

Product Line Engineering Lecture – PL Infrastructures III (5)

Dr. Martin Becker

martin.becker@iese.fraunhofer.de



Schedule - Lectures

Date	Content	Time	Location
29-Oct-10	Introduction	15:30 - 17:00	ZD4.06 J. Nehmer (IESE)
5-Nov-10	Scoping	15:30 - 17:00	ZD4.06 J. Nehmer (IESE)
12-Nov-10	PL Infrastructure I (Variability Modelling)	15:30 - 17:00	ZD4.06 J. Nehmer (IESE)
19-Nov-10	PL Infrastructure II (Variability Realization)	15:30 - 17:00	ZD4.06 J. Nehmer (IESE)
26-Nov-10	no lecture	15:30 - 17:00	ZD4.06 J. Nehmer (IESE)
3-Dec-10	Configuration Management	15:30 - 17:00	ZD4.06 J. Nehmer (IESE)
10-Dec-10	PL Economics and Approaches	15:30 - 17:00	ZD4.06 J. Nehmer (IESE)
17-Dec-10	Requirements Engineering	15:30 - 17:00	ZD4.06 J. Nehmer (IESE)
7-Jan-11	PL-Architectures I	15:30 - 17:00	ZD4.06 J. Nehmer (IESE)
14-Jan-11	PL-Architectures II	15:30 - 17:00	ZD4.06 J. Nehmer (IESE)
21-Jan-11	Component Engineering	15:30 - 17:00	ZD4.06 J. Nehmer (IESE)
28-Jan-11	Quality Assurance	15:30 - 17:00	ZD4.06 J. Nehmer (IESE)
4-Feb-11	Organizational Issues / Adoption	15:30 - 17:00	ZD4.06 J. Nehmer (IESE)
11-Feb-11	Reengineering / Variant Analysis	15:30 - 17:00	ZD4.06 J. Nehmer (IESE)

Schedule - Exercises

Exercises			
Date	Content	Time	Location
12.11.2010	Scoping, Variability Modeling	17:15 - 18:45	Z04.06 J. Nehmer (IESE)
10.12.2010	VM Realization, Configuration Management	17:15 - 18:45	Z04.06 J. Nehmer (IESE)
14.01.2011	PL Architectures	17:15 - 18:45	Z04.06 J. Nehmer (IESE)
21.01.2011	Component Engineering	17:15 - 18:45	Z04.06 J. Nehmer (IESE)
11.02.2011	Adoption, Variant Analysis	17:15 - 18:45	Z04.06 J. Nehmer (IESE)

--- Recap
Product Line Infrastructure
Part II: Variability Realisation ---

**How to realize
variability resolution support?**

Core Assets

Content:

- Product Model, Process Model, Resource

Lifecycle Phase:

- Requirements, System Design, Unit Design, Code, Image, Data, Test, Integration, Documentation, Configuration, Patch

Granularity:

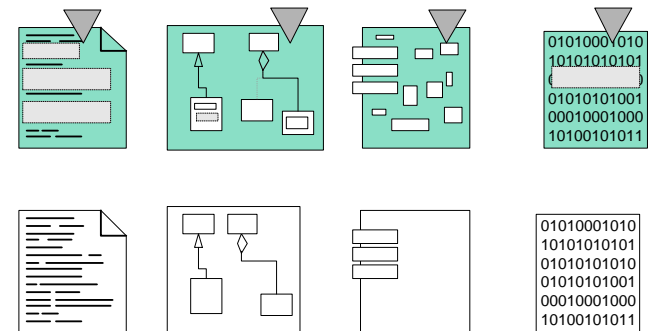
- Subsystem, Component, Folder, Document, Document Fragment / Element

Genericity:

- Generic, Specific

Data Type:

- Model, Structured Text (e.g. XML), Text, Binary



2 Layers of Variability Management

Legend:

- variation point
- optional variant
- alternative variants
- multiple coexist. var.
- refers to

Variability Model (e.g. PuLSE Decision Model (DM))

High-Level Variability Model

Decision	Resolut.	Low-level d.
Time Transm.	N, y	R1.Tim., R2.Tim., A1.Tim., ...
Detect. Type	tilt, drop, noise	R1.Det., R2.Det., A1.Det., ...
Sensors	posit., noise	R1.Sns., R2.Sns., ..., I1.Sns.

Requirements Variability Models

R1	Decision	VP	Resolut.
Time Transm.	</no>		N, y
Detect. Type	<Det.Mod>		Tilt, Drop, Noise posit., noise
Sensors	SensorType		

R2	Decision	VP	Resolut.
Time Transm.	Wireless Tr.		N, y
Detect. Type	Detection sub-feat.		tilt, drop, noise Posit., Noise
Sensors	Sensor-sub-feat.		

Arch./Design Var. Models

A1	Decision	VP	Resolut.
Time Transm.	time_trans.		N, y
Detect. Type	detector subclass		tilt, drop, noise

Realization Var. Models

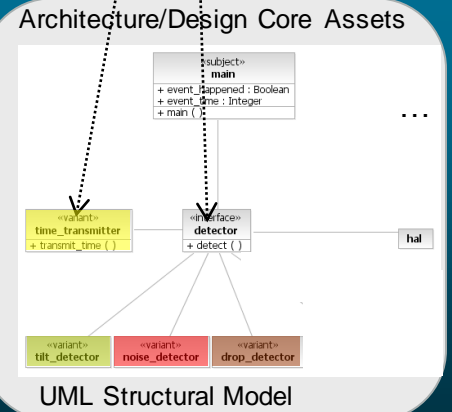
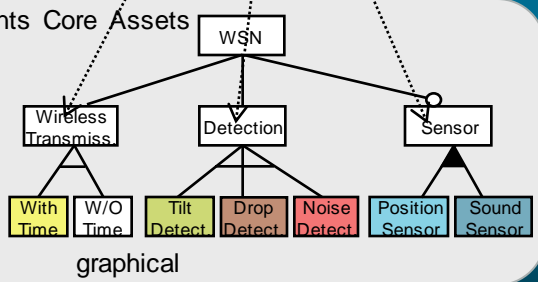
R1	Decision	VP	Resolut.
Sensors	HAS_?_SNS.		X_POS, SOUND

Requirements Core Assets

A WSN must have wireless transmission. Transmitted data has </no> timestamp.

A WSN must have detection capabilities. <DetectionMode> is detected.

A WSN contains sensors. The sensor is a <SensorType> sensor.



Realization Core Assets

```

void main() {
    init();
    #if HAS_X_PCS_SENSOR
    init_x_position();
    #endif
    #if HAS_SOUND_SENSOR
    init_sound();
    #endif
    while(true) {
        if(period_elapsed) {
            period_elapsed=false;
        }
    }
}
    
```

Core Asset Model

UML Structural Model

Condit. Compiled C Code

Summary: Variability Realisation:

What has to happen after the customer has selected his product?

Identify affected locations

- Markup, List Points, Point Cuts

Understand context

- Provide background knowledge

Provide appropriate realisation

- Provision of asset fragments, automated selection, generation, provision of realisation knowledge

Integrate the realisation variant into the Core Asset

- Automated integration (inclusion) of parts

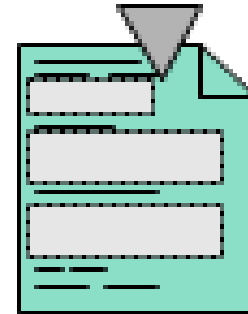
Manage the core asset and the variants

- Separate core and variant (change), support diff and merge

Variation Point

Variation Point ::=

identifies **a location**
at which **variation** will occur
within **core assets**.



Goals: 1) to highlight where variant elements occur
(which makes variation easy to see and control);
2) to improve traceability of variability
(requires that goal 1 has been fulfilled).

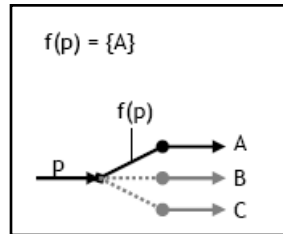
Variability Mechanism

Variability Mechanism ::=

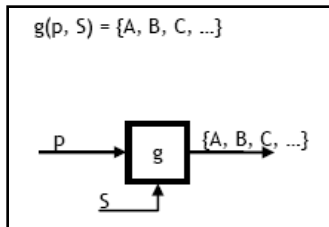
is a **particular way**
of **realizing variation**
in **core assets**.

- Goals: 1) to efficiently package common & variant elements;
2) to reduce evolution effort.

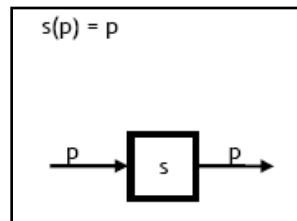
Variability Mechanism Primitives



selection



generation



substitution

■ Selection

- selecting predefined variants
- e.g. component wiring, if-blocks, if-defs

■ Generation

- generating predefined variants
- e.g. model-driven development

■ Substitution

- replacing a variation point by a value
- e.g. parameterization
- e.g. code weaving

General Purpose Approaches

- Templating
- Decision Modeling
- Preprocessing
 - CPP, M4, sed, scripting languages
 - Frame-Technology
 - Model-Editor automation
- Configuration Management

Conditional Compilation: Example

```
#define T9_SUPPORTED  
#undef ATTACH_SUPPORTED
```

T9

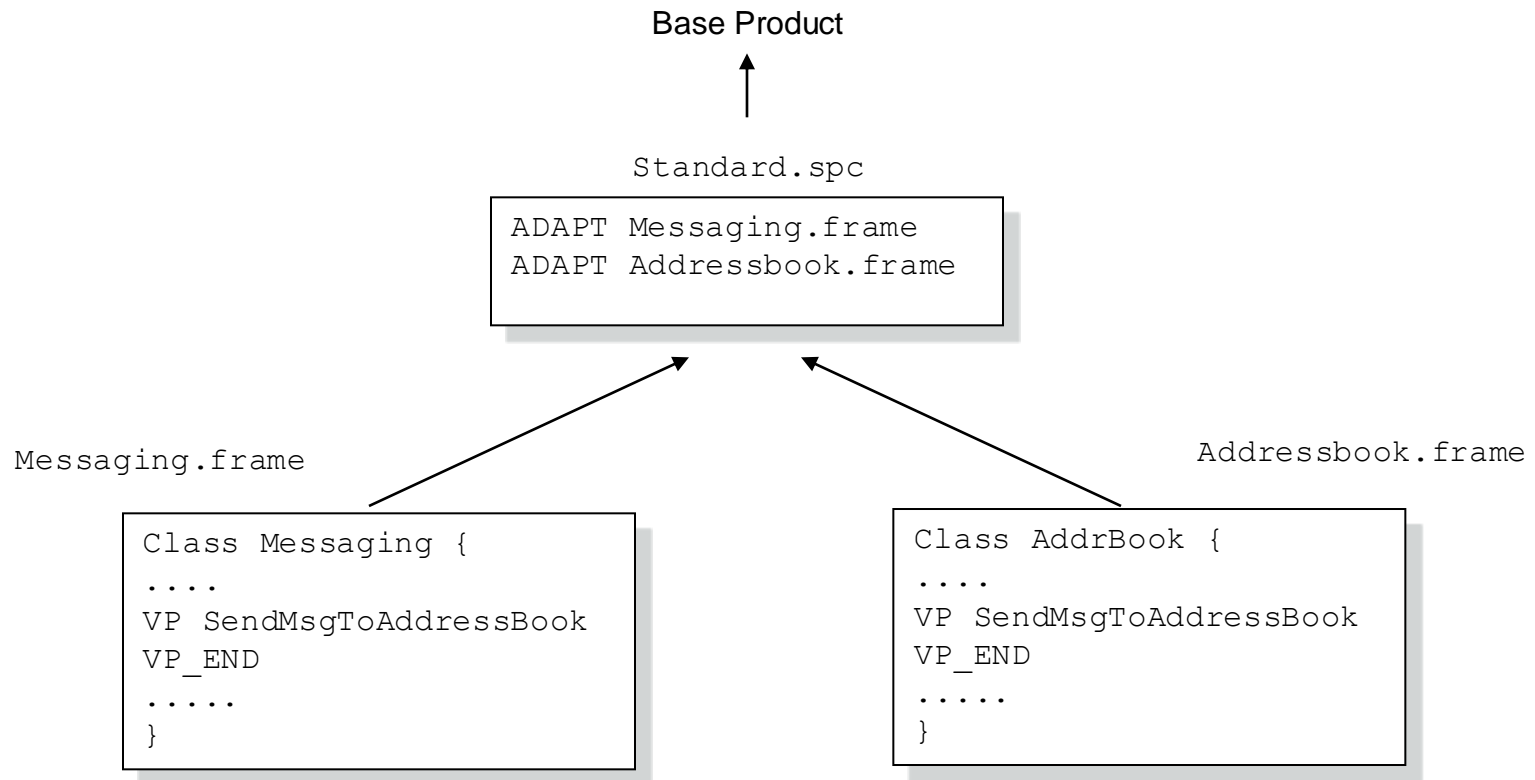
```
class Message {  
public:  
#ifdef T9_SUPPORTED  
    void checkWordList() {...}  
#endif  
  
#ifdef ATTACH_SUPPORTED  
    void enableAttachButton() {...}  
#endif  
};  
class MessageUI {  
public:  
    void edit(Message &msg) {  
#ifdef T9_SUPPORTED  
        if(t9Active) tr.checkWordList();  
#endif  
        // perform editing  
#ifdef ATTACH_SUPPORTED  
        tr.enableAttachButton();  
#endif  
    }  
};
```

Attachment

One
possible
product
Specification
T9 supported

```
class Message {  
public:  
    void checkWordList() {...}  
};  
class MessageUI {  
public:  
    void edit(Message &msg) {  
        if(t9Active)  
            msg.checkWordList();  
    }  
};
```

Frame Technology



**--- Product Line Infrastructure
Part III: Configuration Management ---**

How to manage variants?

