T–76.3601 — Introduction to Software Engineering

Software Engineering Practice
Software Configuration Management

http://www.soberit.hut.fi/T-76.3601/

Casper Lassenius
Casper.Lassenius@tkk.fi

Software Engineering Practice

What is "Practice"?

- Practice is a **broad array of concepts, principles, methods and tools** that you must consider as software is planned and developed
- It **represents the details**—the technical considerations and how to's—that are below the surface of the software process—the things that you'll need to actually build high-quality computer software.

WARNING!

- Most things said during this lecture will sound extremely simple and self-evident
- This is because they basically are
- However, most of the good practices are **not practiced** in many software development projects!

The Essence

- George Polya, in a book written in 1945(!) describes the essence of software engineering practice...
  - **Understand the problem** (communication and analysis)
  - **Plan a solution** (modeling and software design)
  - **Carry out the plan** (code generation)
  - **Examine the result** for accuracy (testing and quality assurance)
- At its core, good practice is **common-sense problem solving**

Core Software Engineering Principles

- Provide value to the customer and the user
- KIS—keep it simple!
- Maintain the product and project *vision*
- What you produce, others will consume
- Be open to the future
- Plan ahead for reuse
- Think!

General Practices for the Basic Software Engineering Activities

- Planning
- Modeling
- Construction
- Deployment
- Communication

Planning Practices

- Principles
  - Understand the project scope
  - Involve the customer and other stakeholders
  - Recognize that planning is iterative
  - Estimate based upon what you know
  - Consider risk
  - Be realistic
  - Adjust granularity as you plan
  - Define how to achieve quality
  - Define how to accomodate changes
  - Track your plan

Planning Practices

- Ask Boehm's questions:
  - **Why** is the system developed?
  - **What** will be done?
  - **When** will it be accomplished?
  - **Who** is responsible?
  - **Where** are they located? (organizationally)
  - **How** will the job be done (technically and managerially)
  - **How much** of each resource is needed?

Modeling Practices

- We create models to gain a better understanding of the actual entity to be built
- **Analysis models** represent the **customer requirements** by depicting the software in three different domains: the *information* domain, the *functional* domain, and the *behavioral* domain
- **Design models** represent characteristics of the software that help **practitioners construct** it effectively: the *architecture*, the *user interface*, and *component*-level detail

Analysis Modeling Practices

- Principles
  - Represent the information domain
  - Represent the software functions
  - Represent software behavior
  - Partition the representations
  - Move from essence towards implementation

Design Modeling Practices

- Principles
  - Design must be traceable to the analysis model
  - Always consider architecture
  - Focus on the design of data
  - Interfaces (user, internal) must be designed
  - Components should exhibit functional independence
  - Components should be loosely coupled
  - Design representations should be easily understood
  - The design model should be developed interatively

Construction Practices

- Preparation principles (before you start coding!)
  - Understand the problem
  - Understand basic design principles and concepts
  - Pick a programming language that meets the needs of the software and environment
  - Select a programming environment that provides tools that make your work easier
  - Create a set of unit tests that will be applied once the component you code is completed (if you use the *test-first approach*)

Construction Practices

- Coding principles
  - Select data structures that meet the needs of your design
  - Understand the architecture and create compliant interfaces
  - Keep conditional logic as simple as possible
  - Create nested loops in a way that makes them easily testable
  - Select meaningful variable names and follow local coding conventions
  - Write self-documenting code
  - Create a visual layout (indentation, blank lines etc.) that aids understanding

Construction Practices

- Validation principles
  - Use code walkthroughs
  - Perform unit tests and correct errors
  - Refactor the code
- Testing principles
  - All test should be traceable to requirements
  - Tests should be planned (!?)
  - The pareto principle applies to testing
  - Testing begins “in the small” and moves toward “in the large”
  - Exhaustive testing is not possible
    - Testing can only show the presence of bugs—not their absence!

Deployment Practices

- Principles
  - Manage customer expectations for each increment
  - A complete delivery package should be assembled and tested
  - A support regime should be established
  - Instructional materials must be provided to end-users
  - Buggy software should be fixed first, delivered later

Communication Practices

- Principles
  - Listen
  - Prepare
  - Facilitate
  - Face-to-face is best
  - Take notes and document decisions
  - Collaborate with the customer
  - Stay focused
  - Draw pictures when things are unclear
  - Move on...
  - Negotiation works best when both parties win!

