

Lecture 11

Configuration and Version Management

31.3.2014

Week	Lecture	Exercise
10.3	Quality in general; Quality management systems	Patterns
17.3	Dependable and safety-critical systems	ISO9001
24.3	Work planning; effort estimation	Code inspections
31.3	Version and configuration management	<i>Effort estimation</i>
7.4	Role of software architecture; product families; software evolution	?
14.4	Specifics of some domains, e.g. web system and/or embedded and real time systems	Break?
21.4	Easter	Break?
28.4	Software business, software start-ups	?
5.5	Last lecture; summary; recap for exam	?

News

- Sprint reviews on week 17:
 - Most of them (if not all) will move to week 18

Learning goals of today

- What are Configuration and Version Management?
- Why they are important?
- How to organize version and configuration management?
- Tools and techniques
- Example tools SVN and GIT
 - Basic principles behind the guidelines you get during other courses

Material

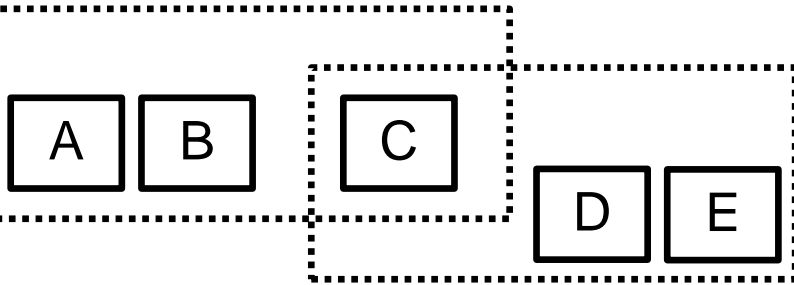
- Haikala & Mikkonen: Chapter 13
"tuotteenhallinta" (Product management)
 - A bit short chapter – reading of additional material recommended
- Sommerville: Chapter 25
- Alexis Leon: Software Configuration Management
 - I have used this book as background material

Motivation

Leenu



Liinu



Leenu wants to compile the system but how to ensure that components D and E are in adequate state?

Liinu has the same problem with components A and B.

They need a certain **version** of different components.

Collection of interoperable versions of modules is called **configuration**.

