Technical Debt
Sonar Analysis

John Heintz john@gistlabs.com jheintz@cutter.com
Senior Consultant, Cutter Consortium
President, Gist Labs

Israel Gat igat@cutter.com
Director of Agile Practice, Cutter Consortium
About John Heintz

- Developer since 1995
- Agilist since 1999
- Founded Gist Labs in 2008
- Also with Cutter Consortium
- Developer, Mentor, Consultant
- Intuitive, Abstract, Precise

Kool-Aids I’ve drank:

Agile/Lean/Kanban, OO, TDD, REST, Mentoring, Craftsmanship, Emergent/Progressive Design, InnovationGames®, Systems and Complexity Theory
What Really is Technical Debt?

• What’s in a Metaphor?
• Why Worry?
• Code Analysis
• Monetized Technical Debt
• Sonar Project Dashboard
What’s in a Metaphor?

- Ward Cunningham’s Metaphor:
  - “A little debt speeds development so long as it is paid back promptly with a rewrite”

- Definition for today:
  - “Quality issues in the code other than function/feature completeness”
    - It is about doing the system right (“Intrinsic Quality”)
    - Not about doing the right system (“Extrinsic Quality”)

- Typical technical debt components:
  - Complexity
  - Duplication
  - Rule violations
  - Test coverage
  - Documentation
Why Worry About Technical Debt?
Error Prone Modules

- Cyclomatic complexity in excess of ~30 per file for a significant number of Java files

(Source: http://www.enerjy.com/blog/?p=198)
Error Feedback Ratio

Time to Complete Testing

Number of Bugs

EFR=0.3  EFR=0.36  EFR=0.396

Source: Gerald Weinberg, Quality Software Management: System Thinking
The Strategic Effect of Technical Debt

- Once on far right of curve, all choices are hard
- If nothing is done, it just gets worse
- In applications with high technical debt, estimating is nearly impossible
- Only 3 strategies
  - Do nothing, it gets worse
  - Replace, high cost/risk
  - Incremental refactoring, commitment to invest
One technical debt tends to pile over another, which piles over yet another technical debt that piles...

- To find your current level of debt, you can’t simply add the week you borrowed last year to the two weeks you borrowed three months ago.
- Rather, you need to inspect the code.
Monetized Technical Debt

- Accrued technical debt in the amount of $500K
- On 200K lines of code
- The makeup of the debt is represented in the pie chart below

**Breakdown of Technical Debt**

- Test coverage
- Duplication
- Rule violations
- Complexity
Since the 0.4.0 release both Complexity (per class) and Technical Debt have increased.
Project Dashboard

- Lines of code: 162,306
- Classes: 1,447
- Comments: 26.6%
- Duplications: 7.1%
- Complexity: 3.1 / method, 30.9 / class, 42.2 / file, Total: 44,773
- Rules compliance: 83.7%
- Violations: 10,072

- Source: Chris Sterling
- Key: org.apache.tomcat
- Language: java
- Technical Debt: 11.0%