

Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

JAN-PHILIPP STAUFFERT
MASTER THESIS

A POOR MAN'S APPROACH TO TECHNICAL DEBT

Submitted on 2 August 2015

Supervisor: Prof. Dr. Dirk Riehle, M.B.A.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander-Universität Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 2 August 2015

License

This work is licensed under the Creative Commons Attribution 4.0 International license (CC BY 4.0), see <https://creativecommons.org/licenses/by/4.0/>

Erlangen, 2 August 2015

Abstract

Technical debt is a metaphor to describe the trade of quality for short term goals. It is used to discuss the effects of short term goals on quality and productivity thereafter.

While most research on technical debt is based on code metrics or qualitative investigations, this thesis explores the possibility to derive a model to compute technical debt based on project management data.

For this, metrics are calculated from the data. They are compared to a known technical debt process to assess their suitability for prediction. Hereby, the quality of the data plays an important role for the usability of the results.

While the use of management data promises more insights in technical debt, it suffers from the many factors that can influence the data.

1 Introduction

1.1 Original thesis goals

Technical debt is a concept that has attracted increased attention recently. The donation of project management data from the SoftwareAG opened an opportunity to use this kind of data to derive a metric for technical debt.

The original goal was to use the data to construct a mathematical model on top of it to derive a measure for technical debt. This could then be used to steer software projects.

The assumption was that management decisions lead to a change in technical debt. It also has an impact on measurable values like the defect rate. To model technical debt, a bijective function that maps those values to technical debt was sought.

1.2 Changes to thesis goals

The data had deficiencies and lacked necessary information to build a model. The goal then shifted to describe what can be expressed with the data and what would be needed to create a proper model.

Instead of a mapping between the data and technical debt, some indicators for technical debt are assessed. The combined indicators are expected to give hints to the progress of technical debt.

2 Research

2.1 Introduction

Software metrics are a widely used tool to estimate software quality and development progress. They serve for example as an early indicator if a project needs more man power or time for a successful completion of a project and are thus used as a base for management decisions. (Stark, Durst & Vowell, 1994).

Technical debt is a metric that is not based directly on source code analysis but builds on top of other metrics. It describes the process of acquiring short term productivity at the cost of quality. Deteriorated quality then endangers development speed in the long term. The accrued technical debt can then be “paid back” by investing effort onto quality improvement to ameliorate the negative long term impacts.

With this metaphor, technical debt tries to recast a technical concept as an economic one (Brown et al., 2010) to facilitate the value of quality to stakeholders (Nord, Ozkaya, Kruchten & Gonzalez-Rojas, 2012).

Existing work on technical debt is mostly concerned with source code, but has expanded to cover different aspects and artefacts of the software development. This leads to definitions of, among others, architectural debt, requirement debt, or documentation debt (Sterling, 2010; Kruchten, Nord & Ozkaya, 2012).

The data that is taken as a base, thus, ranges from source code analysis (Mayr, Plosch & Korner, 2014; Nugroho, Visser & Kuipers, 2011) to qualitative research (Klinger, Tarr, Wagstrom & Williams, 2011).

This thesis tries to evaluate, whether it is possible to use more abstract data, i.e. management data from a bug tracker that was used to log effort spent on work items, as a base for a technical debt model.

This debt model can help to get a better understanding of technical debt in a software development process by giving an additional method to calculate technical debt. On the other hand, technical debt derived this way includes many

of the other technical debt deductions because all the different sources that are taken in isolation for other approaches contribute to the management data looked upon here.

2.2 Related work

Fenton and Neil (Fenton & Neil, 1999) discuss, that software metrics are often introduced to obtain a quality certificate and less because of their helpfulness. They argue, that they say less about the quality of the code but instead hint to the amount of testing done. The metrics employed are often basic metrics like lines of code (LOC) or cyclomatic complexity (McCabe, 1976), while more complex metrics are often neglected.

The Technical Debt metaphor was introduced by Ward Cunningham (Cunningham, 1993) and has attracted more and more attention (Kruchten, Nord, Ozkaya & Falessi, 2013).

According to Klinger et al. (Klinger et al., 2011), technical debt has the problem that different people are responsible for incurring technical debt than for repaying this debt. The technical debt metaphor therefore helps to raise awareness and by quantifying it, provides a tool to talk about quality.

Technical debt differentiates between unintentionally and intentionally incurred debt. While the former cannot be avoided completely and only alleviated through better educated developers and more care while developing, the latter is a conscious decision (McConnell, 2008).

The resulting technical debt is composed from remediation costs and non-remediation costs, which are the costs to remediate the situation and the costs that are accrued if the debt is not remediated respectively (Mayr et al., 2014).

The question which unit is suitable is still under discussion. On the one hand, this is a problem with the abstractness of the metaphor of technical debt that has a lot of influencing factors and is therefore hard to derive a number to measure it. On the other hand, the question arises, which units are best perceived and help the evaluation of management decisions the most. There are approaches with a star system (Nugroho et al., 2011), a traffic light system with green, yellow and red (Eisenberg, 2012) or a monetary expression of technical debt (Mayr et al., 2014).

2.3 Hypotheses

To derive a model for technical debt from project management data, features are sought that prove usable. For this, hypotheses are posed that are related to technical debt. Features of the data are selected that are tested with those hypotheses to assess if they meet the expectations and can therefore be used as indicators for technical debt.

1. Increased technical debt leads to an increased average effort estimation
2. Increased technical debt leads to a more unreliable estimation
3. Increased technical debt leads to more average effort per defect
4. Increased technical debt leads to more defects
5. More work on new functionality in a part of the release leads to more work on defects in the same part
6. More work on new functionality leads to more defects
7. The metrics calculated for the hypotheses above can be used to find a polynomial function that maps them to a value for technical debt.

2.4 Data

The data analysed here was kindly donated from the SoftwareAG from their internal bugtracker. They employ agile development (Kent Beck et al., 2001) and use the bugtracker to also manage their development progress. They keep track of the features and user stories there. Additionally, they provided information about their release cycle and general information about three releases.

The data spans a time of 30 months. The bugtracker data ranges further back in history. There is a small usage before the time that is relevant for the research. For there is not further information available for this time prior to the relevant time period, all observations falling into this time are dropped, i.e. excluded from the further analysis.

2.4.1 Releases

For the 30 months mentioned above, there were three releases of their product. The first release took three months, the second 18 months and the third again three months.

The second release, which took three times the time of the other releases, was split in three phases of six months each. The beginning was conducted in the same way as the other releases. Then, a “hot phase” occurred where the focus was shifted towards the implementation of new functionality. It was doubted that otherwise the goals for this release wouldn’t have been met. In this phase, the fixing of defects was reduced to provide more workforce for this new functionality. In the final phase of this release, the remaining functionality was implemented and the staff went back to a “normal” work state.

This “hot phase” is used to try to derive insights about technical debt, since it is assumed that technical debt in some form was accrued in this phase and was paid back in the following phase. This assumption is deduced from the reported quality which was constant in the first and third release and the first phase of the second release. Each release finished with the same quality it had started. This quality statement is grounded in the agile approach which uses continuous testing for quality assessment.

When talking about the different time periods, the term “phase” is used, where the first phase is the first release, followed by three phases of the second release and the final phase being the third release. When comparing high and low technical debt phases, the third and fourth phase (B2, B3) are assumed to be of high technical debt and the rest of low technical debt. For a figure to visualize this technical debt assumption see figure 2.1. Low quality is connected to high technical debt.

The amount of developers working on the releases is not known but was constant for each release.

