td></td>
</tr>
</tbody>
</table>

Table 18: Positive and negative criticism

Additionally the participants provided the following suggestions for future versions of the tool:

- points on timeline should be bigger if more events occurred at the same date
• possibility to set target values for metrics (like for an alert)
• show the department that made a status change
• mark issues that were reviewed
• possibility to filter for issue types (e.g. only bug issues)
• always display parent issue if one exists for the selected issue
• show title of issues, not only keys
• display how long a ticket is in a specific state
• add tooltips for metrics with description of them
• export metadata from JIRA (e.g. milestone dates)
• extended filters for metric tables
• option to hide metric table if it is not needed any longer
• show points on timeline that are important for metric

6.15 Discussion

The results of the evaluation mostly indicate that the metrics fit to their need. However some drawbacks were found.

The suitability and completeness of the metrics has two drawbacks:

• Workflow mis-usage by employees

• Metric is insufficient

The first problem is that employees often use the workflow in an un-intended way. One example is that a developer should set an issue from its ”Open” state into ”In Progress” as soon as development starts. In many cases an issue was set to ”In Progress” shortly before it was ”Resolved”, which would indicate that this was a small issue. By analyzing those issues in JIRA it was seen that this behavior also occurred on issues that had a longer development time of some days. Such mis-usage influences the tool and makes part of the data unreliable. The second problem were metrics that did not represent the quality of
the organizational structure or issue handling process. This is e.g. \textit{M7 - Commentators of an issue}. Since employees are using other ways to communicate than the JIRA comment section it can not be reflected if there are chatty conversations between multiple team members. Furthermore it would have to be analyzed what the conversations in other channels (like Skype) are about (work or non-work related) to state if there is something wrong in the organization. This is furthermore the case for \textit{M9 - Issue re-assignment without any communication}.

As the result of the questionnaire shows there are some metrics that are very useful for a company. The three most valuable metrics were:

- M1 - Issue resolved/closed date \(\geq\) issue due date
- M3 - Fix version wasn’t changed
- M6 - Assignees of an issue

Also their results showed important and interesting facts. Based on the information of these three metrics group- and team-leads can intuitively find possible problems in a workflow and can communicate them to developers.

In this case the tool was able to compare both organizational structures. However the analysis was based on projects and a real comparison has to be made over a longer timeframe. The analyzed projects showed that the new structure is beneficial for project progress and suggests to switch the whole development team to the new structure.

The metrics showed that performance and coordination in the old organizational structure were below the new structure. In some cases this was caused by false usage of the JIRA workflow but also by too much re-assignments between teams. The amount of re-assignments negatively influenced the progress on an issue. The new team structure prohibited this behavior and every team member was able to focus on a specific task and complete their work.

At the end it has to be stated that the tool is more beneficial for employees in a management or lead role than for a developer. Development leads can analyze how often issues are re-assigned and can find problems in the issue handling process. Whereas project managers can easily see where problems occurred and work lasted longer, so that it can be prevented in future. A developer can analyze the performance of the whole team and can compare themself against the metrics.
7 Conclusion & Future Work

For the thesis the goal-question-metric approach was used. Therefore 4 questions with 13 metrics were created. The metrics answered the questions mainly from the perspective of team leads but also of developers. The goal was to evaluate the current team structure and issue ticket usage on efficiency and coordination effort.

The metrics are calculated by a self-implemented tool. The tool has two parts, where the first is needed for extracting issues from JIRA and putting them as JSON into a CouchDB (JIRAExtractor) and an analyzer which is a simple HTML page with JavaScript that connects to the CouchDB. The analyzer calculates the metrics and displays the results in tables, but it also displays events (e.g. issue re-assignment) on a timeline that represents an issues lifetime.

The evaluation concerning the metrics was based on 4 different projects. 3 of them were developed in the App-team, where also a structural change from the first to third project happened. The fourth project was taken from the Web-team and was used to compare the results of the other projects. The results show that in most cases the third project achieved the best values and demonstrates that re-structuring the teams had a positive effect on issue and project development. In some cases it did not perform as good as the others, but this does not outweigh the more positive benefits of the change. The comparison to the Web-project also demonstrated that the first two App-projects performed similar to it. This results in the assumption that a structural change in the Web-team would also bring those benefits to future projects. To evaluate the tool a questionnaire was created. 2 team-leads and 2 group-leads participated and answered the questions. The result was that the tool is useful and necessary information can be easily found. It was also positively mentioned that calculation of the metrics happens fast. However useful ideas for future improvements were mentioned, these were already shown in section 6.14.2.

Additionally to these points are several aspects that have to be improved in the future. It should be possible to export the data in a statistics and data-analysis tool like SPSS from IBM. There more sophisticated analytics can be made and should not be implemented in the analyzer itself, because it should only give a rough overview. It should also be possible to save result tables as a CSV document. That the tool can be used by a broader audience the extractor has to be extended to support other issue management products beside JIRA. Furthermore more code repositories to retrieve submitted files have to be
accessed (e.g. SVN or Git). Additionally to that change a default issue structure in JSON has to be specified, so that the representation of an issue stays the same wherever it comes from. This positively influences the analyzer since it has not to be changed when a new product is added to the extractor. Despite these points the major point of future work is to analyze every upcoming project. Only when multiple projects are analyzed the data can be compared. This would also enable an automated baseline calculation for specific metrics and if an analyzed project is much higher than the baseline the analyzer can report it.
A Questionnaire

Demographics

Job title: ____________________________

Years on current position: ____________________________

Years in software dev business: ____________________________

Nr of companies worked for: ____________________________

Tool Evaluation

Please take a look at the following issues. From the timelines displayed, can you spot any problem that occurred during the issues lifetimes? If so, briefly describe them.

... can you identify high coordination efforts? Please locate them with a metric or on the timeline.

Which 3 metrics are, in your opinion, the most valuable:

1.

2.

3.
What do you think about the tool? (1 star - lowest rating, 5 star - highest rating)

Responsiveness ★★★★★
Forgiveness (allows exploration) ★★★★★
Intuitivity ★★★★★
Icons & Symbols understandable ★★★★★
Information overload ★★★★★
Low learning curve ★★★★★
Match to the real world (words…) ★★★★★
Flexibility/Efficiency of use ★★★★★

Further criticism:

<table>
<thead>
<tr>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What features would you like to see, what metrics to measure, or which situations to capture in a future version of this tool?
### B Questionnaire tool rating - raw data

<table>
<thead>
<tr>
<th>Measure</th>
<th>U1</th>
<th>U2</th>
<th>U3</th>
<th>U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsiveness</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Forgiveness</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Intuitivity</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Icons &amp; Symbols understandable</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Information overload</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Low learning curve</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Match to the real world</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Flexibility/Efficiency of use</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 19: Tool rating
References


