Technical Debt Research Roadmap

An outcome of the

Dagstuhl Seminar on Technical Debt Research

April 17-22, 2016

At the end of a week of discussions about many aspects of technical debt (TD) research (past, ongoing, and envisioned), the attendees of the Dagstuhl workshop spent a morning sharing ideas about the most important TD-related research problems for the community to work on. The discussion of a research roadmap was grounded in the definition of technical debt formulated during the seminar:

> *In software-intensive systems, **technical debt** is a design or implementation construct that is expedient in the short term, but sets up a technical context that can make a future change more costly or impossible. Technical debt is a contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability.*

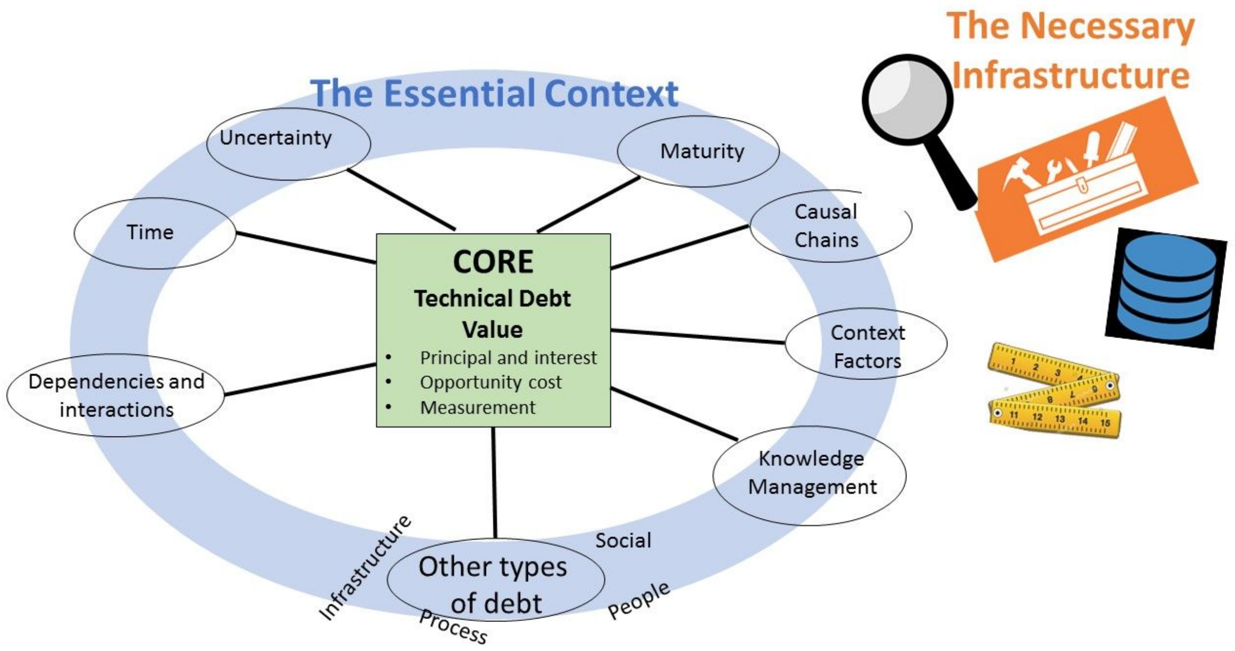
The group spent some time envisioning how the world would be different if all our research efforts in this area were successful. This vision included the following points:

- TD would be managed as carefully as we currently manage defects and new features.
- We would have a clear, operational definition of "good enough" software, including how much TD would be acceptable.
- We would have a way to translate between developer concerns and manager concerns, and this translation would form a basis for making decisions about allocating effort to various tasks.
- TD would be incurred intentionally most of the time.
- Projects that manage TD would be more efficient and effective and sustainable than projects that don't.
- The notion that upfront architectural work (vs. emergent architecture) is worth it would be well supported and accepted.
- There would be tools to support all aspects of TD management that are adopted and used by all stakeholders.
- TD-aware development (practices and tools) would be an accepted standard way of producing software.
- Architectural assessment would be part of standard development policy.

Achieving this vision requires changing development practice through effective communication between research and practice and, most importantly for the research community, showing effectiveness of proposed approaches. The research community must also recognize that our practitioner audience consists of two different groups: managers who make decisions about spending money on eliminating TD, and developers who write and maintain code. Proposed solutions must ultimately appeal to both, but researchers should also be clear about which group is primarily targeted with a new approach or tool.

The group discussed research activities in three broad areas:

1. The Core: defining, understanding, and operationalizing the concept of value with respect to TD,
2. The Essential Context: understanding phenomena that fall outside the core definition of TD, and that have an essential relationship with how TD plays out in practice, and
3. The Necessary Infrastructure: building the shared infrastructure that facilitates all our research activities.



These three areas are depicted in the figure above and outlined in the sections below.

