Software Configuration Management

Institut Teknologi Del
Jl. Sisingamangaraja
Sitoluama, Laguboti 22381
Toba – SUMUT
http://www.del.ac.id

Lecturer : AMS
- KUPEL –
Sem-8 T.A 21/22

Software Configuration Management

- Definition:
  - the art of identifying, organizing, and controlling modifications to the software being built by a programming team.
- Goal
  - maximize productivity by minimizing mistakes (Babich, 1986).
- It is an umbrella activity that is applied throughout the software process.
- Because change can occur at any time, SCM activities are developed to:
  - identify change
  - control change
  - ensure that change is being properly implemented
  - report changes to others who may have an interest.

Software configuration management vs software support

- software support and software configuration management are different
- Software support
  - a set of software engineering activities that occur after software has been delivered to the customer and put into operation.
- Software configuration management
  - a set of tracking and control activities that begin when a software engineering project begins and terminate only when the software is taken out of operation.

Source of change

- There are four fundamental sources of change:
  - New business or market conditions dictate changes in product requirements or business rules.
  - New customer needs demand modification of data produced by information systems, functionality delivered by products, or services delivered by a computer-based system.
  - Reorganization or business growth/downsizing causes changes in project priorities or software engineering team structure.
  - Budgetary or scheduling constraints cause a redefinition of the system or product.

SCM’s important concepts

- Software configuration management is a set of activities that have been developed to manage change throughout the life cycle of computer software.
- SCM can be viewed as a software quality assurance activity that is applied throughout the software process.
- There are important concepts related to change management in SCM:
  - Software configuration items
  - Baselines

Software configuration items

- The output of the software process can be divided into three broad categories:
  - computer programs (both source level and executable forms)
  - documents that describe the computer programs (targeted at both technical practitioners and users)
  - data (contained within the program or external to it).
- The items that comprise all information produced as part of the software process are collectively called a *software configuration.*
- As the software process progresses, the number of *software configuration items* (SCIs) grows rapidly.

Software configuration items (1)

- We have already defined a software configuration item as information that is created as part of the software engineering process.
- In the extreme, a SCI could be considered to be a single section of a large specification or one test case in a large suite of tests.
- More realistically, an SCI can be: a document, or a entire suite of test cases, or a named program component (e.g., a C++ function or an Java package).

Software configuration items (2)

- SCIs are organized to form *configuration objects* that may be cataloged in the project database with a single name. A configuration object has a name, attributes, and is "connected" to other objects by relationships.
- The relationship can be compositional (single-headed arrow) or interrelationship (double-headed arrows).
- If a change were made to the source code object, the interrelationships enable a software engineer to determine what other objects (and SCIs) might be affected.

jika ingin menanyakan suatu, maka ke database

Baselines (1)