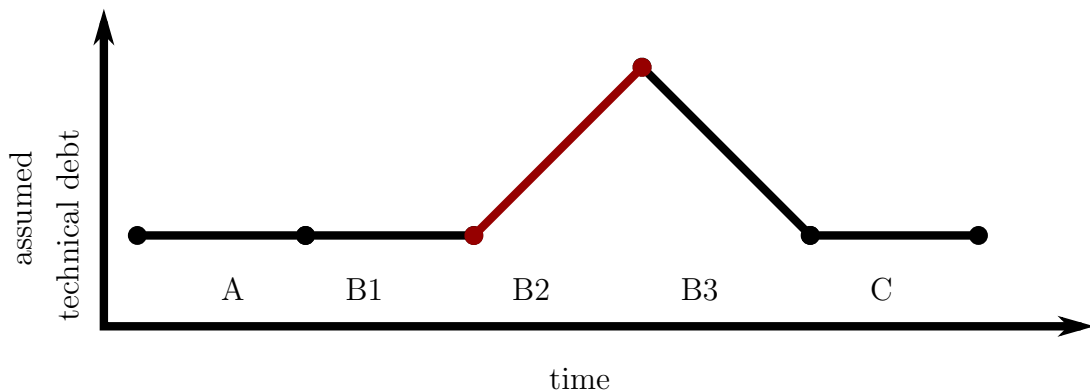


Figure 2.1: The assumed technical debt in the data. In the “hot phase” technical debt is accrued and afterwards paid back.

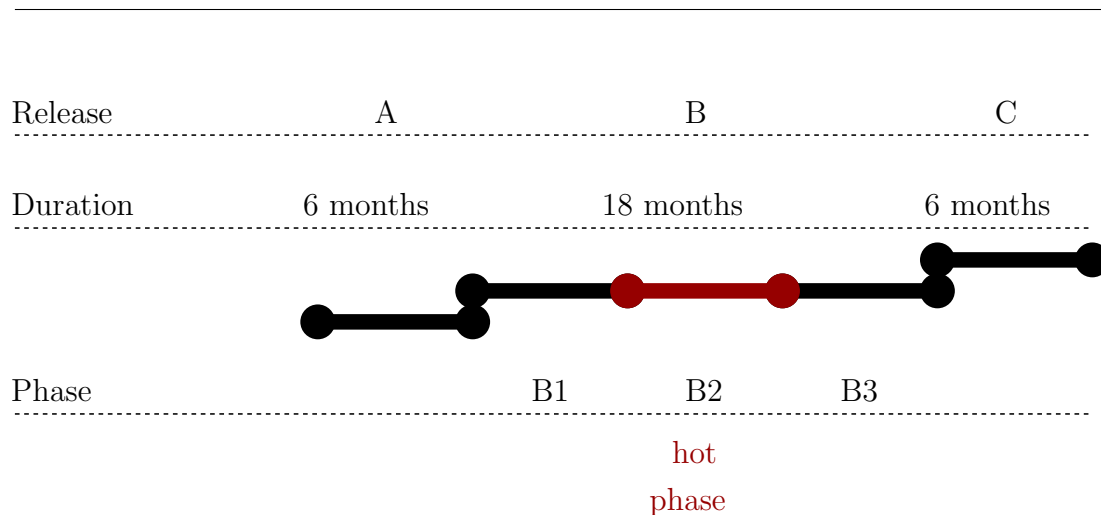


Figure 2.2: Overview over the time that is covered by the data with the abbreviations that will be used to refer to different phases.

2.4.2 Bugtracker data

The data originates from a bugtracker. There are two tables available named “issues” and “worklog”. A row in one of these tables is referred to as an “issue” or “issue item” or “worklog item” respectively.

There are 135455 issues. These are accompanied by 233890 worklog items. Both types will be described in more detail in the following sections.

issues

The following data is associated with one issue.

Type

There are three different types: “Feature”, “User Story” and “Defect”. “User Stories” are “Features” broken into work packets. Only a minority of projects have “Feature” issues, which speaks against the general usage of features, so they will not be used for analysis. “Defects” mark items that stem from the testing section that creates them to be fixed. Errors or bugs that occur during the implementation of a feature or user story from a developer’s mistake usually don’t find their way to the bug tracker but are counted as time for this work unit.

Dates

There are dates for the creation, the beginning and the end of the implementation, when it was fixed, when tested and when completed. The issue’s creation date is always available.

Effort Estimation

The team estimated the effort needed for an item with story points. These points are not normed in the way that a certain value stands for a certain effort but they should indicate if an item has a low, moderate or high expected effort. Only user stories' impact was estimated with 28% of user stories having story points. Defect and Feature effort was rarely estimated (Defects 0.2%, Features 0.02%).

Effort

The effort of an issue, logged in seconds. Outliers, where more than a year is logged for one issue, are removed.

Project/Project Category

Each release consists of multiple projects that are associated with project categories. Furthermore, projects and project categories will not be used for the analysis as there is only additional information available for the releases as a whole.

There are additional columns such as feature key that are for internal use and are not pursued further here.

A selection of more detailed numbers about the analysed data can be found in section 3.1.

worklog

Worklog items describe effort done for an issue further. They always point to the issue they are referring to via a foreign key and carry a date and the effort done.

The relationship to the issue table is a one to many relation, meaning that for each issue, there are zero or more worklog items.

2.4.3 Deficiencies

The following paper describes the analysis of the data as if the data fulfills the description above. However, the data has some deficiencies, that should be mentioned here.

From the relationship of issues to worklog items it can be reconstructed that there are issues missing. While it is possible to have issues without worklog items, the opposite is not true but is encountered in the data. Following the foreign key from the worklog table, there are 91384 unresolved links. With 135455 available issues, the known missing issues are $\approx 40\%$ of the total amount of issues. Those are only

the missing items that can be estimated, while the real number is presumably higher.

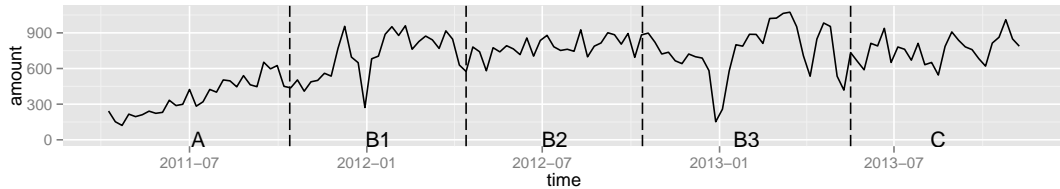


Figure 2.3: Number of issue items missing from the data that are referred to from worklog items.

Figure 2.3 analyses the distribution of known missing data over time, finding that it spans the whole analysed time.

Further, the analysed releases are not equal, which detracts the validity of comparing them. Additionally, they all had special circumstances that influence their issues. The bug tracker was introduced in release A, with the developers adopting to the new tool. Release B was longer than usual which has influenced the way, the work was spread over the time. Release C suffers from cut off effects.

The high amount of missing data threatens the representativeness of the data and will effect the outcoming results of this thesis. Nevertheless the approaches show a universal character.

More properties of the data are described in section 3.2.

2.5 Method

2.5.1 Effort

The main information that the data revolves around is the logged effort. The problem however is, that there is only one number indicating the effort for one issue that has existed for several days or weeks. It is not known, how this effort is distributed over an issue's lifetime. The worklog entries can be used to help clarify this question but it still leaves a room for interpretation.

Most issues that are available don't have a worklog item associated with them, followed by issues with only one.

Therefore, different assumptions, when the effort took place are evaluated.

Before exploring those approaches, it is important to elaborate on the meaning of the effort reconstruction. With the reported constant developer count per release,

it can be assumed that the logged effort stays constant. This is because every developer will log the same amount of hours every day. Only the distribution of effort onto issues changes and the percentage of how much of the total effort is spend on defects or user stories.

This, however, is not the case. Independent of the approach used, there are times with more and times with less effort. The approaches differ in estimating, when an issue's effort took place. If the effort would have been constant, all approaches would yield the same resulting effort at each time. They, however, each shows peaks and valleys.

There is a fixed amount of developers available but only a part of them is working on the issues, that are analysed, and the rest is working on the issues that are not known. In times where more effort is spend, the projects, that are analysed, demand more attention by the developers. There is no possibility to say if those hours spent were more or less productive at a certain time.

The logged effort is therefore used to see the effect of more work in the known projects on the assumed technical debt.

