Agility and Architecture

or: What colours is your backlog?

Philippe Kruchten
July 7, 2011
Outline

1. The frog and the octopus
2. Architecture and agility
3. Release planning
4. Technical debt
5. Architecture, agility,… revisited

A Conceptual Model of Software Development

4 key concepts, 5 key attributes

- Intent
- Product
- Work
- People
- Time
- Quality
- Risk
- Value
- Cost
Intent, Work, People, Product

Intent ➔ realize ➔ represent ➔ implement

Work ➔ drive ➔ produce ➔ isAllocatedTo ➔ execute

People

Frog: “All projects are the same”

Intent ➔ Time ➔ Value

Product ➔ Time ➔ Value

Work ➔ Time ➔ Cost

People ➔ Time ➔ Cost
Octopus: “All projects are different!”

Context
- Size
- Criticality
- Business model
- Team architecture
- Stable architecture
- Team distribution
- Governance
- Corporate & National Culture
- Degree of Innovation
- Rate of change
- Age of the system
- Domain, Industry

Project environment, customer, end-users, competition, legacy, business

Delivered Product

Defects
Enhancements

Wishes, needs, constraints

Legal and Regulatory constraints

Technologies
SW Dev. Project:
Tension between Intent and Product

Intent

Product

Work

Iterations

Time

Copyright © 2011 Philippe Kruchten
Outline

1. The frog and the octopus
2. Architecture and agility
3. Release planning
4. Technical debt
5. Architecture, agility,... revisited

Agile & Architecture? Oil & Water?

• Paradox
• Oxymoron
• Conflict
• Incompatibility
What is Agility?

• Jim Highsmith (2002):
  – Agility is the ability to both create and respond to change in order to profit in a turbulent business environment.

• Sanjiv Augustine (2004):
  – Iterative and incremental
  – Small release
  – Collocation
  – Release plan/ feature backlog
  – Iteration plan/task backlog

Getting at the Essence of Agility

• Software development is a knowledge activity
  – Not production, manufacturing, administration...
• The “machines” are humans
• Dealing with uncertainty, unknowns, fear, distrust
• Feedback loops ➔
  – reflect on business, requirements, risks, process, people, technology
• Communication and collaboration
  – Building trust ➔ rely on tacit information ➔ reduce waste
Software Architecture: A Definition

“It’s the hard stuff.”
“It’s the stuff that will be hard to change”

M. Fowler, cited by J. Highsmith

ISO/IEC 42010

Architecture: the fundamental concepts or properties of a system in its environment embodied in its elements, their relationships, and in the principles of its design and evolution
Software Architecture

Software architecture encompasses the set of significant decisions about

• the organization of a software system,
• the selection of the structural elements and their interfaces by which the system is composed together with their behavior as specified in the collaboration among those elements,
• the composition of these elements into progressively larger subsystems,

Grady Booch, Philippe Kruchten, Rich Reitman, Kurt Bittner; Rational, circa 1995 (derived from Mary Shaw)

Software Architecture (cont.)

... 

• the architectural style that guides this organization, these elements and their interfaces, their collaborations, and their composition.

• Software architecture is not only concerned with structure and behavior, but also with usage, functionality, performance, resilience, reuse, comprehensibility, economic and technological constraints and tradeoffs, and aesthetics.
Perceived Tensions
Agility- Architecture

- Architecture = Big Up-Front Design
- Architecture = massive documentation
- Architects dictate form their ivory tower
- Low perceived or visible value of architecture
- Loss of rigour, focus on details
- Disenfranchisement
- Quality attribute not reducible to stories

Hazrati, 2008
Rendell, 2009
Blair et al. 2010, etc.

Adaptation versus Anticipation

Highsmith 2000
Issues

1. Semantics
2. Scope
3. Lifecycle
4. Role
5. Description
6. Methods
7. Value & cost

Semantics

• What do we mean by “architecture”?

