Variability & Compositional Software Development

Jilles van Gurp

Nokia Research Center, Helsinki

Smart Space Application Platform Team

Computing Structures Lab

NOKIA
Connecting People

Overview

- **Discussion points** for after the presentation
- What is **compositional development** and why does the software world evolve to it
- What are **the implications for SPL development**
  - Specifically for variability management
- What can/should we do about it
  - i.e. **research agenda** for future SPLC confs

- Discussion
  - Whole point of this presentation is to provoke some

Discussion points (originally my last slide)

- **How** can existing variability management solutions be used in compositional development
  - XX MLOC
  - XXXXX Developers
  - Multiple organizations, countries, business units etc.
- **Why** should we try to do this?
  - OSS community seems to be doing fine without SVM overhead
- After ~10+ years of SPL and SVM research, what are the key things that we are going to keep assuming things will get more compositional
  - And what are things we need to rethink

SPL Development = Integrational development

- Develop platform(s)
  - Support product Commonalities
  - Central, configurable feature models
- Derive products
  - Collect requirements
  - Configure platform + Product specific development

- Key concepts
  - Central collection, analysis & modeling of requirements & features
    - Platform level
    - Product level
  - Large reusable asset base, i.e. the platform
  - **Reduce product development effort** by not repeating platform development effort

NOKIA
Connecting People

Trends

- Software systems **keep getting larger**
  - Moore's law mirrored in software size (LOC)
  - Now: Millions of lines of code
  - Tens of thousands of developers
  - Billions $ investments
  - Also true for embedded systems, home ground for SPL methodology
- Existing software platforms widening in **scope**
  - Expand domain & feature set
  - Diversify from competition
- **Cross organizational** boundaries
  - No company has 10000 people in 1 department
  - Subcontractors, Licensensees, Customer platforms
- **Time to market** increasing
  - Especially true for hierarchical platforms

Consequences

- Increasing **testing cost**
  - Repeated testing cost
- Decreasing **differentiating power**
  - Also hard to substitute integrated parts with cheap/free replacements
- Especially long **time to market for differentiating platform features**
- Platform design **decisions limit/constrain** product development
- Product increasingly **smaller percentage of platform code**
- Also **increasing amounts of product specific code**

- Inevitable conclusion:
  - The **"old way"** just does not scale
  - And it **needs to scale** anyway!

This is what we are seeing in Nokia Series 60

- Hierarchical platform
  - E.g. **Panasonic S60 based SPL** was presented at SPLC 2006
  - Based on Symbian platform
- XX **M**LOC (cannot disclose XX)
- XXXX people involved with S60 development **@nokia**

- Hint: no SVM for S60

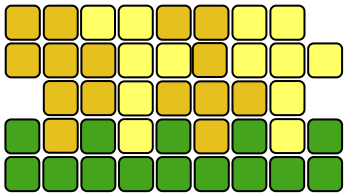
- Nokia is pushing the limit of what is possible in a single company
  - Not many companies with this amount of in-house developed software

- Rest of the industry will experience similar growth in software size

Solution: compositional development

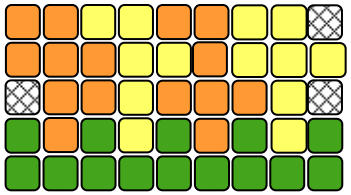
- Key concept: decentralize
  - Everything!
- Accept that
  - No person can oversee XX MLOC systems
    - empirical limit is (X)XX **K**LOC
    - Disqualifies anyone but superman for centrally made technical decisions
  - XX MLOC will become XXX MLOC at some point, BLOC in sight (10-20 years ?)
    - This is actually good (i.e. we are reusing stuff)!
  - You don't own most of 'your' software
    - Fixed cost per person per LOC
    - You're not going to build most of the stuff in your products
- Systems are composed of many components with independent
  - ownership, management, evolution, interests, ...

Derived Products



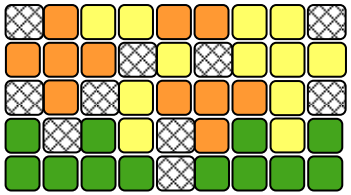
integrated platform

additions + **no** changes to platform



product 1

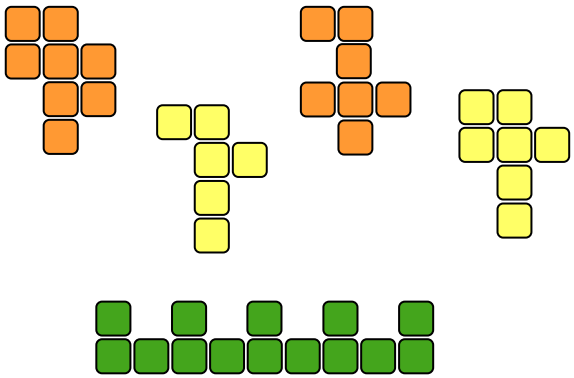
additions + **many** changes to platform



