北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

Advanced Software Engineering

Lecture 4: Software Project Management

by

Professor Harold Liu

Agenda

- □ Introduction
- □ Project Estimation
- □ Progress Management
- □ Configuration Management

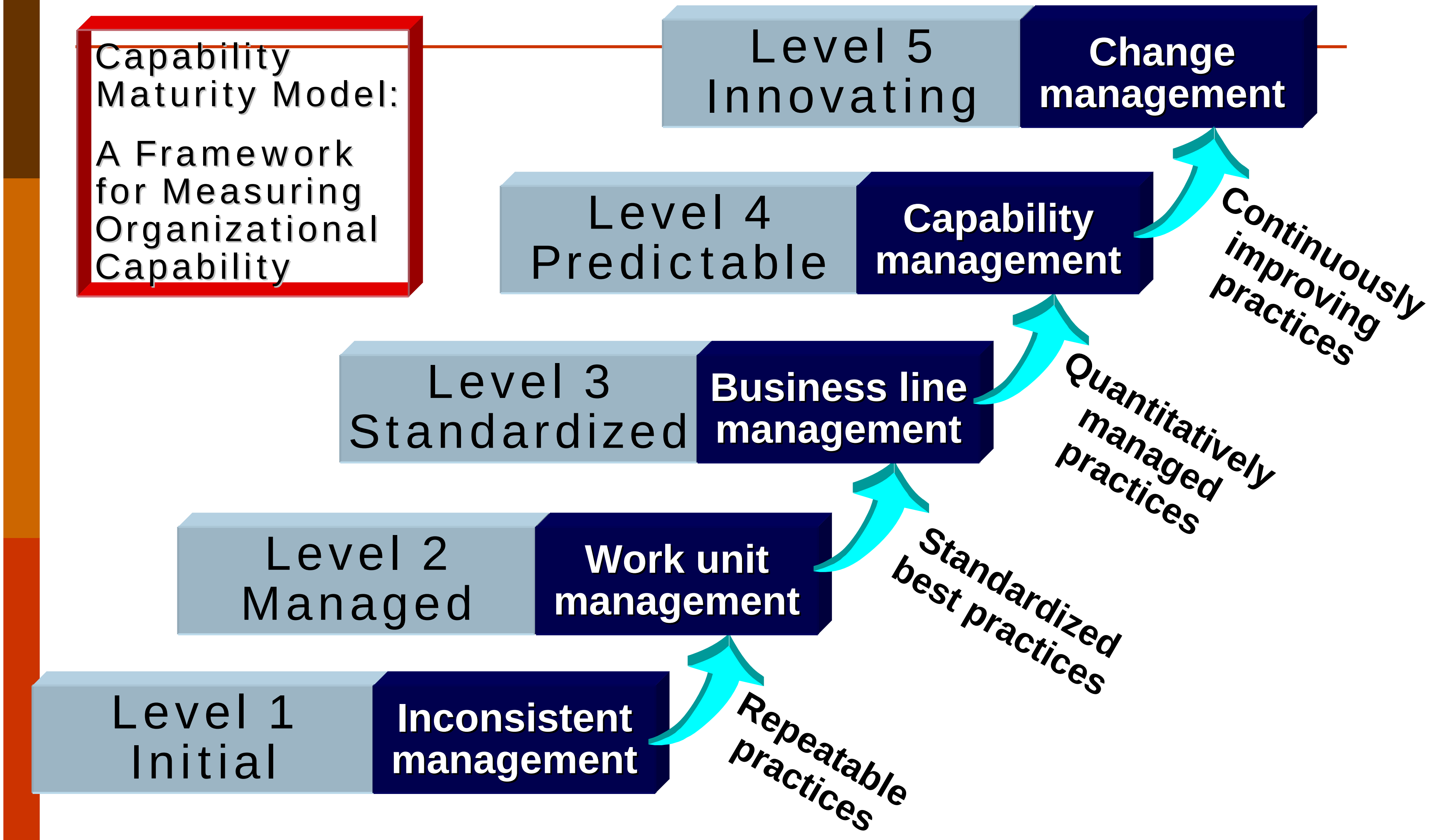
1. Introduction

- What can be regarded as a successful project?
  - To meet the SRS specified requirements by clients
  - Within the deadline specified in the contract
  - Within the budget

Management Scope

- People, Product, Process, and Project
- People management
  - Bill Curtis from CMU in 1994 proposed the P-CMM (people capability maturity model）

The Five Capability Levels



Capability Maturity Model:

A Framework for Measuring Organizational Capability

Level 5 Innovating — Change management

Level 4 Predictable — Capability management

Level 3 Standardized — Business line management

Level 2 Managed — Work unit management

Level 1 Initial — Inconsistent management

Repeatable practices
Standardized best practices
Quantitatively managed practices
Continuously improving practices

