



SAAB

Developing Large Systems In Small Steps

Seminar “INCOSE@LiU” 2010-09-01

Ulrik Pettersson

Saab Aerosystems

ulrik.pettersson@saabgroup.com

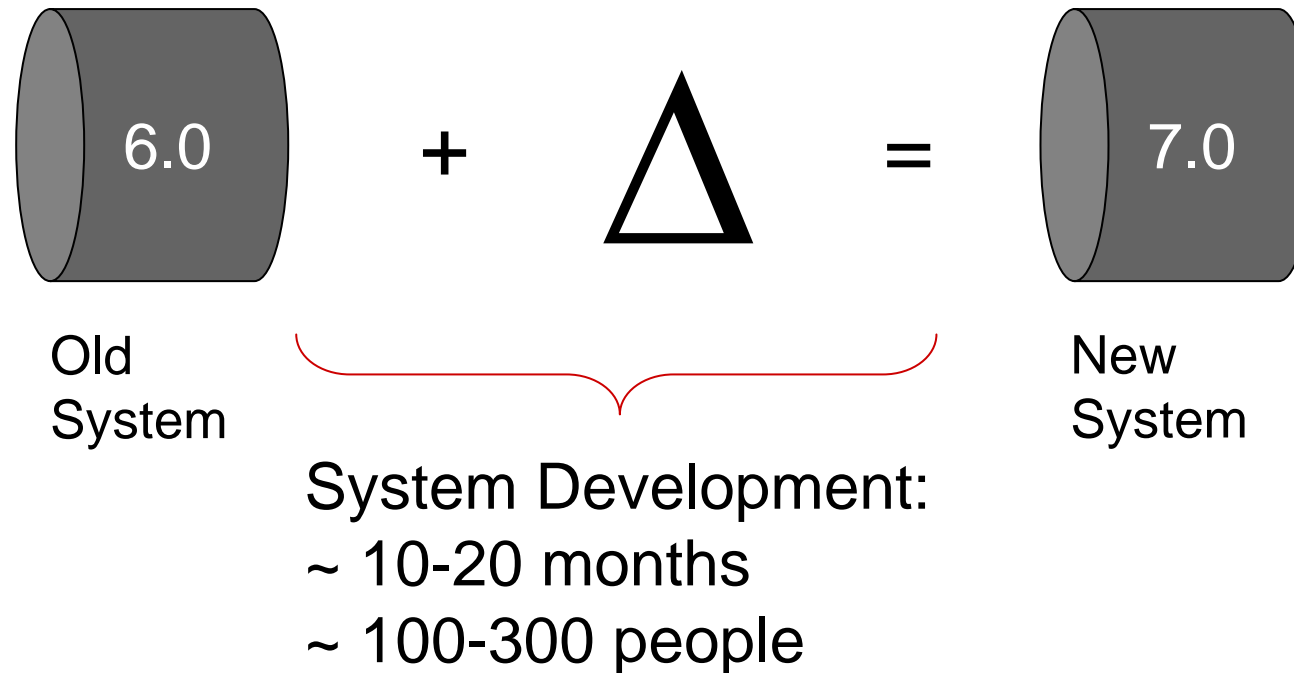
Content

- Why “Small Step Development”?
- Small Step Development for Large Systems – The Strategy
- *Some Alternative Approaches – How to form teams?*
- *Some Words About Documentation – Forward vs. Backward*

WHY “SMALL STEP DEVELOPMENT”?

The Context

Complex Large Scale Development



Development complexity mainly due to:

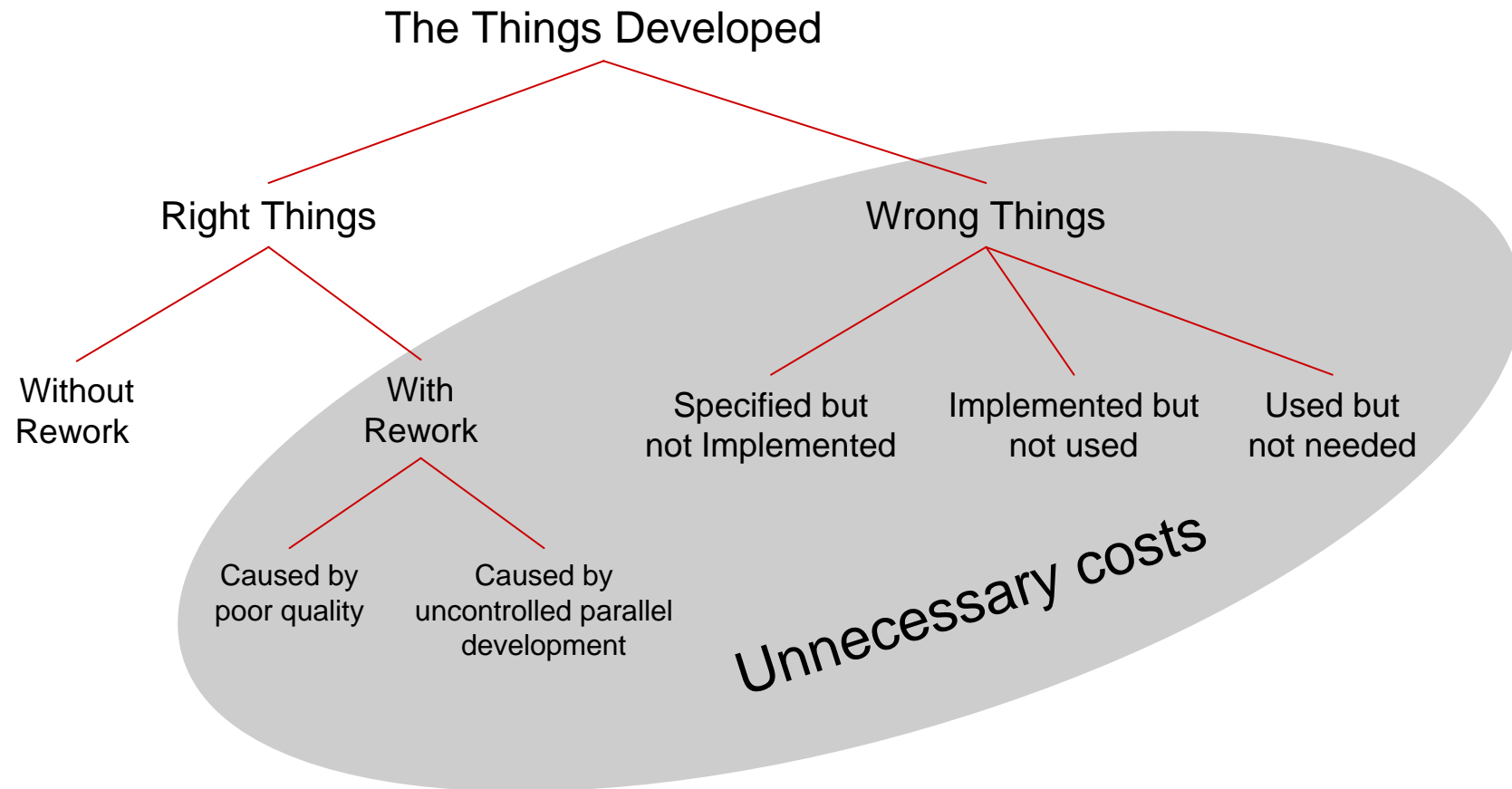
- Many people from different development sites involved
- The system is part of an even larger system of inter-connected systems
- The system is in operation

There is an ocean of solutions!

