# Modelling Propagation of Technical Debt

Johannes Holvitie*†, Sherlock A. Licorish‡, and Ville Leppänen*†

* *TUCS*
*Turku Centre for Computer Science,*
*Software Development Laboratory,*
*Turku, Finland*

† *University of Turku,*
*Dept. of Information Technology,*
*Turku, Finland*
*{jjholv, ville.leppanen}@utu.fi*

‡ *University of Otago,*
*Department of Information Science,*
*Dunedin, Otago, New Zealand*
*sherlock.licorish@otago.ac.nz*

***Abstract*—Noting the overwhelming speed during software development, and particularly in environments where rapid delivery is the norm, the lack of accumulated technical debt information could result in ineffective management. We introduce technical debt propagation channels in this paper to advance software maintenance research on two accounts: (1) We describe the fundamental components for the channels, allowing identification of distinct channels, and (2) we describe a procedure to identify and abstract technical debt channels in order to produce technical debt propagation models. Our propagation models pursue automation of technical debt information maintenance with program analysis results, and translation of the maintained information between existing–and currently disconnected–technical debt management solutions. We expect the immediate technical debt information to enhance applicability and effectiveness of existing technical debt management approaches.**

***Keywords*-technical debt propagation; software analysis;**

## I. Introduction

Sub-optimalities in the software emerge due to trade-offs, oversight, or environmental changes, and they persistently affect future iterations until seen to [1]. Technical debt management pursues introducing structure and order into these sub-optimalities so as to resolve them adequately to the software development project. Prior research on technical debt has successfully introduced technical debt identification, estimation, and decision making approaches, or described how solutions from other domains can be adopted for these phases (e.g. [2], [3]). The majority of the solutions however come with preset technology or project contexts which is problematic. Indeed, Holvitie et al. [4] have noted that technical debt is capable of propagating between components that exist in different phases of the software development life-cycle, and they have further postulated that technical debt is capable of leaving its original technology context [5]. Since both the identification and estimation phases are context dependent (assessed sub-optimalities reside in predefined technology contexts like source code implemented in the Java language), research on how technical debt propagates within and between these contexts is required, but currently absent.

Hence, in this paper we make the proposal for technical debt propagation models, which are abstractions from technical debt propagation channels observed during software development undertakings. The models contribute to technical debt management by explaining how technical debt information transforms from one context to another.

## II. Related Work

### A. Technical Debt Propagation and its Estimation

McGregor et al. [6] hypothesized that there are two ways for technical debt to propagate within ecosystems. Firstly, the debt of a new software asset equals “the sum of technical debt incurred by the decisions made during the asset’s development and some of the technical debt from the assets that were integrated to it”. They also noted that multiple implementation layers can diminish debt. Secondly, they establish that the user of an asset did not accumulate technical debt directly, but felt its effects indirectly. Finally, they note that compounding debt may become larger than the sum of its sources [6].

Schmid [7] provides a formal definition for technical debt accumulation. An evolution step is defined as an externally observable behavior change that introduces a characteristic to a system. Technical debt accumulation (interpretable as the cumulative effect of technical debt propagation) is described as the difference in costs to implement a sequence of evolutionary measures in the current system, in comparison to an optimal system.

Regarding, especially value, estimation of identified technical debt, Zazworka et al. [8] note from their case-study that principal and interest characteristics of technical debt are not bound to the type of technical debt. Eisenberg [9] notes that threshold based management approaches require defining the cost associated with reducing each type of technical debt. Falessi et al. [10] collect requirements for technical debt tool support. For valuation of the debt’s interest, they note that a single debt may affect diverse quality characteristics differently. Falessi et al. also note McGegor et al.’s [6] compound property.

### B. Software Entity Interconnections

Kim et al. [11] discuss an approach for classifying software changes. They first extract change history for projects from software configuration management systems. The bug-introducing changes are then identified and feature extraction is applied for them in order to produce a classifier.

