

Software Product Line - Economics and Technical Overview

Presented to INCOSE
by

Dharmalingam Ganesan, Mikael Lindvall

Fraunhofer Center for Experimental Software Engineering Maryland (FC-MD)

Contents

- Introduction of Fraunhofer
- Overview of SPL
 - Economic model
 - Architecture and SPL
 - Migration to SPL
- Discussion

Fraunhofer Center – Maryland (FC-MD)

- Applied Research and Tech Transfer
- Affiliated with
 - University of Maryland, College Park and
 - Fraunhofer Germany
- Close ties to
 - NASA, FDA
- Contract research for industry, government
- Clients/partners:
 - Bosch, Biofortis, DOD, FDA, JHU, JHU/APL, NASA, Optimal etc.

Fraunhofer Center – Maryland (FC-MD)

- Focus on Software Engineering
 - Especially Software Quality
 - Quality assurance -> Build quality in
- Applied research in
 - software product line, software architecture,
 - verification & validation,
 - process improvement, and measurement



About Us

The Park

Location

News + Events

Home

Fraunhofer Center – Maryland (FC-MD) Located at MSquare



University of Maryland Research Park

A DYNAMIC LOCATION



M Square, the **University of Maryland Research Park**, is a dynamic location for science and technology companies. Adjacent to the **University of Maryland**, ranked among the nation's top 20 public research universities, and located just 8 miles from the Nation's Capital, the **University of Maryland Research Park** offers flexible space

M SQUARE IS:

- Affiliated with University of Maryland
- Minutes from I-95 and I-495
- Adjacent to MARC and Metro



SEARCH

locations from incubator space to large office space to suit options for larger technology clients in Maryland's largest research park.

© 2011 Fraunhofer USA, Inc.

A Tasty Product Line

FEATURED ITEMS



NEW! STUFFED CRUST PAN PIZZA

New! Stuffed Crust Pan Pizza – with a ring of melty cheese stuffed deep into our golden crispy Pan Pizza crust.

[Order Now](#) » | [View Larger Image](#) »



NEW! BIG EAT TINY PRICE™ MENU – STARTING AT \$5

Is your appetite bigger than your wallet? Choose from 4 BIG eats – yours for a tiny price.

[Order Now](#) » | [View Larger Image](#) »



NEW! TRIPLE MEAT ITALIANO

The Savory Flavor of Three Italian Meats: All-natural pepperoni, all-natural Italian sausage, and baked ham.

[Order Now](#) » | [View Larger Image](#) »



NEW! DAN'S ORIGINAL

The Hometown Classic for 50 Years: All-natural pepperoni, all-natural Italian sausage, and fresh white mushrooms.

[Order Now](#) » | [View Larger Image](#) »



NEW! SPICY SICILIAN

An Italian Tradition with a Fiery Kick: 100% real seasoned beef, all-natural Italian sausage, fresh sweet red onions, and spicy jalapenos.

[Order Now](#) » | [View Larger Image](#) »



NEW! HAWAIIAN LUAU

An Island Escape: Baked ham, tropical pineapple, and smoked bacon.

[Order Now](#) » | [View Larger Image](#) »

SPL – Examples ...

Nokia Mobile Phones

Product lines with 25-30 new products per year versus 5 per year originally.

Across products there are

- varying number of keys
- varying display sizes
- varying sets of features
- 58 languages supported
- 130 countries served
- multiple protocols
- needs for backwards compatibility
- configurable features
- needs for product behavior
- change after release



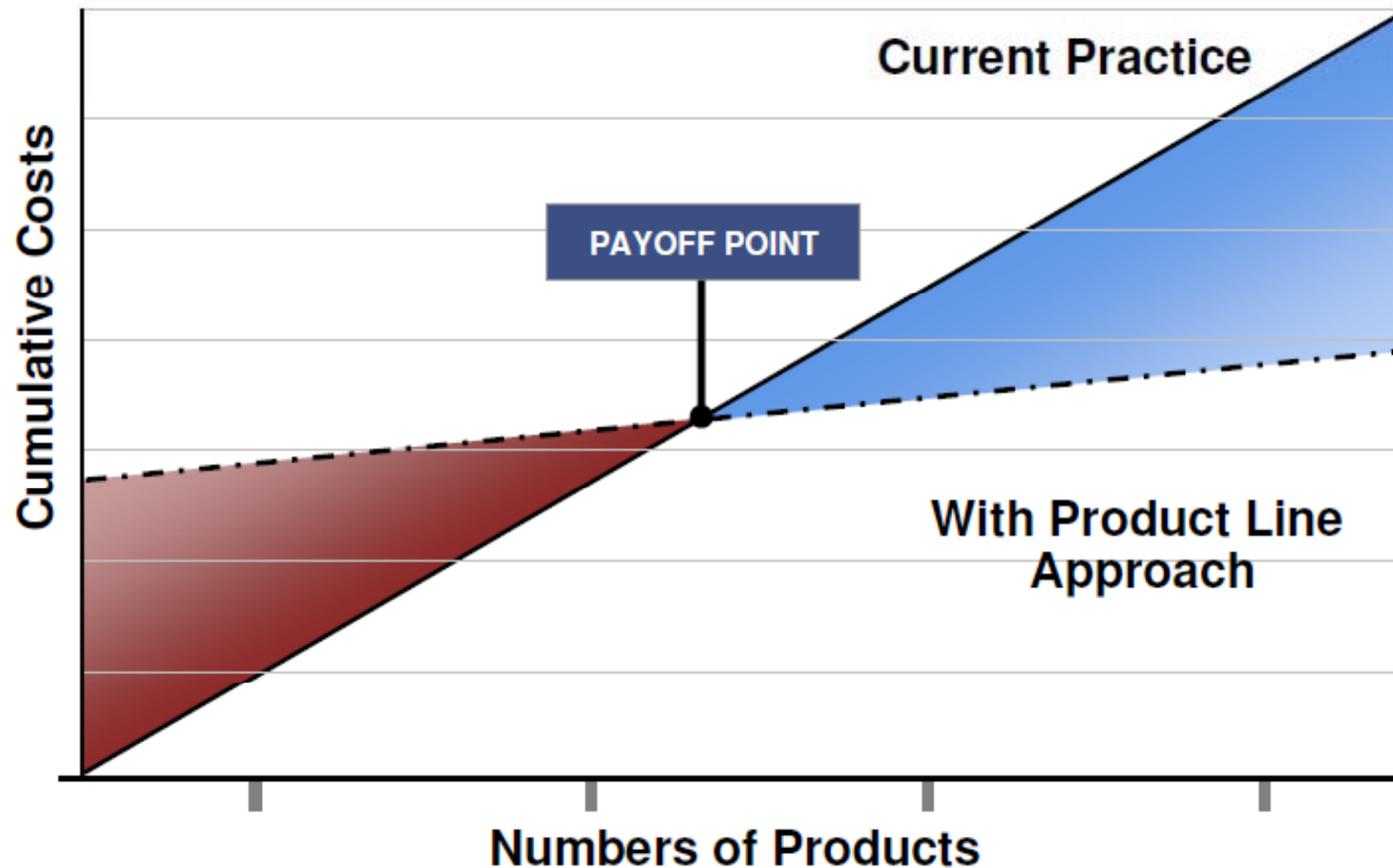
When should we consider PL?