Evolution

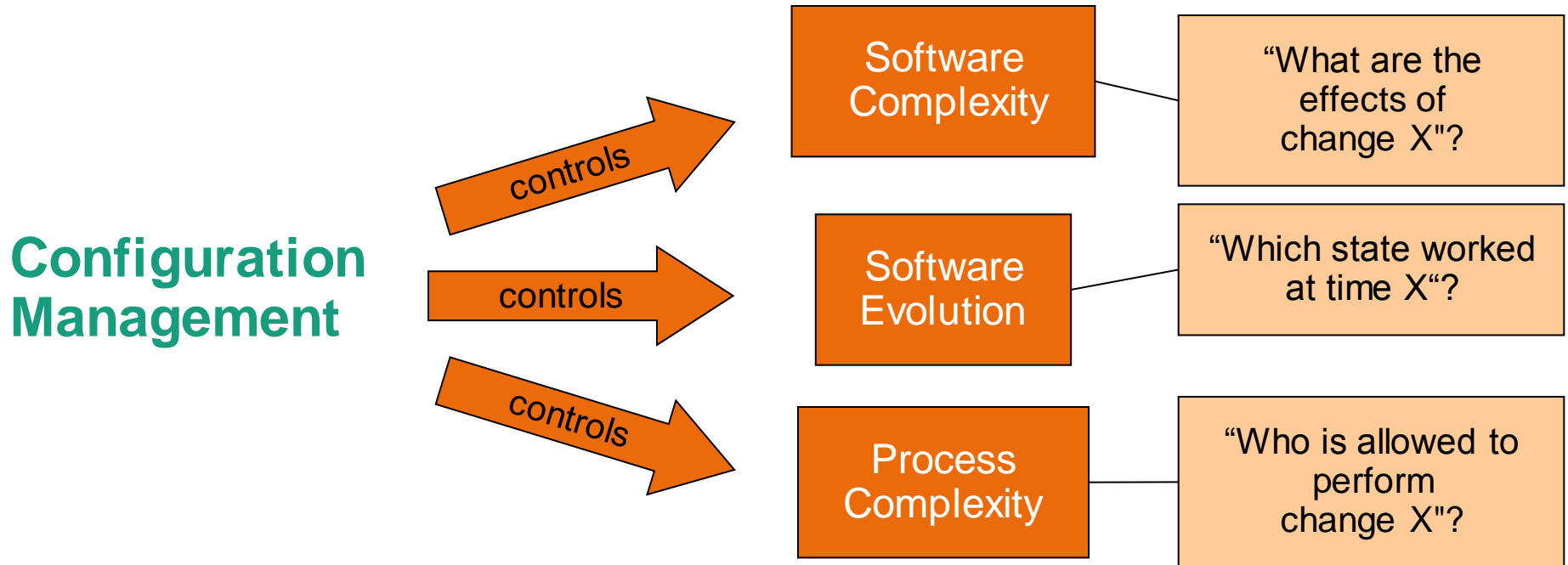
- Lehman's first law

- *"A program that is used in a real-world environment necessarily must change or become progressively less useful in that environment"*

- Software components continue to evolve

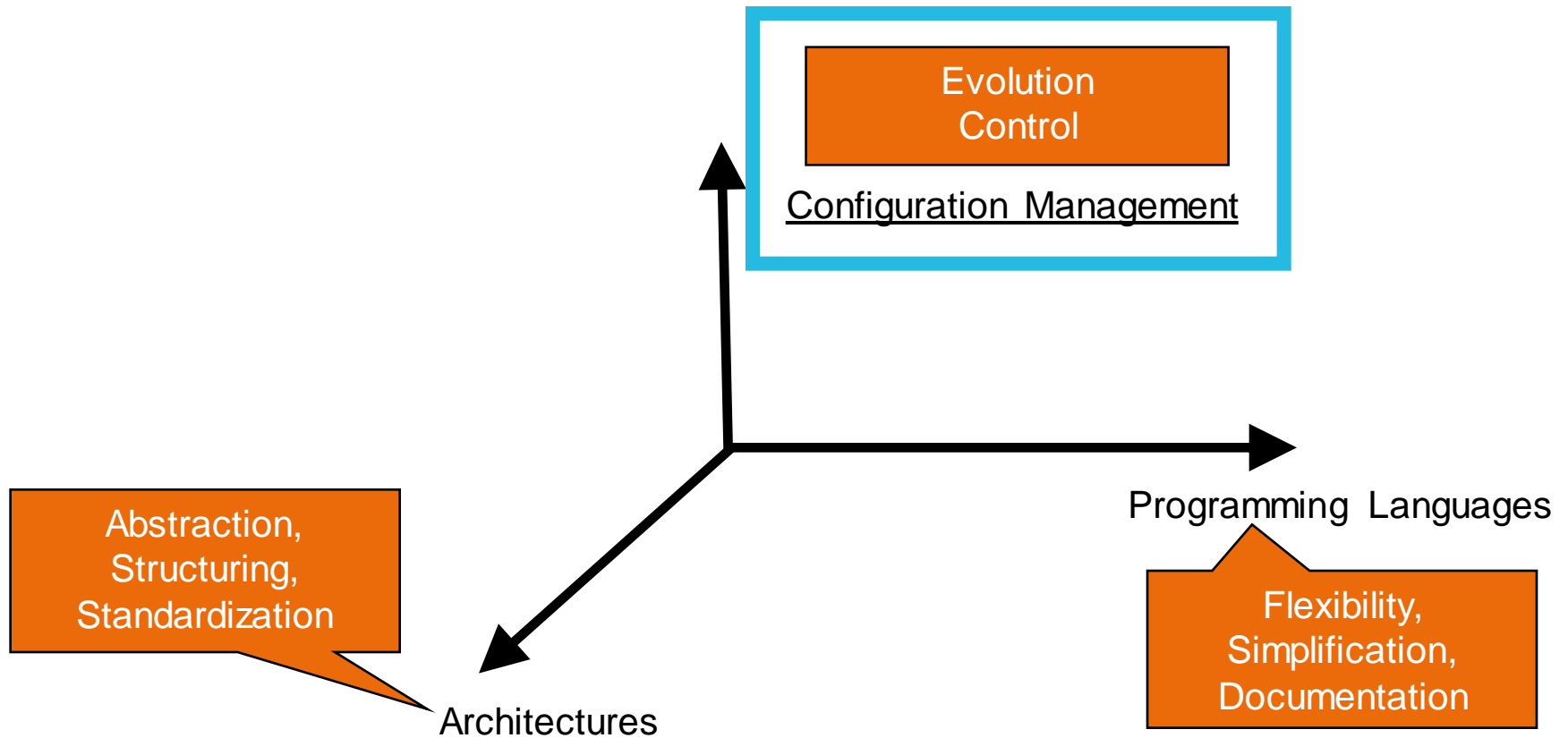
- Possibly independent of each other
- Possibly at different sites

Configuration Management == Controlling



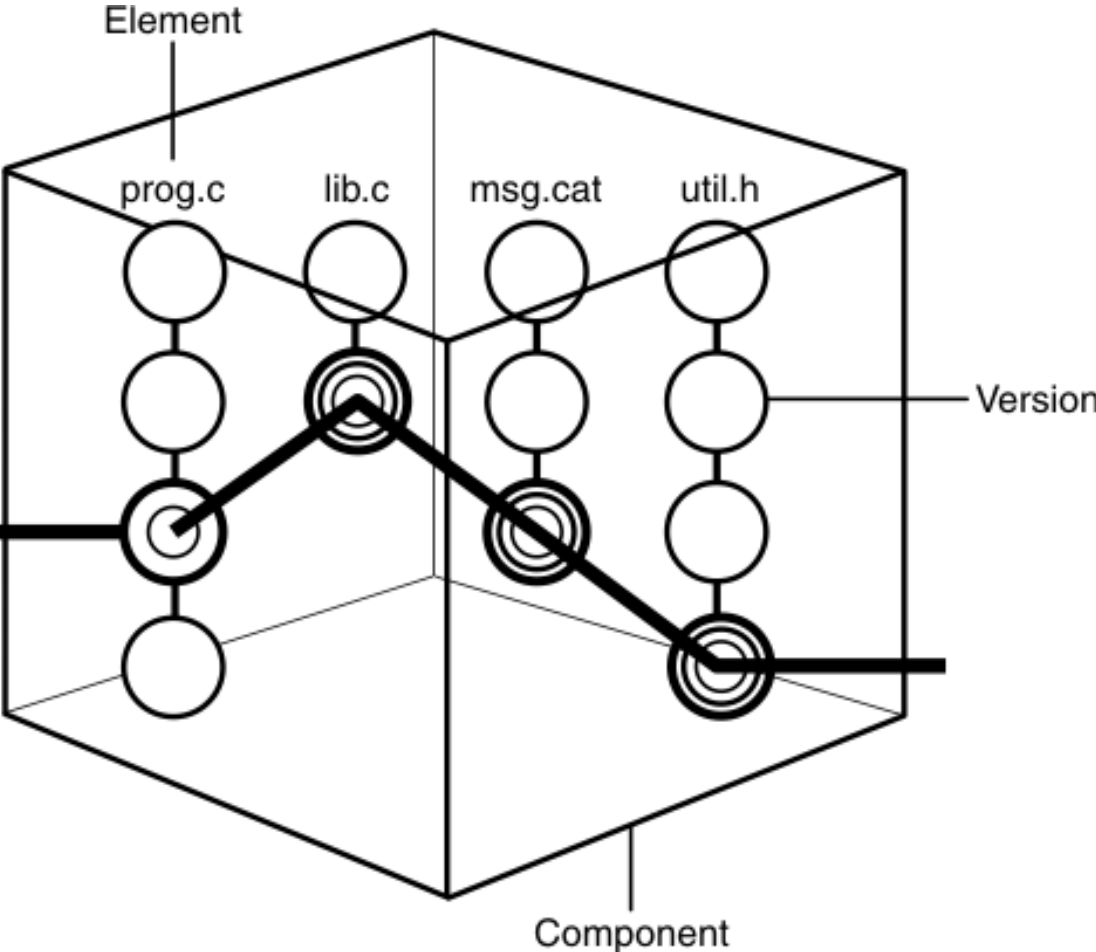
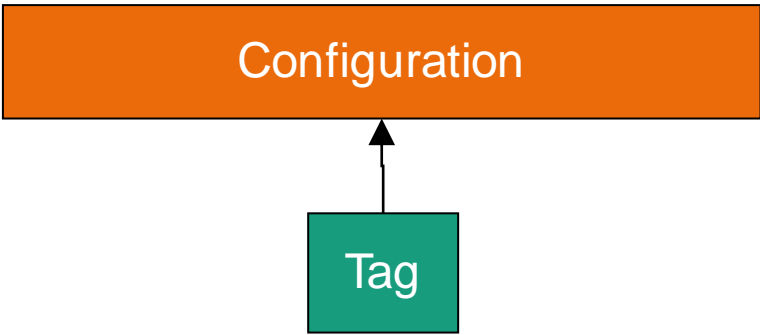
The goal is to improve/maintain a good level of maintainability of the system.

Paths to Maintainability



What is this **Configuration** we want to control??

A **Configuration** is a snapshot of the software being developed (with all the relevant artifacts) at a given point in time.



The configuration is represented as a **TAG** in the SCM versioning Tools.