Motivation – becomes difficult very fast

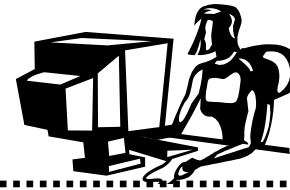
Leenu



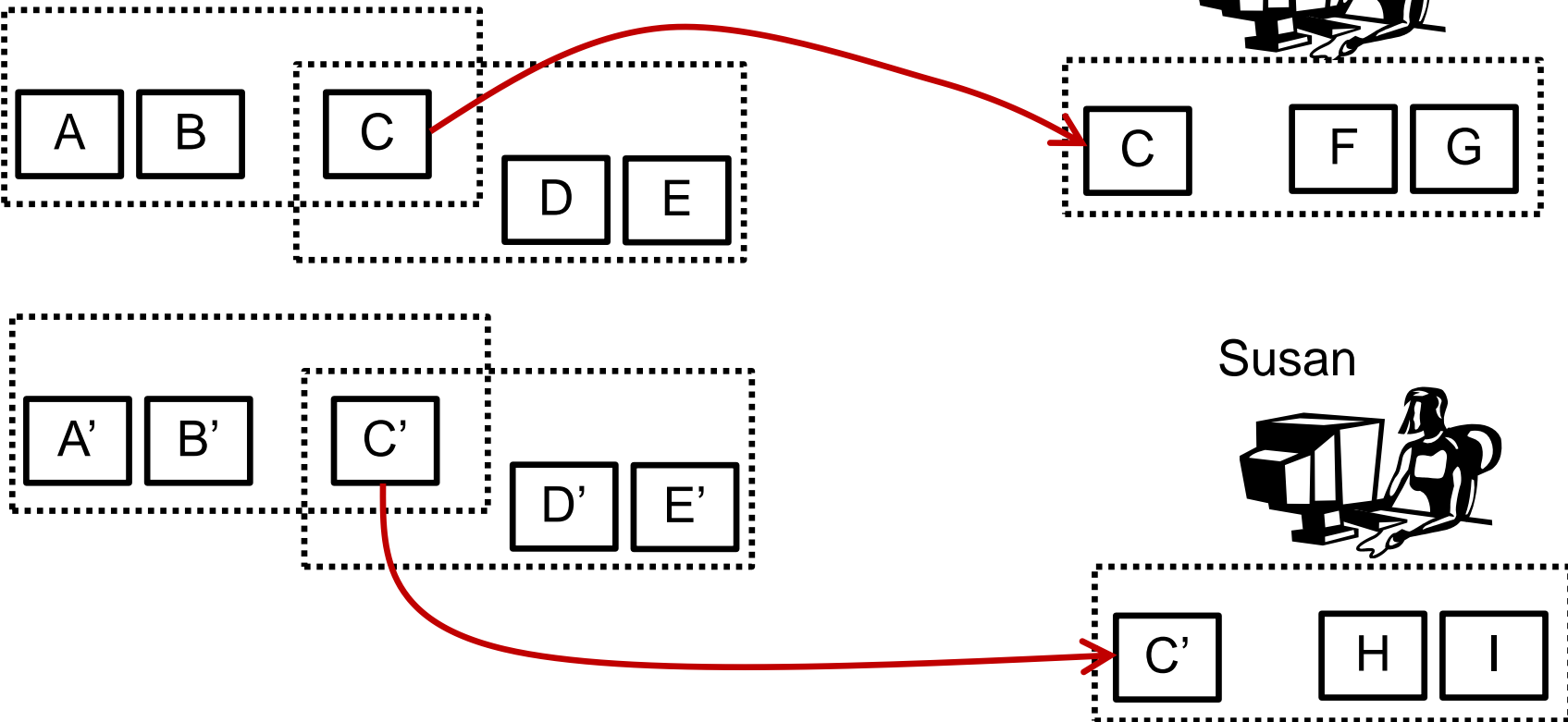
Liinu



Ann



Susan

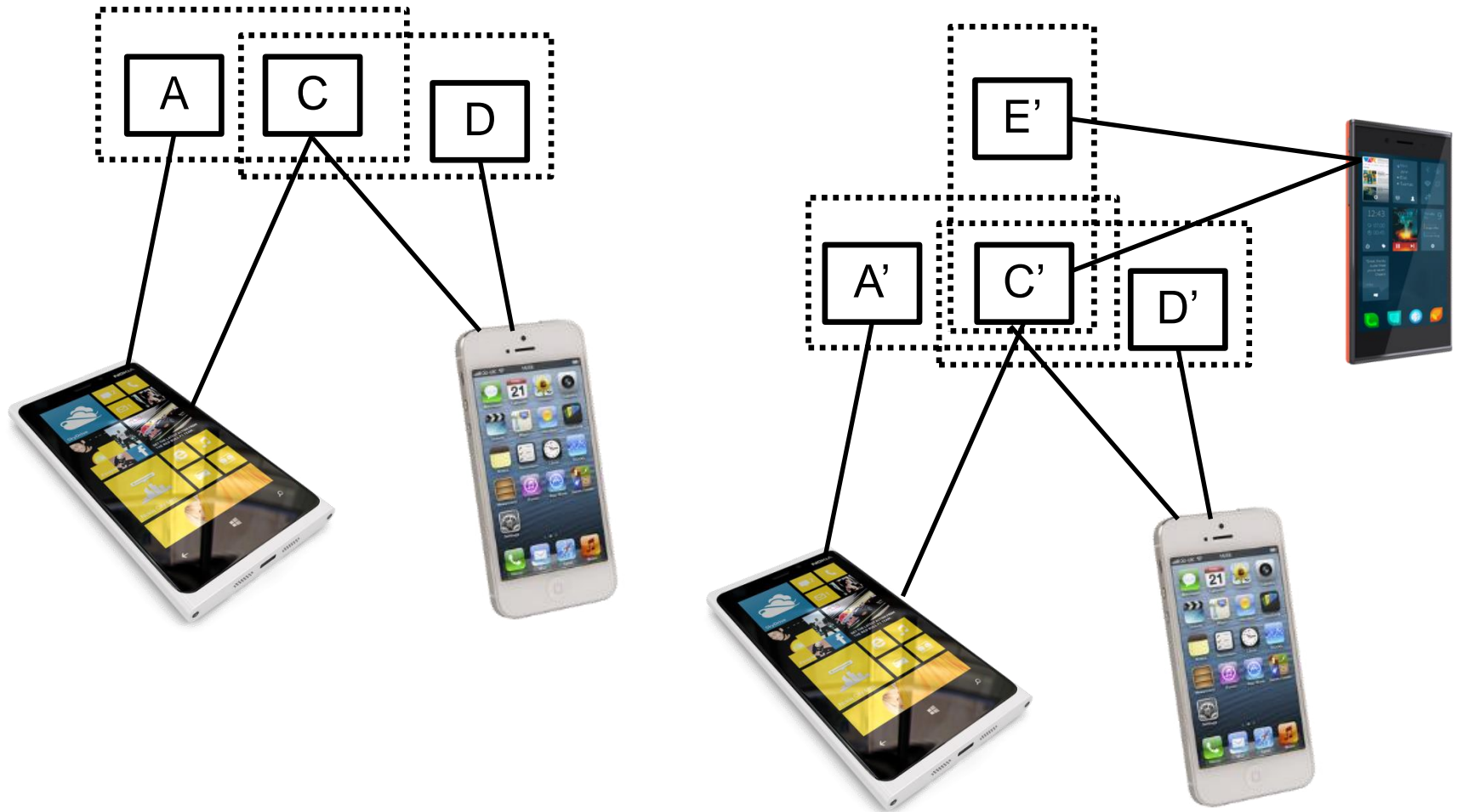


Some definitions

- Wikipedia
 - In software engineering, software configuration management (SCM) is the task of tracking and controlling changes in the software, part of the larger cross-discipline field of configuration management
 - SCM practices include revision control and the establishment of baselines.
 - If something goes wrong, SCM can determine what was changed and who changed it.
 - If a configuration is working well, SCM can determine how to replicate it across many hosts.

- The configuration of a system is the functional and physical characteristic of hardware or software as set forth in technical documentation or achieved in a product;
it can also be thought of as a collection of specific versions of hardware, firmware or software items combined according to specific build process or serve a particular purpose.
- Configuration management (CM), then, is the discipline of identifying the configuration of a system at distinct points in time for the purpose of systematically controlling changes to the configuration and maintaining the integrity and traceability of the configuration throughout the system life cycle.

One example more (Nokia, Apple and Jolla Images)

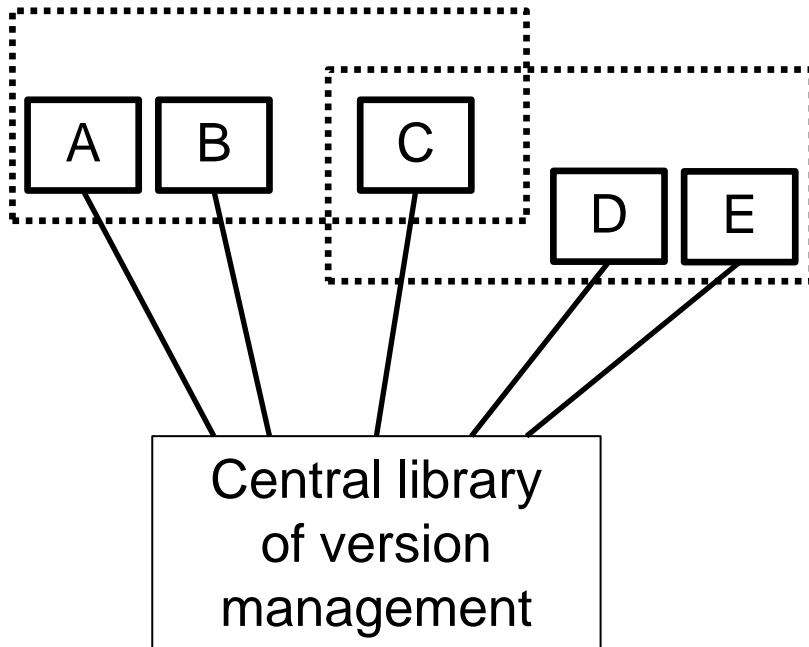


Typical solutions

Leenu



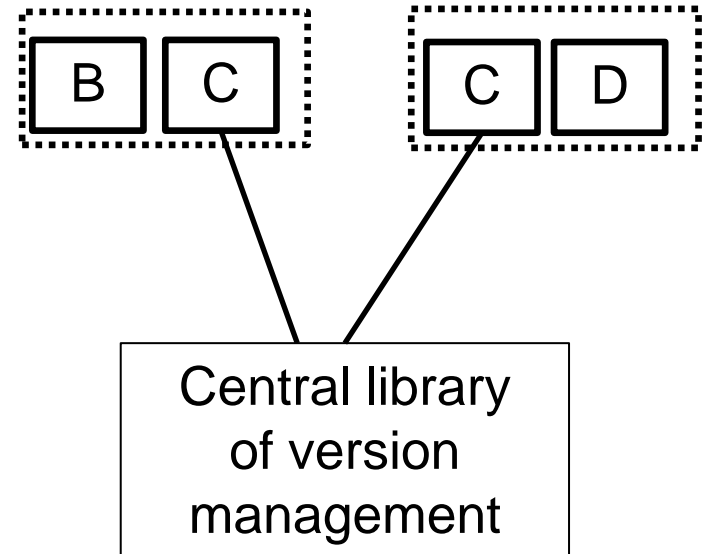
Liinu



Leenu



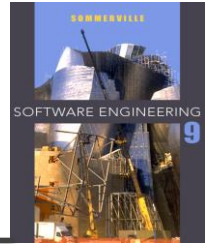
Liinu



So, what is configuration management?

