Applying inner source development in product line engineering

Frank van der Linden
CTO Office
Spider
April 19, 2011

frank.van.der.linden@philips.com
Overview

Background
   – Philips Healthcare
   – Funded EU projects
      • Software product lines & open source
   – Inner source

Conclusion
Imaging systems supporting medical diagnosis

- Extensive image processing
- Image storage
- Image exchange
- Image viewing

Different “modalities”:
- X-Ray, MRI, Ultrasound, PET
  …
Philips Healthcare
increase of scope

Moving to imaging support during intervention
  – Hearth catheterisation
  – Minimal invasive surgery

Moving to cover the whole care cycle
  – Prevention, screening, diagnosis, treatment, management, surveillance
Philips Healthcare
Businesses and products

General X-ray
Cardio/Vascular X-ray
Ultrasound
Computed Tomography
Magnetic Resonance Imaging
Nuclear Medicine
Positron Emission Tomography

Radiation Therapy Planning
Cardiac and Monitoring Systems
Healthcare Informatics
Customer Services
Overview

Background
- Philips Healthcare
- Funded EU projects
  - Software product lines & open source
- Inner source

Conclusion
Funded EU projects

EU Esprit projects
• ARES & Praise (1995-2000)
ITEA projects
• ESAPS-CAFÉ-FAMILIES (1999-2005)
  – Introduction of product lines
• COSI (2005-2008)
  Co-development using inner & Open source in Software Intensive products
  – Open source development

13 years of European partnership projects
Total > 40 European partners
Why Software Product Line Development?

Business Drive
- Improve software production process
  - Learn from mechanical engineering technologies
- Fast generation of software
  - Software platform development
    • Managed reuse
- Fast production of variants
  - Software mass customisation
    • Managed variability

Software development improvement
- Reduce development cost
- Reduce product lead-time
- Reduce maintenance
- Feature propagation
- Quality
- Common look-and-feel
- ...

Managed reuse
- Variability management & platform
Product Line development experienced advantages

Product cost reduction 60-70%
Improved productivity
  – Factor 2-6 higher output
Investment reduction
  – av. 50%, up to 90%
Quality improvement
  – Product defect density < 50%
  – Reuse of test cases 40-60%
Product lead-time reduction
Ease of configuring
  – av. 50% faster, up to 95%
Maintenance cost reduction
Portfolio complexity reduction
  – av. 50% less components
  – Training time reduction

Data based on measurements by industrial partners in the projects

CTO Office, Frank van der Linden, April 19, 2011
Product Line development in embedded systems

Product lines in embedded systems
  – Since ~1990

Enable managed reuse and managed variability

Main elements
  Two development processes

Explicit variability

Means
  – Platforms

Explicit
  – Variation model:
    • Variation points & variants
    • Configurations
  – Decision support on variants
Distributed development: COSI

How to manage the complexity of heterogeneous distributed development

Challenges

- Large groups working on the same software
- Development groups are distributed
- Software shifts to commodity
- 3rd party software is increasingly used

COSI Consortium:
5 Industries
8 SME
2 Research institutes
4 Universities
Commodification of software

Software that was originally differentiating gets to be obtained as commodity

- losing intellectual property
- COTS - Open source
- wasting valuable engineering resources
- commodity
- differentiating
- basic for the business

CTO Office, Frank van der Linden, April 19, 2011
BAPO concerns of software engineering

**Business**
- Costs & profits, strategy, planning → Business Orientation

**Architecture**
- Technical means to build the software
  → Variability Management
  → Architecture-Centric Development

**Process**
- Roles, responsibilities & their relationships
  → Two-Lifecycle model

**Organisation**
- People & organisational structures
  → Two-Lifecycle model
Leveraging OS opportunities

1 developing with OSS practices
   → inner source
2 using OSS tools in developing products
3 developing products containing OSS
4 developing OSS products
5 engaging and leverage the community
Inner source

Exploit the OS distributed development advantages within a company

- Easy access to all information of the product line
- Release early and often
- Distributed ownership and control of domain assets
- Cooperative eco-system of development teams
- Avoid problems with planning, ownership and control

Use organisation mechanisms

- Escalation of conflicts
- Roadmaps

Results:

- Improved involvement of the application teams in the domain
- Improved platform use
- Reduced time-to-market
Using open source tools

Open source software not in the final product
- Easy for the user to comply to licences
- Beware: for maintenance reasons tools may become part of product

Many open source tools already available
- Not many product line specific tools

Examples:
- Stylebase for Eclipse
  - reuse and sharing of architectural knowledge
- Subversion
  - version management
- Semantic MediaWiki
  - support collaborative development
Using open source components

Similar to the use of 3rd party such as COTS
Planning of 3rd party software outside control
- Architecture compliance & interfaces is an issue
- Advanced knowledge is necessary

COTS
- Need good contacts with the supplier
- Disruptive releases

Open Source
- Involvement in the community
  - Leads also to (limited) control
- Continuous evolution
  - Fast incorporation is possible
  - Issuing bug reports and corrections keep open source at quality
Using open source components

Product line issue

Open source in domain or application
  – latest versions only in “trial” applications

Licensing: Communicated and managed
  – Ignoring licenses may lead to consequences for other departments

Open source use in application development indicate a move towards commodification
  – move to domain?
Opening up products of the product line

