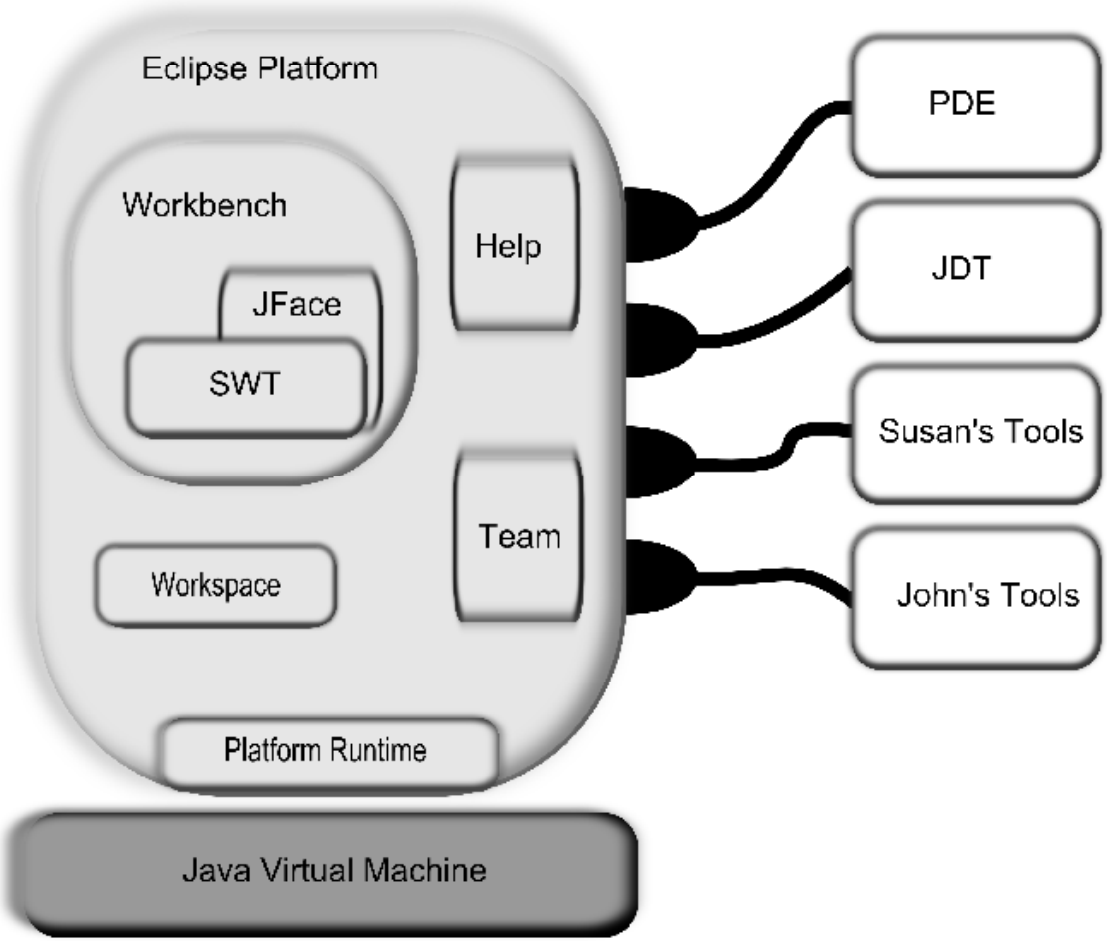
CPSC 875

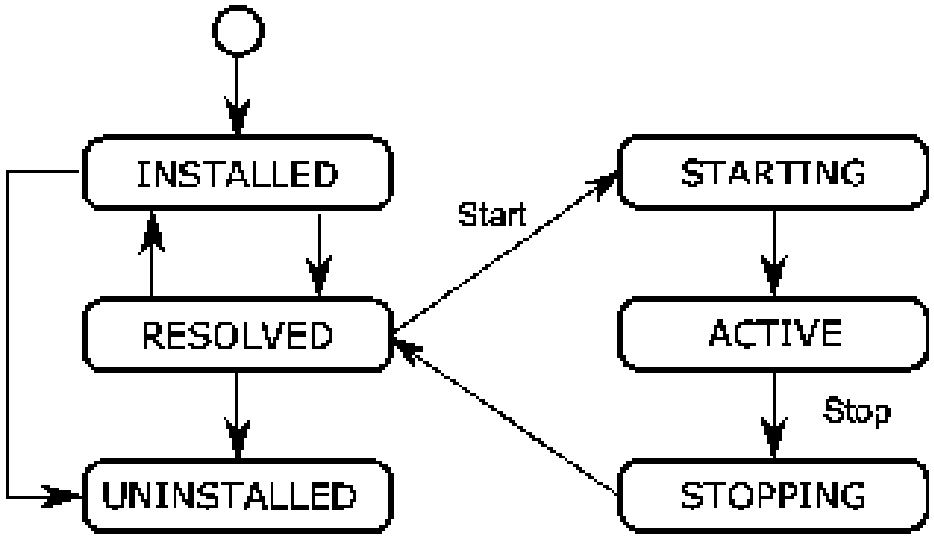
John D. McGregor

C15 – Variation in architecture

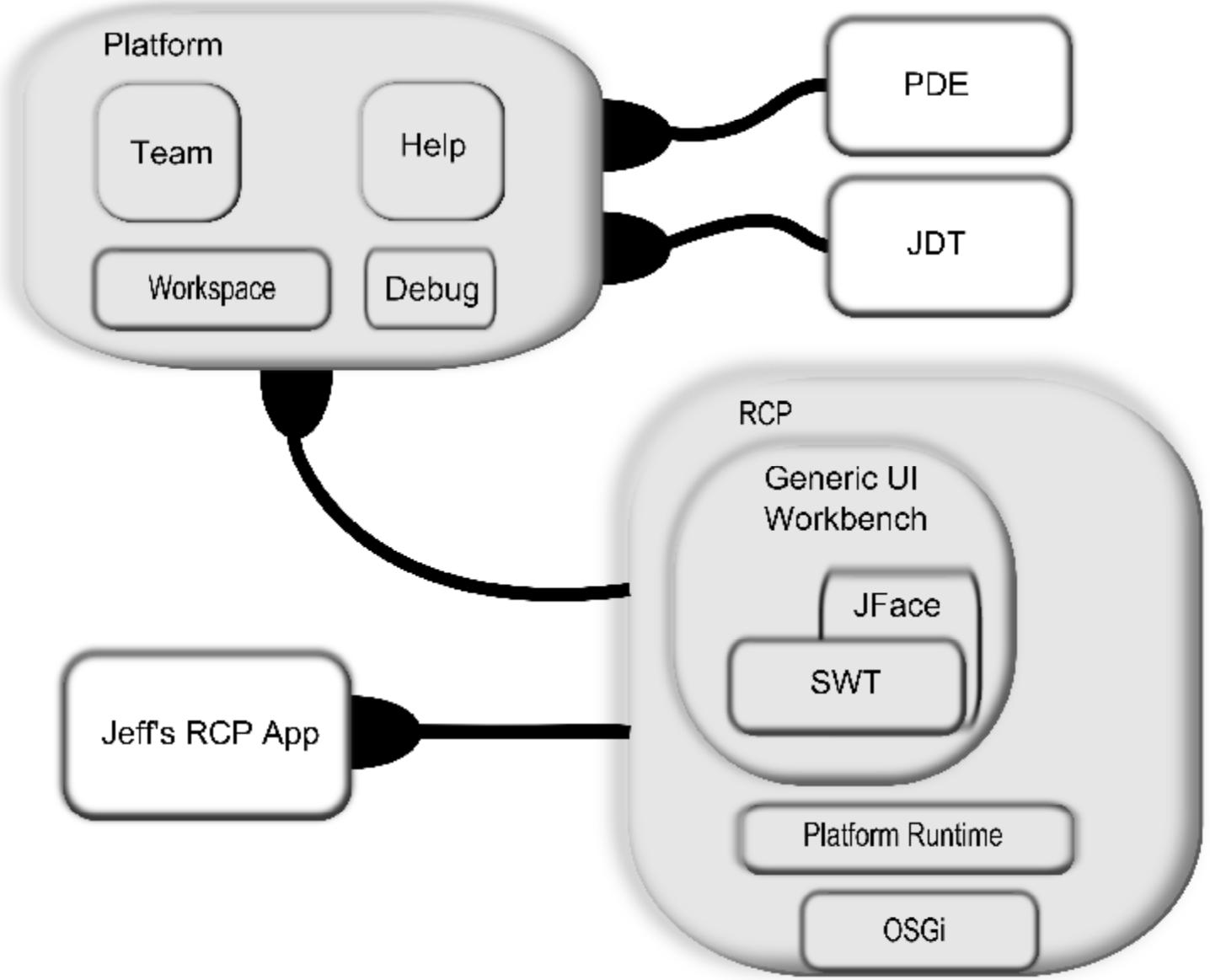
Early Eclipse