Effort reconstruction

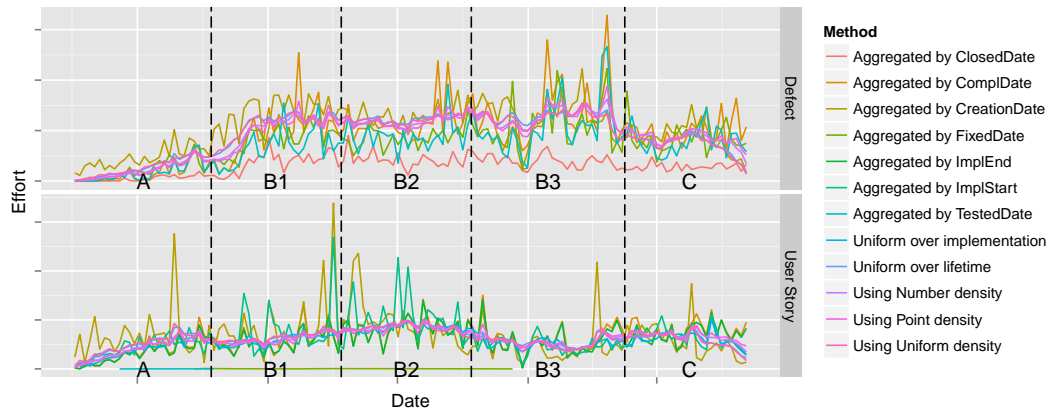


Figure 2.4: All effort reconstruction approaches plotted. Approaches that aggregate by one of the dates fluctuate more but have the same tendencies.

There are different possibilities, how to estimate the effort that was needed at a certain point in time. To smooth daily fluctuations, the following aggregations are conducted over time windows of a week each. Figure 2.4 shows the results of the different reconstruction approaches. Methods that are far below the rest of the results suffer from less items that can be used.

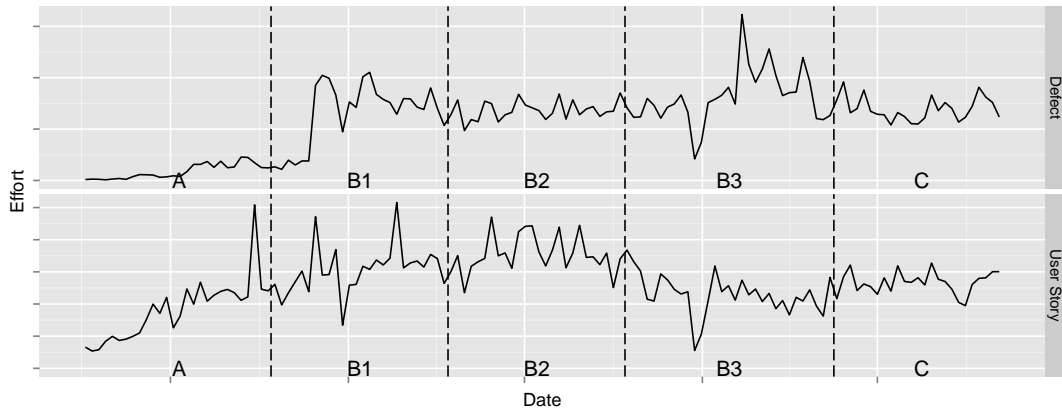


Figure 2.5: Effort logged in the worklog aggregated by the date it was logged.

The simplest and most obvious approach is to take the effort and assume, it was done at one specific day. For worklog items it is the day, they are logged. For issues, there are different dates, though with the exception of the creation date, the dates are not always available.

In the graph, those results are the lines whose description start with “Aggregated by”. They fluctuate more than the approaches presented below.

Figure 2.5 shows the aggregation of the worklog data by its logged date. This differs from the approaches with issues because they also include possible partial data about the missing issues.

Although the worklog is expected to be more detailed in relation to when exactly the effort was spent and less influenced by the mere creation of an issue, the same trends are emerging., which are an increase in user story effort in the first release, an increase in phase B2 and a decrease in effort in phase B3. Many issues are created in the beginning of phase B2, which influences the issue effort graphs (see figure 2.4). The worklog graph shows no such extreme spike at the beginning of phase B2 but a constantly high effort consumption throughout phase B2.

To improve the effort representation, the duration of an issue is taken into consideration. A simple approach is to assume, the effort was distributed uniformly over an issue’s lifetime. Figure 2.4 shows this for the uniform effort distribution over each issue’s lifetime and implementation time.

It is unlikely that each issue was worked on with a constant effort. Assuming issues are worked off in a similar way, the worklog items are used to derive a density of when in an issue’s lifetime how much of its effort was done. For this, the time is normalised with 0 being an issue’s creation date and 1 its completion date. The densities are computed over the time from zero to three due to some worklog items taking place after its issue was already completed.

The three different proposed densities are depicted in figure 2.6.

“Number” describes the density that is obtained when counting when a worklog item was logged in its issues lifetime and aggregating over this time. The amount of worklog items is roughly constant over the issue lifetime with an increase in the end. This is because of the fact, that the issues with a worklog item have at least one item that is logged when the issue is closed.

This effect is even bigger in the “Point” density, which takes the percentual amount of effort a worklog item has from its issue and aggregate over the worklog items time in its issue’s lifetime.

If it is assumed that the logged effort of each worklog item was done in a uniform way since the last time a worklog item was logged for the same issue or since the issue’s creation time, the “Uniform” density results.

The effort that results if the respective density is taken to describe the effort distribution of each issue over its lifetime is again shown in figure 2.4.

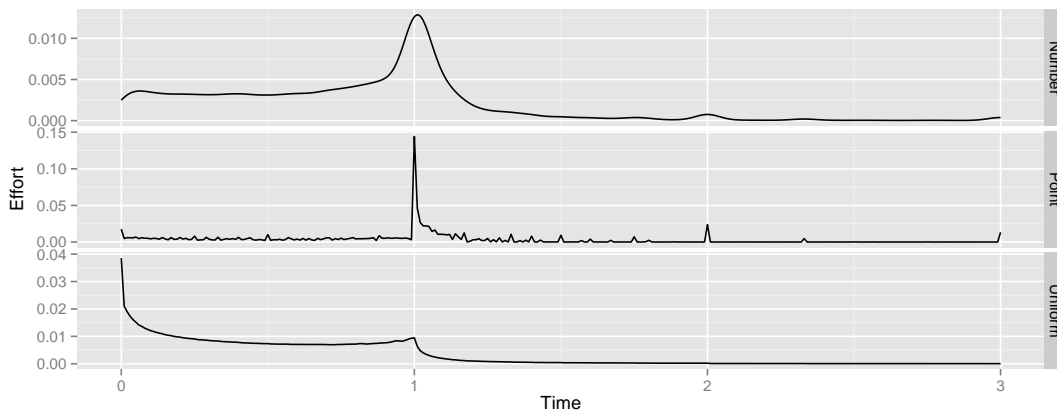


Figure 2.6: Densities used for the effort reconstruction

For the effort created with the “uniform” density, the mean and variance over different time windows is shown in figure 2.7.

2.5.2 Effort estimation

Story points are used to estimate the effort of an item before it is started. This is done to understand the impact of user stories on the workload ahead of time. More story points should indicate more effort spend on the respective issues.

Figure 2.8 shows the mean and standard deviation of story points over different sized time windows. The mean and standard deviation are computed for every month or every phase.