Low Maturity Organizations

Rework

Rework

Firefighting

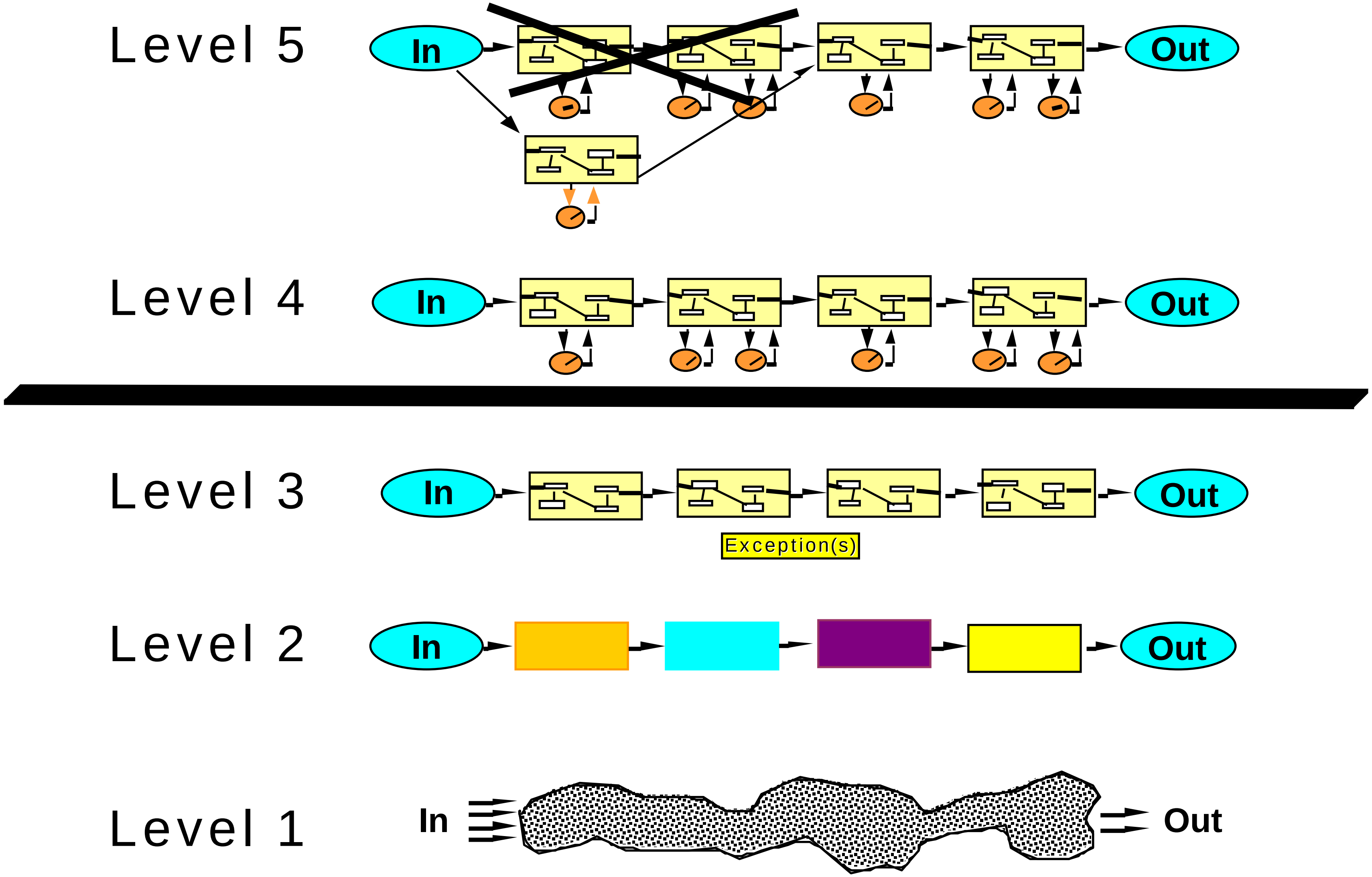
Heroes

People capability variation

Transaction workers: +/- 100%

Knowledge workers: +/- 1000%

Management Visibility



Level 5

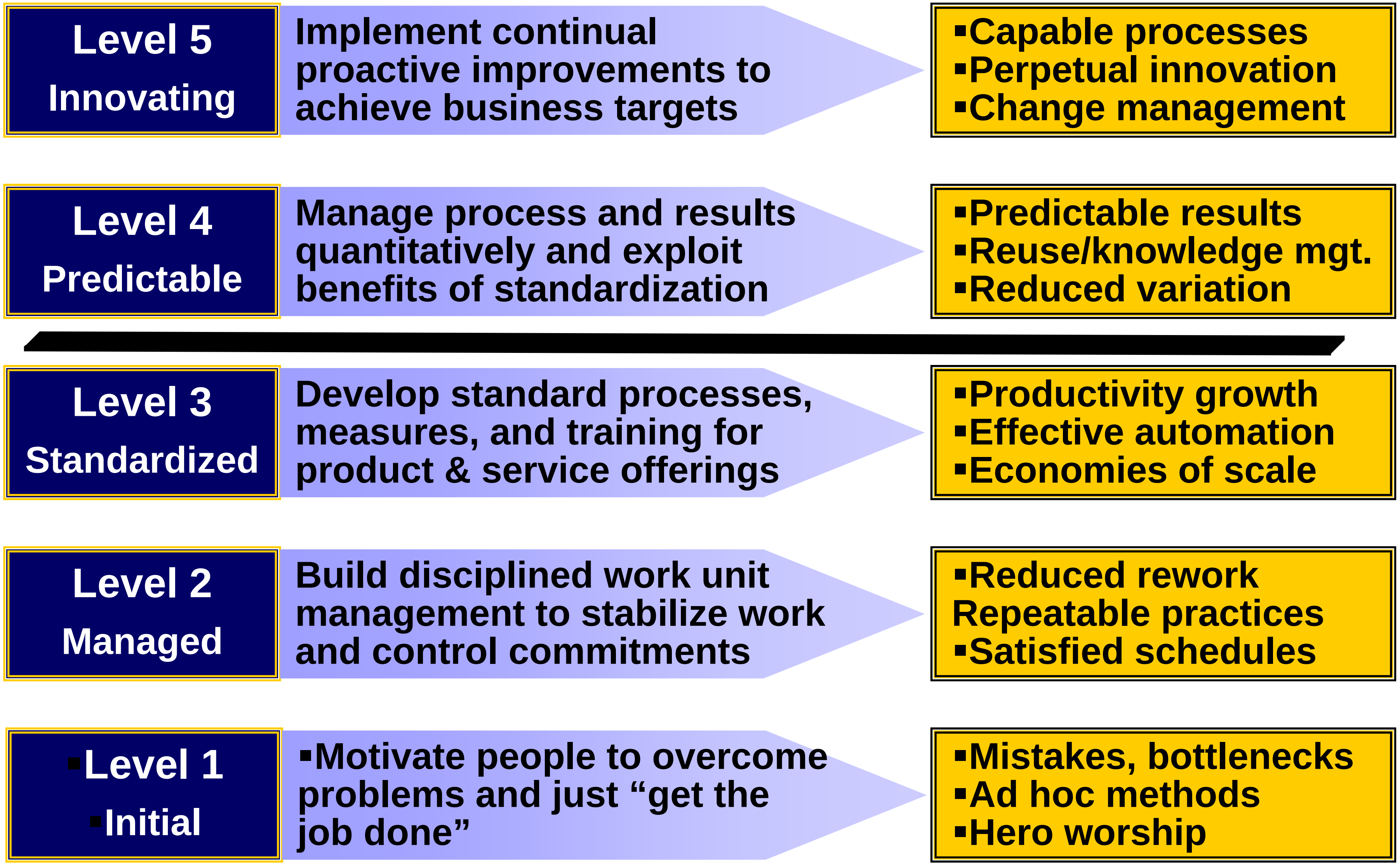
Level 4

Level 3

Level 2

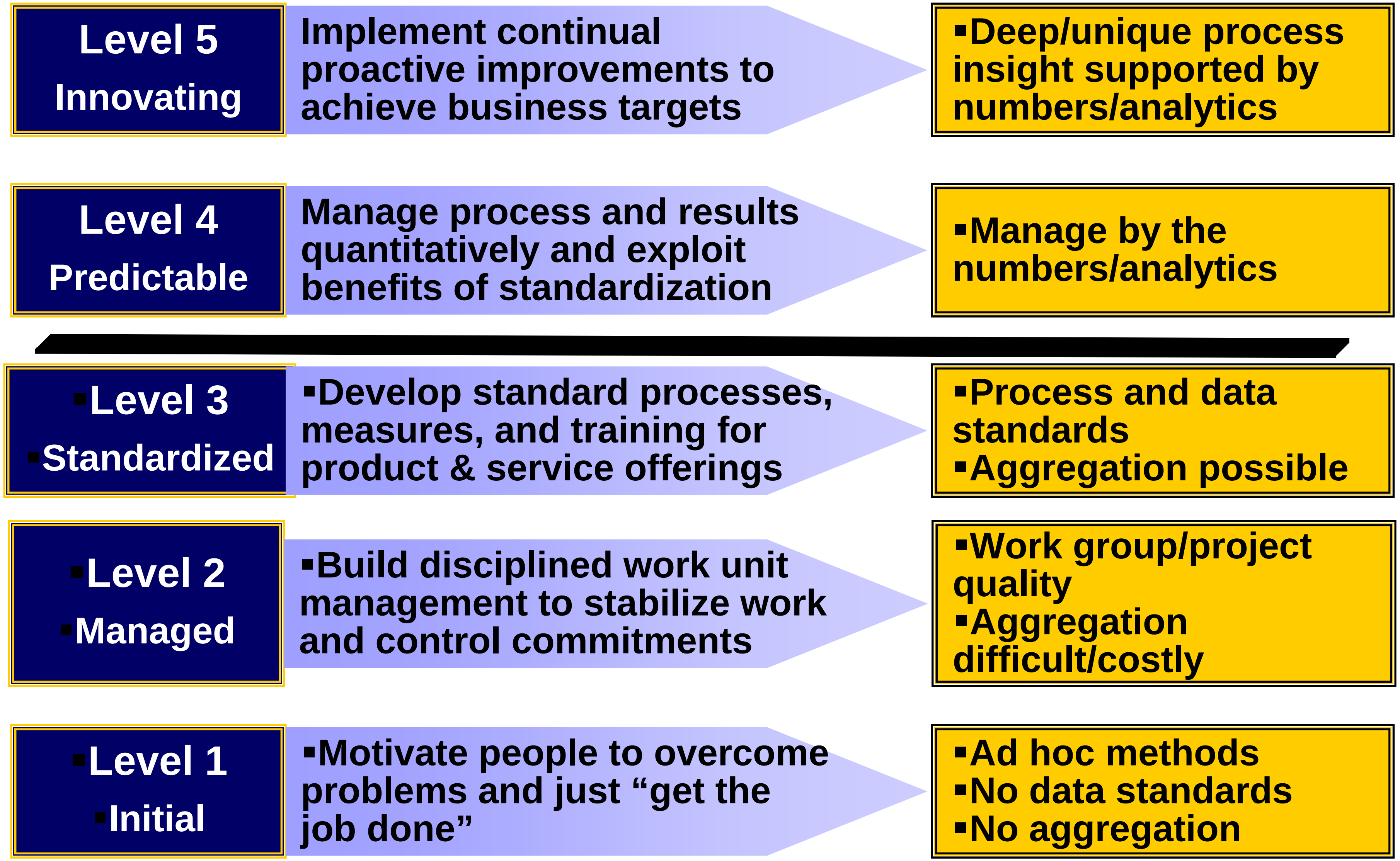
Level 1

How the MM’s Works-General



Level 5 Innovating	Implement continual proactive improvements to achieve business targets	▪Capable processes ▪Perpetual innovation ▪Change management
Level 4 Predictable	Manage process and results quantitatively and exploit benefits of standardization	▪Predictable results ▪Reuse/knowledge mgt. ▪Reduced variation
Level 3 Standardized	Develop standard processes, measures, and training for product & service offerings	▪Productivity growth ▪Effective automation ▪Economies of scale
Level 2 Managed	Build disciplined work unit management to stabilize work and control commitments	▪Reduced rework Repeatable practices ▪Satisfied schedules
▪Level 1 ▪Initial	▪Motivate people to overcome problems and just “get the job done”	▪Mistakes, bottlenecks ▪Ad hoc methods ▪Hero worship

How the MM’s Work-Measurement



Level 5 Innovating	Implement continual proactive improvements to achieve business targets	▪Deep/unique process insight supported by numbers/analytics
Level 4 Predictable	Manage process and results quantitatively and exploit benefits of standardization	▪Manage by the numbers/analytics
▪Level 3 ▪Standardized	▪Develop standard processes, measures, and training for product & service offerings	▪Process and data standards ▪Aggregation possible
▪Level 2 ▪Managed	▪Build disciplined work unit management to stabilize work and control commitments	▪Work group/project quality ▪Aggregation difficult/costly
▪Level 1 ▪Initial	▪Motivate people to overcome problems and just “get the job done”	▪Ad hoc methods ▪No data standards ▪No aggregation

□ Who to Manage?