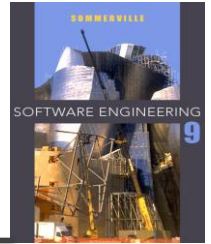
- **Change management:** managed way to decide which change ideas to implement and when.
- **Version management:** keep track of multiple versions of components and ensure that changes by different developers do not disturb each other.
- **System building:** collect and assemble correct versions of required components and then compile.
- **Release management:** prepare for external releases and keep track of external releases.

CM terminology



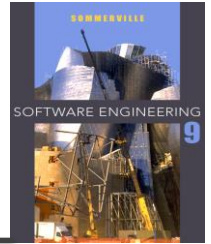
Term	Explanation
Configuration item or software configuration item (SCI)	Anything associated with a software project (design, code, test data, document, etc.) that has been placed under configuration control. There are often different versions of a configuration item. Configuration items have a unique name.
Configuration control	The process of ensuring that versions of systems and components are recorded and maintained so that changes are managed and all versions of components are identified and stored for the lifetime of the system.
Version	An instance of a configuration item that differs, in some way, from other instances of that item. Versions always have a unique identifier, which is often composed of the configuration item name plus a version number.

CM terminology



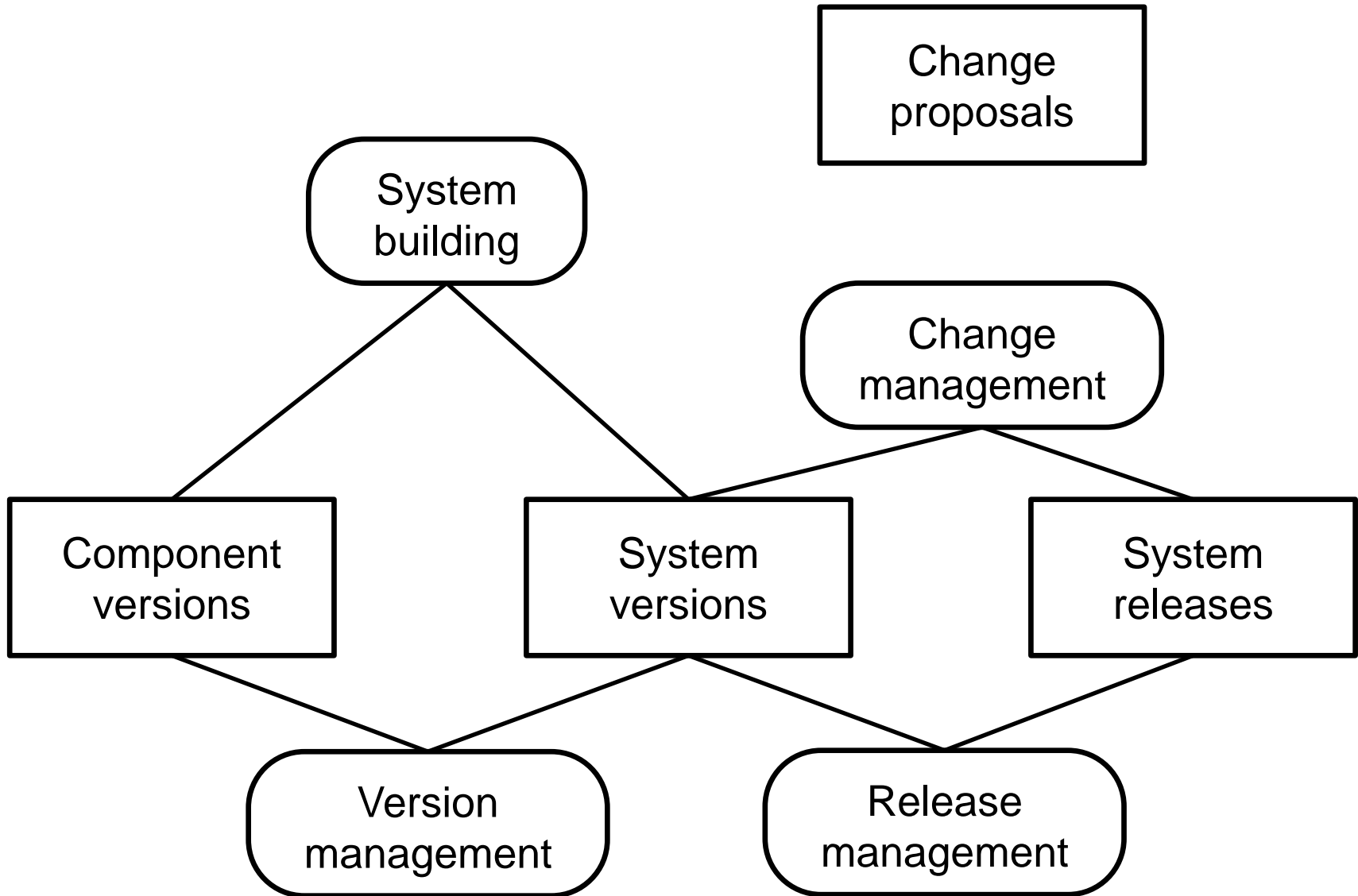
Term	Explanation
Baseline	A baseline is a collection of component versions that make up a system. Baselines are controlled, which means that the versions of the components making up the system cannot be changed. This means that it should always be possible to recreate a baseline from its constituent components.
Codeline	A codeline is a set of versions of a software component and other configuration items on which that component depends.
Mainline	A sequence of baselines representing different versions of a system.
Release	A version of a system that has been released to customers (or other users in an organization) for use.

CM terminology



Term	Explanation
Workspace	A private work area where software can be modified without affecting other developers who may be using or modifying that software.
Branching	The creation of a new codeline from a version in an existing codeline. The new codeline and the existing codeline may then develop independently.
Merging	The creation of a new version of a software component by merging separate versions in different codelines. These codelines may have been created by a previous branch of one of the codelines involved.
System building	The creation of an executable system version by compiling and linking the appropriate versions of the components and libraries making up the system.