- When creating a set of new, similar products from scratch (unusual)
- When faced with several similar products that are costly to maintain (very common)
 - Similar product markets emerged over time
 - The company acquired another company and ended up with a set of similar products

Motivation for SPL

- Achieve large scale productivity gains
 - Improve time to market
 - Enable mass customization
 - Get control of diverse product configurations
 - Improve product quality
 - Increase predictability of cost, schedule, quality
- *Success is in the product line **architecture!***
- More about this later

Economics of SPL



Weiss, D.M. & and Lai, C.T.R..

Software Product-Line Engineering: A Family-Based Software Development Process 10

Reading, MA: Addison-Wesley, 1999. © 2011 Fraunhofer USA, Inc.

Calculating ROI

- A cost model developed at Fraunhofer in collaboration with other organizations
- **Costs:**
 - for developing software for reuse
 - for customizing reusable software
 - for developing product specific features
 - for training to introduce a SPL

Components of the cost model

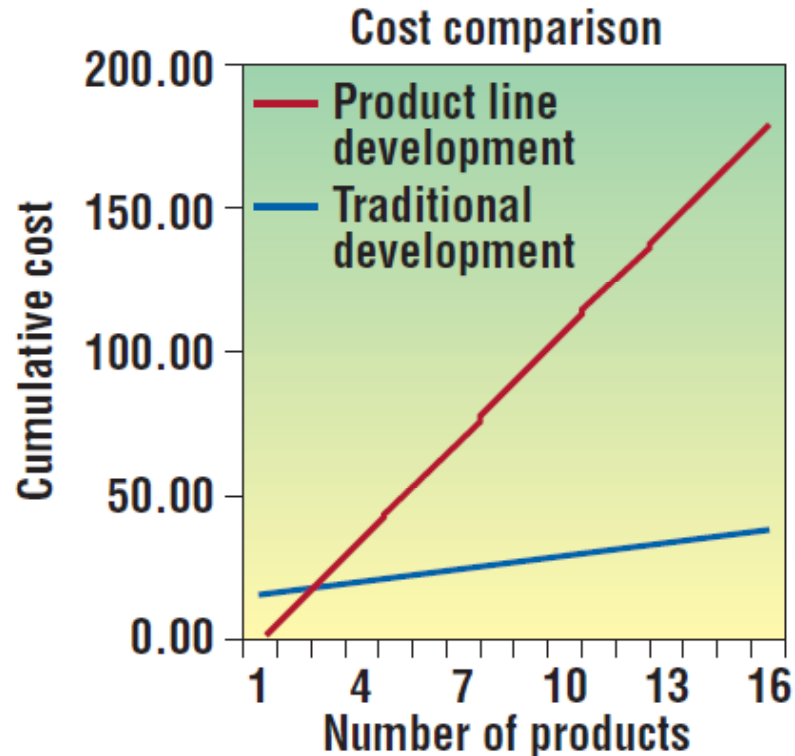
- C_{org} – The cost to adopt the PL approach
- C_{cab} - The cost to develop core assets
- C_{unique} - The cost for unique features
- C_{reuse} - The cost to reuse core assets
- C_{prod} – The cost of a traditional product

ROI = Cost savings / Cost of investment

$$\frac{\text{Cost of old way} - \text{Cost of new way}}{C_{org} + C_{cab}}$$

ROI – when does PL pay off?