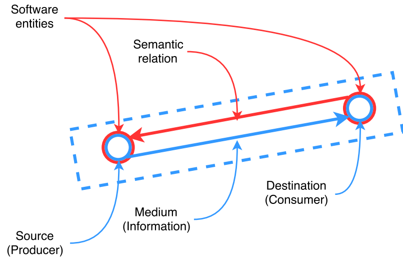


Figure 1. Software entity relationship (red; top descriptions) with a super imposed technical debt channel (blue; bottom descriptions)

Notably, bug-introducing changes are identified by back-tracking from the bug-fixing change, and feature extraction takes associated log messages into account [11].

The Software Process Engineering Metamodel (SPEM) pursues formalization of software processes via definition of process components, component relations, and the impulses flowing within. Effects of the interconnections are not described by the model, but Rochd et al.'s [12] work can be seen explaining this via superimposing synchronization for the modeled components.

## III. Technical Debt Propagation Channels

### A. Channel Features

Our objective is to describe technical debt propagation channels capturing the effects of sub-optimal software entity alterations. The alterations correspond to software changes as argued by Holvitie et al. [5]. These changes (entity alterations) are captured here as Entity-Relationships (ER) aligned with Kagdi et al.'s [13] definition of software change as "the addition, deletion, or modification of any software artifact such that it alters, or requires amendment of, the original assumption of the subject system".

As a technical debt channel captures a particular, distinct, instance of technical debt accumulation, the channel's definition is always comprised of a single entity-entity-relationship. Their combination would correspond to an instance of a synchronized SPEM (describing the propagation process for—i.e. the channels available to—instances of technical debt in the project specific context set).

Assume a collection of interconnected software entities (e.g. variable declarations and calls in the implementation technology's context and their descriptions in the documentation technology's context) to form a graph. The potential channels for software change is a super-set of the interconnection graph since the definition for a software change also considers assumptions (implicit channels in Section III-A1).

Figure 1 depicts one instance of a potential software change. As per the previous description of a software change between software entities, this directed relationship nay house a technical debt channel. If so, the entity which invokes a potential software change is the *source* of technical debt (entity on the left), the relation which delivers the invocation corresponds to a channel *medium*, and the entity in which the potential change will take place is the *destination* of technical debt; further definition follows.

*1) Medium:* "a system with the capability of effecting or conveying something" (c.f. "medium", Merriam-Webster, 2016). In the technical debt context a medium is described through the information that is carried and through the system capable of conveying the information. The information that is carried (1) *describes changes within the source*, and (2) *indicates changes in the destination*.

Suovuo et al. [14] have argued that the medium is either explicit or implicit. An explicit system relies on pre-existing context semantics (e.g. dependency invocation). Implicit channels do not have a formal counterpart and may thus expand to areas that formality disallows, especially in relation to unions of software contexts (e.g. developer's conceptualization between a component's design documentation and its implementation). Due to their unobtrusive nature, implicit channels are difficult to observe [14].

*2) Sources and Destinations:* of a technical debt channel capture the *information producers* and *consumers* of the medium respectively. A source is an entity that exists in a context. It produces information regarding changes in the entity. The information regarding the change must be observable from outside the entity in order for the information to ever reach the medium. Hence, a valid source entity is a declaration type that can be referred. Thus, the source entity types correspond to the hosting software entity's context's referable type definitions.

Similarly, the destination exists in a context and is capable of receiving and consuming information regarding the source entity by way of being connected to it through a medium. Hence, valid channel destination entity types are the software entity's context's definitions capable of making references. The source and destination entities can exist in different contexts. The source entity can not be the destination entity as the information would be consumed where produced with no outside observable effects, deviating from the definition of a software change(s) (c.f. [13]).

### B. Information Properties

A technical debt instance has the following properties [15], [2]: a location, a principal, an interest, and an interest realization probability. The location property is directly related to the entities forming the sources and the destinations of the technical debt channels. The rest of the properties are related in the following.