Commodity software can be shared

– Supporting a standard
  • Improve interoperability with competition
– Move to de-facto standard
  • Own products comply
  • Devalue propriety solution of a competitor
– Drive acceptance of the software
  • Visibility of the source
– Sharing maintenance
  • De-support strategy
– Enable the sales of something else
  • Services!
– Increase security, safety
Symbiotic relationship

Active involvement in community
- Ensure that the right issues are addressed

Obtain improved components/tools
- Issue bug reports
- Donate software & patches solving them (partially)

Shared maintenance
- Software that is not maintained properly
Overview

Background
  – Philips Healthcare
  – Funded EU projects
    • Software product lines & open source
  – Inner source

Conclusion
Philips Inner source

Development with open source practices

Philips experienced SPL Challenges:

– Growing platform adoption
  • Growing customer base
– High number of feature requests
  • Increasingly difficult to honour
– Products increasingly dependent on platform
  • Release schedules inherently misaligned
– R&D groups distributed across 3 continents
  • Communication more complex and cumbersome
Inner source: Why resolve bottleneck at domain engineering

Inspiration from OSS development:

Bring software engineers closer together
  – Direct communication
  – Platform knowledge sharing and exchange

Business units should also contribute to developing domain assets
  – BUs are not dependent on the platform group
  – Components are developed in the BU that has the best expertise
Background

Problems:

- Lack of Domain Expertise
- Stepwise deployment
- Alignment with (System) Product Roadmaps (priority setting)

InnerSource

- All groups contribute components based on their need and expertise
Inner source: What
Increase trust in platform

Improve feedback by being open
  – Publish source code & relevant documentation
  – Improved platform and product quality

Involve customers early
  – Works-In-Progress (bimonthly)
  – Snapshots (biweekly)
  – Bleeding edge (instantaneous)
  – Improved quality through early feedback
  – Short time to market
Inner Source principles

Easy, but controlled, access
- All development information

Release early and often
- Flexibility
- Major (and minor) releases
- Snapshots & Bleeding edge

Distributed Ownership and Control
- The platform team owns and develops components
- The customer is allowed to change components,
  - Ownership rules

Patch
- Patches are improvements that may be offered back to the platform
Collaboration models

1. **Use as is**
   - Use the platform unchanged
     + efficient
     – product team dependent on platform team

2. **Patch**
   - Change some components
   - Modifies the platform
     + flexibility for application engineering
     – inefficient: less re-use
Collaboration models

Contribute
   Change some components
   Offers change back
   + flexibility
   – re-use: contributions may need rework
   – does not work for big changes

Virtual team
   Application engineers temporarily join the domain engineering
   + flexibility and re-use
   + works for large scope
   ± must be managed well
Business aspects

Who pays for contributions by application engineering system-groups?
  - Profit/loss responsibility
  - Effort needed to make components re-useable

How to handle the maintenance and support?
  - Contributing group gets maintenance/support obligations
  - Role of the platform group

Internal Business Model
  - encouraging active collaboration and contribution
Necessary collaboration tooling support

Provide a scalable solution for global and inter-organizational collaboration
  – Simplify development model
  – Increase collaboration agility
  – Reduce administration overhead
  – Reduce merge overhead

Retaining the strong points of ClearCase
  – Merge tracking
  – Rename handling
Collaboration environment

Collab.net, Semantic Wiki & Subversion

– Mailing lists for support and technical discussions
– Document and file sharing
– Subversion for sharing source code
– Subversion for collaborating on new code

Staged introduction of the Collaboration environment
Experience with global collaboration

Much simplified development process
  – Removed all unnecessary merging
  – More flexible delivering (no serialization needed)

Lightweight: easy to learn,
minimal impact on other processes
Experiences openness

Openness/easy access to information:
- Support mailing lists successfully replaced the formal help desk
- More and earlier feedback improves quality and reduces lead time
- Document sharing through Wiki

Inner Source collaboration models:
- Patching supported by a simple add-on script
Size and health of the community

Percentage of the Philips SW community using the collaboration environment

- Steady growth
  - over 1000 users of 1800 developers
- About 50% of users are active users (has been constant over time)
Results

Three times more product groups served
  – Limited growth of the platform team
Substantially improved product quality
  – Improved feedback from product groups
  – Product groups find defects early
Significant time to market gains
  – Product groups can start integrating earlier
  – Product groups can provide features themselves
Growing and active Inner Source community
  – Over 60% of the PH software community involved
  – Many collaborations inside and outside Philips
Inner source
Conclusions

New environment boosted collaboration enormously

– Many collaborations running at any time
– More feedback
  → quality improvements and shorter lead time

Key functions

– Subversion (version control)
– Discussion services (mailing lists)
– Information sharing (Wiki, fixed documents less important)
– Role-based access and distributed project management
Inner source

Conclusions

The new environment is well adopted and liked
- Steadily growing user base; active discussion lists
- Low learning curve; engineers like the new environment
- Network performance (corporate proxies) is a bottleneck

Subversion:
- Enables large scale distributed development
- Drastically simplified platform development
Conclusions

Inner Source established within Philips Healthcare
  – Detailed model underlying Inner Source
  – Global collaboration infrastructure essential
  – Adoption high and rising
  – Inner Source helped break the platform bottleneck

Semantic Wiki established
  – More people create documentation
  – Wider variety of documentation created
  – People are enthusiastic and keep coming back
  – Less “trivial” questions on the support list