• What do we mean by “software architecture”?
Issues

1. Semantics
2. Scope
3. Lifecycle
4. Role
5. Description
6. Methods
7. Value & cost

Scope

• How much architecture “stuff” do you really need?

• It depends...

• It depends on your context
Context attributes

1. Size
2. Criticality
3. Age of system
4. Rate of change
5. Business model
6. Domain
7. Team distribution
8. Governance

All software-intensive systems have an architecture

- How much effort should you put into it varies greatly
- 75% of the time, the architecture is implicit
  - Choice of technology, platform
  - Still need to understand the architecture
- Novel systems:
  - Much more effort in creating and validating an architecture
- Key drivers are mostly non-functional:
  - Runtime: Capacity, performance, availability, security
  - Non runtime: evolvability, regulatory, i18n/L10n…
Lifecycle

• When does architectural activities take place?
• The evil of “BUFD” = Big Up-Front Design
• “Defer decisions to the last responsible moment”
• YAGNI = You Ain’t Gonna Need It
• Refactor!

Architectural Effort During the Lifecycle

Majority of architectural design activities
Little dedicated architectural effort

![Diagram showing Inception, Construction, and Transition phases with minimal pure architectural activities and ideal realm of agile practices]

Iterations and Phases

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
</table>

- **Iterations with focus on architecture**
- **Iterations with main focus on features**

An **architectural iteration** focuses in putting in place major architectural elements, resulting in a baseline architectural prototype at the end of elaboration.
Team Structure over Time (Very Large)

Inception | Elaboration | Construction and Transition
--- | --- | ---
Management team | Architecture team | Management team
Architecture team | Feature team 1 | Architecture team
Prototyping team | Feature team 2 | Feature team 3
Management team | Infrastructure team A | Infrastructure team B
Prototyping team | Integration team

Teams using agile development practices

Inception | Elaboration | Construction and Transition
--- | --- | ---
Management team | Architecture team | Management team
Architecture team | Feature team 1 | Architecture team
Prototyping team | Feature team 2 | Feature team 3
Management team | Infrastructure team A | Infrastructure team B
Prototyping team | Integration team
Issues

1. Semantics
2. Scope
3. Lifecycle
4. Role
5. Description
6. Methods
7. Value & cost

New Role – Agile Architect?

- A. Johnston defines the agile architect, but it does not seem to be any different from a software architect before agile methods came in.
- Combination of
  - Visionary - Shaper
  - Designer – making choices
  - Communicator – between multiple parties
  - Troubleshooter
  - Herald – window of the project
  - Janitor – cleaning up behind the PM and the developers
Functions of the software architect

**Definition of the architecture**
- Architecture definition
- Technology selection
- Architectural evaluation
- Management of non-functional requirements
- Architecture collaboration

**Delivery of the architecture**
- Ownership of the big picture
- Leadership
- Coaching and mentoring
- Design, development and Testing
- Quality assurance

---

Two styles of software/system architects

- **Maker and Keeper of Big decisions**
  - Bring in technological changes
  - External collaboration
  - More requirements-facing
  - Gatekeeper
  - *Fowler: Architectus reloadus*

- **Mentor, Troubleshooter, and Prototyper**
  - Implements and try architecture
  - Intense internal collaboration
  - More code-facing
  - *Fowler: Architectus oryzus*

---

Only big new projects need both or separate people
A. Reloadus and A. Oryzus ecological niches

Role

Architecture owner
Issues

1. Semantics
2. Scope
3. Lifecycle
4. Role
5. Description
6. Methods
7. Value & cost

Architectural description

- Metaphor (XP)
- Prototype
- Software architecture document

- Use of UML?
- UML-based tools?
- Code?
UML 2.0

- A notation
- Better “box and arrows”
- Crisper semantics
- Almost an ADL?