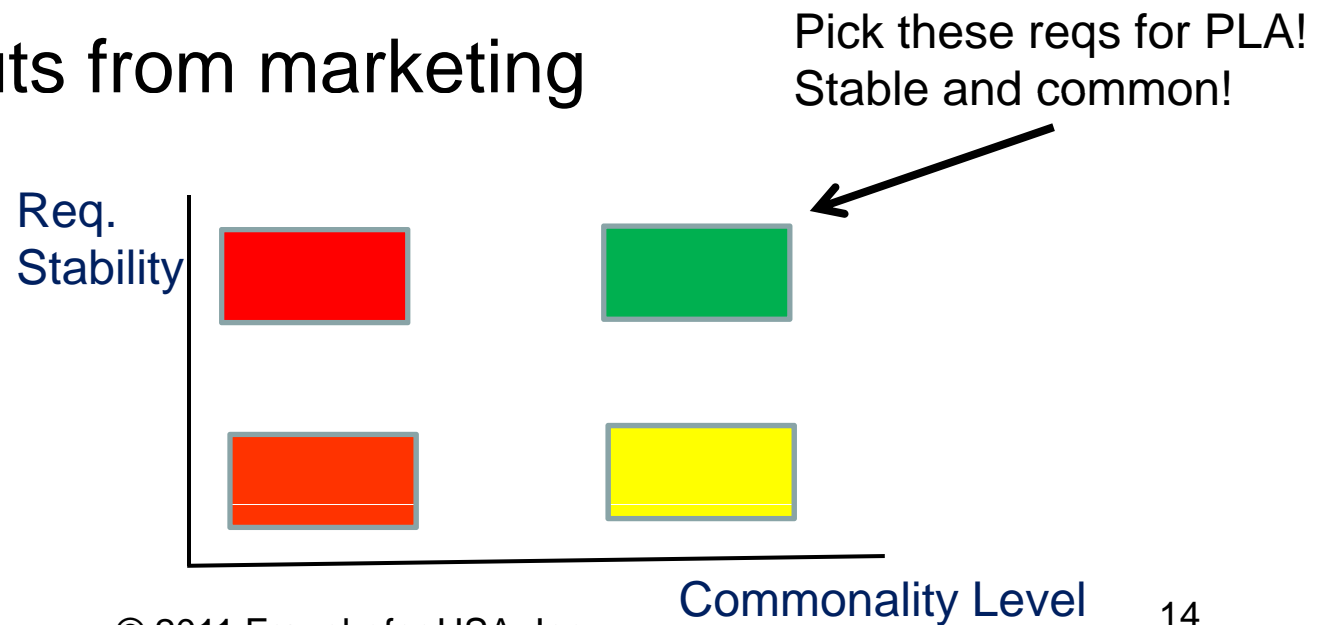
C_{prod} 12 C_{org} 2.4 C_{cab} 13 C_{unique} 0.72 C_{reuse} 0.84



Number of products	Product line development	Traditional development
0	15.40	0.00
1	16.96	12.00
2	18.52	24.00
3	20.08	36.00
4	21.64	48.00
5	23.20	60.00
6	24.76	72.00
7	26.32	84.00
8	27.88	96.00
9	29.44	108.00
10	31.00	120.00
11	32.56	132.00
12	34.12	144.00
13	35.68	156.00
14	37.24	168.00
15	38.80	180.00

Requirements and SPL

- Commonality and Variability Analysis
 - Develop a feature-product matrix
 - Identify common and variable features
 - Identify **features worth for reuse**
 - Need inputs from marketing



Organizational structure

- Our project partners have different organizational models
- Model 1:
 - Separate core team for reusable assets
 - Separate product teams to build products
 - Core team funded by corporate
- Model 2:
 - Virtual core team
 - Product teams contribute to reusable core
 - No explicit funding

Organizational Structure ...

- Model 3:
 - “Dynamic” project based team structure
 - No fixed core and/or project teams
- Model 4: Some mix of above models
- Factors influencing org. structure for SPL:
 - Domain and technical competency
 - Global software development
 - Company culture
 - Open source and external collaboration
 - Software architecture

Architecting for SPL

- Two fundamental needs in defining a PLA
 - to be able to generalize or abstract from the individual products to capture the important aspects of the product line
 - to be able to instantiate an individual product architecture from the product line architecture

Architecting for SPL ...

- Use Architectural styles to handle variation
- For example, a Pipe-and-Filter style can be used to produce different product instances
 - Select Filters of interest and derive a new product
- Use a Layered architectural style to “hide” variations (e.g., in OS and Hardware types)
 - Important to define Abstract APIs with alternative implementations
- Use a Publish-Subscribe style for:
 - Substituting various publishers and subscribers
 - Plug-in new publishers and subscribers
 - Run-time adaptation

