

Configuration Management

7.5 ECTS



1: Introduction and motivation

Lars Bendix

*Department of Computer Science
Lund Institute of Technology
Sweden*

<https://cs.lth.se/EDAN10/>

© Lars Bendix – Lund University, Sweden 7.5-ECTS Configuration Management, Lecture 1 – ver. 1.03

Agenda

Part I: Motivation for SCM



Short break – 5 min

Part II: Pedagogic setup for this course

Short break – 5 min

Part III: Overview of the SCM empire



© Lars Bendix – Lund University, Sweden 7.5-ECTS Configuration Management, Lecture 1 – ver. 1.03

Part I

MOTIVATION

© Lars Bendix – Lund University, Sweden 7.5-ECTS Configuration Management, Lecture 1 – ver. 1.03

SCM examples



Canadian Telecom:

- 8 MLOC
- new release every six months
- 300-500 KLOC changed
- production time 8-11 months
- 1000+ programmers
- x variants

AT&T - Bell:

- software older than 20-25 years
- no documentation
- no source code

© Lars Bendix – Lund University, Sweden 7.5-ECTS Configuration Management, Lecture 1 – ver. 1.03






SCM reality

Mobile phones:

- ?? MLOC
- ?? people/teams (distributed)
- short – very short – product life time
- several customers in same country
- (same) customer in many (all) countries
- high- to low-end product range
- first-to-market wins
- in-house => out-house/mixed development



© Lars Bendix – Lund University, Sweden 7.5-ECTS Configuration Management, Lecture 1 – ver. 1.03

Other CM examples

- DoD torpedos
- Kasper's VW microbus
- Carlo's lemon marmalade
- Ole's crane hooks
- car engine tester software
- Concorde accident
- Copenhagen Metro
- Citroën C3 fires
- Danish Railways



© Lars Bendix – Lund University, Sweden 7.5-ECTS Configuration Management, Lecture 1 – ver. 1.03

Signs of a problem

- cannot meet deadlines
- cannot release multiple fixes
- do not know what went into a release
- two developers fix the same bug
- bug-fixes disappear
- do not know what has been tested
- have no visibility into work status

© Lars Bendix – Lund University, Sweden 7.5-ECTS Configuration Management, Lecture 1 – ver. 1.03

Problems



Identification:
You should be able to identify the single components and configurations.

- *This worked yesterday, what has happened?*
- Do we have the latest version?
- I have already fixed this problem. Why is it still there?

Change tracking:
Helps in tracking which changes have been made to which modules and by whom, when and why.

- Has this problem been fixed?
- *Who is responsible for this change?*
- This change looks obvious - has it been tried before?

© Lars Bendix – Lund University, Sweden 7.5-ECTS Configuration Management, Lecture 1 – ver. 1.03

Problems

Software production:
Construction of a program involves pre-processing, compilation, linking, etc.

- I just corrected this, has something not been compiled?
- *How was this binary produced?*
- Did we do all the necessary steps in the right order?

Concurrent updating:
The system should offer possibilities for concurrent changes to components.

- *Why did my changes disappear?*
- How do I get these changes into my version?
- Are our changes in conflict with each other?

© Lars Bendix – Lund University, Sweden 7.5-ECTS Configuration Management, Lecture 1 – ver. 1.03



Excuses for not using CM?



- CM only applies to source code
- CM is not appropriate during development because we use rapid prototyping
- It's not that big a project
- You can't stop people from making a quick patch
- We lower our cost by using only minimum-wage persons on our CM staff because CM does not require much skill



Common heritage



- re-use things
- sharing things
- memory/history
- collaboration
- communication
- co-ordination



How does a programmer spend his time?



- 50% interacting with other team members
- 30% working alone
- 20% non-productive activities



Wayne Babich



Sometimes it is embarrassing to be a computer programmer. What other profession has such a remarkable rate of schedule and cost overrun and outright failure? [...]

Our failures are not of the individual contributors; most of us design, code and debug adequately or even well. Rather, the failure is one of coordination. Somehow we lack the ability to take 20 or 30 good programmers and meld them into a consistently productive team.

Wayne A. Babich, 1986



Part II



PEDAGOGICS



“Logical” schedule – themes



• read flyer + browse literature
Overview *lecture*
• study/read literature and notes

• read the flyer – watch the video
• read/study literature
In-depth *seminar* (discussions – in groups)
• organize notes

Exercises (groups)
• organize notes

• read lab description
Lab (groups)
• organize notes/write report



Groups



Groups (3-4 people):

- Exercises
- Tool labs
- Opposition on papers
- Synopsis and project
- Exam
- (Reading literature)
- (Discussing literature)



Lectures/Seminars



Some of them I do live (with possibility for interaction):

- Introductions/overviews
- Do interrupt – do ask questions

Others I have done on video (**seminars** – flipped classroom):

- Going more in depth with a theme/topic
- Based on a paper/chapter
- 10-15 min. introduction/overview video
- Look&listen **before** you read the paper
- **Opponent groups** find questions to discuss
- **Everyone** discuss the questions in groups
- **We** discuss the results in plenary



Exercises



Always on Wednesdays:

- In groups
- Groups can be temporarily merged
- Discussions about “theme of the week”
- Concrete result (one “slide” – pen&plastic)
- **Do use the concepts and principles as much as possible!**



Time-boxing



Most things on the course are “time-boxed”:

- I hope that you spend the number of hours allocated
- I do **not** want you to spend more time
- you do *as good as you can* **within the time slot**
- if you have problems with the number of hours – tell me



Tool labs



Always on Fridays:

- As a group
- In pairs for 4-person groups
- Can be done outside of official hours? But not alone!
- Concrete results (small reports)
- **Progressive “independence”:**
 - CVS: nursed
 - Perforce: helped
 - git: off the deep end
- not really *tool* labs – concepts and principles labs
- different scenarios (“test cases”)
- *exploration* and *evaluation* of version control tools



Project



Running throughout the course:

- In groups
- Digging deeper into special interests
- Anything related to CM
- Some “standard” proposals
- Synopsis (December 16):
 - Sketch of: problem domain, method and “results”
 - Feedback provided
- Report (January 5?)
 - “Used” for the oral exam – but *not* graded
- I do **not** expect any work between 24/12 and 1/1!



