Speed and Innovation through Architecture

Jan Bosch
Professor of Software Engineering
Chalmers University of Technology
Gothenburg, Sweden.
www.janbosch.com

November 2011
Landelijk Architectuur Congres 2011
“If you are not moving at the speed of the marketplace you’re already dead – you just haven’t stopped breathing yet”

Jack Welch
Three Key Take-Aways

- Increasing **SPEED** trumps ANY other improvement R&D can provide to the company – the goal is **continuous deployment** of new functionality.

- Software engineering is at an **inflection point** – from “integration-oriented” to “composition-oriented” software engineering.

- **Software architecture** is central in allowing for independent, continuous deployment to customers.
Overview

- Vem är jag? Wie ben ik? Who am I?
- Trends in Software: Need for Speed
- Innovation Experiment Systems
- Architecture & Scale
- Does this apply to me?
- Conclusion
From Research to Industry

- Open Innovation (Intuit, USA)
- Engineering Process (Intuit, USA)
- Head of research lab (Nokia, Finland)
- Professor of software engineering (RuG, Netherlands) (BIT, Sweden)

Innovation
Industrial development
Industrial research
Academia (+ consulting)
Software Center @ Chalmers

- **Mission:** Improve the software engineering capability of the Swedish Software-Intensive Industry with an order of magnitude
- **Theme:** Fast, continuous deployment of customer value
- **Founding members**
  - **ERICSSON**
  - **VOLVO**
  - **SAAB**

- **Dual success metrics**
  - Academic excellence
  - Tangible industrial impact
Overview

- Vem är jag? Wie ben ik? Who am I?
- **Trends in Software: Need for Speed**
- Innovation Experiment Systems
- Architecture & Scale
- Does this apply to me?
- Conclusion
Trend: Products to Services
Trend: Capitalism 3.0
Trend: Need for Speed

Value Creation Shifts

Emerging companies highlight importance of user contribution and social connectedness

<table>
<thead>
<tr>
<th>Founded</th>
<th>1984</th>
<th>1995</th>
<th>2004</th>
</tr>
</thead>
<tbody>
<tr>
<td>1M users</td>
<td>~6 years</td>
<td>30 months</td>
<td>10 months</td>
</tr>
<tr>
<td>50M users</td>
<td>N/A</td>
<td>~80 months</td>
<td>~44 months</td>
</tr>
</tbody>
</table>
Need for Speed in R&D – An Example

- Company X: R&D is 10% of revenue, e.g. 100M$ for a 1B$ product
- New product development cycle: 12 months

- Alternative 1: improve efficiency of development with 10%
  - 10 M$ reduction in development cost
- Alternative 2: reduce development cycle with 10%
  - 100M$ add to top line revenue (product starts to sell 1.2 months earlier)

No efficiency improvement will outperform cycle time reduction
Traditional Software Engineering

- software product lines
- global software development
- software ecosystems

causing

unacceptable complexity and coordination cost
Need for Speed - Principles

Team
- 2 pizza’s
- self-selected, directed and managed
- quantitative output metrics

Architecture
- simplicity – 3 API rule
- backward compatibility – no versions!
- focus on compositionality

Release process
- continuous, independent deployment
- all the way to customers – installed base
- measure usage to feed back into development
Overview

• Vem är jag? Wie ben ik? Who am I?
• Trends in Software: Need for Speed
• Innovation Experiment Systems
• Architecture & Scale
• Does this apply to me?
• Conclusion
What Do These Product Have in Common?
## Example: Apple

<table>
<thead>
<tr>
<th>The Myth</th>
<th>The Reality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inspired innovation</td>
<td>Create and winnow 10 pixel-perfect prototypes</td>
</tr>
<tr>
<td>Inspired design</td>
<td>Build a better backstory (intricate layers of business design behind the products)</td>
</tr>
<tr>
<td>Brilliantly inspired marketing</td>
<td>Engineer the perfect customer experience to create customer experience and buzz</td>
</tr>
</tbody>
</table>

R&D as an Experiment System

Learning: the company running the most experiments against the lowest cost per experiment wins

Goal: increase the number of experiments (with customers) with an order of magnitude to ultimately accelerate organic growth

Three types of functionality
- Customer-requested
- Strategy driven
- Experiments

Decisions should be based on DATA, not opinions
Stairway To Heaven

- R&D as an Experiment System
- Continuous Deployment
- Continuous Integration
- R&D All Agile
- Some Agile Teams
- Traditional Development
## Techniques

<table>
<thead>
<tr>
<th>Pre-Development</th>
<th>Development</th>
<th>Evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASES testing</td>
<td>Independently deployed extensions</td>
<td>Random selection of versions</td>
</tr>
<tr>
<td>Advertising</td>
<td>Feature alpha</td>
<td>Instrumentation of usage metrics</td>
</tr>
<tr>
<td>Solution jams</td>
<td>Product alpha</td>
<td>Surveys</td>
</tr>
<tr>
<td>Mock-ups</td>
<td>Labs website</td>
<td>Ethnographic studies</td>
</tr>
<tr>
<td></td>
<td>Product beta</td>
<td></td>
</tr>
</tbody>
</table>
Overview