Cross Functional Teams Black Box Measurement CMMI
SEPG DSDM Test Automation Refactoring Baseline mgmt
Pair Programming TQM Inspections Python
Virtual Server MBD PROPS Empowerment Simulink
UML **GREAT! But what is the problem?** JP Work Packages
TeamCenter PDCA Lean Quality Ranking OO Purify
CleanRoom Traceability Function Points
Daily Build SCRUM Code Reviews 6Sigma Use Case
CCB JBuilder TEST RT Rhapsody Eclipse Visual Planning
Anatomies Prototyping Reuse Dimension GQM
Dymola DOORS Resource mgmt Agile
Test Driven Development Target Emulator

The General Overall Problem

We Must Reduce Development Cost!



Improving development performance is primarily reducing unnecessary development costs – **waste** – and must start with their identification.

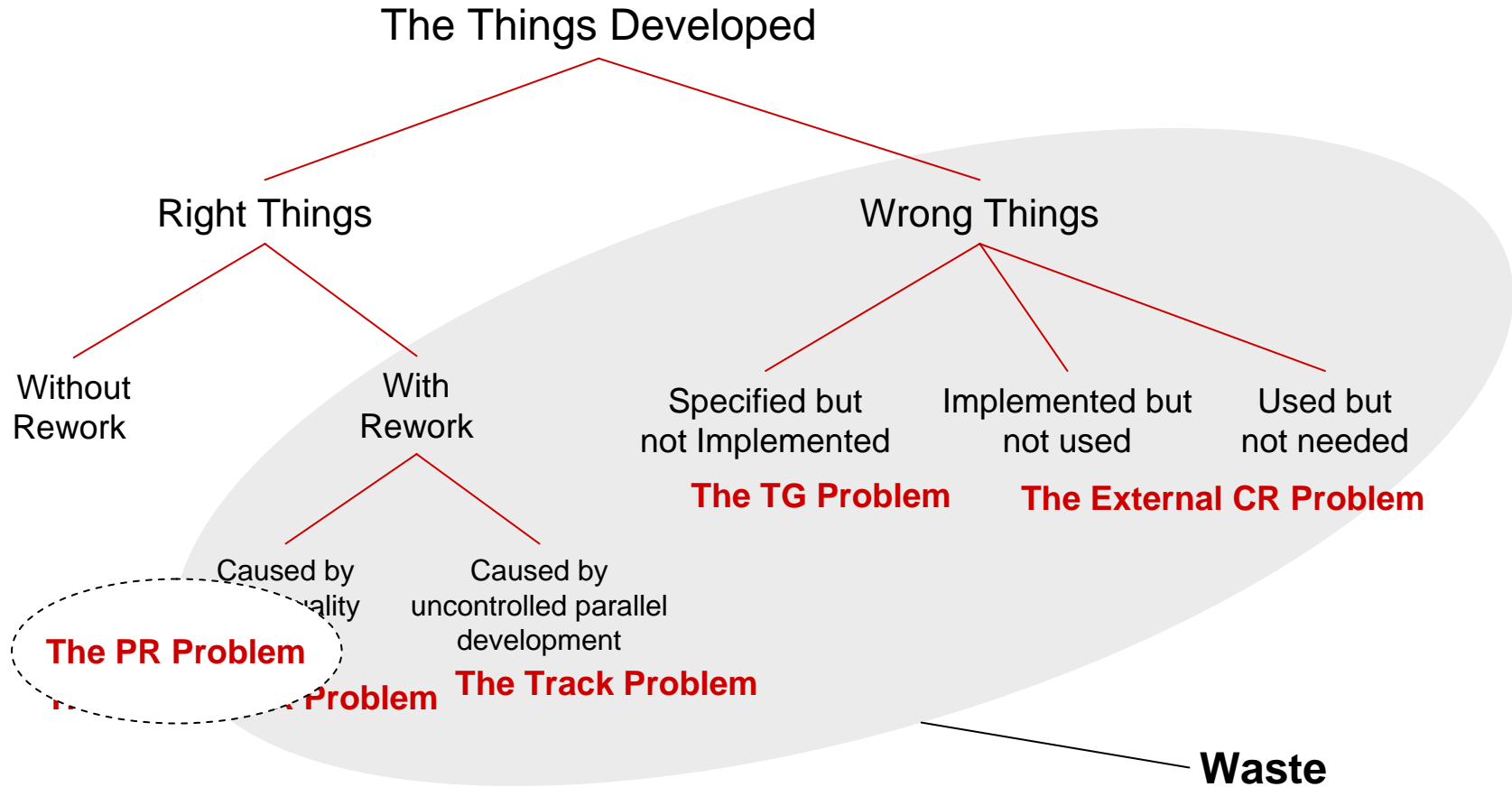
Some Examples

Problems Causing Unnecessary Costs

Unnecessary Costs	Problems (possible to measure!)
Rework caused by poor quality.	<u>The PR Problem</u> We find far too many costly faults and unwanted characteristics far too late.
	<u>The Internal CR Problem</u> The output from our “system/product analysis” is of insufficient quality.
Rework caused by uncontrolled parallel development.	<u>The TRACK Problem</u> We have far too many system- and component-versions that we develop and maintain in parallel.
Developing the wrong things.	<u>The External CR Problem</u> We are unable to efficiently respond to the rapidly changing environment in which our systems must serve.
	<u>The TG Problem</u> We don't stop the development of wrong things in time.

