At the Intersection of Technical Debt and Software Maintenance Costs

Arlene Minkiewicz, Chief Scientist

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Agenda

- Introduction
- Why Debt?
- Technical Debt Definition and Categorizations
- Measuring Technical Debt
- Intersection of Technical Debt and Software Maintenance
- Software Maintenance Cost Estimation Considerations
- Conclusions and Future Work

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

PRICE MAKING A DIFFERENCE FOR 40

"Shipping first time code is like going into debt. A little debt speeds up development, as long as it is paid back promptly with a rewrite"

Ward Cunningham

Introduction

PRICE MAKING A DIFFERENCE FOR 40

- Technical debt is a metaphor used to ease understanding between business leaders.

Metaphors

A comparison in which one thing is said to be another.

Example:

She is a walking dictionary.

Introduction

- As a metaphor technical debt applies to the situation where some **short cut** in process or standards is applied in order **to meet a short term goal** such as:
  - Forgoing standards to get to market faster
  - Quick patches (violating best practices) to fix a bug in order to satisfy an immediate customer demand
  - Poorly documented code to accelerate development to meet a demanding schedule
- As a metaphor it creates a situation allowing business leaders to make good trade-off decisions based on what they expect to gain versus what they will have to invest in future releases.

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence* 5

Why Debt?

- **Sometimes** debt is wise and reasonable
- If you want to start a business and have reasonable expectations that it will be a success as you have planned it – it makes sense to take on some debt to get it started
  - As you make sales, you will have the means to pay both the principal and the interest
- Similarly with software there are times when it makes sense to skip a few steps to get to market or please a customer
  - Conscious decision is made that the short term goal is worth the cost
  - The principal is thought of as the amount of effort (story points, function points, etc.) required to remove any violations created by the shortcut or deviation from standards
  - The interest is the increased cost created each time the debt ridden code is touched

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence* 6

Technical Debt for Software

- Forms of debt for a software application include the following types of things:
  - Lack of documentation
  - Poor or missing comments
  - Lack of adherence to best practices, process or standards
  - Missing tests or poor test coverage
  - Lack of modularization
  - Architecture that is not well thought out or scalable
  - Overly complex function, modules or classes
  - Failure to keep up with technology

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Technical Debt Defined

- As a metaphor ….
  - Technical debt facilitates discussions between development teams and business leaders to support sensible decisions about Return on Investment (ROI)
  - Technical debt does not include the results of an inexperienced junior developer writing bad code because they don't know any better. – This is just bad code.
- As a metric applied to an application…
  - Technical debt includes all violations of good coding practices in a code base or application – regardless of whether they were consciously made
  - Technical debt includes both debt assumed consciously and that which results from sloth or inexperience.

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

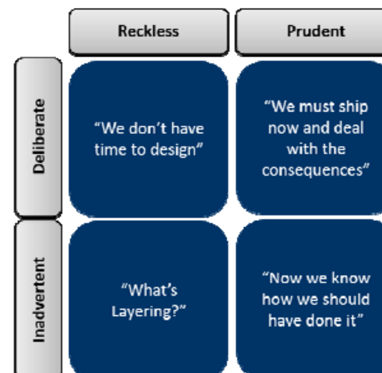
Technical Debt Defined

- Technical debt is not a measure of the defects in the code – these represent the functional quality of code.
- Technical debt speaks to the structural quality of code.
  - Many side effects of technical debt will not be observable to the user
  - End users of systems with accumulating technical debt may see performance degradation, decreases in updates and bug fixes, corrupted data, etc.

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Technical Debt Classification

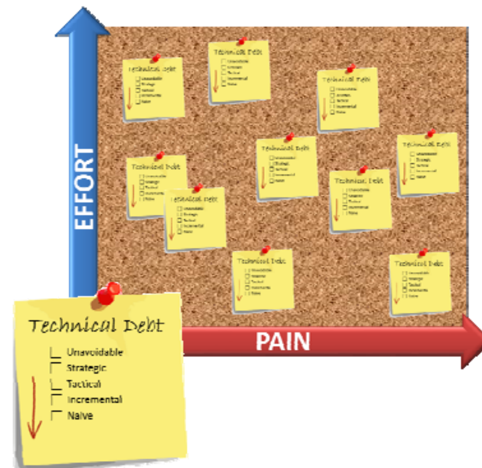
- Fowler classifies technical debt in two dimensions – reckless or prudent, deliberate or inadvertent [1]:
  - Reckless and inadvertent
  - Reckless and deliberate
  - Prudent and inadvertent
  - Prudent and deliberate



© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Technical Debt Classification