The Core: understanding value

The concept of value and what it means with respect to TD came up often and in many contexts throughout the seminar. It's not clear that we all mean the same thing when we talk about value, but it is clear that it is central to delivering effective mechanisms for managing TD in practice. Value comes into play both when making decisions about whether or not to incur TD (i.e. the value of a proposed TD item) as well as when deciding whether, when, and which TD to pay off (i.e. the value of eliminating an existing TD item). At one level, the notions of principal and interest serve to capture the short-term benefits (principal) and long-term costs (interest) of incurring TD, as well as the cost/benefit analysis related to paying it off. But this does not seem to be adequate to capture all aspects of value.

One aspect that does not seem to be easily captured by principal and interest is the idea of opportunity costs. That is, one of the common benefits of incurring TD is the ability to take advantage of an opportunity (because resources are freed up) that might not be available at another time. In theory, this benefit is part of the principal of the TD being considered, but principal does not capture the time element (the fact that the opportunity is time-sensitive) nor does it allow for the fact that the resources freed by incurring TD and the resources needed to take advantage of the opportunity might not be comparable.

Another aspect of value that is even harder to make concrete is the value of TD management, and TD-related information, to the quality of decision making. Capturing the “quality” of decisions is difficult to begin with, but somehow demonstrating the benefits of considering TD in management decisions is a key area for TD researchers. A related question is that of defining “good enough” software, i.e. software that has a level of quality (and a level of TD) that cannot be cost-effectively improved. Helping managers define “good enough” in their context would be extremely useful, and better defining the value of TD is an important step in this process.

The quantification of value (and other TD properties) has always been an important aspect of TD research, and was part of the research agenda set out in the first TD research roadmap in 2010 (Brown et al., 2010). Efforts to quantify principal and interest, and to somehow combine them into a measure of “value” have proved difficult, but continued work in this area is crucial to furthering our vision of effective TD management in practice. Useful measures would both be at a level of granularity that is useful to developers in tracking and monitoring activities, as well as be understandable in a business sense.

Steps towards providing effective operationalizations of value include proposing proxies for value, collecting data based on these proxies, designing validation procedures (i.e. study designs) to show that these measures are meaningful and useful, and conducting case studies to carry out these validations. Case studies can also be used to explore existing notions of value in practice, which would ground the choice of proposed metrics. Such exploratory case studies could also be used to discover “grass roots” approaches to TD management, i.e. environments without an explicit strategy to manage TD, but where, out of necessity, techniques have emerged to deal with the most important aspects.

The Essential Context: understanding related phenomena

When the Dagstuhl group agreed on our definition of TD (see above), we also recognized that the definition does not encompass all topics that are important to study. TD exists in a context, and exists in a variety of forms.

The TD definition limits TD to phenomena closely tied to source code (e.g. design debt, code debt, architecture debt), which leaves out other important types of “debt” that are at least partially analogous to financial debt, e.g. Social Debt, People Debt, Process Debt, and Infrastructure Debt. These other types of debt are related to TD in various ways. Some are part of the causal chain leading to TD, i.e. the impacts of these other types of debt can lead to TD as defined. Others appear to co-occur with TD, or create an environment that affects the ability to manage TD in some way.

Other aspects of TD context that need to be studied within the TD research agenda include:

- Uncertainty – building representations of uncertainty into TD value measures and using that uncertainty information in decision making
- Development and organizational context – studying what aspects of context affect how TD is most effectively managed
- Time – better accounting for the passage of time (not necessarily measured in time units, but in relevant events, such as code changes) in all aspects of TD management, e.g. the value of TD over time, the effect of time-based events on value, etc.)

- Dependencies and interactions – between TD items, between TD items and development artifacts
- Knowledge management – what TD-related information needs to be captured and disseminated for effective TD decision making?
- Causal chains – what is the constellation of "causes" leading up to the creation of TD?
- TD maturity – the creation of a maturity model that depicts the different levels at which TD could be managed, or possibly incorporating TD concerns into existing maturity models

The Necessary Infrastructure: data, tools, etc.

One of the advantages of forming a coordinated, active research community in a particular area of study is the ability to share resources. The group at Dagstuhl identified a number of resources that could be developed by members of the community and contributed to the TD research infrastructure to enhance the level of progress of the entire field. Identifying relevant pieces of infrastructure is not possible until a community reaches a certain level of maturity. We feel that the TD research community has reached that level and that identifying and building common infrastructure would now be useful.

- **Datasets** – OSS projects provide some useful data, but they generally do not include effort data, which is essential to addressing most relevant questions in TD research. "Data" provided with published studies is often really the analysis of the raw data, not the data itself.
- **Benchmarks** – ground truth in this area is difficult to come by, but we can strive for inter-subjectivity, i.e. widespread agreement on an essentially subjective proposition. This can apply to, for example, tools that detect code smells or calculate code-based quality measures. Designating certain tools as "reference" implementations of these functions could also be useful.
- **Common metrics** for outcome variables (maintenance effort, defects, etc.)
- **Tools** – pluggable, validated, benchmarked – also recognize that tools have to implement an underlying, understandable process, as some contexts will not allow for using a specific tool
- **Infrastructure for replication** as well as for new work, so that it can be usefully compared to existing work

References:

Nanette Brown, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan MacCormack, Robert L. Nord, Ipek Ozkaya, Raghvinder S. Sangwan, Carolyn B. Seaman, Kevin J. Sullivan, Nico Zazworka: Managing technical debt in software-reliant systems. FoSER 2010: 47-52