1. The stakeholders:

- Senior Mangers—Business related decision making
- PM—project planning and execution, including staffing
- Developers
- Client—propose requirements and interact with developers
- Users—directly use the product

2. Team leaders:

- for small project, PM is the team leader
- for big projects that include many design, implementation and testing teams, team leaders will be responsible for each working item, particularly interact with team members

3. Team: most important is WHO to do WHAT in an cooperative way

- self-responsibility
- mutual trust and support
- Adequate communication

- Product Management

  Before the project is started, PM needs to make sure:

  - execution environment
  - functionality and performance
  - I/O

  then PM can do project estimation, risk analysis and planning

- Process Management

  to decide what activities are needed and their requirements and sequence of doing that

- Project Management

  The goal is how to optimally use the available resources, carefully plan the agreed project and make sure to deliver the required product

3 Responsibilities for Project Management:

- Planning: including to make the plan, project estimation, risk analysis and management, process management, plan tracking and monitoring

- Resource management: people, cost, information

- Result management: quality assurance and delivery configurations

2. Project Estimations

- It happens ***right AFTER*** the confirmation of the project goal and basic functionalities. It serves as an important input to the project planning.
- Planning

(1) What are features of this project?

(2) Choose a project lifecycle model, and tasks of each stage

(3) milestones/prototypes, and final delivery

(4) estimations: size, efforts, costs, resources

(5) Develop the progress plan

(6) Risk analysis

(7) Develop the project plan

- Estimation:
  - Size
    - measured in Kilo/thousand line of code (KLOC) on the *FINAL* software product
    - However, when planning the project, we cannot know the accurate KLOC for the final product, thus only the FP (Functional Point), estimated from the SRS
  - Effort: man month
  - Cost: more often we talk about the staffing cost

● Cost Estimation:

A software company can compute its own productivity and value per man month, based on the statistics after the completion of some software projects

- Productivity: KLOC per man month, or FP per man month
- Value per man month

Effort = Size / Productivity

Cost = Effort × Value per man month

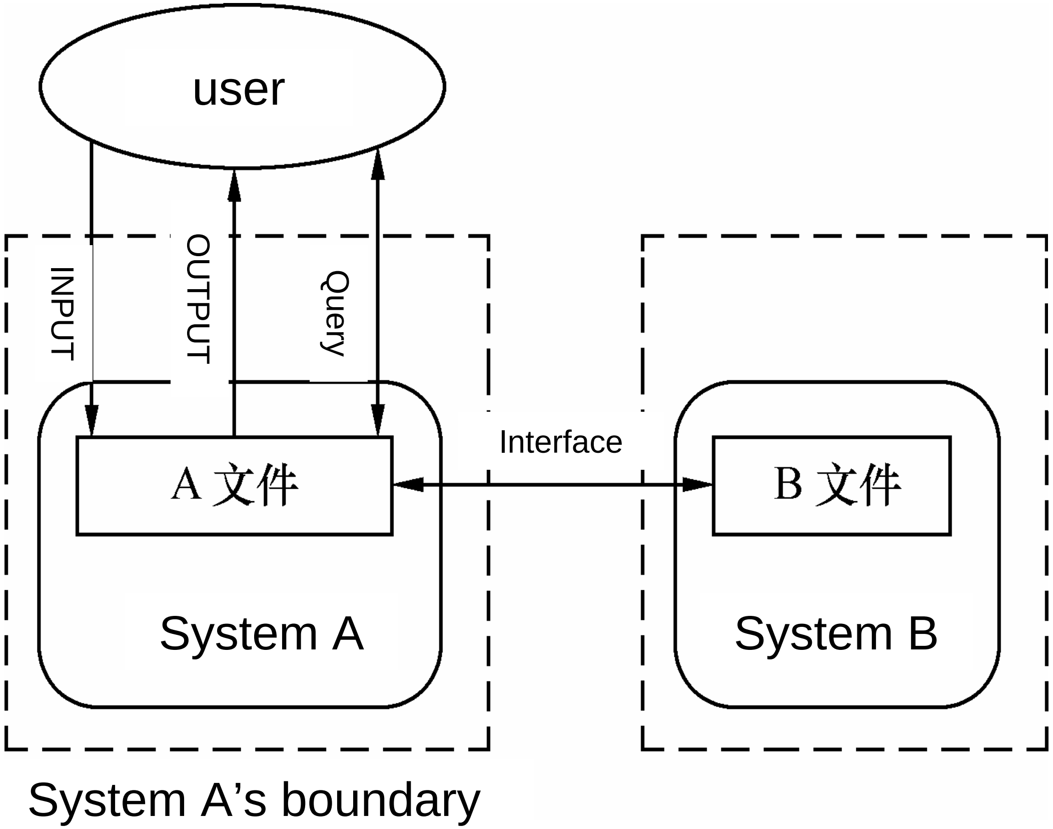
Size Estimation by FP

- To overcome the difficulties of knowing the size of the project as a priori
- Use the software functionality for computation
- Assumption:
  - Size is related to functionality
  - Size is irrelevant to how to describe/implement the functionality

Function Types

- System/application boundary

In the figure, system A has 4 functionalities to cross the system boundary, and thus treated as the external functionalities

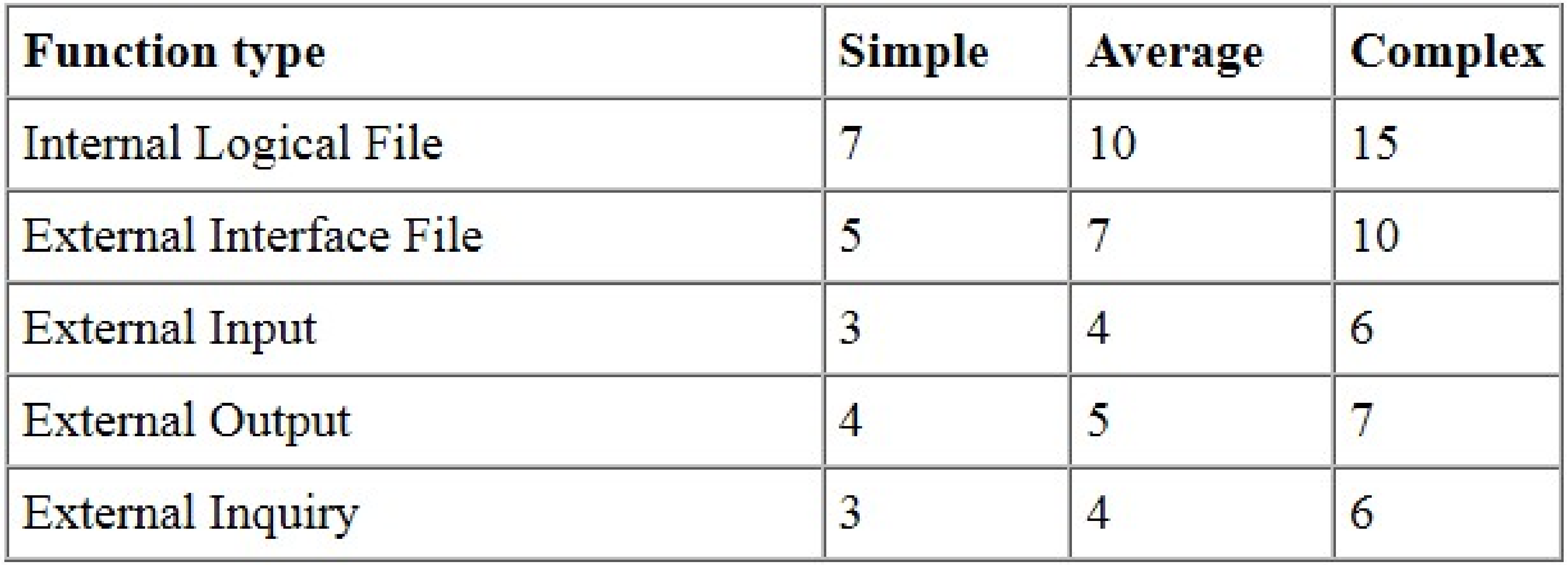


- ***The 5 function types identified are***
  - *external input* which receives information from outside the application boundary
  - *external output* which presents information of the information system
  - *external enquiries* which is special kind of an external output. An external inquiry presents information of the information system based on a uniquely identifying search criterion, without applying additional processing (such as calculations).
  - *internal logical files* contains permanent data that is relevant to the user The information system references and maintains the data and
  - *external interface files* also contains permanent data that is relevant to the user. The information system references the data, but the data is maintained by another information system

