Hildesheimer Informatik-Berichte

Klaus Schmid

Technical Debt — From Metaphor to Engineering Guidance: A Novel Approach based on Cost Estimation

20.01.2013

Report No. 1/2013, SSE 1/13/E

Software Systems Engineering • Institut für Informatik
Universität Hildesheim • Marienburger Platz 22 • D-31134 Hildesheim

Abstract

In this report, we discuss the notion of Technical Debt and formalize the concept as a basis of disciplined study. This formalization allows us to characterize and measure Technical Debt independent of its source. We will then introduce approximations to measuring Technical Debt that can be more easily applied in practice. As a result we arrive at a method for deciding on tackling Technical Debt, which relies on a formal approach, with clearly identified approximations. We will also illustrate our approach with an example.

Contents

List of Figures

Chapter 1

Introduction

The notion of Technical Debt has gained significant attention in the scientific literature as well as in industrial practice, lately. There is an increasing awareness that a sole focus on creating the next customer-relevant feature might be insufficient for longterm sustainable development [NOS12, BCG+10].

The problem, which is created by a development style, which centers on the next feature to ship, is that software structures are favored, which may include the feature, but may make it at the same time harder to support future change. This is often characterized as *Technical Debt* [Kru12, KNO12].

In this report, we will discuss the basic concept of Technical Debt, which is often only treated on a rather informal level, and will provide a thorough analysis of its conceptual foundations. This critical appraisal of the concept will also support us in providing a more precise definition of Technical Debt and derive from this a well-founded metric as a basis for decision making with respect to technical debt.

According to Kruchten [Kru12] Technical Debt has been introduced by Ward Cunningham at OOPSLA 1992. He is cited with the words

> *"Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite...*
>
> *The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a standstill under the debt load of an unconsolidated implementation, object-oriented or otherwise."*

Technical debt can become particularly visible if different development paths are compared. While some lead to very significant rework, others do lead to little or no rework, even for the same system [ODPGC10]. A similar phenomenon, which may occur in the context of different adoption paths for product lines has been discussed in [Sch03].

Many authors discussed technical debt and related issues over the years, emphasizing the importance and broad applicability of the concept. See Kruchten [Kru12] for a good overview and a list of sources.

It is important to note here, that the concept of Technical Debt was initially introduced merely as a metaphor. A metaphor has many positive qualities, like being easy to understand and seemingly intuitive. However, a metaphor is essentially an analogy, thus it can usually not be precisely mapped to reality. Rather there are certain aspects, where the metaphor simply does not adequately describe the real phenomenon under study; this is sometimes also described as the "analogy breaks down".

In order to better understand what technical debt is — or more precisely is meant to be — let's look at where the technical debt metaphor breaks down:

- Financial debt increases over time (based on the interest rate), independent of what we are doing, even if we do nothing. This is not true for technical debt: if there are no further changes (either internally or externally motivated), it does not change.

- Financial debt is absolute in the sense that it can be measured independently of any specific activities. This is not true for Technical Debt, depending on the specific future development activities, we will see different levels of problems, thus observing different levels of Technical Debt. For example, for an information system, the way existing functionality is realized may be regarded as a liability from the perspective of adding access rights to different user groups, while the same system might appear simple to evolve if only a new functionality should be added.

If we try to phrase the concepts of debt in financial terms, this helps to emphasize the differences. The concept of financial debt FD can be described as:

$$FD = f(b, r, t) \tag{1.1}$$

Thus, the financial debt is a function of a certain base value b (the amount of money that was initially taken out), the interest rate r and the duration over which the interest accrues t. Looking at these parameters, it is not easy to identify a direct mapping to the concept of technical debt described earlier (e.g., as described above: time cannot be directly mapped to time in a development). However, we can roughly map the debt to an initial decision in the realization, which turns out to be problematic from the perspective of future evolution. The debt (in terms of effort) can be interpreted as the difference between what should be done vs. what was done. It gets compounded by additional development that leads to further problems. This relates both to the rate r and the time t.

More generally, we can say that Technical Debt is strongly related to software evolution, respectively the effect a degradation of software quality has on the possibility of evolution. Thus, we will continue our analysis by first focusing on how the underlying problem—degradation of software quality—can be adequately described and measured. Throughout the following analysis, we will focus on two research questions:

1. *Can we develop a precise definition of the concept of technical debt, along with a metric to measure it?*
2. *Can we use this as a basis for deciding when to restructure a system in order to ensure long-term optimal evolution of the architecture?*

If we want to measure the impact of Technical Debt on software development, we need to determine some form of utility loss. Different possibilities exist as a basis to define this utility loss. Which concept we will use will have a fundamental impact on the further details of our analysis approach. Thus, we want to discuss briefly some options:

Cost-avoidance: We can use the reduction in total development costs (i.e., in the long-term) as a basis.

Improved time-to-market: This has benefits on two sides: return is earlier, thus, if we use discounted-cash-flow (DCF) analysis as a basis for characterizing the benefit, we improve the results by earlier returns. But this might also give more returns by getting a higher market-share (higher revenues).

Customer benefit: Customers have a higher benefit, if they have more time to use the system.

...

Many different approaches to modeling utility are possible, just like for utility (or economic advantage) in product line engineering, for which we provided a detailed discussion in [Sch01]. We will follow roughly the guidance, we gave there and start our analysis with the simplest type of utility (cost-avoidance, which can be roughly equated in industrial practice with effort reduction), while neglecting for the moment the aspect of time. (This would amount to a Level-1 model in [Sch01].) The reason is that only once we determine cost (or more precisely effort), we are even able to make any analysis of time as development time in turn depends on the required effort as we discussed in [Sch01]. This is then needed for determining DCF-aspects or to perform an analysis based on lost customer benefit. The next section will formalize Technical Debt from a pure cost perspective. Introducing further aspects of the effect of Technical Debt will be the subject of future work. In this regard our approach to Technical Debt is simpler than other work (like [NOKGR12]), however, on the other hand, we provide a significantly more detailed and formal analysis based on this. We also plan to extend our approach in future work to further utility dimensions.

Chapter 2

Technical Debt as Future Cost

The way we rephrased the problem of measuring Technical Debt in the previous section, we regard it mainly as an indicator to determine when an improvement must be done. However, more precisely the term Technical Debt only emphasizes negative effects (aka a metric), not the decision outcome itself. We will later explicitly address the issue of decision making based on Technical Debt. In this section, we will focus on refining Technical Debt as a metric that can be used to characterize positive effects of structural improvement of software.