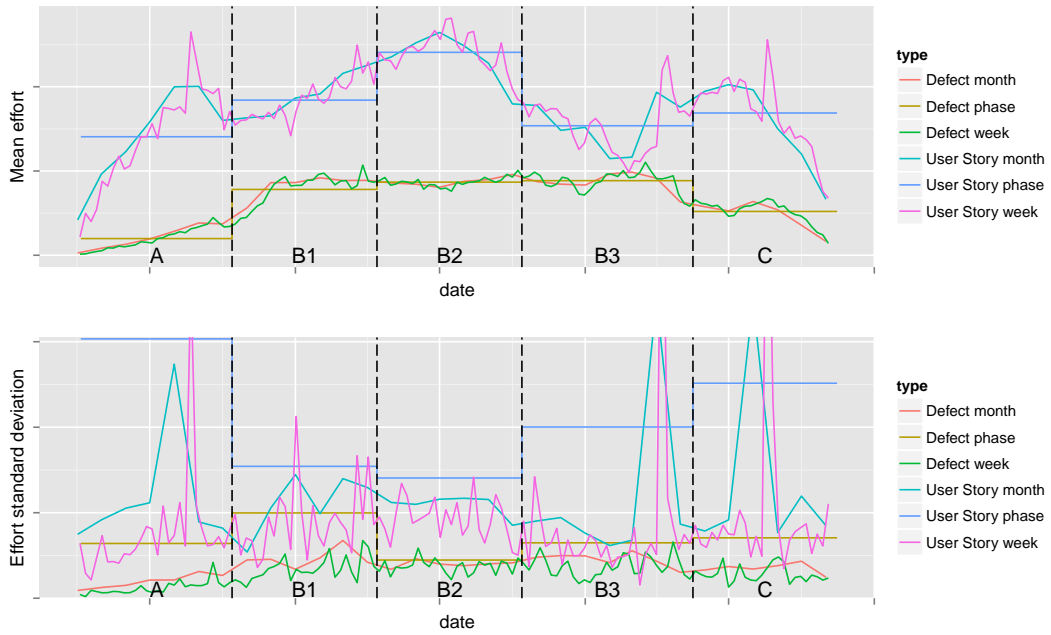


Figure 2.7: The mean and standard deviation of the effort described by the “uniform” density aggregated for different time windows

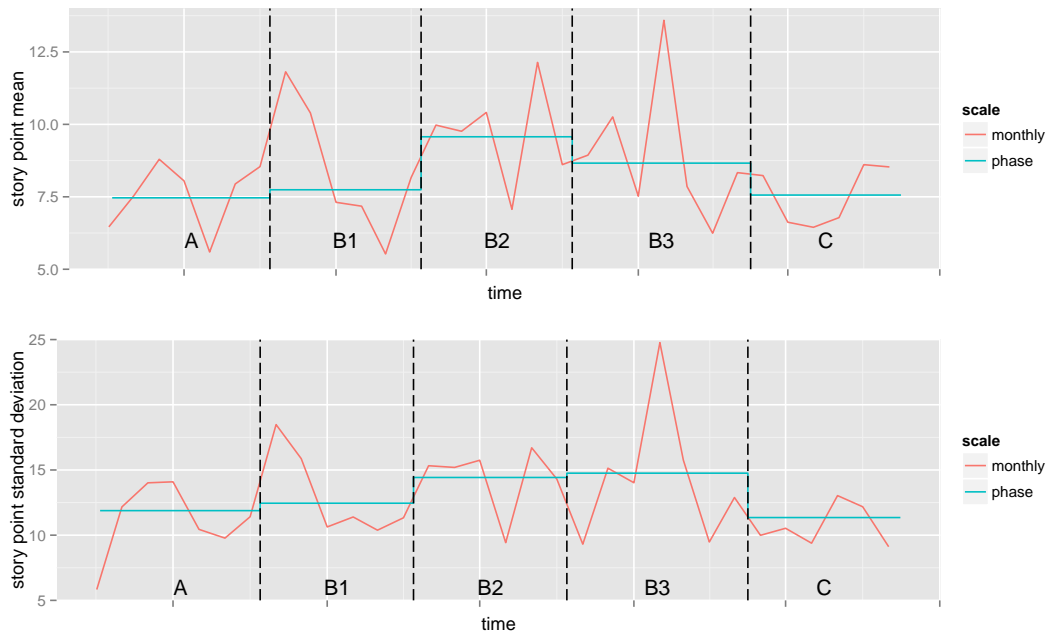


Figure 2.8: Mean and standard deviation of story point values over different sized time windows

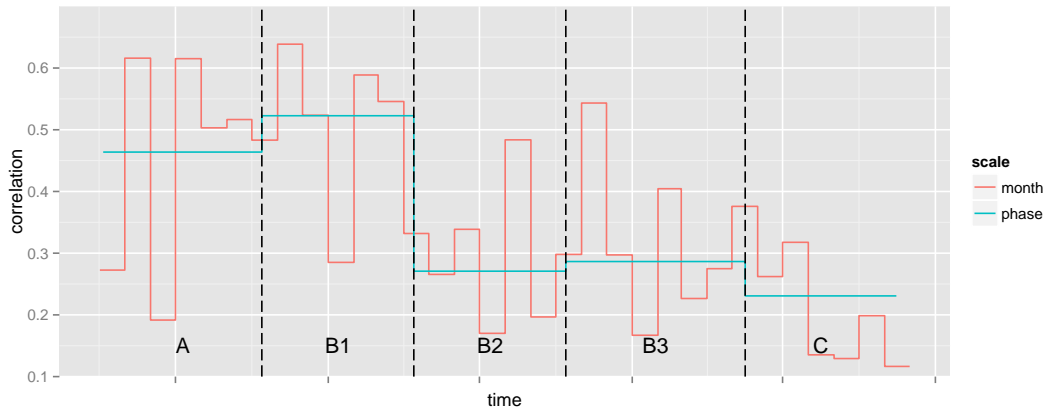


Figure 2.9: The correlation between the effort estimation (story points) and the actual effort for differently sized time windows. The values are not strongly correlated and are less correlated with advanced time.

Figure 2.9 shows the development of the correlation between effort estimation with story points and the actual logged effort over time. It is computed over different time windows, as well.

2.5.3 Defects

Technical debt is supposed to influence defects both in frequency as in time to fix.

Figure 2.10 shows how many defects are created over the time aggregated by week and phase. Interesting is the drop in the beginning of phase B3.

Figure 2.11 shows the mean effort per defect over time aggregated over different time windows. While it is low in the beginning, it is much higher in the beginning of release B and then decreases.

2.6 Results

This section uses the described metrics to run statistical tests in order to prove the posed hypotheses. Figure 2.12 plots the compared metrics against each other to visualise the findings.

Hypothesis 1 *Increased technical debt leads to an increased average effort estimation*

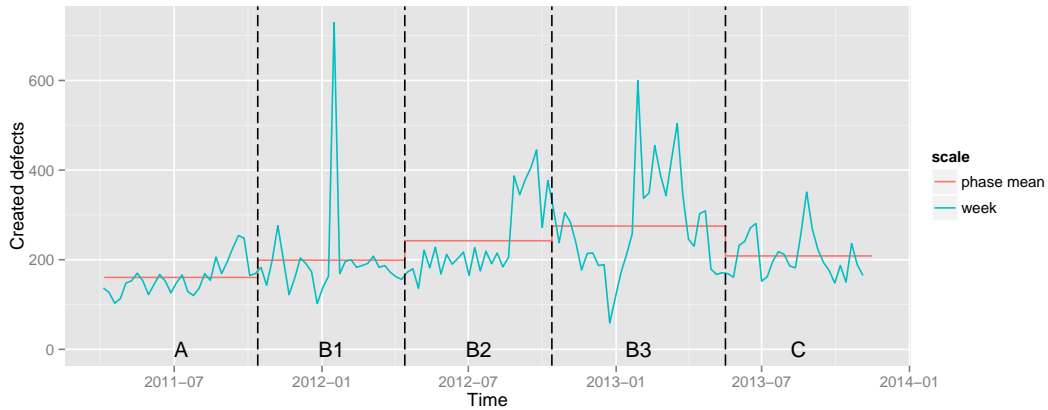


Figure 2.10: Defects created in the system over time.

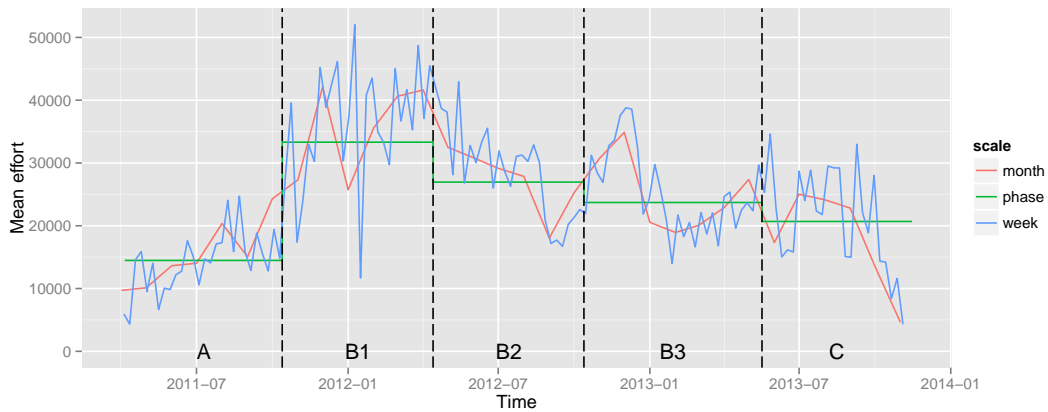


Figure 2.11: Average defect effort over time aggregated by the issue’s creation time.