Figure 25.1 in Sommerville



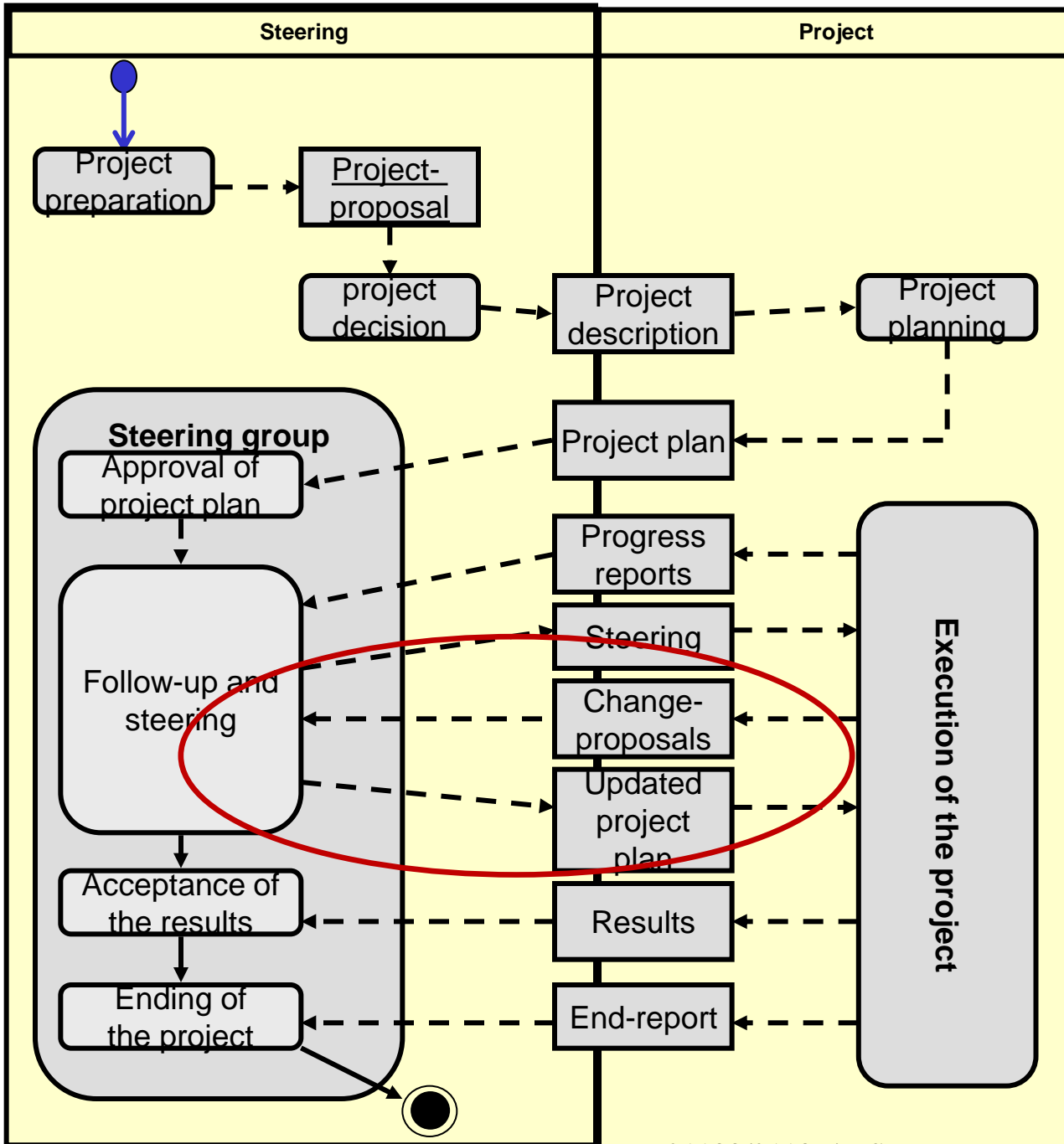
Software configuration management vs Configuration management software

- Latter is about managing system, taking care of updates, installation etc.

Change management

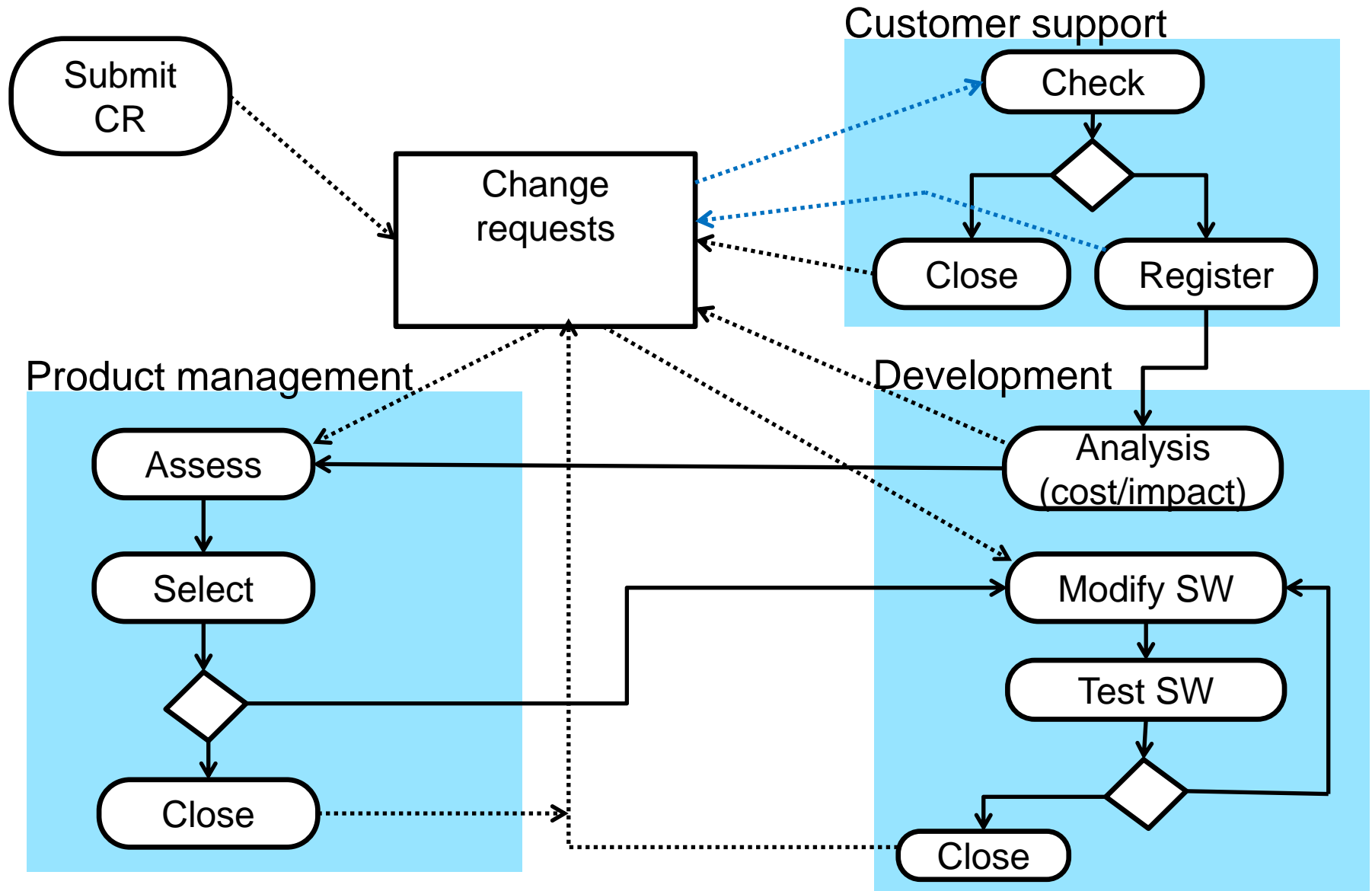
Development vs. maintenance

- Most of the development is actually about applying changes
 - Products often have several versions that are implemented in separate projects
 - In a project plans or requirements change during the project
 - Software might be in a maintenance mode
- Different types of changes:
 - Bug fixes
 - Performance improvements
 - React to environmental changes (HW, legal, ...)
 - New features of other requirement changes
- Contrast to maintenance of machines, e.g. cars, where maintenance is due to physical and chemical wearing



