Construx®
Software Development Best Practices

# Managing Technical Debt

Steve McConnell
www.construx.com

Construx®
Software Development Best Practices

## Copyright Notice

## 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)



## Technical Debt Example

"Guys, we don't have time to dot every i and cross every t on this release. Just get the code *done*. It doesn't have to be perfect. We'll fix it after we release."

# Technical Debt Example

"We don't have time to reconcile these two databases before our deadline, so we'll write some glue code that keeps them synchronized for now and reconcile them after we ship."



# Favorite Classroom Examples of Technical Debt (input from ~150 practitioners 2012-2013)

- 50% of code for the whole system in one class (~500 KLOC system)
- 1 routines 4,000 LOC
- One stored procedure 15,000 lines long
- Maintaining code when two groups each implemented solution to the same problem
- Increasing use of "bad patches", which increases number of related systems that must be changed in parallel
- Hundreds of customer-specific branches on same code base
- Severe lack of information hiding (every header file includes every other header file)
- "Friendly" additions to interfaces (developer to developer)
- Continuing to build on a foundation of poor quality legacy code
- Legacy systems (not migrating clients off the legacy systems and then having to maintain them)
- Porting code from different platforms and hacking it to make it work
- Products that aren't end of life (after thought they were going to be)
- Prototype that turns into production code

## Reasons to Discuss Technical Debt

- Helps to educate technical staff about business decision making
- Helps to educate business staff about technical decision making
- Raises awareness/transparency of important issues that are often buried
- Allows technical debt to be managed more explicitly
- The analogy is rich and produces numerous insights



## Objectives of this Talk

- Introduce the technical debt metaphor
- Explore the full scope of what's possible with the concept

My focus is mostly on upstream aspects of technical debt

## Talk Roadmap

- A Debt Primer
- Ins and Outs of the Technical Debt Analogy
- Deciding to Take on Technical Debt
- Tracking Technical Debt
- Paying Down Technical Debt
- Summary: What Can we Do with Technical Debt?



Construx®
Software Development Best Practices

# A Debt Primer

Reasons to Take on Technical Debt

# The Business View

- Shortening time to market
- Preserving startup capital
- Delaying development expense

- Business staff tend to be optimistic (or ignorant) about the benefit of taking on the debt and about the costs of carrying the debt



Reasons to Take on Technical Debt

# The Technical View

Debt can create a vicious circle

Reasons to Take on Technical Debt

## The Technical View

- Technical staff tends to be pessimistic about the benefit of taking on the debt and about the costs of carrying the debt
- Attitude toward debt can be religious
- Attitude toward debt can be aesthetic



## Business vs. Technical Viewpoints

- Business staff tends to be optimistic about debt
- Technical staff tends to be pessimistic about debt
- These attitudes are often *not conscious*
- Debt is a tool, neither inherently good nor bad
- The Technical Debt metaphor gives these two groups a constructive way to approach some important discussions

# Short-Term vs. Long-Term Debt

Short-Term Debt

- Financial Debt
  - Used to finance temporary cash flow issues, e.g., to cover receivables
  - Expectation is normally that debt will be repaid in the short term
- Technical Debt
  - Skipping unit tests to get a release out the door
  - "We don't have time to implement this the right way; just hack it in and we'll fix it after we ship."
  - Violating coding standards to upload a hot fix



# Short-Term vs. Long-Term Debt

Long-Term Debt

- Financial Debt
  - Used to finance longer term investments
  - Expectation is normally that there is some strategic reason for taking on long-term debt
- Technical Debt
  - "We don't think we're going to need to support a second platform for at least 3 years, so our design supports only one platform."

# Intentional vs. Unintentional Debt

Intentional Debt

- "If we don't get this release done, there won't be a second release." (Rationale for cutting corners)
- "We don't have time to build in general support for multiple platforms. We'll support iOS now and build in support for Android, etc., later."
- "We didn't have time to write unit tests for all the code we wrote the last 2 months of the project. We'll write those after we release."