2. Complexity of Functionality Types

Simple, Average and Complex

Assign different weight factors to represent



Function type	Simple	Average	Complex
Internal Logical File	7	10	15
External Interface File	5	7	10
External Input	3	4	6
External Output	4	5	7
External Inquiry	3	4	6

3. Unadjusted Function Points

$$\mathrm{UFP} = \sum_{i=1}^{5}\sum_{j=1}^{3}\omega_{ij}C_{ij}$$

where:

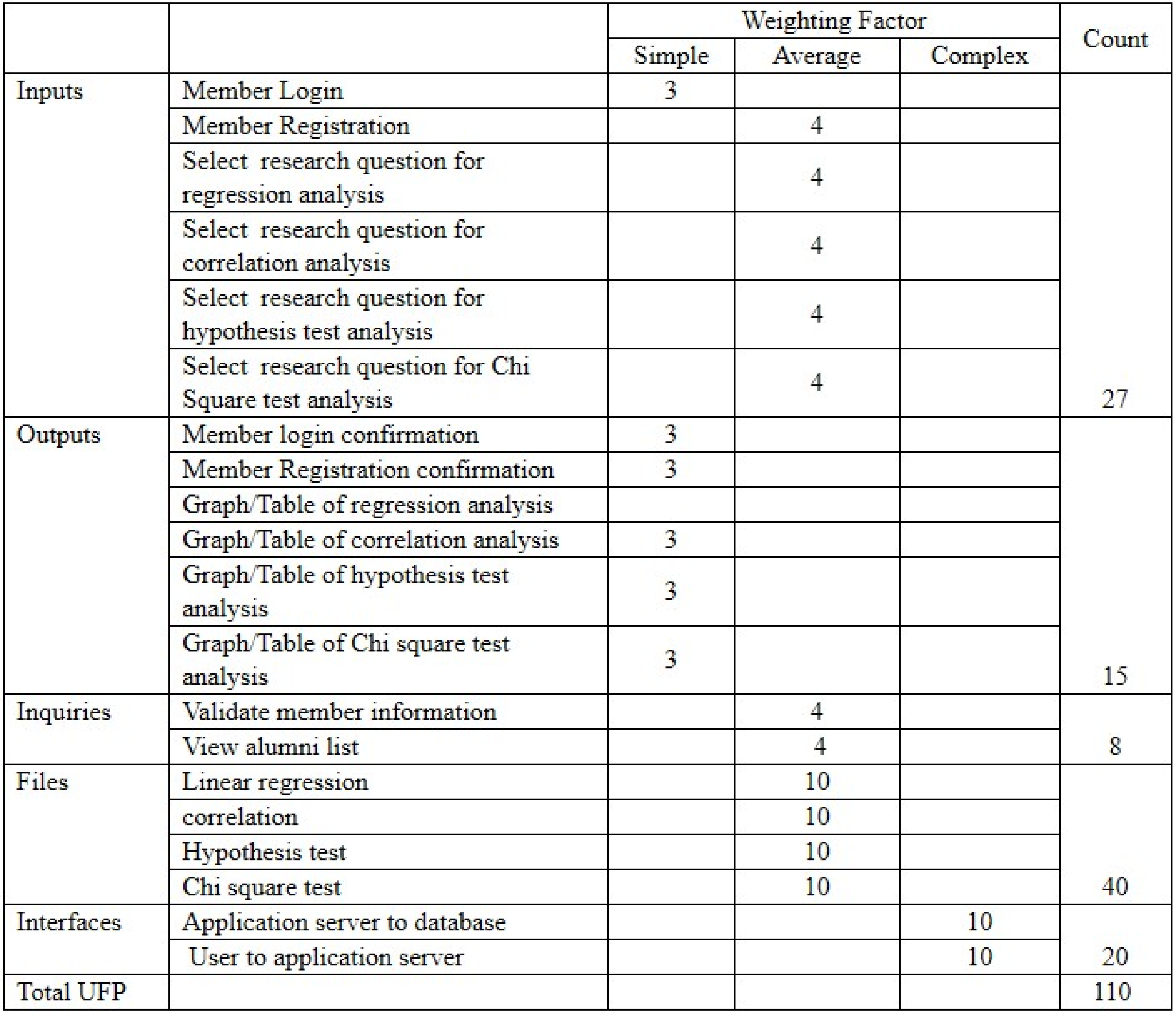
i denotes the index of a type of functionality, i.e., i=1…5

j denotes the degree of the complexity, j=1,2,3

ω_{ij} denotes weight factor of the i-th functionality and the j-th level of complexity

C_{ij} denotes the number of functionalities of i-th functionality and the j-th level of complexity

Example

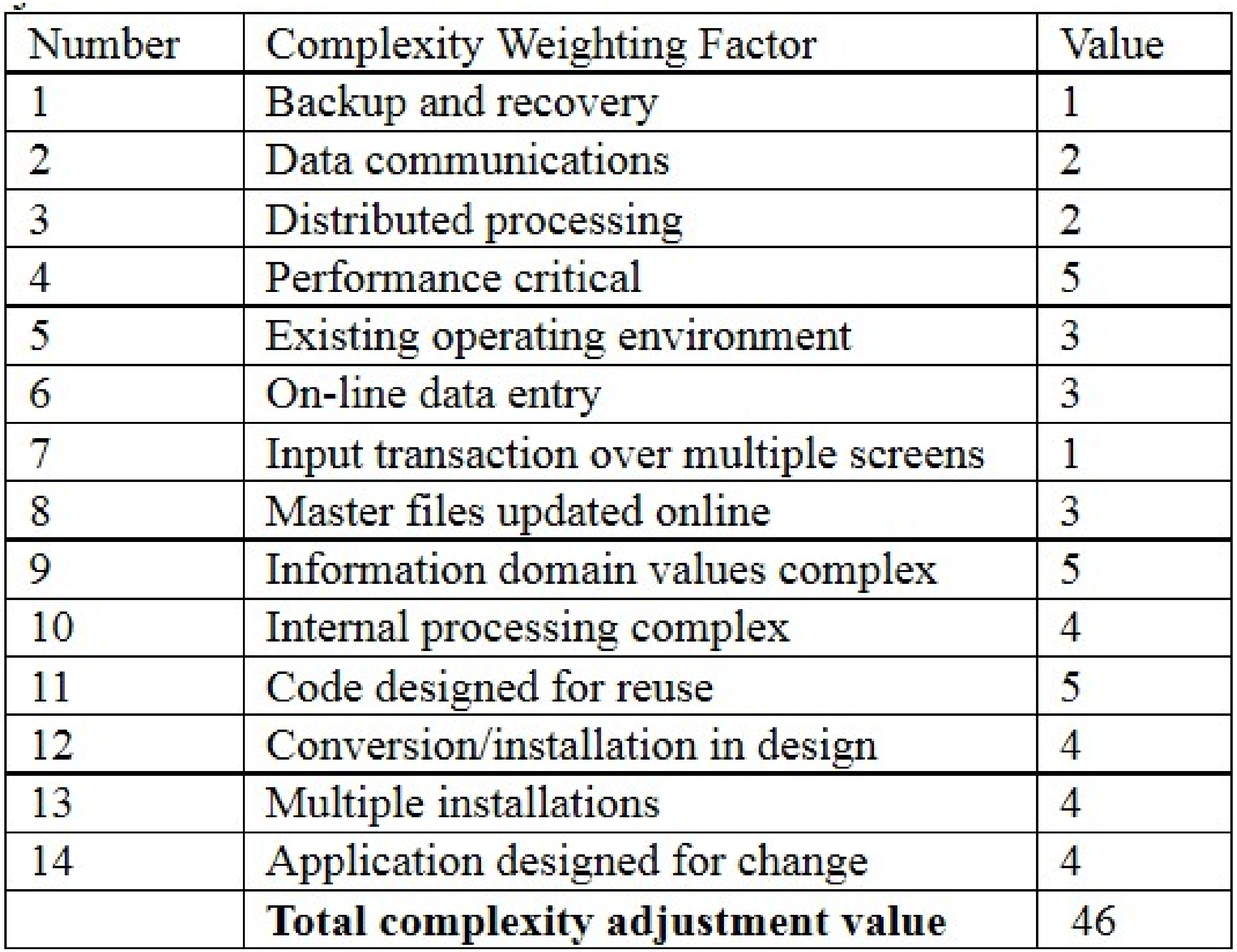


		Weighting Factor			Count
		Simple	Average	Complex	
Inputs	Member Login	3			27
	Member Registration		4		
	Select research question for regression analysis		4		
	Select research question for correlation analysis		4		
	Select research question for hypothesis test analysis		4		
	Select research question for Chi Square test analysis		4		
Outputs	Member login confirmation	3			15
	Member Registration confirmation	3			
	Graph/Table of regression analysis				
	Graph/Table of correlation analysis	3			
	Graph/Table of hypothesis test analysis	3			
	Graph/Table of Chi square test analysis	3			
Inquiries	Validate member information		4		8
	View alumni list		4		
Files	Linear regression		10		40
	correlation		10		
	Hypothesis test		10		
	Chi square test		10		
Interfaces	Application server to database			10	20
	User to application server			10	
Total UFP					110

