Paying Down Your Technical Debt: A Less Insane Approach

Michael Hofmockel
LAS Research IT
Biology IT
hof@iastate.edu

WebSIG
March 26, 2015
Hello

Polling the demographics of the audience.

• Name?

• Role?

• Biggest technical debt right now?
It’s just one band-aid

401(K) 2012, “Doctors Fees”, flickr.com
All the band-aids

Rosa Luxemburg-Stiftung, Band-aids, flickr.com
Technical Debt

Eventual consequences of poor system design

“Technical Debt” coined by Ward Cunningham while refactoring the WyCash financial product. He used this metaphor to help explain the burden to client.
Causes

- **Business pressures**, where the business considers getting something released sooner before all of the necessary changes are complete, builds up technical debt comprising those uncompleted changes.

- **Lack of process or understanding**, where businesses are blind to the concept of technical debt, and make decisions without considering the implications.

- **Lack of building loosely coupled components**, where functions are not modular, the software is not flexible enough to adapt to changes in business needs.

- **Lack of test suite**, which encourages quick and risky band-aids to fix bugs.

- **Lack of documentation**, where code is created without necessary supporting documentation. That work to create the supporting documentation represents a debt that must be paid.

- **Lack of collaboration**, where knowledge isn’t shared around the organization and business efficiency suffers, or junior developers are not properly mentored

- **Parallel development** at the same time on two or more branches can cause the buildup of technical debt because of the work that will eventually be required to merge the changes into a single source base. The more changes that are done in isolation, the more debt that is piled up.

- **Delayed refactoring** – As the requirements for a project evolve, it may become clear that parts of the code have become unwieldy and must be refactored in order to support future requirements. The longer that refactoring is delayed, and the more code is written to use the current form, the more debt that piles up that must be paid at the time the refactoring is finally done.

- **Lack of alignment to standards**, where industry standard features, frameworks, technologies are ignored. Eventually, integration with standards will come, doing sooner will cost less (similar to ‘delayed refactoring’).

- **Lack of knowledge**, when the developer simply doesn’t know how to write *elegant* code.

Understanding changes

Knowing when to refactor and doing so at that moment is the greatest skill in avoiding technical debt.

Michael Heiss, “Management of Complexity”, flickr.com
Cost of Software

Most of the cost of software ownership arise after delivery.

• Fixes
• Changed to meet new needs
• Improved in performance or maintainability
• …
The National Science Foundation approved a $465,000 research grant in 2009 on technical debt, and research firm Gartner has just released its study on this subject, which it calls "IT debt."

Gartner put the IT debt bill at $500 billion worldwide and says it will double in five years to $1 trillion.
Cost per line of code


```php

```
For the Cost Geeks

Manny Lehman
Father of Software Evolution

Wrote many papers from the mid 70s onwards, proposing “Laws of Software Evolution” for “E-type systems”.

Systems classified into:

• S-type: formally specified and verified; static by definition

• E-type: real-world system
Lehman’s laws
(adapted from 2001 talk by MML)

• Continuing Change
  An E-type system must be continually adapted else it becomes progressively less satisfactory in use

• Increasing Complexity
  As an E-type system is evolved its complexity increases unless work is done to maintain or reduce it

• Self regulation
  Global E-type system evolution processes are self-regulating

• Conservation of Organisational Stability
  Average activity rate in an E-type process tends to remain constant over system lifetime or segments of that lifetime

• Conservation of Familiarity
  In general, the average incremental growth (growth rate trend) of E-type systems tends to decline

• Continuing Growth
  The functional capability of E-type systems must be continually enhanced to maintain user satisfaction over system lifetime

• Declining Quality
  Unless rigorously adapted to take into account changes in the operational environment, the quality of an E-type system will appear to be declining as it is evolved

• Feedback System
  E-type evolution processes are multi-level, multi-loop, multi-agent feedback systems

Kinds of technical debt

- Reckless
  - "We don't have time for design"
- Prudent
  - "We must ship now and deal with consequences"
- Deliberate
  - "What's Layering?"
- Inadvertent
  - "Now we know how we should have done it"

So what to do?

• Soldier on
• Reengineer
• Scrap

The attempt to understand the system is an essential part of the decision process.