Variation within Components

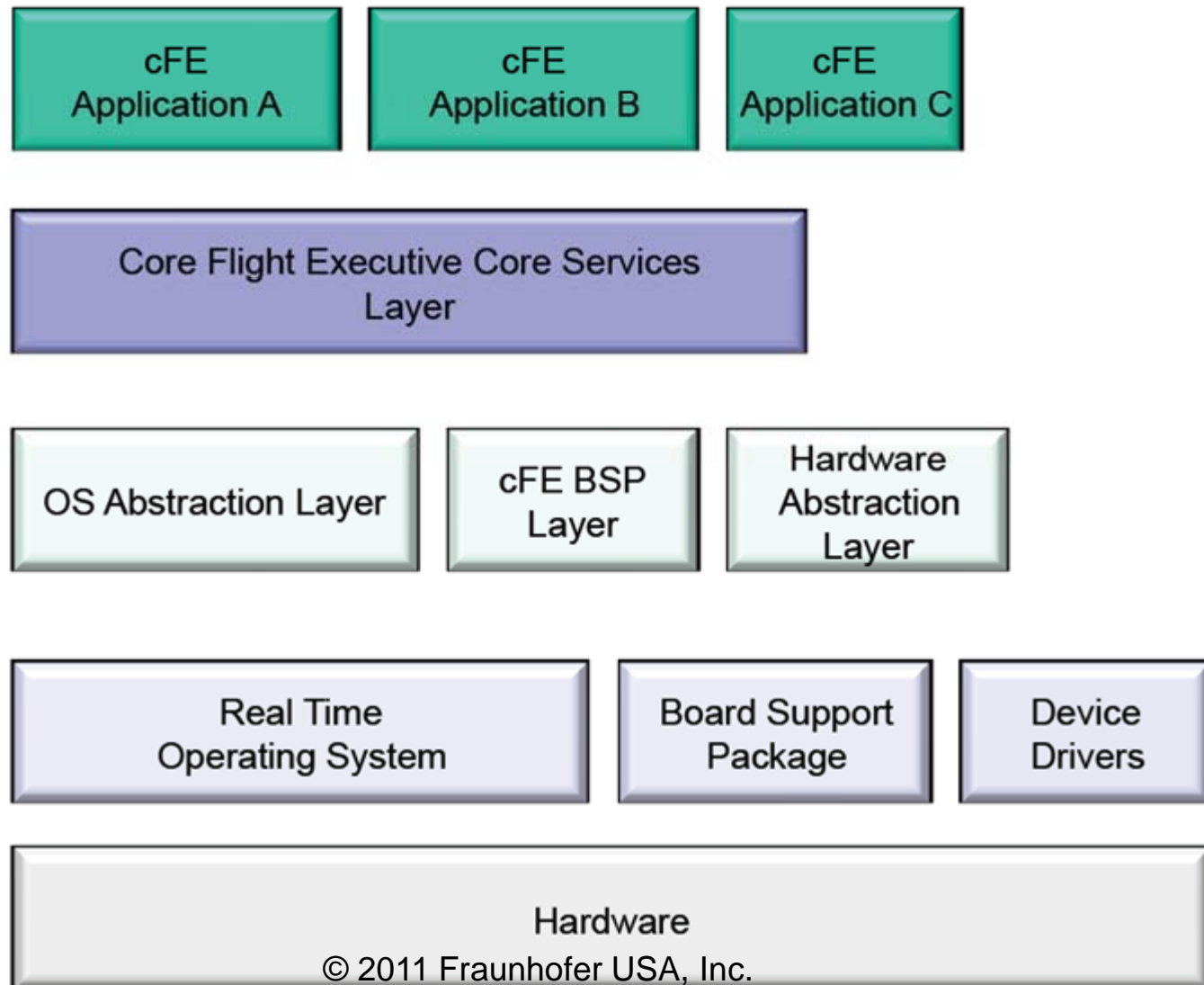
- Configurable component (e.g., Properties, #ifdef)
 - Variation is supported within the component
 - Select appropriate values for config. parameters
 - Enable/disable parts of code using config. parameters
- Extensible component (e.g., Inheritance)
 - Consists of an extensible base
 - Variable parts are separate sub-components
 - Generic and specific functionality are separated
- Abstract component (e.g., Java Interface)
 - Only the interface is provided
 - Different products must provide own implementation
 - Suitable when implementation differs a lot