From lecture 3
(project
management)

Figure 25.1 in Sommerville



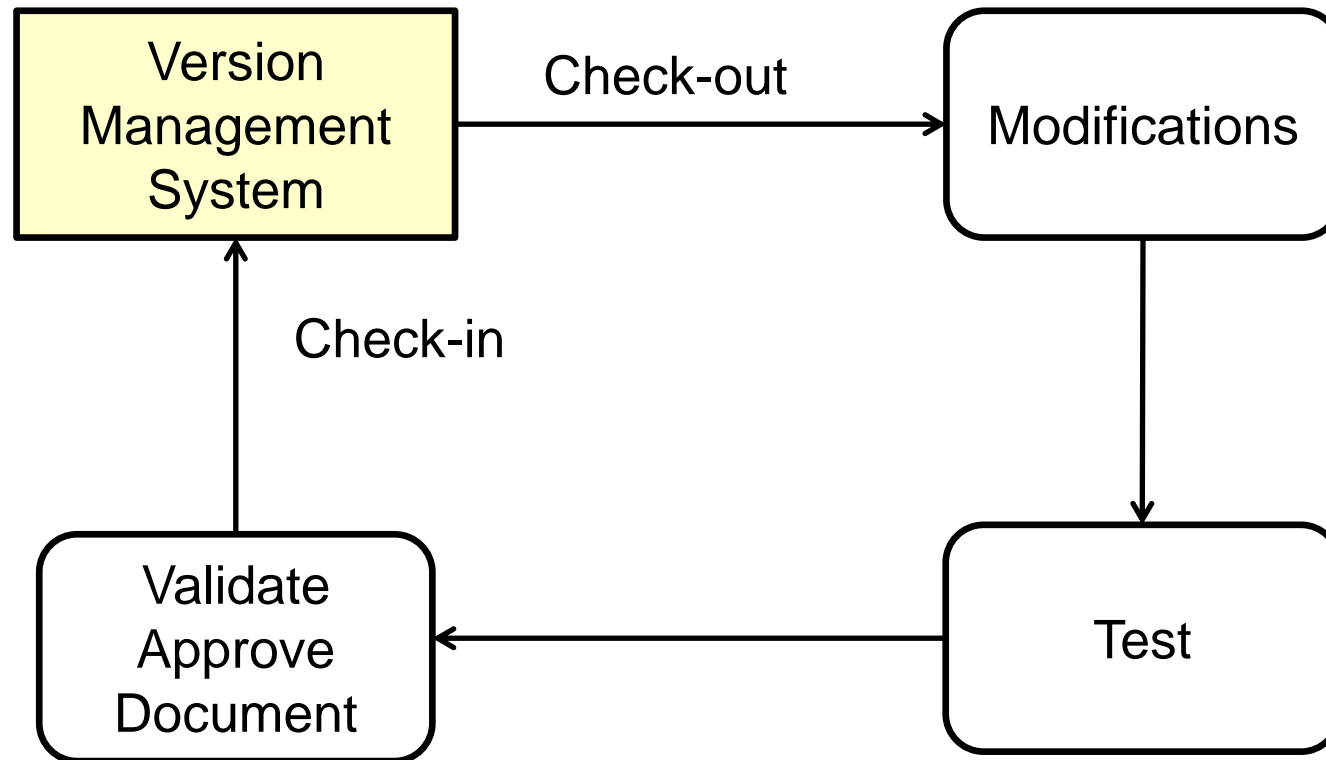
Question

(discussion during lecture)

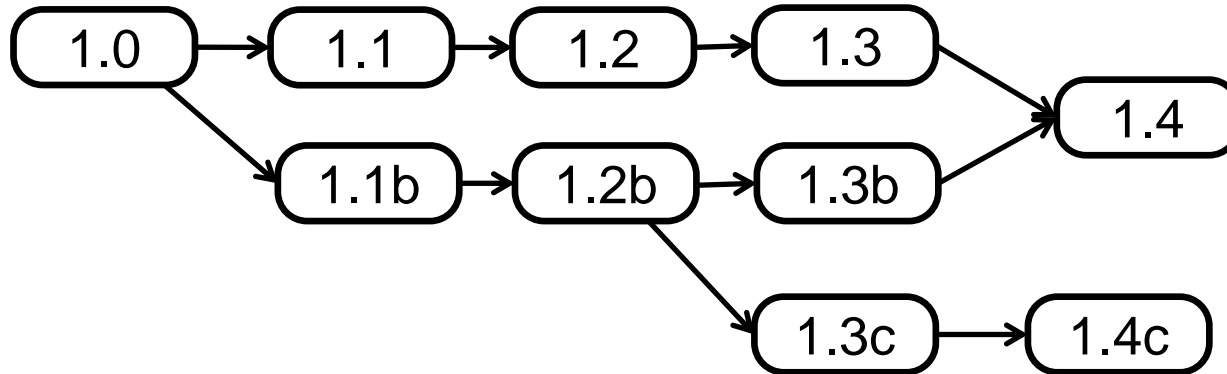
- How should change management be done in Scrum

Version Management

Simple view of version management



Versions; version trees



Functions of version management

- Version and release identification
- Storage management to optimize storage usage
- Change history recording
- Independent development
- Project support

A short history of version control systems

- SCCS – the first widely used
 - Origin from 1972
 - Was a standard part of many Unix systems
 - Sccsid string:
`static char sccsid[] = "@(#)ls.c 8.1 (Berkeley) 6/11/93";`
- RCS was built as an alternative to SCCS
 - Stores latest version and backward "deltas"
 - Supported binary files
- CVS introduced client-server architecture
 - Based on RCS
- SVN & GIT the most common open source tools today
- Many commercial alternatives

SVN vs GIT

- Different models
 - With SVN users work on common repository and **commit** changes to the latest version
 - In GIT users get their own copy of repository and commits to that. The changes are made visible to others after merging
- Concept of version
 - SVN has version numbering
 - GIT has names
but you can simulate numbering by
`% git tag 1.6.1 -m 'Release 1.6.1'`

Start work with

```
git clone url
```

```
svn checkout url
```

Make branch to you

```
git checkout -b branch  
origin/branch
```

```
svn switch url
```