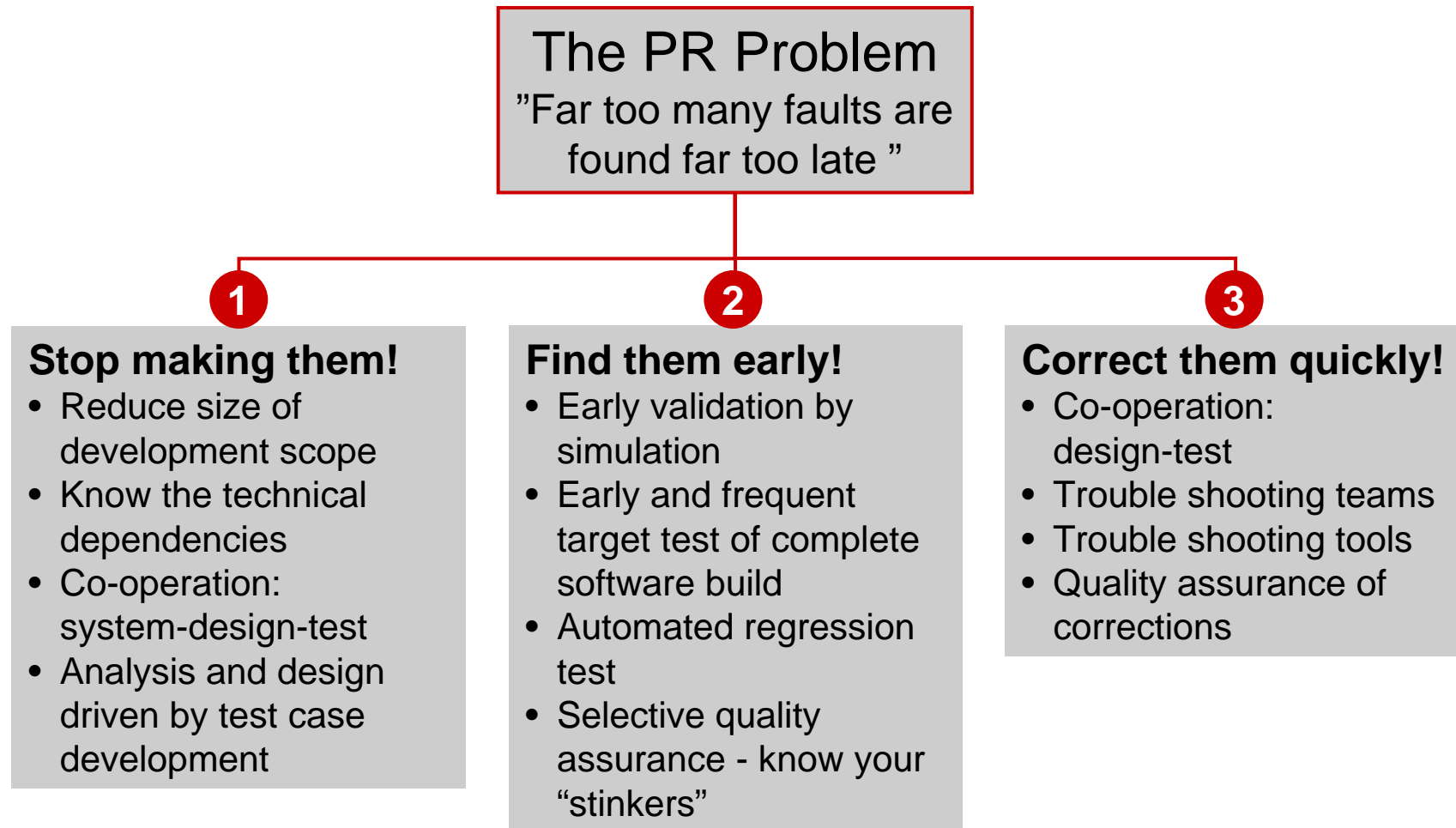
The General Overall Problem

We Must Reduce Development Cost!



Let's look closer to one of the five problems

How can we get rid of the "PR Problem"?



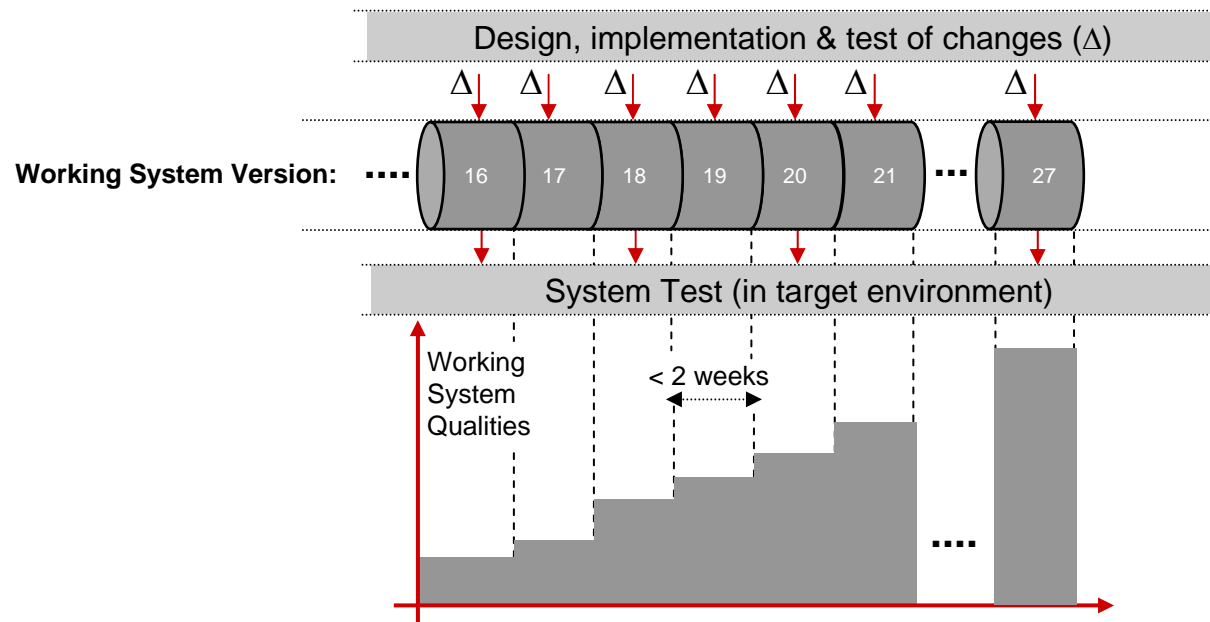
Develop in Small Steps!

DEVELOPING LARGE SYSTEMS IN SMALL STEPS

The Strategy

How do we get rid of the “PR Problem”?

Develop the system in many small steps!

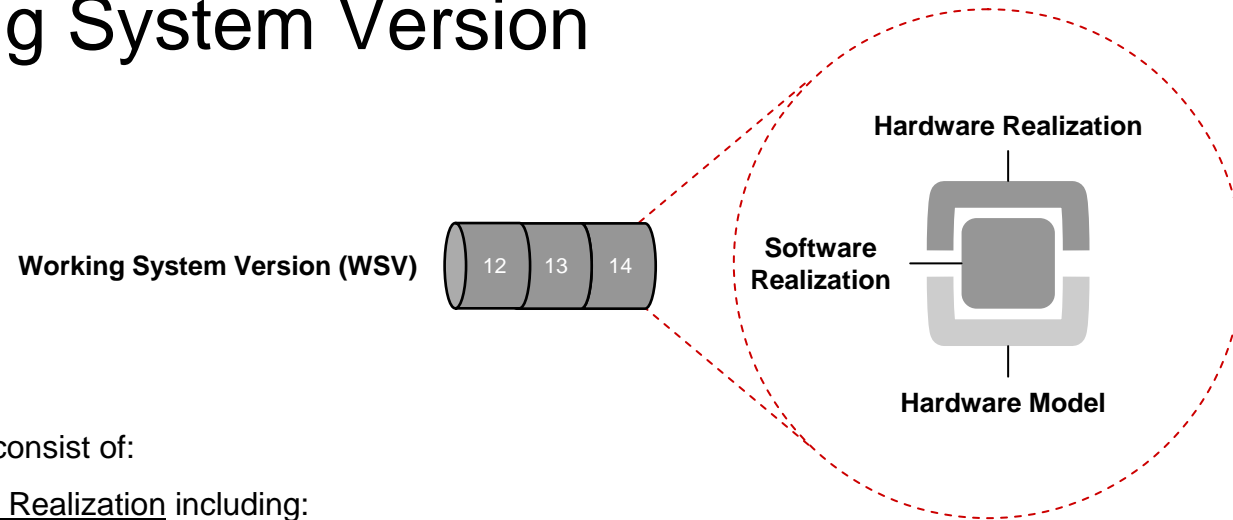


Each new step:

- is the realization of a planned system change (Δ)
- results in a working system version ready for advanced tests in target environment
- is made within a few weeks

What's inside?