Solution: Configuration Management (CM)

- Unique identifier
- Version number
- Descriptive name
- (Check sum)

■ Identification

- Identifies the units to be controlled

■ Controlling

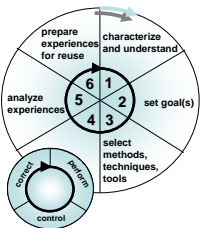
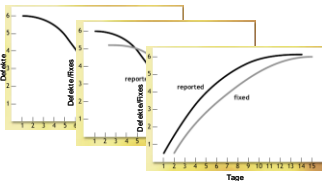
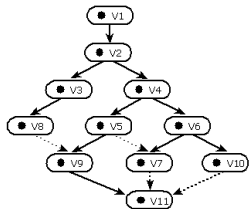
- Determines rules for the execution of changes

■ Accounting

- Packages information and statistics

■ Auditing

- Checks the fulfillment of requirements



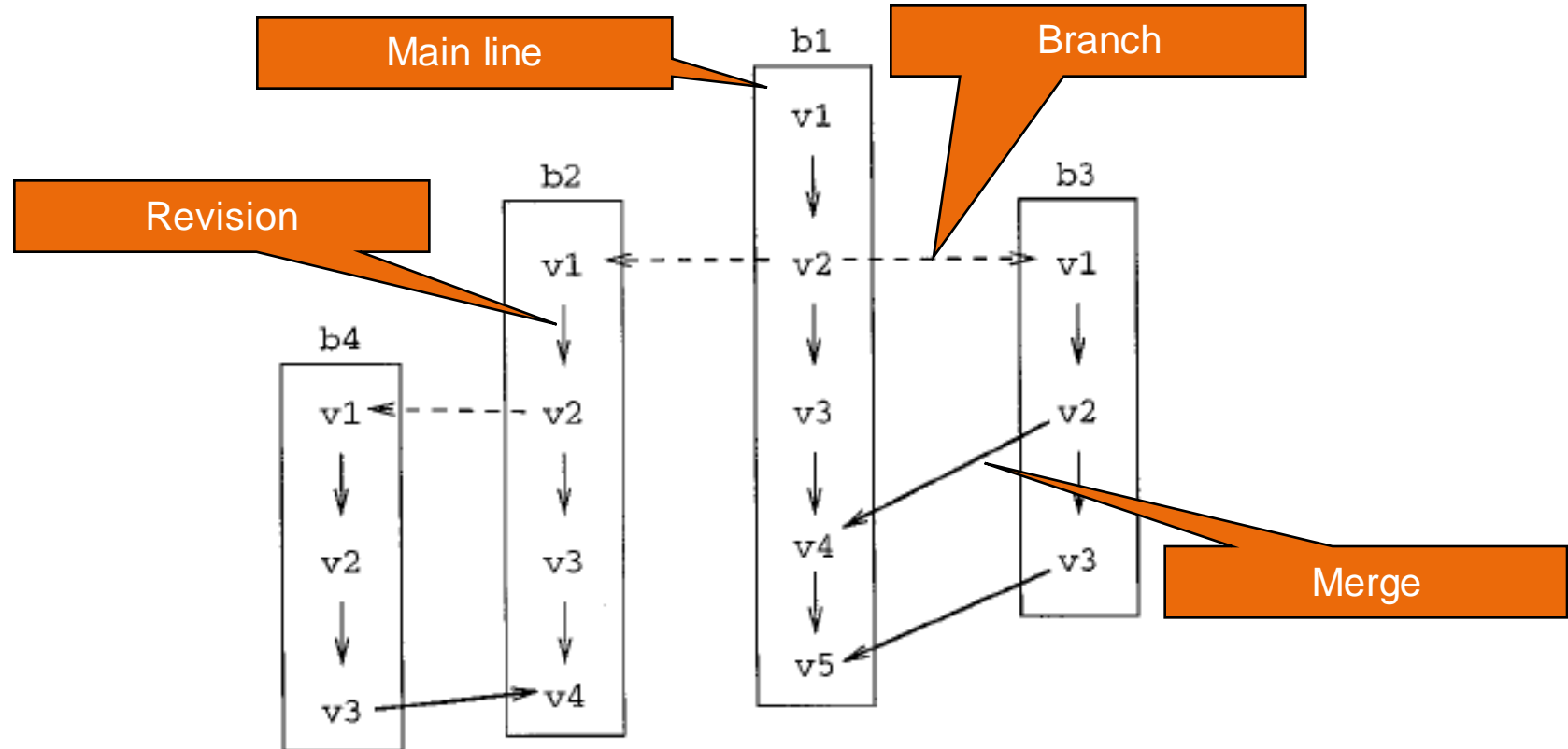
Configuration Identification

- Define the elements that need to be controlled
 - The controlled elements are called **Configuration Items (CIs)**
- Aspects of identification
 - Selection (what is a CI, what is not a CI)
 - Structure (CI hierarchies)
 - Naming (unique identification)
 - Characterization (further meta data)
 - Access (CI server location, access rules)

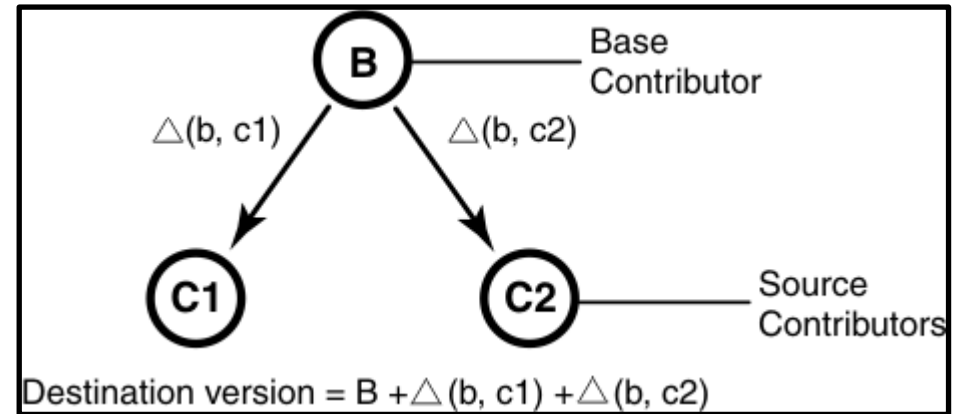
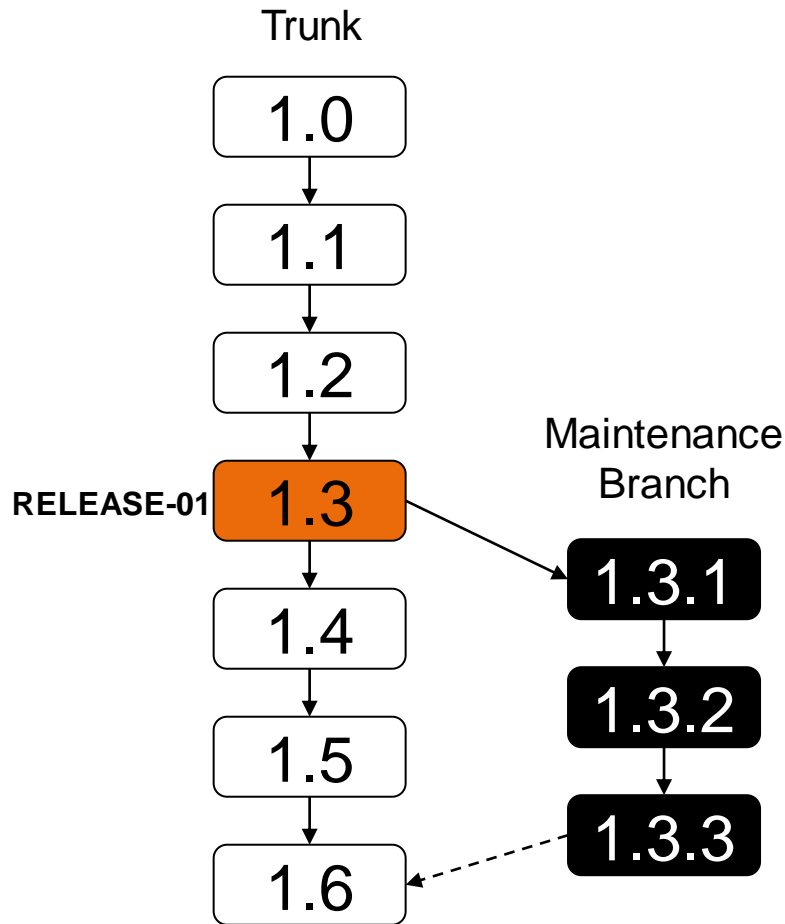
Configuration Control

- Main Component of configuration management
- Configuration Control areas
 - Version management
 - Change management
 - Build management
 - Release management

Version Management



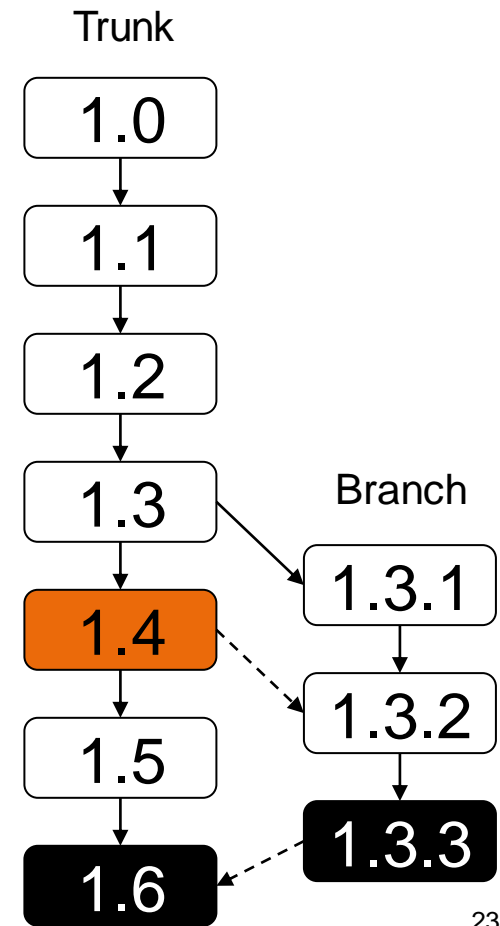
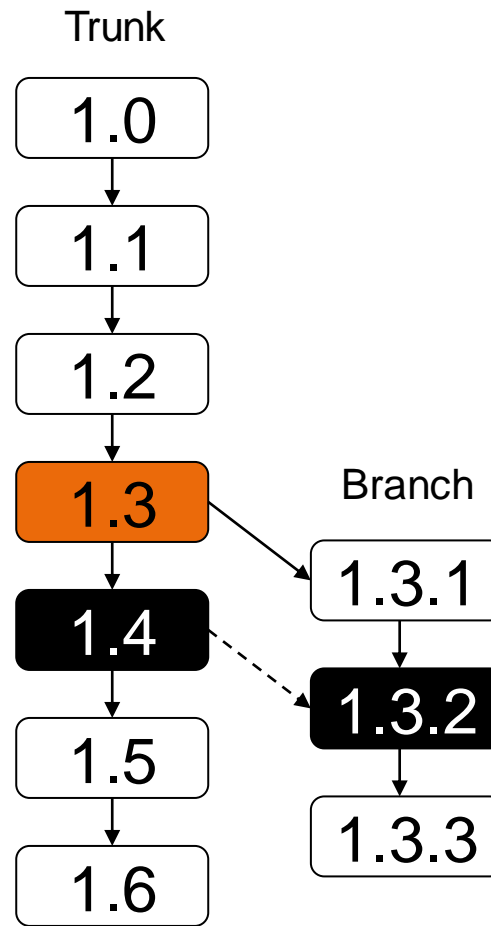
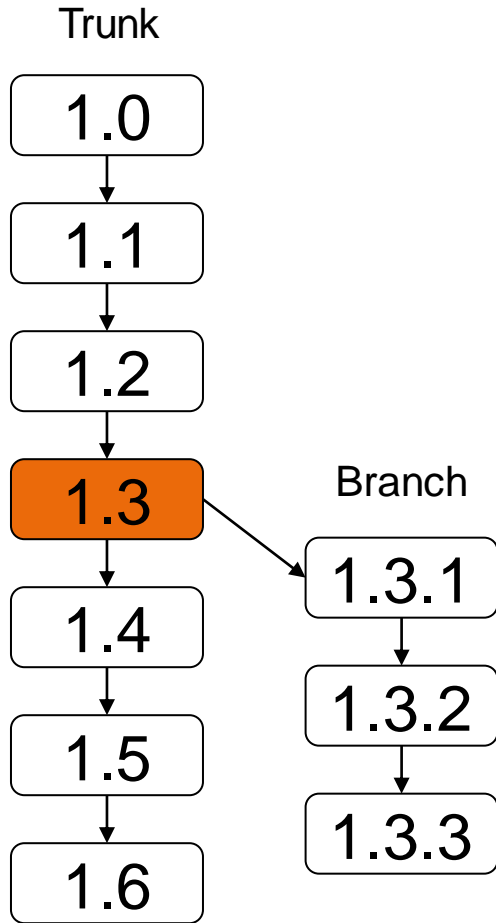
Branching and Merging Scenarios