2.1 A Basic Formalization of Technical Debt

An important observation is that technical debt is — neither from the perspective of the metaphor, nor from the viewpoint of a technical metric — purely an architectural issue, although the discusssion of this often links it to architecture [KNO12, Kru12]. Rather, it can come in many forms that may result from all parts of the development lifecycle and be contained in all types of artifacts. Thus, we can more adequately describe it as a system (realization) aspect. Architecture decisions only address some part of it, albeit an important one. Due to the importance of architecture to system realization it is sometimes possible to approximate technical debt as architectural debt:

$$TD = f(S) \approx f(A_S), \text{ where } S \text{ denotes a system and } A_S \text{ its architecture.} \tag{2.1}$$

Further on, we will always address Technical Debt as a property of the system (realization) as a whole. We are also fully aware that the notion of debt is sometimes explicitly connected with issues that are not part of the realization (e.g., requirements debt), but we will not further address this. On the other hand we will include aspects of the realization like build system debt (cf. [MGSB12]).

Technical Debt essentially characterizes a problem of the system realization. However, if we are interested in the relevance to the development, we can measure it in terms of the costs that it creates. The cost is not the root cause, which is technical, but it is the key metric that impacts the future development. In this sense, Technical Debt can be seen as a form of cost: *the extra cost of further evolution* that needs to be paid on top of what is absolutely necessary from the point of view of the selected capabilities. So, if we want to define technical debt more precisely, we also need to define the notion of evolution more precisely. In order to do so, we introduce the notion of an evolution step e as one atomic change that is made to a system. It is atomic in the sense that after applying the step e to the system, the system has a meaningful new property. This property can be as small as a correction of a small defect or as large as the introduction of a new feature. We can describe this by writing

$$e(S) = S', \text{ } S \text{ is a system, } e \text{ an evolution step, and } S' \text{ a new system version created by evolving } S \text{, as described by } e. \tag{2.2}$$

We can now introduce the change cost function CC to describe the cost of evolving the system S in the evolution step e:

$$CC(S, e)$$

Using this terminology, we can describe Technical Debt as the amount that the change cost is higher than would be necessary in an ideal setting. For this purpose let $Sys(S)$ denote the set of all systems that are behaviorally equivalent to S and are meaningful technical alternative realizations of the behavior shown by S.[1] Thus, $S, T \in Sys(S)$ have a different implementa-

[1] A key issue here is the notion of *meaningful technical alternative realizations* as we want to not make any particular restriction on how s.th. is implemented, but at the same time, we need to exclude special cases, like the effect of the evolution step is already realized, but not active and then simply activated by a change of a single line of code. This would not be considered meaningful as the implementation would already include effort that was not relevant to the existing behavior of S.

tion, if $S \neq T$, but cannot be distinguished based on their runtime behavior.[2] It should be noted that we do not differentiate between evolution of functions vs. runtime qualities, i.e., qualities that have an observable effect at system runtime. Thus, both a new functionality, as well as a performance enhancement could be an evolution step.

Definition 1. Let S be a system, e an evolution step, and $\hat{S} \in Sys(S)$ an optimal system such that:

$$CC(\hat{S}, e) \leq CC(S', e) \text{ for all } S' \in Sys(S) \tag{2.3}$$

Then we can define the *Technical Debt of S* (wrt. to change cost and the evolution step e) as:

$$TD(S, e) = CC(S, e) - CC(\hat{S}, e) \tag{2.4}$$

We call $\hat{S}$ also the *(cost-)optimal system* relative to e.

We can write this sligthly differently as:

Corollary 1. Let S be a system, e an evolution step, then

$$TD(S, e) = \max\{CC(S, e) - CC(S', e) | S' \in Sys(S)\} \tag{2.5}$$

While this does not describe the root cause of the technical debt, it describes the impact it has on development pretty well.

Our approach to Technical Debt has some further properties, which we regard as adequate, if not positive, but they are not necessarily undisputed:

- Technical Debt is independent of where it came from, whether it was taken on willingly, or not [Fow09]. We only take into account the current state of the system.
- If the expected evolution changes, also the corresponding debt changes. This relates to the fact that what for one evolution path may be a liability, may be irrelevant for another one (or even positive).
- Our concept of Technical Debt is intimately related to modifiability and evolvability. The amount of Technical Debt we acquired as part of the development determines how much harder than necessary it is to evolve s.th.

[2]Of course, in principle, two different implementations are always different also in the runtime behavior, e.g., wrt. execution times. However, we will use this as a shorthand for *not significantly different wrt. their requirements (functional + quality).*

2.2 Extending the formalization to evolution sequences

However, usually, we are not interested in the cost of a single evolution step, but rather, we are interested in the effect of a whole sequence of changes (an evolution scenario or evolution sequence).

Definition 2. Let E be the set of all possible evolution steps (for S). Then E^* denotes the *set of all sequences over* E, i.e., $\vec{e}$ is a finite sequence of evolution steps $\vec{e} = (e_0, e_1, \ldots, e_n) \in E^*$, where the first step in the sequence is e_0. The empty sequence is written as ϵ.

If e is a step in an evolution sequence $\vec{e}$, we denote this as $e \lessdot \vec{e}$. We denote the length of a sequence $\vec{e}$ as $|\vec{e}|$, i.e., $|(e_0, e_1, \ldots, e_n)| = n + 1$. We also describe the concatenation of two evolution sequences $\vec{e} = (e_0, e_1, \ldots, e_n), \vec{e'} = (e'_0, e'_1, \ldots, e'_m) \in E^*$ as $e \oplus e' = (e_0, \ldots, e_n, e'_0, e'_1, \ldots, e'_m)$.

Definition 3. Let E be the set of all possible evolution steps (for S). Further, let $\vec{e}$ be a finite sequence of evolution steps $\vec{e} = (e_0, e_1, \ldots, e_n) \in E^*$ with E^* the set of all sequences over E.

Then the *change cost of system* S, relative to the evolution sequence $\vec{e} = (e_0, e_1, \ldots, e_n)$ is given as:

$$CC(S, \vec{e}) = CC(S, e_0) + CC(e_0(S), e_1) + \ldots + CC(e_{n-1}(\ldots(e_0(S)\ldots)), e_n) \tag{2.6}$$

Alternatively, we can define the change cost for a sequence recursively as:

Corollary 2. Let S be a system, E be the set of evolution steps for S and $\vec{e} = (e_0, e_1, \ldots, e_n) \in E^*$ an evolution sequence, then we can write $CC(S, \vec{e})$ recursively as:

$$CC(S, \epsilon) = 0 \tag{2.7}$$

$$CC(S, (e_0, e_1, \ldots, e_n)) = CC(S, e_0) + CC(e_0(S), (e_1, \ldots, e_n)) \tag{2.8}$$

Using the above analysis, we can also turn again to the issue of technical debt and extend technical debt to sequences of evolution steps:

Definition 4. Let S be a system, $\vec{e} \in E^*$ an evolution sequence, then we call $TD(S, \vec{e})$ the *technical debt relative to* $\vec{e}$ given by

$$TD(S, \vec{e}) = \max\{CC(S, \vec{e}) - CC(T, \vec{e}) | T \in Sys(S)\} \tag{2.9}$$

and we call T *the optimal system relative to* $\vec{e}$.