*1) Principal:* A technical debt instance captures the increase in effort caused by sub-optimalities in a particular location within a software development project. The principal is the portion from the effort increase that corresponds to bringing the initial accumulation point for the difference to optimum [15], [2]. In Schmid's formalization [7] (see Section II) technical debt is accumulated when software

evolution consumes more resources than the optimal evolution would. Hence, the information carried by the technical debt channel accumulates principal, for the instance, in this entity if 1) *the software change indicates additional resource consumption* and 2) *the entity hosts the technical debt instance's initial accumulation point*.

*2) Interest:* of a technical debt instance captures the extra resources that are spent due to the principal's existence, but in entities that do not host the principal [1]. Thus, the information carried by the technical debt channel accumulates interest, for the instance, in this entity if (1) *the software change indicates extra resource consumption*, and (2) *the entity does not host the initial accumulation point of the technical debt instance*.

*3) Realization Probability:* of a technical debt instance is the chance that further resource consumption is initiated by this debt. From the perspective of propagation channels, the realization probability is a measure of an entity-entity-relationship's existence. The source entity hosts technical debt from the instance, the destination is an entity wherein currently observed resource consumption has not yet taken place, and the realization probability measure indicates the chance of this system becoming a technical debt propagation channel. By the definition of principal and interest information, if the observed realization probability is lower than one (i.e. certainty) the channel is not a technical debt propagation channel as no technical debt information is delivered yet.

## IV. Technical Debt Modelling

### A. Process

Technical debt channels describe systems for accumulating technical debt. Operationalizing such a system should hence dissipate technical debt. The software development life-cycle has multiple implementations of these systems (e.g. refactoring and -modelling) producing historical data. This can be used to identify technical debt dissipation, and be inverted in order to produce technical debt channels.

*1) Fixing the Observation Level:* of the historical software change information is a prerequisite for identifying the channels, as we must pinpoint, for each software entity, the specific pieces of change information that describe evolution solely for this entity. Formally, the observation level must provide such time and partition granularity for the change information that it allows identifying each software entity's $e \in E$ evolution as a sequence of states $e : (s_1, s_2, ..., s_n)$.

*2) Identifying Technical Debt Channels:* Observing technical debt instances' propagation, from historical data, corresponds to identifying cause-and-effect relations for the software changes observed for the entities [5]. The relations are captured for the entities' state sequences as pairs

$$r = ((e_1, s_i), (e_2, s_j)) \in R \mid (e_1, s_i) \to (e_2, s_j). \quad (1)$$

Pair $r$ indicates that entity $e_1$'s state $s_i$ has caused $s_j$ in another entity $e_2$. Further, let $d(e, s)$ be the time stamp that relates to entity $e$'s state $s$, and $D_{e_1,d_0} = \{s \mid d(e_1, s) \geq d_0\}$ be entity $e_1$'s group of states for which the time stamp is greater than or equal to $d_0$. Hence, the prerequisite for pair $r$'s causality in Eq. 1 is that $s_j \in D_{e_2,d(e_1,s_i)}$.

As Section III-A2 describes the source and destination entities of a technical debt channel as the information producer and consumer respectively, we find the components of a technical debt channel capturing $r$ as follows. The channel's source entity $e_s$ produces the information in $r = ((e_1, s_i), (e_2, s_j))$. Hence, from Eq. 1 we get $e_s = e_1$. Analogously for the destination, $e_d = e_2$. Last, the channel's medium is described as carried information and thus corresponds to the information realizing $(e_s, s_i) \to (e_d, s_j)$.

Section III-B describes the properties for information that corresponds to technical debt propagation. In associating the information to entities, presence of these properties should be ensured in order to only capture technical debt propagation channels (not e.g. change propagation caused by feature addition efforts).

Finally, it is evident that technical debt can exist without related software changes. If an entity is created with principal for a new technical debt instance, no changes record alterations for this debt. Hence, arguably, identification of technical debt propagation channels requires historical data as it alone can record how the debt has realized.