# Intentional vs. Unintentional Debt

Unintentional Debt

- A junior programmer writes bad code
- A contractor writes code that doesn't follow the coding standard
- A major design strategy turns out poorly
- Your company acquires a company that has a lot of technical debt
- A comprehensive refactoring project goes sideways

## Interest / Debt Service

- Interest is a premium paid above and beyond the principal that is in essence a service charge for being allowed to take on the debt
- Interest can be paid periodically or all at once when the debt is repaid
- To qualify as "debt," there must be some kind of interest—now or later or both



## Examples of "Interest Payments" on Technical Debt

- Lack of unit tests on legacy code causes new development, testing, and debugging to take longer
- Overly simple design does not readily support changes in the environment, and seemingly simple environment changes require massive code rework
- High product support cost for a buggy system (non-software interest payment)
- Brittle system means each bug fix introduces unintended side effects (i.e., new bugs), so each simple bug fix becomes a multi-bug-fix project
- Bug reports are so frequent that time spent fixing bugs in the production system prevents any work on new functionality
- Overly lengthy edit-compile-debug times due to poor development environment (non-code cause of technical debt)

*Interest and Debt Service*

## Not All Delayed Work is Debt

- Many kinds of delayed or incomplete work are not debt:
  - feature backlog, deferred features, cut features
- If the work doesn't create an interest payment of some kind, it isn't debt



## Summary of Categories of Technical Debt—What Debt *Is* Incurred

Unintentional Debt. Debt incurred unintentionally due to low quality work.

Honest Mistakes. "I wish we'd known Framework 2.1 would be so much better than Framework 2.0."

Careless Mistakes. "Design? What design? Design is for sissies."

Intentional Debt. Debt incurred intentionally

Short-Term Debt. Short-term debt, usually incurred reactively, for tactical reasons

Focused Short-Term Debt. Individually identifiable shortcuts (like a car loan, e.g., partial implementation of a class)

Unfocused Short-Term Debt. Numerous tiny shortcuts (like credit card debt, e.g., not following the coding standard)

Long-Term Debt. Long-term debt, usually incurred proactively, for strategic reasons

## Summary of Categories of Technical Debt—What Debt *Should be* Incurred

- Unintentional Debt. Debt incurred unintentionally due to low quality work.
  - Honest Mistakes. "I wish we'd known Framework 2.1 would be so much better than Framework 2.0."
  - Careless Mistakes. "Design? Real men don't do design."
- Intentional Debt. Debt incurred intentionally
  - Short-Term Debt. Short-term debt, usually incurred reactively, for tactical reasons
    - Focused Short-Term Debt. Individually identifiable shortcuts (like a car loan, e.g., partial implementation of a class)
    - Unfocused Short-Term Debt. Numerous tiny shortcuts (like credit card debt, e.g., not following the coding standard)
  - Long-Term Debt. Long-term debt, usually incurred proactively, for strategic reasons



Construx®

Software Development Best Practices

# Ins and Outs of the Technical Debt Analogy

*Why Take on Debt?*

# Expected Value

- From statistics, the expected value is the probability of an amount occurring times the amount
- The expected value (i.e., expected cost) of Task A is much lower if I *might* do it sometime in the future than if I'm actually doing it right now



*Why Take on Debt?*

# Present Value

- Would you rather have $100 today or $100 in 2 years?
- Likewise, would you rather pay $100 today or $100 in 2 years?
- In software projects, calculation of present value is tied up with opportunity cost, expected value, and other factors

*Why Take on Debt?*

## Opportunity Cost

- Opportunity Cost = What else could I be doing with this money/resource?
- The expected value of whatever else you could be doing with that money/resource is the opportunity cost
- This is a key concept in time-sensitive markets where an extra $1 in development cost can translate into $10 in lost revenue opportunity
  - A debt decision that increases the technical cost from $1 now to $5 later can be a good decision if it also increases revenue from $1 to $10



## Debt Service Coverage Ratio (DSCR)

- In finance, a company's DSCR is a specific, common measure of financial health
- In software, the analogous DSCR can refer to the ratio of work spent on advancing the software vs. keeping the software from sliding backwards
- This can be used as a proxy measure for technical debt

# Credit Rating

- Different consumers/businesses have different borrowing power based on their credit rating
- Teams have different abilities to carry technical debt responsibly
  - A team with a lot of unintentional debt due to low quality work will have less ability to take on debt for strategic reasons than a team with lower unintentional debt