4. Adjusted Function Points

To consider the software characteristics

（1）Impact factors and Weights



Number	Complexity Weighting Factor	Value
1	Backup and recovery	1
2	Data communications	2
3	Distributed processing	2
4	Performance critical	5
5	Existing operating environment	3
6	On-line data entry	3
7	Input transaction over multiple screens	1
8	Master files updated online	3
9	Information domain values complex	5
10	Internal processing complex	4
11	Code designed for reuse	5
12	Conversion/installation in design	4
13	Multiple installations	4
14	Application designed for change	4
	Total complexity adjustment value	46

(2) Complexity Weighting Factor

- □ assigned a value (complexity adjustment value) that ranges between 0 (not important) to 5 (absolutely essential)
- □ Collectively, ranges from 0 to 70

(3) Complexity Adjustment Factor

$$CAF=0.65+0.01N$$

ranges from 0.65 to 1.35, i.e., maximum adjustment ratio is 35%。

5. Delivered Function Point (DFP)

Functional point value after adjustment

$$DFP = CAF \times UFP$$

6. Delivered lines of code (DLOC)

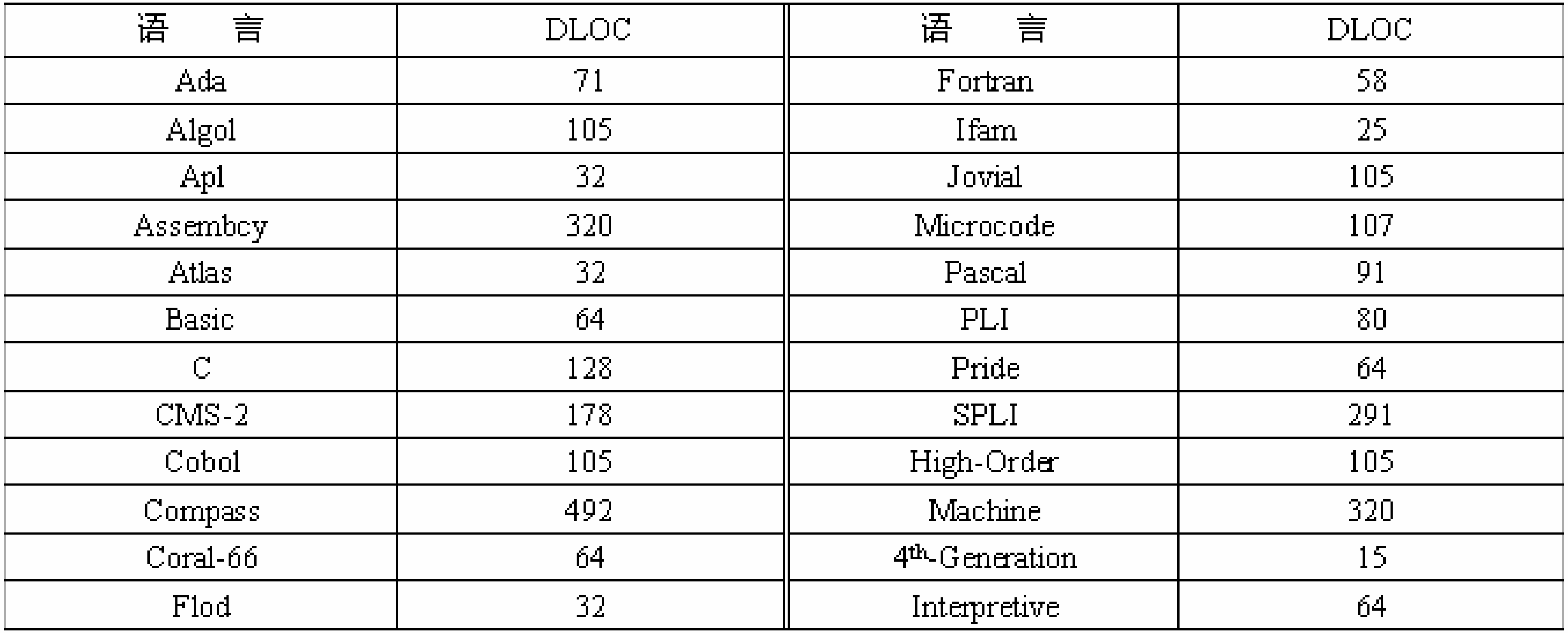
□ Represent the software size

□ Facilitate the cost estimation

Relationship:

e.g., 1 DFP = 105 DLOC (COBOL)

1 DFP = 128 DLOC (C)



语言	DLOC	语言	DLOC
Ada	71	Fortran	58
Algol	105	Ifam	25
Apl	32	Jovial	105
Assembcy	320	Microcode	107
Atlas	32	Pascal	91
Basic	64	PLI	80
C	128	Pride	64
CMS-2	178	SPLI	291
Cobol	105	High-Order	105
Compass	492	Machine	320
Coral-66	64	4th-Generation	15
Flod	32	Interpretive	64

7. FP Advantages

（1） only relates to SRS

（2） DFP is irrelevant of the implemented language

8. FP Disadvantages

（1） Subjective factors

- different analyzers have different understandings of the SRS
- different complexity analysis

（2） currently no automatic tools support

Cost Estimations

1. Delphi Method

based on expert evaluations

Steps:

1 distribute a form that contains the SRS and evaluation criterions to each expert

2 Experts investigate the SRS, then organizer hosts the discussion meeting where experts exchange their ideas

3 Experts evaluate 3 metric values, and anonymously fill in the form:

a_i——minimum size of the software (min KLOC)

m_i——most probable size of the software (poss. KLOC)

b_i——maximum size of the software (max KLOC)

4 Compute the expected value E_i, and then average expectation E (suppose experts are indexed by i, i=1…n)

$$E_i = \frac{a_i + 4m_i + b_i}{6}; \qquad E = \frac{1}{n}\sum_{i=1}^{n} E_i$$

n is the number of experts

5 Meeting again, to discuss the differences, and experts re-evaluate the size of the software product again

- □ Repeat Step 4 to 5 a few rounds of iterations, and finally can receive an agreed (by all experts) size of the software product (in terms of KLOC)
- □ Base on the software company statistics, estimate the “COST per KLOC”, and times by the size of the product, returns the total cost.

2. COCOMO Model

- □ COCOMO stands for COnstructive COst MOdel
- □ It is an open system First published by Dr Barry Bohem in 1981
- □ Worked quite well for projects in the 80’s and early 90’s
- □ Could estimate results within ~20% of the actual values 68% of the time

- COCOMO has three different models (each one increasing with detail and accuracy):
  - **Basic**, applied early in a project
  - **Intermediate**, applied after requirements are specified.
  - **Advanced**, applied after design is complete
- COCOMO has three different modes:
  - **Organic** – "relatively small software teams develop software in a highly familiar, in-house environment"
  - **Embedded** – operate within tight constraints, product is strongly tied to "complex of hardware, software, regulations, and operational procedures"
  - **Semi-detached** – intermediate stage somewhere between organic and embedded. Usually up to 300 KDSI (Kilo/thousand *Delivered* Source Instructions)

3 Models

□ Basic Model

- static, single variable, based on the KLOC
- Effort: $E = a_b (\mathrm{KLOC})^{b_b}$ (man month)

a_b and b_b are model factors

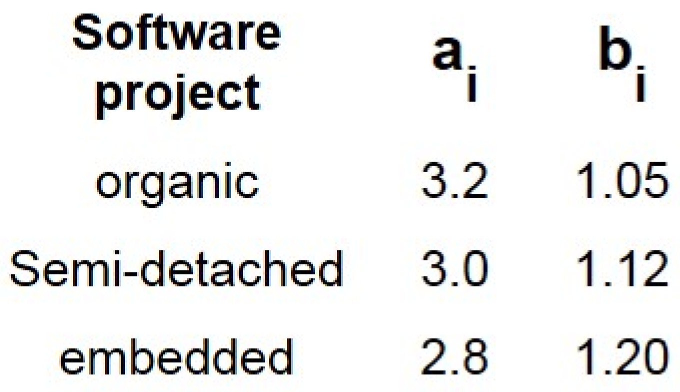
- development cycle: T=c*E^d (months)