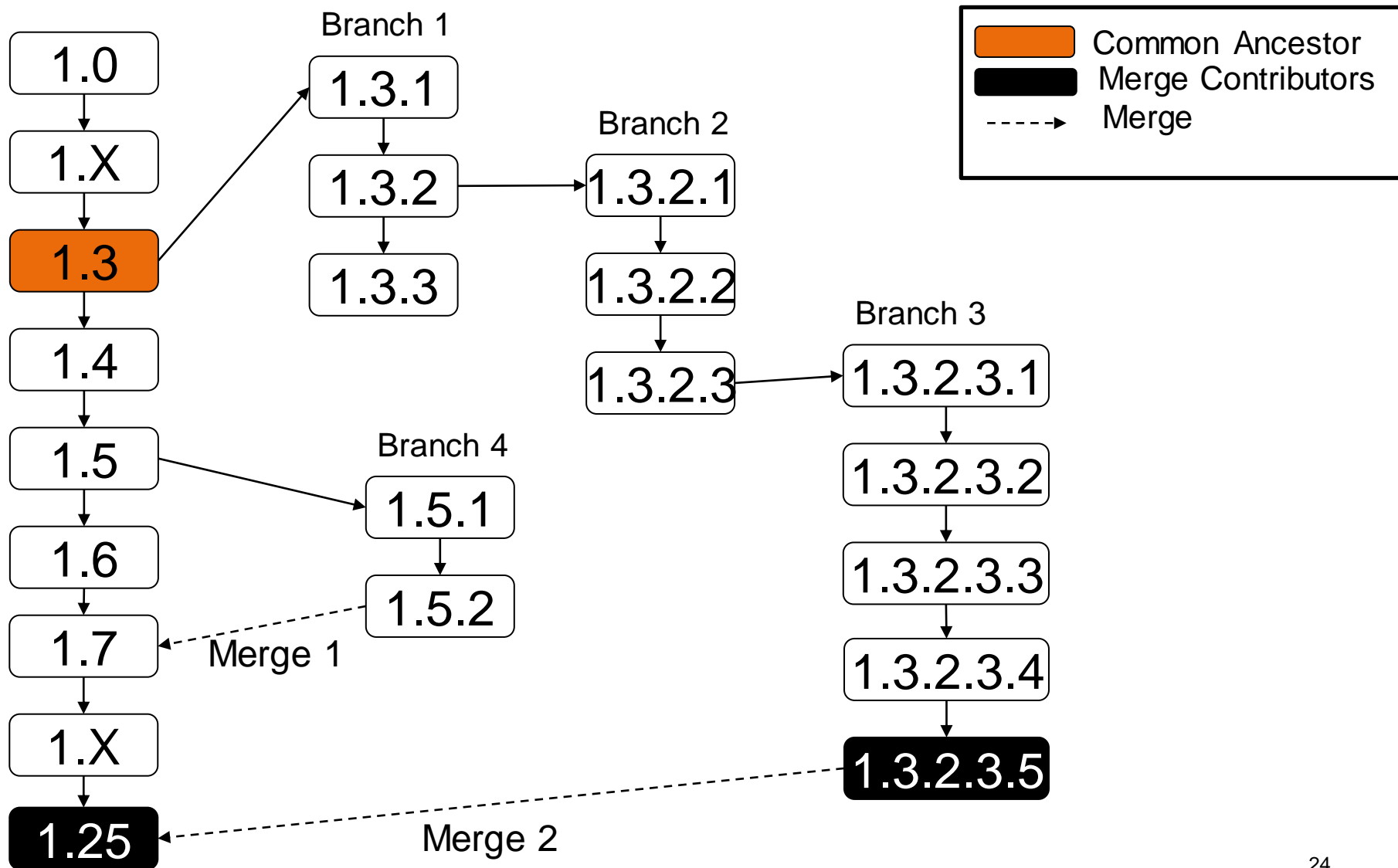
The more distant the contributors are from the common ancestor the more difficult the merge is (more changes to merge)

Ideally the distant must be kept small to facilitate the final merge

Ancestor Update



Exercise: Which Merge is more difficult?



Merge Early and Often

prog.c

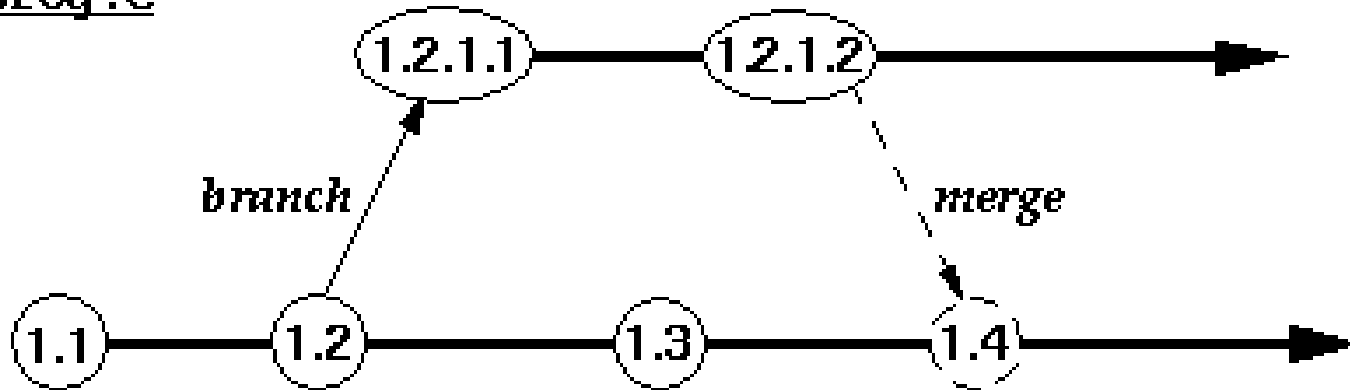


Figure 4: Merging back to the parent branch for file prog.c

- Branching policies must be established. Use it wisely!
- The **complexity** of a merge can range from simple to impossible
 - Merging a complex change to a heavily changed main line may not be possible without significant manual intervention

Change Management

- Change management makes the process of a change systematic
- Changes must be formally requested and approved

- Core concept 1:

Change Request (CR)

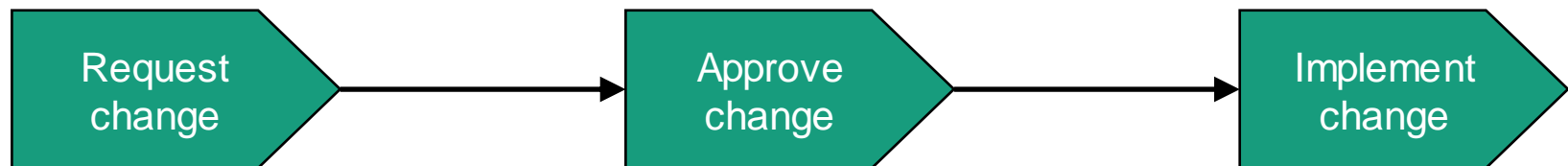
Submitted in a Bug tracking tool such as Bugzilla, Trac, Mantis, etc...

- Core Concept 2:

Change Control Board (CCB)

Responsible role to analyze and approve CR's

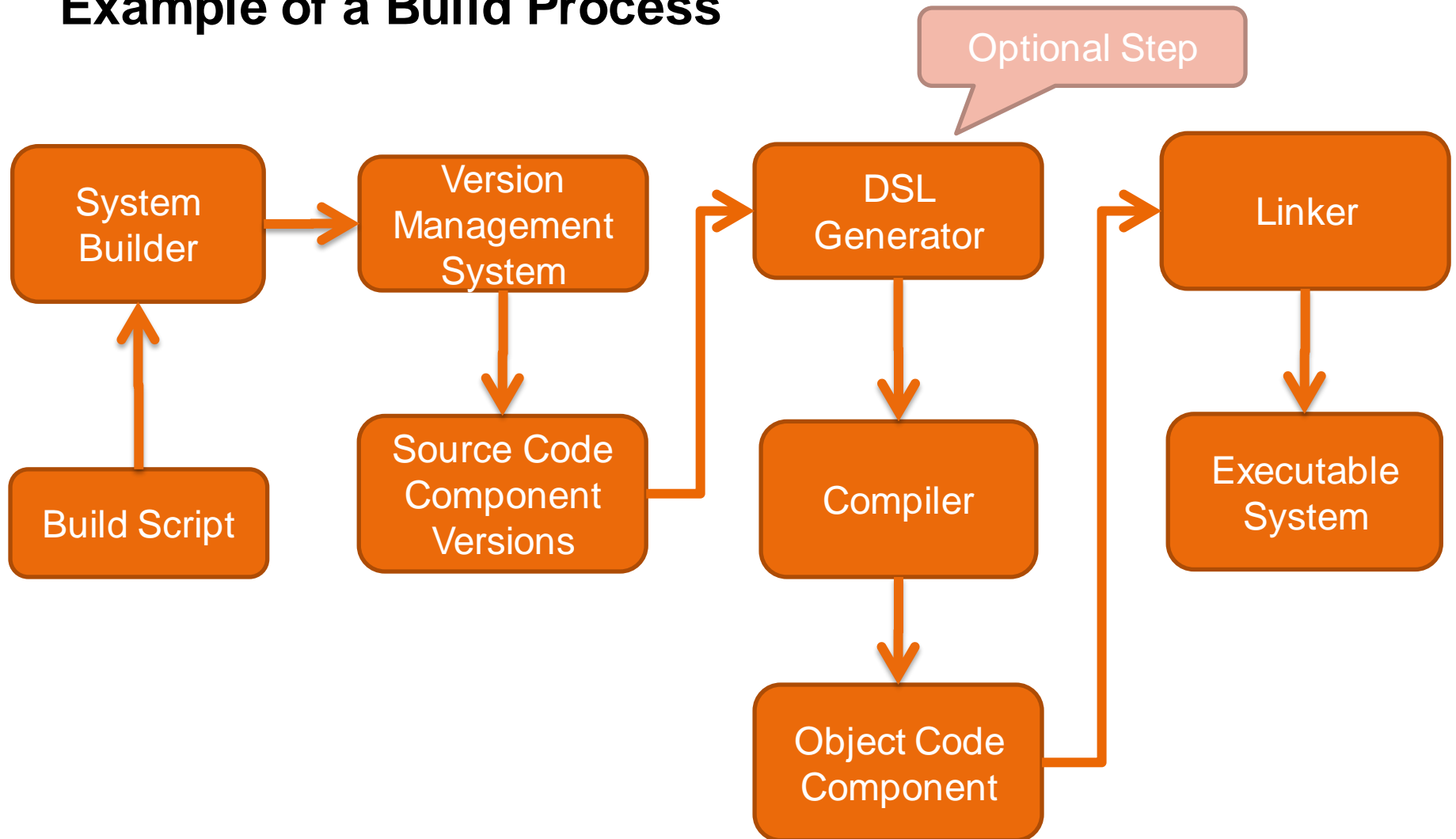
- A Simplified Change Management Process:



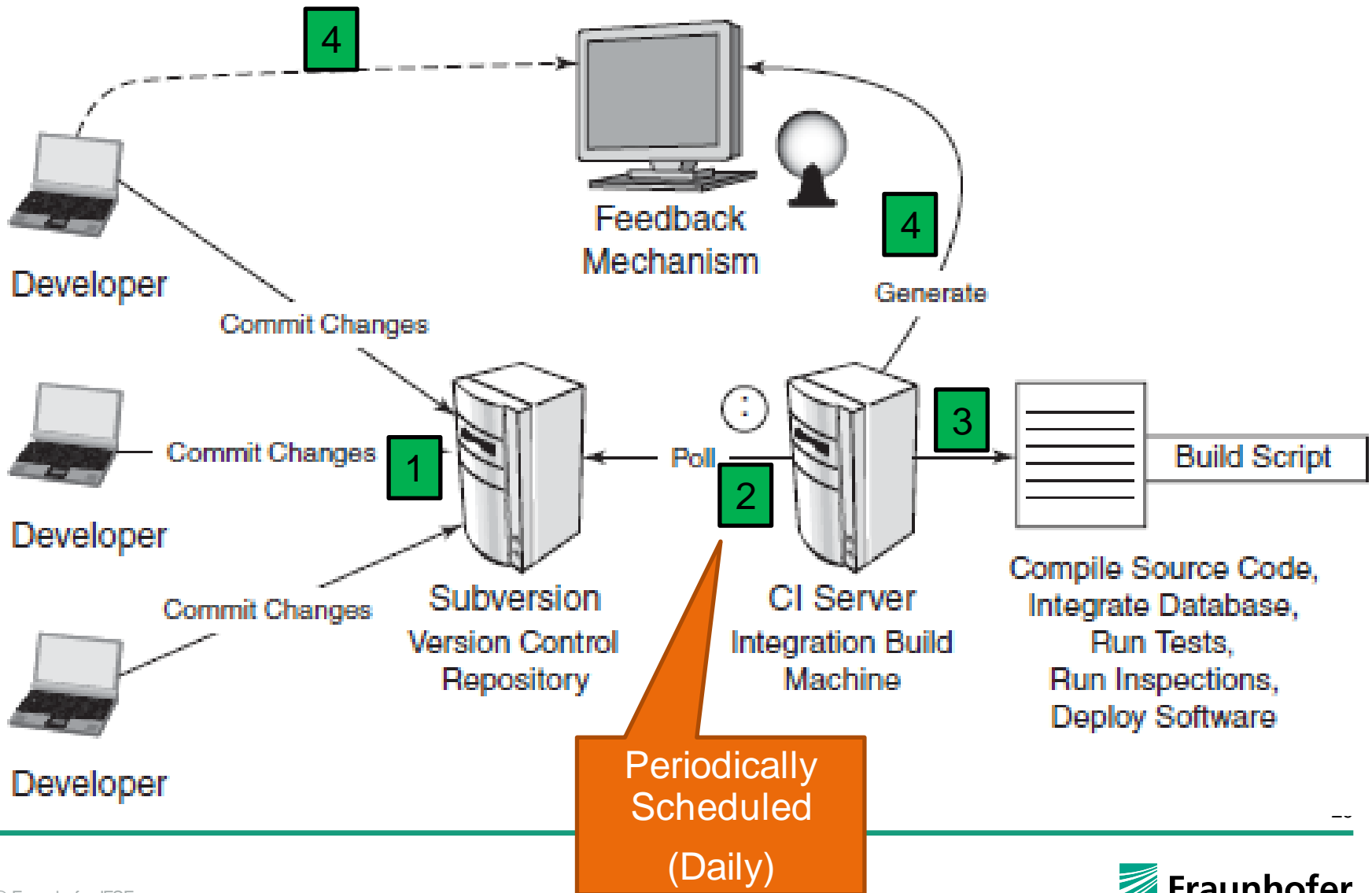
Construction (Build)