Note, that while the costs are additive, the technical debt is not. Rather, the following inequalities hold.

Theorem 1. Let S be a system, $\vec{e} = \vec{e_1} \oplus \vec{e_2}$ a composed evolution sequence, then the following properties holds:

$$TD(S, \vec{e}) \leq TD(S, \vec{e_1}) + TD(S, \vec{e_2}) \tag{2.10}$$

$$TD(S, \vec{e}) \leq TD(S, \vec{e_1}) + TD(\vec{e_1}(S), \vec{e_2}) \tag{2.11}$$

The reason is basically, that while for the individual technical debts, we can have different "optimal" systems as reference, there is only one base system as a basis for the combined evolution sequence. Both inequalities will only be equalities if the steps e_1 and e_2 will be completely independent.

2.3 Probabilistic analysis

While the above conceptualization implies that we know the relevant evolution steps upfront, this is often unrealistic. Actually, in the context of an agile development, it would be fundamentally inappropriate to assume that this can be done. Thus, how do we deal with this situation? The approach we take is to use the *expected* value of the Technical Debt as a proxy.

Definition 5. Let S be a system and E^* the set of sequences of evolution steps in E. Further, let p be a probability measure over E, i.e., $p(\vec{e}) \in [0 \ldots\ 1]$ for $\vec{e} \in E$ such that $\sum_{\vec{e} \in E^*} p(\vec{e}) = 1$. Then we define the *Expected Technical Debt* $\widehat{TD}(S)$ as:

$$\widehat{TD}(S) = \sum_{\vec{e} \in E^*} p(\vec{e}) * TD(S, \vec{e}) \tag{2.12}$$

As the set of evolution sequences is potentially infinite, one may be worried that the $\widehat{TD}(S)$ might become infinite as well. However, it should be observed that it still is a technical debt and thus is upper bounded by the effort it takes to transform the existing system into an optimal platform for further evolution.

Chapter 3

Decision Making

Using the definition of Technical Debt introduced above, we will now address the question of how to make development decisions based on Technical Debt. In an ideal and rational development approach, if faced with the choice between a technical improvement (i.e., reducing Technical Debt) and the possibility to add more features, the technical improvement would be done if (and as soon as) this would lead overall to faster development, compensating for the added cost incurred by handling the Technical Debt (restructuring). Otherwise it would not be done as it should be considered gold-platting. We take this idea as a basis for our decision-making approach.

In order to describe systematic decision making, we need to determine the *decision scenario*: the situation when a decision needs to be made. For this let us assume, that development is structured in the form of increments. This might be the next sprint in an agile development or it might be a half-year increment in a more classical development approach. Now, it is time to decide on the development plan for the next period. The key question is: should technical debt be taken on and if so, to what extent?

The basic situation can be characterized as follows: we have a sequence of evolution steps that should be performed in this planning interval (shown as $\vec{e}_{plan}$ in Figure 3.1). Beyond this planning interval, we are not sure yet about the exact further evolution, so we only know potential evolution steps (shown as $\vec{e}_{pot}$ in Figure 3.1). The question is now: shall we incur the restructuring cost (c_{rest} in Figure 3.1)? Or shall we postpone such a restructuring? Note that we do not differentiate between the evolution steps done starting in S vs. in S'; they are really the same, however, their cost is not the same as they are performed on systems with different Technical Debt.

Rationally it is now preferable to first perform a restructuring, if this

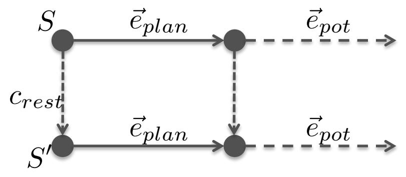


Figure 3.1: Development alternatives

leads to total lower cost.[1] Thus, the restructuring should be done, if

$$c_{rest}+CC(S',\vec{e}_{plan})+CC(\vec{e}_{plan}(S'),\vec{e}_{pot}) \leq CC(S,\vec{e}_{plan})+CC(\vec{e}_{plan}(S),\vec{e}_{pot}) \tag{3.1}$$

It should be noted that it is *not* sufficient to demand

$$c_{rest} + CC(S',\vec{e}_{plan}) \leq CC(S,\vec{e}_{plan}) \tag{3.2}$$

as this would actually mean that the longer term costs (beyond the immediate increment) would be ignored. In particular for very short iterations (like in agile development) this would directly lead to avoiding paying technical debt forever.

In our decision procedure given above as inequality 3.1 the notion of Technical Debt is not explicit.

However, we can easily establish this relation: Let $T \in Sys(S)$ be the optimal system relative to $\vec{e}_{plan}$. Let further be $U \in Sys(\vec{e}_{plan}(S))$ the optimal system relative to $\vec{e}_{pot}$, then we can subtract $CC(T,\vec{e}_{plan})+CC(U,\vec{e}_{pot})$ from both sides in Equation 3.1. This yields:

$$\begin{aligned} c_{rest} + (CC(S',\vec{e}_{plan}) - CC(T,\vec{e}_{plan})) + (CC(\vec{e}_{plan}(S'),\vec{e}_{pot}) \\ - CC(U,\vec{e}_{pot})) \leq (CC(S,\vec{e}_{plan}) - CC(T,\vec{e}_{plan})) \\ + (CC(\vec{e}_{plan}(S),\vec{e}_{pot}) - CC(U,\vec{e}_{pot})) \end{aligned} \tag{3.3}$$

We can now rewrite the parts in braces as technical debt, as follows:

$$c_{rest}+TD(S',\vec{e}_{plan})+TD(\vec{e}_{plan}(S'),\vec{e}_{pot}) \leq TD(S,\vec{e}_{plan})+TD(\vec{e}_{plan}(S),\vec{e}_{pot}) \tag{3.4}$$

Alternatively, we can simplify the description given in Equation 3.1 by combining the two steps $\vec{e}_{plan}, \vec{e}_{pot}$ into a single step $\vec{e}_{plan} \oplus \vec{e}_{pot}$. This leads

[1] It should be noted again that throughout this paper, we abstract from issues like earlier time to market. Only under the assumption that we have a pure cost-focus this decision rule is adequate.

to the following equation:

$$c_{rest} + CC(S', \vec{e}_{plan} \oplus \vec{e}_{pot}) \leq CC(S, \vec{e}_{plan} \oplus \vec{e}_{pot}) \tag{3.5}$$

If we now observe that $S, S' \in Sys(S)$ and assume $\hat{S} \in Sys(S)$ with minimal cost for the change $\vec{e}_{plan} \oplus \vec{e}_{pot}$ exists, then we can rewrite this as:

$$c_{rest} + TD(S', \vec{e}_{plan} \oplus \vec{e}_{pot}) \leq TD(S, \vec{e}_{plan} \oplus \vec{e}_{pot}) \tag{3.6}$$

which can be simply interpreted as paying the debt off right now (c_{rest}) should actually be offset by future interest that no longer needs to be paid.