Mode	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

□ Intermediate Model

- Besides KLOC, consider EAF (Effort Adjustment Factor), derived from the Cost Drivers, EAF for the basic model is 1
- Effort computation:

$$E = a_i(\mathrm{KLOC})^{b_i} \times \mathrm{EAF}$$



Software project	a_i	b_i
organic	3.2	1.05
Semi-detached	3.0	1.12
embedded	2.8	1.20

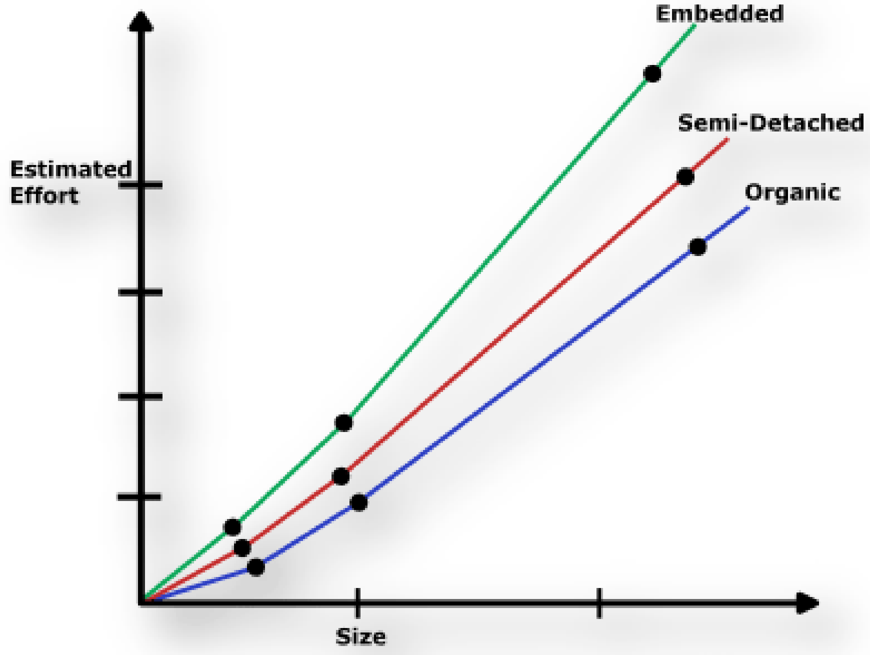
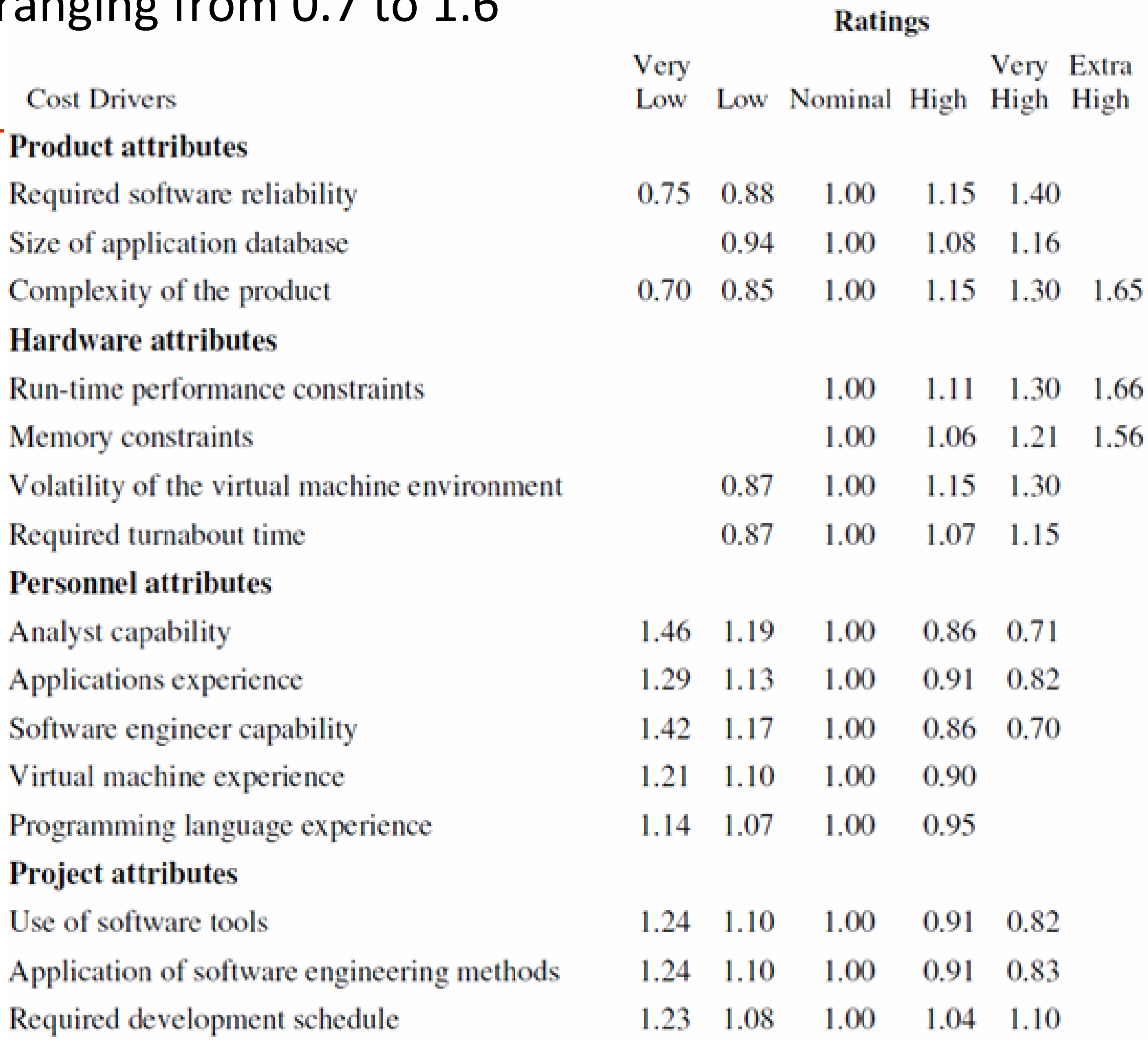


Fig. Effort Verses Product size

EAF, ranging from 0.7 to 1.6



	Ratings					
Cost Drivers	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Use of software tools	1.24	1.10	1.00	0.91	0.82	
Application of software engineering methods	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

Advanced Model

- It incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.

stage ratings	Req & HL D	Low Level D	Prog and UT	Integra T	overall
Very low	0.80	0.80	0.80	0.60	0.75
low	0.90	0.90	0.90	0.80	0.88
normal	1.00	1.00	1.00	1.00	1.00
high	1.10	1.10	1.10	1.30	1.15
Very high	1.30	1.30	1.30	1.70	1.40

Example

- □ Project is ***a flight control system*** (mission critical) with *319,000 LOC* in *embedded* mode
- □ Reliability must be very high (RELY=1.40).

So we can calculate:

- □ **E** = 1.40*2.8*(319)^1.20 = 3961 MM
- □ **T** = 2.5*(3961)^0.32 = 35.4 months
- □ **Average Staffing** = 3961 MM/35.4 months = 112

Discussions

- COCOMO is the most popular method however for any software cost estimation you should really use more then one method
- Best to use another method that differs significantly from COCOMO so your project is examined from more then one angle
- Even companies that sell COCOMO based products recommend using more then one method. Softstar (creators of Costar) will even provide you with contact information for their competitor’s products

3. Progress Management

- □ WHY we need progress management?
- □ Gantt Chart
- □ Time Scalar Network
- □ PERT Chart

