Towards Compositional Software Engineering

Jan Bosch
VP, Engineering Process
Professor of Software Engineering

September 22nd, 2010
“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 – it is the foundation for everything else

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

• In a world of **continuous deployment** and **software ecosystems**, automation is fundamental
Overview

• Vem är jag? Wie ben ik? Who am I?
  • Introducing Intuit
  • Speed matters: implications for software engineering
  • Building delightful products
  • Software ecosystems
  • Implications for software engineering automation
  • Conclusion
From Research to Industry

Engineering Process (Intuit, USA)

Head of research lab (Nokia, Finland)

Professor of software engineering (RuG, Netherlands) (BIT, Sweden)

Industrial development

Industrial research

Academia (+ consulting)
Who We Are...

A leading provider of business and financial management solutions

- Founded in 1983
- FY 2010 revenue of $3.5 billion
- Intuit is traded on the NASDAQ: INTU
- Employs around ~8,000 people
- Major offices across the U.S. and in Canada and the United Kingdom
- More than 40 million people use our QuickBooks, Payroll, Payments, TurboTax, Digital Insight and Quicken products and services.
Mission: why we exist as a company...

To be a premier innovative growth company that improves our customers’ financial lives so profoundly... they can’t imagine going back to the old way

We serve these end customers
- Consumers
- Small Businesses

...and those who serve them
- Accountants
- Financial Institutions
- Health Care Players

“Better Money Outcomes”
- **Financial**... making & saving money, grow & profit
- **Productivity**... turning drudgery into time for what matters most
- **Compliance**... without even having to think about it
- **Confidence**... from the wisdom & experience of others
Proven formula: lots of delighted customers...

Help families find $1,000 annually... $400M in consumer savings

Help people get the maximum tax refund... $33B in tax refunds, 1 out of every 3 tax returns e-filed

Improve FI profit per customer by 20%... IB customers equal to the 5th largest U.S. bank

Help small businesses be 20% more profitable... Customers revenues ~20% of U.S. GDP, pay 1 in 12 American workers

Help accountants be 20% more productive today... Serve half of all accounting firms

Improving 40M Lives
Proven Formula: talented & engaged employees

Most Admired: Software Industry

7 Years in a row
2004 2005 2006 2007 2008 2009 2010

Fortune Top 100 Places to Work

Strong Employee Engagement

World Class = > 93%
Best in Class = > 84%
Secular Shifts: transforming our company...

**Trends**

- Demographic Shifts
- Value Creation Shifts
- Technology Shifts
- Geographic Shifts

**Implications**

Intuit is driving: "Connected Services"

- Software-Advantaged Services
- Software-as-a-Service
- Platform-as-a-Service

Intuit is embracing:

**Social**

capitalize on our large and growing customer bases to unleash the collective power of user contributions, behaviors and data

**Mobile**

deliver “in the pocket” when that is the preferred solution

**Global**

employ the world’s talents to find & solve important problems around the globe
Overview

• Vem är jag? Wie ben ik? Who am I?
• Introducing Intuit
• Speed matters: implications for software engineering
  • Building *delightful* products
  • Software ecosystems
  • Implications for software engineering automation
• Conclusion
Where are we going? How fast?

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*
Integration-centric software engineering

software product lines
global software development
software ecosystems

causing

unacceptable complexity and coordination cost
Focus on one thing: Minimize Dependencies

Web 2.0 Rules to SW Development (1/2)

Team size
- 3x3 = 3 persons x 3 months (Google)
- 2 pizza rule (Amazon)
- Principle: What is required is a team, where the roles are defined and each member has the right skill for that role, and following a lean, agile, method — all focused on the customer.

Release cycle
- Weeks, not months
- Continuous deployment
- Principle: short cycles are key for agility, speed and decoupling

Architecture
- 3 API rule
- Mash-ups and web services
- Principle: architecture provides simplicity, compositionality and is designed in parallel with software development
Web 2.0 Rules to SW Development (2/2)

Requirements and Roadmapping
- Each team (3 persons) announces what they intend to release.
- Some (QA) requirements are shared across the board, e.g. performance, latency, etc.
- Principle: the cost of overlapping teams is much lower than the cost of synchronized, planned roadmaps and plans.

Process
- CMMI and other maturity approaches address the symptoms, not the root cause.
- Control is an expensive illusion causing LOTS of inefficiency in the system.
- Principle: Architecture not process should manage coordination and alignment.

From the Cathedral to the Bazaar

The POWER of Public Humiliation
Towards Composition ...