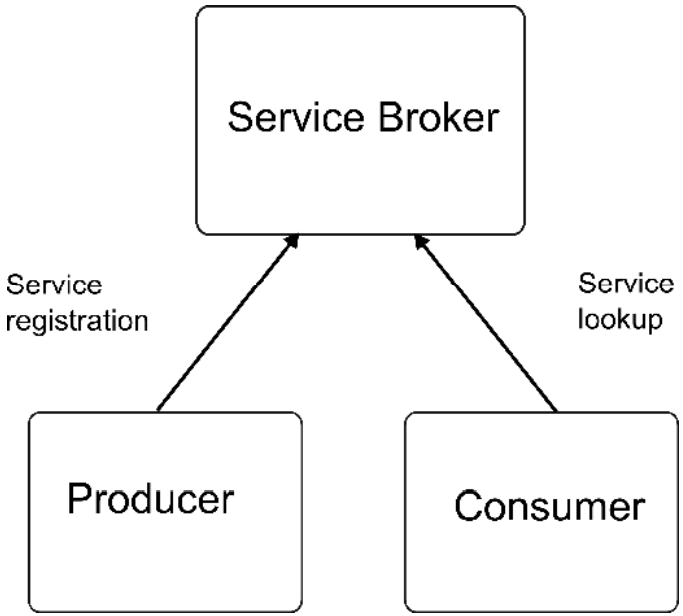
OSGi Bundle life cycle



Rich Client

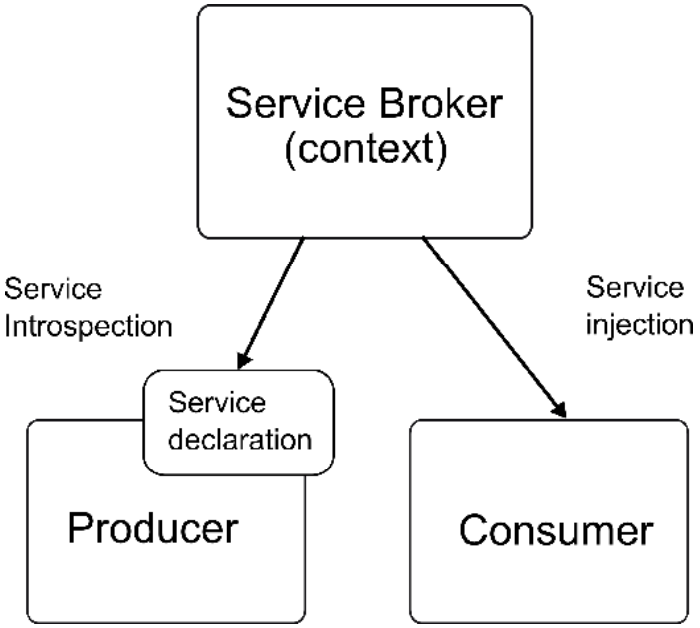

Platform
Team
Help
Workspace
Debug
PDE
JDT
RCP
Generic UI
Workbench
JFace
SWT
Jeff's RCP App
Platform Runtime
OSGi

3.4 -> 4.0



Plugins had to understand the location and structure of what they wanted to load.

Plugins tell what they want and the service broker delivers the current version and automatically sends updates.



