Why Should Government Care About Technical Debt and Software Architecture?

Ipek Ozkaya
(ozkaya@sei.cmu.edu)

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Copyright 2013 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

This material has been approved for public release and unlimited distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM-0000765
Objective

Understand what technical debt is
Provide a different perspective on software development and architecture through managing technical debt
Support for Delivery Over Time

Projects should not simply produce a product design; they should plan a desired state that enables teams to quickly deliver releases that stakeholders value (or in terms of lean practices, design a profitable operational value stream for rapidly delivering that product).

Technical Debt*

A design or construction approach that's expedient in the short term but that creates a technical context in which the same work will cost more to do later than it would cost to do now (including increased cost over time)  

S. McConnell

Some examples include:

- Continuing to build on a foundation of poor quality legacy code
- Prototype that turns into production code
- Increasing use of "bad patches", which increases number of related systems that must be changed in parallel

## Technical Debt – Steve McConnell

<table>
<thead>
<tr>
<th>Type 1</th>
<th>Type 2</th>
</tr>
</thead>
</table>
| unintentional, non-strategic; poor design decisions, poor coding | intentional and strategic: optimize for the present, not for the future.  
2. A short-term: paid off quickly (refactorings, etc.)  
2. B long-term |

Implemented features (visible and invisible) = assets = non-debt

---

Technical Debt – Jim Highsmith

- Only on far right of curve, all choices are hard
- If nothing is done, it just gets worse
- In applications with high technical debt estimating is nearly impossible

Technical Debt Analogy

When and how was the debt signed under?
What is the interest rate?
What is the payback strategy?
Taking on Debt

First more capabilities

Then, more infrastructure

First more infrastructure

Then, more capabilities

Standard iteration management in agile development ➞ functional, high-priority stories allocated first.

Tracking and monitoring mechanism is solely based on customer features delivered.
Standard iteration management in architecture-centric development processes

→ up-front requirements and design tasks allocated first.

No explicit and early tracking and monitoring mechanisms that is development artifact specific.
Only Three Strategies

Do nothing, it gets worse

Replace, high cost/risk

Incremental refactoring, commitment to invest
Tactics to consider

Align feature and system decomposition.

Create an architectural runway.

Use matrix teams and architecture.
Why Should Government Care About Technical Debt and Software Architecture?

Practical Approaches from the Ground

Warren Ellmore
Technical Debt is good (as long as it’s managed)

• Technical Debt is essentially the result of trade-offs - deferred decisions, deferred priorities, deferred capabilities, deferred skills.

• Technical Debt arises when current sprint work is unblocked by a decision on what can be implemented now and what can be deferred.

Example: You know you need security but decide to defer that for a later sprint so that functional capability can continue to be developed/implemented.

Example: You need to interface with a legacy system but that API isn’t ready yet so you decide to hack a quick stub just for now.

Government Context: Unfortunately, Technical Debt is often misunderstood by business owners and frequently ignored in favor of business functionality. Managing and resolving Technical Debt is often more difficult because of contract terms, size and complexity, and a general lack of skills and experience in risk management as it relates to Agile development.
Technical Debt can be Managed

• Requires more than just logging a “ToDo” or adding to the backlog
• All parties must be involved and have insight – PMO responsibility
• Establish and refine an understanding of:
  • Scope of impact and the accumulated risk curve
  • “Value” and priority within the release strategy
  • Dependencies of scheduled functionality on resolution (partial or full)
  • Ongoing decisions can ease or exacerbate a particular debt
• Choose Technologies, Architectures and Frameworks that meet the business/mission requirements and minimize Technical Debt impact
  • Available skills, known technologies vs. new “shiny objects”
  • Leverage componentization – separation of concerns, service architecture
  • Reuse existing capability – services, components, models, patterns, specifications …
Technical Debt can be addressed in Sprints

- Plan for periodic refactoring sprints
- Run parallel Architecture/Technical Capability sprints
- Run parallel Integration sprints targeting releases
- Start running functional/performance testing asap and scale with codebase
Architecture can Reduce the Accumulation and Impact of Technical Debt

• Aim for a “Fuller-stack” Service Architecture
  • Provides isolation reducing change impact scope
  • Provides abstraction for new/untested technology
  • Provides for asset/capability reuse and extension

• Leverage Architectural Models
  • Impact analysis, traceability, knowledge management
  • More easily identify separation points for dividing the work across multiple teams/contracts/providers

• Evolve to Model-Driven Architecture/Development
  • Business Process Orchestration
  • Code generation, injectable architectural framework
Key Takeaways

• Technical debt is unavoidable and can be good – if managed
  • Make architecture features and technical debt visible.

• Plan for its resolution – increase for new/unknown
  • Different kinds of technical debt call for different approaches, e.g. new technology versus low code-quality

• Bridge the gap between the business and technical sides.
  • Associate technical debt with risk.

• Reduce technical debt impact with architecture – sufficient upfront, add capability in parallel sprints.

• Discover unseen technical debt as early as possible by starting continuous integration and system testing following Sprint 1
  • Integrate technical debt into planning and standard operating procedures (e.g., planning, reviews, retrospectives).