The Working System Version



A **System Version** may consist of:

A version of the Software Realization including:

- Software models, source code files, build procedures and load modules.
- Product documents that describe the function, characteristics, and construction of the software realization.

A version of the Hardware Realization including:

- Models for validation and design together with hardware components and equipment.
- Product documents that describe the function, characteristics, and construction of the hardware realization.

A version of the Hardware Model including:

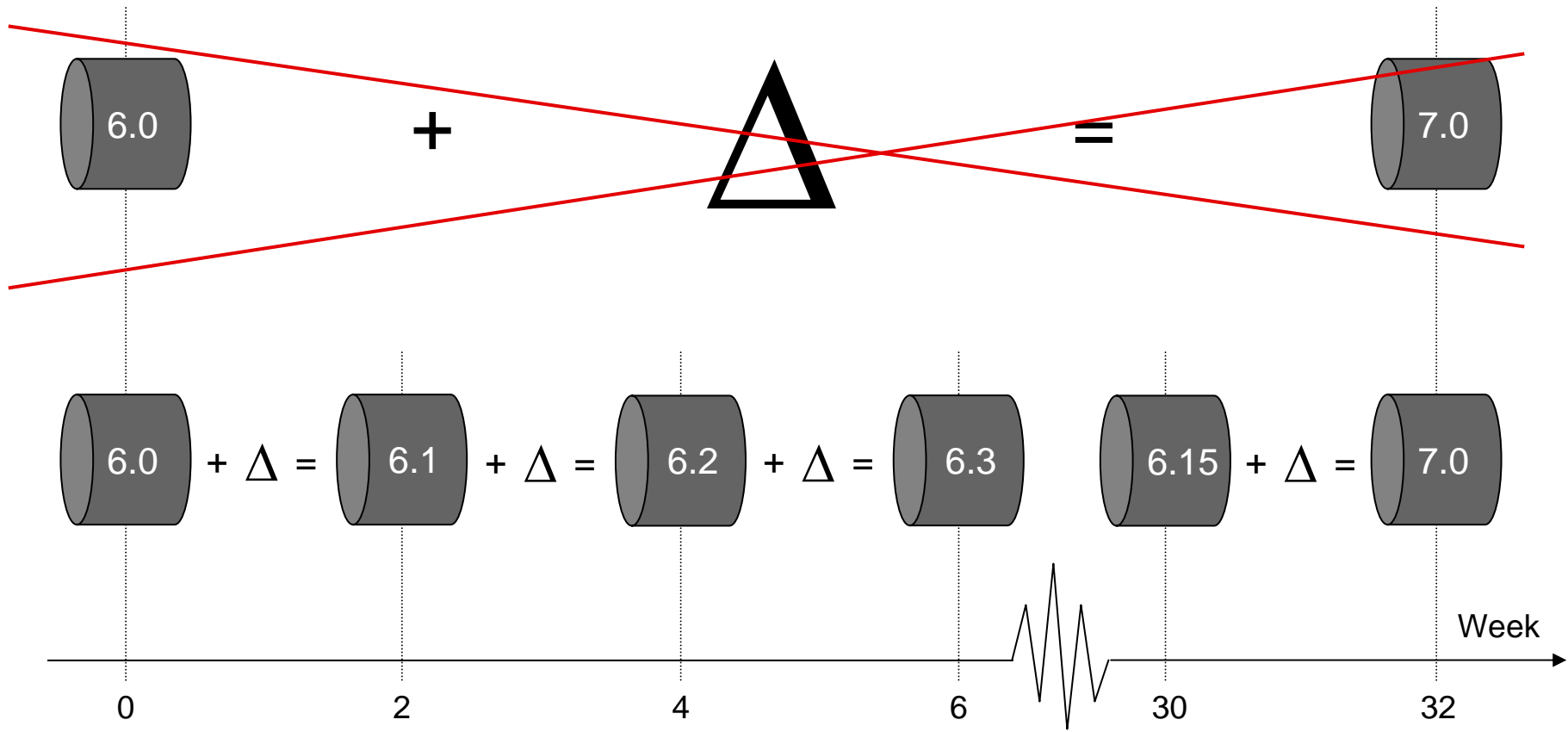
- Models simulating the hardware realization of the system. Used for test of the software realisation and other systems.
- Documents that describe the use and design of the hardware model.

A **System Version is Working** when:

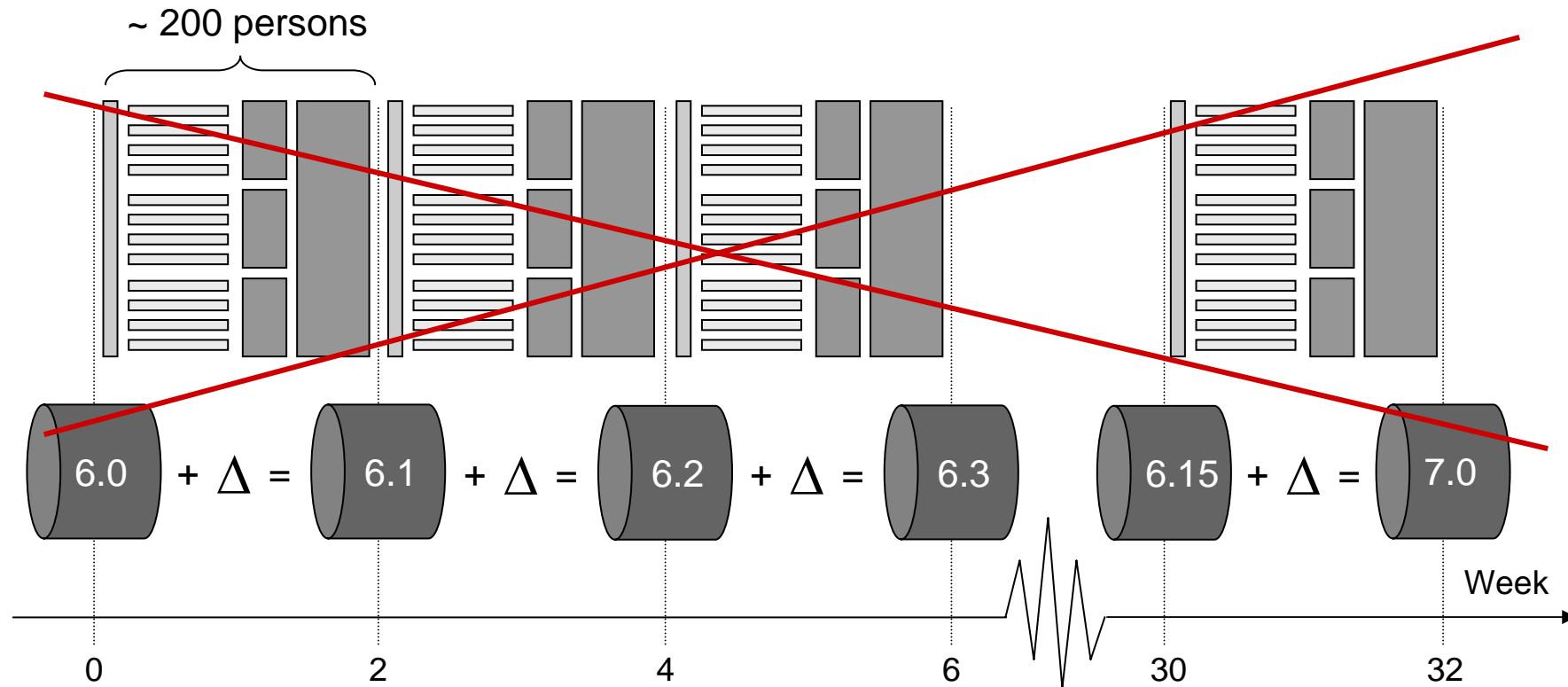
The software realization together with the hardware realization, or hardware model, has been demonstrated to work without any major remarks in a target like test environment.