- Another way to classify technical debt focuses on how it is incurred [2]:
  - Unavoidable debt – due to changes in law or certification requirements – generally unavoidable
  - Strategic debt – usually incurred proactively
  - Tactical debt – no time to do it right – reactive
  - Incremental debt – lost of little short cuts that add up
  - Naïve debt – back to the bad code mentioned earlier



© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Measuring Technical Debt

- There are several approaches an organization can employ:
  - Organizations that embrace the metaphor have an excellent grip on the debt intentionally incurred
  - Some organizations use home grown or industry metrics as a proxy for technical debt:
    - *Number of tests per size measure*
    - *Code coverage*
    - *Coupling or cohesion factors*
    - *Cyclomatic complexity*
    - *Comments per line of code*
    - *Etc.*
  - Static analysis tools are available (both commercial and open source) that will examine code looking for violations of standards and best practices based on
    - *software engineering standards*
    - *Consortium for IT Software Quality (CISQ),*
    - *Software Engineering Institute (SEI)*
    - *Object Management Group (OMG)*
    - *etc.*

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Measuring Technical Debt

- According to Curtis et. al. [3],[4],[5] technical debt is assessed by examining the following characteristics of the code:
  - Robustness
    - *Stability and resilience of an application – how well it avoids outages and how quickly it can recover from them*
    - *Indications include poor memory management, uncontrolled data access, open resources not being closed, etc.*
  - Performance efficiency
    - *Application speed and efficiency with which it consumes resource*
    - *Indications include complex queries on big tables, large indices on large tables, etc.*
  - Security
    - *Application is able to prevent unauthorized intrusions and protect data*
    - *Indications include buffer overflows, uncontrolled format strings, etc.*

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Measuring Technical Debt

- Transferability
  - *Ease with which new team can acclimate and understand the application to become productive working with it*
  - *Indications include lack of comments, misuse of inheritance, naming convention violation, etc.*
- Changeability
  - *Ease with which modification can occur without injection of new defects*
  - *Indications include lack of tests, high cyclomatic complexity, duplicated code, etc.*

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Software Maintenance

"It doesn't take a lot of skill to get a program to work. The skill comes in when you have to keep it working"

Robert Martin

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Software Maintenance and Technical Debt

- Software Maintenance is generally considered to be any change made to an application after it has been released for production
- On average, 80% of maintenance activities go towards tasks other than correcting bugs
- Software maintenance is typically categorized as follows:
  - Corrective – fixing defects
  - Perfective – adding new capabilities
  - Preventive – addressing thing in the code that are not causing faults but tend to make the code error prone and hard to maintain
  - Adaptive – making the code continue to thrive as hardware and technology progress
- According to the Guide to the Software Engineering Book of Knowledge (SWEBOK Guide) 40-60% of most maintenance tasks are devoted to understanding the code being maintained[6]

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Software maintenance and technical debt

- Adaptive and Preventive maintenance seem to be focused specifically on addressing the technical debt in an application
- An understanding of technical debt will drive maintenance 'size' associated with addressing that technical debt.
- Regardless of the types of maintenance being accomplished, all software maintenance estimates should be informed by the technical debt associated with transferability, changeability and understandability.

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Software Maintenance and Technical Debt

- Some possible measures to indicate level of technical debt associated with transferability and changeability:
  - Software Size (SLOC, Function Points, etc.)
  - Class Coupling
  - Cyclomatic Complexity
  - Halstead Volume
  - Average Depth of Inheritance
  - Average Size of Functions, Modules or Classes
  - Average Number of Comments per size unit for Functions, Modules or Classes
  - Number of tests per function, Module or Class
  - Maintainability Index

$$MI = 171 - 5.2\ln(aveV) - 0.23aveV(g') - 16.2\ln(aveLOC) + 50\sin\sqrt{2.46\,perCM}$$

- None of these measures has conclusively been determined to be a sole driver of software maintenance costs
- There is evidence that some (or some combination of these measures) should be included, along with other factors of the maintenance project, as part of an effort or cost estimate for that project.

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Software Maintenance Estimation Considerations

- Consider the activities that need to be completed for each of the types of Software Maintenance:

Type of Maintenance	Tasks
Corrective	
	Code Understanding
	Problem Identification and Analysis
	Design
	Implementation
	Regression Testing
	Acceptance Testing
Perfective	
	Code Understanding
	Requirements Analysis
	Design
	Code and Unit Test
	Software Integration and Test
	Acceptance Testing
Preventive and Adaptive	
	Code Understanding/Reverse Engineering
	Problem Identification and Analysis
	Forward Engineering
	Implementation
	Regression Testing
	Acceptance Testing

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Software Maintenance Estimation Considerations