- The goal of construction is to create a *build* , i.e., an executable version of the system or of a component
- Each build process must be reproducible
- Build processes are automated
 - With the help of build scripts and tools (e.g., make, ant, etc...)
 - Build scripts are important configuration elements

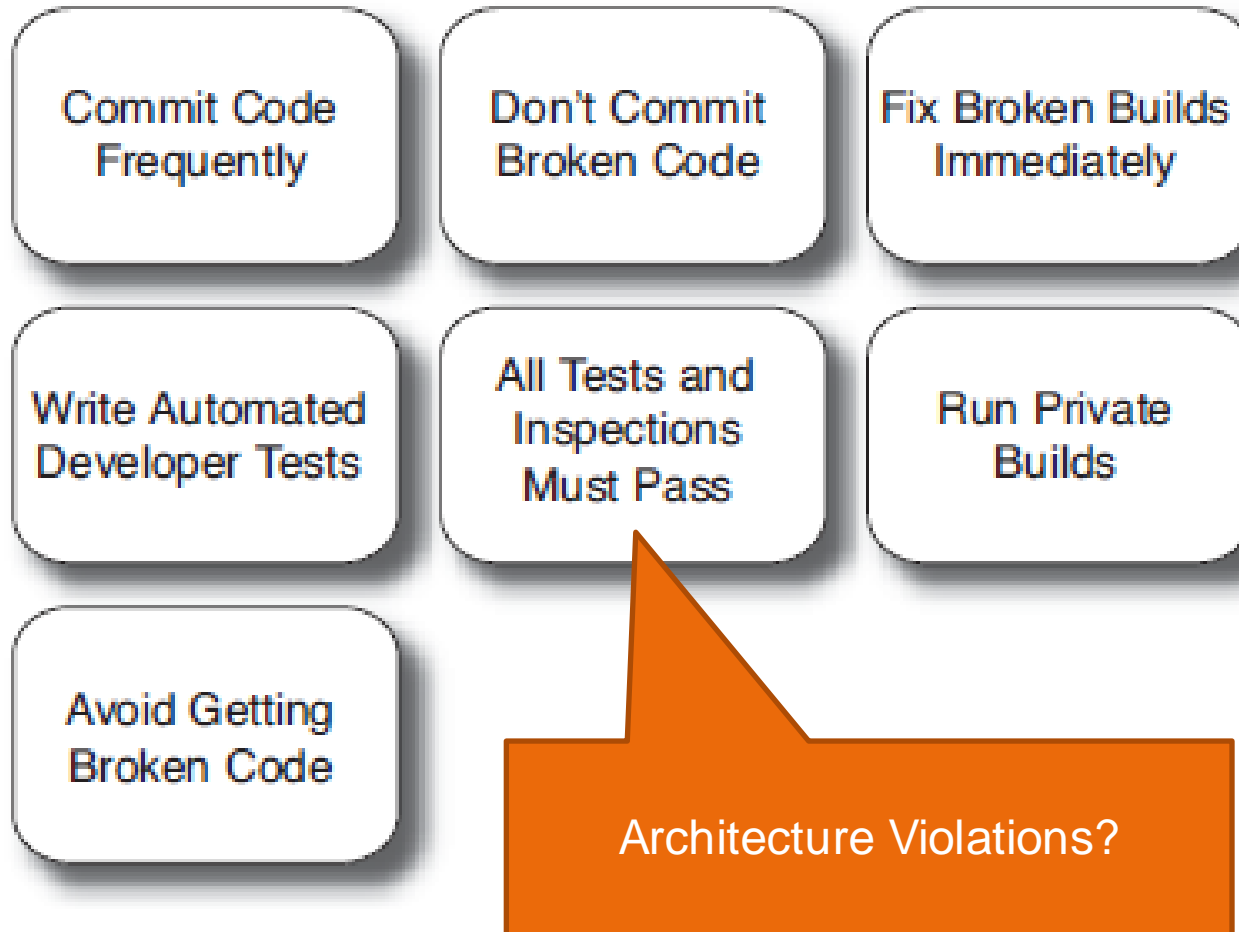
Example of a Build Process



Trend: Continuous Integration (1/2)

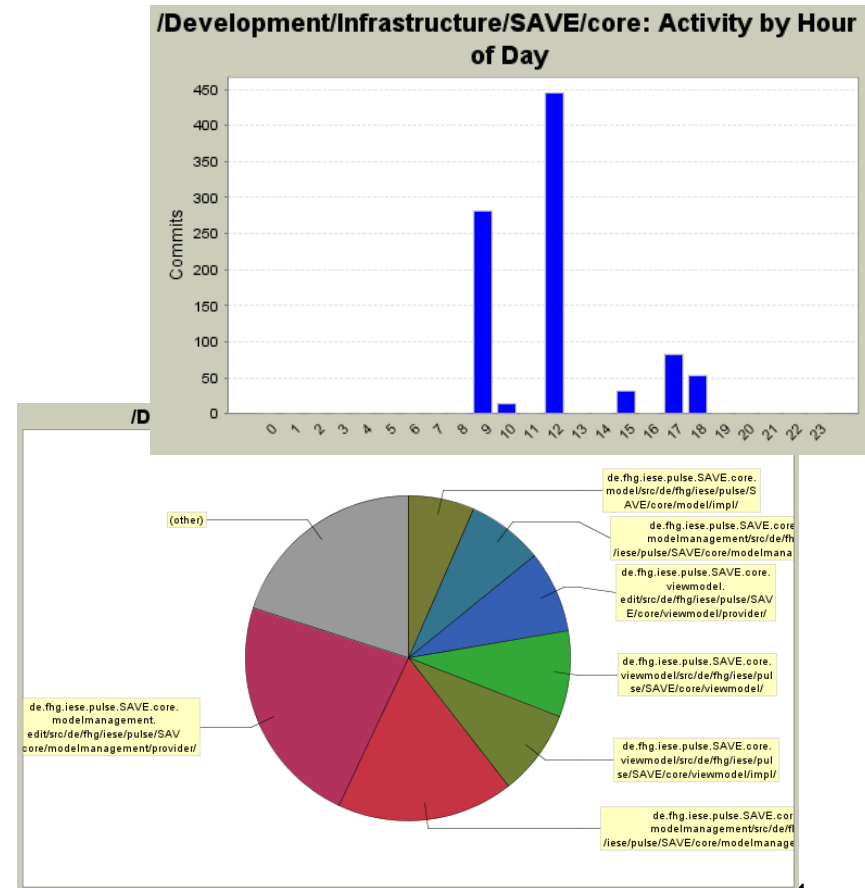


Trend: Continuous Integration (2/2)



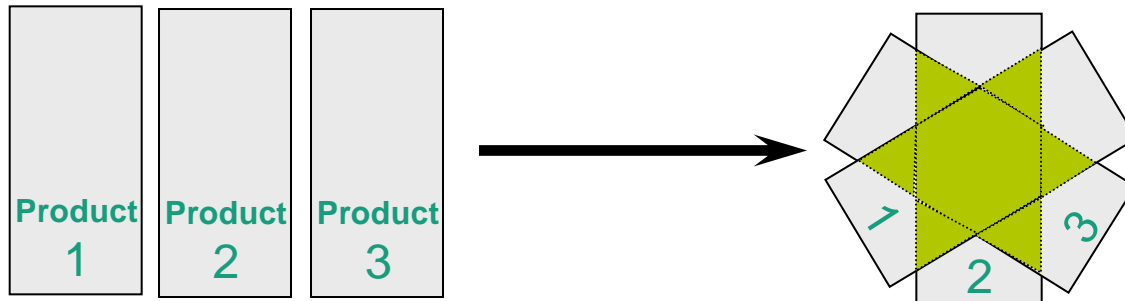
Accounting

- The goal of accounting is to collect and provide information from the CM system
- Examples of information
 - Status of a CE
 - Status of a change request
 - Status of all open changes
 - Information about the assignment of CEs
 - How many check-outs this month?



**Software Configuration
Mangement
&
Software Product Lines**

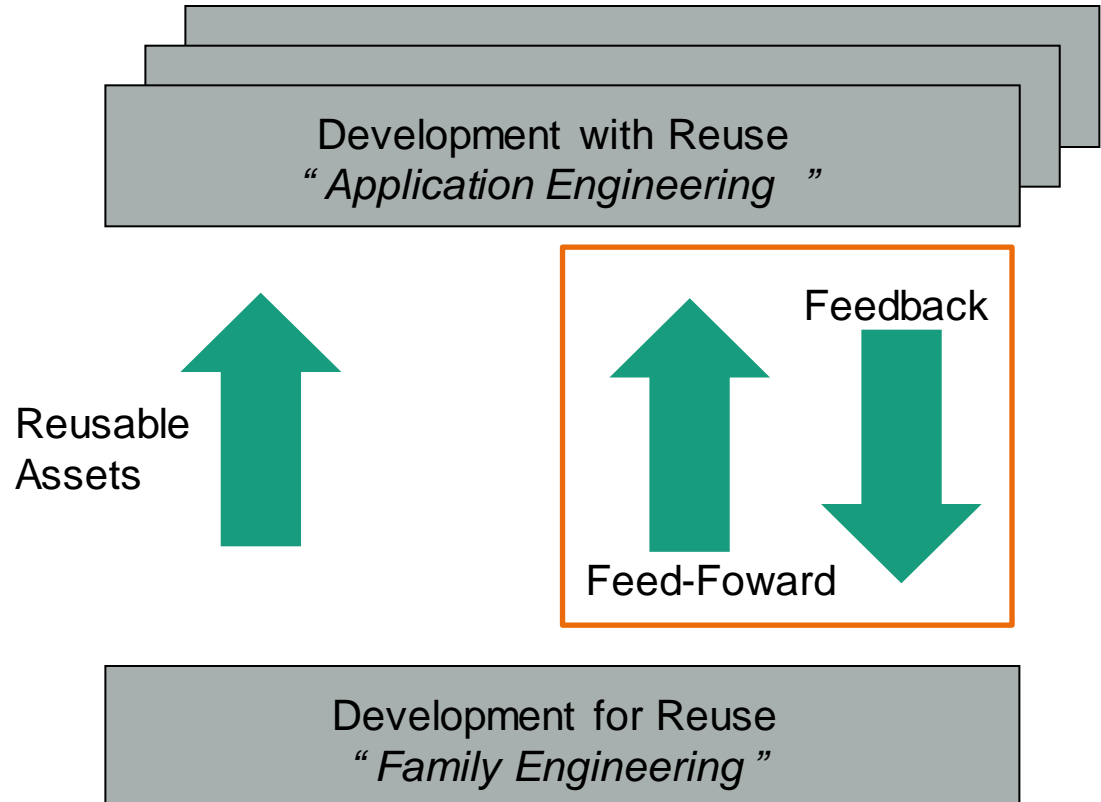
Variant-Rich Systems



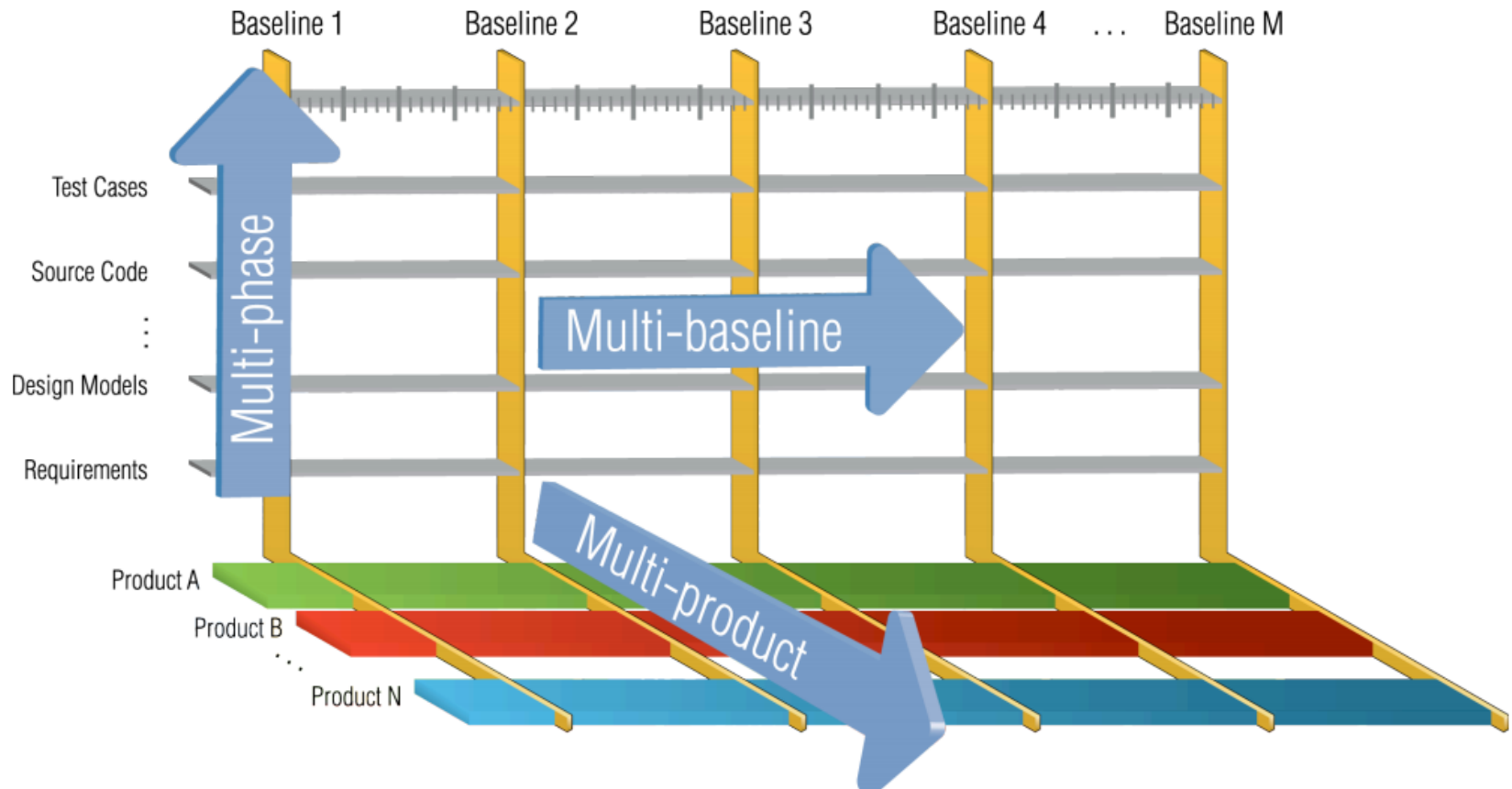
- Systematic reuse
- Exploitation of **commonalities**
- Determination of **variabilities**