# Acquired Debt

- Debt was taken on for important strategic reasons, but you weren't there at the time
- Debt was taken on for reasons that you (or your team) would never have bought into if you had been there at the time
- Sometimes you acquire debt from another company in an acquisition

## 90 Days Same as Cash!

- Sometimes this option is available, i.e., no interest accrues if you pay the debt off quickly enough

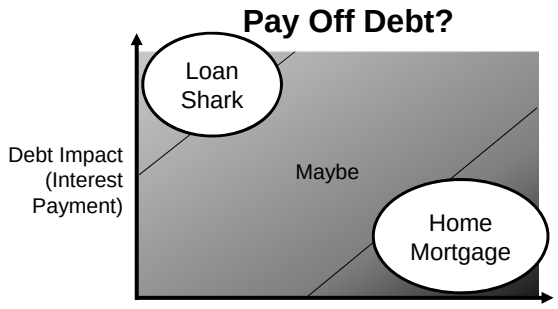


## Low Monthly Payments!

- When debt is incurred, what threshold level of activity is needed to make the "minimum payment?"
  - Roll up a new version on a website?
  - Send out DVDs?
  - Replace a chip in a computer?
  - Replace a box on an airplane?
- Ability to pay off debt depends in part on the kind of software the debt applies to

# Consider the Debt's Interest Rate

- Not all debt is equal
- Pay off high interest debt before you pay off low interest debt
- Some low interest debt may be low enough that you never pay it off (moral equivalent of a home mortgage at 3.0%)






# Retiring Debt

- Teams normally find "working off debt" to be motivating

Special Case: End of Life System

- Unlike financial debt, when a system is retired, all its technical debt is retired with it
- As a system approaches end of life, cost justifying anything other than the most expedient solution becomes increasingly difficult

Construx®
Software Development Best Practices

# Deciding to Take On Technical Debt

## Debt Decision Making

Option 1

Immediate cost of Good Solution: 10 staff days
Deferred cost to retrofit Good Solution: 0 staff days

| | |
|---|---|
| Option 1 cost now: | $5,000 |
| Option 1 cost later: | $0 |
| Option 1 total cost: | $5,000 |

# Debt Decision Making

Option 2

Immediate cost of Quick & Dirty solution with possible interest payment: 2 staff days
Deferred cost to retrofit Good Solution: 12 staff days
Estimated cost of "interest payments": 1-3 staff days

| | |
|---|---|
| Option 2 cost now: | $1,000 |
| Option 2 cost later: | $6,000 |
| Option 2 interest payment: | $500-$1,500 |
| Option 2 total cost: | $7,500-$8,500 |



# Debt Decision Making

Option 3

Immediate cost of Quick & Dirty solution with no interest payment: 4 staff days
Deferred cost to retrofit Good Solution: 12 staff days

| | |
|---|---|
| Option 3 cost now: | $2,000 |
| Option 3 cost later: | $6,000 |
| Option 3 interest payment: | $0 |
| Option 3 total cost: | $8,000 |

# Debt Decision Making

Option 3

Immediate cost of Quick & Dirty solution with no interest payment: 4 staff days

Deferred cost to retrofit Good Solution: 12 staff days

| | |
|---|---|
| Option 3 cost now: | $2,000 |
| Option 3 cost later: | $6,000 |
| Option 3 interest payment: | $0 |
| Option 3 total cost: | $8,000 |

*This may be the only cost we ever have to care about*



# Key Point in Taking on Debt

- Avoid binary "debt/no debt" decisions—the universe of options is rarely that limited
  - Especially look for "zero interest" options
  - Give yourself the option of *never* paying off the debt

## Key Questions to Ask Before Deciding to Take on Debt

- Do we have estimates for the debt and non-debt options?
- Do we *believe* the estimates?
- How much do we reduce our effort *now* by taking on the specific technical debt?
- *How much* will the quick & dirty option cost now?
- *How much* will the "clean" option cost now?
- How much will it *cost later* to replace the Q&D option with the clean option?