Eclipse Evolution

- http://michel.wermelinger.ws/chezmichel/2009/10/the-architectural-evolution-of-eclipse/

Goal

- *The goal of variability in a software product line is to maximize return on investment (ROI) for building and maintaining products over a specified period of time or number of products.*

Different kinds of product variation

- Differentiation
- Evolution

- There is also data variation

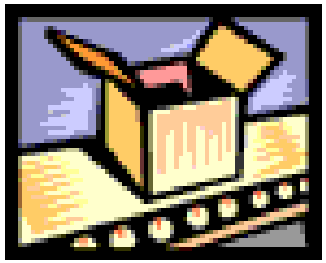
Software Product Line

- Multiple products, each a bit different from the others
- The differences are encapsulated in variation points
- A variation point is not a single location in the code
- Corresponds to a subset of the requirements

Product Line Definition

- *A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.*

A frequent misconception is that the core assets, the reusable pieces, are the product line. As you can see from the definition, the product line comprises the products.



Product Line Definition - 1

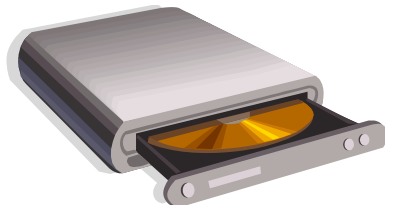
- *set of software-intensive systems*
  - *The product line is the products*
  - *The product line organization produces the products*

- *Set of airline reservation systems*
- *Software controllers for diesel engines*
- *Ground satellite control software systems*

Product Line Definition - 2

- *common, managed set of features*
  - *Common – identifying ahead of time common features of the products and the variations in products*
  - *Managed – evolution is anticipated, variation is controlled, and the inventory of features is what we sell*

- *Data storage and management actions*
- *Image analysis techniques*
- *Information classification techniques*



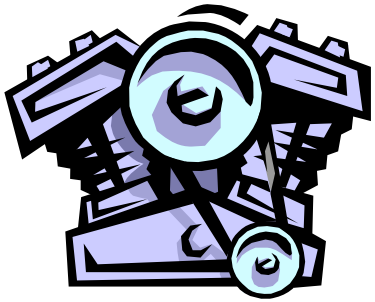
Product Line Definition - 3

- *particular market segment or mission*
  - *Focusing makes the percentage of commonality higher*
  - *The culture of the market segment determines specific quality levels*

- *Medical devices*
- *Video games*

Product Line Definition - 4

- *common set of core assets*
  - *A "core" asset is anything used to produce multiple products*
    - *Source code – yes, but also*
    - *Software architecture*
    - *Test infrastructure, test cases, and test data*
    - *Production plans*
    - *….*
  - *The assets are designed to handle the range of variability defined in the product line scope*
  - *Each asset is accompanied by an attached process, which explains how to use the asset in building a product*
- *Implementation of doppler compensation algorithms*
- *Test scenarios for engine controllers*



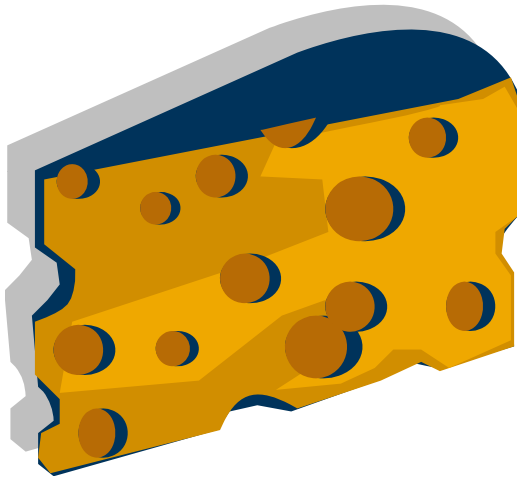
Product Line Definition - 5

- *prescribed way*
  - *A production strategy coordinates the business goals with the development of core assets and products*
  - *A production plan describes the way in which each product is to be produced*

- *Architecture-centric management*
- *Traditional programmer-centric code development*
- *Model-driven automatic code generation*

Product line architecture

- The product line architecture is the architecture for a family of systems
- Is more abstract, not every thing is completely defined
- There are holes in its specification, but the architecture constrains how the holes can be filled



