An Exploration of Agile Project Management & Technical Debt

TREO Talk Paper

Christopher Sibona
University of North Carolina Wilmington, Wilmington, NC
sibonac@uncw.edu

Abstract

Concerns about technical debt have risen with the adoption of agile project management. The relationship between technical debt and agile techniques is still unclear (Kruchten, 2012). Cunningham (1992) describes technical debt as immature code that works fine and is acceptable to the customer but needs to be rewritten before negative outcomes will occur. The negative outcomes of technical debt occur in two major areas in software development; issues of evolvability and maintainability (Kruchten, 2012). Kruchten notes technical debt can be visible (e.g., additional time to generate new features, the number of defects, low external quality) and invisible (e.g., low internal quality, documentation debt, architectural debt). Some debt will be measurable and quantifiable where other debt is hard to assess and qualitative, e.g. software engineers will discuss code smells for code that is not quite right.

Many methods fall under the agile project management umbrella, including Scrum and Extreme Programming (XP). Agile methods generally follow an iterative and incremental approach to product development where software is frequently delivered (Beck, 2001). Scrum has little to no effect the engineering of the software code directly but is more project management focused. XP includes both engineering practices and management practices and attempts to address technical debt through refactoring, specifically refactoring mercilessly, the application code. The extent to which agile methods discourage or encourage debt is unclear (Holvitie, 2014).

Technical debt is in the software code, and software engineers are responsible for generating the technical debt. But technical debt is both a software problem and a management problem. Management and engineering are responsible for management of the debt.

How might technical debt concerns be addressed from the perspective that management and engineering share responsibility? What would technical debt look like from an ideal perspective? Who drives technical debt, the software engineers, management, both? Technical debt has some positive aspects like the ability to speed-up the short-term delivery of working software and negative aspects slowing down the longer-term delivery, or susceptibility to bugs. What is the role of engineering, who creates the debt, in the elimination of the debt? How can management be informed of both the quantifiable concerns and the qualitative concerns of technical debt?

References


