SOEN 6431 Software Maintenance and Program Comprehension

Week 1 - Technical Debt

Concordia University

GINA CODY SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

Juergen Rilling, PhD
Professor
Department of Computer Science and Software Engineering

MAILING ADDRESS
1455 DE MAISONNEUVE BLVD. W., EV-3.211
MONTREAL, QUEBEC, CANADA H3G 1M8
CONCORDIA.CA

514-848-2424 ext. 3016
juergen.rilling@concordia.ca
concordia.ca/ginacody
STREET ADDRESS
1515 Ste. Catherine St. W., EV-3.211

What is debt (from a financial perspective)?

Do you have any financial debts?

What are some of the reasons?

Is “debt” (always? ever?) a bad thing?

What about:
Technical Debt ?

Some Familiar Experience

As the process goes on, costs are rising and customer responsiveness is falling.

Using an obscure/incomplete error message because it is too hard to figure out how to do it right.

Rushing for a shippable product to meet to contract so getting something done in its own way.

After a couple of years, the system is paralyzed and so it is even cheaper to develop a new one from scratch.

What are some consequences of Technical Debt

- slapdash architecture
- poor design
- hasty coding (versus rapid)
- lack of quality focus
- others?

The danger occurs when the debt is not repaid quickly. Every minute spent on not-quite-right code results in interest on that debt.

Technical Debt

The **cost associated with a lesser software system** that became difficult to maintain and improve. The debt within the system becomes a constant anchor on software productivity and impedes the ability to efficiently add new features. [3]

Delayed technical work that is incurred when **technical short cuts** are taken, usually in pursuit of calendar-driven software schedules. Just like financial debt, some technical debts can serve valuable business purposes. [1]

Good vs bad: Reasons for technical debt

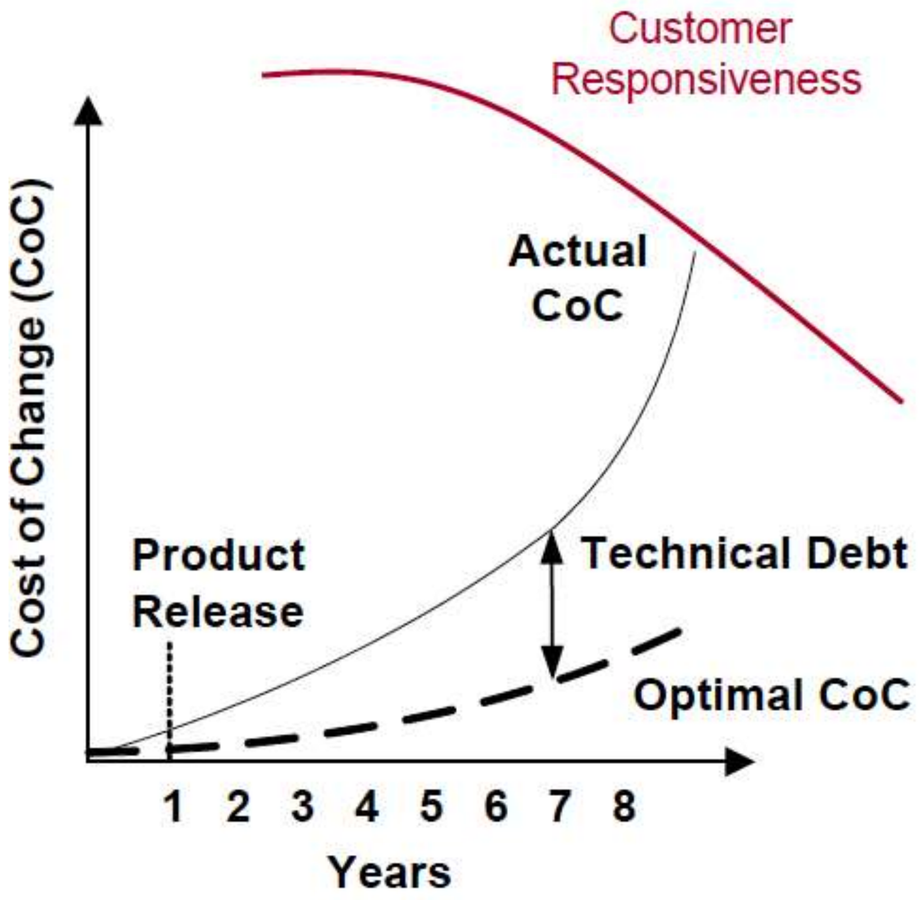
Just like financial debt, there can be good reasons for technical debt—but it is important to know that going in, so debt does not get ahead of the team, slowing down progress and future deliveries.