*3) Abstracting Channels to Models:* corresponds to identifying a class $T$ of technical debt channels $t$ which have identical propagation capabilities $\mathbf{P}_T$, and abstracting this class to form a model $M$. Technical debt channel $t$ has a source $source_t = type(e_s)$, a destination $dest_t = type(e_d)$, and an information type $info_t = type((e_s, s_i) \to (e_d, s_j))$ which capture its propagation capabilities $\mathbf{P}_t = (source_t,\ dest_t,\ info_t)$. Hence, $t \in T \iff \mathbf{P}_t = \mathbf{P}_T$. For the software entities, the type was their context dependent—referable or referring—type definition while the content is the information type. Observation level fixing ensures that the observed types adhere to these requirements.

Abstracting the model corresponds to removing all implementation specific details $\theta$ (e.g. names of specific methods) from the technical debt channels forming a class (i.e. $\forall t \in T$) to make the model applicable for all scenarios where the observed propagation capabilities $\mathbf{P}_T$ are identical. Hence, the abstraction of $M$ corresponds to a reduction: $T \mapsto_\theta M$.

### B. Applying the Process

We provide initial validation for the technical debt modelling process described in Section IV-A by applying it to a technical debt instance: A bug from the Eclipse IDE (#73950 examined in [5]). According to the process description in Section IV-A, the first phase is observation level fixing. For the bug, we identify historical data and software entities via the bug report (c.f. https://bugs.eclipse.org/bugs/show_bug.cgi?id=73950) and the corresponding fix commit (c.f. https://git.eclipse.org/c/platform/eclipse.platform.debug.git/

```
@@ -344,4 +344,8 @@ public class AddMemoryBlockAction
extends Action implements IselectionListener
...
+
+   protected void dispose()
+     DebugPlugin.getDefault().removeDebugEventListener(this);
+   }

@@ -75,7 +75,7 @@ public class MemoryView
extends PageBookView implements IDebugView, IMemoryBlock
    private TabFolder emptyTabFolder;
    protected Hashtable tabFolderHashtable;

-   private Action addMemoryBlockAction;
+   private AddMemoryBlockAction addMemoryBlockAction;
    private Action removeMemoryBlockAction;
    private Action resetMemoryBlockAction;
    private Action copyViewToClipboardAction;

@@ -621,6 +621,7 @@ public class MemoryView
extends PageBookView implements IDebugView, IMemoryBlock
    public void dispose() {
      removeListeners();
+     addMemoryBlockAction.dispose();

      // dispose empty folders
      emptyTabFolder.dispose();
```

(1) 2 3 4 (5) (6)

Figure 2. Transcript from an Eclipse version commit, demonstrating implicit and explicit technical debt propagation channels with directions

commit?id=9d0372b5e5159743ef53b2ec0ddaf1bfbb58a0ce). The commit describes changes at the source code level, and this allows us to observe evolution sequences $e$ at the level of single software entities.

The second phase identifies technical debt channels. Backtracking the dissipation, the iterative process of finding producers and consumers should stop when software entities that produce the information about changes which overcome the root cause, the principal of the instance, are found. For the bug in question, we associate the bug report's call for disposing `MemoryBlockAction` properly into changes in the fixing commit. Figure 2 is a transcript of the fixing commit. Lines in green and starting with a plus sign indicate addition, while the ones in red and starting with a minus sign indicate deletion. Numbered arrows indicate identified technical debt propagation channels—forming the propagation path for a technical debt instance—while the arrow colors indicate classes of channels with possibly similar propagation capabilities.

The Java context (c.f. https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf) applies for technical debt channel four (4). This is a pair $r = ((e_s, s_i), (e_d, s_j))$ where for source entity $e_s$ `addMemoryBlockAction.dispose()` the type $source_t = type(e_s)$ is a *method invocation*. The state $s_i$ *statement creation* is likely the first for $e_s$, and it has invoked another *statement creation* state $s_j$ for the destination entity $e_d$ `dispose()`, whose type $dest_t = type(e_d)$ is a *method declaration*. The information type $info_t = type((e_s, s_i) \rightarrow (e_d, s_j))$ is *invocation of a non-existent method declaration* as the method is created in the commit. Channels from one (1) to three (3) are implicit channels in Figure 2, and manual analysis is required to indicate these relationships [14]. In particular, the commit transcript cannot be solely used to decide $e_s$ and $e_d$ for channel three (3).