The General Idea

Many small steps instead of a single big one

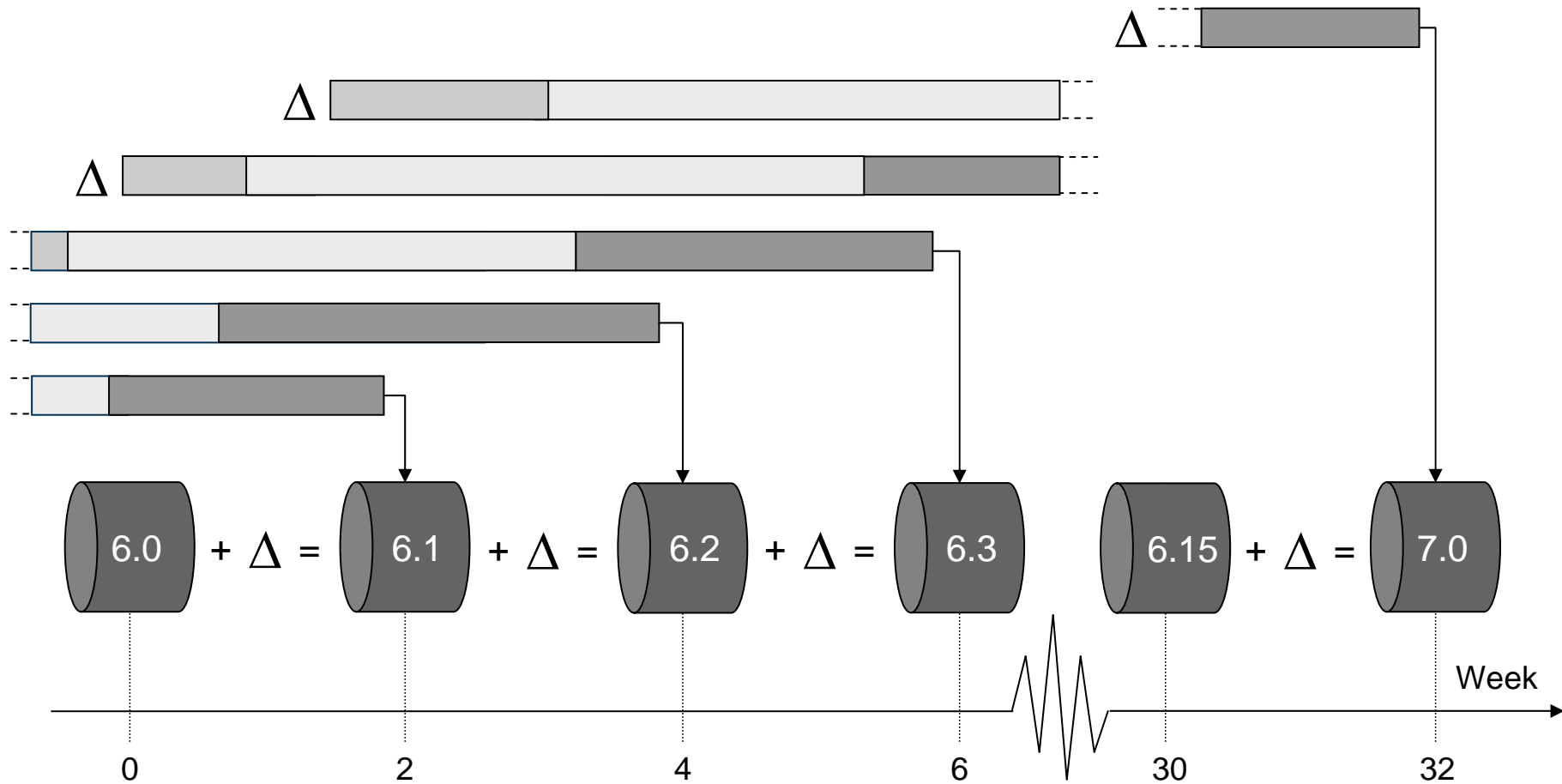


For large systems, sequential incremental development doesn't work!



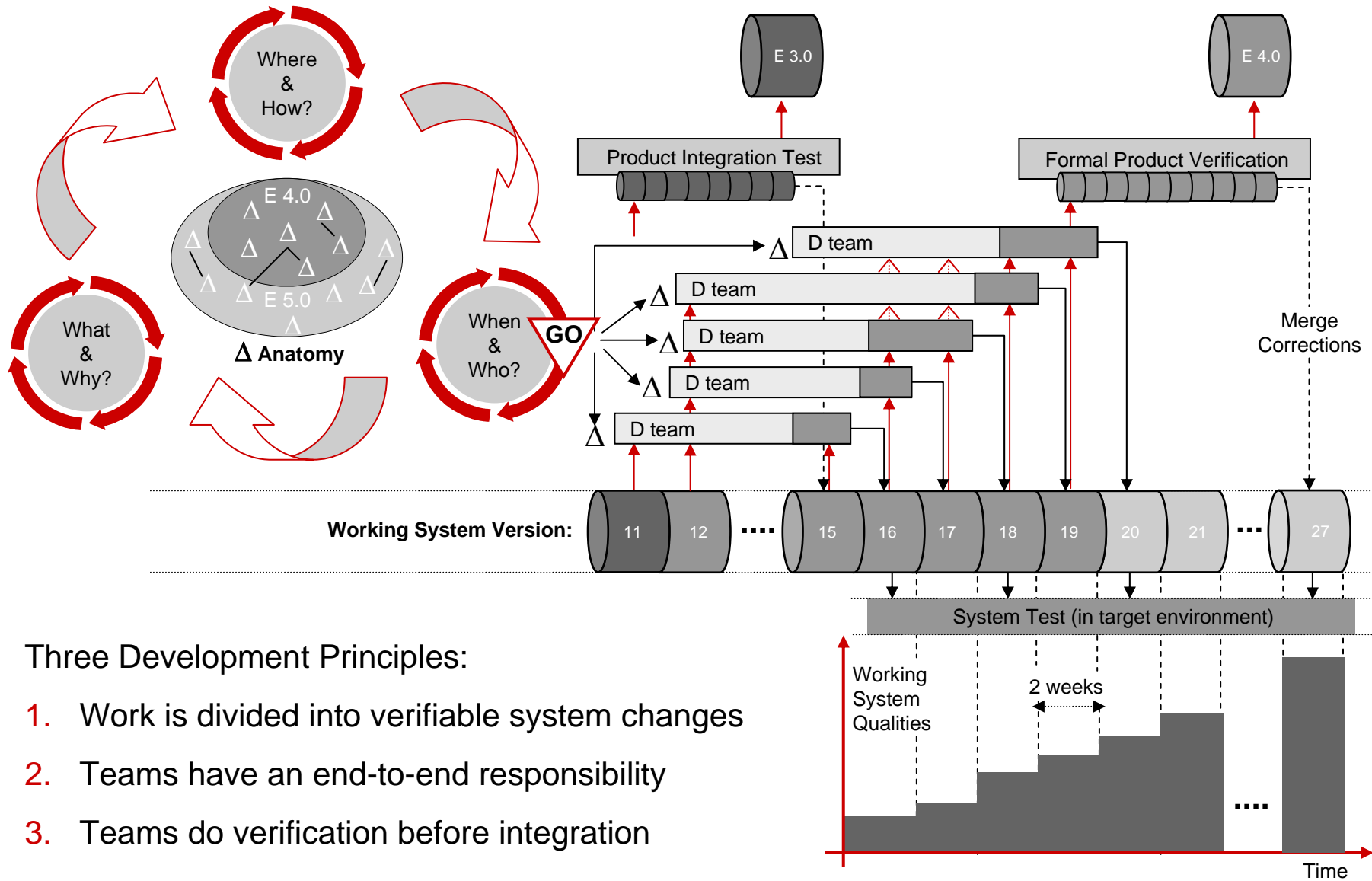
Limiting system changes to be “something that can be designed, integrated and tested within two weeks” will make the process slow and inefficient. This since we neither can define nor coordinate “two week changes” that will involve and make use of all our resources in an efficient way