- From an estimation perspective, corrective and perfective maintenance should be handled very similarly to an estimate of new development
  - Size is determined based on the amount of functionality to be added and the amount of functionality changed
  - The only area where there are new influences are in the area of maintainability and understanding
- The COCOMO II Model – which has a software maintenance effort model will be used to demonstrate one approach where technical debt measures may influence maintenance estimates. This same methodology could be applied to any software maintenance model

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Software Maintenance Estimation Considerations

- The COCOMO II model has a software maintenance effort model
- This model is very similar to the development model with the following exceptions:
  - Maintenance Size = Size * Maintenance Change Factor (MCF) * Maintenance Adjustment Factor (MAF) where:
    - *MCF is a user input indicating how much functionality is being changed (Added Functions + Changed Functions)/Total Functions*
    - *MAF is 1 + (Software Understanding * Programmer Familiarity) where*
      - Software Understanding is a user input ranging from 10 to 50 to indicate how maintainable the software is (very similar to the Maintainability Index mentioned above)
      - Programmer Familiarity indicate how close the maintenance team is to the application being maintained
  - RUSE (Design for Reuse) and SCED (Schedule Compression) are not considered necessary for the maintenance effort
  - RELY (Reliability) has the inverse impact on effort in maintenance than it does in development
- Understanding and Familiarity are significant cost drivers

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Software Maintenance Estimation Considerations

- It is interest to note, the assumption is made that even with the highest score for maintainability an organization can expect to deal with a 10% increase in ‘size’ to compensate for learning the code to be maintained
- Further, there are cases where size adjustments for understandability and familiarity might not be the best approach
  - While understandability and familiarity are very important during requirements, design and code they may be less important than other factors such as number of automated tests or test coverage during test related maintenance phases
- Certainly the measures that are available should determine the best approach for adapting maintenance effort calculations

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Software Maintenance Estimation Considerations

- When estimating software maintenance for preventive or adaptive maintenance, the issues are not as clear cut
- Clearly the issues of maintainability and understandability continue to apply.
- But size estimation is complicated by the fact that, as estimators, we tend to think of software size as directly related to the functionality being delivered
- With adaptive and preventive maintenance, no new functionality is being delivered.
- Estimators need to work with engineers for assistance in
  - Understanding what technical debt will be addressed for a particular release
  - How much of the software's functionality will be touched in this context
  - What if any overlaps should be expected with new/changed functionality in the release
  - Their assessment of technical debt in the code

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Conclusions and Further Work

- Technical debt is used in the software industry in two distinct ways:
  - As a metaphor to help business and development make wise trade-offs
  - As a measure of the number of 'violations' in a code base or application
- Technical debt is not a measure of the defects in code, rather it is an indicator of structural quality
- Technical debt information can be helpful in estimating software maintenance costs in two ways:
  - Indication of how understandable and maintainable existing software is
  - Identification of areas where rework is necessary – facilitating discussion between engineers and estimators to help 'size' that rework
- Data collection is on-going comparing code complexity and maintainability metrics per software unit with the effort spent maintaining that unit

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

Q&A SESSION

Arlene Minkiewicz
Chief Scientist
Arlene.Minkiewicz@PRICESystems.com

Arlene F. Minkiewicz
Chief Scientist
PRICE® Systems, LLC.
17000 Commerce Parkway – Suite A
Mt. Laurel, NJ 08054
Office 856-608-7222 **Mobile 856-630-9408**

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*

References

[1] Codabux, Z., Williams, B.,"Managing Technical Debt: An Industrial Case Study, 2013, International Workshop on Managing Technical Debt, San Francisco, CA, May 2013, p8-15

[2] Ergin, L., "Technical Debt- Do not Underestimate the Danger", available at http://www.slideshare.net/lemiorhan/technical-debt-do-not-underestimate-the-danger (Retrieved 3/12/2015)

[3] Curtis,B., Sappidi, J., Szynkarski, A., "Estimating the Principal of an Application's Technical Debt", IEEE Software, vol 29, no. 6, pp34-42, Dec 2012

[4] Curtis,B., Sappidi, J., Szynkarski, "Estimating the size, code and types of Technical Debt", Third International Workshop on Managing Technical Debt 2012, 2012

[5] Curtis, B, "Measuring and Managing Technical Debt", available at http://omg.org/news/meetings/tc/tx-14/special-events/cisq-presentations/CISQ-Seminar-2014-6-17-BILL-CURTIS-Measuring-and-Managing-Technical-Debt.pdf, (Retrieved 3/12/2015)

[6] SWEBOK Version 3 and Guide to SWEBOK available at http://www.computer.org/web/swebok/v3

© 2013 PRICE Systems, LLC All Rights Reserved | *Decades of Cost Management Excellence*