However, the formalization given by inequality 3.4 is still imprecise as it does not take into account the uncertainty regarding the evolution beyond the current increment. While Figure 3.1 shows only one path, in practice there will be many alternatives for $\vec{e}_{pot}$. Thus, we should replace this with the expected technical debt. This is only necessary for $\vec{e}_{pot}$, as $\vec{e}_{plan}$ is by definition fixed:

$$c_{rest} + TD(S', \vec{e}_{plan}) + \widehat{TD}(\vec{e}_{plan}(S')) \leq TD(S, \vec{e}_{plan}) + \widehat{TD}(\vec{e}_{plan}(S)) \tag{3.7}$$

According to Inequality 3.7 we can determine the (long-term) best course of action, if we can determine the technical debt that is visible for the two alternative realizations in the planned development and in the potential future evolution. This formalization can, however, hardly be used effectively in decision making, as many future aspects are unknown and handling an infinite number of potential scenarios does not work in practice. Thus, we will devote the remainder of this report on determining a set of approximations that can be practically used.

Chapter 4

Practical Approximation

So far, we only focused on providing a *precise* model of Technical Debt, independent of how easy or difficult it is to apply it in practice. In order to derive this model we made a number of assumptions and restrictions, but were always careful to explicitly identify them. In particular, we made along the way the following decisions as part of the modeling:

1. We restrict our analysis of Technical Debt completely to the aspect of cost impact (for now). This was done to simplify the approach while still keeping it meaningful.

2. We define the concept of Technical Debt in a particular way that relies on future evolution steps. This is in line with other work on Technical Debt like [KNO12], but is not representative of all work on Technical Debt.

While the first point must be considered a restriction, the second simply expresses the way we think Technical Debt should be seen. It should also be noted that the way we address Technical Debt is completely independent of its source. Thus, the technical debt might be due to architectural issues [NOS12], or due to build debt [MGSB12], or due to any other source. Our approach is independent of this.

However, so far, the approach described here cannot be readily used in practice. This is due to several issues like that our concept of Technical Debt so far relies on identifying the *optimal* system structure, which is not practicable. As a consequence, we will stepwise introduce further assumptions into our approach that will lead towards a practical decision procedure albeit at the cost of some imprecision. The further assumptions, which we will use are the following:

1. We will assume that under certain conditions costs of an evolution can actually be estimated.
2. We will relax the requirement of knowing the *optimal* system realization with only requiring a reference realization.
3. We will introduce the assumption that the various evolution steps are independent in order to simplify their analysis.
4. We will reduce the potentially infinite set of evolution steps to a finite one.

We will now discuss each of these approximations in turn.

4.1 Cost Estimation

For the further discussion, we will assume that development costs can be estimated, e.g., for a transformation of a system from a certain structure to a new structure we can make a reasonable estimate of this refactoring costs. This assumption is widely used in practice. However, this is less than obvious as the resulting estimate is typically rather imprecise. For generic cost models a margin of error within 30% of the actual value, 80% of the time can be considered very good [CBS99]. Further, this description of margin of error also ignores that a size measure is the basis for estimation, which might in turn have significant uncertainty.

While sometimes the argument is made that higher accuracy can be achieved using models that are calibrated to a specific development environment, the results found by Kitchenham et al. in a meta-study are less conclusive [KMT07]. Several studies could not find a discernible improvement. Actually, the highest intra-organizational cost estimation precision that was found, was that only 20,8% of all projects were outside of a 25% range. (Other studies reported worse results.)

In all these analysis the focus was cost-estimation for projects as a whole. However, in the context of Technical Debt we are interested in the development costs of modifying an implementation only structurally. We can assume that uncertainty of these costs is probably even larger, although there are no explicit cost models for this.[1] Given the high degree of uncertainty that comes with estimates for the cost of restructuring operations, it

[1]While many arguments are made about the impact of refactoring on maintenance costs, this is mostly anecdotal or based on a posteriori analysis. In particular, we could not identify a model that aims at estimating refactoring costs.

seems plausible to allow simplifications of our model even though they may reduce the precision of the results.

4.2 Optimal System Realization

A major problem in our definitions from a practical point of view is that we take as reference an optimal structure of a system. Of course, as there are in principle infinitely many different ways of realizing a system, it is very difficult, if not impossible, to determine the optimal system realization and thus also the difference in costs. As a consequence, we propose to use the concept of Technical Debt not absolutely, but to compare two different possible realizations, respectively the cost of evolving them. We believe that this is also from a practical point of view the most useful approach

As a consequence, we define the *Relative Technical Debt*: this is the Technical Debt measured relative to a different implementation.

Definition 6. Let S be a system, S' an alternative implementation of S, i.e., $S' \in Sys(S)$. Let e be an evolution step, then the *Relative Technical Debt RTD* is given as:

$$RTD(S, S', e) = CC(S, e) - CC(S', e) \tag{4.1}$$

It should be noted that according to this definition the Relative Technical Debt can also be negative. However, we will usually choose S and S' such that $RTD(S, S', e) > 0$.

Analogous to the previous definition of Expected Technical Debt, we can define the Expected Relative Technical Debt $\widehat{RTD}(S, S')$ as follows:

Definition 7. Let S be a system, $S' \in Sys(S)$ an alternative implementation and E^* the set of sequences of evolution steps (for S). Further, let p be a probability measure over E^*, i.e., $p(\vec{e}) \in [0 \dots 1]$ for $\vec{e} \in E^*$ such that $\sum_{\vec{e} \in E^*} p(\vec{e}) = 1$.

Then we define the *Expected Relative Technical Debt $\widehat{RTD}(S)$* as:

$$\widehat{RTD}(S, S') = \sum_{\vec{e} \in E^*} p(\vec{e}) * RTD(S, S', \vec{e}) \tag{4.2}$$

Again we will use this definition typically in such a way that the resulting values are positive.

4.3 Independence of Evolution Steps

A further complication that we need to address to achieve a practically more useful technique is that Equation 4.2 requires to look at all potential sequences of evolution. This is of course impractical, as it leads to a combinatorical explosion. In particular, all possible permutations of a sequence would need to be analyzed. The reason why this is needed, in principle, is that there might be some interdependence among the costs associated with the different steps: if step A follows step B its realization costs may differ from the case that step A is realized directly. However, in most cases for short sequences of steps and most combinations of evolution steps there will be only very little interdependence among the steps and hence the costs will be independent of the ordering of the steps.