Student peer review



Students will review students’ work:

- a possibility to see another solution
- an opportunity to reflect on learning
- (extra) feedback on own solution

Occasions:

- exercises (informal)
- lab reports (for the Perforce lab)
- paper review (same paper)
- synopsis/report (not implemented yet)



Examination



- Oral
- In groups
- “Friendly” discussion
- Overview – everything (“pass/fail”)
- In-depth/details – selected parts (“grade”)
 - motivated answers (why)
 - advantages and drawbacks
 - alternative possibilities
- Individual grades
- In the exam weeks ((18/12) + 7-17/1) – flexible?



Learning objectives



- Formal/overall (course plan):
 - Purpose and goal
 - Knowledge and understanding
 - Skills and abilities
 - Judgement and approach
- Detailed/specific:
 - webpage(s) for theme
 - lecture slides for theme (slide 3+/-)



ChatGPT/Copilot



Generative AI – but:

- do not **cheat**
- academic **dishonesty**
- *optional* exercises:
 - critical sense
 - prompt engineering
 - “constructive” use



Part III



QUICK OVERVIEW



What is SCM?



Software Configuration Management:

is the discipline of organising, controlling and managing the development and evolution of software systems. (IEEE, ISO,...)

The goal is to maximize [developer] productivity by minimizing [coordination] mistakes. (Babich)



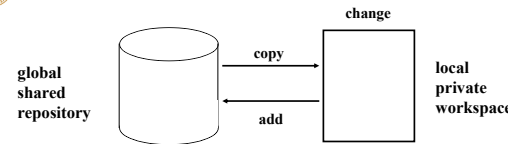
Outline of the course



- Introduction
 - Construction site
 - The study
 - The library
 - Traditional CM
 - CM++
 - Wrap up + industry
- Team
Company
How to
- synopsis
done
in
groups,
handed in
and used for
- Written project
 - Exam (oral in groups)



Model



Principle: components are **immutable**

- Babich:
- shared data
 - double maintenance
 - simultaneous update



Co-ordination



Working in isolation:

- local dynamicity
- global stability
- problems:
 - double maintenance

Working in group:

- global dynamicity
- problems:
 - shared data
 - simultaneous update



Identification



The buildfile for a system contains the description of:

- which components go into the system
- how they were derived and put together

Therefore it is important to associate the actual buildfile (Software Bill-of-Materials - SBOM) to the generated system.

Identification should work both ways (traceability).



Baselines



A baseline is a reference configuration which is stable (over a certain period of time) relative to the work of the single programmers.

Ensures stability in relation to "the outside world".

Allows safe dynamicity in the single programmers "private" work.

Acts as a communication channel for changes.

Spiderman vs. Rain-man



Configuration Management – Overview



Configuration Identification

Configuration Change Control

Configuration Status Accounting

Configuration Audit

....



Configuration Identification



Definition: the selection, designation and description of **configuration items**.

Purpose/goal: to capture, preserve and *make available* all the *important things* in a project – and to make sure that we have *unique identification* for these.

Concepts: configuration item (CI), CMDB, traceability, Software Bill-of-Materials (SBOM)



Configuration Status Accounting



Definition: the **recording and reporting of information** needed to manage and work on a project.

Purpose/goal: to allow people to easily get all the *information* that they need to carry out their work in an informed way.

Concepts: queries on the CMDB, answering questions



Configuration Change Control



Definition: the proposal, evaluation, coordination, approval or disapproval, and implementation of approved **changes to baselined CIs**.

Purpose/goal: to ensure that proposed changes are classified and *evaluated* and that approved changes are implemented, documented and verified.

Concepts: Change Request (CR), change process, Change Control Board (CCB)



Configuration Audit



Definition: an independent evaluation of a (changed) CI to **ascertain compliance** to specifications, standards, contractual agreements and other criteria.

Purpose/goal:

- to verify that the CI *matches the description* in the specification and documentation.
- to ensure that *work has been performed* in the correct way, that is, in conformance with the development standards and guidelines.

Concepts: baseline, quality gate



Different SCM-roles



- users of SCM
 - developers (ordinary/"build meisters")
 - project leaders
 - testers/designers/RE/...
 - customers
 - ...
- designers of SCM
 - processes
 - tools
- technical SCM personnel



Does it pay?



Norwegian experiment:

- defect reports down by 36 % (more defects found internally)
- defect fix time down by 6 %
- maintenance resource usage down by 33 %
- development resource usage up by 22 %
- less effort spent on: system build, deliveries and maintenance
- pay back time - 7 months



What is more important?



- tools
- process
- people



Why should CM be automated?



- simple tasks
- repetitive
- accuracy
- precision
- discipline
- saves space
- saves time