But developing increments in parallel works!



Mastering **parallel incremental development** is the key to successful stepwise development of large systems

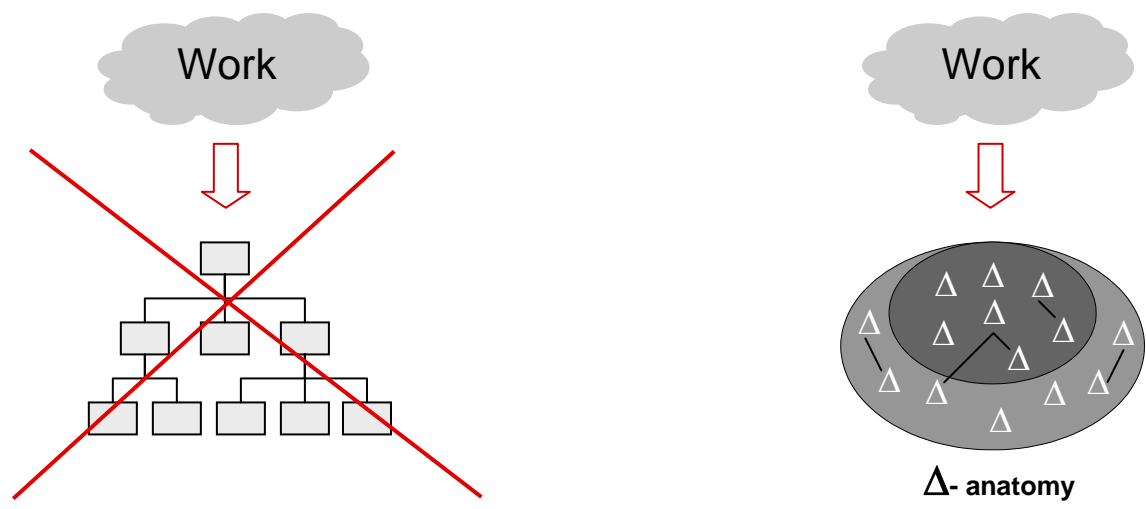
The System Development Strategy



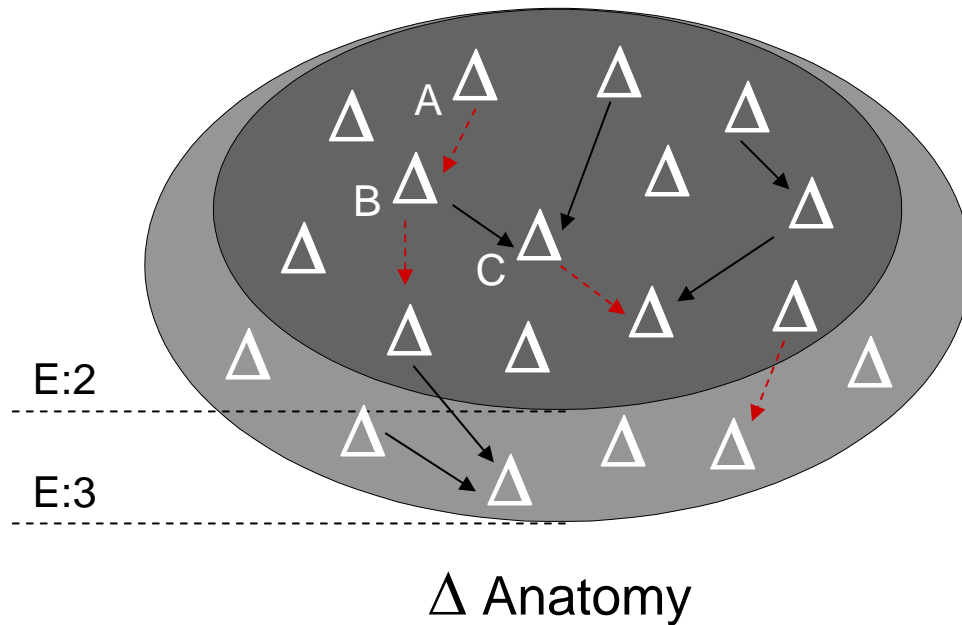
Development Principle #1

Work is divided into verifiable system changes

The scope for a system release (edition) is divided into small and verifiable system changes instead of distributed on sub-systems and organisational units



Parallel incremental development is mastered with the Delta Anatomy



The anatomy shows all currently planned system changes (Δ) and their dependencies.

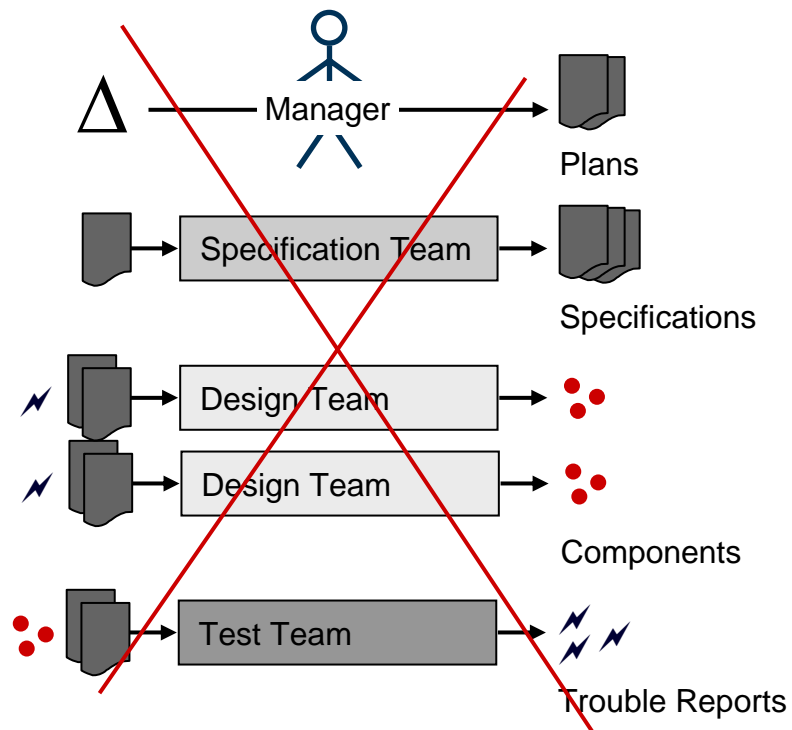
The dependencies constrain the order in which changes can be done, and determines the possible level of parallelism.