Get latest from server

```
git pull
```

```
svn update
```

Add/remove files

```
git add file  
git rm file
```

```
svn add file
```

```
svn rm file
```

Commit changes

```
git commit -a  
(git pull)  
git merge  
git push
```

```
svn commit
```

Installing and setting up a tool is not enough for your project

- You need to agree and document your practices
- When to commit
- When to branch
- How to tag

Log Messages - D:\user\systa\Papers\ResourceDescription

From: 13. 2.2014 To: 29. 3.2014 Messages, authors and paths

Revision	Actions	Author	Date	Message
229		hylli	9:50:28, 28. maaliskuuta 2014	intro pikkukorjauksia
228		hylli	9:10:44, 28. maaliskuuta 2014	samuelin muokkaukset lisätty
227		systa	19:57:37, 27. maaliskuuta 2014	
226		systa	19:46:04, 27. maaliskuuta 2014	
225		systa	19:39:54, 27. maaliskuuta 2014	
224		systa	18:06:31, 27. maaliskuuta 2014	
223		systa	18:01:43, 27. maaliskuuta 2014	

samuelin muokkaukset lisätty

Action	Path	Copy from path	Revision
Modified	/dokumentit/resource_description/description.tex		
Modified	/dokumentit/resource_description/document.tex		
Modified	/dokumentit/resource_description/implementation.tex		
Modified	/dokumentit/resource_description/intro.tex		

Showing 52 revision(s), from revision 151 to revision 235 - 1 revision(s) selected.

☐ Hide unrelated changed paths
☒ Stop on copy/rename
☐ Include merged revisions

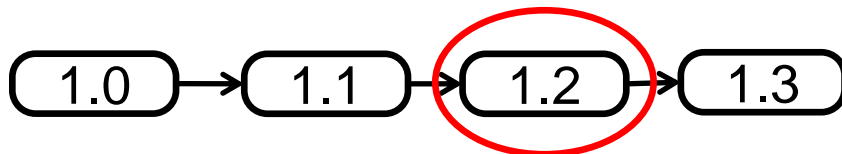
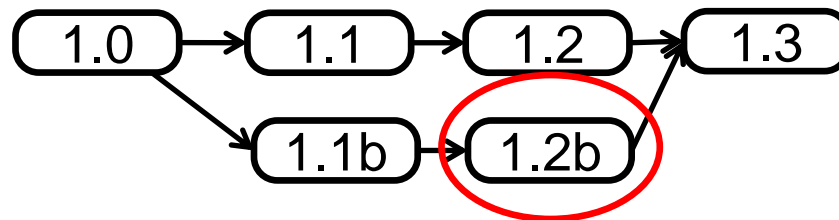
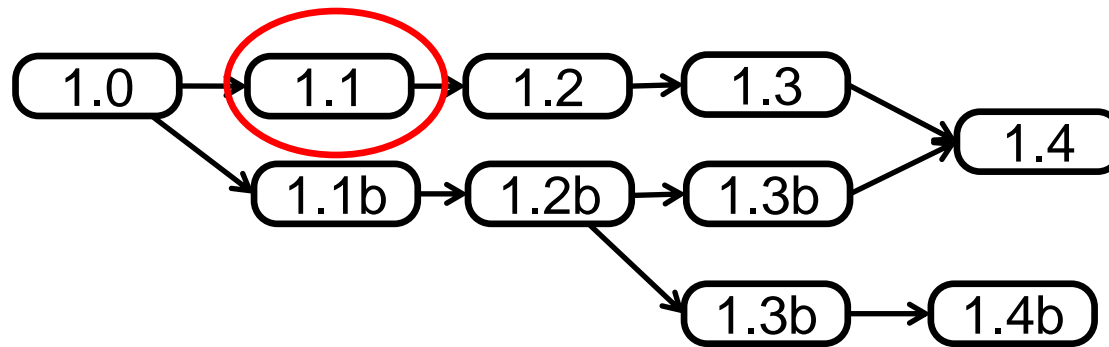
Show All Next 100 Refresh

Statistics Help OK

Don't
do as
your
teacher
does,
do as
he
says.

Configuration management

Configuration: collection of certain versions of components



Current project-based version control tools like SVN and GIT can (sometimes) also be used for configuration management.

Build management

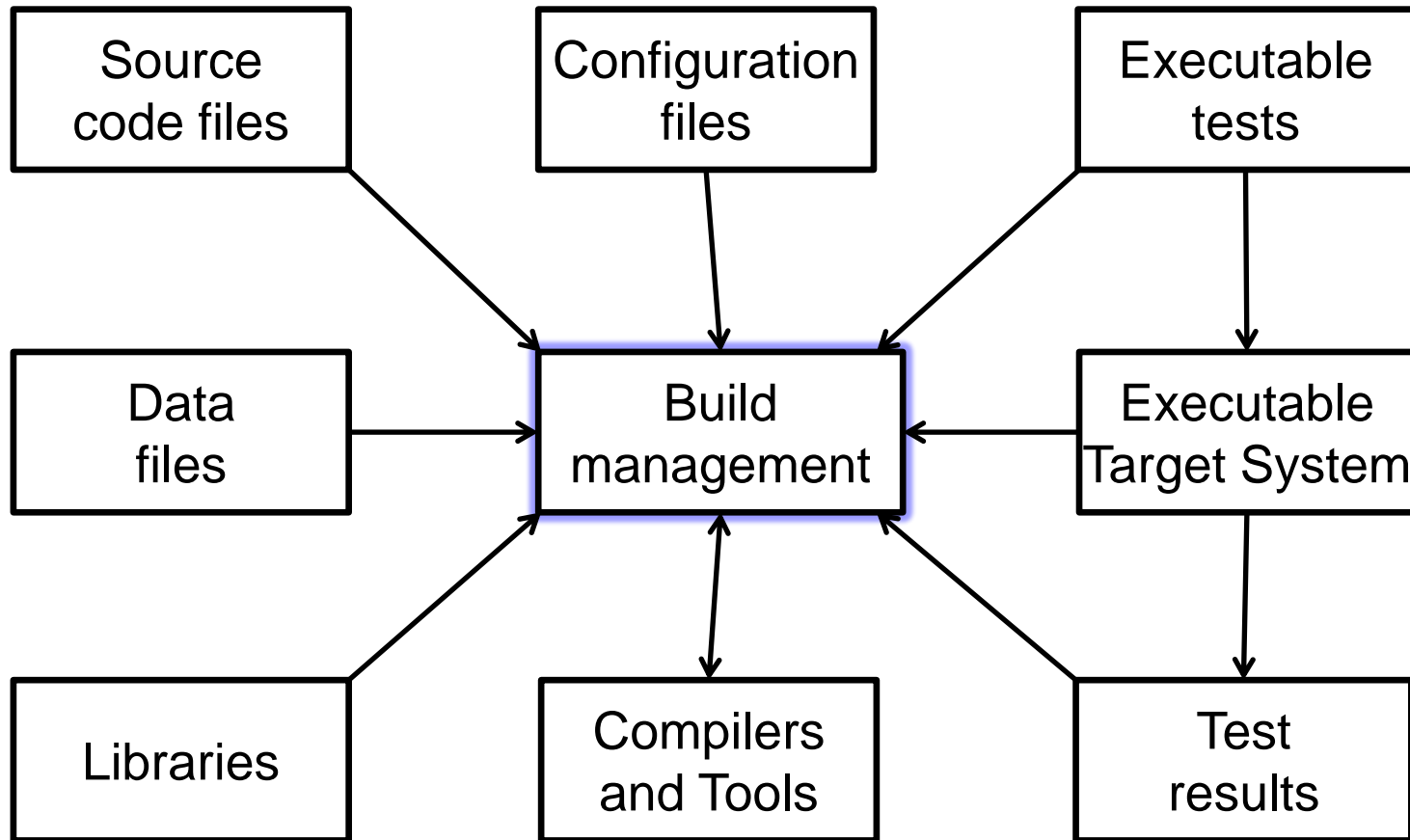
The early days: Makefile

```
CC          = gcc
CFLAGS      = -O
DEST        = ${HOME}/bin
EXTHDRS     = /usr/include/ctype.h /usr/include/stdio.h
HDRS        = tree.h
LDFLAGS     =
LIBS        =
LINKER      = gcc
OBJS        = tree.o treealloc.o treemain.o treeprint.o treeword.o
PROGRAM     = a.out
SRCS        = tree.c treealloc.c treemain.c treeprint.c treeword.c
all:        $(PROGRAM)
$(PROGRAM): $(OBJS) $(LIBS)
            $(LINKER) $(LDFLAGS) $(OBJS) $(LIBS) -lm -o $(PROGRAM)
clean::      rm -f $(OBJS)
install:     $(PROGRAM)
            install -s $(PROGRAM) $(DEST)
tree.o: tree.h /usr/include/stdio.h
treealloc.o: tree.h /usr/include/stdio.h
treemain.o: tree.h /usr/include/stdio.h
treeprint.o: tree.h /usr/include/stdio.h
treeword.o: tree.h /usr/include/stdio.h /usr/include/ctype.h
```