• Vem är jag? Wie ben ik? Who am I?
• Trends in Software: Need for Speed
• Innovation Experiment Systems
• Architecture & Scale
• Does this apply to me?
• Conclusion
Role of Software Architecture

- Simplify, Simplify, Simplify
- Decoupling
  - Components
  - Teams
  - Organizations
- Lean and agile at scale
- End to end quality requirements
- Fight design erosion
Simplify, Simplify, Simplify

• Each architectural design decision adds design rules and constraints that cause complexity

• Insist on simplicity (3 APIs rule)

• How
  • Push down in the stack
  • Hide
  • Automate
  • Redesign
Decouple Teams and Organizations

- Interconnected teams and organizations asymptotically reduce productivity to zero

- Decouple teams and make sure no continuous interaction is needed

- How
  - Continuous deployment
  - No versions
  - No concurrent development
Decoupling: No Versions!

- Shared Component, e.g. Engine, etc.
  - Provided interface (SOA style, maximal decoupling)
  - Configuration interface
  - Automated test suites for each interface
  - No offering or shared component may depend on the implementation

- Frequent (4 week) releases of production quality component

- Shared Component, e.g. Engine, etc.
  - Required interface
  - Respect Independent Deployment: still usable in context where this interface can not be bound

Syntactically and semantically equivalent until a deliberate sunset is planned.
Decouple Components and Teams

1. Sequential feature development (90%)
2. Concurrent development, independent deployment enforced (8%)
3. Exploratory development (2%)

Independent deployment!!!
Strive For Continuous Deployment

• Software engineer checks in code => system compiles, links, tests and deploys the new code

• The automated QA infrastructure, NOT the engineer, is responsible for making sure the system does not go down

• If that’s too much, aim for Independent Deployment

• If that’s too much, aim for Release Trains
Lean and Agile at Scale

- Achieving lean & agile in large, legacy systems with large R&D organization considered an oxymoron
- Google, Amazon and Intuit are examples that it can be done
- How
  - Small teams
  - Short cycles
  - Direct customer connection
  - Clear success metrics
End to End Quality Requirements

- Functionality
- Usability
- Performance
- Reliability
- Efficiency
- Scalability
- Extensibility
- Security
- Maintainability
Evolve Architecture; Fight Erosion
Overview

• Vem är jag? Wie ben ik? Who am I?
• Trends in Software: Need for Speed
• Innovation Experiment Systems
• Architecture & Scale

• Does this apply to me?
• Conclusion
Implications

• Go where the money is: “after-market sales” & services

• Disrupt your industries business model

• Develop the ability of your products to evolve constantly

• Minimize R&D investment between customer proof points (data, not opinions!)
Shadow Beliefs

• Humans are better than machines in identifying known and new reliability issues – we are building five 9 systems, after all!

  My experience: data always trumps opinion; test and validation systems pre-deployment and extensive data-collection post-deployment inform decision making

• Embedded systems (large, complex, tough requirements) are different and approaches from other domains do not apply

  My experience: system failure is devastating in several industries and avoided in Internet systems while adopting agile and continuous deployment

• We should avoid or delay adoption of new, more efficient engineering approaches

  My experience: getting first to market with new functionality that closely aligns to customer needs is a significant competitive advantage that drives growth and results in market leadership
Guidelines

1. Modularize the system in critical and (less or) not critical parts
2. Adopt agile and continuous deployment approaches for the not (so) critical part first
3. Deeply engage with customers to develop optimal solutions to their real pain points
4. Invest in testing infrastructure that continuously and thoroughly tests systems with no human involvement
5. Instrument systems for pre- and post-deployment data collection, concerning at least reliability and usage metrics
6. Architect your systems for maximum decoupling and modularization between different components to allow for independent deployment
7. Replace commoditizing functionality with Open Source or COTS components; focus R&D on truly differentiating parts
Overview

• Vem är jag? Wie ben ik? Who am I?
• Trends in Software: Need for Speed
• Innovation Experiment Systems
• Architecture & Scale
• Does this apply to me?

• Conclusion
Speed

Increasing **SPEED** trumps **ANY** other improvement R&D can provide to the company – the goal is **continuous deployment** of new functionality

- If you’re not a front-line engineer, there is only ONE measure that justifies your existence: how have you helped teams move faster?
- Don’t optimize efficiency, optimize speed
Software engineering is at an inflection point – from “integration-oriented” to “composition-oriented” software engineering

- Design for automated compositionality, not manual integration
- Minimize dependencies
- Focus on small teams of engineers, give them direction and get out of their way
Software Architecture 2.0

- **Software architecture** is central in allowing for independent, continuous deployment to customers

- Architecture happens (in parallel)
- A la Thoreau: Simplify, Simplify, Simplify
- Decouple components, decouple teams and decouple organizations
- Lean and agile at scale
Not My Job?!

Strong LEADERSHIP needed from YOU
Thank you!