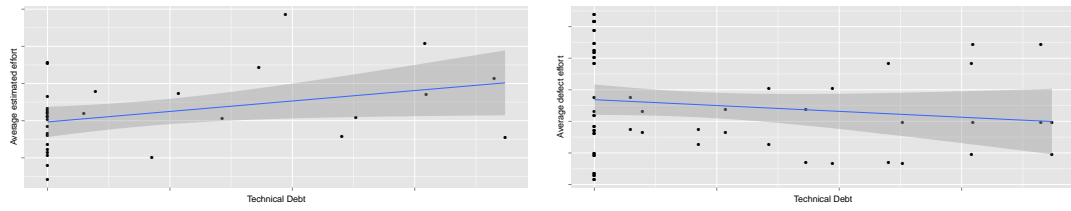
The correlation test between the technical debt as depicted in figure 2.1 with the average monthly effort estimation as shown in figure 2.8 yields a p-value of 0.023 with a correlation of 0.36. There is a significant correlation with a 95% confidence interval.

While this is a significant finding, the use of the average estimation for a model for technical debt is severely limited by its volatility.

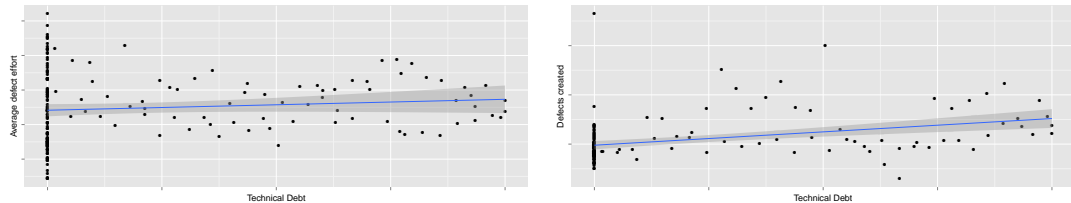
Hypothesis 2 *Increased technical debt leads to a more unreliable estimation*

The overall correlation between story points and logged effort is moderate with 0.359. The correlation in the “hot phase” is slightly smaller with 0.27 than in the other phases that have a correlation of 0.433.

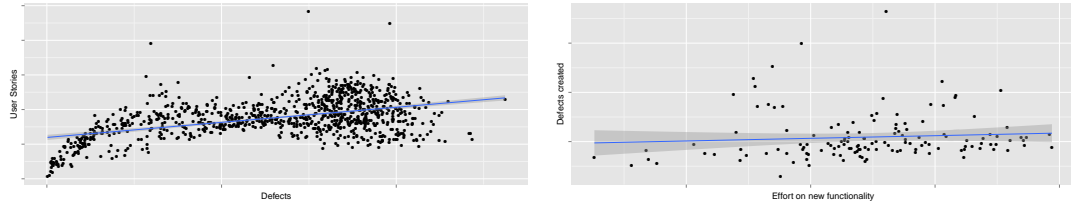
To test the hypothesis, the correlation between the technical debt as in figure 2.1



(a) Correlation between technical debt and the monthly average effort estimation (b) Technical debt correlation with the correlation of logged and estimated effort.



(c) Correlation between technical debt and the monthly average effort per defect (d) Correlation of technical debt with the amount of newly created defects



(e) Correlation between the work on new functionality and work on defects (f) Correlation between the work on new functionality and newly created defects

Figure 2.12: Visualisation of the values compared to each other to test the hypotheses

is compared to the correlation between story points and logged effort aggregated for each month as in figure 2.9. With a correlation of -0.14, the correlation test cannot confirm the hypothesis with a p-value of 0.132.

Hypothesis 3 *Increased technical debt leads to more average effort per defect*

Figure 2.11 shows a decrease in average effort in the phases B2 and B3 in comparison to phase B1, which already is a strong objection against this hypothesis.

Additionally, the correlation test of the technical debt as depicted in figure 2.1 with the average defect effort as depicted in figure 2.11 cannot confirm the hypothesis with a p-value of 0.0918 and a correlation of 0.1.

Hypothesis 4 *Increased technical debt leads to more defects*

Performing a correlation test with the per week aggregated numbers and the

assumed technical debt depicted in figure 2.1 supports the hypothesis with a p-value of 2.133772e-06, where the correlation is 0.37.

While this supports the hypothesis, there are observations that speak against this finding. There is a small trend of increased defect amount in the end of release A as is in the end of release B. This effect can originate from the project lifecycle, where more defects are detected and worked on shortly before the release. There is a spike of created defects in phase B1 where constant, little defects due to the low technical debt are assumed. Additionally, there is a trough in the graph in the transition between phases B2 and B3, where the highest value is assumed.

Hypothesis 5 *More work on new functionality in a part of the release leads to more work on defects in the same part*

The correlation test between the effort logged for user stories, i.e. new functionality, and defects as shown in the figure 2.4 using the uniform density returns a p-value close to zero with a correlation of 0.47.

This means, that if the developers spend more time for a certain project, they also spend more time to work on defects. While this is important to know for model building, it is not directly helpful for a technical debt analysis.

Hypothesis 6 *More work on new functionality leads to more defects*

For this hypothesis, the effort for user stories depicted in figure 2.4 using the uniform density, aggregated for each week, is compared to the amount of created defects per week as depicted in figure 2.10. The correlation test does not reject the null hypothesis with a p-value of 0.171 based on a correlation of 0.082.

Hypothesis 7 *The metrics calculated for the hypotheses above can be used to find a polynomial function that maps them to a value for technical debt.*

To approximate a polynomial function, a neural net is used. The metrics elaborated on above are used as input with the technical debt as in figure 2.1 is used as the output. 3/4 of the data is used for training with the rest being used for testing. 10 iterations with different partitioning are conducted. The mean absolute error is 0.259303. With the technical debt being normalised to an interval from 0 to 1, this mean prediction error of approx. 25% is too high to conclude a polynomial relationship.

2.7 Limitations

From the existence of outliers it can be deduced that there is human error involved. The analysis described suffers if the bookkeeping is not done properly.

In the section about the data source, it was already mentioned that the data analysed here has deficiencies. Those can have severely influenced the results.

The bugtracker captures the development at a high abstraction level. This means, that there are many factors that can influence the data such as code quality, managerial decisions, attitudes towards the bug tracker or experience of the developers.

These many influencing factors impede the creation of a model because many of them are not capturable. Especially management decisions can disrupt a model in many different ways that are hard to track.

For the data is very abstract, it is hardly possible to isolate factors that disrupt a possible model.

2.8 Discussion

The biggest problem when analysing the data is that there is only a very basic understanding of the progress of technical debt available as well as little information about the development process in general.

The effort done at each point in time is hard to estimate from the issue data. Depending on the way it is derived, it provides different insights. Without more additional knowledge of the development process, there is no possibility to validate one of the approaches. Since the bugtracker leaves space for interpretation, the usage, that is documented in our data set, can deviate from the usage in other companies.

Companies try to optimise their processes and therefore the process shown here can have changed. Each release was special in some way, the first was with the introduction of the bugtracker as a tool, the second was longer and with a special phase in between and the last release suffers from cut off effects. It was not possible to determine a baseline that is common among releases due to the lack of more comparable releases.

There is also no possibility to relate user stories and defects in a way that it is evident that a particular user story is the source for some defects. Without this knowledge, the defect information is a detached work stream, whose relation to the user story work can only be estimated. A possible model, that would build

upon the relationship between quality, user stories and defects is described in section 3.3.

2.9 Conclusion

Some features changed in times of high technical debt. Those features were not conclusive enough to derive some sort of measure for technical debt.

The very basic understanding of the technical debt that was present limited the analysis. It was reported that in the releases A and C as well as in phase B1, there was a constant quality that was held. Independent of the approach used, effort estimation, effort logged, items created, there were spikes in these “normal” periods. This means that either the selected features are of do not help to assess technical debt or the reported process is wrong.