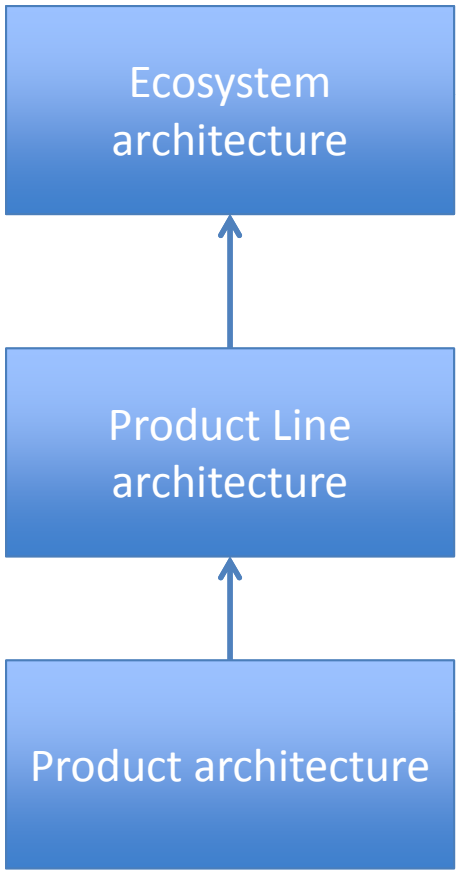
Multiple levels

- Product family :
  - Is a set of products with many commonalties and few differences.
  - Is intra-organizational.
- Product Populations:
  - Is a set of products with many commonalities but also many differences.
- Product Line:
  - Is a top-down, planned, proactive approach to achieve reuse of software within a family (or population, see the next section) of products.

Abstraction

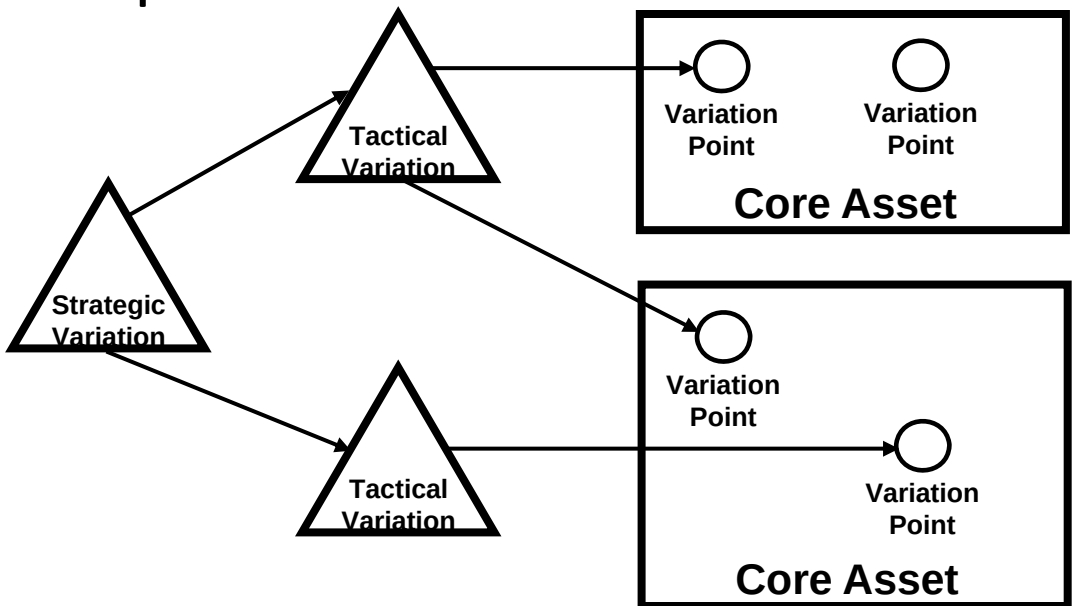
- The broader the scope of the architecture the higher the level of abstraction.
- The higher level of abstraction the less specific the details of the architecture.

- http://www.star.cc.gatech.edu/documents/PeterAbowd/SEI.pdf



Variation

- Products vary from one another in specific ways - the allowable contents of the holes in the architecture.
- Strategic variations at the business unit level.
- Tactical variations at the technical manager's level
- Variation points at the implementation level.