Common Architectural Styles for SPL

- Layer
- Pipe-and-Filter
- Publish-Subscribe
- Implicit-Invocation

Layered Style example

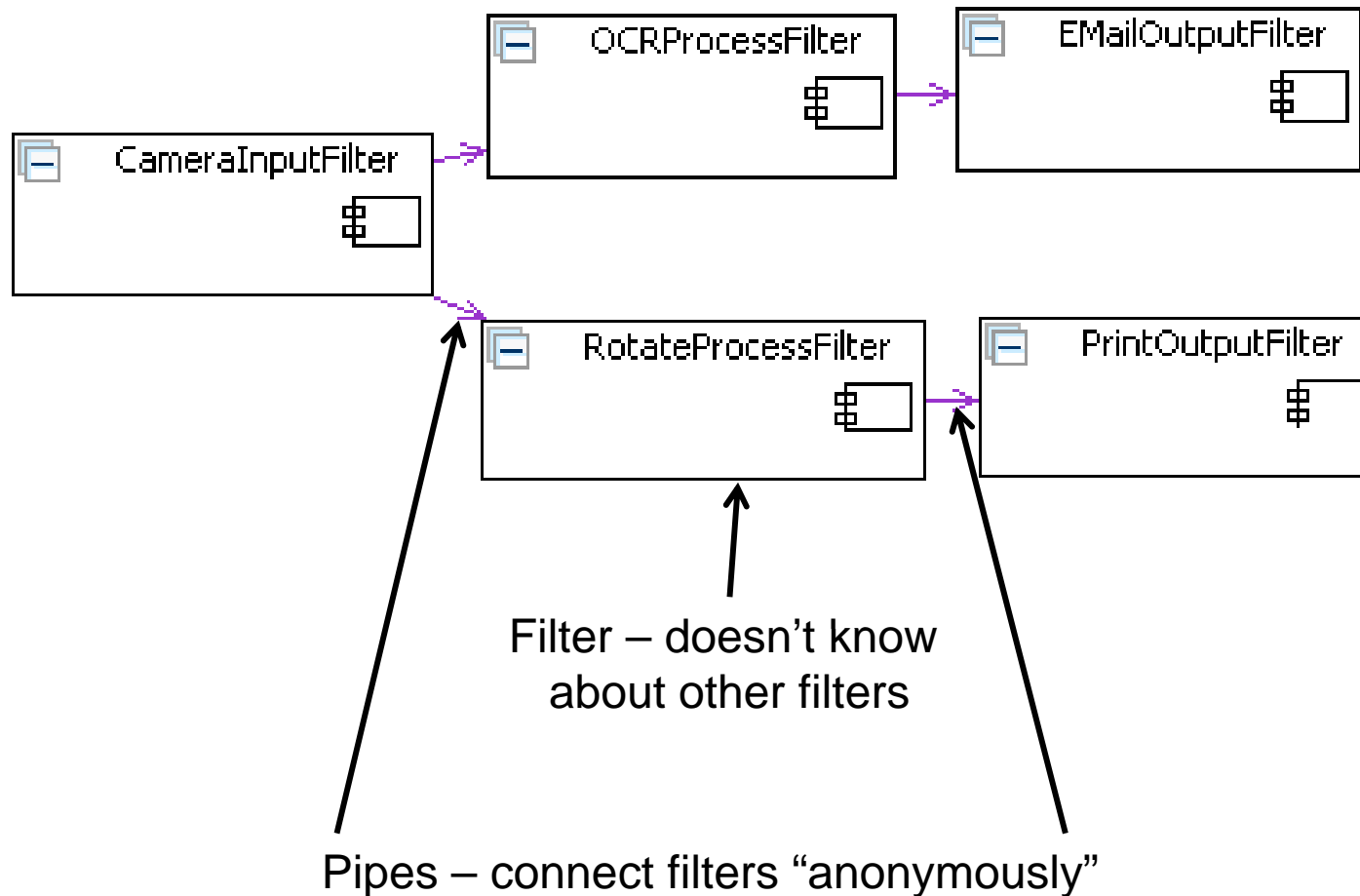
core Flight Executive (cFE) software



Layered Style - Benefits

- Enables incremental testing from the bottom to top layer
- Plug-out and Plug-in of a layer with a new layer conforming to the same API
 - Helpful in producing variants of a product
- Helps distribute the work to different teams
 - Often different teams work on different layers

Pipe-and-Filter Style - Example



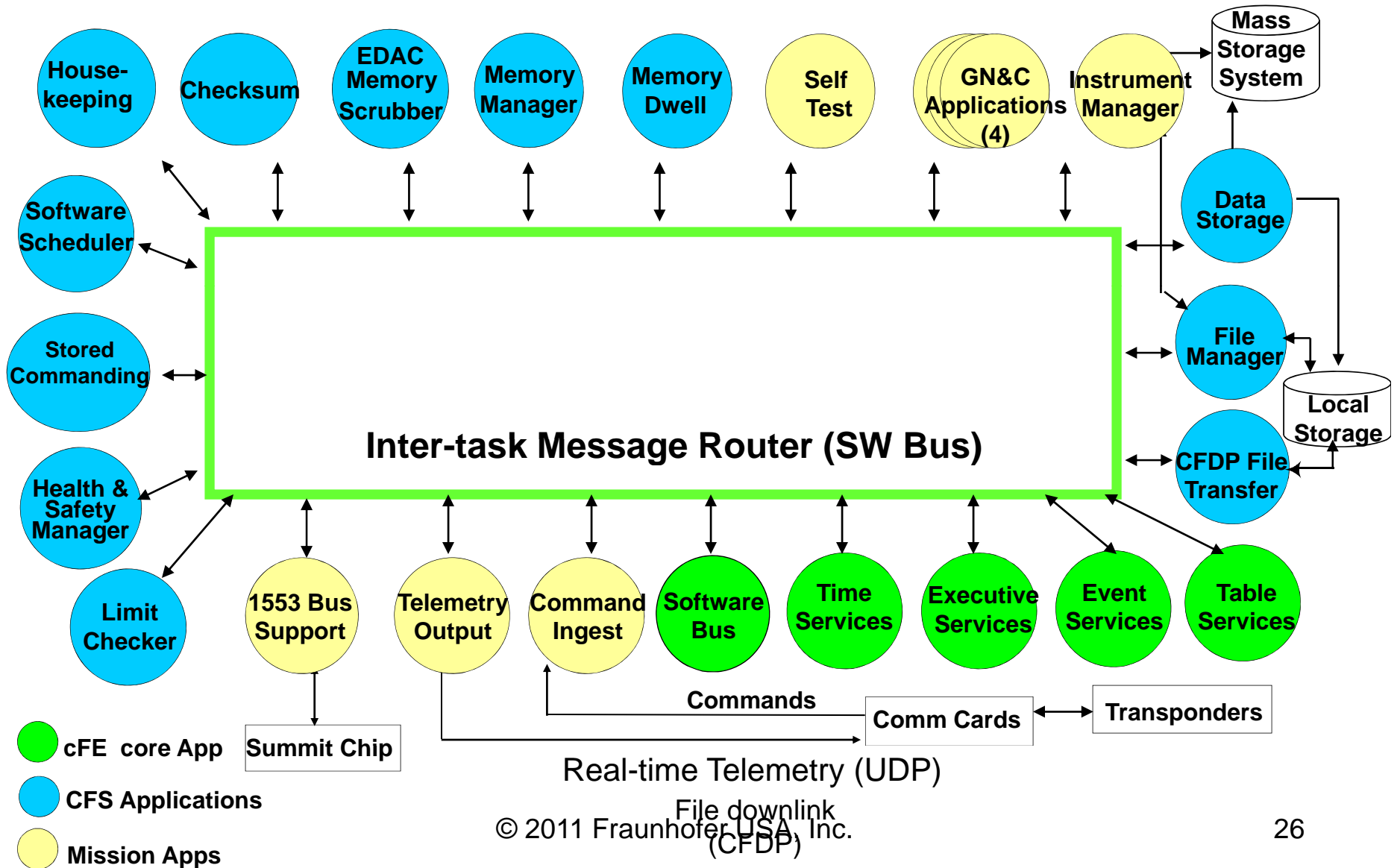
Pipe-and-Filter Style - Benefits

- Filters are not directly coupled
- Filters can be composed with different filters to produce new configurations
 - Helps in producing system variants
- Filters can be tested independently
- Filters can be executed concurrently

Publish-Subscribe style

- Components don't interact directly
- Components publish messages to a message bus
- Components subscribe to messages of interest
- The message bus delivers messages to components