Software Product Lines

Keyword:
Product Lines



Management of Product Line Variation



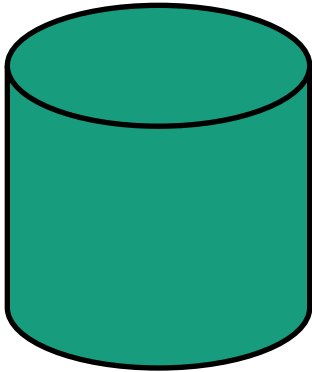
Variation in Space x Time x Lifecycle

[source: Biglever: The Systems and Software Product Line Lifecycle Framework]

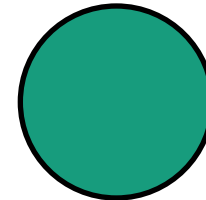
35

Build Management: Standard in the case of Variant Richness

Variant-Rich System



Build Management

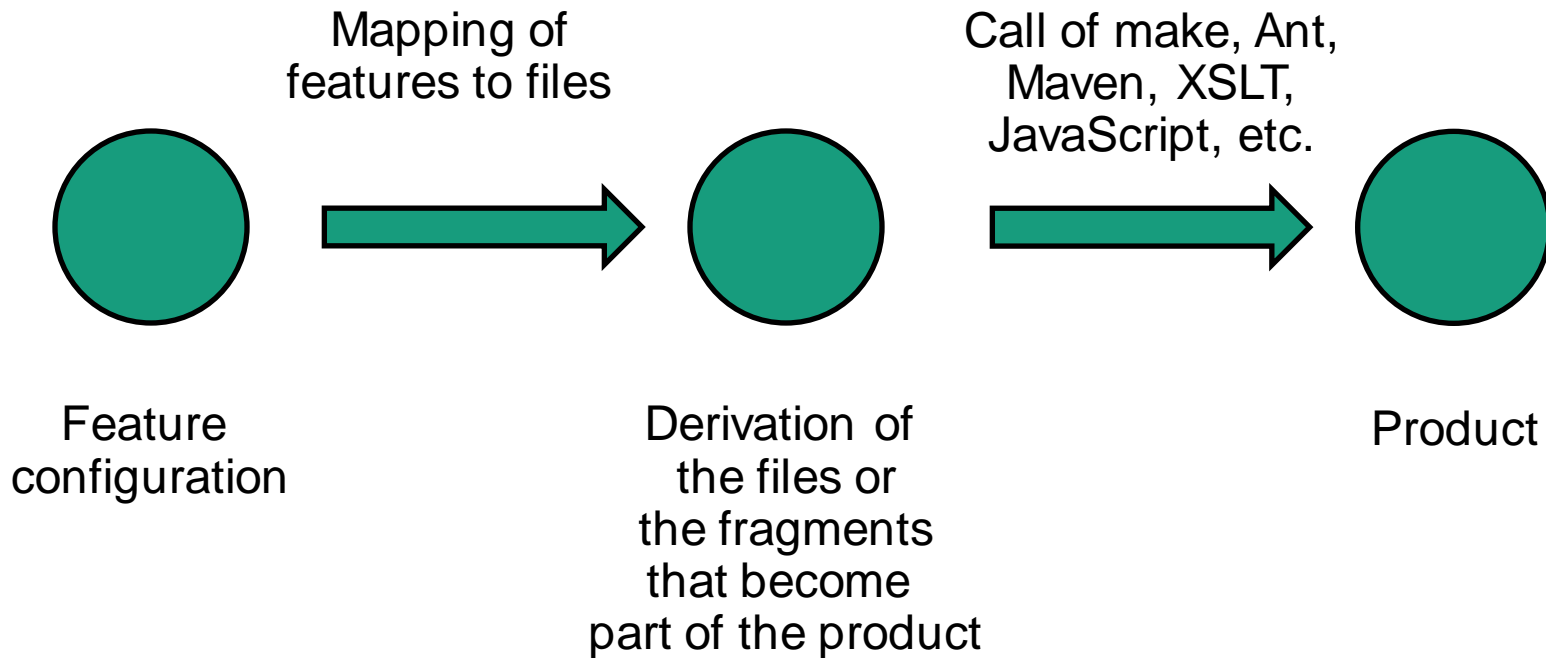


Product

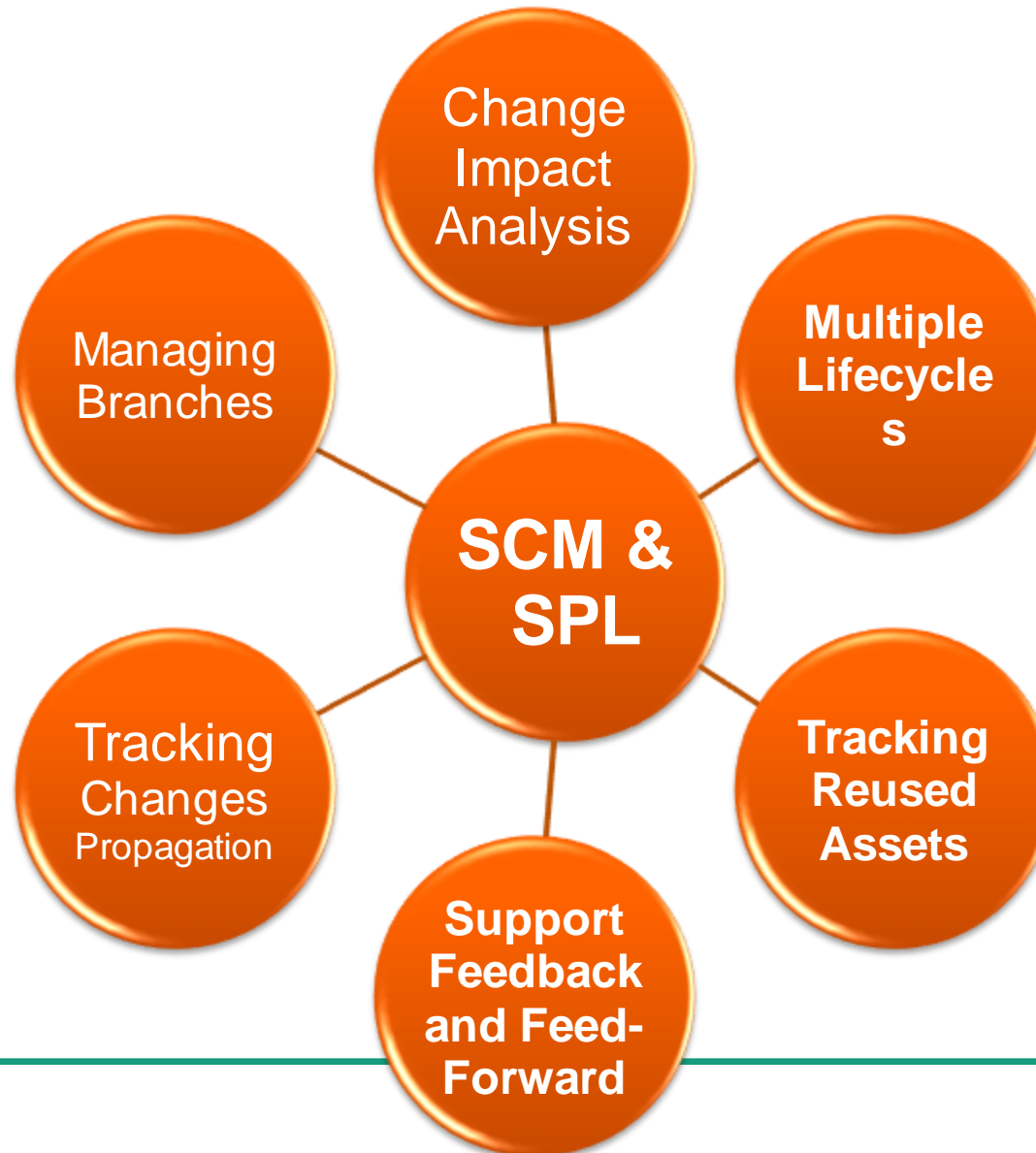
Common + variant parts
are clearly defined

→ Architecture is fundamental here

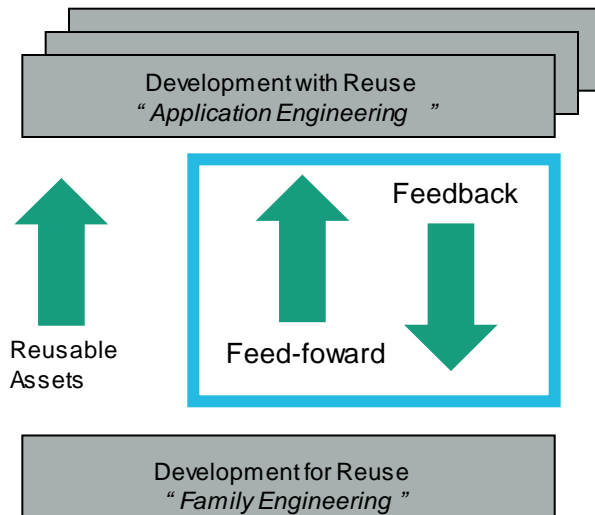
Construction of the Product after Configuration



SPL and SCM Challenges



Software Product Lines Erosion

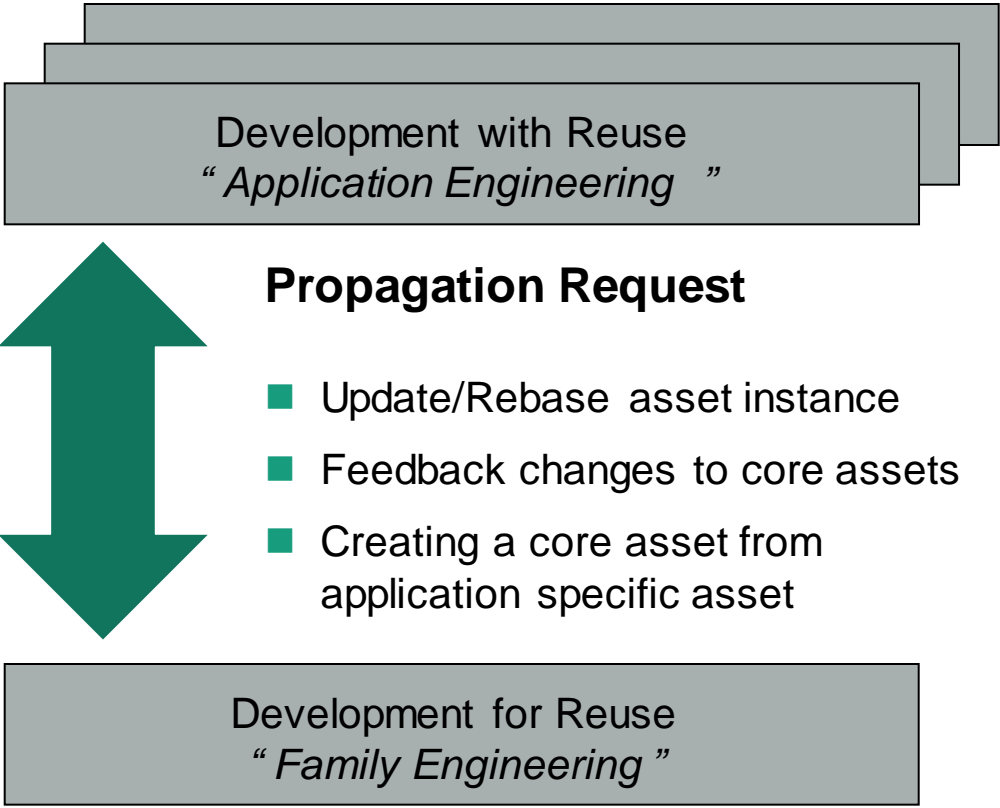


- Erosion is a problem that should be avoided in product lines
- It refers to a situation where reusable artifacts are ultimately not reused
- One reason for this is the lack of feedback or feed-foward from Application to Family Engineering and vice-versa

Feedback and Feedforward Changes

■ Core Concept:

Propagation Request (PR)



Development with Reuse
“Application Engineering”

Propagation Request

- Update/Rebase asset instance
- Feedback changes to core assets
- Creating a core asset from application specific asset

Development for Reuse
“Family Engineering”

Benefits of PR's

Changes Propagation are...