Variation mechanisms

- An instance of the architecture resolves certain variations
- Mechanisms
  - One system definition extends another
  - A system definition is included or excluded
  - Subprograms have parameters that are not primitive data types

Variation Mechanisms

- replacement, omission, and replication of architectural elements
- object-oriented (OO) techniques
  - inheritance
  - specialization
  - delegation
  - application frameworks
- parameterization (including macros and templates)
  - Special case: compile-time selection of different implementations or implementation fragments (e.g., *#ifdef*)
- generation and generators
- aspect-oriented programming
  - an approach for modularizing system properties that otherwise would be distributed across modules

Binding time

- The reason that some variation is not resolved is because the binding time for the variation is after architecture instantiation time
- The binding time is partially determined by the architect
- To do this
  - Who will do the binding?
  - When do they touch the system?
  - For example, a marketing person decides a feature is included – can only happen at requirements time

Binding Times

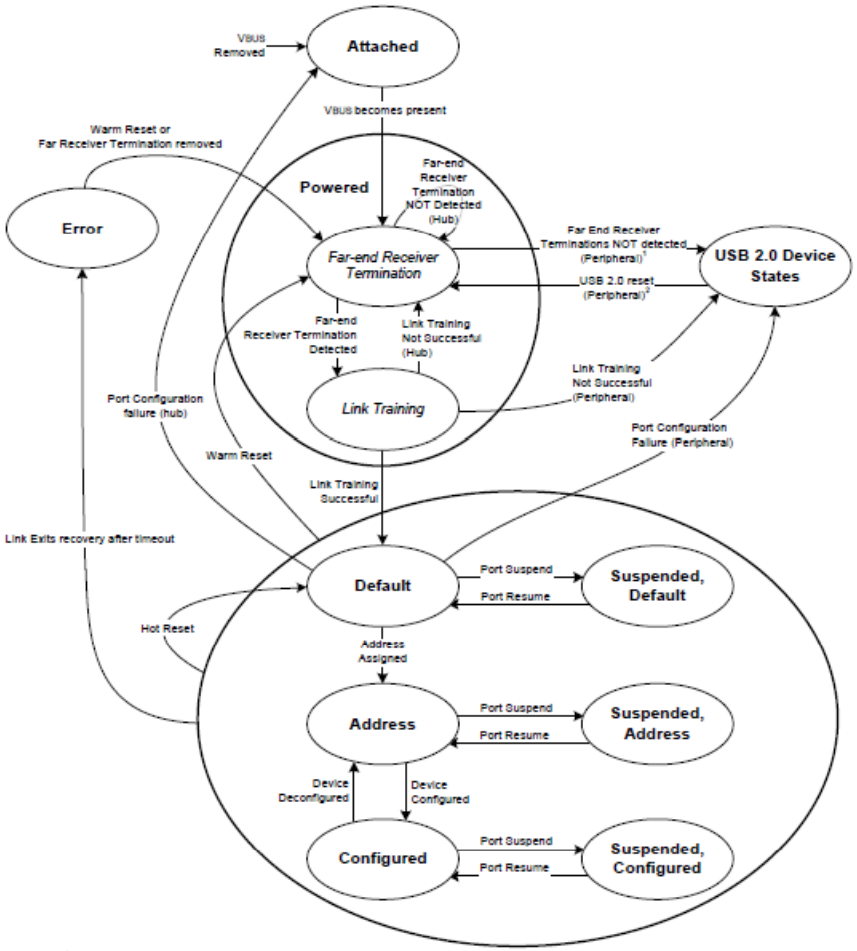
- Source reuse time. Decisions bound when reusing a configurable source artifact
- Development time. Decisions bound during architecture, design, and coding
- Static code instantiation time. Decisions bound during assembly of code just prior to a build
- Build time. Decisions bound during compilation or related processing
- Package time. Decisions bound while assembling binary & executable collections
- Customer customizations. Decisions bound during custom coding at customer site
- Install time. Decisions bound during the installation of the software product
- Startup time. Decisions bound during system startup
- Runtime. Decisions bound when the system is executing

Eliminating variability

- Some apparent variability can be reduced to commonality
  - A standard interface can be placed between the commonality and the apparent variability with the result that we don’t care what is on the other side of the interface. The BlueTooth interface for example.