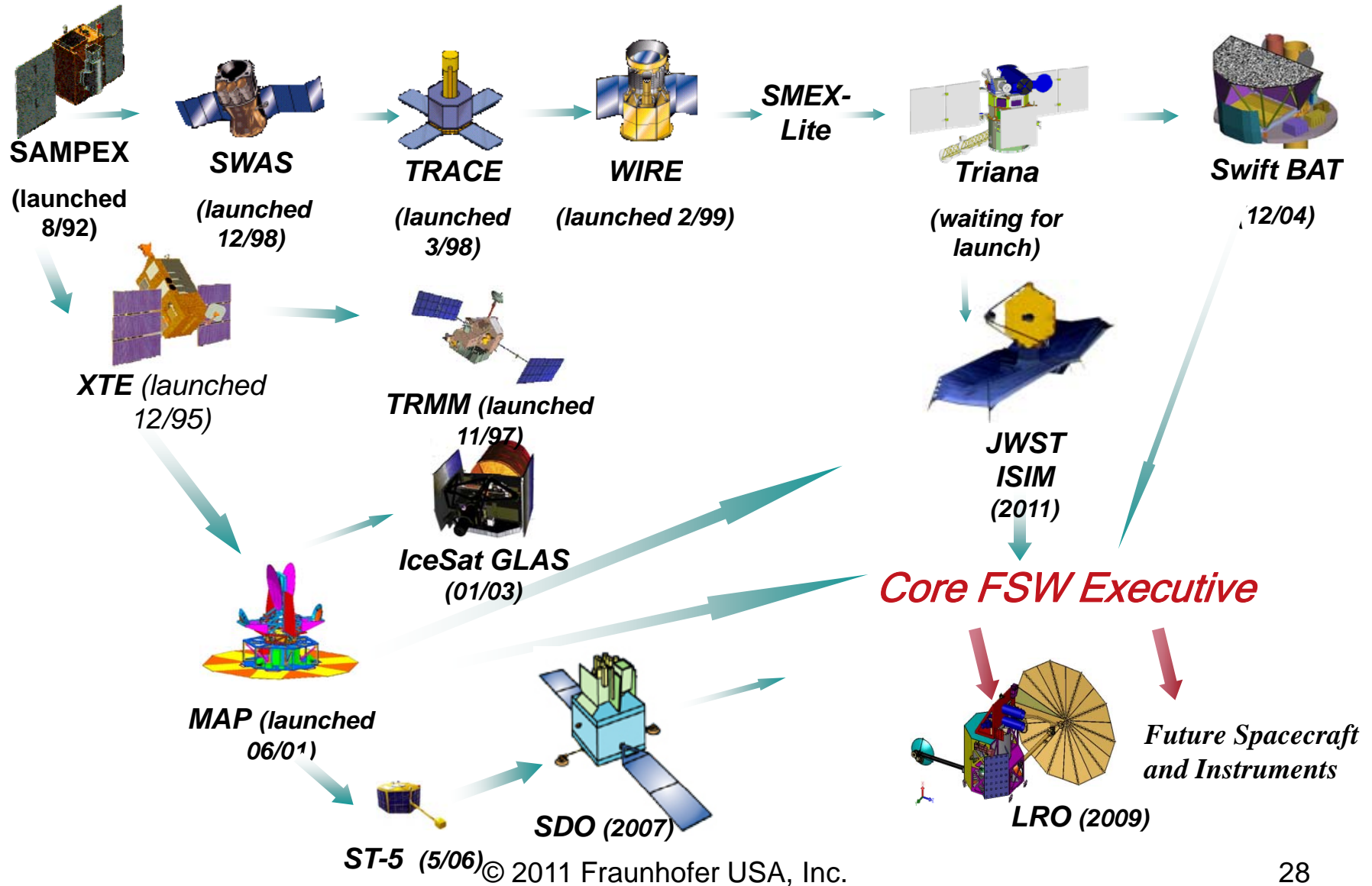
Publish-Subscribe



Publish-Subscribe - Benefits

- Components are not coupled to each other
- New components can be added/removed at run-time
- Components can also dynamically unsubscribe to subscribed messages
- New variants of the system can be produced with different configurations of components

Migration to SPL-an example



Migration to SPL – Some scenarios

- Scenario 1: Your company owns two or more similar products
 - Either in the same or different business units
- Scenario 2: Your similar products were from the common base but branched over time into independent products
- Scenario 3: Your company bought other companies producing similar products

Migration to SPL

- Extract requirements of existing products from available artifacts
- Extract architectures of existing products from source code
- Develop architecture-based migration strategy based on
 - how existing products are structured
 - where common and variable parts are

Migration to SPL

- Fraunhofer has developed methods and tools for reverse architecting
 - Analyze the source code and extract as-built architecture of each product
 - Analyze the source code to suggest common code among existing products
- Our tools can semi-automatically extract static modular as well as runtime structure of existing products
 - Tools were used in several industrial scenarios

Migration to SPL –options

- Option 1: Loose integration of products
 - E.g. using databases, wrappers, etc.
- Option 2: Merge existing products into one common architecture
 - Bring common components together
- Option 3: Discard all but one product
 - Generalize one product to cover existing & future products
- Option 4: Start from scratch!

Testing SPL

- Testing core assets is important because they are reused in several products
- It is a challenge to test a SPL because of variation or configuration points and interactions
- Testing strategy depends on several factors including:
 - architectural styles
 - way variants are handled

Testing SPL ...

- Unit tests, e.g. JUnit, are popular, for example, because such tests can be automatically executed
- However,
 - Setting up test cases takes time and effort
 - Test cases can be brittle and hard to maintain
 - Covering all behaviors including “corner

Generating Test Cases from API

- Incrementally model the behavior of the software to be tested
 - Expected results are embedded by the model
- Based on the model, the test generator automatically generates test cases
- Any type of test cases can be generated!
 - JUnit, XUnit, soapUI, Selenium, etc etc!

