Technical Debt Item (TDI) Classification Guidance

Purpose: The goal of this classification scheme is to help identify existing technical debt issues in a project as they are described by the software development team in their issue trackers. Use this classification guidance to identify technical debt items, as opposed to items that are new features, user stories, common tasks such as documentation, new requirements, defects, and the like. The purpose of such an identification exercise is to separate technical debt items from other issues, not to analyze the technical debt items to determine whether the proposed fix is good or bad, the impact of the proposed fix, or other concerns.

General comments: Technical debt is not necessarily bad. Technical debt includes not only shortcuts taken for expediency or deferred code cleanup and refactoring but also technical evolutionary drift (a "technological gap" in the technical debt landscape). Try to use a forward-looking lens as you review the issues, and stay focused on the future impact of technical debt on executable artifacts. For example, a policy issue (cause) may have resulted in a poor design decision. We would be more concerned with the system component (executable artifact) affected by that design decision than with the policy issue that caused it. Try not to interpret beyond what is written in the issue description.

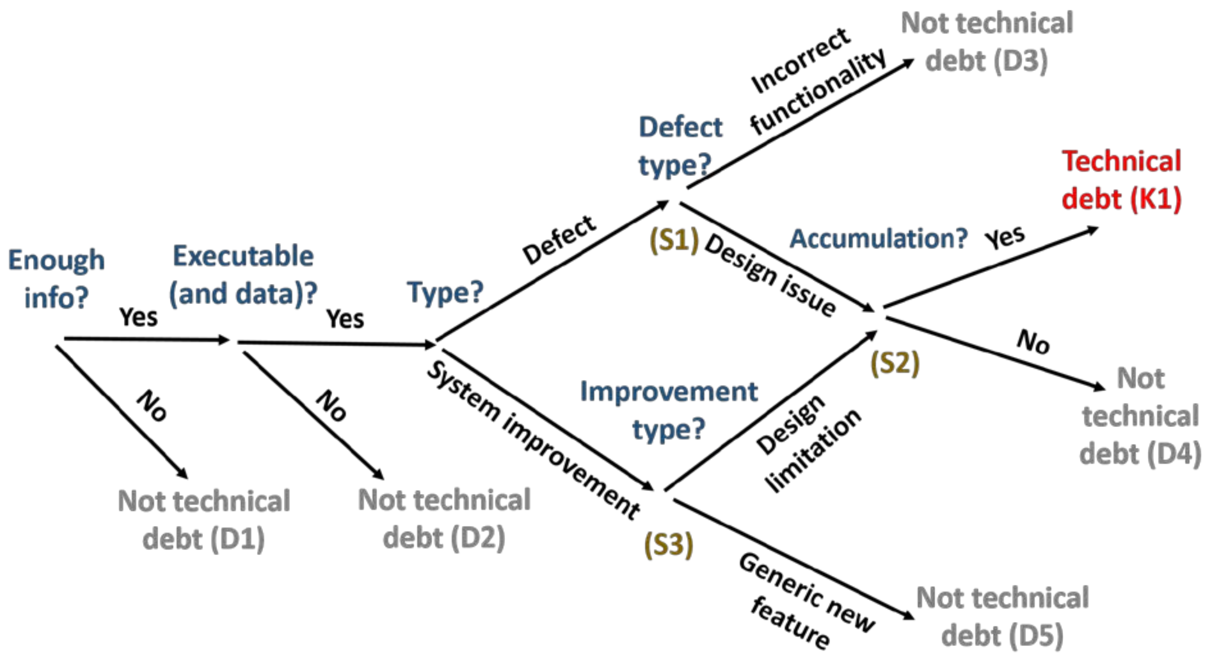

Figure 1. TDI Decision Tree

Terms

Enough info: At any point in the decision tree, do you lack enough information to make a decision? If so, exit at D1.

Executable and/or data: An *executable* system artifact identifies development artifacts that compose the running system (e.g., code, data, and build scripts as opposed to the software requirements specification document or the team wiki). Is an executable artifact identified as requiring alteration in some way? Keep executable artifacts such as code, automated tests, build scripts, and models. Discard non-executable artifacts such as process issues, policy issues, documentation, and requirements specifications. *Data* means that data problems are technical debt (e.g., poorly structured database record content, duplicate records, data model design issues).

Incorrect functionality: If the issue is a defect, is it incorrect functionality (e.g., a button does not work in the UI), a non-functional defect (e.g., lack of conformance to specifications), or an implementation mistake?

Design issue: An issue may be characterized as a design issue if it meets the following criteria:

- mentions a defect that limits correct functioning of the system
- has implications for a quality attribute (e.g., availability, security, performance, modifiability)
- includes cleanup activities such as removing duplicate code, two versions (functionality, components, libraries, etc.), non-standard binding (assumes duplicating "standard"), type mismatch, two source code structures, different implementations, unused classes (dead code), etc.

Design limitation: If the issue is a system improvement, is it a design limitation that causes accumulation (e.g., inability to add a new feature quickly, the current technology not supporting the improvement needed), and/or is the remediation likely to involve adding or modifying a design pattern, reference architecture, or runway component?

Accumulation: Evidence of accumulation may include, but is not limited to, reoccurrence of the issue, increased time or effort to make implementation changes, problems with artifact alignment (automated tests are not supporting the refactored classes), increased risk (e.g., security vulnerability), and increased time to find the root cause of a problem. Another characteristic of accumulation is that if the problem is not fixed, effort to maintain a component or part of the system will grow, or the cost to fix it later will be significantly greater than fixing it now. There are two dimensions of accumulation:

- anticipated cost of deferring the decision (in other words, future pain caused by a decision)
- remediation, which is <u>out of scope for this study</u>

Process Steps for Experiment

Precondition: Provided a set of issue records taken from an issue tracker

For each issue:

1. Review the issue descriptive text (title, description, resolution, etc.) for the issue.

2. Use the decision tree, general guidelines, and term descriptions to walk through the tree and either Discard or Keep each technical debt item in the issue data set.

3. Capture the reason (e.g., D2) in the designated column in the spreadsheet provided.

4. Add comments/rationale in the designated column in the spreadsheet. Consider these questions as you reflect.

    - Did you have enough information at each step to make a decision? If not, where did you run into a problem, and what was missing?
    - Were the decision points and term descriptions clear enough to make a decision? Why or why not?

Repeat for the next issue.

Post-experiment Questionnaire

1. Was the data adequate to make decisions?

2. Was the decision tree generally easy or hard to use, and why or why not?

3. What terms need further clarification?

4. Do you think the decision tree produced an accurate result? Why or why not?

5. What might you improve?

Additional Resources

This material supplements the following research paper: Bellomo, S.; Nord, R.; Ozkaya, I.; & Popeck, M. “Got Technical Debt? Surfacing Elusive Technical Debt in Issue Trackers,” in *Proc. of Mining Software Repositories ICSE 2016*.

This study is part of a wider SEI effort on technical debt, including an ongoing research effort. Learn more about our work at http://www.sei.cmu.edu/architecture/research/arch_tech_debt/index.cfm.