jangan maju sebelum ada baseline

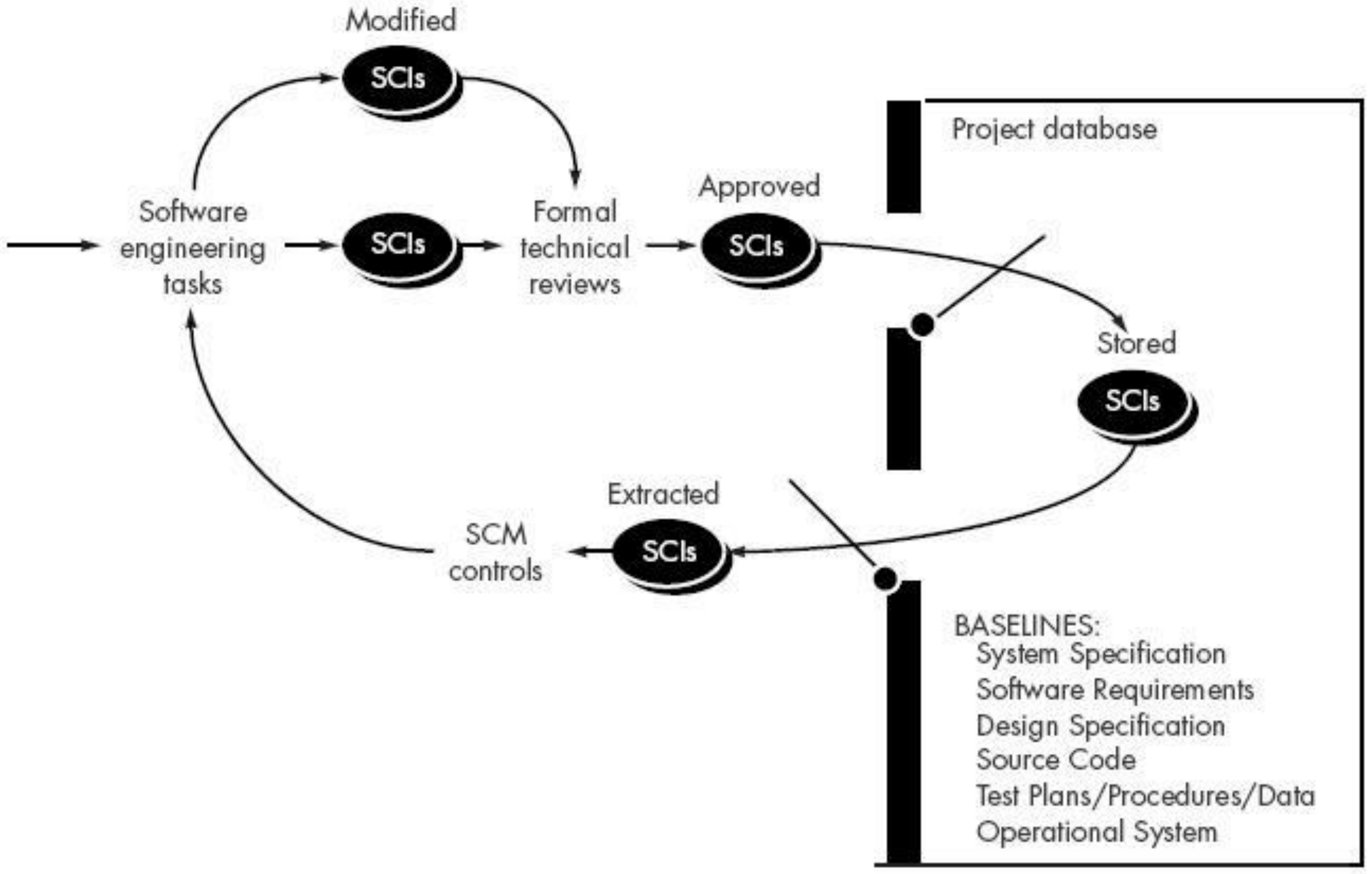
- A *baseline* is a software configuration management concept that helps us to control change without seriously impeding justifiable change.
- The IEEE (IEEE Std. No. 610.12-1990) defines a baseline as:

  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.