- ▶ “Design Dependency” (A should be delivered before design of B can start)
- ▶ “Verification Dependency” (B must be delivered before verification of C can start)

Development Principle #2

Teams have an end-to-end responsibility

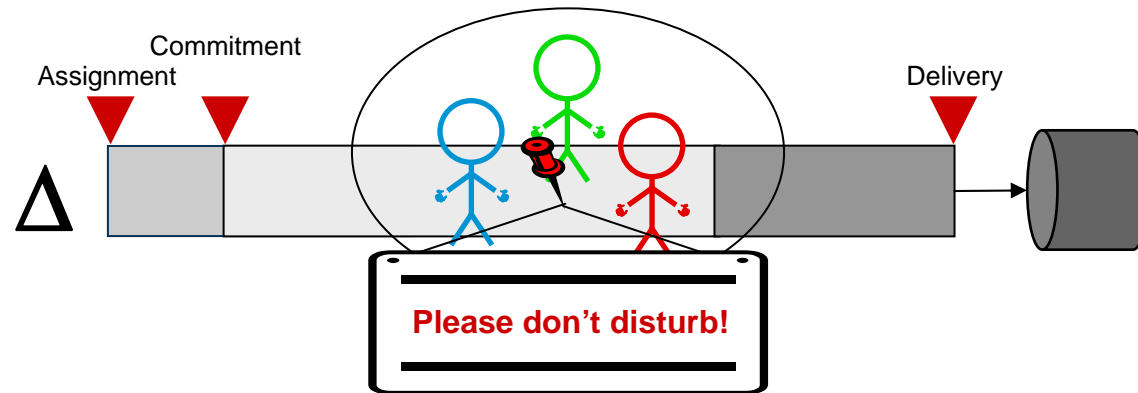
We have cross functional teams with an end-to-end responsibility for the realisation of a new system version instead of specialist teams delivering parts to each other



Needed System Change



Preconditions for an end-to-end responsibility



- Form teams that really have an end-to-end scope (from “needed change” to “new system version”)
- State clear boundaries in terms of criteria that must be fulfilled (1) before an assignment can start, and (2) before a delivery can be made
- Give teams the possibility to make a true commitment, that is, time for analysis, planning, staffing, and negotiation
- Let the teams decide how to do the work, and do everything to provide them with the environment and support they ask for
- Encourage and ease communication within and among teams (“team rooms“, “daily stand-up meetings“, “coordination meetings” etc.)

Development Principle #3

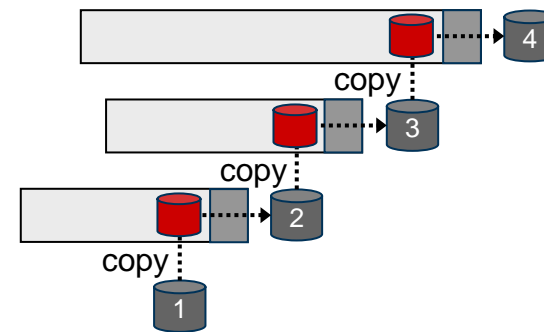
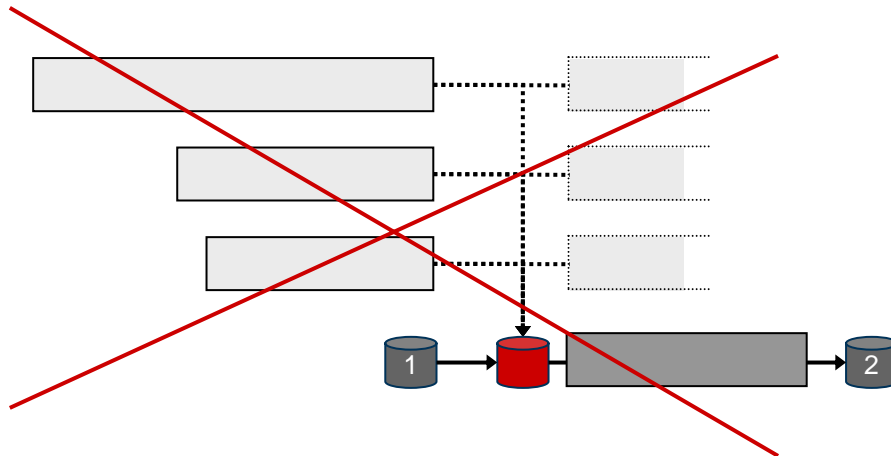
Teams do Verification Before Integration

Teams do stepwise integration of verified system qualities instead of “big bang” integration of unverified components

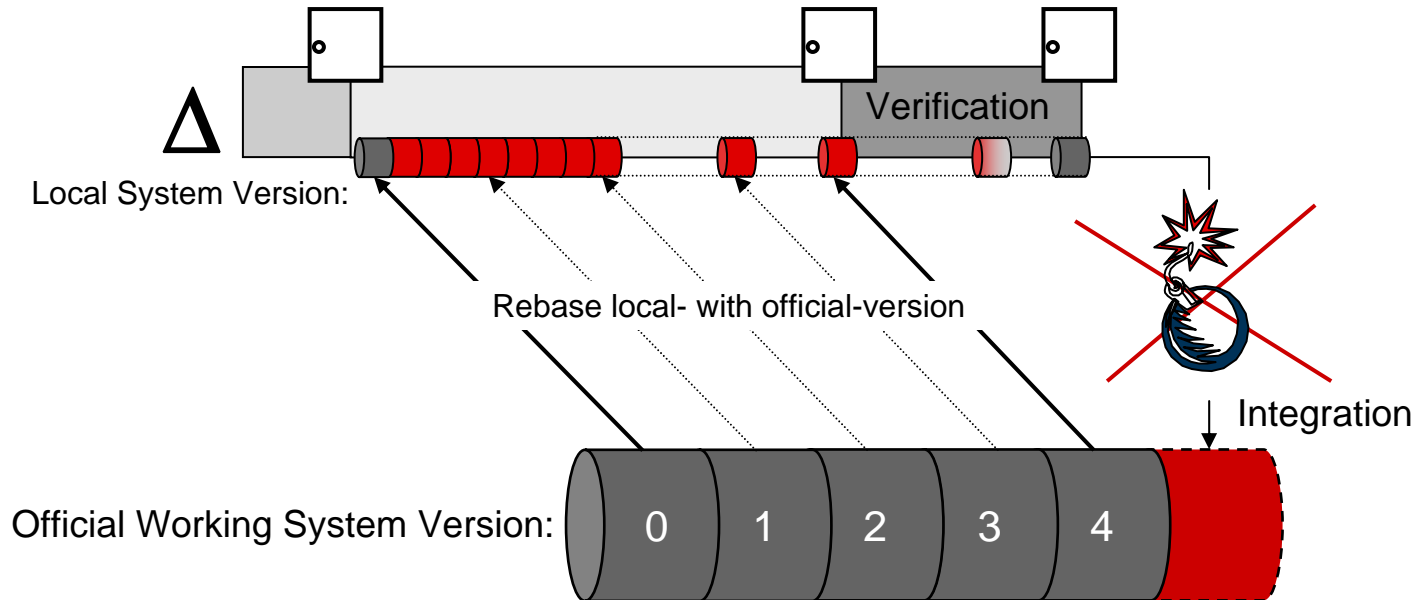
“Big bang” integration of unverified components