Functions of build management

- Build script generation
 - Makefile (make) runs the commands
- Integration to version management
 - Already in the early days "make" has SCCS integration
- Minimal recompilation
 - "Make" check time stamps of files
- Executable system generation
- Test automation
- Reporting
- Document generation

Figure 25.11 in Sommerville



Another tool Ant (<http://ant.apache.org/>)

- Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other.
- The main known usage of Ant is the build of Java applications. Ant supplies a number of built-in tasks allowing to compile, assemble, test and run Java applications.
- Ant can also be used effectively to build non Java applications, for instance C or C++ applications.
- More generally, Ant can be used to pilot any type of process which can be described in terms of targets and tasks.

Ant vs make

Ant says:

- Ant is different. Instead of a model where it is extended with shell-based commands, Ant is extended using Java classes.
- Instead of writing shell commands, the configuration files are XML-based, calling out a target tree where various tasks get executed.
- Each task is run by an object that implements a particular Task interface.

But one reason is that Ant knows Java and can optimize use of Java compiler

You can find several debates from the Internet

And there are other alternatives, too. E.g. *maven* or *rake*.

Continuous integration

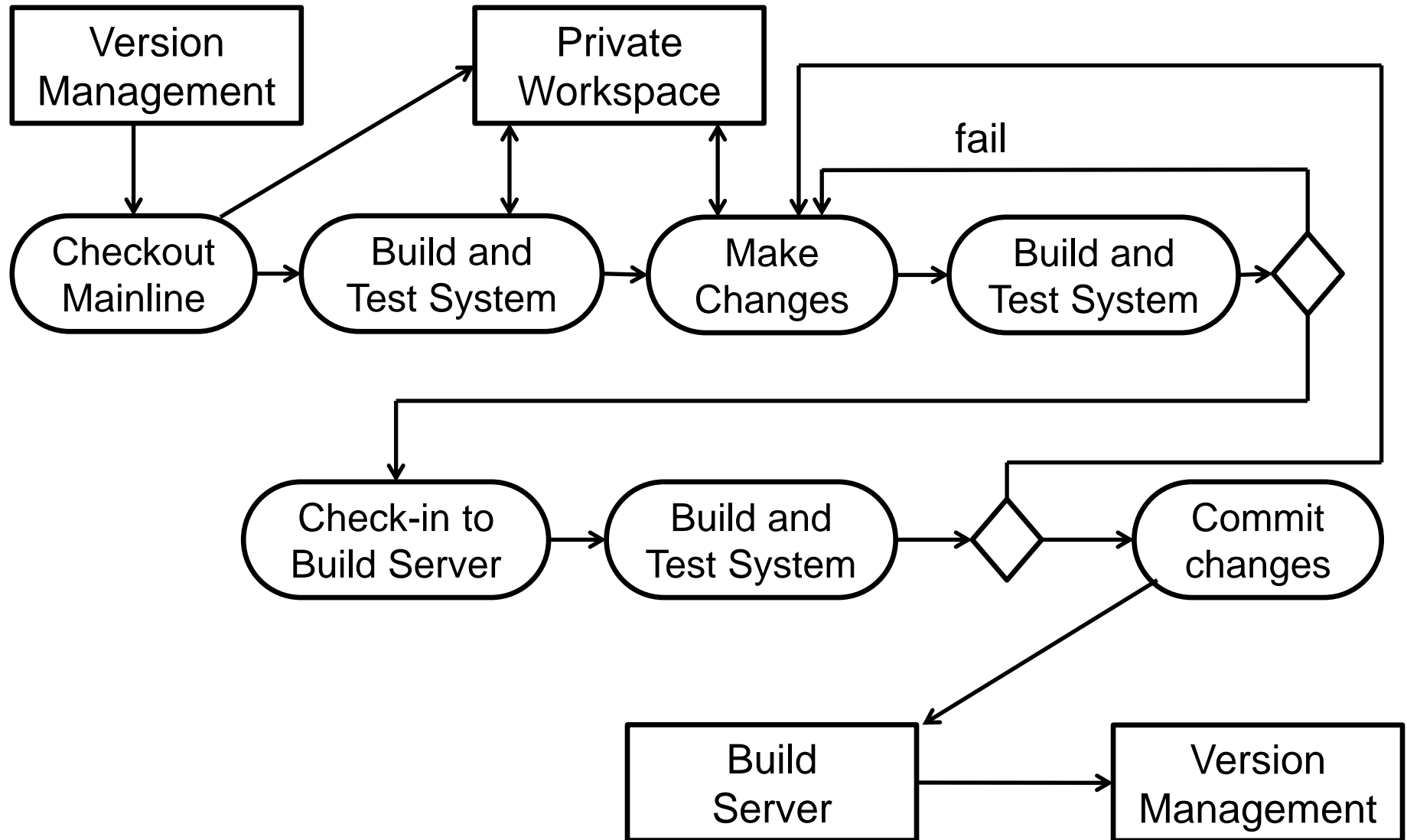
(<http://www.martinfowler.com/articles/continuousIntegration.html>)