## Key Questions to Ask Before Deciding to Take on Debt

- How much will the *"interest payment"* on the Q&D option be?
- How will the interest payment be *structured*, i.e., when will we have to pay "interest"?
- Are there any issues involved in *paying back* the debt?
- *Why* do we believe that it is better to incur the effort later than to incur it now? What is expected to change that will make the payment more palatable in the future than it is now?
- Can we *track* this specific debt?
- Have we considered all options—especially have we considered any *zero interest* options that we can implement at low cost now?
- *Who in the business* is going to *own* the debt?

## How Much Debt is Enough/Too Much?

- There is no one right answer for business debt, and there is no one right answer for technical debt
- Technical staff's attitude is sometimes extreme: "zero debt"
- Decreasing velocity can be an indicator
- Business staff tends to have a higher tolerance for technical debt (but weaker memory of it)
- Work is required to ensure the organization remembers its debt decisions



## Indicators of Undesirable Debt

- High interest rate
- High minimum payment
- Required, ongoing payments
- Debt is untrackable
- Debt is not intentional
- Low ROI

Construx®
Software Development Best Practices

# Tracking Technical Debt

## Tracking Debt

- All “good debt” can be tracked (by definition)
- Log as defects
- Include in product backlogs
- Monitor project velocity
- Monitor amount of rework

## Ways to Measure Debt

- Total of debt in product backlog
- Maintenance budget (or fraction of maintenance budget)
- “Aged” customer work backlog
- Measure debt in money, not features, e.g., “50% of R&D budget is non-productive maintenance work”



Construx®
Software Development Best Practices

# Paying Down Technical Debt

## Good and Bad Reasons to Pay Down Debt

- Good reasons are normally based on business need/business benefit
- Bad reasons (aka, reasons that business people won't accept) tend to be very technical sounding



## Example of a Good Reason to Pay Down Debt

Scenario: Pay off short-term debt

Example:

"We took some conscious short cuts to get the last release out the door, and now we need to ensure the codebase is stable before we start writing new functionality."

## Example of a Good Reason to Pay Down Debt

Scenario: Debt service (in the form of longer release cycles) is undermining business operations

Example:

"We'll be able to get future releases out in 5 months instead of 6 if we can spend X amount of time reducing our debt"



## Example of a Good Reason to Pay Down Debt

Scenario: Debt itself (in the form of low quality) is undermining business operations

Example:

"Some of the 'shortcuts' we have been taking have started to become visible to the customer, and the number of customer-reported defects has been increasing. Until we pay down our debt we will have an increasingly difficult time assuring quality prior to release."

## Example of a Good Reason to Pay Down Debt

Scenario: Debt (in the form of technical limitation) is preventing access to new business opportunities

Example:

"If we spend X weeks working on technical infrastructure in Area Y, we'll be able to add features A, B, and C, which we can't add now because of shortcuts we took when we originally implemented Area Y."



## Bad Reasons to Pay Down Technical Debt

- Business is rarely motivated to pay down technical debt because the design is "bad," "old," "crufty," "not object-oriented," "not agile," etc.
- Justification for reducing debt needs to be explained as a *business benefit*
- Some debt reduction can be planned into normal work flow *as part of doing a good job* and doesn't have to be justified separately

## Approaches to Actually Paying Down Technical Debt

- Do a debt reduction iteration immediately following full product release (for short-term debt)
- Amortize small debt payments into each iteration (e.g. 10% of effort each iteration)
- Above a certain threshold (~100 hours? ~250 hours?) debt reduction has to be approved as a separate project
- Individuals rotate through debt-reduction duty
- Offshore resources are used to reduce debt



Construx®
Software Development Best Practices

# Summary: What Can We Do With Technical Debt?

## “Reification”

- The main value of “technical debt” is reifying a concept that’s otherwise too intangible to be handled well



## What Can We Do With Technical Debt?

We can:

- Talk with technical staff about it
- Communicate to business staff the implications of it
- Measure the amount of it
- Make explicit decisions about whether we should take on more of it
- Get input from the business about whether the business believes we should have less or more of it
- Strategize how to avoid it
- Track it
- Make explicit decisions about when and how to reduce it

Construx®

Software Development Best Practices

Construx Software is committed to helping individuals and organizations improve their software development practices. For information about our training and consulting services, contact stevemcc@construx.com.

**Construx**

10900 NE 8th Street, Suite 1350
Bellevue, WA 98004
+1 (866) 296-6300
www.construx.com