USB state machine from standard spec

We do worry about conformance of the architecture to abstract specifications such as standards.



Telematics variations

- Color/trim packages
- Types of infotainment devices
- Apps
- Towing packages
- Power operated doors

But what about variations in quality attribute levels?

- One product needs to be airworthy certified but others do not
- One needs real-time performance another does not
- One must be secure another one does not

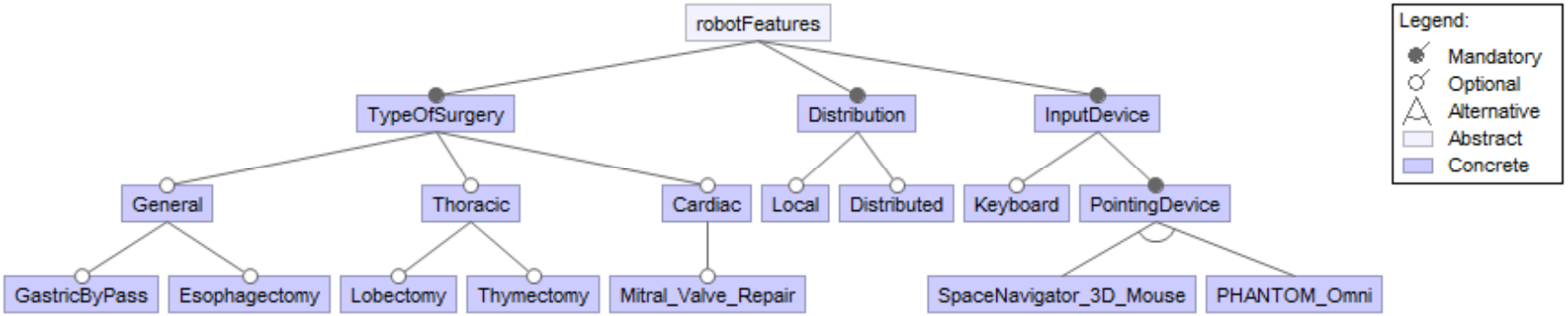
What to do?

- Would you
  - Make everything meet the toughest standard?
  - Re-implement all the assets?
- Tactic: reduce and isolate – encapsulate the section that differs among products; refactor when possible to reduce the area; hide behind interfaces

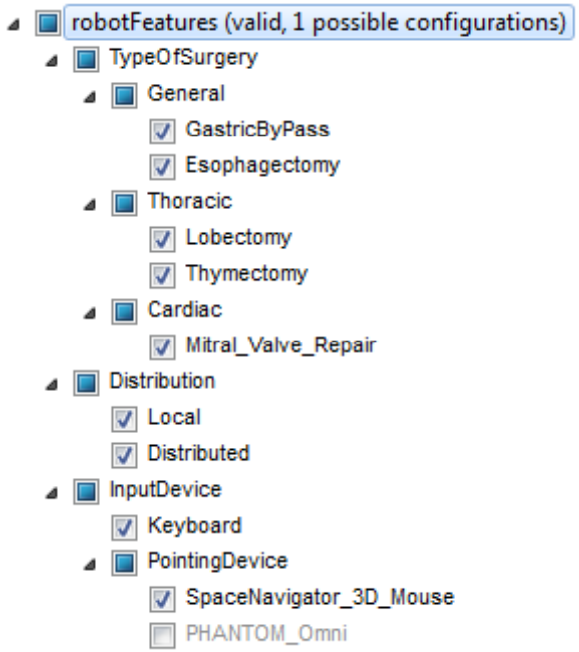
Use cross cutting techniques

- ‘Aspects’ cut across the system decomposition
- Other language idioms such as “mix-ins” also cross cut
- Look for a technique where fragments are maintained separately

Feature model



Configuration editor



Here's what you are going to do…

- Submit a new version of the architecture that addresses the variations in your telematics project
- Pay particular attention to variation in quality attributes
- Include a readme file that describes the changes you make
- By March 4th at 11:59pm
  - Identify at least two variation points and the possible variants
  - Determine an appropriate variation mechanisms and add to your architecture description
  - Create a feature model for your system
    - http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/
  - provide a new release of your architecture that
    - Represents the variation we want in our system.
    - Addresses any issues raised during the ATAM