- Tracked individually
- Analyzed individually
- Analyzed WHEN it is time
- Maybe Rejected after evaluation
- Maybe Approved after evaluation
- Implemented according to release plans

Product Line

- Each product has independent configuration management after it is instantiation from the core assets
- 100 product instances == 100 independent divergent product evolutions
- A separate engineering activity is required to refactor changes made to a product instance back into the core assets and other products (PR)

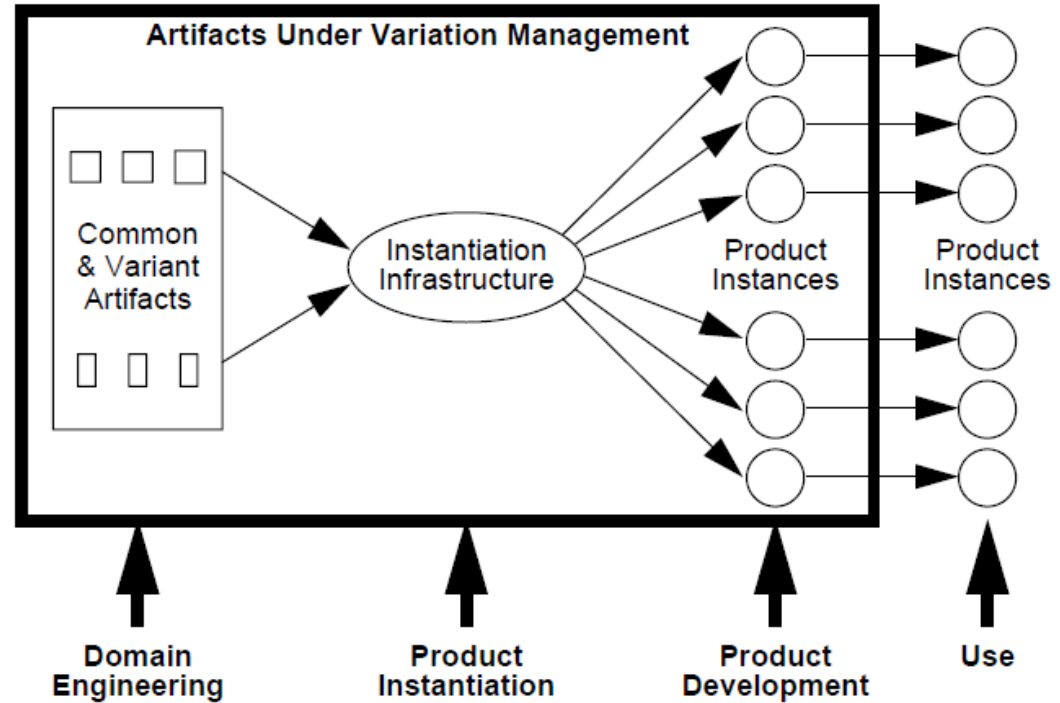


Figure 1 Variation Management of a Product Line

Production Line

- Individual products should be treated as transient outputs of the production line that can be discarded and re-instantiated as needed
- Reduces the $n+1$ dimensional configuration management problem to the well known 1 dimensional problem

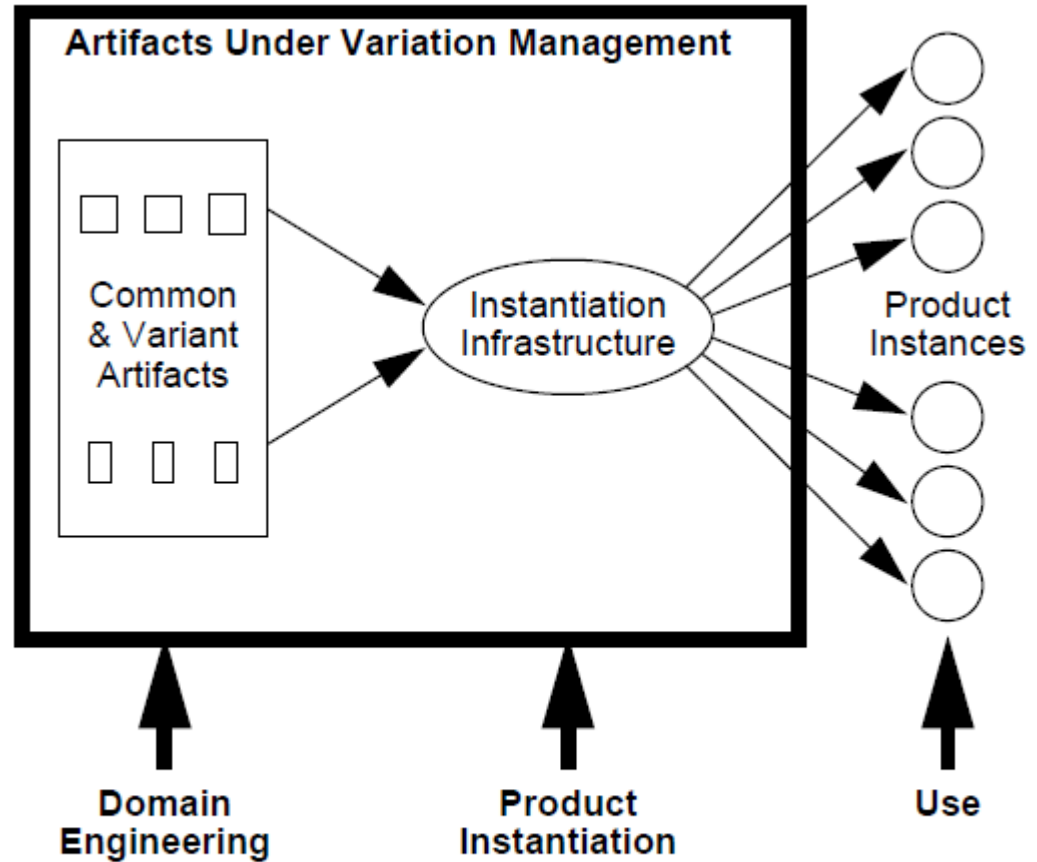
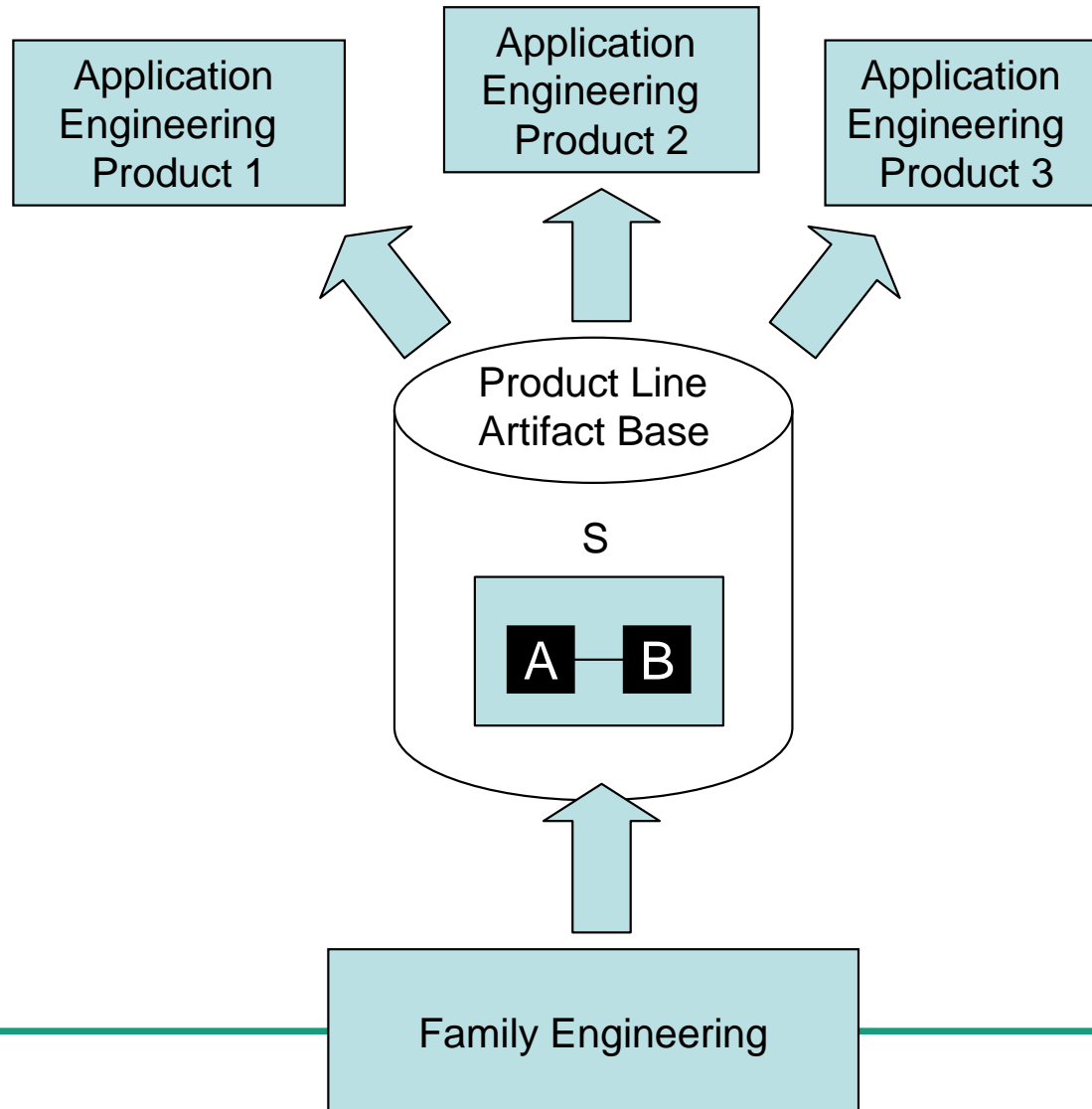


Figure 2 Variation Management of Production Line

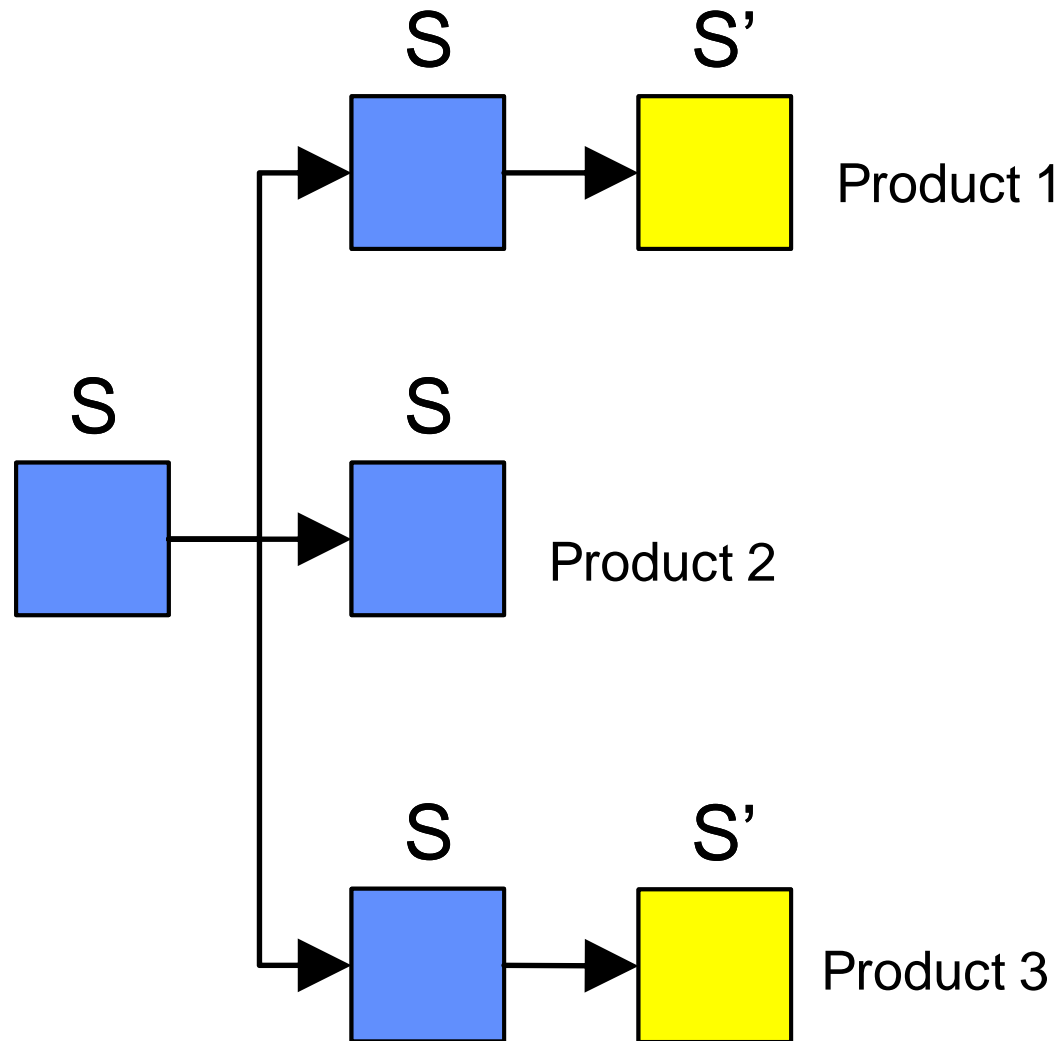
Tracking changes - How do we know...