The posed hypotheses were formulated to express the expected development of the graphs to technical debt. The data features, however, are subject to many influences, often, hard to measure, human decisions. Further exploration with more background information has to be undertaken to better evaluate the suitability of management data for technical debt derivation.

If those features can be used for technical debt calculations, there will still be a big social factor that influences this data. Only people that are involved in the process will be able to properly interpret the output.

2.10 Further work

More data with more insight into the development, e.g. in form of associated code metrics, might help in finding a relation between management data and technical debt. Approaches were presented here that can be extended with more background knowledge and then analysed again to get more reliable results.

3 Elaboration of Research

Here, some additions to the research chapter are provided. It starts with some numbers of issues in section 3.1 to give a better understanding of the data. Then, some aspects of the data are further explored and emphasised in section 3.2. Finally, in section 3.3, an approach of how to create a model to simulate technical debt is proposed, that was not included in the main research due to a lack of usable results.

3.1 Data numbers

3.1.1 Amount of items with story points

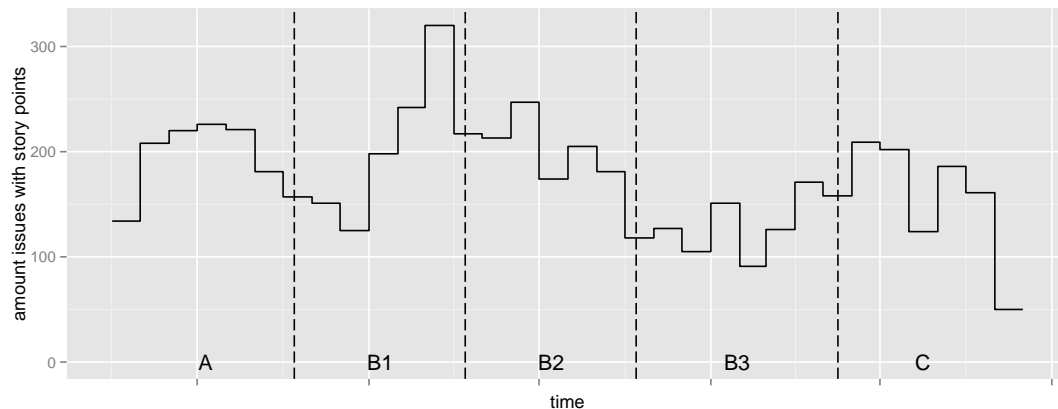


Figure 3.1: The amount of items with story points over time. The number doesn't fluctuate too much to inhibit any findings derived from story points.

To ensure the validity of the story point analysis, the amount of issues with story points are assessed. This is to be sure that there is no drop of the usage of story points that endangers the analysis.

Figure 3.1 shows how many usable items there are for each month. There is a decrease in items in the phases B2 and B3, but no period with a lack of story points.

3.1.2 Number of issues that have certain dates available

Time available	Features	Defects	User Stories
Any	4642	104475	26338
Implementation	1810	207	16124
Completion	2504	70053	23053

Table 3.1: Overview, how many items are available for analysis. The number of items is given that has the respective dates set.

Table 3.1 shows how many issues there are that have certain dates available. Any is valid if there is any of the possible dates set, which is always the case and therefore shows the total amount of available issues.

Implementation demands the dates ImplStart and ImplEnd to be set, which is mostly done for user stories but not for defects, which are usually just closed without giving the implementation time.

Completion demands the ComplDate to have an end date for the issue. Especially in the end of release C, there are many issues that are not yet completed. For analyses that use an issue's lifetime, the completion date is mandatory else the issue is not used for this analysis.

3.1.3 Amount of worklog items per issue

amount	frequency	percentage
0	93530	69.0
1	22375	16.5
2	10273	7.6
3	3972	2.9
4	1971	1.5
5	1037	0.8

Table 3.2: This table shows how many issues have a certain amount of worklog items associated with them. The majority doesn't have any, followed by issues with one worklog entry and only a minority with more.

Table 3.2 shows an overview, how many issues have a certain amount of worklog items associated with them. Most of them don't have a worklog item connected, which limits the possibility to analyse when the effort took place. This is why some approaches are tried for a better reconstruction of the effort.

3.2 Data elaboration

3.2.1 Project Categories

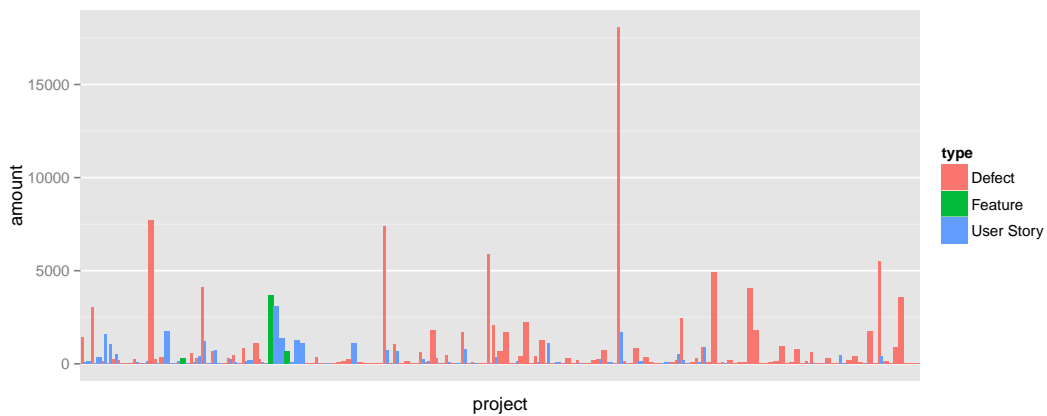


Figure 3.2: Number of elements per project and type. Projects vary a lot in how many issues they have accumulated. Some only have defects, some only user stories and only very few have features

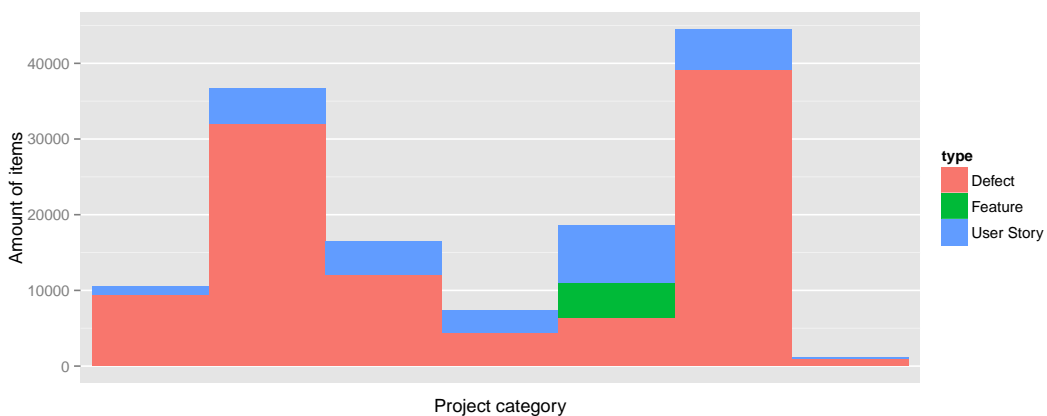


Figure 3.3: This plot shows how many issues there are for each project category. It should be noted that only one category has “Features”.

Each release consists of multiple projects that are associated with project categories.

There are 161 projects for 7 categories. Figure 3.3 shows how many items of which type are associated with each category and figure 3.2 how many are associated with each project.

Only one category has features associated with it. Those are assigned to three projects. This speaks against a general usage of features as they are all collected in one spot.

3.2.2 Constant Workforce

It was reported, that there was a close to constant worker count during a release. This number might have changed from release to release but is not available.

Figure 2.5 shows that the effort spent on defects fluctuates around a constant value, which is lower in release A than in releases B and C. The difference between the mean of the releases can be taken to deduct the ratio of changed workforce to normalise the data. Following Fenton et al. (Fenton & Neil, 1999), this can only be used to estimate the change of the size of the testing department, but not for the developers.

