Evolving the Notion of Software Architecture: Three Dimensions

Jan Bosch, Intuit
UC Irvine, ISR Research Forum 2008, June 6

Overview

Vem är jag? Wie ben ik? Who am I?
- Introducing Intuit
- Trends
- Software Product Line Architecture
- Architecture for the Ecosystem
- Dynamic Software Architectures
- 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)

Intuit Company Information

Who We Are...

A leading provider of business and financial management solutions

- Founded in 1983
- FY 2007 revenue of $2.67 billion
- Intuit is traded on the NASDAQ: INTU
- Employs more than 8,000 people
- Major offices across the U.S. and in Canada and the United Kingdom
Great Brands and Great Products

Fortune top 100 places to work
Did You Know?

**Intuit has Some of the Strongest Brands in Business.**

FY 2007 Year-to-Date Retail Unit Share

- **78%**
  - *Quicken*

- **89%**
  - *QuickBooks*

- **79%**
  - *TurboTax*

- **15 million Quicken customers**

- **Nearly 7 million small businesses are Intuit customers**

- **More than 14 million federal desktop and Web TurboTax units** (Tax Year 2006)

Source: NPD, company estimates
Overview

- Vem är jag? Wie ben ik? Who am I?
- Introducing Intuit

Trends

- Software Product Line Architecture
- Architecture for the Ecosystem
- Dynamic Software Architectures
- Conclusion

Where are we going? How fast?

Innovation Adoption

*technology maturity* = 50 million users in the US

- radio: 38 years
- phone: 25 years
- TV: 13 years
- cable: 10 years
- internet: 5 years

Trend: software size

- software needs in products constantly increasing

R&D as percentage of sales is pushed to unacceptable levels

person years per product

**Trend: systems of systems**

systems increasingly need to be integrated with other systems
- Information systems
  - from manual data exchange to behavioural integration
- embedded/technical systems
  - from stand-alone to signal exchange to behavioural integration

**Unilateral control of system functionality is diminishing**

**Trend: Variability**

Variability needs in software are constantly increasing because
- variability moves from mechanics and hardware to software
- design decisions are delayed as long as economically feasible

**# of variation points is increasing**
**time of binding is constantly delayed**
**party performing the binding is changing**
Later Binding

- trend is towards later binding and increased automation

New solutions are needed to
- guarantee system properties
- facilitate post-deployment flexibility
- deal with changing contexts

Software Platforms – The Rules are Changing!

- Scope of platform team R&D
  - S40 – 100% of product functionality
  - S60 – 60% of product functionality
  - MAEMO – 30% of product functionality
- Requirements management
  - S40 – complete control
  - S60 – control, but shared with Symbian, licensees and 3rd party developers
  - MAEMO – only UI is controlled (but not completely), rest is influence-based
- Architectural control
  - S40 – complete control
  - S60 – major controller, but significant influences elsewhere
  - MAEMO – largely through influencing and collaborating with Open-Source community

From “The Cathedral” to “The Bazaar”
Overview

- Vem är jag? Wie ben ik? Who am I?
- Introducing Intuit
- Trends

Software Product Line Architecture
- Architecture for the Ecosystem
- Dynamic Software Architectures
- Conclusion

Software Challenge

increasing SW size per product  
develop fewer products

increasing variability requirements

develop more products

Software Product Lines
Defining Software Product lines

- Product consists of:
  - Product specific components
  - Shared components configured for the product
  - Shared components used as is
  - Externally developed components
- Software Product Line consists of:
  - Software product line architecture
  - Shared components
  - Commonality and variability model

A Brief History of Software Reuse

Technology orientation
- Module
- Object & class

Technology and process
- Component
- Object-oriented frameworks

Complete intra-organizational perspective
- Software product lines

Inter-organizational (or eco-system) perspective
- Open software platforms
- Open-source software communities
What Success Looks Like

- Business perspective: SPL technology forms the basis for “the next S-curve of growth” for the company
- R&D perspective: Order of magnitude richer product portfolio against stable or slightly increased R&D investment
- Ways to achieve success:
  - Product portfolio diversity
  - Common user experience for products in the portfolio
  - Much more customizable customer products
  - Higher quality products due to reliable shared core
- Often ignored advantage
  - Low opportunity cost of new product experiments
  - Improvements become available for all products at once
  - Improved productivity due to specialization of teams

Dimensions of SPLs
Focus areas
Vision and early design of processes, tools and operating mechanisms. Initiate work in the most critical process areas, i.e. SCM, automated test, deployment, etc.