Table I. A Technical Debt Model

| Part | Definition |
|---|---|
| Source entity | Method Invocation |
| Destination entity | MethodDeclaration |
| Information | Invocation of a non-existent method declaration |

The third phase of the process identifies a class of channels to abstract into a technical debt model. If we consider the channel four (4) to be the sole representative for its class, the abstraction results to a technical debt propagation model displayed in Table I (where the context removal $\theta$ disregards naming for $e_s$ and $e_d$).

We may review the information properties for the captured model (see Table I). The common property for technical debt channel information required that the software change indicates additional resource consumption. The *information* of the model adheres to this as the implementation of a method declaration is indicated. The unique property of the information described if it accumulated either principal or interest for a technical debt instance. This required identifying if the additional resource consumption occurred in an entity that hosted the instance's root cause. The model's instances, the unique technical debt propagation channels, must be consulted for this. An argument for interest accumulation can be made for channel four (in Fig. 2), if we interpret channels one through three with their entities to precede it in the instance's propagation.

## V. Discussion

### A. Strengths and Implications

The most important strength of the proposed approach is the accumulated library of technical debt propagation channel classes. These models can be easily applied to estimate the technical debt propagation capabilities of new projects (i.e. we may assess models like the one in Table I for newly encountered similar components). This allows the project to: (1) expose possible propagation paths for newly developed entities by relating them to known source types, (2) provide enhanced explanation for problem targets by relating the target entities to known destination types, and (3) expose gaps in project communication by way of demonstrating the possible ways of propagation between project entities as the known information types. These strengths directly contribute to ongoing research efforts (c.f. [15], [2]), and has potential implication for practice.

The models also expose an interface that allows programmatic evaluation of the representations; especially important from the perspective of automating information maintenance for constantly evolving projects. As models derived from the explicit channels capture technical debt propagation in contexts where the semantics are known, their evaluation can be implemented by means of static program analysis. For implicit channels, while the semantics can be unattainable

and thus posing a challenge to full automation, the proposed approach collects the possible source and destination types which should allow for programmatic identification of their instances. Automation would arguably increase the effectiveness of technical debt management frameworks [15], [2], and pave way for more established evaluation methods [3].

Lastly, there is no foreseeable obstacle to associating the models with value production (e.g. return-on-investment for expedited reparation of instances of the model in Table I). However, to associate the model with a cost value, the historical data needs to include decomposable value information (i.e. refactoring effort).

### B. Potential Challenges

Firstly, determining directions for, especially implicit, technical debt propagation channels can be difficult. As they are directed by definition, it is possible to model them from both directions (e.g. channel tree (3) in Fig. 2). Second, the identification of classes as channels is based on type libraries. Given that the amount of type defining contexts is remarkable, the amount of possible channel classes is numerous. To overcome this, arguably, a hierarchical channel taxonomy is required where the grouping dimensions exploit pre-existing taxonomies.

Third, two challenges relate to analyzing historical data to produce technical debt channels. Firstly, channel identification relies on distinguishing technical debt inclined change from the decomposed information. While the formal description of technical debt provides a basis for this, practical identification can be seen to rely on relating items to previously described instances of technical debt which is not exhaustive. Examination of common change inducers could be a partial solution to this [14]. Second, the channels can only be constructed from where historical data captures technical debt dissipation. Hence, there can be channels that accumulate debt, but for which no data exist or the debt is never acted upon. The latter is arguably almost invisible to the software project, but the former should be captured. Tracking of software projects' efficiency and addition of suitable documentation procedures to capture the missing evolution characteristics are avenues for pursuing this.