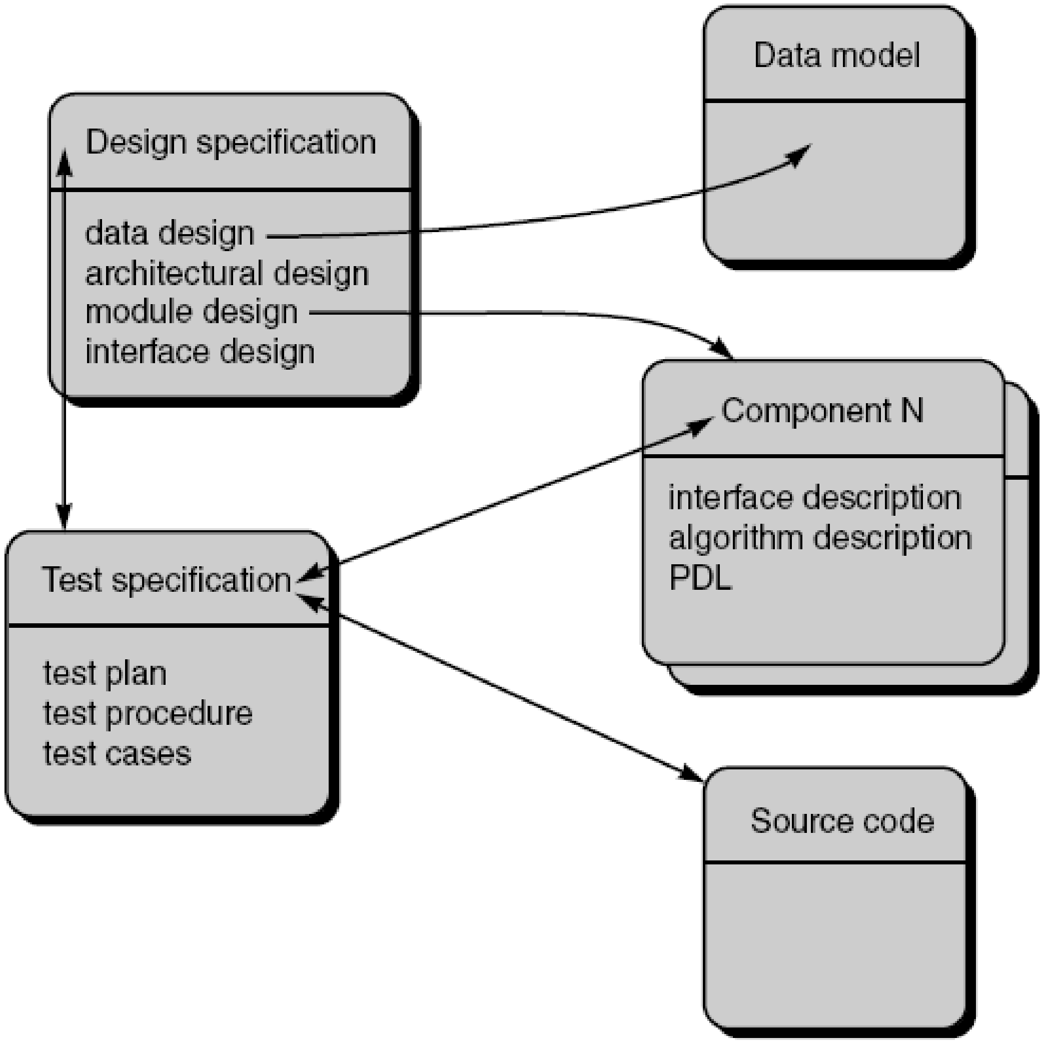
Baselines (2)

- Before a software configuration item becomes a baseline, change may be made quickly and informally.
- However, once a baseline is established, changes can be made, but a specific, formal procedure must be applied to evaluate and verify each change.
- In the context of software engineering, 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 progression of baselines



Configuration objects: example



The SCM process

- Followings are questions to discuss about SCM:
  - How does an organization identify and manage the many existing versions of a program (and its documentation) in a manner that will enable change to be accommodated efficiently?
  - How does an organization control changes before and after software is released to a customer?
  - Who has responsibility for approving and ranking changes?
  - How can we ensure that changes have been made properly?
  - What mechanism is used to appraise others of changes that are made?
- These questions lead us to the definition of five SCM tasks: *identification, version control, change control, configuration auditing,* and *reporting.*

(1) Identification of objects in the software configuration

- To control and manage software configuration items, each must be separately named and then organized using an object-oriented approach.
- The following terms are important in identification of object:
  - Basic objects and aggregate objects
  - Features
  - Relationships
  - Object evolution

Basic objects and aggregate objects

- Two types of objects can be identified (Choi & Scacchi, 1989): *basic objects* and *aggregate objects.*
- A basic object is a "unit of text" that has been created by a software engineer during analysis, design, code, or test. E.g., a basic object might be a section of a requirements specification, a source listing for a component, or a suite of test cases that are used to exercise the code.
- An aggregate object is a collection of basic objects and other aggregate objects. E.g. **Design Specification**. Conceptually, it can be viewed as a named (identified) list of pointers that specify basic objects such as **data model** and **component N**.

sequence diagram maka membentuk sebuah versi

Features

- Each object has a set of distinct features that identify it uniquely: a name, a description, a list of resources, and a "realization."
- The object name is a character string that identifies the object unambiguously.
- The object description is a list of data items that identify the SCI type (e.g., document, program, data) represented by the object, a project identifier, change and/or version information.
- Resources are "entities that are provided, processed, referenced or otherwise required by the object." E.g., data types, specific functions, or even variable names may be considered to be object resources.
- The realization is a pointer to the "unit of text" for a basic object and null for an aggregate object.