3.2.3 Open items

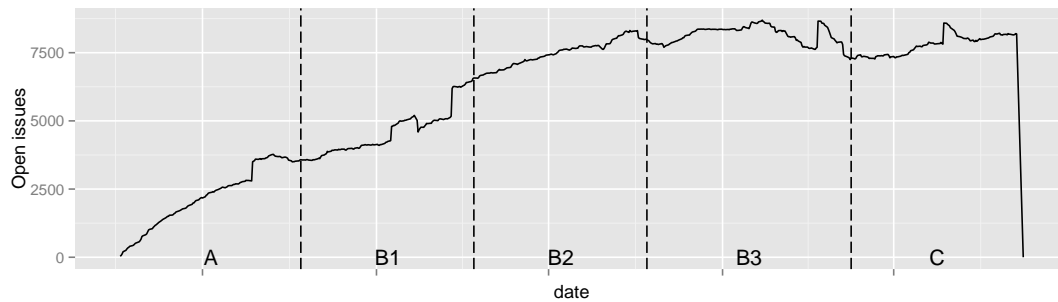
At each point in time, there is a certain amount of issues in progress, which means that they were created by not yet closed. From the premise that an agile development was employed, a constant amount of open issues is expected. This is, however, not the case.

Figure 3.4a shows, how many issues are open at each point in time. The amount is increasing over the progress of releases with short spikes. For each release, the amount of open items increases over time as shown in figure 3.4b. The issues opened in one release are then slowly closed in subsequent releases.

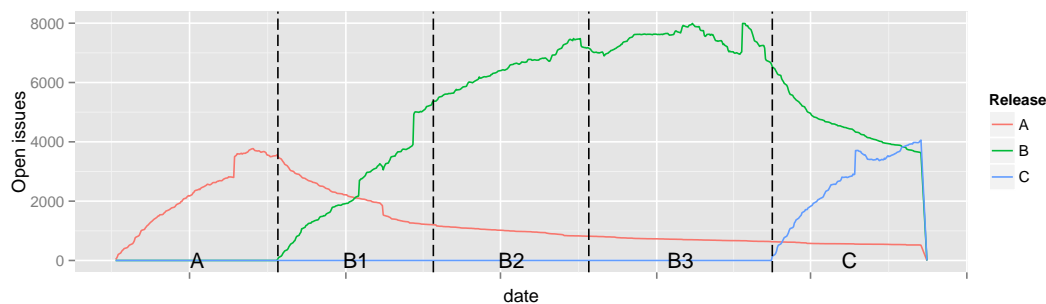
This means, that releases are not separated units but blend into each other.

3.2.4 Percentages of work

The effort fluctuates in the part of the releases that is covered by the data. A factor that adapts to this fluctuation is the percentage of this effort that is spend on user story work and on defect work.



(a) Amount of open issues over time.



(b) Amount of open issues that were created in the respective release.

Figure 3.4: Observation how many issues are open at each time.

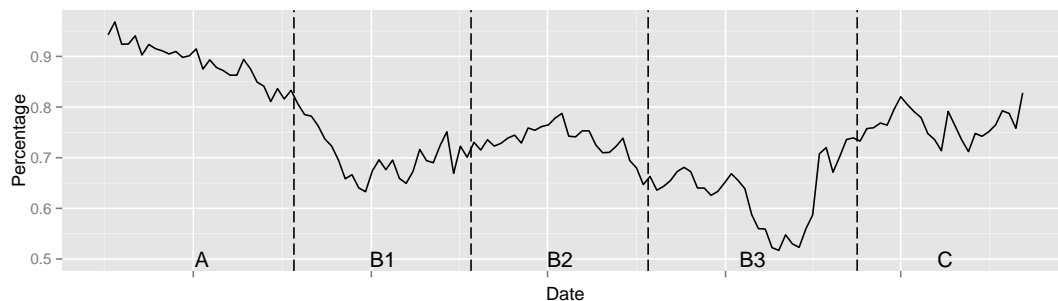


Figure 3.5: Percentage of the effort of user stories of the overall effort

The percentage is a value that can be determined by the management. Thus, it is a way to influence the development. While a direct influence on technical debt was not found, it should be mentioned here as another possible metric to deduct technical debt from. Additionally, the percentage of work distribution can be seen as a factor to influence a technical debt model as done in section 3.3.

Figure 3.5 shows which percentage of the total effort done in each week was spend on user stories.

3.3 Model building

This section describes a model that uses the data to build a model to calculate technical debt. Deficiencies in the data, however, prevent the validation of the model.

3.3.1 Theory

In order to build a model for technical debt, influencing factors are sought as well as their reaction to technical debt and vice versa. Ideas of the relationship between them were:

1. While working on new functionality, errors are introduced that manifest themselves as defects later.
2. Defects are created whose size and amount correlate to the effort spent on new functionality.
3. Effort spent on new functionality accrues technical debt. With a low probability, the opposite effect can take place.
4. Effort spent on defects alleviate technical debt.
5. Increased technical debt leads to more effort spent on each issue - user stories and defects - as it becomes more difficult to realise the same functionality.
6. Technical debt can partially vanish with a low probability independent of the progress. This happens when parts of the code become obsolete and therefore the technical debt associated with this part becomes obsolete as well.
7. The factor, that can be influenced from outside is the ratio, the effort is spent on user stories to defects.

The problem with building this model is the low correlation between work on new functionality to work on defects. The defects logged here don't have any connection to the user story work, they originated from.

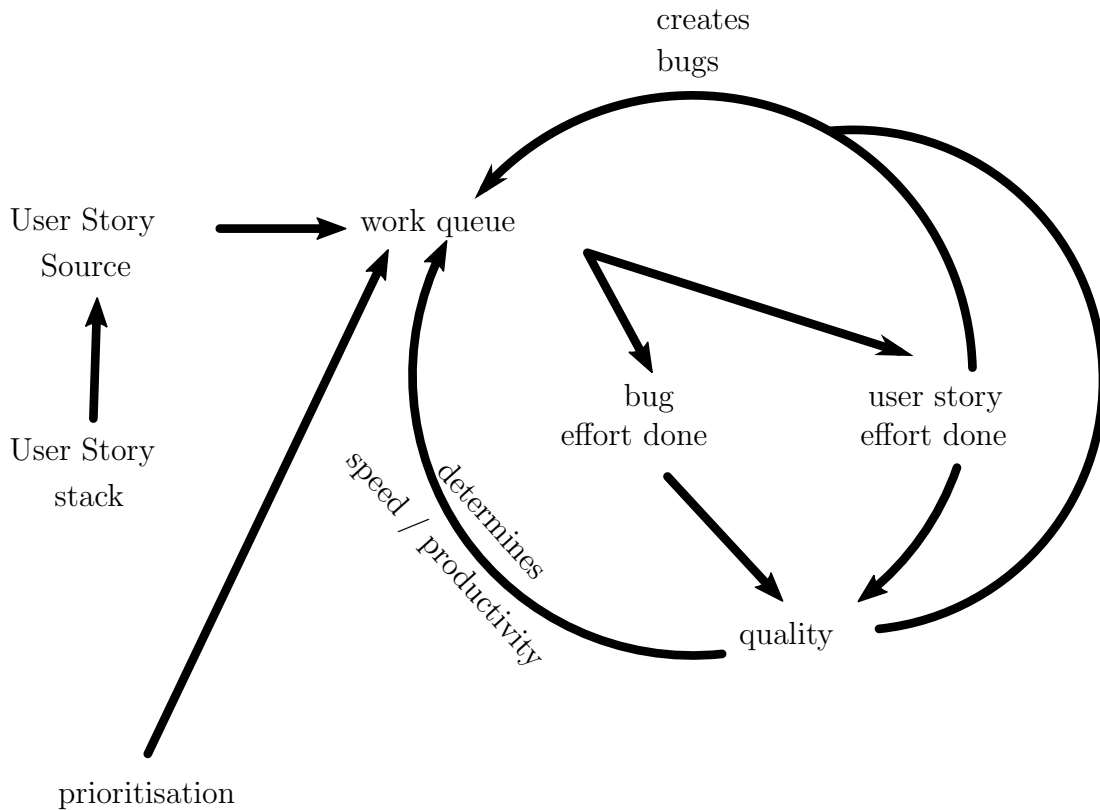


Figure 3.6: Shows a possible modeling approach

3.3.2 Method

Model

Figure 3.6 shows a possible modelling approach. Each release consists of a defined amount of user stories that describe the work that has to be done to reach the release goal. They are stored in the “User Story stack”. Every sprint, which is an iteration in the agile development method, a certain amount of user stories are taken to get implemented. This is the function of the “User Story Source”. For a simulation approach, the “User Story stack” can be omitted and user stories are created at the source according to a certain distribution that is empirically derived. This allows to evaluate the process in a steady state.