Stepwise integration of tested system qualities

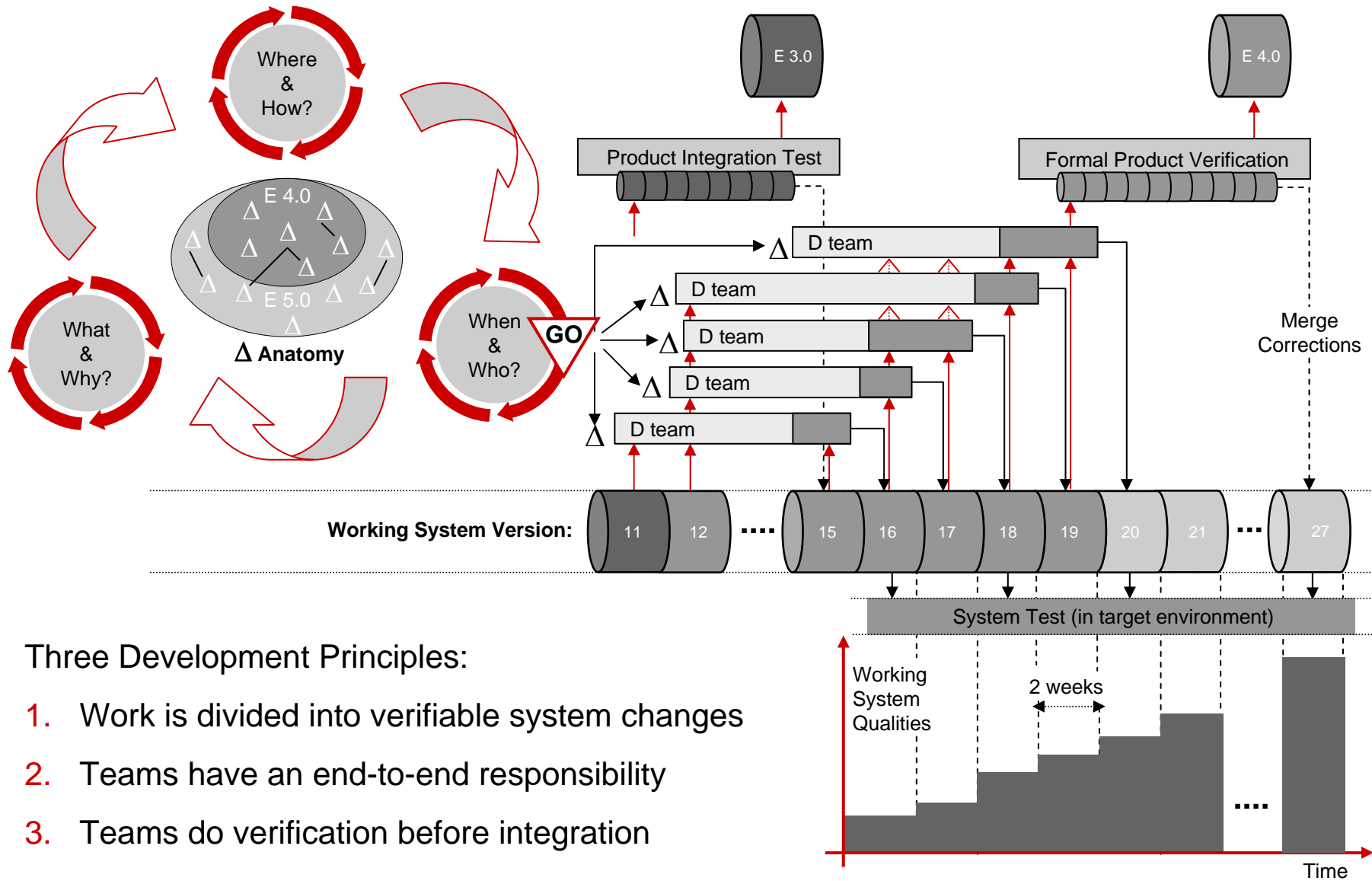


Verification Before Integration - Avoid The Bomb!



- Δ -development is isolated to a local system version (development branch)
- The local version is rebased whenever a new official version is published. The latest official version is used for “verification before integration”
- A number of “quality doors” must be passed before delivery

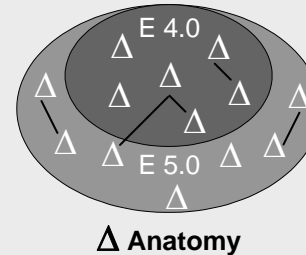
The System Development Strategy



Managing System Development

System Engineering

Transform edition content into small verifiable system changes (Δ) and clarify their dependencies in an anatomy.

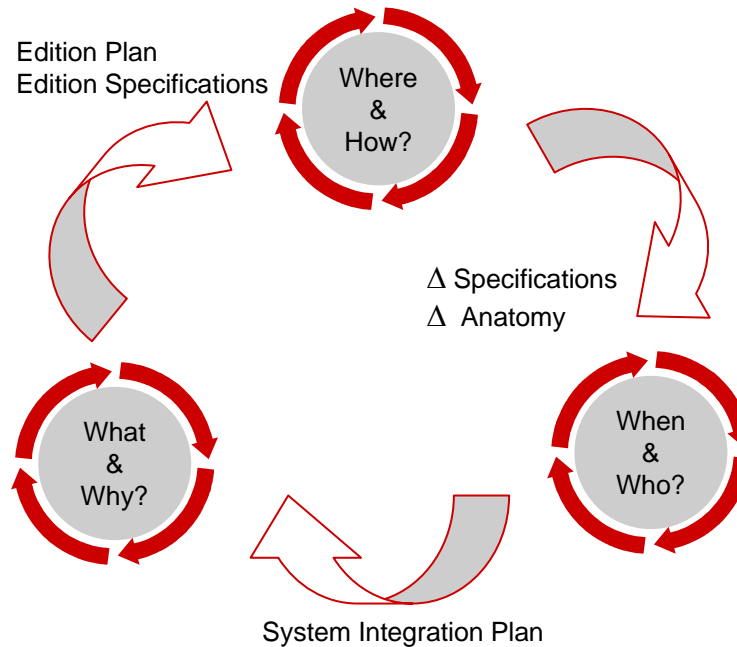


System Mgmt



System Edition Specifications

Specify the content of system editions in terms of needed functions and qualities, and maintain a system edition plan.



System Project Mgmt

Add new Δ into the integration plan and kick off their realization.