Software Configuration Management

The "First Law"

- No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle.

Bersoff, et el. 1980

What are these Changes?

changes in
business requirements

changes in
technical requirements

changes in
user requirements

other
documents

software models

Project
Plan

Test

code

data

Three Basic Problems

- **Double maintenance** problem
  - Multiple copies of a file on different computers. All must be updated.
- **Shared data** problem
  - Single copy of file on server. Only one person can work on it at a time.
- **Simultaneous update** problem
  - Master copy of file on server, developers have work copies. Changes might get overwritten.

Software Configuration Management

- Version management
  - files and documentsn (not only source files!)
- Build management
  - building the executable from the right sources
  - “manufacturing process”
- Change management
  - what changes are implemented, and how
- Status accounting
  - recording and reporting
  - visibility
- Release management
  - what is delivered to the customer

Why Version Management?

- During the lifetime of a system, files change
- Old versions of files can be needed
  - troubleshooting
  - reverting to a tested and working version
- Different versions might be incompatible

Configuration Item

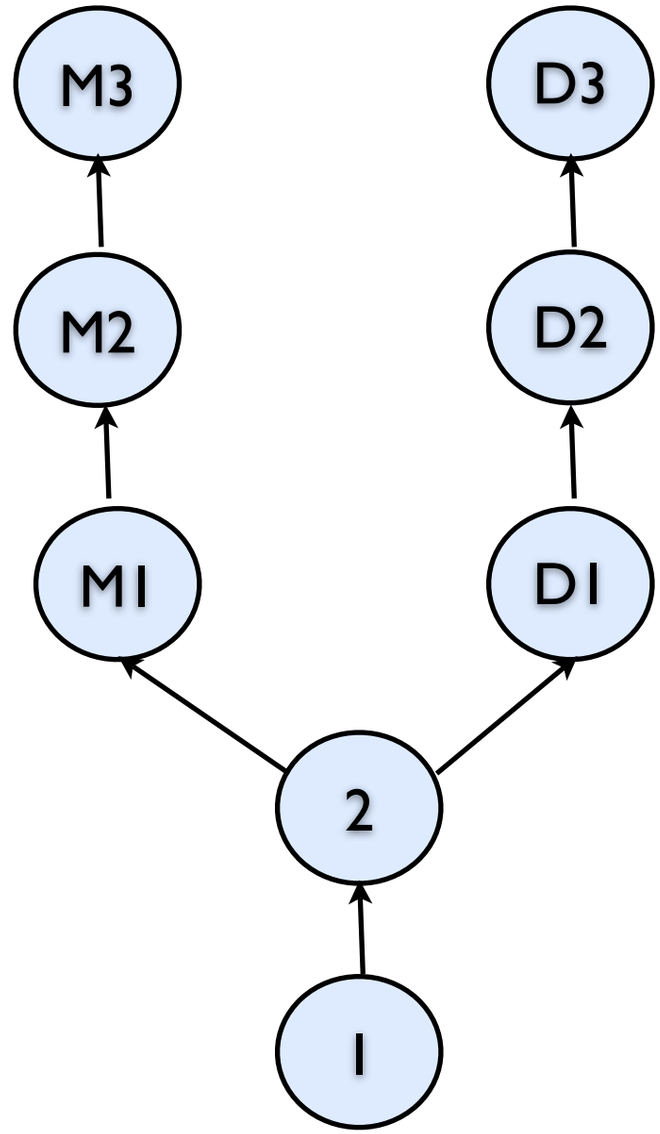
- Something that is versioned
- An atomic unit from the SCM point of view
- Examples
  - files
  - documents
  - components

Version

- An instance of a configuration item that differs from other instances of the same configuration item