- ... which products will be affected after a core asset change?
- ... which product instances changed a certain core asset in the application level?
- ... which products reproduces a certain bug after we find the bug in one product instance?

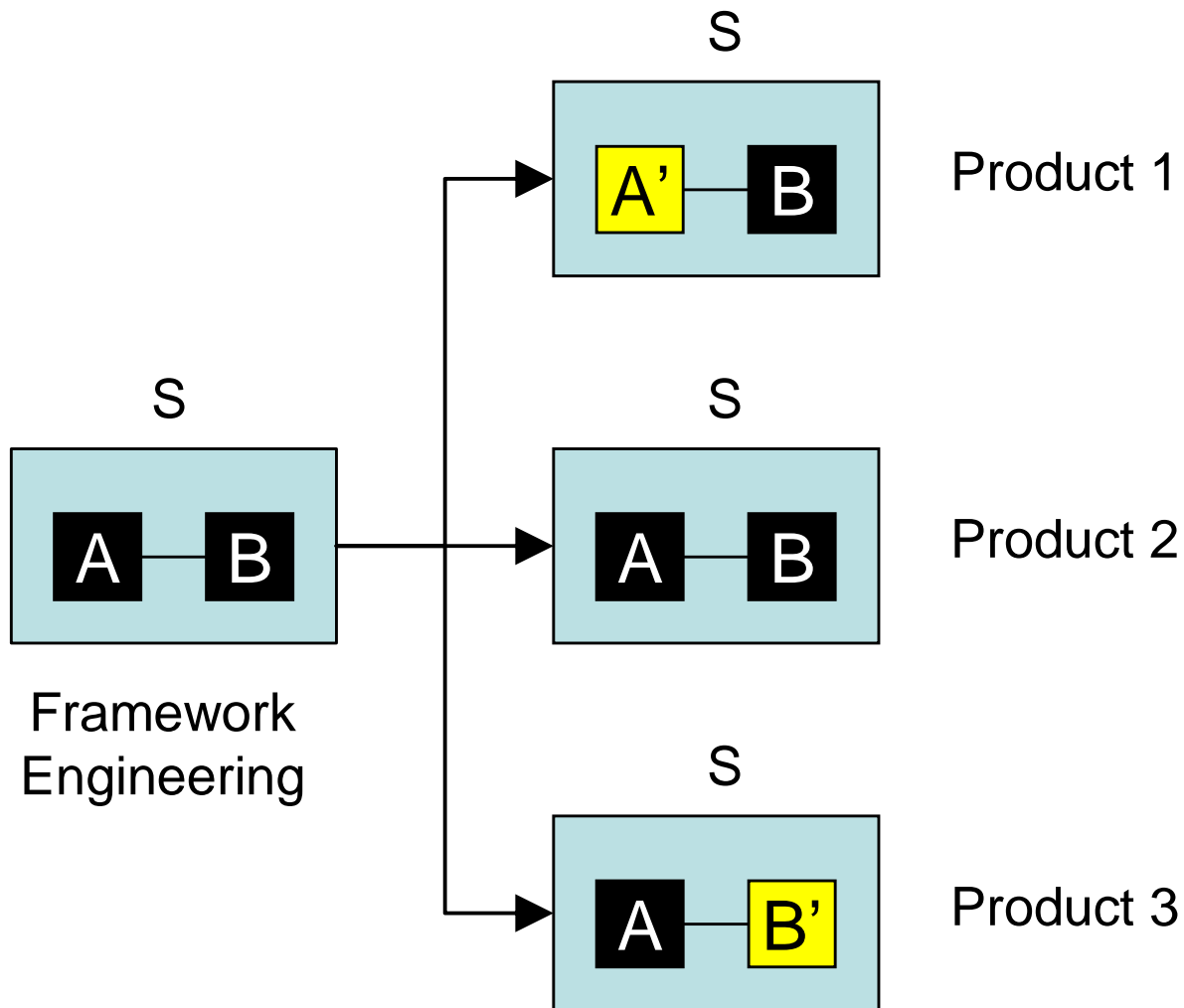
Illustrative Example: Tracking changes in products



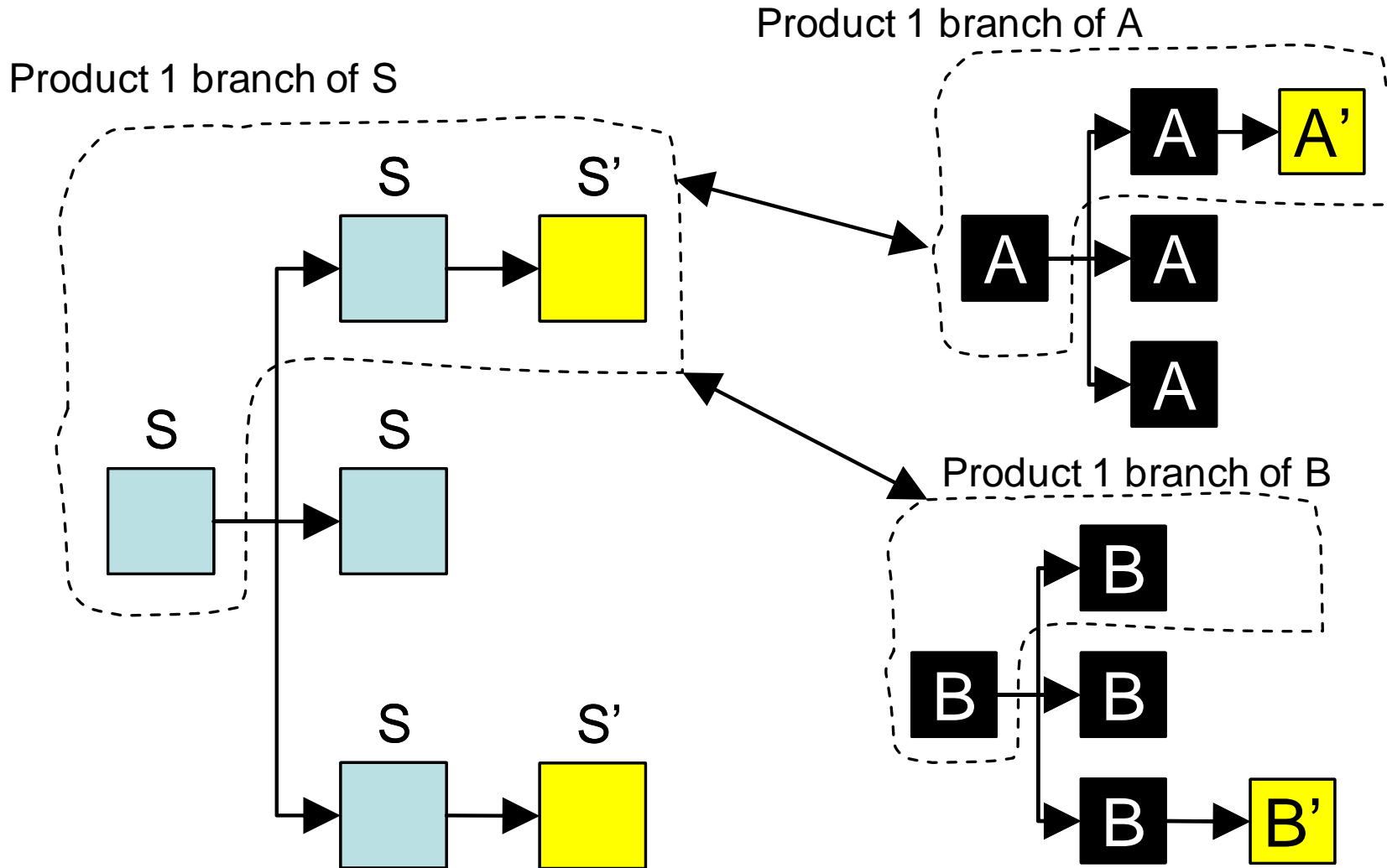
Illustrative Example: Sample Instantiations (black box)



Illustrative Example: Sample Instantiations (white box)



Illustrative Example: Traceability given through CM



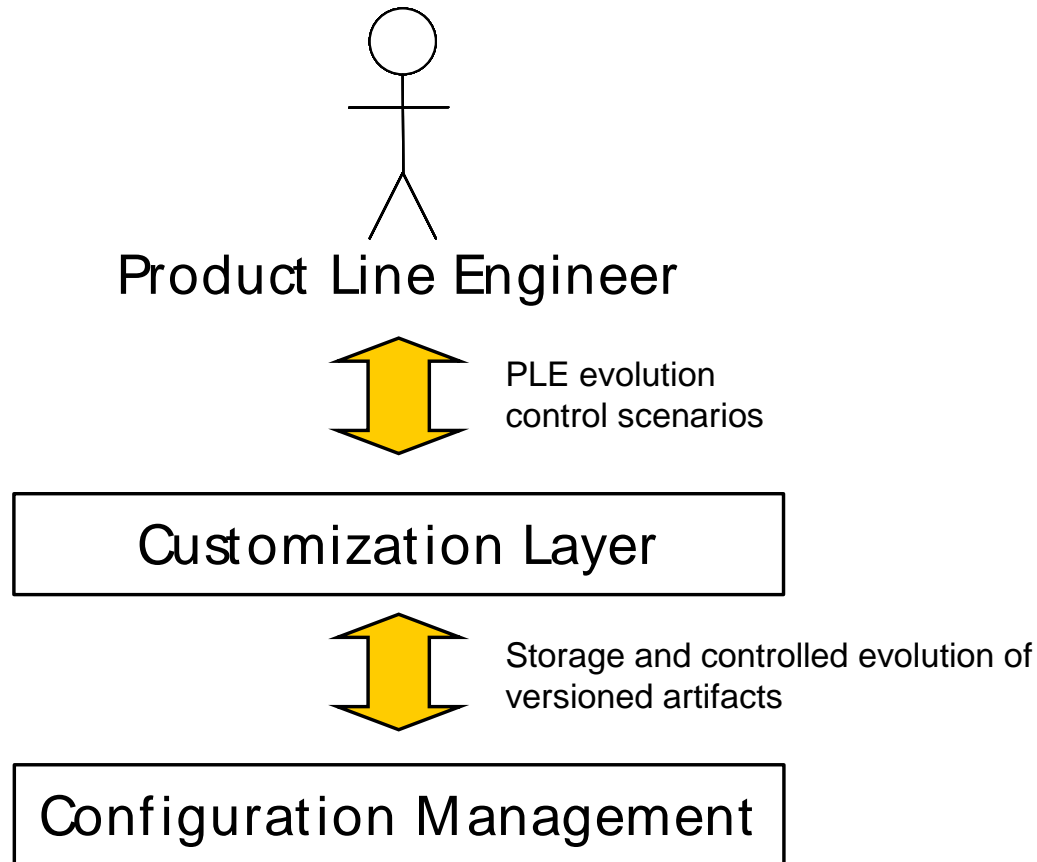
Illustrative Example: Complexity of CM

1 Activity requires 5 operations

Find out if something has changed in the instances of S.

1. Open the version graph of S
2. Identify the product branches (there may be many other temporary branches next to the product branches)
3. For each product branch look for new S versions since the last synchronization between family and Application Engineering
4. For each new version of S query the configuration management system for the changes made in that version
5. Filter out product-specific changes and identify changes that may affect S

Solution: Layer on top of configuration management



Encapsulating Configuration Management



Framework Engineer



Application Engineer

add-core-asset("Collection")

- add directory in repository
- mark it with special tag
- add all directory entries
- mark them with special tag
- enable according permissions

release-core-asset("Collection")

- enable according permissions for making "Collection" available for reuse

show-instance-diff("Collection")

- check, is "Collection" a core asset?
- search history of "Collection" for branches marked with special tag
- traverse to branches, check history

instantiate-core-asset("Collection", "Product1")

- check, is "Collection" a core asset?
- do I have permissions?
- create branch of core asset elements
- mark branch with special tag

commit

- common commit

show-core-diff("Collection")

- check, have I branched off "Collection"?
- check history of core asset "Collection"
- is there a released change in the core asset?