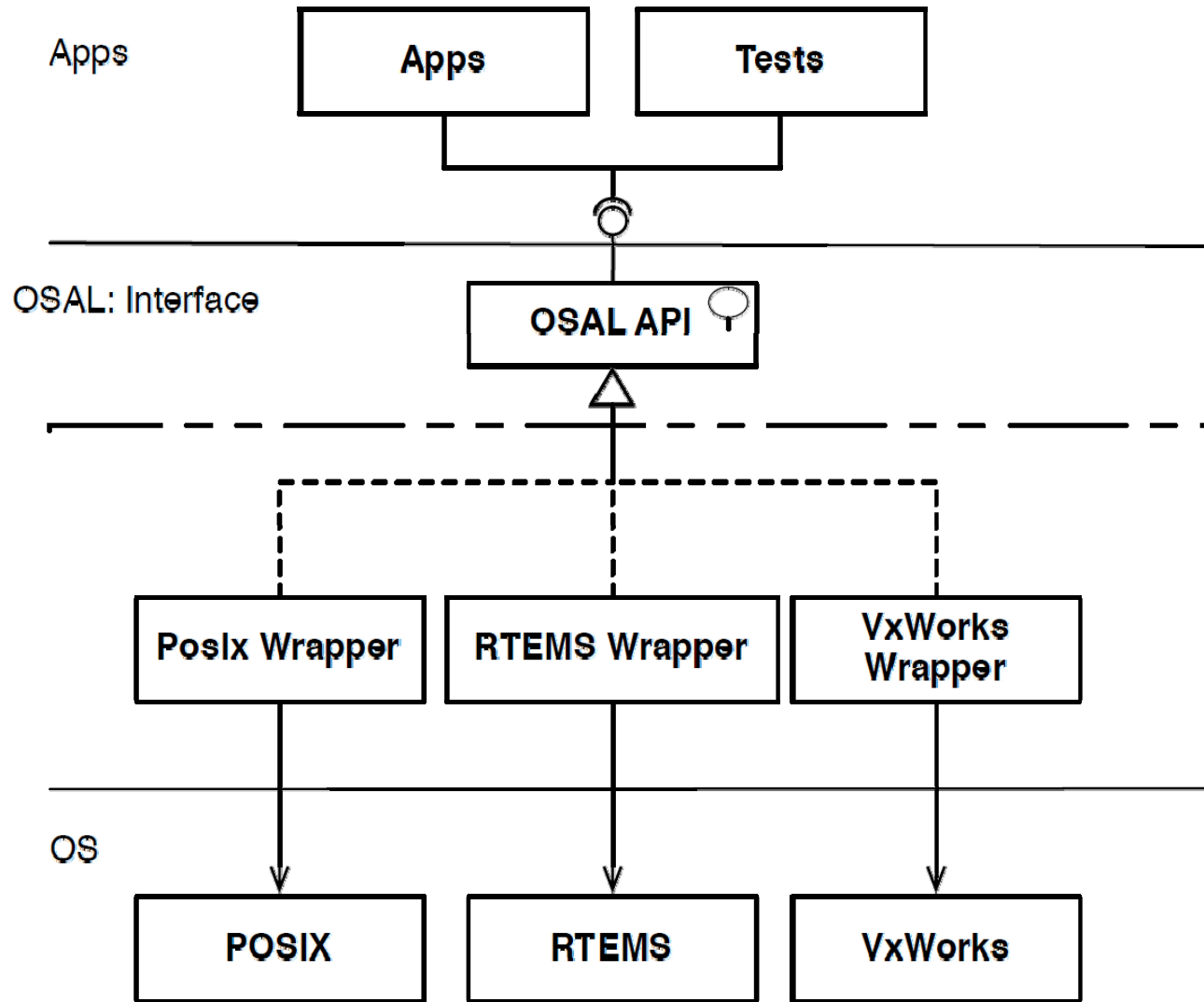
NASA OSAL – an example

- **Operating System Abstraction Layer**
- Isolates flight software from real time operating systems and hardware
- Provides “Write once, run everywhere (somewhere)” at compile level
- Used for mission critical embedded systems