Relationship (1)

- Configuration object identification must also consider the relationships that exist between named objects. An object can be identified as <part-of> or compositional of an aggregate object. The relationship <part-of> defines a hierarchy of objects. For example, using the simple notation:

  E-R diagram 1.4 <part-of> data model;

  data model <part-of> design specification;

Relationship (2)

- The relationship can also be interrelationship (cross structural relationship). E.g.

  data model <interrelated> data flow model;

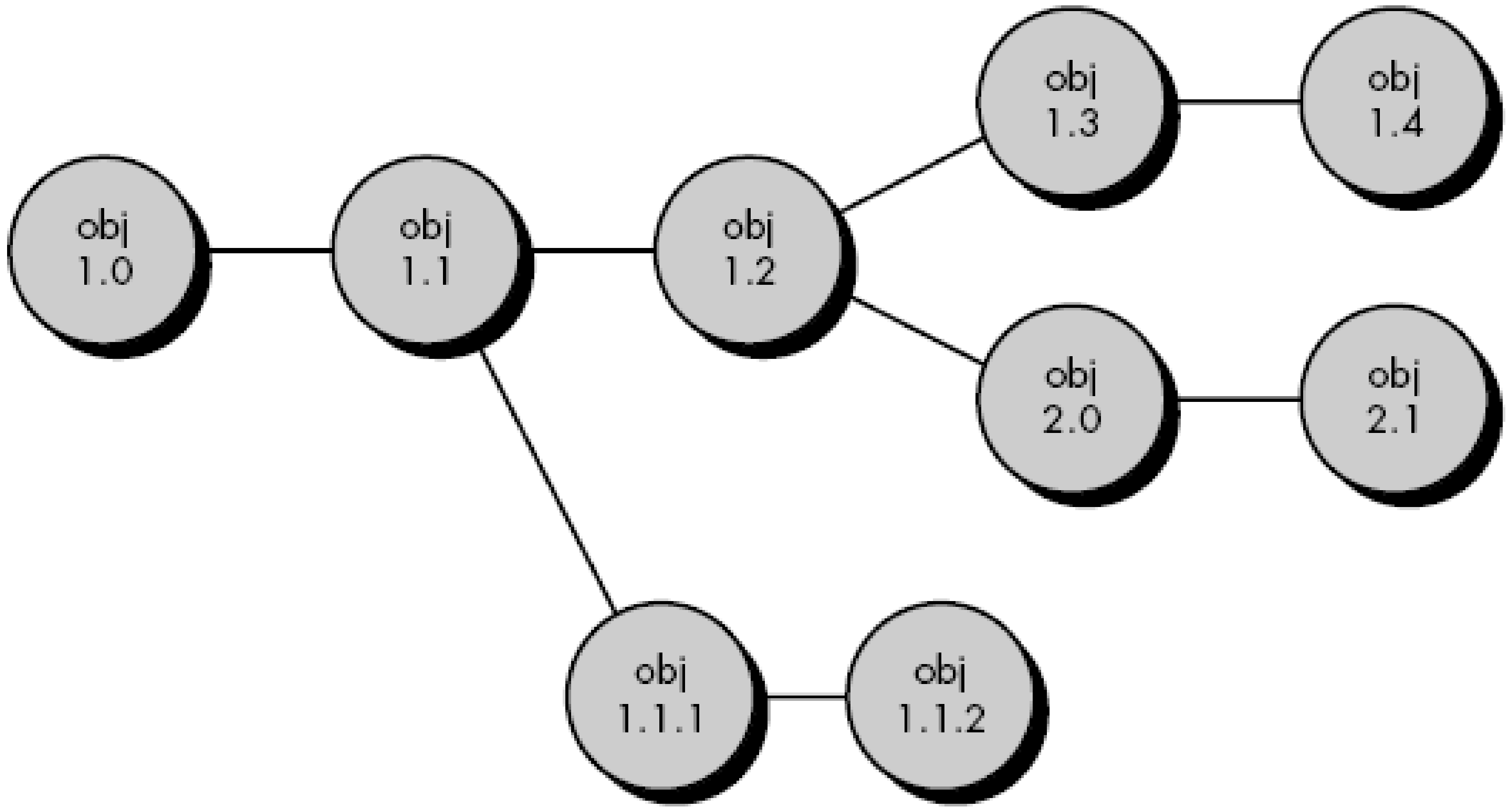
  data model <interrelated> test case class m;

- In the first case, the interrelationship is between a composite object, while the second relationship is between an aggregate object (**data model**) and a basic object (**test case class m**).