- Model-driven design,
- Model-driven architecture.
Again, it depends on the context

1. Size
2. Criticality
3. Age of system
4. Rate of change
5. Business model
6. Stable architecture
7. Team distribution
8. Governance

Adaptation versus Anticipation

Highsmith 2000
Issues

1. Semantics
2. Scope
3. Lifecycle
4. Role
5. Description
6. Methods
7. Value & cost

Architectural design methods

- Many agile developers do not know (much) about architectural design
- Agile methods have no explicit guidance for architecture
  - Metaphor in XP
  - Technical activities in scrum
- Relate this to Semantics and Scope issue
- May have to get above the code level
Architectural Methods

• ADD, ATAM, QAW (SEI)
• RUP (IBM)
• SAV,… (Siemens)
• BAPO/CAFR (Philips)
• Etc. ....

• Software Architecture Review and Assessment (SARA) handbook

Architectural Design

Source: Hofmeister, Kruchten, et al., 2005, 2007
Iterative Architecture Refinement

- There are no fixed prescriptions for systematically deriving architecture from requirements; there are only guidelines.
- Architecture designs can be reviewed.
- Architectural prototypes can be thoroughly tested.
- Iterative refinement is the only feasible approach to developing architectures for complex systems.

Issues

1. Semantics
2. Scope
3. Lifecycle
4. Role
5. Description
6. Methods
7. Value & cost
Value and Cost

- Value: to the business (the users, the customers, the public, etc.)
- Cost: to design, develop, manufacture, deploy, maintain

- Simple system, stable architecture, many small features:
  - Statistically value aligns to cost
- Large, complex, novel systems?

Outline

1. The frog and the octopus
2. Architecture and agility
3. Release planning
4. Technical debt
5. Architecture, agility,... revisited
Sprint Planning

Feature Requests

Priority

Sprint Backlog

Time-box

Staff

Work (≈Cost)

Time

Copyright © 2011 Philippe Kruchten
Time-boxes: Releases

Release 1  R2  R3  R4

Time

Time-boxes: Iterations (sprints)

Release N

ITERATION 1  It. 2  It. 3

Time
Features

Work and Cost

• How much work is associated to a feature?
• Work is strongly related to cost in software development (a human-intensive activity)
• Overall budget is roughly the size of the time-box(es)
• Time-box = budget
• Features must fit in budget
• Q: How do we select what goes in the box?
Maximizing value

Highest value first
Ignore time

Value = Cost?

Only for simplest cases
Value and Cost

- Value: to the business (the users, the customers, the public, etc.)
- Cost: to design, develop, manufacture, deploy, maintain

- Simple system, stable architecture, many small features:
  - Statistically value aligns to cost
- Large, complex, novel systems?
Efficiency vs. Effectiveness

**Efficiency**
- relationship between the output in terms of goods, services or other results and the resources used to produce them

**Effectiveness**
- relationship between the intended impact and the actual impact of an activity
“Spent-Cost” System

Invisible Features

Copyright © 2011 Philippe Kruchten
Invisible Features

- Architecture
- Infrastructure
- Common elements
- Framework
- Libraries
- Reuse
- DSL
- *Product line*
Release Planning

- Time-box = budget
- Fill the time-box with a combination of visible and invisible features
- ... while maximizing value

- Easy, no?
Tension

• Product manager: maximize value (green stuff)

• Project manager: maximize budget utilization
  — i.e., minimize cost

• Techie: maximize the fun stuff (yellow) ?

Value for the yellow stuff:
Heuristics

• Value of invisible feature = Max (value of all dependents)
• Value of invisible feature = Max + f(number of dependents)
• Value of invisible feature = total value achievable if implementing it – total value achievable without implementing it
• ...
  (Not there yet, more research need to happen)
More on value & cost

- CBAM = Cost Benefit Analysis Method
  - Chap 12 in Bass, Clements, Kazman 2003
- IMF: Incremental Funding Method
  - Denne & Cleland-Huang, 2004
    *Software by numbers*
- Evolve* - Hybrid
  - Günther Ruhe & D. Greer 2003, etc...