Finally, whilst two approaches [7], [6] addressed technical debt propagation, we note that neither capture the various forms and ways of technical debt propagation; focusing rather on the propagation's characteristics and capabilities. This lack of differing approaches to technical debt modelling is a challenge, as it hinders providing comparisons.

## VI. Conclusion

This paper provided a theoretical description for technical debt channels as information mediums with producers and consumers. It also presented an approach for capturing technical debt channels, identifying classes of channels, and abstracting them into propagation models. In addition to advancing the technical debt research with theoretical basis for technical debt accumulation, the proposed method should deliver programmatically assessible models for automating the maintenance of manually identified technical debt information.

Future work includes exploring mechanisms for identifying taxonomies of technical debt information producers and consumers. Such mechanisms would facilitate the production of an accurate technical debt channel classification scheme. A direct application of the scheme is the identification of overlooked technical debt management areas, and indication of enhancements for existing management solutions.

## References


[1] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya *et al.*, "Managing technical debt in software-reliant systems," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*. ACM, 2010, pp. 47–52.

[2] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. Da Silva, A. Santos, and C. Siebra, "Tracking technical debt - an exploratory case study," in *27th IEEE International Conference on Software Maintenance*. IEEE, 2011, pp. 528–531.

[3] C. Seaman, Y. Guo, C. Izurieta, Y. Cai, N. Zazworka, F. Shull, and A. Vetrò, "Using technical debt data in decision making: Potential decision approaches," in *3rd International Workshop on Managing Technical Debt*. IEEE, 2012, pp. 45–48.

[4] J. Holvitie, V. Leppänen, and S. Hyrynsalmi, "Technical debt and the effect of agile software development practices on it-an industry practitioner survey," in *Sixth International Workshop on Managing Technical Debt*. IEEE, 2014, pp. 35–42.

[5] J. Holvitie and V. Leppänen, "Examining technical debt accumulation in software implementations," *International Journal of Software Engineering and Its Applications*, vol. 9, no. 6, pp. 109–124, 2015.

[6] J. McGregor, J. Monteith, and J. Zhang, "Technical debt aggregation in ecosystems," in *3rd International Workshop on Managing Technical Debt*. IEEE, 2012, pp. 27–30.

[7] K. Schmid, "A formal approach to technical debt decision making," in *Proceedings of the 9th International ACM SIGSOFT Conference on Quality of Software Architectures*. ACM, 2013, pp. 153–162.

[8] N. Zazworka, R. O. Spínola, A. Vetro, F. Shull, and C. Seaman, "A case study on effectively identifying technical debt," in *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2013, pp. 42–47.

[9] R. J. Eisenberg, "A threshold based approach to technical debt," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 2, pp. 1–6, 2012.

[10] D. Falessi, M. Shaw, F. Shull, K. Mullen, M. S. Keymind *et al.*, "Practical considerations, challenges, and requirements of tool-support for managing technical debt," in *Managing Technical Debt, 2013 4th International Workshop on*. IEEE, 2013, pp. 16–19.

[11] S. Kim, E. J. Whitehead Jr, and Y. Zhang, "Classifying software changes: Clean or buggy?" *Software Engineering, IEEE Transactions on*, vol. 34, no. 2, pp. 181–196, 2008.

[12] A. Rochd, M. Zrikem, A. Ayadi, T. Millan, C. Percebois, and C. Baron, "Synchspem: A synchronization metamodel between activities and products within a spem-based software development process," in *Computer Applications and Industrial Electronics, 2011 IEEE International Conference on*. IEEE, 2011, pp. 471–476.

[13] H. Kagdi, M. L. Collard, and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 19, no. 2, pp. 77–131, 2007.

[14] T. Suovuo, J. Holvitie, J. Smed, and V. Leppänen, "Mining knowledge on technical debt propagation," in *14th Symposium on Programming Languages and Software Tools*. CEUR-WP, 2015.

[15] C. Seaman and Y. Guo, "Measuring and monitoring technical debt," *Advances in Computers*, vol. 82, pp. 25–46, 2011.