As a consequence we argue that a meaningful approximation can be achieved by treating the steps as independent. In case there is a high interdependence between two evolution steps A and B they can be integrated into a single step AB. As long as the number of these cases is very small, it is a practical approach.

Now, if we make the assumption that we can actually approximate the Expected Relative Technical Debt given by Equation 4.2 as follows:

Theorem 2. Let S be a system, $S' \in Sys(S)$ an alternative implementation, E the set of evolution steps (for S) and E^* the set of sequences of evolution steps (for S). Further, let p be a probability measure on E^* with $p(\vec{e}) \in [0 \dots\ 1]$ for $\vec{e} \in E^*$ such that $\sum_{\vec{e} \in E^*} p(\vec{e}) = 1$, according to the definition of *Expected Relative Technical Debt* $\widehat{RTD}(S)$.

Now, let $\tilde{p}$ be a probability measure over E, which is defined for all $e \in E$ as:

$$\tilde{p}(e) = \frac{\sum_{\vec{e} \in E^*, e < \vec{e}} p(\vec{e})}{\sum_{e \in E} \sum_{\vec{e} \in E^*, e < \vec{e}} p(\vec{e})} \tag{4.3}$$

then we approximate the *Expected Relative Technical Debt* $\widehat{RTD}(S)$ as:

$$\widehat{RTD}(S, S') \approx \sum_{e \in E} \tilde{p}(e) * RTD(S, S', e) \tag{4.4}$$

Thus, if we assume (mostly) independence among the individual evolution steps (also from a cost perspective), then we can approximate the technical debt by analyzing only costs of individual evolution steps.

4.4 Representative Evolution Steps

However, in a realistic situation, we will also not be able to look at all possible evolution steps as there are just too many (typically infinite). Rather, we need to focus on a subset of the evolution steps.

This has already been described above as Approximate Technical Debt. Our definition of Approximate Technical Debt does, however, not particularly restrict the probabilities. If we can assume that our set of selected evolution steps is representative, the final result will be representative as well.

However, an infinite set of evolution sequences is still rather unwieldy from a practical perspective. Thus, we also define an approximation to the expected value based on a set of selected (representative) evolution sequences.

Definition 8. Let S be a system and E^* the set of sequences of evolution steps in E. Further, let p be a probability measure over E^*, i.e., $p(\vec{e}) \in [0 \ldots\ 1]$ for $\vec{e} \in E^*$ such that $\sum_{\vec{e} \in E^*} p(\vec{e}) = 1$. $\widehat{TD}(S)$ shall denote the Expected Technical Debt.

$\tilde{p}$ shall be a function over E^* and $\tilde{E} \subseteq E^*$ a (representative) finite set of evolution sequences, such that for at least one $\vec{e} \in \tilde{E}, \tilde{p}(\vec{e}) \neq 0$. Then $\tilde{p}$ is given by:

$$\tilde{p}(\vec{e}) = \begin{cases} \dfrac{p(\vec{e})}{\sum_{\vec{e} \in \tilde{E}} p(\vec{e})} & , \vec{e} \in \tilde{E} \\ 0 & , \vec{e} \in E^* \setminus \tilde{E} \end{cases} \tag{4.5}$$

Then $\tilde{p}$ is a probability measure, i.e., $\tilde{p}(\vec{e}) \in [0 \ldots\ 1]$ for $\vec{e} \in E$ such that $\sum_{\vec{e} \in E^*} \tilde{p}(\vec{e}) = 1$. Further,

$$\widetilde{TD}_{\tilde{E}}(S) = \sum_{\vec{e} \in \tilde{E}} \tilde{p}(\vec{e}) * TD(S, \vec{e}) \tag{4.6}$$

defines the *$\tilde{E}$-Approximate Technical Debt.* If $\tilde{E}$ contains exactly n elements $\vec{e}$ such that $\tilde{p}(\vec{e}) \neq 0$, then we also write $\widetilde{TD}_{n,\tilde{E}}$. If $\tilde{E}$ is chosen so as to minimize the difference $|\widetilde{TD}_{n,\tilde{E}}(S) - \widehat{TD}(S)|$, then we also write $\widetilde{TD}_n(S)$ to denote the *n-Approximate Technical Debt.* The more elements n we allow, the more precise the approximation can be.

Using the above definitions, it is clear that we ideally use n-Approximate Technical Debt-approximations. In general, this will include the sequences with particularly high probability of occurrence. However, a sequence with a very high value of Technical Debt and somewhat lower probability may sometimes be more relevant, so that the probability is not the only parameter. Using the above definitions, we can also make the following observation:

Corollary 3. Let S be a system and $\widehat{TD}(S)$ the Expected Technical Debt and $\widetilde{TD}_n(S)$ the n-Approximate Technical Debt, then $\lim\limits_{n\to\infty} \widetilde{TD}_n(S) = \widehat{TD}(S)$

Thus, with increasing effort (considering more evaluation sequences), we will successively approximate the true Technical Debt.

However, the above approximation has again the annoying problem that the set of sequences is rather large. We can thus combine it with the approximation described in Section 4.3 to only use the individual steps.

In order to capture this idea, we introduce the following definition:

Definition 9. Let S be a system and E^* the set of sequences of evolution steps in E. Further, let p be a probability measure over E^*, i.e., $p(\vec{e}) \in [0\ldots\ 1]$ for $\vec{e} \in E^*$ such that $\sum_{\vec{e}\in E^*} p(\vec{e}) = 1$. $\widehat{TD}(S)$ shall denote the corresponding Expected Technical Debt.

$\tilde{p}$ shall be a function over E and $\tilde{E} \subseteq E$ a (representative) finite set of evolution steps, such that for at least one $e \in \tilde{E}, e \lessdot \vec{e}, p(\vec{e}) \neq 0$. Then $\tilde{p}$ is given by:

$$\tilde{p}(e) = \begin{cases} \dfrac{\sum_{\vec{e}\in E^*, e\lessdot\vec{e}} p(\vec{e})}{\sum_{e\in\tilde{E}} \sum_{\vec{e}\in E^*, e\lessdot\vec{e}} p(\vec{e})} & , e \in \tilde{E} \\ 0 & , e \in E \setminus \tilde{E} \end{cases} \tag{4.7}$$

Then $\tilde{p}$ is a probability measure over E, i.e., $\tilde{p}(e) \in [0\ldots\ 1]$ for $e \in E$ such that $\sum_{e\in E} \tilde{p}(e) = 1$.