- **A good reason** for incurring technical debt is often because a delivery is more important than the internal cleanliness of code or smoothness of functionality. If the product works for the user, despite not being the best or cleanest product, then delivery may behoove your company in terms of time to market, revenue, etc. If, however, business needs require perfect design, you will take your time to deliver a cleaner product.
- **A bad reason** for incurring technical debt is because the team chose to focus on other areas that are more innovative or interesting but less important.

Technical Debt

Technical debt in two axis:
Reckless/prudent &
Deliberate/Inadvertent

- Reckless-deliberate: “we do not have time for design.”
- Reckless-inadvertent: “What is layering?”
- Prudent-deliberate: “We must ship now and deal with consequences.”
- Prudent-inadvertent: “Now we know how we should have done it.”



Technical Debt   Video

Symptoms of Technical Debt

Poor customer responsiveness	Management – customers
Long delivery times	For new releases, time-to-market
Late deliveries	Time consuming bug fixes
Lots of defects	Major portion of development capacity goes to defect fixing
Rising Development Costs	Higher man-hours, higher costs
Frustrated and Poor Developer Teams	Clash between rushing for deadline – “doing it right” [1]

Technical Debt Metrics

- ❑ Questions to answer for eliminating existing debt & preventing future debt:
  - ❑ How can I efficiently measure how much debt I already have?
  - ❑ What aspect of debt in my code are impacting me the most, or should I be most concerned about?
  - ❑ How much debt is “too much”; how much is “manageable”?
  - ❑ How much will it cost me to get out from under my high interest debt?
- ❑ To characterize and to measure the existing debt related to project.
  - ❑ Static and dynamic tools are on service. (i.e. Sonar)

Technical Debt Metrics

- ❑ Duplicate code (Static)
- ❑ Rules Compliance
- ❑ Comments – Interfaces (Static)
- ❑ Comments – General (Static)
- ❑ Package Interdependencies (Static)
- ❑ Method/Class Complexity (Static)
- ❑ Automated Test Coverage (Dynamic)
- ❑ Manual Test Coverage (Dynamic)

This list of metrics also represent the priority to capture.

Definition and Cost Of Technical Debt From Different Perspectives

Customers:

Biggest victims in this affair. However; they usually give more credit on features than the long-term maintainability, security and reliability. Particular customers are in a different case.

Help Desk:

Personnel in this branch suffer from almost every aspect of technical debt (poor interface, lack of documentation, slow performance…) Even though, they do not have direct access to solve the problem.

Operations:

Especially in service-oriented business; this branch pay for the outcomes of decision-makers. Recently emerged "DevOps" approach (Operations people and developers work together) [2]

Definition and Cost Of Technical Debt From Different Perspectives

Engineers:

Developers (implement the code) and maintainers (repair, modify, extend the existing code). It is perceived as developers and strongest yet maintainers do the hardest job.

Marketing:

Customer facing people; usually must deal with the displeasure of the customers in the cases of technical debt. Long release intervals push force them against customers. (No need to mention about the situations in which the new release does not work properly.)

Management:

Conducting good risk management and balancing demands from different branches & Favor a single branch/department and neglect the others which leads taking too much technical debt or missing the advantage of taking a right amount. [2]

Making a Repayment Plan

- ❑ Engagement, motivation and a plan lead exponential value to customers.
- ❑ Engaging executive management in explaining what technical debt is.
- ❑ Explaining how it is impacting responsiveness, costs, delivery and morale.
- ❑ Getting buy in to make the necessary investments to pay down technical debt and proactively manage it going forward.
- ❑ Creating a realistic plan and appropriate governance model.

- ❑ When interacting with high rank executives; it would help to explain technical debt as analogous to financial debt.[1]

Strategies for Paying Down Technical Debt

After tracking technical debts in a project by using related *metrics* and forming a *repayment plan*

Poor coding and testing practices:

- Often conducts from lack of time and/or knowledge to do it right. Each team member should invest in their own knowledge and education on how to write clean codes, related business domain and how to do their jobs optimally. [1]

Poor system design:

- Most architects over-focus on user features and under-focus on the actual important qualities of a software like maintainability, performance, reuse and adaptability which directly relate to the technical debt.[1]

Poor Communication and Collaboration

- Directly related to technical debt. At least a daily stand-up meetings (15 minutes at max) for entire team to synchronize each member's schedule. People can adjust and collaborate on accomplishing important tasks.[1]

Strategies for Paying Down Technical Debt

After tracking technical debts in a project by using related *metrics* and forming a *repayment plan*

Lack of standard development processes:

- Companies often use a blended way of their own instead of a proven method in which cases documentation does not reflect how it is done in practice. [1]

Poor requirements with no common *definition of done*:

- For each requirement, use case and user story there should be a common *definition of done* which must be validated with management before and after development phase. [1]

Lots of manual effort for a release:

- A few team members who are fully focused on automated tools can accomplish a great deal of integration / testing / database setup work in a relatively short time.

