Relating Cost Modeling, Incremental Development and Architecture Trade-offs

Ipek Ozkaya, Robert Nord
{ozkaya, rn} sei.cmu.edu

Research, Technology & System Solutions Program
Software Engineering Institute

October 16, 2012
Agenda Today

Value-driven incremental development and architecture
- Balancing cost, value, and waste

Measuring architecture rework as a proxy for short-term versus long-term trade-offs, a.k.a technical debt
- in collaboration with Philippe Kruchten, University of British Columbia and Raghu Sangwan, Penn State

Ongoing work and future directions
Value-Driven Incremental Development
An Architecture-focused view

Assess the impact of
- delivered utility
- cost of delay, rework to determine efficient options for increments.

Cost of over-architecting and unnecessary assurance activities delays capabilities to reach the field.

Ability to adjust course with empirical basis

Observed reduction in delivery tempo triggers evaluation of decisions early.

Accumulated suboptimal architecture and need to wait for assurance impact overall capability to reach the field.
Waste and Software Architecture

Understanding waste can be very informative, bringing visibility to the tradeoff space of:

- the cost of delayed delivery waiting for complete architecture or architecture for unneeded requirements
- and the cost of rework due to an incomplete architecture or discovering defects late.

Where is the “sweet spot”
Quantifying Architectural Value

Should I take on architectural debt to adapt the system for the immediate need for a given story?
Is there an architectural investment that I can make now that will reduce the future cost to implement the story?
  • How likely is it that the future need will arise?
If so, what is the cost of this architectural investment? (e.g., cost to implement, opportunity cost, etc.)
What are the relative economics of meeting the future need with or without having made the prior architectural investment?
  • (e.g., relative cost and time to implement with or without prior architectural investment, potential opportunity cost from delay in meeting the future stakeholder need, etc.)
Technical Debt

“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.”

### Technical Debt – Steve McConnell

<table>
<thead>
<tr>
<th>Type 1</th>
<th>Type 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>unintentional, non-strategic; poor design decisions, poor coding</td>
<td>intentional and strategic: optimize for the present, not for the future.</td>
</tr>
<tr>
<td></td>
<td>2.A short-term: paid off quickly (refactorings, etc.)</td>
</tr>
<tr>
<td></td>
<td>2.B long-term</td>
</tr>
</tbody>
</table>

Implemented features (visible and invisible) = assets = non-debt

*Technical debt is 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.*

Technical Debt Analogy

When and how was the debt signed under?
What is the interest rate?
What is the payback strategy?
Mapping to architecture

Interest = unintended maintenance costs and additional cost of adding features

Payback strategy = Cost of rearchitecting
Calculating rework

Total cost $T$ of release $n$

$$T = F(C_i, Cr)$$

The implementation cost $C_i$ for release $n$ is computed as

$$\sum_k C_i(E_k) \text{ for all new elements } E_k$$

The rework cost for release $n$ is computed as

$$\sum_k Cr(E_k) \text{ for all new elements } E_k$$

and

$$Cr(E_k) = \sum_j Cr(E_j) \text{ for all pre-existing elements } E_j$$

If $E_j$ is an element implemented in a prior release,

$$Cr(E_j) = D(E_j, E_k) \times C_i(E_j) \times Pc(n - 1)$$

where $D(u,v)$ is the number of dependencies from $u$ to $v$, $C_i$ is the implementation cost, and $Pc(n - 1)$ is the change propagation of release $n - 1$. 


Taking on Debt

DR-NEP Community
- Planning
- Training
- Response
- Restoration

First more capabilities

need to monitor technical debt to gain insight into life-cycle efficiency

will the benefits pay off

more infrastructure
## Dependency Analysis of the DRNEP paths

### PATH 1 DELIVER SOON

| Path | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | Cost |
|------|---|---|---|---|---|---|---|---|---|-----|-----|-----|-----|-----|-----|-----|-----|
| Controller | 1 | X |   |   |   |   |   |   |   |     |     |     |     |     |     |     |     | 16  |
| I2SimAdapter | 2 | X | X |   |   |   |   |   |   |     |     |     |     |     |     |     |     | 3   |
| MTAdapter | 3 | X | X |   |   |   |   |   |   |     |     |     |     |     |     |     |     | 3   |
| MT_I2Sim_Trans | 4 | X | X | X |   |   |   |   |   |     |     |     |     |     |     |     |     | 8   |
| EPANetAdapter | 5 | X | X |   |   |   |   |   |   |     |     |     |     |     |     |     |     | 3   |
| Epa_I2Sim_Trans | 6 | X | X | X |   |   |   |   |   |     |     |     |     |     |     |     |     | 8   |
| Epa_MT_Trans | 7 | X | X | X |   |   |   |   |   |     |     |     |     |     |     |     |     | 8   |
| BrahmsAdapter | 8 | X | X |   |   |   |   |   |   |     |     |     |     |     |     |     |     | 3   |
| Brahms_I2Sim_Trans | 9 | X | X | X |   |   |   |   |   |     |     |     |     |     |     |     |     | 8   |
| Brahms_MT_Trans | 10 | X | X | X |   |   |   |   |   |     |     |     |     |     |     |     |     | 8   |
| Brahms_EPANet_Trans | 11 | X | X | X |   |   |   |   |   |     |     |     |     |     |     |     |     | 8   |
| FloodAdapter | 12 | X | X |   |   |   |   |   |   |     |     |     |     |     |     |     |     | 3   |
| Flood_I2Sim_Trans | 13 | X | X | X |   |   |   |   |   |     |     |     |     |     |     |     |     | 8   |
| Flood_MT_Trans | 14 | X | X | X |   |   |   |   |   |     |     |     |     |     |     |     |     | 8   |
| Flood_EPANet_Trans | 15 | X | X | X |   |   |   |   |   |     |     |     |     |     |     |     |     | 8   |
| Flood_Brahms_Trans | 16 | X | X | X |   |   |   |   |   |     |     |     |     |     |     |     |     | 8   |
| Data Persistence | 17 | X | X |   |   |   |   |   |   |     |     |     |     |     |     |     |     | 5   |