Object evolution

- The identification scheme for software objects must recognize that objects evolve throughout the software process.
- Before an object is baselined, it may change many times, and even after a baseline has been established, changes may be quite frequent.
- It is possible to create an *evolution graph* (Gustavsson,1989) for any object. The evolution graph describes the change history of an object

Evolution graph: example



(2) Version Control

- Combines procedures and tools to manage different versions of configuration objects that are created during the software process.
- Each version of the software is a collection of SCIs (source code, documents, data), and each version may be composed of different variants.

(2) Version Control … Example

- Consider a version of a simple program that is composed of entities 1, 2, 3, 4, and 5.
  - Entity 4 is used only when the software is implemented using color displays.
  - Entity 5 is used only when the software is implemented when monochrome displays are available.
- Therefore, two variants of the version can be defined:

  (1) entities 1, 2, 3, and 4;

  (2) entities 1, 2, 3, and 5.

(2) Version Control … Example

- Consider a version of a simple program that is composed of entities 1, 2, 3, 4, and 5.
  - Entity 4 is used only when the software is implemented using color displays.
  - Entity 5 is used only when the software is implemented when monochrome displays are available.
- Therefore, two variants of the version can be defined:

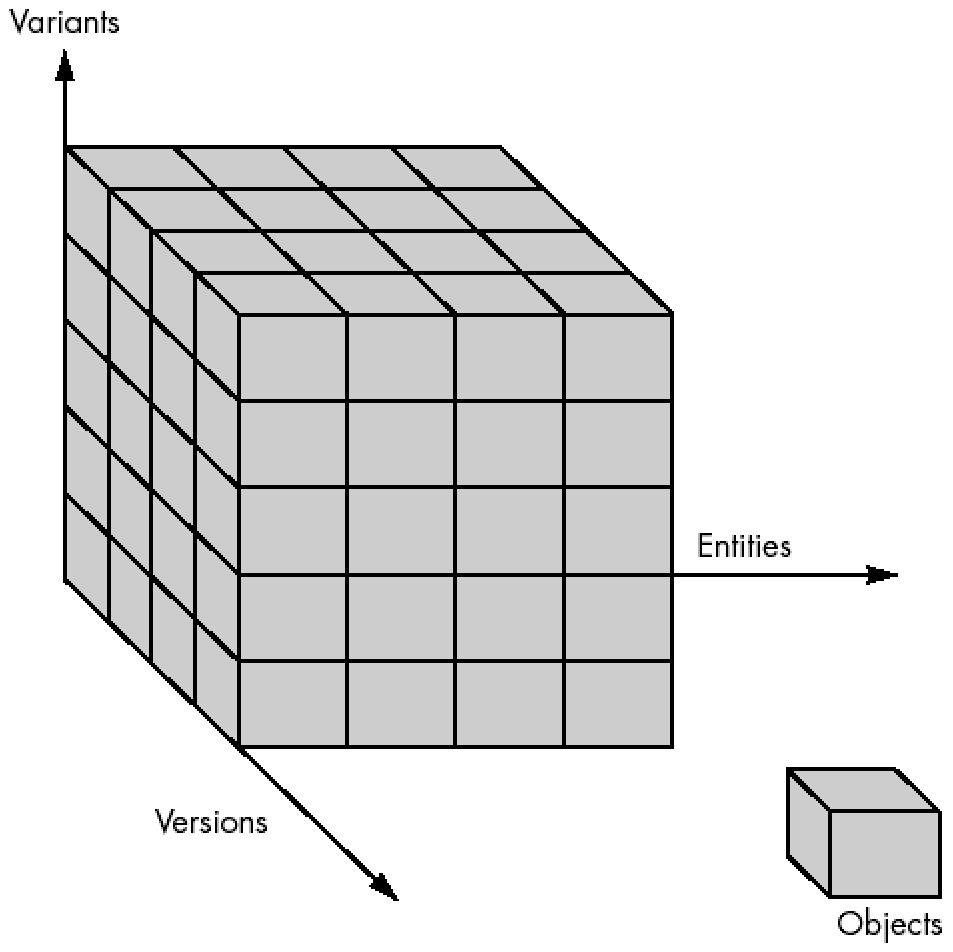
  (1) entities 1, 2, 3, and 4;

  (2) entities 1, 2, 3, and 5.