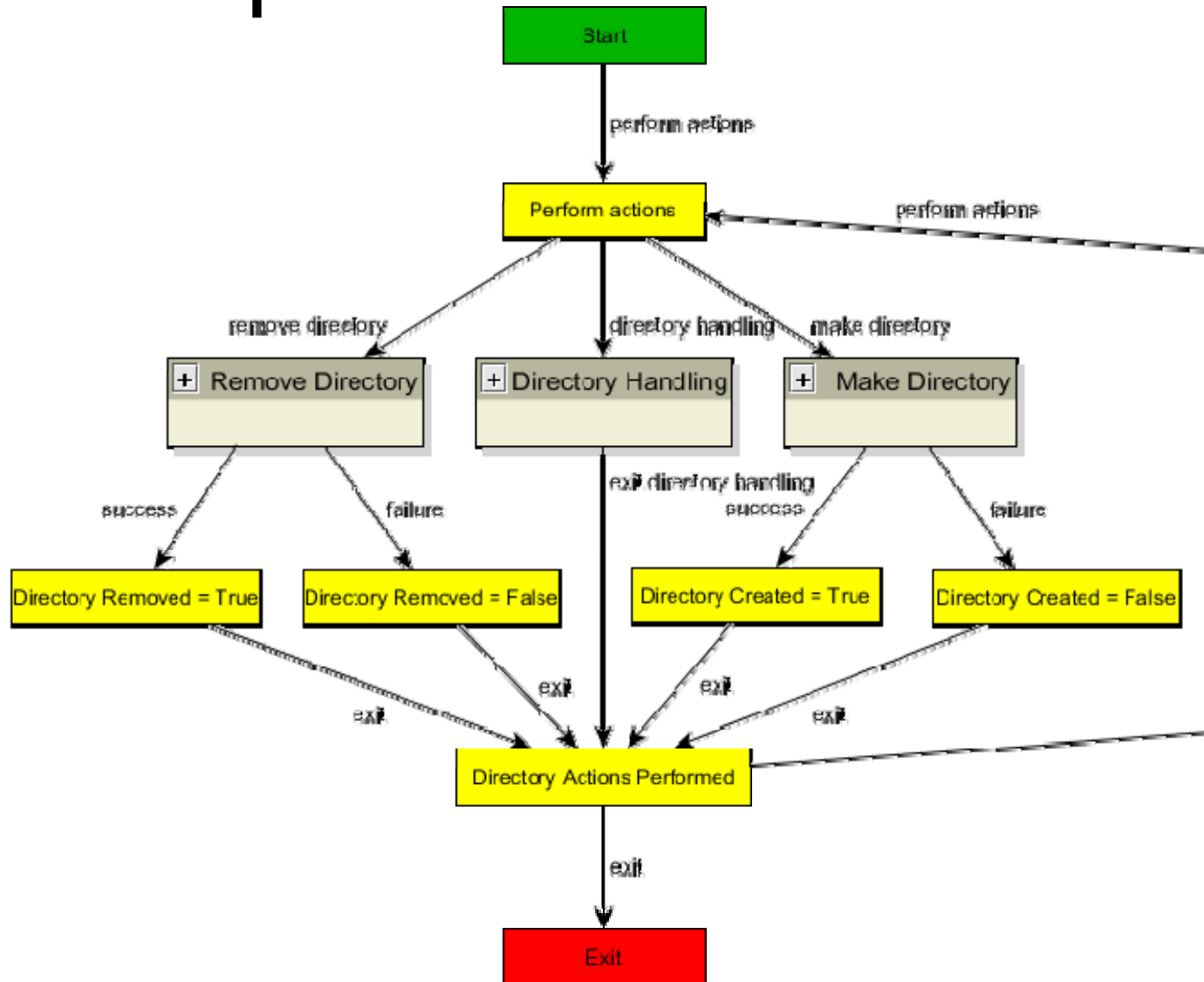
Benefits

- Generate several hundred (or thousands) ready-to-run test cases
- The test code embedded in the model is reused many times
- Immediate return-on-investment
 - You get test cases for your current model that can be part of the daily build
- No editing of test cases
 - Source code changes means updating the model and regenerating test cases
- A software product line that is very well tested!

NASA OSAL – Architecture

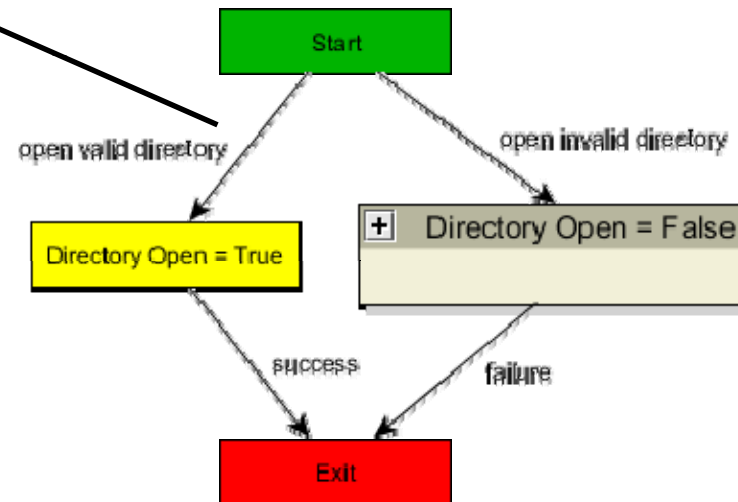


Example of an OSAL model



Example of an OSAL model

```
pointer = openDirectoryValid();  
CuAssertTrue(tc, pointer !=  
NULL);
```



Some Cons of SPL

- Dependency on core team!
- If the core is not flexible, product teams may not like it
- Danger of core being skewed towards some products
- Issues with quality properties not managed well across product line
- Challenges due to uncertainty in requirements and common features

Conclusion

- SPL is a mixture of economics, organization, and architecture
- Migration to SPL is often context specific
- Reverse architecting of existing products would help in planning for migration
- Testing is an important challenge due to the configuration space

Fraunhofer Services

- Evaluate the state of existing products to develop a migration strategy
- Information retrieval based analysis of requirements
 - To identify common and variable features
- Architectural analysis after SPL is deployed
 - To make sure architecture is evolving in the right direction
- Model-based Automated Testing of SPL

References: Fraunhofer papers

- *Calculating ROI for software product lines.* IEEE Software, May-June 2004
- *Predicting return-on-investment for product line generations.* Software Product Line Conference (SPLC), 2006
- *Architecture-Based Unit Testing of the Flight Software Product Line.* SPLC 2011

References: Fraunhofer papers

- *Architectural Analysis of Systems Based on the Publisher-Subscriber Style.* WCRE 2011
- *Verifying architectural design rules of the flight software product line.* SPLC 2009
- *Defining a strategy to introduce a software product line using existing embedded systems.* EMSOFT 2006:

References: Fraunhofer papers

- *Comparing Costs and Benefits of Different Test Strategies for a Software Product Line: A Study from Testo AG. SPLC 2007*
- *Discovering Organizational Aspects from the Source Code History Log during the Product Line Planning Phase--A Case Study. WCRE 2006*
- *Asset Recovery and Their Incorporation into Product Lines. WCRE 2005*

Contacts

- Dharma (dganesan@fc-md.umd.edu)
- Mikael (mlindvall@fc-md.umd.edu)