Technical Debt
How Software Organizations Can Stay Solvent

Prof. dr.ir. Paris Avgeriou - paris@cs.rug.nl
Software Engineering and Architecture Group
http://www.cs.rug.nl/~paris/
Rankings - Top 100 university

#80  Times Higher Education Worldwide

#72  Academic Ranking of World Universities

#86  U.S. News ‘Best Global Universities Ranking’

Founded in 1614
Research Philosophy

› Core business: Software Architecture
› With Dutch & European industry (real problems)
  • Embedded Systems & Enterprise Applications
› Automated Software Engineering
› Evidence-based Software Engineering
  • Evidence matters - empirical research methods
Outline

› Introducing the metaphor
› Emergence of TD
› Concepts of TD and management
› Present and Future
“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 ‘92
Technical Debt is a collection of design or implementation constructs* that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible.

Dagstuhl April 2016

* 1. Immature artifacts
2. Postponed tasks
Technical Debt metaphor

› Debt is a necessary tradeoff
  • **Loan** for *investment*
  • **Quality**— for *business value++*
› Pay back *principal* (fix TD) + *interest* (maintain SW)
› Complete payoff may be unrealistic
› Debt should be monitored and managed
  • Risk – accumulation may spiral out of control

› Both a metaphor and a SW Dev artifact
Outline

› Introducing the metaphor
› **Emergence of TD**
› Concepts of TD and management
› Present and Future
For every 100 KLOC an average software application had approx. US$361,000 of technical debt*

Technical Debt
Is this really new?

Communities
› Maintenance & evolution
› Reengineering / refactoring

Terms
› Aging
› Decay
› Sustainability
› Little progress
› “Dull” topic
Convergence of SE disciplines

- Program analysis/comprehension
- SW Quality measurement
- Qualitative research methods
- SW risk management

MTD>sum of parts!
Z. Li et al., A systematic mapping study on technical debt and its management, JSS 2015
› Introducing the metaphor
› Emergence of TD
› **Concepts of TD and management**
› Present and Future
Debt = Principal + Interest

Ampatzoglou et al., A Financial Approach for Managing Interest in TD, BMSD ’15
Vicious circle of technical debt

Business Pressure

Lower Dev Velocity

Incur TD
Ampatzoglou et al., A Financial Approach for Managing Interest in TD, BMSD ‘15
Types of TD

- Reckless
- Prudent
- Deliberate
- Inadvertent
Technical debt is pervasive
Code
Requirements
Architecture
Design
Test
Build
Documentation
Infrastructure
Versioning

Complex dependencies
Architecture smells
Architecture drift
Low code coverage
Lack of test automation
Expensive tests
Residual defects not found
Just the code?

- Code
- Requirements
- Architecture
- Design
- Test
- Build
- Documentation
- Infrastructure
- Versioning

Insufficient/incomplete/out of date
Lack of code comments
Managing TD

› TD prevention  
› TD identification  
› TD measurement  
› TD prioritization  
› TD monitoring  
› TD repayment  
› TD representation/documentaton  
› TD communication
Managing TD

› TD prevention
› TD identification
› TD measurement
› TD prioritization
› TD monitoring
› TD repayment
› TD representation/documentation
› TD communication

Mathematical models
Code metrics
Human estimation
<table>
<thead>
<tr>
<th>ATD item</th>
<th>Benefit of ATD item in Release V16.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATD1</td>
<td></td>
</tr>
<tr>
<td>ATD2</td>
<td></td>
</tr>
<tr>
<td>ATD3</td>
<td></td>
</tr>
<tr>
<td>ATD4</td>
<td></td>
</tr>
<tr>
<td>ATD5</td>
<td></td>
</tr>
<tr>
<td>ATD6</td>
<td></td>
</tr>
<tr>
<td>ATD7</td>
<td></td>
</tr>
<tr>
<td>ATD8</td>
<td></td>
</tr>
<tr>
<td>ATD9</td>
<td></td>
</tr>
<tr>
<td>ATD10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ATD cost amount (Person-day)</th>
<th>ATD benefit amount (Person-day)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>300</td>
</tr>
<tr>
<td>200</td>
<td></td>
</tr>
</tbody>
</table>

**Legend**
- Red: Cost of ATD item in Release V16.1
- Green: Benefit of ATD item in Release V16.1
- Red: Cost of ATD item in Release V16.2
- Green: Benefit of ATD item in Release V16.2

Li et al. *Architecture viewpoints for documenting ATD*, Elsevier 2016
Managing TD

- TD prevention
- TD identification
- TD measurement
- TD prioritization
- TD monitoring
- TD repayment
- TD representation/documentation
- TD communication

TD dashboard
  - Backlog
  - Dependency visualization
  - Code metrics visualization
  - TD propagation visualization
Technical debt in Backlog

<table>
<thead>
<tr>
<th>Positive Value</th>
<th>Visible</th>
<th>Invisible</th>
</tr>
</thead>
<tbody>
<tr>
<td>New features Added functionality</td>
<td>Architectural, structural features</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Negative Value</th>
<th>Visible</th>
<th>Invisible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defects</td>
<td>Technical debt</td>
<td></td>
</tr>
</tbody>
</table>

Philippe Kruchten, Technical debt – From metaphor to theory and practice, IEEE SW2012.
Outline

› Introducing the metaphor
› Emergence of TD
› Concepts of TD and management
› **Present and Future**
› Technical debt grows
› Managing TD is **dominant** in SW evolution
  • Established as a core SE practice
› It’s the next big thing
  ++ Investment
  -- Bankruptcy
State of the art

› From source code to the whole lifecycle
› Glossaries and ontologies (convergent)
› Tooling (industrial & prototypes)
› Economic theories
Breaking the metaphor

- People who collect TD ≠ people who repay TD
- Relating TD to an interest rate or interest period
- TD can be unintentional
- TD does not always have to be repaid
- TD does not necessarily have a bad side
SW engineers
› Understand the concept and challenges
› Deal with it during maintenance
› TD management in place but with constraints
   • Resource-intensive
   • Realistically only a portion managed
SW Engineers don’t like TD

Managers don’t mind TD

Short deadline vs.

Long-term sustainability

Communication bridge

Investment opportunity
Join the community!

› Source code -> architecture
  • Automatic detection of (architecture) smells
› Economic theories for sound investments
  • Business value to intrinsic qualities/refactoring
› Automating identification and measuring
  • Data mining in SW repositories
› Benchmarking
› Teach TD in school
  • Throughout the curriculum
Credits:
Philippe Kruchten
Robert Nord
Ipek Ozkaya
Carolyn Seaman
Zengyang Li
Peng Liang
Areti Ampatzoglou
Apostolos Ampatzoglou
Alexander Chatzigeorgiou