Further,

$$\widetilde{TD}_{\tilde{E}}(S) = \sum_{e\in\tilde{E}} \tilde{p}(e) * TD(S, e) * \sum_{e\in\tilde{E}} \sum_{\vec{e}\in E^*, e\lessdot\vec{e}} p(\vec{e}) \tag{4.8}$$

defines the *Stepwise $\tilde{E}$-Approximate Technical Debt.* If $\tilde{E}$ contains exactly n elements e such that $\tilde{p}(e) \neq 0$, then we also write $\widetilde{TD}_{n,\tilde{E}}$. If $\tilde{E}$ is chosen such that the difference $|\widetilde{TD}_{n,\tilde{E}}(S) - \widehat{TD}(S)|$ is minimized, then we also write $\widetilde{TD}_n(S)$ to denote the *Stepwise n-Approximate Technical Debt.*

Note, that in this case, in general $\lim\limits_{n\to\infty} \widetilde{TD}_n(S) = \widehat{TD}(S)$ will not hold. This is due to equation 2.10.

Chapter 5

An Approximate Decision Procedure

In the previous section, we discussed the possible approximations that help to significantly reduce the total effort of determining — if not an exact value of the Technical Debt — at least a reasonable approximation. In particular, we discussed in Section 4.1 that in general the uncertainty of cost estimates is rather significant and will thus significantly overshadow other smaller errors that result from approximations. It also tells us that for any approach to dealing with Technical Debt on a cost basis, we need to take very significant (on the order of 30%) deviations into account. Adding further utility contributions like time to market, user benefit, etc. will lead to even larger margins of error. Thus, while most approaches typically ignore this, this leads us to argue that some systematic loss of precision is probably acceptable, given that the intrinsic uncertainty of any such approach is already significant.

In this section, we will first discuss how to use the various approximations that have been discussed in the previous sections and will use them to derive an approximate version of the decision procedure outlined in Section 3. In a second step, we will illustrate how this can be integrated in a simplified version into a development environment.

5.1 Putting it together

We will now discuss how the decision procedure that was given in Section 3 and the approximations given in Section 4 can be combined. Major parts of the approximation have already been done. We will now use equation 3.1 as a starting point and combine it with the following approximations to arrive at a refined approximate decision procedure:

- We will use an alternative, given system realization as opposed to an abstract and theoretically optimal system realization.
- We will focus on individual evolution steps, instead of evolution sequences (i.e., we will assume independence of the steps).
- We will use a representative set of steps instead of all possible evolution steps.

The reason why we use equation 3.1 and not, for example, equation 3.4 is that the former makes the combination with a selected reference realization easier.

If we start with Equation 3.4 and simply use S' as the "optimal" reference realization, i.e., S' represents the alternative realization with the Technical Debt removed. Then we can rewrite it by substracting the cost-terms relating to S' from both sides:

$$c_{rest} \leq RTD(S, S', \vec{e}_{plan}) + RTD(\vec{e}_{plan}(S), \vec{e}_{plan}(S'), \vec{e}_{pot}) \tag{5.1}$$

We now need to take into account that the potential evolution $\vec{e}_{pot}$ is not actually known. We thus need to replace this by the Expected Technical Debt, as given in 4.2. This yields:

$$c_{rest} \leq RTD(S, S', \vec{e}_{plan}) + \widehat{RTD}(\vec{e}_{plan}(S), \vec{e}_{plan}(S')) \tag{5.2}$$

We can now define the Stepwise n-Approximate Relative Technical Debt in analogy to Definition 9 as follows:

Definition 10. Let S be a system, and $S' \in Sys(S)$, and E^* the set of sequences of evolution steps in E. Further, let p be a probability measure over E^*, i.e., $p(\vec{e}) \in [0 \ldots 1]$ for $\vec{e} \in E^*$ such that $\sum_{\vec{e} \in E^*} p(\vec{e}) = 1$. $\widehat{RTD}(S, S')$ shall denote the corresponding Expected Relative Technical Debt.

$\tilde{p}$ shall be a function over E and $\tilde{E} \subseteq E$ a (representative) finite set of evolution steps, such that for at least one $e \in \tilde{E}, e \lessdot \vec{e}, \tilde{p}(\vec{e}) \neq 0$. Then $\tilde{p}$ is given by:

$$\tilde{p}(e) = \begin{cases} \dfrac{\sum_{\vec{e} \in E^*, e \lessdot \vec{e}} p(\vec{e})}{\sum_{e \in \tilde{E}} \sum_{\vec{e} \in E^*, e \lessdot \vec{e}} p(\vec{e})} & , e \in \tilde{E} \\ 0 & , e \in E \setminus \tilde{E} \end{cases} \tag{5.3}$$

Then $\tilde{p}$ is a probability measure over E, i.e., $\tilde{p}(e) \in [0 \ldots 1]$ for $e \in E$ such that $\sum_{e \in E} \tilde{p}(e) = 1$.

Further,

$$\widetilde{RTD}_{\tilde{E}}(S, S') = \sum_{e \in \tilde{E}} \tilde{p}(e) * RTD(S, S', e) * \sum_{e \in \tilde{E}} \sum_{\vec{e} \in E^*, e \lessdot \vec{e}} p(\vec{e}) \tag{5.4}$$

defines the *Stepwise Relative $\tilde{E}$-Approximate Technical Debt.* If $\tilde{E}$ contains exactly n elements e such that $\tilde{p}(e) \neq 0$, then we also write $\widetilde{RTD}_{n,\tilde{E}}$. If $\tilde{E}$ is chosen so as to minimize the difference $|\widetilde{TD}_{n,\tilde{E}}(S) - \widehat{TD}(S)|$, then we also write $\widetilde{RTD}_n(S)$ to denote the *Stepwise Relative n-Approximate Technical Debt.*

We can now use this definition and rewrite equation 5.2 as follows:

$$c_{rest} \leq RTD(S, S', \vec{e}_{plan}) + \widetilde{RTD}(S, S') \tag{5.5}$$

We can replace $\vec{e}_{plan}(S)$ by S (and similar for S') by using the assumption of the independence of the steps in $\vec{e}_{plan}$ and in $\vec{e}_{pot}$.

Further, we can apply the assumption of step-independence also to the $\vec{e}_{plan}$-term of the decision procedure. This yields:

$$c_{rest} \leq \sum_{e \lessdot \vec{e}_{plan}} RTD(S, S', e) + \widetilde{TD}(S, S') \tag{5.6}$$

The advantage of this formulation is that now all evaluations are broken down into computations based on individual steps. Inherently, the only difference is that the steps in $\vec{e}_{plan}$ are 100% certain as they are planned for the next iteration, while the potential ones have a probability smaller than 100%.

5.2 Integrating it into a Practical Methodology

We can now put the pieces together and integrate the individual steps in a methodology that takes advantage of the approximations and decision rules discussed and that can be integrated in actual software development with limited effort.