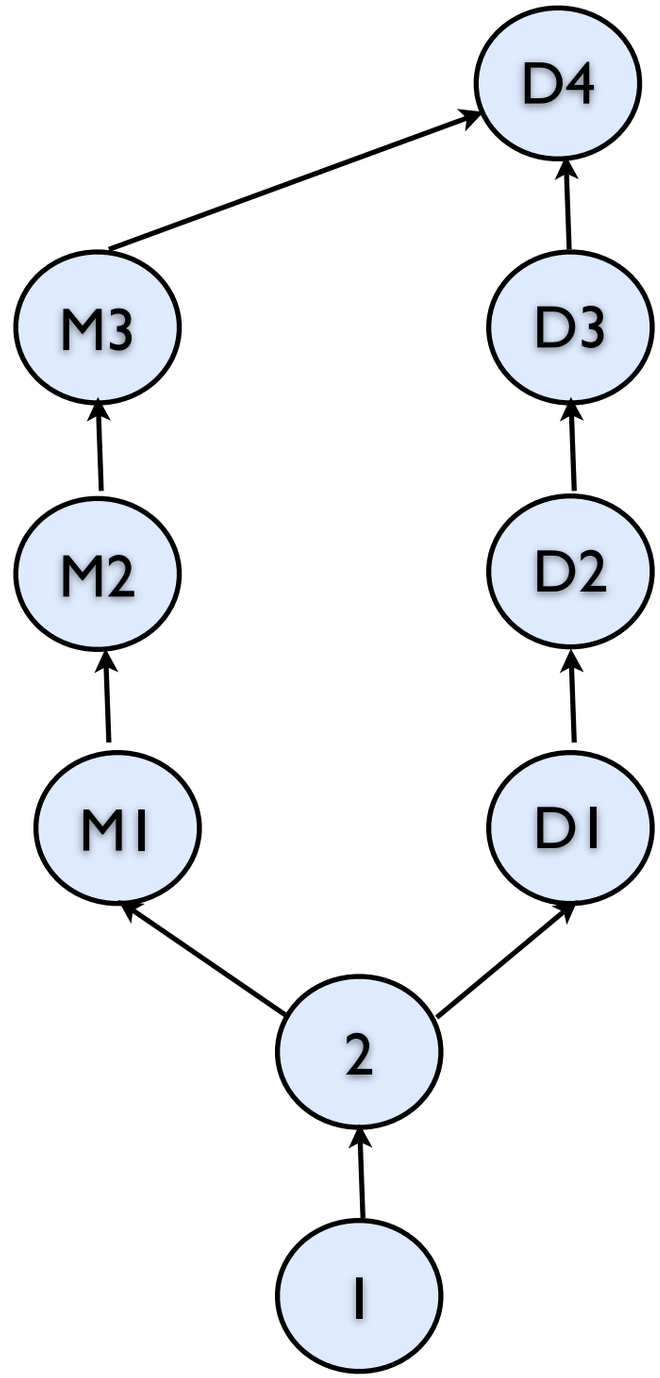
Versions: Revisions and Variants

- Variant
  - an alternative to another version of the same CI
    - typically used, e.g. for customization
- Revision
  - replaces another (previous) version of the same CI



Merging Variants

- Textual
- Syntactic
- Semantic
- Version management systems or IDEs typically contain tools helping with merging (e.g. diff)



