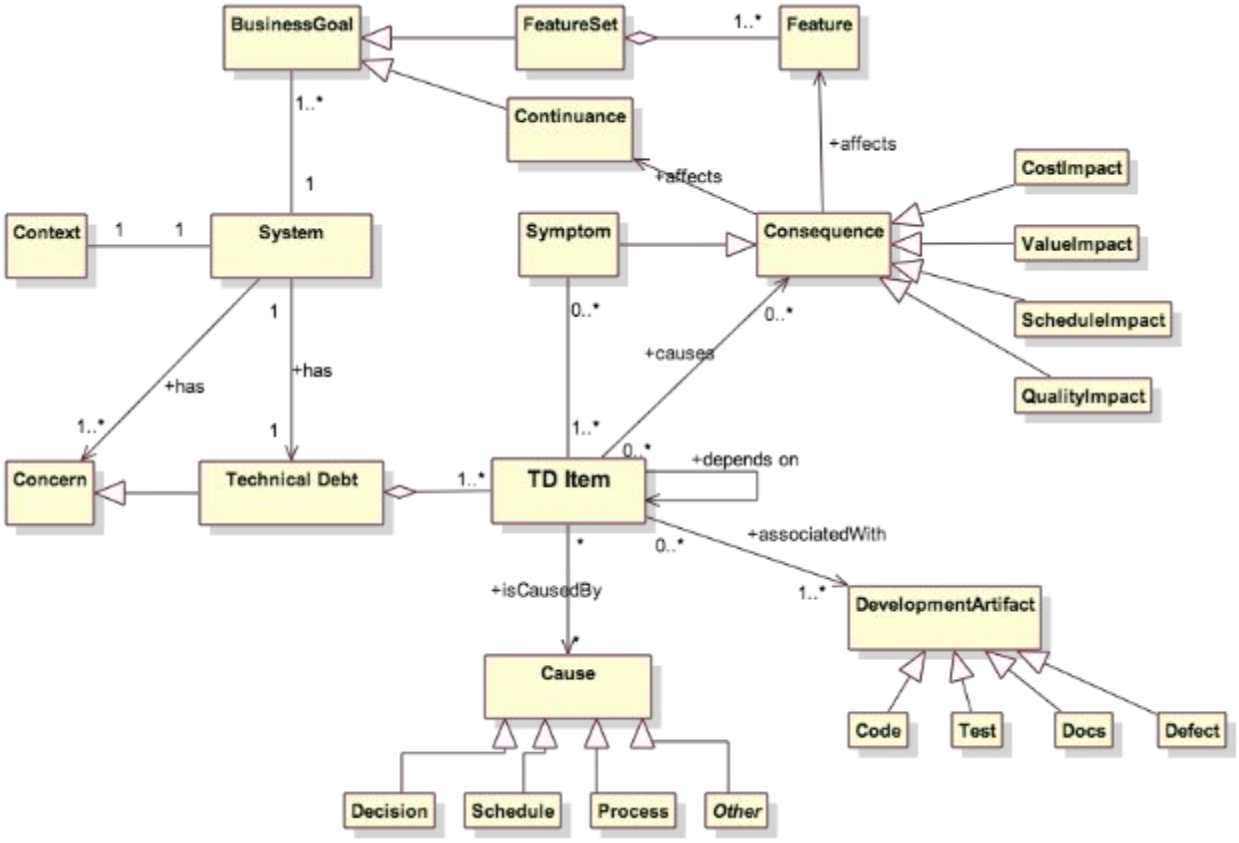
A conceptual model of technical debt

Ph. Kruchten, I. Ozkaya, R. Nord
January 2015

This document describe a simple conceptual model of technical debt in software-intensive systems, and its relationship with related concepts.

A simple UML model



The *technical debt* associated to a software-intensive system is composed of a set of *technical debt items* (or TD items) and this technical debt is one of the many *concerns* associated with the system.

Technical debt items have both causes and consequences. The *causes* of technical debt can be a process, a decision, an action (or lack of action) or an event that triggers the existence of that debt item; schedule pressure, unavailability of a key person, lack of information about a technical feature, etc.

The *consequences* of a technical debt items are many: it ha effect on the value of the system, on the costs, past present and future, directly or through schedule or future loss of quality. The business objectives of the sponsoring organization developing or maintaining the software system are affected in two ways: either through a delay or loss of quality of some of the features of the system, or difficulties in maintaining the system operational (continuance).

A technical debt item is associated with one or more concrete, tangible artifacts of the software development process, primarily the *code*, but also to a certain extent the *documentation,* the known *defects*, and the *tests* associated with the system.

To keep with the financial metaphor, the cost impact of technical debt can be seen as composed of a *principal* and *interests*. The principal is the cost savings gained by taking some initial approach or shortcut in development (the initial principal, often the initial *benefit*), or the cost that it would take now to develop a different or better solution (the current principal).

The *interest* are costs that add up as time passes by. There are *recurring interests*: additional cost incurred by the project in the presence of technical debt, due to reduced velocity (or productivity), induced defects, and loss of quality (maintainability ifs affected). And there are also *accruing interests*: the additional cost of the developing new software depending on “not quite right code” (evolvability is affected).

For further reading

The notion of concerns associated with a system comes from ISO 42010-2015. Li, Liang and Avgeriou (2014) have proposed an alternative conceptual model, where they separate benefits and costs of a technical debt item.

References

Li, Z., Liang, P., & Avgeriou, P. (2014). Architectural Debt Management in Value-Oriented Architecting. In I. Mistrik, R. Bahsoon, R. Kazman, & Y. Zhang (Eds.), *Economics-Driven Software Architecture*, (pp. 183-204). Amsterdam: Elsevier.