Rewrite or Refactor

When to declare technical bankruptcy

Laura Thomson (laura@mozilla.com)

OSCON - July 22, 2010

Technical debt

"Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite... The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise."

> - Ward Cunningham, "The WyCash Portfolio Management System", OOPSLA 1992

"Doing things the quick and dirty way sets us up with a technical debt, which is similar to a financial debt. Like a financial debt, the technical debt incurs interest payments, which come in the form of the extra effort that we have to do in future development because of the quick and dirty design choice. We can choose to continue paying the interest, or we can pay down the principal by refactoring the quick and dirty design into the better design. Although it costs to pay down the principal, we gain by reduced interest payments in the future."

- Martin Fowler

(Source: http://martinfowler.com/bliki/TechnicalDebt.html

Technical Inflation

"Technical Inflation could be viewed as the ground lost when the current level of technology surpasses that of the foundation of your product to the extent that it begins losing compatibility with the industry. Examples of this would be falling behind in versions of a language to the point where your code is no longer compatible with main stream compilers."

- Scott Wood

Technical bankruptcy

"Technical bankruptcy is a level of technical debt where at least one of the following applies:

- debt is accumulating faster than it can be cleared
- the time to clear debt would be longer than re-implementing the system from scratch cleanly
- sustainable people power to clear debt is unavailable"

- me, 2010

You might have a technical debt if...

“We’ll write the documentation later.”

“We’ll write the documentation later.”

“Where’s the documentation?”

"We'll write the documentation later."

"Where's the documentation?"

"There are some comments, a wiki page someone put together over there, and you should read these five bug reports."

“We don’t have any tests.”

"We don't have any tests."

"It's normal to have 86 failing tests. Those are out of date. If you have 87 fail, then you have something to worry about."

"We don't have any tests."

"It's normal to have 86 failing tests. Those are out of date. If you have 87 fail, then you have something to worry about."

"We have some Selenium tests for the front end. It was the only way to start testing 500K lines of code."

"You can just leave a TODO."

"You can just leave a TODO."

"Better mark that HACKHACKHACK. We'll fix it later."

"You can just leave a TODO."

"Better mark that HACKHACKHACK. We'll fix it later."

"Don't touch that code marked XXX. Nobody knows how it works and last time we tried to change it everything broke."

"Ignore the deprecation warnings from the compiler."

“Ignore the deprecation warnings from the compiler.”

“You’d better turn the error reporting level down.”

"In order to upgrade to the newest version of the framework, we'll need to port our local changes."

"In order to upgrade to the newest version of the framework, we'll need to port our local changes."

"This upgrade will take about six months. Maybe longer."

"In order to upgrade to the newest version of the framework, we'll need to port our local changes."

"This upgrade will take about six months. Maybe longer."

"We need to decide whether to use their implementation or ours."

"This cycle we closed 25 bugs. 37 new bugs were filed."

"This cycle we closed 25 bugs. 37 new bugs were filed."

"Adding that new feature will take 3 months. Yes, I know it's trivial."

"Joe went to work for Google. Mary quit to do a knitting startup. We don't know what happened to Phong, he just stopped showing up."

"Joe went to work for Google. Mary quit to do a knitting startup. We don't know what happened to Phong, he just stopped showing up."

"Nobody wants to work on the project."

"Joe went to work for Google. Mary quit to do a knitting startup. We don't know what happened to Phong, he just stopped showing up."

"Nobody wants to work on the project."

"Which one of you sent that to dailywtf/Coding Horror?"

Debt reduction strategies

Getting out of debt

- What would we need to do to:
  - write the missing documentation?
  - fix the tests / get decent test coverage?
  - make the code less ugly /fragile?
- How long would this take?
- Should we refactor, or just rewrite it?

Take stock

- Audit what you have:
  - Documentation? Tests? Ugly/fragile modules?
  - Process?
- What's the worst part? (What do you dread?)
- How long would each of these take to fix?
  - If it's as simple as catching up on tests or docs, hire a contractor/intern and just do it

Refactoring project plan

- Treat refactoring like any other project or feature:
  - Break into a series of achievable tasks
  - Come up with a realistic timeline and resource requirements
  - Work on the pieces in isolation or in parallel with other projects
  - Staff it seriously
  - If working in parallel, account for dependencies

Starting over

- If we threw the system away and started over:
  - How long would it take to get to where we are now? Is this less or more time than getting totally out of debt?
  - Can we cope with a lack of visible forward progress for that amount of time?
    - Still need to fix critical and security bugs on old system
  - What makes us think a new system will be any better?

Starting over with new tools

- Are there platforms or components which will get us there faster?
- Common argument is that switching toolset will improve all kinds of things:
  - Is it just greener grass?
  - Does the team have the skills to pull it off?
  - Will a system written by novices look better in 2-3 years?
  - Is our potential use of the new tool weird/different/larger scale than existing users?