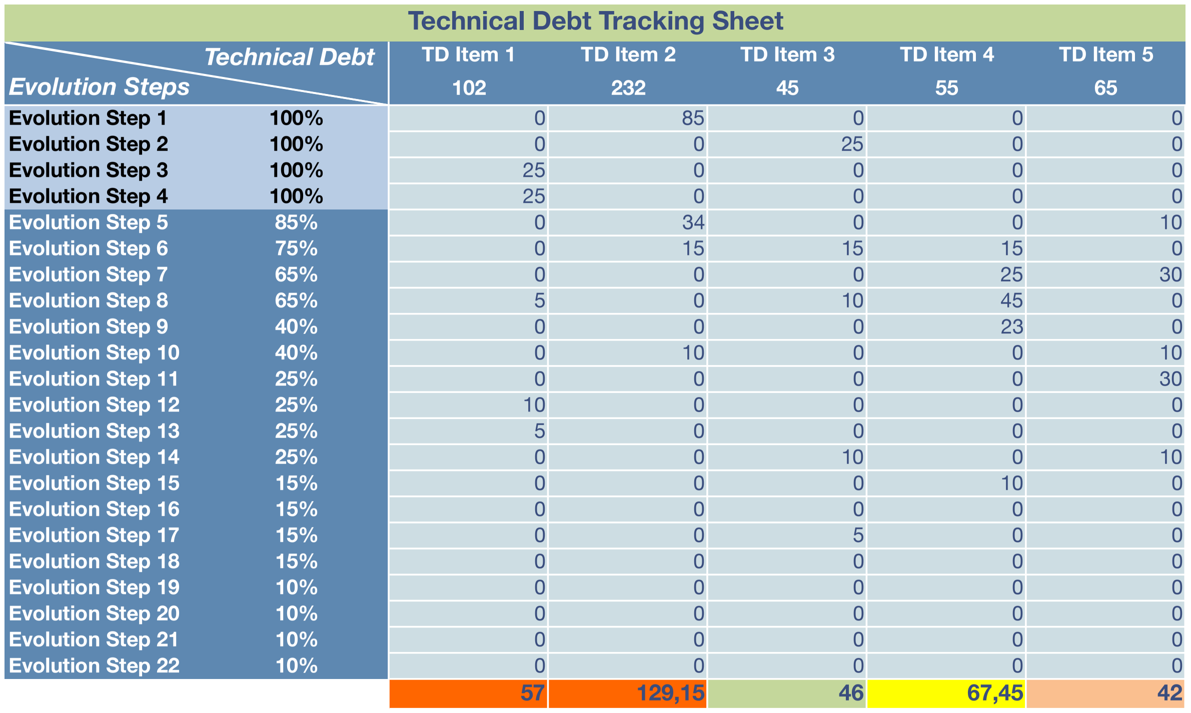
The core idea is that future, planned developments should be sub-divided into two planning areas: upcoming development steps that relate to capability enhancements (functional or quality wise), corrections and so forth and on the other hand technical improvements that relate to different ways of paying of debt, which have no impact on the external, runtime properties of a system. Using the concepts of agile development, we could see this as a traditional product backlog (the capabilities) and a new concept the *Technical Backlog*. The two should be kept separate as they should be treated differently.

The planned capability list is subdivided (as usual) in two parts: the sprint plan and the product backlog. All evolution steps that are scheduled for the sprint plan are treated as certain, while all evolution steps that are in the remaining backlog are treated as uncertain. We can now implement the approach outlined above in a simple way, by adding the following information:

- Per element in the planned capability list, i.e., sprint plan + product backlog, we will capture its probability. This will be updated once an entry goes onto the sprint plan to 100%.
- Per element in the technical backlog we give the estimated costs for its realization. (This corresponds to the cost c_{rest}.)
- Moreover, for each combination of entries in the planned capability list and the technical backlog, we estimate how much the cost of implementing this capability would be reduced, if previously the technical improvement would be made. Note, this corresponds to the Stepwise Relative Technical Debt.

This can be implemented easily in the form of a matrix. Moreover, we assume that this matrix will be sparsely populated, i.e., only few elements of the technical backlog will have an impact on any given capability implementation. Moreover, the number of technical backlog items should be small. This should only contain the non-trivial activities that seem to be relevant to reduce the cost of further development. This is shown in Figure 5.1.

In Figure 5.1 we see five different examples for items on the technical debt list. We also see four items as planned in the rows (the probability is 100% and the items are written in black). These four items would determine the content of the next iteration. The cost of performing the technical debt removal is given on the top of the column, directly under the name of the technical debt item. The benefit gained (the right-hand side of equation 5.6) is given on the bottom of the column. Colors are used to distinguish different situations:



Technical Debt / Evolution Steps		TD Item 1	TD Item 2	TD Item 3	TD Item 4	TD Item 5
		102	232	45	55	65
Evolution Step 1	100%	0	85	0	0	0
Evolution Step 2	100%	0	0	25	0	0
Evolution Step 3	100%	25	0	0	0	0
Evolution Step 4	100%	25	0	0	0	0
Evolution Step 5	85%	0	34	0	0	10
Evolution Step 6	75%	0	15	15	15	0
Evolution Step 7	65%	0	0	0	25	30
Evolution Step 8	65%	5	0	10	45	0
Evolution Step 9	40%	0	0	0	23	0
Evolution Step 10	40%	0	10	0	0	10
Evolution Step 11	25%	0	0	0	0	30
Evolution Step 12	25%	10	0	0	0	0
Evolution Step 13	25%	5	0	0	0	0
Evolution Step 14	25%	0	0	10	0	10
Evolution Step 15	15%	0	0	0	10	0
Evolution Step 16	15%	0	0	0	0	0
Evolution Step 17	15%	0	0	5	0	0
Evolution Step 18	15%	0	0	0	0	0
Evolution Step 19	10%	0	0	0	0	0
Evolution Step 20	10%	0	0	0	0	0
Evolution Step 21	10%	0	0	0	0	0
Evolution Step 22	10%	0	0	0	0	0
		57	129,15	46	67,45	42

Figure 5.1: Technical Debt Log

- The first two columns (orange/dark grey) show the situation of two technical debt items that are impacting planned capabilities, however, the benefit that would be accrued in the planned and in the potential capabilities would be sufficient to recover the costs. Thus, the recommendation would be to *not* adress this.

- TD item 3 is marked green (light grey) as the cost of addressing this technical debt (i.e., performing the corresponding restructuring) would be offset by planned and future savings. However, this would only be achieved barely. Moreover, if only planned and not also potential capabilities (the product backlog) would be taken into account, it would not be worth addressing this. Hence, this hinges critically on the consideration of the product backlog.