- Suggested by many agile methods, for example XP.
- The team integrates latest changes to complete system frequently – even several times a day.
- Often combined with test-driven development
- Benefits
 - Reduced risk (and earlier discovery)
 - Helps in getting rid of bugs
 - Avoid chaos at the end (integration hell)
 - Changes behavior of programmers, more carefully written and simpler code

Practices of continuous integration

- Practices of Continuous Integration
- Maintain a Single Source Repository.
- Automate the Build
- Make Your Build Self-Testing
- Everyone Commits To the Mainline Every Day
- Every Commit Should Build the Mainline on an Integration Machine
- Keep the Build Fast
- Test in a Clone of the Production Environment
- Make it Easy for Anyone to Get the Latest Executable
- Everyone can see what's happening
- Automate Deployment

Figure 25.12 in Sommerville



An example tool for CI: Jenkins

- Very commonly used open source tool
- Run build (e.g. Ant) and automatic test (e.g. Junit) scripts
- Integrates with very many version management systems
- Can be triggered automatically by version management
- Can be triggered by sending email
- Build running in batch mode – users can see the status
- Let look at tutorial at:
<http://www.vogella.com/tutorials/Jenkins/article.html>

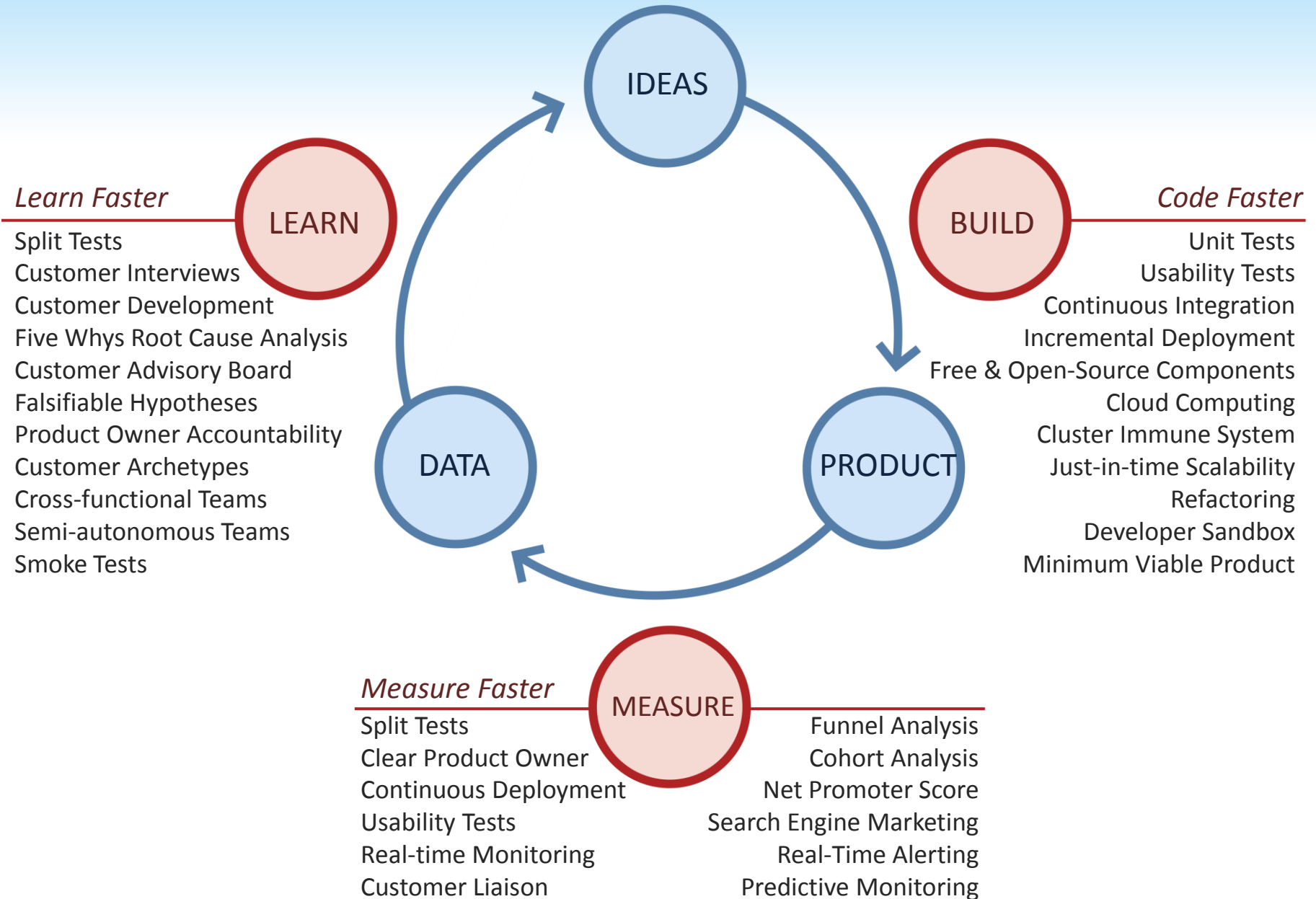
**31.03.2014 WE RUN OUT OF
TIME AND STOPPED HERE.
FOLLOWING SLIDES WILL BE
COVERED 7.4**

Release management

About release management

- Releases go to external customers/users the vendor should be able to answer questions on that particular release.
- Often include major and minor releases
 - Powerpoint 14.0.7116.5000 (32-bit)
 - Thunderbird 17.0.11
- Customer-specific and mass-market SW impose different challenges
- When problem occurs the HW configuration should be available
- Full traceability is expected
- Releases should be well tested, well documented, ...
- Installation/deployment need to be planned
- Updates need to be planned
 - Technical, commercial

But on the other hand, sometimes ...



Continuous delivery/deployment; A/B testing

- Sometimes it is important to get fast feedback from market
 - Lean Startup
- Also part of DevOps
- Used for development of customer software and Internet-services
- A/B testing (split testing):
 - Randomly give different users different versions of the system and systematically compare.

One claim

(<http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment>)

CONTINUOUS DELIVERY



CONTINUOUS DEPLOYMENT



Week	Lecture	Exercise
10.3	Quality in general; Quality management systems	Patterns
17.3	Dependable and safety-critical systems	ISO9001
24.3	Work planning; effort estimation	Code inspections
31.3	Version and configuration management	<i>Effort estimation</i>
7.4	Role of software architecture; product families; software evolution	?
14.4	Specifics of some domains, e.g. web system and/or embedded and real time systems	Break?
21.4	Easter	Break?
28.4	Software business, software start-ups	?
5.5	Last lecture; summary; recap for exam	?