- Teams releases frequently, when they want
- Teams are self-selected (2 pizza rule)
- Teams are self-directed (little roadmapping)
- Architecture prioritizes simplicity (3 API rule)
- Components are backward compatible and negotiate interfaces
- Architectural compositionality
- Customers compose their own products
- Teams can be external (ecosystem)
Implications for Software Engineering

- From process to architecture
- From centralized to decentralized
- From planning to experimentation
- From long cycles to short cycles
- From large teams to small teams
- From internal to ecosystem
- From CMM(I) to agile
- From cathedral to bazaar
Classification – Five Approaches

- Traditional product development
- Integration-centric development
- Release groupings
- Release trains
- Independent deployment
- Open ecosystem
- Ecosystem development
- Traditional product development
Overview

- Vem är jag? Wie ben ik? Who am I?
- Introducing Intuit
- Speed matters: implications for software engineering
  - Building *delightful* products
  - Software ecosystems
  - Implications for software engineering automation
  - Conclusion
What Do These Products Have in Common?
Designing Pleasurable Products

Hierarchy of Consumer needs

“People seek pleasure”

Jordan’s four pleasures framework (based on Tiger 1992):

Physio-pleasure
  - Pleasure from sensory organs, e.g. tactile feedback

Socio-pleasure
  - Enjoyment from social interactions

Psycho-pleasure
  - Cognitive and emotional responses, e.g. usability

Ideo-pleasure
  - Supporting people’s values, e.g. green values

“Design for Delight” at Intuit

Going **beyond customer expectations** in delivering **ease and benefit**, evoking positive emotion throughout the customer journey...

**...So folks buy more & tell their friends**

**Benefit** = the improvement in the customer’s life or business outcome

**Growing our business is the goal**
The “How”

Uncover what’s most important to customers
In that focus, create better solutions, within available resources

Observe

Understand

Pick a Focus

Prototype lots of Ideas

Test

Fast Iteration

Repeat

Intuit Design4Delight Framework
Overview

- Vem är jag? Wie ben ik? Who am I?
- Introducing Intuit
- Speed matters: implications for software engineering
- Building delightful products
- **Software ecosystems**
- Implications for software engineering automation
- Conclusion
Towards Web 3.0

3 **Atomisation.** Globalisation and networking technologies will enable firms to use the world as their supply base for talent and materials. Processes, firms, customers and supply chains will fragment as companies expand overseas, as work flows to where it is best done and as information digitises. As a result, effective collaboration will become more important. The boundaries between different functions, organisations and even industries will blur. Data formats and technologies will standardise.

“...My prediction would be that Web 3.0 will ultimately been seen as applications which are pieced together. There are a number of characteristics: the applications are relatively small, the data is in the cloud, the applications can run on any device, PC or mobile phone, the applications are very fast and they're very customizable. Furthermore, the applications are distributed virally: literally by social networks, by email. You won't go to the store and purchase them... That's a very different application model than we've ever seen in computing.” —Eric Schmidt
Toward Product Composition ...

- Architectural guidelines guarantee composability
- Components/subsystems guarantee quality

- Reference integrations are created
- InnerSource

- Compositional platform
- Hierarchical platform
- Integration-oriented

- Platform
- BG platform/InnerSource
- Product
From Pre-Packaged Offerings to Customer-Assembled

application

platform

traditional

offering

componentized platform

prosumer asset

3rd party asset

ecosystem platform

each customer his/her offering

the vision
One View of the Intuit Ecosystem

World of opportunities: jobs to be supported and automated

- compositional applications, e.g. accounting, customer, employee, payments, etc.
- domain functionality, e.g. accounting, customer, employee, payments, etc.
- non-differentiating, generic functionality, e.g. subscription, billing, entitlement, etc.

experiment & innovate
platformize
maintain sustainable competitive advantage
commoditize
minimize TCO through putting in OSS or replace with COTS

value to individual user
breadth of applicability
## Classifying Software Ecosystems

<table>
<thead>
<tr>
<th>end-user programming</th>
<th>Yahoo! Pipes, Microsoft PopFly, Google’s mashup editor</th>
<th>none so far</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>SalesForce, eBay, Amazon, Ning</td>
<td>none so far</td>
</tr>
<tr>
<td>operating system</td>
<td>Google App Engine, Yahoo developer, Coghead, Bungee Labs</td>
<td>Nokia S60, Palm, Android, iPhone</td>
</tr>
<tr>
<td>category platform</td>
<td>desktop</td>
<td>web</td>
</tr>
</tbody>
</table>

- **end-user programming**
  - MS Excel, Mathematica, VHDL

- **application**
  - MS Office

- **operating system**
  - MS Windows, Linux, Apple OS X
Questions One Might Ask...