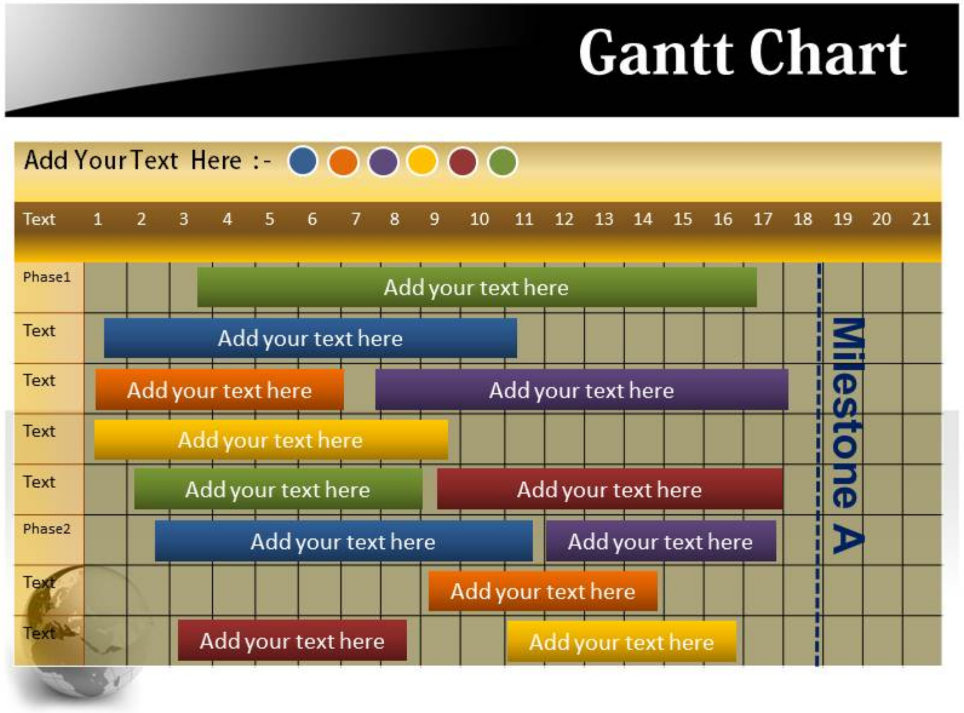
Problem: WHY and HOW

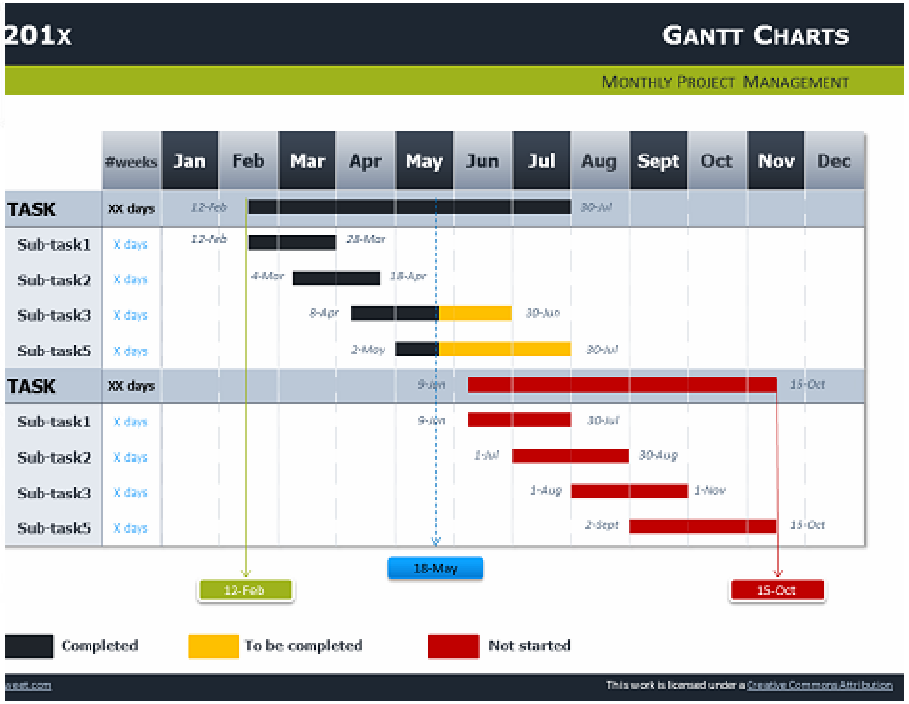
- Delivery delays always happen although we have plans

- Factors to consider to make a progress?
  - Project division into sub-tasks, in multiple levels
  - Make sure the causal relationship between sub-tasks
  - Time needed to complete a task, in man day effort
  - Make sure the effort committed is actually there!
  - Responsibility of each staff
  - Result of each sub-task, as part of overall goal
  - Milestones (baseline)

Gantt Chart

- Horizontal axis: time
- Vertical axis: sub-tasks





- CANNOT clearly describe the how subtasks are connecting together in a sequel.

Characteristics

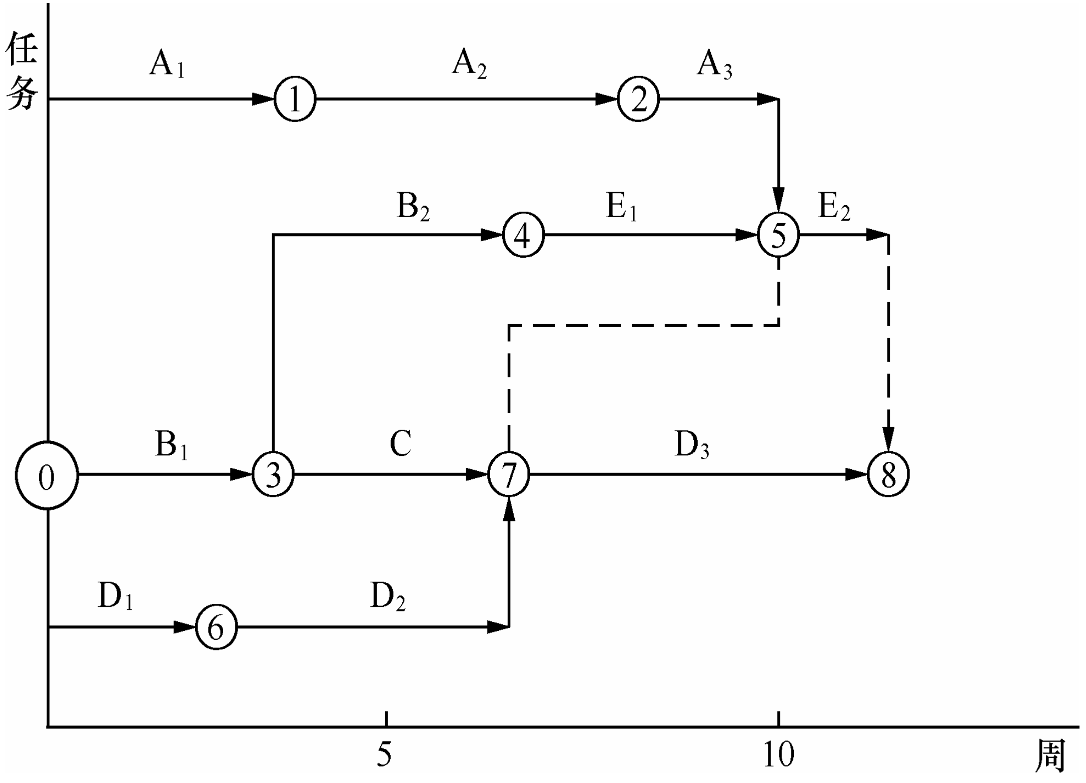
- The bar in each row identifies the corresponding task
- The horizontal position of the bar identifies start and end times of the task
- Bar length represents the duration of the task
- Task durations can be compared easily
- Good for allocating resources and re-scheduling
- Precedence relationships can be represented using arrows
- Critical activities are usually highlighted
- Slack times are represented using bars with doted lines
- The bar of each activity begins at the activity earliest start time (ES)
- The bar of each activity ends at the activity latest finish time (LF).

- Advantages
  - Simple
  - Good visual communication to others
  - Task durations can be compared easily
  - Good for scheduling resources

- Disadvantages
  - Dependencies are more difficult to visualise
  - Minor changes in data can cause major changes in the chart

Time Scalar Network

- Directional line to indicate the sub-task connectivity
- Number the connector point to clearly illustrate the sub-task dependency



PERT Chart

- □ Program Evaluation and Review Technique
- □ OR: Critical Path Method (CPM)
- ● Directional arrow as the edge to represent the sub-tasks: name, length (time to completion)
- ● Numbered circle as the vertices
- ● Edges and vertices construct the MESH topology, then form a path, easy for further computation and analysis