IFM: Incremental Funding Method

- MMF = Minimum Marketable Features
- AE = Architectural elements
- Cost
- MMF depends on AE
- Time and NPV = Net Present Value
- Strands = Sequences of dependent MMFs
- Heuristic
  
  Denne & Huang,  www.softwarebynumbers.org
Visible & Invisible Features

Estimation

- Cost estimation
- Work
- Estimate
  - Ideal case?
    - Things go wrong
  - Worse case?
    - $\sum$ all worse cases = impossible implementation
Buffers

- E. Goldratt: *Theory of constraints*
- D. Anderson: *Agile Project Management*

- Buffer: unallocated effort (work)
- Shared by all staff members and all explicit work

Time-box with Buffer
Defects

• Defect = Feature with negative value

• Fixing a defect has a positive cost (work)

• Time/place of discovery
  – Inside development (in-house, in process)
  – Outside development (out-house?) in a released product (escaped defects)

Escaped Defect has Value

Perfect product

Imperfect product

Defect
Fixing a Defect has a Cost

- Defects have both value and cost
- Value of fixing a defect = −Value of the defect
- Cost of fixing a defect (estimated, actual)

- Defects have dependencies
  - Defect fix depend on invisible feature
  - Visible feature depending on a fix
Outline

1. The frog and the octopus
2. Architecture and agility
3. Release planning
4. Technical debt
5. Architecture, agility,... revisited

Technical Debt

- Concept introduced by Ward Cunningham
- Often mentioned, rarely studied
- All experienced SW developers “feel” it.
- Drags long-lived projects and products down
Origin of the metaphor

• Ward Cunningham, at OOPSLA 1992

“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.”

Cunningham, OOPSLA 1992

Technical Debt (S. McConnell)

• Implemented features (visible and invisible) = assets = non-debt
• Type 1: unintentional, non-strategic; poor design decisions, poor coding
• Type 2: intentional and strategic: optimize for the present, not for the future.
  – 2.A short-term: paid off quickly (refactorings, etc.)
    • Large chunks: easy to track
    • Many small bits: cannot track
  – 2.B long-term

McConnell 2007
Technical Debt (M. Fowler)

Reckless
“We don’t have time for design”

Prudent
“We must ship now and deal with consequences”

Deliberate

Inadvertent
“Now we know how we should have done it”

Fowler 2009, 2010

Example

First more capabilities

Then, more infrastructure

neglected cost of delay to market

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

underestimated re-architecting costs

Ozkaya, SEI, 2010

First more infrastructure
Then, more capabilities
Technical Debt (Chris Sterling)

• Technical Debt: issues found in the code that will affect future development but not those dealing with feature completeness.

Or

• Technical Debt is the decay of component and intercomponent behaviour when the application functionality meets a minimum standard of satisfaction for the customer.

Time is Money (I. Gat)

• Time is money:
  Think of the amount of money the borrowed time represents – the grand total required to eliminate all issues found in the code
TD is the sum of...

- Code smells 167 person days
- Missing test 298 person days
- Design 670 person days
- Documentation 67 person days

**Totals**
- Work 1,202 person x days
- Cost $577,000

Israel Gat, 2010
Tech Debt (Jim Highsmith)

- 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
  1. Do nothing, it gets worse
  2. Replace, high cost/risk
  3. Incremental refactoring, commitment to invest

Copyright © 2011 Philippe Kruchten

Technical Debt (1)

<table>
<thead>
<tr>
<th></th>
<th>Cost of Change (CoC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>$15, $16, $18</td>
</tr>
<tr>
<td>12</td>
<td>$5, $3</td>
</tr>
<tr>
<td>12</td>
<td>$20, $19, $18</td>
</tr>
</tbody>
</table>

Copyright © 2011 Philippe Kruchten
Technical Debt

• Defect = Visible feature with negative value
• Technical debt = Invisible “feature” with negative value

• Cost .... of fixing
• Value .... of repaying technical debt ???

“Interests” ?