product 2

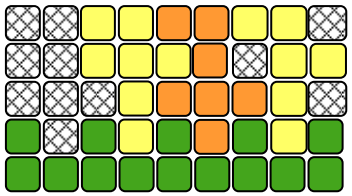
Integration Platform

Compositional platform



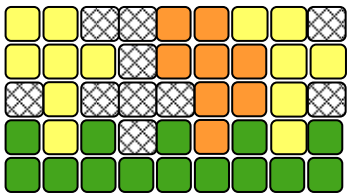
base platform + components

base platform + **reuse** comp. + **prod specific** component



product 3

base platform + **reuse** comp. + **prod specific** component



product 4

NOKIA
Connecting People

Consequence of decentralization

- No central
  - Ownership
  - Decision making
  - Architecture
  - Feature models
  - Requirements collection
  - Requirements analysis
  - **Variability management** (needs all of the above)
- Open source =
  - You can take ownership (i.e. you have the right to modify)
  - But generally you won't due to the cost
- Lesson: most software that you don't own is used unmodified
  - You can configure it or extend it in its intended way(s)

Does this sound familiar?

- It should because this is how the **open source community** works
- And they produce **vast amounts** of **good quality** software **fast**
- The question is no longer if you should use OSS but if you should adopt an OSS **mode of operation**

NOKIA
Connecting People

Does this sound familiar (2)?

- Is this just repeating **COTS**?
- No, COTS fragmented into
  - Integrated Platforms
  - SPLs
  - Open Source
  - 'True' COTS
- Compositional development tries to combine best of both worlds in OSS and SPL development
  - Arguably the **two most successful forms of software reuse**

Variability (a very brief history)

- Originally: **planned** reuse
  - Analyse requirements
  - Identify where variation are
  - Plan for use of appropriate techniques
- Later: **supported** reuse
  - Provide feature model of software platform
  - Guide product derivation process using this model
- Later (2): **automated** reuse
  - Provide feature model + software configuration model
  - Auto generate working/valid product configurations from feature configurations

Variability tooling

- Support product derivation process
- Product configuration validation
- Build configuration tooling
- MDA
- Feature Modelling
- Sales support
- Software License enforcement/configuration
- ....

Impact of decentralization on variability management

- Variation points are introduced in software by **component owners**
- Important architecture **decisions are taken locally** rather than centrally
- ***Centrally maintained models of features or software variation do not get updated when that happens***
- ***Nor can you make the owners make these modifications for you***

- Conclusion, any tool/method depending on centralized models is not going to work in combination with decentralizing development as implied in compositional development
- This affects most existing SPL approaches
  - Not all, e.g. Van Ommering

So now what?

- Are central models really essential?
- Was it all for nothing?
- What bits and pieces can we reuse?

- I think not all is lost.
  - Just need to re think a few things

Variability management is about provided variability

- **Provided** variant features
  - Variability actually present in the implemented software
- **Required** variant features
  - Variability needs emerging from the requirements during requirements analysis
- Same for provided and required arch & des.
  - Most Rational Rose licenses are actually used to document rather than design

- SPL research contributions depend mostly on
  - models of provided variant features in software and ...
  - ... mappings to variation points in the provided software design

- Independently developed software components can still have explicit provided variant features and explicit variation points
  - Nothing inherently central to this

Problems

- Documenting features & design is not likely to be done consistently in a compositional development environment
  - No incentive for component owners
  - Could be done centrally
    - whole point is not doing that anymore
  - Could be sanctioned from management that all owners do this 'properly'
    - except that implies central governance which we no longer have
- Consistent enforcement of any processes, methods & tools is hard due to lack of central governance
- Conflicts of interest between parties

Nice research topics

- How to automatically aggregate decentralized feature models & design into larger models such that
  - They can support product development (similar to how current variability tools do)
- First class representation in software for variation & variant features
  - E.g. using annotations
- Dealing with cross cutting features and components that are not centrally owned
  - E.g. security
- Locally using selected SPL methods & techniques
  - Without imposing them on all components
- Micro kernel like architectures with variability management support
  - E.g. OSGI

Discussion points (again)

- **How** can existing variability management solutions be used in compositional development
  - XX MLOC
  - XXXXX Developers
  - Multiple organizations, countries, business units etc.
- **Why** should we try to do this?
  - OSS community seems to be doing fine without SVM overhead
- After ~10+ years of SPL and SVM research, what are the key things that we are going to keep assuming things will get more compositional
  - And what are things we need to rethink