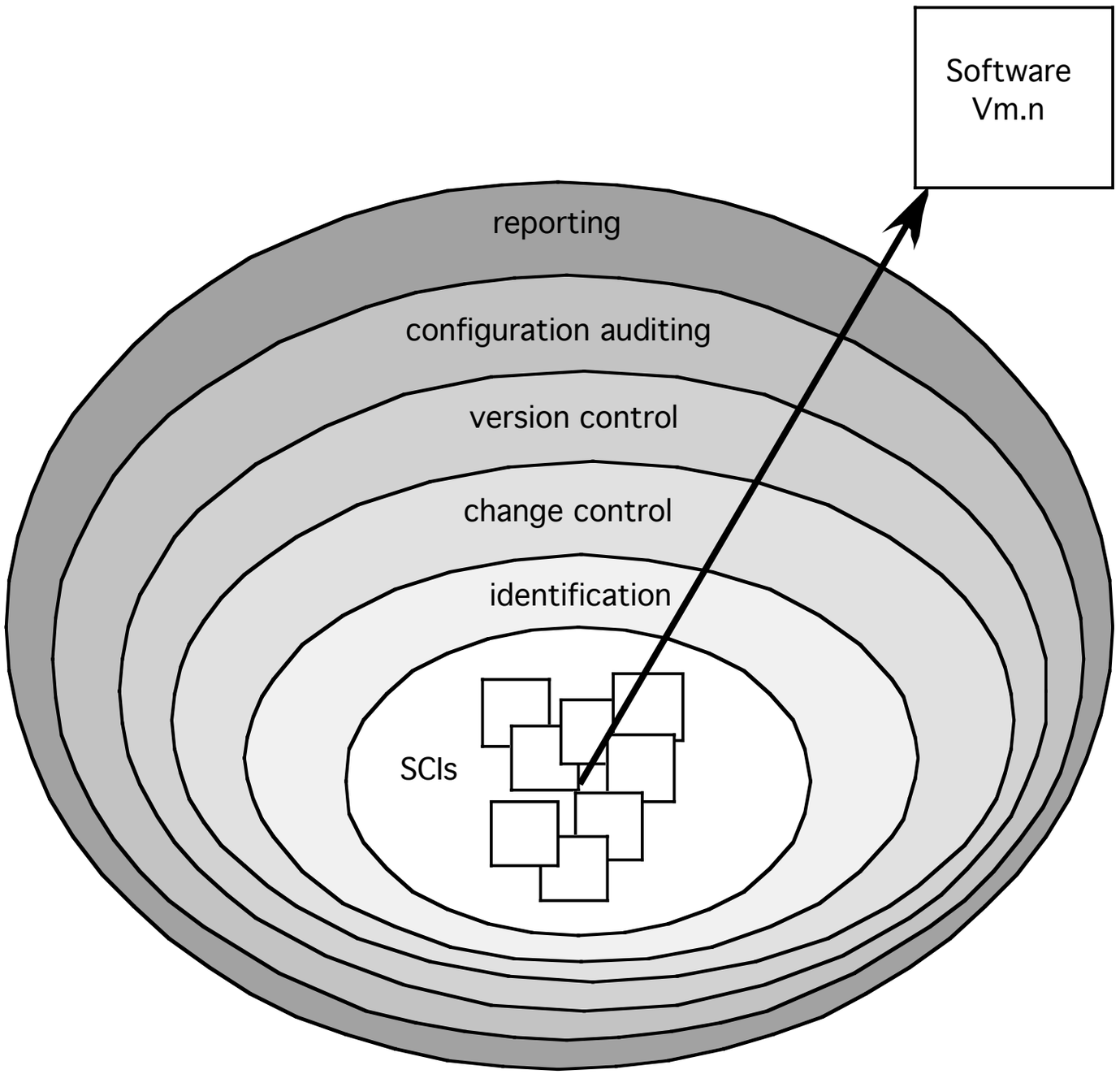
Configurations

- Configuration
  - a set of logically connected configuration items
  - a collection of CI versions (max 1 version/CI)
- Work set
  - current configuration
- Baseline
  - a permanent configuration created for a purpose (testing, release)
  - possible to return to later
  - implemented, e.g. by tagging CIs
- Release
  - permanent configuration that is delivered to the customer
  - subset of deliverable structure baseline

Baseline

- The IEEE Std 610.12–1990 defines a baseline as:
  - *A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.*
- a baseline is a milestone in the development of software that is marked by the delivery of one or more software configuration items and the approval of these SCIs that is obtained through a formal technical review