Three Important Elements in Version Control

- Version: defined when major changes are made to one or more objects
- Entity: a collection of objects at the same revision level.
- Variant: a different collection of objects at the same revision level and therefore coexists in parallel with other variants
- The relationship between configuration objects and entities, variants and versions can be represented in a three-dimensional space **called object pool**

Object Pool



Version Control Approach

- A number of different automated approaches to version control have been proposed over the past decade.
- The primary difference in approaches
  - the sophistication of the attributes that are used to construct specific versions and variants of a system
  - the mechanics of the process for construction.

(3) Change Control

- Facts:
  - Too much change control, we create problems.
  - Too little, and we create other problems.
- For a large software engineering project, uncontrolled change rapidly leads to chaos.
- For such projects, change control combines human procedures and automated tools to provide a mechanism for the control of change.

Systematic of Change Control Process (1)

- A change request is submitted and evaluated
  - to assess technical merit, potential side effects, overall impact on other configuration objects and system functions, and the projected cost of the change.
- The results of the evaluation are presented as a change report,
  - which is used by a change control authority (CCA)—a person or group who makes a final decision on the status and priority of the change.
- An engineering change order (ECO) is generated for each approved change.
- The ECO describes the change to be made, the constraints that must be respected, and the criteria for review and audit.
- The object to be changed is "checked out" of the project database, the change is made, and appropriate SQA activities are applied.
- The object is then "checked in" to the database and appropriate version control mechanisms are used to create the next version of the software.

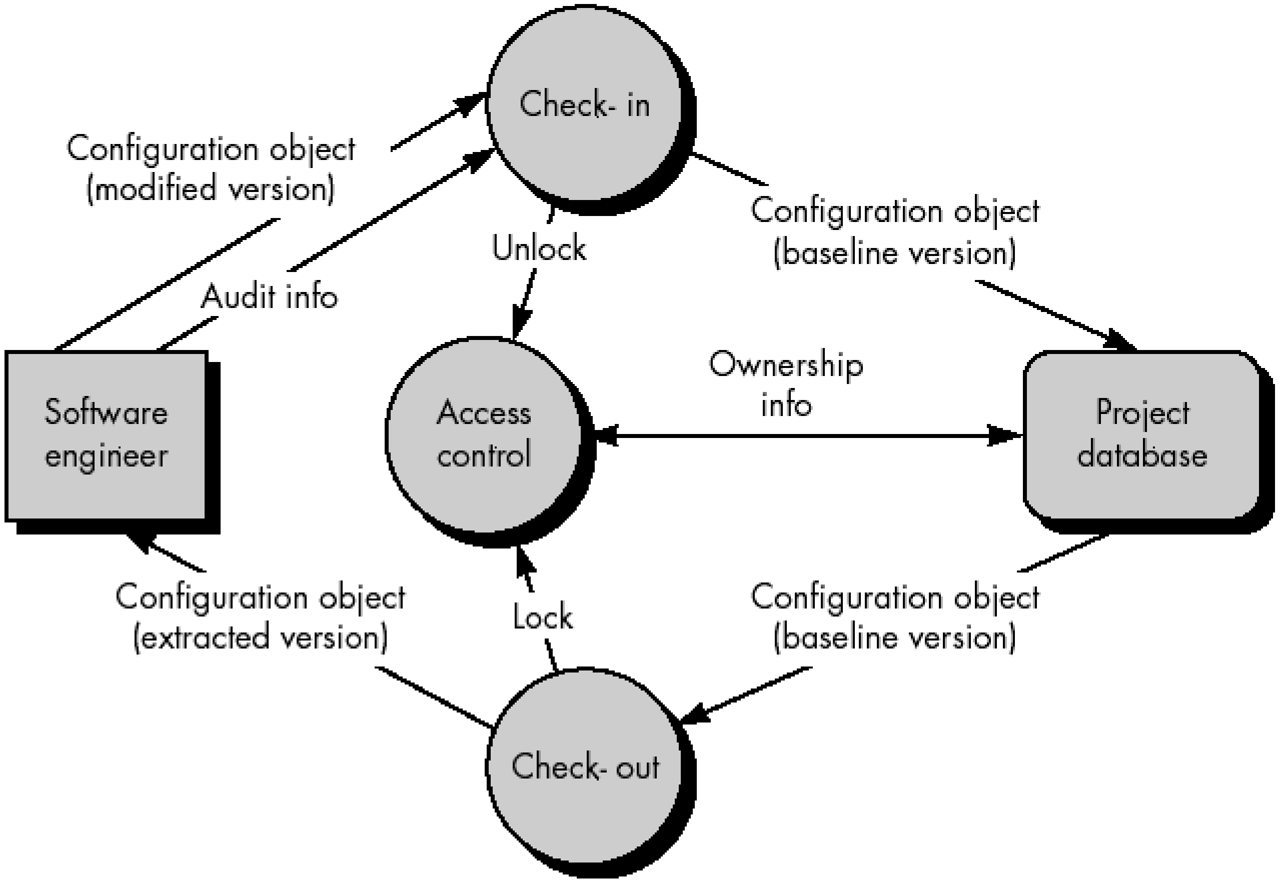
Access and Synchronization Control (1)