- Can ecosystem developers store data outside Intuit’s repository?
- Can ecosystem developers host their solutions outside Intuit’s hosting infrastructure?
- Are other platforms allowed in the ecosystem and, if so, are these integrated?
- Can ecosystem joblets have the same access to domain services and data as Intuit joblets?
- Do we charge developers for developing and hosting in our ecosystem (beyond revenue share)?
- How do we maintain consistent user experience between Intuit and ecosystem joblets?
- How do we manage variability & configurability for customers?
- Does Intuit provide access to data defined by ecosystem developers to other ecosystem developers?
- How do we manage dynamic composition of joblets by customers?
- What mechanisms exist to insert ecosystem domain services into other domain service workflows?
- How do we maintain consistent user experience between Intuit and ecosystem joblets?
- How do we manage dynamic composition of joblets by customers?
- How rich is the set of basic services provided by the platform, e.g. authentication, authorization, monitoring, billing, search, marketing, etc.
- Can ecosystem developers access aggregated customer (DAAA) data?
### Comparing Existing Ecosystems

<table>
<thead>
<tr>
<th></th>
<th>Salesforce</th>
<th>eBay</th>
<th>Facebook</th>
<th>Amazon</th>
<th>LongJump</th>
<th>Ning</th>
<th>PopFly</th>
<th>AppEngine</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Customer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configurability</td>
<td>No/limited</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Consistent UX</td>
<td>No/limited</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic composition</td>
<td>No/limited</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Ecosystem developer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equal access</td>
<td>Supported</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Behavioral integration</td>
<td>No/limited</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hosting alternatives</td>
<td>No/limited</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3rd party data access</td>
<td>No/limited</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Developer environment</td>
<td>No/limited</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External data storage</td>
<td>No/limited</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAAA access</td>
<td>No/limited</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charges</td>
<td>No/limited</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Platform architecture</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Platform services</td>
<td>Supported</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Desktop application sync</td>
<td>No/limited</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External ecosystems</td>
<td>No/limited</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **No/limited support**
- **Some support**
- **Supported**
- **N/A**
Overview

• Vem är jag? Wie ben ik? Who am I?
• Introducing Intuit
• Speed matters: implications for software engineering
• Building delightful products
• Software ecosystems
  • Implications for software engineering automation
• Conclusion
Implications

• Simplify, simplify, simplify
  – Make it easy to do the right thing, e.g. no versioning
  – Decouple components, teams and organizations

• Continuous Deployment

• Scale: Make teams effective in large systems

• Support software ecosystems

• Help manage design erosion
Simplify, Simplify, Simplify

Our life is frittered away by detail. Simplify, simplify, simplify! I say, let your affairs be as two or three, and not a hundred or a thousand; instead of a million count half a dozen, and keep your accounts on your thumb-nail – Henry Thoreau (Walden)

• Each design decision adds design rules and constraints that need to be observed by engineers

• Collectively, these decisions cause major complexity that decrease productivity

• **What to do:** Hide it, platformize it, make it happen automatically
Decoupling: No Versions!

Shared Component, e.g. Engine, etc. V1.01

Configuration interface

Provided interface (SOA style, maximal decoupling)

No offering or shared component may depend on the implementation

Syntactically and semantically equivalent until a deliberate sunset is planned

Frequent (4 week) releases of production quality component

Shared Component, e.g. Engine, etc. V1.02

required interface

Respect Independent Deployment: still usable in context where this interface can not be bound

Automated test suites for each interface
Decouple Components and Teams

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

Open research topic: tool support for continuous deployment in safety/business critical systems
Scale: Make teams effective in large systems

- Components, especially over time, build major dependencies, complicating development
- Component teams are dependent on each other along the lines of the component dependencies
- Feature teams tend to make changes in all components affected by the feature, adding risk and affecting release schedules
- Systems that require software assets from multiple organizations suffer even more from dependencies between components

Open research topic: tool support for feature teams that mitigate aforementioned risks
Support Software Ecosystems

- Software ecosystems do not allow for process-based coordination
- Architecture forms the basis for coordination
- Enforced process by platform owner major source of frustration for external developers

Open research topic: Tool support to minimize/remove “process-based” interaction
Evolve architecture; fight erosion
Overview

- Vem är jag? Wie ben ik? Who am I?
- Introducing Intuit
- Speed matters: implications for software engineering
- Building *delightful* products
- Software ecosystems
- Implications for software engineering automation

• Conclusion
Increasing **SPEED** trumps ANY other improvement R&D can provide to the company – it is the foundation for everything else

- As a process, methods or tools professional, there is only ONE measure that justifies your existence: how have you helped teams move faster?
- Don’t optimize efficiency, optimize speed
Inflection Point

- 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.
Automated Software Engineering 2.0

- In a world of **continuous deployment** and **software ecosystems**, automation is fundamental

- Simplify, simplify, simplify
- Continuous Deployment
- Scale: Make teams effective in large systems
- Support software ecosystems
- Help manage design erosion
Not My Job?!

Strong LEADERSHIP needed from YOU
Software Engineering Research

What it is NOT

• Science, i.e. proving that something can be done
• Research is leading
• Validation can be done “in-vitro”
• Researchers understand what is important
• Research can focus on one narrow, often technological, aspect
• A niche activity

What it is

• Engineering, i.e. establish the benefit of a solution
• Industrial practice is ahead of research
• Validation occurs “in-vivo”
• Without industrial experience, relevant research is hard
• Research needs to be holistic, addressing organizational, process and business issues
• Supporting a multi-billion dollar industry and critical
THANK YOU

Mount Shasta (CA) - 4,322m, July 2009