• In presence of technical debt:
  Cost of adding new features is higher

• When repaying (fixing), additional cost for retrofitting already implemented features

• Technical debt not repaid => lead to increased cost, forever
• Cost of fixing increases over time
**TD and Real Options**

NPV \( P_1 \) = \(-2\text{M} + 0.5\times4\text{M} + 0.5\times1\text{M} = 0.5\text{M} \)

Source: K. Sullivan, 2010 at TD Workshop SEI 6/2-3

**TD and Real Options (2)**

NPV \( P_2 \) = \(-1\text{M} + 0.5\times3\text{M} + 0.5\times1\text{M} = 1\text{M} \)

Taking Technical Debt has increased system value.

Source: K. Sullivan, 2010
TD and Real Options (3)

\[ \text{NPV}(P_3) = -1\text{M} + 0.67 \times 2.5\text{M} + 0.33 \times 1\text{M} = 1\text{M} \]

More realistically:
Debt + interest
High chances of success

TD and Real Options (4)

Not debt really, but options with different values...
Do we want to invest in architecture, in test, etc...

Source: K. Sullivan, 2010
• Technical debt is more a rhetorical category than a technical or ontological category.
  – The concept resonates well with the development community and the business community
  – Both sides “get” the metaphor.

• Technical debt is a concept that bridges the gap between:
  – Business decisions makers
  – Software designers/developers
Colours in your Backlog

- **Visible**
  - Positive Value: Visible Feature
  - Negative Value: Visible defect

- **Invisible**
  - Hidden, architectural feature
  - Technical Debt

• **YAGNI** = You Ain’t Gonna Need It
  – But when you do, it is technical debt
  – Technical debt often is the accumulation of too many YAGNI decisions

• Again the tension between the yellow stuff and the green stuff.
Visible & Invisible
Features + Defects fixing
+ Technical Debt payment

Time and depreciation
Net Present Value

\[
\text{Net Present Value (NPV)} = \sum_{t=1}^{T} \frac{\text{Cash Flow}_t}{(1 + i)^t} - \text{Initial Cash Investment}
\]

- \( t = \text{Cash Flow Period} \)
- \( i = \text{Interest Rate Assumption} \)

Value decreases

<table>
<thead>
<tr>
<th>Time</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td>7.5</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

Copyright © 2011 Philippe Kruchten
Technical debt: increase (?)

### 1.0 RCI - Release Candidate 1

<table>
<thead>
<tr>
<th>Ticket</th>
<th>Summary</th>
<th>Cost</th>
<th>Value</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>#54</td>
<td>Switch to email-based usernames</td>
<td>High Cost</td>
<td>High Value</td>
<td>joshwagamen</td>
</tr>
<tr>
<td>#54</td>
<td>Who's online listing</td>
<td>Medium Cost</td>
<td>High Value</td>
<td>francos</td>
</tr>
<tr>
<td>#54</td>
<td>National office content</td>
<td>Medium Cost</td>
<td>High Value</td>
<td>francos</td>
</tr>
<tr>
<td>#55</td>
<td>Multiple levels of membership in a group</td>
<td>High Cost</td>
<td>Medium Value</td>
<td>Medium Value</td>
</tr>
<tr>
<td>#56</td>
<td>Suggested communities</td>
<td>Medium Cost</td>
<td>Medium Value</td>
<td>Medium Value</td>
</tr>
<tr>
<td>#56</td>
<td>Notifications for group invitations / requests</td>
<td>Low Cost</td>
<td>Medium Value</td>
<td>Medium Value</td>
</tr>
</tbody>
</table>