- The "check-in" and "check-out" process implements two important elements of change control—access control and synchronization control.
- Access control governs which software engineers have the authority to access and modify a particular configuration object.
- Synchronization control helps to ensure that parallel changes, performed by two different people, don't overwrite one another

Access and Synchronization Control (2)

- Based on an approved change request and ECO, a software engineer checks out a configuration object.
- An access control function ensures that the software engineer has authority to check out the object, and synchronization control locks the object in the project database so that no updates can be made to it until the currently checked out version has been replaced.
- Note that other copies can be checked-out, but other updates cannot be made. A copy of the baselined object, called the extracted version, is modified by the software engineer. After appropriate SQA and testing, the modified version of the object is checked in and the new baseline object is unlocked.

Access and Synchronization Control (3)


Check- in
Configuration object (modified version)
Audit info
Unlock
Configuration object (baseline version)
Software engineer
Access control
Ownership info
Project database
Configuration object (extracted version)
Lock
Configuration object (baseline version)
Check- out

(4) Configuration Audit

- Identification, version control, and change control help the software developer to maintain order in what would otherwise be a chaotic and fluid situation.
- How can we ensure that the change has been properly implemented?
- The answer is (1) formal technical reviews and (2) the software configuration audit.
- The formal technical review focuses on the technical correctness of the configuration object that has been modified.
- The reviewers assess the SCI to determine consistency with other SCIs, omissions, or potential side effects.
- A formal technical review should be conducted for all but the most trivial changes.

Configuration Audit … cont'd

- A software configuration audit complements the formal technical review by assessing a configuration object for characteristics that are generally not considered during review.
- The audit asks and answers the following questions:
  1. Has the change specified in the ECO been made? Have any additional modifications been incorporated?
  2. Has a formal technical review been conducted to assess technical correctness?
  3. Has the software process been followed and have software engineering standards been properly applied?
  4. Has the change been "highlighted" in the SCI? Have the change date and change author been specified? Do the attributes of the configuration object reflect the change?
  5. Have SCM procedures for noting the change, recording it, and reporting it been followed?
  6. Have all related SCIs been properly updated?

(5) Status Reporting

- Configuration status reporting (sometimes called status accounting) is an SCM task that answers the following questions: (1) What happened? (2) Who did it? (3) When did it happen? (4) What else will be affected?
- The flow of information for configuration status reporting (CSR):
  - Each time an SCI is assigned new or updated identification, a CSR entry is made.
  - Each time a change is approved by the CCA (i.e., an ECO is issued), a CSR entry is made.
  - Each time a configuration audit is conducted, the results are reported as part of the CSR task.
  - Output from CSR may be placed in an on-line database, so that software developers or maintainers can access change information by keyword category.

Status Reporting … cont’d

- Configuration status reporting plays a vital role in the success of a large software development project.
- When many people are involved, it is likely that "the left hand not knowing what the right hand is doing" syndrome will occur.
- CSR helps to eliminate these problems by improving communication among all people involved.

References

- Babich, W.A., *Software Configuration Management,* Addison-Wesley, 1986.
- Choi, S.C. and W. Scacchi, "Assuring the Correctness of a Configured Software Description," *Proc. 2nd Intl. Workshop on Software Configuration Management,* ACM, Princeton, NJ, October 1989, pp. 66–75.
- Gustavsson, A., "Maintaining the Evoluation of Software Objects in an Integrated Environment," *Proc. 2nd Intl. Workshop on Software Configuration Management,* ACM, Princeton, NJ, October 1989, pp. 114–117.