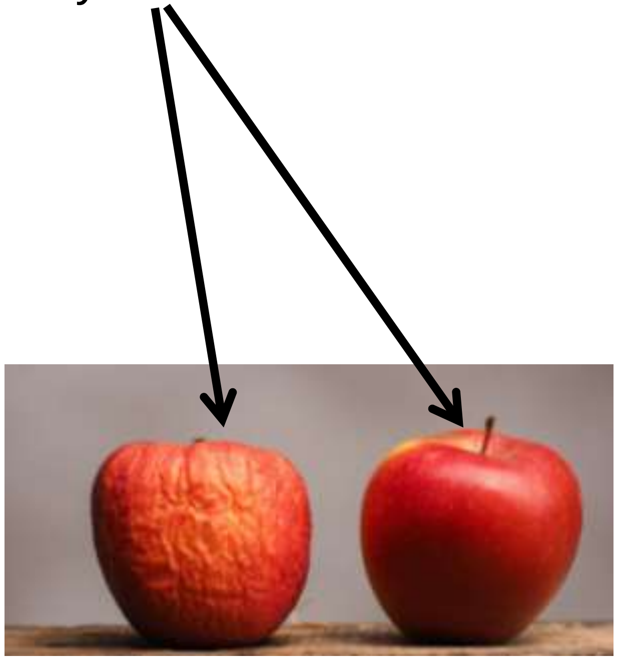
Lack of proactive investment in paying down debt:

- Sometimes technical debt should be taken on only with a conscious decision (not recklessly) Management side must have good deal of knowledge about technical debt and tackling it. [1]

A more detailed look how we can remove technical debt….

1. Re-Do Project

- Business decides on a total re-design and re-write
- Happens rarely, hard to justify
- Hard for execs to see the value
- Often timed with new technology introduction
- How to ensure that the new system is really better?
  - Rigorous quality plan
  - Strong Definition of Done
  - Iterative development
  - Feedback loops
  - Inspect & adapt
  - Watchful eyes



Not an answer: “better people”

2. “Small Chunks” Approach

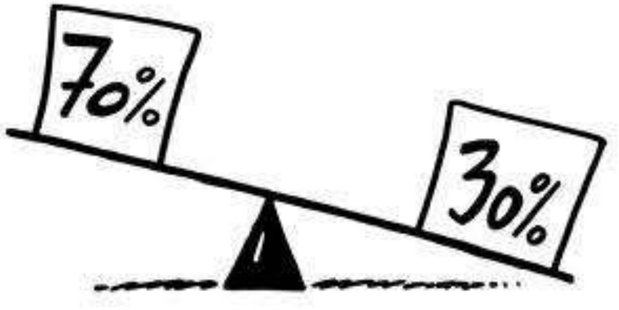
- Smaller chunks of effort
- In conjunction with normal dev flow
- Can work well across an API abstraction
- Often requires an architectural assessment
  - Can require code changes to other parts of legacy base (risky)
- Happens more often than Re-Do projects
- Must insure end result is worth the effort!

3. 70/30 Rule

- 70% - new user stories
- 30% - paying down the debt of a legacy system

Rule can also be used to manage <u>new</u> technical debt

- Helps to have an expert who knows the history of the legacy system.
- Socialize that legacy improvements are part of this project!
  - Ask for commitment
  - Ask to be held accountable
  - Do your part
  - Ask others to continue the tradition



4. "While There" Rule

While adding a new feature

Any legacy function modified for the feature must be rewritten according to quality plan

- Variable names
- Coding guidelines
- Braces alignment
- Etc.