Initiation phase
Educate transitioning staff on mindset, process, tools and architecture. Initiate an engineering “heartbeat” facilitating coordination and integration of assets and processes.

Build out phase
Drive scalability of processes and tools. Safeguard architectural integrity through selected refactoring efforts. Manage domain and software variability.

Institutionalized phase
Focus on process optimization. Institutionalize management of commoditization of functionality. Address design erosion by architectural refactoring efforts.

Risks inherent to Reuse Strategies

- **Complexity** – the “gravity” of software engineering: Reuse can add complexity by creating dependencies between previously autonomous organizational units.
- **Web of dependencies**: Can lead to a “lockstep” evolution model in which everyone has to evolve synchronously.
- **Coordination cost**: Dependencies require significant synchronization and alignment, diminishing the benefits of strategic reuse.
- **Offering integration cost**: Often the cost of offering integration is higher than expected due to the complexity of configuring and integrating the selected shared assets.
- **Process & tool divergence**: Teams with diverging “external” interfaces, e.g. different release cycles and mechanisms, “creative” interface management, immature requirements management, lacking quality management, etc. cause significantly higher offering creation cost and jeopardize the product line effort.

Strategic reuse creates competitive advantage as long as we manage to these risks
Mitigation Strategies

- **Decoupling**: replace coordination mechanisms with a uniform set of architecture and team responsibilities
- **Independent deployment**: maximize the ability of components to evolve independently through upward and downward interface compatibility requirements
- **Knowledge management**: design for integration, interface management, documentation and production plans help consumers be productive against minimal investment
- **“Step function” change**: as we are at an ”inflection point”, radical change can be imposed, i.e. harmonizing processes and tools
- **Specialization**: Separate development from process and tools evolution

Proven techniques must be proactively applied to ensure success

Issues While Transitioning

- **key issue**: Adopting SPLs requires **changes to all aspects of the business** – this is often ignored
- **potential issues**:  
  - mismatch between shared components and product needs  
  - design erosion of shared components  
  - complex interface  
  - high degree of ”organizational noise”  
  - inefficient knowledge management  
  - evolution causes ripple effects through the R&D organization
Overview

• Vem är jag? Wie ben ik? Who am I?
• Introducing Intuit
• Trends
• Software Product Line Architecture
  Architecture for the Ecosystem
• Dynamic Software Architectures
• Conclusion
Building Web 3.0 at Intuit?

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

From Pre-Packaged Offerings to Customer-Assembled

application
platform

traditional
contemporary

offering
componentized platform
3rd party asset
prosumer asset
ecosystem platform

the vision
Overview

- Vem är jag? Wie ben ik? Who am I?
- Introducing Intuit
- Trends
- Software Product Line Architecture
- Architecture for the Ecosystem
  
  Dynamic Software Architectures
  
- Conclusion
Assumptions about Architecture

• software architecture is hard to change
• consequently, design architectures carefully
  – architecture assessment
  – architecture design
• software architecture is static, the stable part of the system

inflexible architecture is good/fact of life!

Flexible Architectures?

• why is SW architecture hard to change?
• ignored aspect of the problem
• loss of design knowledge – vaporizes during
  – architecture design
  – component development
  – system evolution

⇒ architecture design decisions are lost
Example – Fire Alarm System

Architecture Design Decisions

architecture design decisions consists of
- restructuring effect
- design rules
- design constraints
- rationale - new principles, guidelines, etc.
and are taken in response to
- functional requirements
- quality requirements
Dynamic Software Architectures

- Later binding trend also applies to software architecture
- Design decisions are taken in response to functional and quality requirements
- For dynamic systems, the desired behaviour of the system is required to change post-deployment
- This requires design decisions to be reversed and/or replaced
- First class representation of design decisions, at run-time, would facilitate this behaviour

Alternative perspectives
- Variability requirements in software are constantly increasing
- Achieving superior user experience increasingly requires user specific intelligent behaviour
- Context changes in mobile systems

Conclusion

- Vem är jag? Wie ben ik? Who am I?
- Introducing Intuit
- Trends
- Software Product Line Architecture
- Architecture for the Ecosystem
- Dynamic Software Architectures

Software architecture is more important than ever
The notion of architecture needs to evolve with the needs of the software industry