### None

<table>
<thead>
<tr>
<th>Ticket</th>
<th>Summary</th>
<th>Cost</th>
<th>Value</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>#56</td>
<td>Fix and forget ill for signing up email addresses to the main list</td>
<td>Medium Cost</td>
<td>High Value</td>
<td>Medium Value</td>
</tr>
<tr>
<td>#57</td>
<td>Topic creation preview</td>
<td>Medium Cost</td>
<td>High Value</td>
<td>Medium Value</td>
</tr>
<tr>
<td>#58</td>
<td>Feedback system</td>
<td>Low Cost</td>
<td>High Value</td>
<td>Medium Value</td>
</tr>
<tr>
<td>#59</td>
<td>Test many messages</td>
<td>Low Cost</td>
<td>High Value</td>
<td>Medium Value</td>
</tr>
<tr>
<td>#70</td>
<td>Clicking on a tag causes an error</td>
<td>Low Cost</td>
<td>High Value</td>
<td>Medium Value</td>
</tr>
<tr>
<td>#80</td>
<td>Private messaging to input users</td>
<td>Medium Cost</td>
<td>High Value</td>
<td>Medium Value</td>
</tr>
<tr>
<td>#81</td>
<td>Password strength check</td>
<td>Medium Cost</td>
<td>High Value</td>
<td>Medium Value</td>
</tr>
</tbody>
</table>

Copyright © 2011 Philippe Kruchten
Risks & Uncertainties

Rule of thumb:

- Green stuff: move up
  - defer
- Yellow stuff: move down
  - Experiment now

Karl Wiegers, 1999
RUP, 1998
Outline

1. The frog and the octopus
2. Architecture and agility
3. Release planning
4. Technical debt
5. Architecture, agility, ... revisited

What would Frog say?

```
<table>
<thead>
<tr>
<th>Intent</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td>Quality</td>
<td>Quality</td>
</tr>
<tr>
<td>Risk</td>
<td>Risk</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Work</th>
<th>People</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td>Quality</td>
<td>Quality</td>
</tr>
<tr>
<td>Risk</td>
<td>Risk</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost</td>
<td>Cost</td>
</tr>
</tbody>
</table>
```
What would Octopus say?

- Domain, Industry
- Age of the system
- Criticality
- Degree of Innovation
- Rate of change
- Business model
- Corporate & National Culture
- Governance
- Team distribution
- Stable architecture
- Organizational Maturity

Architecture: Value and Cost

- Architecture has no (or little) externally visible “customer value”
- Iteration planning (backlog) is driven solely by “customer value”
- YAGNI, BUFD, Metaphor…
- “Last responsible moment!” & Refactor!
- Ergo: architectural activities are not given proper attention
- Ergo: large technical debts
Role of Architecture

- Novel system
- Gradual emergence of architecture
- Validation of architecture with actual functionality
- Early enough to support development

Zipper model...
- Not just BUFD
- No YAGNI effect

Planning

- From requirements derive:
  - Architectural requirements
  - Functional requirements
- Establish
  - Dependencies
  - Cost
- Plan interleaving:
  - Functional increments
  - Architectural increments
Zipper: Weaving the functional and architectural work items

Benefits

• Gradual emergence of architecture
• Validation of architecture with actual functionality
• Early enough to support development
• Not just BUFD
• No YAGNI effect
Suggestions for project management

• Separate the processes for estimation of cost and value
• Avoid monetary value (points & utils)
• Identify invisible features and make them more visible to more stakeholders
• Allocate value to invisible feature
• Use nominal and worse case estimates for cost (effort); create shared buffers

Suggestions (cont.)

• Manage all elements together

• Make technical debt visible
  – Large chunks (McConnell type 2)
• Assign some value to technical debt type 2.B and include in backlog

• Exploit different type of buffers
Manage all colours in your backlog!

3 Kinds of Buffers

To care of:
- Defect correction
- Debt repayment
- Estimate uncertainties
- Straightforward work
Toward the 1st release

A later release
Adaptation versus Anticipation

Four take-aways

• Put it in context

$ vs $ • Distinguish value and cost

• Define an “Architecture owner”

• Expose & manage technical debt
Game to Introduce Tech Debt

www.sei.cmu.edu/architecture/tools/hardchoices/

Mission to mars
A Release Planning game
philippe.kruchten.com/mtm

with J. King
SoftEd, Aus.
Starting with software architecture


Agility & architecture

References (1)


References (2)

References (3)


References (4)