Works well for support team also (while fixing bugs)

CAUTION: may introduce bugs

Preventing Technical Debt: Draw the line…

- **Culture shift:**
  - From this day forward …
  - We no longer allow technical debt!
  - Only exception: tactical or strategic business justification

OSS Definition of "Done"

Story

- Code Documented
- Rally updated
- Passed unit tests
- Pass FIT tests – mock data okay
- Pass UI tests
- .MSIs/ Build for QA/Dev/ Release

Sprint

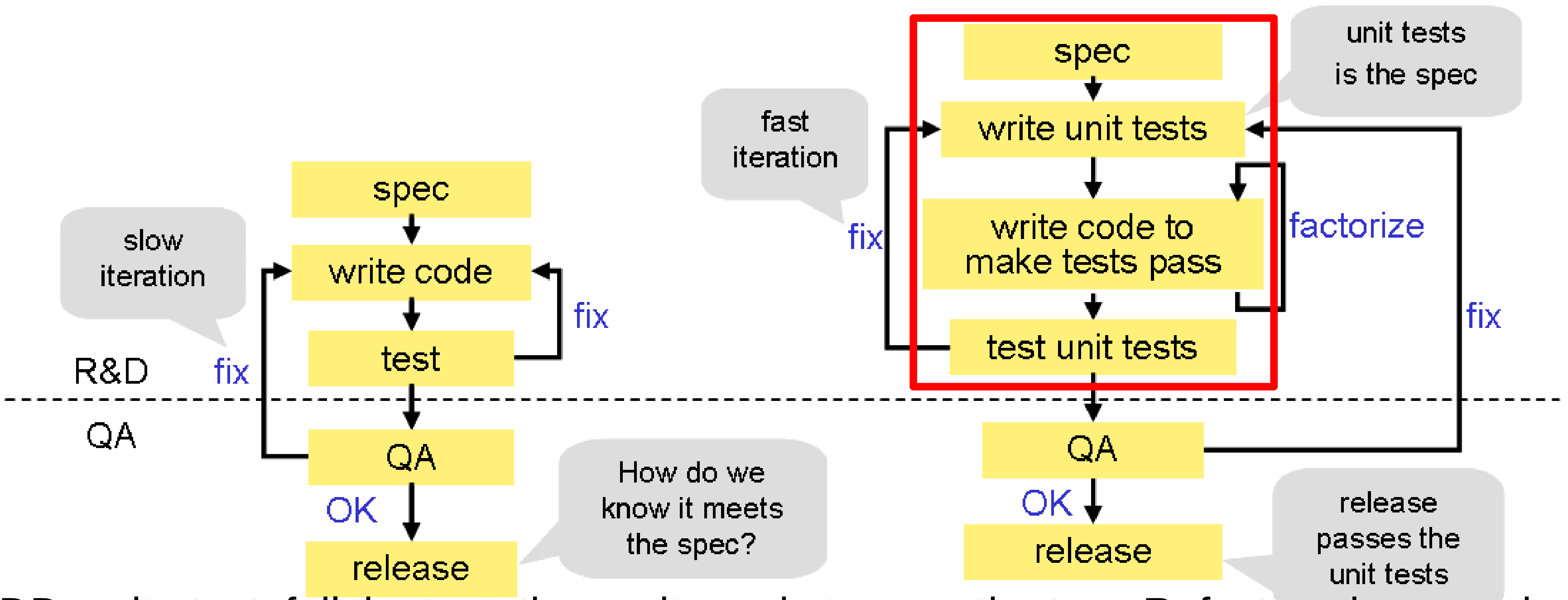
- Pass Fit tests
- Approval UAT and QA
- Problem Mgt Ticket
- Coverage Review
- Rally updated

Release

- Pass Fit tests: live data reqd
- Update website/web pages
- Test Plan approved (SOX)
- Test Cases with Expected Results
- Comm to employees
- CR
- Rally updated

Impediments

- Vendor SW quality issues
- Vacation and Out of Office
- Working code in dependent systems
- Trust
- Team commit to delivery
- Non-OSS distractions



TDD: write test, fail the test, then write code to pass the test. Refactor when needed.

Develop higher quality streamlined code!

Preventing Technical Debt
- Collaborative Designs

- Whiteboard sessions on design topics
- Leverage collective experience of the team
- Wisdom of the crowd
- User Design Studio
- Less reliance on subject-matter experts
- Etc.

Increase the chance of an optimal design!

References:

[1] Shriver, Ryan. "Seven Strategies for Technical Debt". (2009). www.projectmanagement.com (former www.gantthead.com)

[2] Allman, Eric. "Managing Technical Debt. (2012). ISSN 1542-7730. Queue – Development Magazine Volume 10 Issue 3, March 2012 pp 10-17. New York – USA.

[3] Eisenberg, Robert. (2012). ACM SIGSOFT Software Engineering Notes, ISSN 0163-5948. March 2012, Volume 37, Issue 2 pp 1-6.

[4] www.sonarsource.org