SUMO:
a case study in technical debt

SUMO: a case study

- SUMO is support.mozilla.com
- SUMO began in 2007 and was launched in 2008 in time for the release of Firefox 3
- Based on TikiWiki 1.10 with (necessary) extensive customizations
- In 2009 we reached the camel point (more on this in a minute)

Scaling SUMO

- Lot of work done to make TikiWiki scale:
  - replication
  - memcache
  - tons of profiling, query changes, code rewrites, cutting includes
  - rewrote security code to make it faster
  - I have an hour long presentation on this but basically we went from serving 8 requests per second per webhead to 300+

Camel point

- TikiWiki was now at version 3 (with 4 in the oven)
- Our patches had not made it into trunk, making it hard to upgrade
- Still slow; timeouts on admin and edit pages in particular made localizers sad
- Adding new features made developers sad too
- Our debt had become unmanageable; we needed to find some relief (and not from a late night 1-800 number)

Possible options

- Upgrade: work hard on upstreaming all our changes into trunk, then switch to 4.1
- Fork: Refactor and rewrite the app as needed, heading in a different direction from the Tiki project
- Port: go to a completely different platform

Upgrade

- We decided initially to go with this option, working with the Tiki community to get our changes into trunk (being good Open Source citizens)
- All our changes were reviewed (18 months worth of code) to see if they were appropriate for Tiki, and upstreamed
- We reviewed Tiki 4 to see how it had changed

Revisiting our decision

- After several months decided to revisit decision:
  - Many of our local changes were not accepted into trunk, so would have to be maintained locally
  - Some of our issues with the code had not changed, and some had become worse
  - Many features in the code we didn't use, and these were expanding rapidly
  - Still limited tests

Rewrite

- Evaluated possible solutions as part of the initial process; upgrade initially looked good for the same reason we chose TikiWiki in the first place
- If no upgrade possible, then only one choice remained: rewrite from scratch.
  - Developers hugely enthusiastic
- Project began at the end of Q1 and has now partially launched; SUMO is now running on a hybrid of TikiWiki and Django
  - addons.mozilla.org also being rewritten in Django (from CakePHP)

Morals of the story

- It's not always as simple as "write some tests"
- Developer happiness is a critical factor
- Maintaining local changes is always painful
- Sometimes your debt is just too big to recover from. You need to declare technical bankruptcy and start over.

Some other minor case studies

- Trade magazine (Joomla -> Wordpress)
  - Speed, developer availability
- addons.mozilla.org (CakePHP -> Django)
  - Too many local modifications, developer happiness
- Electric company website (perl -> PHP)
  - Death march project, needed fresh start

Why rewrites fail

“This will solve all our problems”

“This new language is faster/cooler/more maintainable/scales better”

“This will solve all our problems”

“This new language is faster/cooler/more maintainable/scales better”

“This new platform/framework is better/faster/cooler/more maintainable/scales better”

“This will solve all our problems”

“This new language is faster/cooler/more maintainable/scales better”

“This new platform/framework is better/faster/cooler/more maintainable/scales better”

“This new database is better/faster/doesn’t use SQL”

“This will solve all our problems”

“This new language is faster/cooler/more maintainable/scales better”

“This new platform/framework is better/faster/cooler/more maintainable/scales better”

“This new database is better/faster/doesn’t use SQL”

“This new development team is better/more experienced/wear cooler hats”

“This will solve all our problems”

“This new language is faster/cooler/more maintainable/scales better”

“This new platform/framework is better/faster/cooler/more maintainable/scales better”

“This new database is better/faster/doesn’t use SQL”

“This new development team is better/more experienced/wear cooler hats”

“This new project manager is better/certified/always says yes”

“This will solve all our problems”

“This new language is faster/cooler/more maintainable/scales better”

“This new platform/framework is better/faster/cooler/more maintainable/scales better”

“This new database is better/faster/doesn’t use SQL”

“This new development team is better/more experienced/wear cooler hats”

“This new project manager is better/certified/always says yes”

“Now that we’re agile, we won’t have any problems.”

“Now that we’re rewriting it…”

“...we may as well add all these new features we’ve wanted for a long time, and couldn’t build because the old system was so hard to work with.”

“Now that we’re rewriting it…”

“...we may as well add all these new features we’ve wanted for a long time, and couldn’t build because the old system was so hard to work with.”

“...let’s incorporate these five other systems and just have one system that does everything. It will reduce duplication of effort.”

“Now that we’re rewriting it…”

“...we may as well add all these new features we’ve wanted for a long time, and couldn’t build because the old system was so hard to work with.”

“...let’s incorporate these five other systems and just have one system that does everything. It will reduce duplication of effort.”

“We won’t need any documentation because FooPlatform is so easy to understand/self-documenting/more maintainable.”

“A rewrite will take less time...”

“...because we’ve already built the system once, so we know what we’re doing.”

“A rewrite will take less time...”

“...because we’ve already built the system once, so we know what we’re doing.”

“...because our new toolkit allows for faster development.”

Think

- What will we do differently this time?
  - Will we tweak the environment to ensure we write docs and tests?
  - Will we allow enough time to build the system properly?
  - Will we beat scope creep to death every time it appears?
- Simply rewriting or changing platforms won't solve structural, environmental, or cultural problems (here be dragons)

A brave new world...

- Questions, feedback, comments...
- laura@mozilla.com