- TD item 4 is clearly (cost of 55 and savings of 67.45) necessary to address. However, based on the decision rule given above, this would not be addressed now, as it has no impact on any planned capability. Only as soon as a sprint would include an evolution step with a a relation to this TD item, this would be evaluated anew.

- TD item 5 is currently not worth addressing, but it also impacts no capability that is currently planned. Thus, it will just stay in the Technical Debt Log. As the development moves on the probabilities may change and hence the evaluation.

It should be noted that the table given in Figure 5.1 is artificial, with the purpose to present the core ideas. In particular, we expect that in real world situations most technical debt items will only have an impact on few capabilities, leading to a matrix less populated then the one given in Figure 5.1.

5.3 Discussion of the Approach

It should be observed that the described approach is a faithful realization of the decision procedure given by equation 5.6. Hence, we know that it rests on a sound formal foundation, but we also know that it has some explicitly identified approximations. For example, if several Technical Debt items or several evolution steps (the capabilities) are strongly related our approximation of addressing only individual steps will not be sufficient. This should then by treated with an adapted approach, addressing explicitly the evolution sequences instead of only evolution steps.

There might be also some practical problems. Our decision procedure has two characteristics that might make it hard to apply in practice. These are:

- It might be that a technical debt is initially determined that it should not be addressed (e.g., in Technical Debt item 1 in Figure 5.1), but at a later stage it might turn into something necessary to address. This could happen either because additional potential items are brought on the backlog or, because the probabilities change. In this case cost (in the example 50) could have been avoided, but was taken on. This is undesirable, but still it is the most rationale decision, given the information at hand, at the corresponding point in time.
- Waiting till the technical debt item influences a planned item, might make it very hard to take on the cost for addressing the technical debt in this very moment. It might entail that the next sprint would be completely absorbed by technical debt removal. From this practical point of view an approach like the *architectural runway* might be more appropriate [BNO12], even though it is less optimal from a pure cost point of view. However, our approach could be used in combination with this approach rather easily. The only difference would be that we would need to drop the condition that only technical debt items relevant to the planned capabilities are taken into account.

Chapter 6

Conclusion and Outlook

In this article, we discussed the concept of Technical Debt and provided a detailed and formal analysis. In particular, we focused on deriving a metric for Technical Debt along with a decision procedure, which helped us to decide whether a certain Technical Debt-item should be addressed or not.

We introduced several approximations that helped us to simplify our decision procedure in a way, so that it can be effectively applied in real-world situations as it creates little overhead. We even expect that this approach is simple enough that it can be easily applied in the context of an agile approach. In the special case of technical debt items that address architectural improvements, the approach would thus also contribute to reconciling architecture and agile development, a challenge identified as important by Abrahamsson et al. in [ABK10]. We did also illustrate this with a simple example.

While we assume that this approach is sufficient for covering Technical Debt from a cost-perspective, for the future many open issues remain like addressing time, both from the perspective of discounted cash flow as well as from the perspective of lost user benefit or lost revenue, if a capability is delivered later. We also plan to refine our approach in a way that will support the selection of technical activities on a more fine-grained level as the current version does.

Acknowledgements

I would like to thank Ipek Ozkaya and Robert Nord for numerous patient discussions on technical debt. Thanks go also to Felix Bachmann for many discussions and some early reviewing of this work and to David Garlan for some interesting discussions on this subject. Finally, I would like to Linda Northrop and the time at the SEI/RTSS that made my stay and this work possible.

Bibliography

[ABK10] Pekka Abrahamsson, Muhammad Ali Babar, and Philippe Kruchten. Agility and architecture: Can they coexist? *Software, IEEE*, 27(2):16–22, march/april 2010.

[BCG$^+$10] Nanette Brown, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan MacCormack, Robert Nord, Ipek Ozkaya, Raghvinder Sangwan, Carolyn Seaman, Kevin Sullivan, and Nico Zazworka. Managing technical debt in software-reliant systems. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, FoSER '10, pages 47–52, New York, NY, USA, 2010. ACM.

[BNO12] Felix Bachmann, Robert L. Nord, and Ipek Ozkaya. Architectural tactics to support rapid and agile stability. *CrossTalk*, pages 20–25, May/June 2012.

[CBS99] Sunita Chulani, Barry Boehm, and Bert Steece. Bayesian analysis of empirical software engineering cost models. *IEEE Transactions on Software Engineering*, 25(4):573–583, July/August 1999.

[Fow09] Martin Fowler. Technical debt quadrant. Bliki [Blog], Available at: `www.martinfowler.com/bliki/TechnicalDebtQuadrant.html`, 2009. Last visited 20.01.2013.

[KMT07] Barbara A. Kitchenham, Emilia Mendes, and Guilherme H. Travassos. Cross versus within-company cost estimation studies: A systematic review. *IEEE Transactions on Software Engineering*, 33(5):316–329, May 2007.

[KNO12] Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya. Technical debt: From metaphor to theory and practice. *IEEE Software*, 29(6):18–21, November/December 2012.

[Kru12] Philippe Kruchten. Tutorial: Technical debt — from metaphor to theory and practice. Online available at: `http://pkruchten.files.wordpress.com/2012/08/kruchten-120821-techdebt.pdf`, Last visited: 20.01.13, 2012.

[MGSB12] J. David Morgenthaler, Misha Gridnev, Raluca Sauciuc, and Sanjay Bhansali. Searching for build debt: Experiences managing technical debt at google. In *Proceedings of the Third International Workshop on Managing Technical Debt*, pages 1–6, 2012.

[NOKGR12] Robert L. Nord, Ipek Ozkaya, Philippe Kruchten, and Marco Gonzalez-Rojas. In search of a metric for managing architectural technical debt. In *10th Working IEEE/IFIP Conference on Software Architecture (WICSA 2012)*. IEEE Computer Society, 2012.

[NOS12] Robert L. Nord, Ipek Ozkaya, and Raghvinder S. Sangwan. Analysis of dependencies during software release planning to guide architectural decision making amid competing interests in value and costs. *Journal of Systems and Software*, 2012.

[ODPGC10] Ipek Ozkaya, Andres Diaz-Pace, Arie Gurfinkel, and Sagar Chaki. Using architecturally significant requirements for guiding system evolution. In *Proceedings of 14th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 129–138, 2010.

[Sch01] Klaus Schmid. An initial model of product line economics. In *Proceedings of the 4th International Workshop on Product Family Engineering (PFE4)*, pages 38–50, 2001.

[Sch03] Klaus Schmid. A quantitative model of the value of architecture in product line adoption. In Frank van der Linden, editor, *Proceedings of the 5th International Workshop on Product Family Engineering (PFE5)*, number 3014 in LNCS, pages 32–43. Springer, 2003.