Why you should care about Technical Debt

Prof. dr.ir. Paris Avgeriou - paris@cs.rug.nl
Software Engineering and Architecture Group
http://www.cs.rug.nl/~paris/
The Known Universe
#83  Times Higher Education Worldwide
#59  Academic Ranking of World Universities
#86  U.S. News ‘Best Global Universities Ranking’

Founded in 1614
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
› **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
Technical Debt illustrated

Images from https://refactoring.guru/smells
Technical Debt metaphor

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

- Taking more time to build a feature or fix defects
- Changes ripple through the system
- Rework is often and unexpected
- Deadlines/milestones continuously slipping
- Velocity drops
- Testing becomes very expensive
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*

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

Managing TD > sum of parts!
Z. Li et al., A systematic mapping study on technical debt and its management, JSS 2015
Outline

› Introducing the metaphor
› Emergence of TD
› **Concepts of TD and management**
› Present and Future
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
Ampatzoglou et al., A Financial Approach for Managing Interest in TD, BMSD ‘15
Not quite right
› Code
› Requirements
› Architecture
› Design
› Test
› Build
› Documentation
› Infrastructure
› Versioning

...
Just the code?

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

Complex dependencies
Architecture smells
Architecture drift
Low code coverage
Lack of test automation
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
Although the architectural complex problems only account for 8% of the defects, they absorb 52% of the effort spent in repairing defects.

Bill Curtis, CISQ

https://insights.sei.cmu.edu/sei_blog/2015/07/a-field-study-of-technical-debt.html
Managing TD

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

Li et al., Architectural Debt Management in Value-oriented Architecting, Elsevier ‘14
Managing TD

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

Code analysis
Dependency analysis
Solution comparison
Reverse engineering
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
Managing TD

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

Refactoring
Automating manual tasks
Utility classes should not have public constructors

Utility classes, which are collections of static members, are not meant to be instantiated. Even abstract utility classes, which can be extended, should not have public constructors.

Java adds an implicit public constructor to every class which does not define at least one explicitly. Hence, at least one non-public constructor should be defined.

**Noncompliant Code Example**

```java
class StringUtils { // Noncompliant
    public static String concatenate(String s1, String s2) {
        return s1 + s2;
    }
}
```

**Compliant Solution**

```java
class StringUtils {
    public static String concatenate(String s1, String s2) {
        return s1 + s2;
    }
}
```
Digkas et al., The Evolution of TD in the Apache Ecosystem, ECSA ‘17
Large variation in survivability of issues
  • 10% fixed within the first month
  • 50% in the first year
  • Some take up to ten years
Very few issues types with fixing rate >50%
Duplication and exception handling
  • Frequently encountered
  • Rarely fixed

Digkas et al., How Do Developers Fix Issues and Pay Back TD in the Apache Ecosystem, SANER '18
Outline

› Introducing the metaphor
› Emergence of TD
› Concepts of TD and management
› Present and Future
Short deadline vs. Long-term sustainability

SW Engineers don’t like TD
Managers don’t mind TD

Communication bridge
Investment opportunity
State of the art

› Whole lifecycle but mostly code and design
› Basic concepts are mature
› Tooling (industrial & prototypes)
› Economic theories
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
Interplay between qualities

> Theory: Qualities studied as islands
> Practice: Qualities interplay
  > Run-time vs. design time
> Communities needs to interact
> Interoperability
  > Methods and tools

https://sdk4ed.eu/
### Technical Debt

<table>
<thead>
<tr>
<th>Violation</th>
<th>File</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct this &quot;&amp;&quot; to &quot;&amp;&amp;&quot;.</td>
<td>foo.bar.File1.java</td>
<td>if(errorCode != null &amp; errorDesc != null)</td>
</tr>
<tr>
<td>Change this &quot;try&quot; to a try-with-resources.</td>
<td>bar.foo.File2.java</td>
<td>try { .... }</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Energy

<table>
<thead>
<tr>
<th>Violation</th>
<th>File</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define a constant instead of duplicating this literal N times.</td>
<td>foo.bar.File21.java</td>
<td>tabArray.add(Messages.getString(locale, &quot;NPL&quot;));</td>
</tr>
<tr>
<td>Replace the synchronized class &quot;StringBuffer&quot; by an unsynchronized one such as &quot;StringBuilder&quot;</td>
<td>bar.foo.File22.java</td>
<td>StringBuffer sb = new StringBuffer();</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Security

<table>
<thead>
<tr>
<th>Violation</th>
<th>File</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>'PASSWORD' detected in this expression, review this potentially hardcoded credential.</td>
<td>foo.bar.File301.java</td>
<td>String PARAM_PASSWORD = &quot;Password&quot;;</td>
</tr>
<tr>
<td>Use the recommended AES (Advanced Encryption Standard) instead.</td>
<td>bar.foo.File302.java</td>
<td>Cipher des = Cipher.getInstance(&quot;aes/ecb/NoPadding&quot;);</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
› Bridging the gap between research and practice
› Join efforts
<table>
<thead>
<tr>
<th>Credits:</th>
<th>Zengyang Li</th>
</tr>
</thead>
<tbody>
<tr>
<td>Philippe Kruchten</td>
<td>Peng Liang</td>
</tr>
<tr>
<td>Robert Nord</td>
<td>Areti Ampatzoglou</td>
</tr>
<tr>
<td>Ipek Ozkaya</td>
<td>Apostolos Ampatzoglou</td>
</tr>
<tr>
<td>Carolyn Seaman</td>
<td>Alexander Chatzigeorgiou</td>
</tr>
</tbody>
</table>