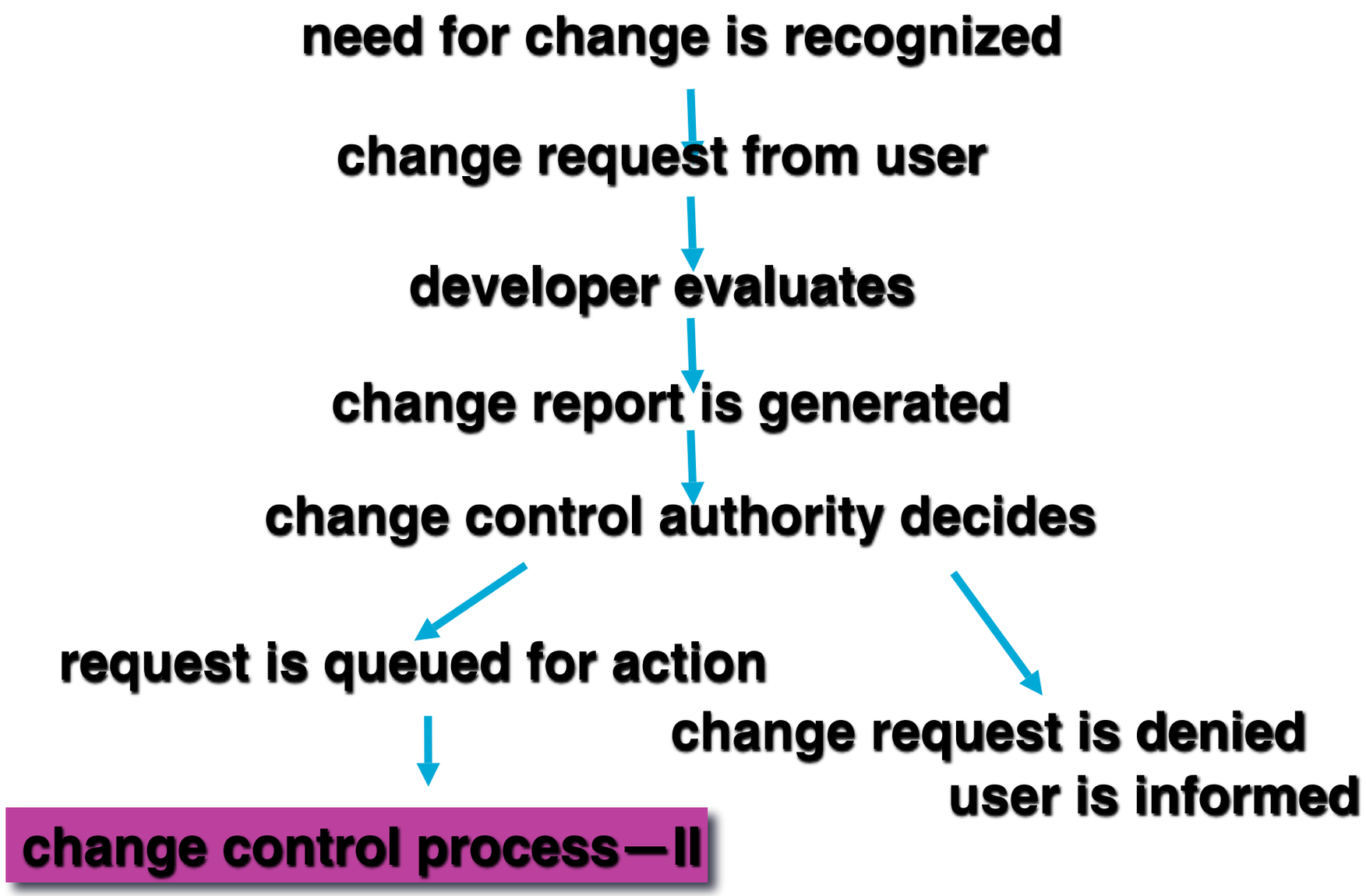
The SCM Process



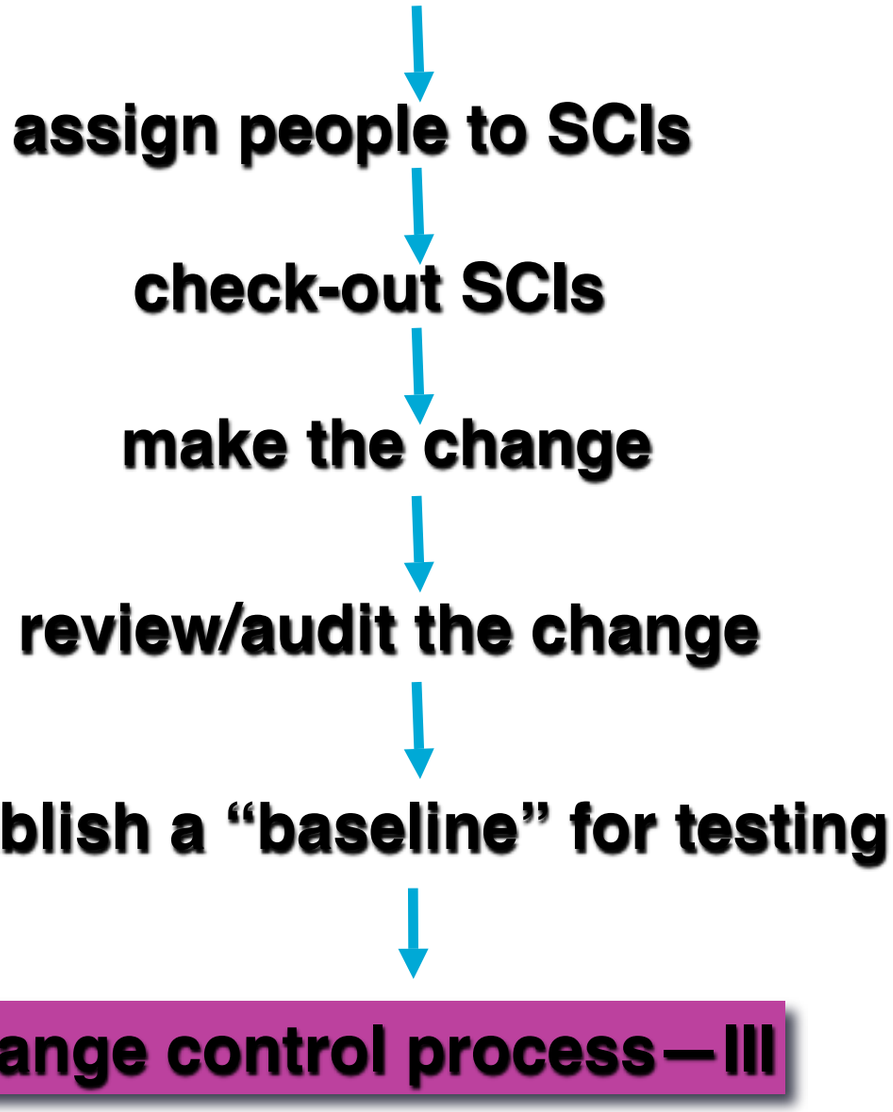
Change Management

- Changes to approved items must be controlled
- Change management is a procedure by which a change to an item is requested, evaluated, approved or rejected, scheduled and tracked
- The purpose is to avoid uncontrolled changes to software in order to
  - minimize quality problems
  - control schedule and effort
  - avoid gold plating and feature creep
- Uses a well-defined configuration as a starting point

CM Process I/III



CM Process II/III



CM Process III/III

perform SQA and testing activities

check-in the changed SCIs

promote SCI for inclusion in next release

rebuild appropriate version

review/audit the change

include all changes in release

Change Request (CR) Content

- Submitter (customer, tester)
  - description of problem
  - who found
  - version
  - error messages/symptoms
- Analyst
  - cause
  - affected components/ subsystems
  - effort estimation
- Manager
  - decision fix/postpone/ drop
  - target release
  - responsible developer
- Developer
  - actions taken
  - affected files
  - release notes
  - actual effort
- Tester
  - retesting
  - regression testing

Correct Level of CM

- Depends on many factors, e.g.
  - size of team
  - criticality of software
  - risk
  - scope of change
  - criticality of change
  - release process
  - ...

Build Management

- The manufacturing process for software
- Selecting the right sources for a build
- Performing the right steps
- Compiling only the necessary parts of the system depending on what has been changed
- Typically automated
  - make, ant, ...

Build Cycle

- Heartbeat of a project
  - build and smoke-test e.g. daily
  - show concrete progress
- Frequent builds and automated tests
  - minimize integration risk
  - reduce quality risk
  - easier debugging
  - improve morale

Status Accounting

- Making status of the development visible
  - Is the developer finished with a file?
  - Is a file reviewed?
  - Is a CR rejected?
- Status reflects the lifecycle of a CR or bug report

Release Management

- About managing what gets released, under what conditions, and to whom
- In product environments, *release cycles* are common
  - E.g. major release every 1-3 years, yearly minor updates, unscheduled (on a need basis) patch releases (service packs)

Release Classifications

- Alpha release
- Beta release
- Final release
- Update and upgrade release
- Patches and emergency fixes

Distribution of Releases

- Physical media
  - CD ROM, DVD, tapes, chips
- Networked media
  - Internet
  - NFS
- Software as a service
  - Not installed on client computers, but site upgraded

Questions?