Each user story has a predefined effort it demands at a minimum, the base effort. In the model, a user story also has a predefined base duration, i.e. a duration after which it should be completed. This base duration should be matched as closely as possible. For simulation, the base effort and the base duration are taken from probability distributions.

The created user stories are fed into the “work queue”, which is the queue of actively developed on user stories. Analogous to user stories, there are defects in the work queue. There is a total amount of effort each time step, that is divided between the open items in the work queue. For each issue, the base effort that is needed in order to complete it is distributed over its base duration according to the densities depicted in figure 2.6. Issues, where the discrepancy between the needed effort and assigned effort is big, are prioritised. If an issue was assigned enough effort to “complete” it, it drops out of the work queue.

The percentage of available effort that is assigned to work on defects and work on user stories is controlled with the “prioritisation” variable.

The effort that is available to be spend onto the work items is influenced by the quality. If the quality is bad, a certain percentage of the effort does not contribute towards the completion of the active user stories and defects, but is assigned to them without being deducted from the still needed base effort.

Each time step, the effort spend on user stories is taken. Effort spend on features creates new bugs. A new bug is created according to an exponential probability function with the feature effort as the time value. Both the base effort and the base duration are taken from probability distributions.

Effort spend on user stories has a negative impact on the quality. Effort spend on bug fixing has a positive impact on the quality. This is only a general statement. There should be probability functions that mend or deteriorate quality depending on the effort spend on either issue type.

Technical debt is then the amount of effort that has to be spend without progressing issues due to a deteriorated quality. To calculate the technical debt accrued in the model at time t , the effort used to deduct the advancement in base effort from the overall available effort.

$$\text{technical debt}(t) = \text{available effort}(t) - \text{advanced base effort}(t)$$

Since this effort that is not used to advance the issues’ progress is proportional to the quality, technical debt is proportional to quality.

$$\text{technical debt} \propto \text{quality}$$

Distribution fitting

For a discrete simulation of the above described model, probability densities are needed. Exemplary, the amount of user stories generated each day is described in more detail to illustrate the process.

The amount of user stories generated each day is modelled with a negative binomial distribution. Here, a negative binomial with μ 19.8 and size parameter 1.6 is found, which is visualised in figure 3.7.

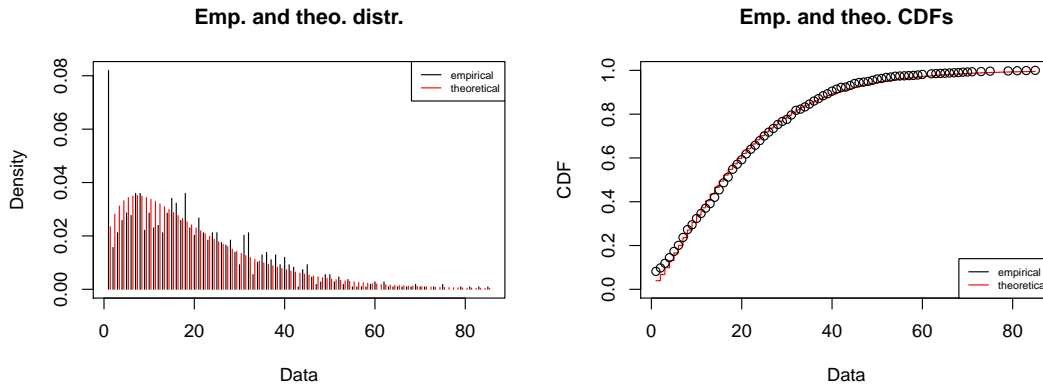


Figure 3.7: Example fitting for the number of created user stories each day.

For other data, the fitting should be repeated and assessed with P-P plots and Q-Q plots as well as goodness of fit tests.

Analogous to this, the size of user stories and their duration can be fitted by a probability function. The base effort and base duration are exponentially distributed.

If probability functions are found for all required variables, their development over a test period has to be assessed. The amount of created bugs follows a probability distributions, whose parameters depend on the quality. Additionally, changes that depend on the life cycle can be tracked this way.

3.3.3 Limitations

The data analysed here didn't contain all the information necessary to build the described model.

There is no described connection between user stories and defects. This is because bugs that occur while working on an user story are fixed immediately and not entered in the data set. The defects available stem from a testing section that is different from the developers. They only find bugs but don't note down where they came from.

Additionally, the quality information is very basic. A rough estimate was used in the thesis as described in figure 2.1, where the quality is strongly related

to technical debt. For a model as it was described above, more information is necessary to assess the implications of the quality on the whole development.

Also, the assumption that effort spend on defects and user stories influence quality and defect creation in a positive and negative way can be too general. A distinction between refactoring work, i.e. quality improving work and implementation tasks would allow for a better calculation of the impact of work on quality.

3.3.4 Conclusion

An approach to create a model based on management data was presented. This model supports simulations of different influences and is easily extended. However, to derive the needed values, more information has to be provided. It was therefore not yet possible to validate the model.

References

- Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., . . . others (2010). Managing technical debt in software-reliant systems. In *Proceedings of the FSE/SDP workshop on Future of software engineering research* (pp. 47–52). ACM.
- Cunningham, W. (1993). The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger*, 4(2), 29–30.
- Eisenberg, R. J. (2012, April). A threshold based approach to technical debt. *ACM SIGSOFT Software Engineering Notes*, 37(2), 1. doi: 10.1145/2108144.2108151
- Fenton, N. E. & Neil, M. (1999, July). Software metrics: successes, failures and new directions. *Journal of Systems and Software*, 47(23), 149–157. doi: 10.1016/S0164-1212(99)00035-7
- Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, . . . Dave Thomas (2001). *Manifesto for Agile Software Development*. Retrieved 2015-06-03, from <http://agilemanifesto.org/>
- Klinger, T., Tarr, P., Wagstrom, P. & Williams, C. (2011). An enterprise perspective on technical debt. In *Proceedings of the 2nd Workshop on managing technical debt* (pp. 35–38). ACM.
- Kruchten, P., Nord, R. L. & Ozkaya, I. (2012). Technical debt: from metaphor to theory and practice. *IEEE Software*(6), 18–21.
- Kruchten, P., Nord, R. L., Ozkaya, I. & Falessi, D. (2013, August). Technical debt: towards a crisper definition report on the 4th international workshop on managing technical debt. *ACM SIGSOFT Software Engineering Notes*, 38(5), 51. doi: 10.1145/2507288.2507326
- Mayr, A., Plosch, R. & Korner, C. (2014, October). A Benchmarking-Based Model for Technical Debt Calculation. In (pp. 305–314). IEEE. doi: 10.1109/QSIC.2014.35
- McCabe, T. J. (1976). A complexity measure. *Software Engineering, IEEE Transactions on*(4), 308–320.
- McConnell, S. (2008). Managing technical debt. *Construx Software Builders, Inc.*

- Nord, R. L., Ozkaya, I., Kruchten, P. & Gonzalez-Rojas, M. (2012, August). In Search of a Metric for Managing Architectural Technical Debt. In (pp. 91–100). IEEE. doi: 10.1109/WICSA-ECSA.212.17
- Nugroho, A., Visser, J. & Kuipers, T. (2011). An empirical model of technical debt and interest. In *Proceedings of the 2nd Workshop on Managing Technical Debt* (pp. 1–8). ACM.
- Stark, G., Durst, R. C. & Vowell, C. (1994, September). Using metrics in management decision making. *Computer*, 27(9), 42–48. doi: 10.1109/2.312037
- Sterling, C. (2010). *Managing software debt: building for inevitable change*. Addison-Wesley Professional.