**COST** 116

### PATH 2 REDUCE REWORK

<table>
<thead>
<tr>
<th>Path</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2SimService</td>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>MTSERVICE</td>
<td>2</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>EPASERVICE</td>
<td>3</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>BrahmsService</td>
<td>4</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>FloodService</td>
<td>5</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>Web</td>
<td>6</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>WASClient</td>
<td>7</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>Controller</td>
<td>8</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>Service</td>
<td>9</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>ServiceBusiness</td>
<td>10</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>SimulatorServiceClient</td>
<td>11</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>Entities</td>
<td>12</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>Common</td>
<td>13</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

**COST** 94
Results

Path 1: value focused; functionality first.

Path 2: cost focused; architecture push.

<table>
<thead>
<tr>
<th>Path #1</th>
<th>Cumulative value</th>
<th>Release 1</th>
<th>Release 2</th>
<th>Release 3</th>
<th>Release 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of total value</td>
<td>18%</td>
<td>41%</td>
<td>68%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Cost (Ci + Cr)</td>
<td>35</td>
<td>64</td>
<td>101</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>% of total implementation cost</td>
<td>37%</td>
<td>68%</td>
<td>108%</td>
<td>155%</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Path #2</th>
<th>Cumulative value</th>
<th>Release 1</th>
<th>Release 2</th>
<th>Release 3</th>
<th>Release 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of total value</td>
<td>18%</td>
<td>41%</td>
<td>68%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Cost (Ci + Cr)</td>
<td>67</td>
<td>76</td>
<td>85</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>% of total implementation cost</td>
<td>71%</td>
<td>81%</td>
<td>90%</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>
Creating a payback strategy

Eliciting debt and quantifying the impact is not a repeatable engineering practice yet.

- Factors to consider include defects, velocity, cost of rework
- Mapping such indicators onto cost of development
- Comparing with the value of paying back debt versus not

Analysis tools, mostly looking at code metrics, provide quality insights
Metrics for Quantifying Architecture Quality

Challenges

• Insufficient and unproven metrics for quantifying architecture quality to guide the re-architecting process.

• Code-level refactoring techniques do not scale effectively to support architecture-level evaluation for re-architecting.

There has been an increasing focus on tools for the purpose of structural analysis:

• increasing sophistication,

• support for some structural analysis in addition to code analysis,

• first steps towards analyzing financial impact by relating structure analysis to cost and effort for rework.
Quality Metrics

Achilles Heel

- The algorithm for rework is directional and relative rather than absolute
- The cost of each architectural element, the number of dependencies impacted by each architectural change, and the overall change propagation metric of the system as proxies for complexity
- There is a reliance on project management data with estimates
- Propagation cost metric does not reflect the strength of a dependency and is too sensitive to false positives in small systems
Sensitivity Analysis

Can we identify propagation cost patterns with known evolution patterns
1. SOA-like
2. Strict Layering
3. Dependency inversion
4. Short circuit
5. Module splitting
Propagation cost

“Density” of the DSM
• Proposed by McCormack et al. in 2006
• Several limitations as a tool to measure T.D.

Improved PC:
• Boolean to continuous value (dependency strength)
• Changes not uniformly spread throughout the code
• Less sensitive to size of code
Example of PC: Evolution of Ant

Technical debt reduction
Future Directions in –
Open Areas of Investigation

The challenge in large-scale, long-term projects is balancing the needs of today with those of tomorrow:

• There is an optimization problem where optimizing for the short-term puts the long-term into economic and technical jeopardy
• Design shortcuts can give the perception of success until their consequences slow down projects.
• Software development decisions, especially architectural ones, need to be continuously analyzed and actively managed as they incur cost, value, and debt.
GUTSI
Grand Unified Theory of Software Improvement
References

• Nord, R., I. Ozkaya, P. Kruchten, M. Gonzalez: In Search of a Metric for Managing Architectural Technical Debt. WICSA 2012
NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this presentation is not intended in any way to infringe on the rights of the trademark holder.

This Presentation may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.