Example

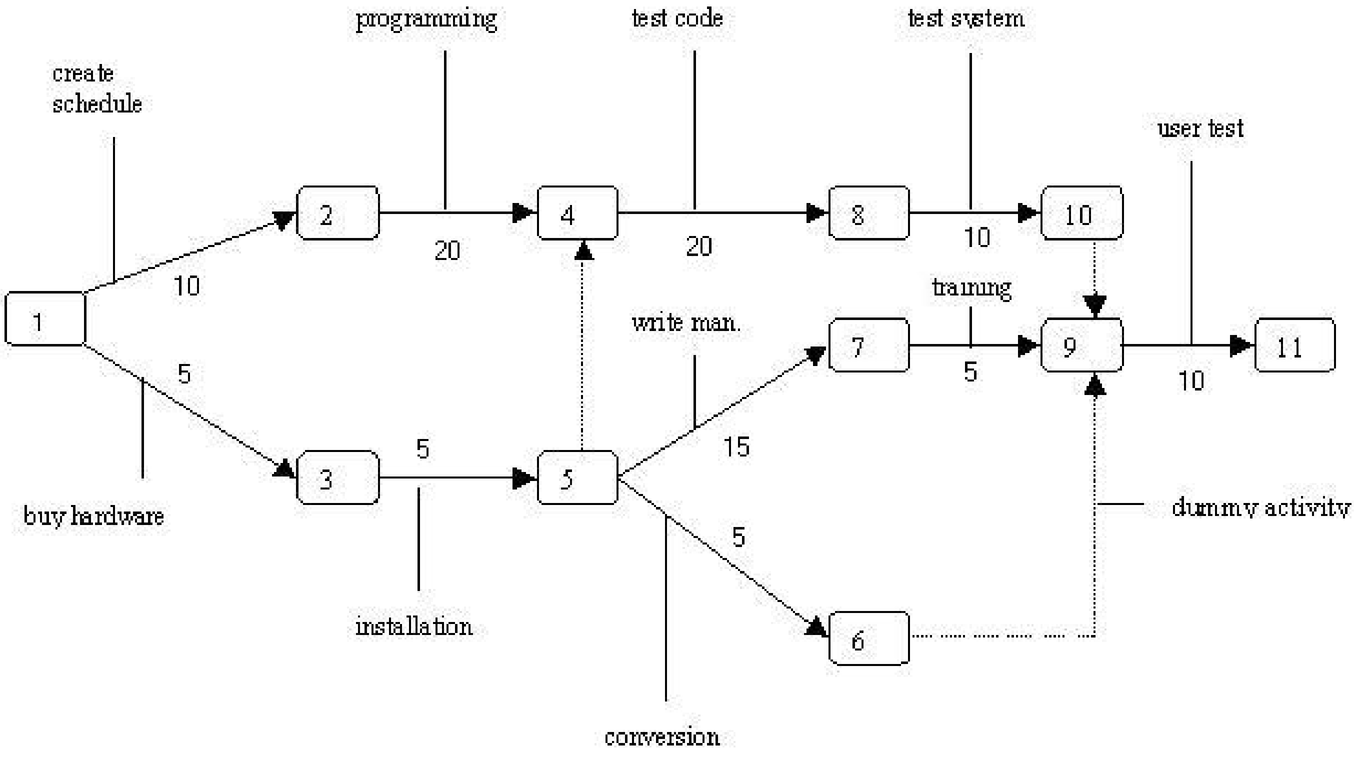
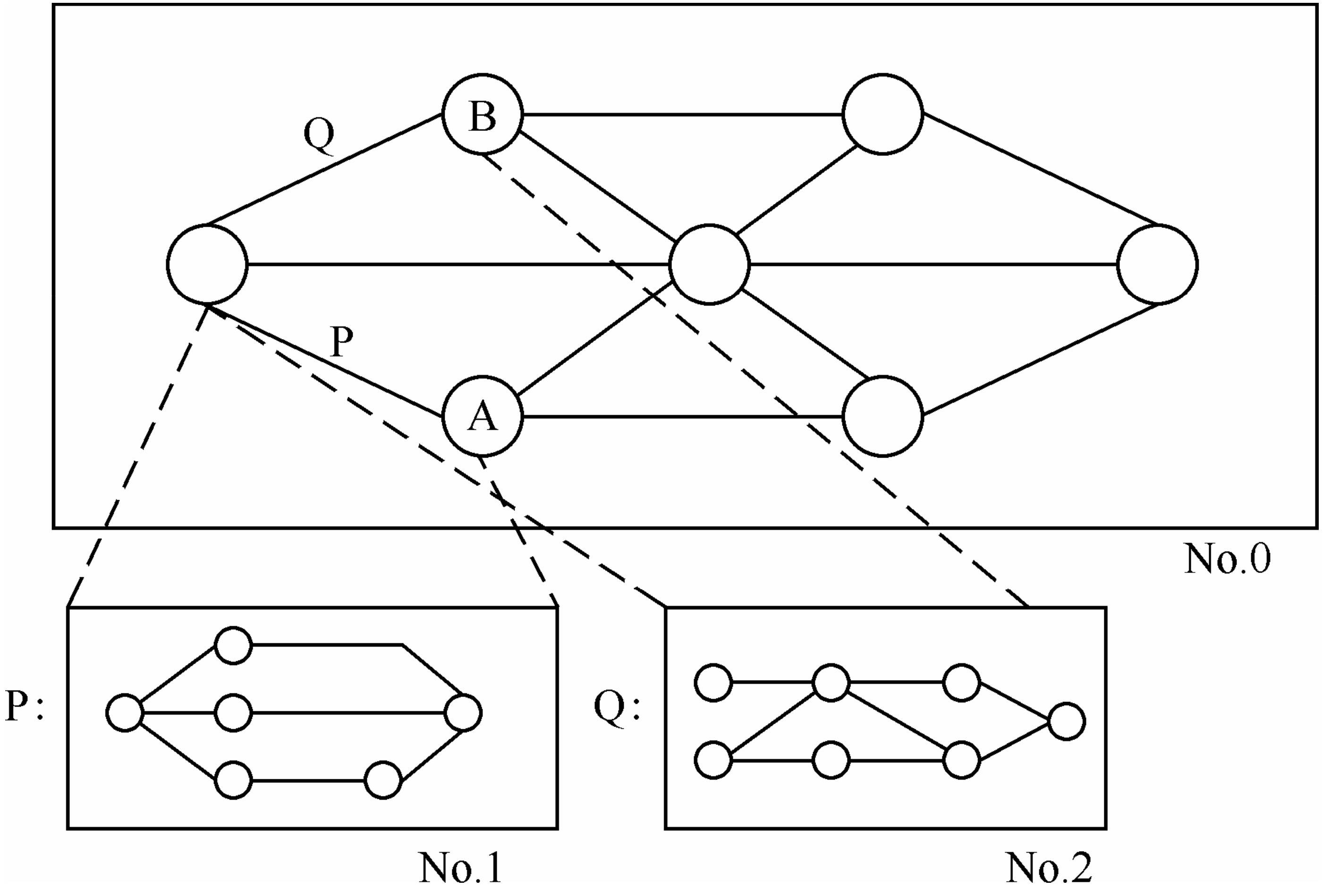


Fig. 1:
PERT Chart

* Numbered rectangles are nodes and represent events or milestones.
* Directional arrows represent dependent tasks that must be completed sequentially.
* Diverging arrow directions (e.g. 1-2 & 1-3) indicate possibly concurrent tasks
* Dotted lines indicate dependent tasks that do not require resources.

Layered PERT Chart



4. Software Configuration Management: WHY?

- The problem:
  - Multiple people have to work on software that is ***changing***
  - ***More than one version*** of the software has to be supported:
    - System(s) under development
    - Custom configured systems (different functionality)
    - Released systems
  - Software must run on different machines and OS

↙ ***Need for Coordination***

- Software Configuration Management (SCM)
  - manages evolving software systems
  - controls the costs involved in making changes to a system

What is SCM?

□ "*SCM is the control of the evolution of complex systems,…, for the purpose to contribute to satisfying quality and delay constraints.*"

– Jacky Estublier

□ "*SCM provides the capabilities of identification, control, status accounting, audit and review, manufacture, process management, and teamwork.*"

– Susan Dart

What is SCM? (cont'd)

- CM is a key process in Capability Maturity Model
  - **Level 1-Initial**: ad hoc/chaotic
  - **Level 2-Repeatable**: basic PM and documentation
  - **Level 3-Defined**: standard and complete process control and procedures
  - **Level 4-Managed**: predictable process performance and precise measurements
  - **Level 5-Optimizing**: continuous and recursive improvement to performance
- CM operates through the software life cycle

What is NOT SCM?

- Not just version control
- Not just for source code management
- Not only for development phase
- Selecting and using tools are important, but design and management of CM process are more crucial for project success

Some Simple CM Scenarios

- Developer A wants to see latest version of foo.c and its change history since last week
- B needs to revert foo-design.doc to its version two days ago
- B makes a release of the project and he needs to know what items to include and which version
- A lives in New Dehli, India and B lives in Boston, US, they want to work on HelloWorld.java together
- In the latest release, a serious bug is found and manager C wants to track what changes caused the bug, who made those changes and when
- C wants to get reports about current project progress to decide if she needs to hire more programmers and delay the alpha release

SCM Roles

- Configuration Manager
  - identify configuration items
  - define the procedures for creating promotions and releases
- Change control board member
  - approving or rejecting change requests
- Developer
  - Creates promotions triggered by change requests or the normal activities of development.
  - checks in changes and resolves conflicts
- Auditor
  - selection and evaluation of promotions for release and for ensuring the consistency and completeness of this release

Terminology

- □ Configuration Item (CI)
- □ Version, Variant, and Revision
- □ Configuration
- □ Baseline
- □ Workspace (Repository)

Configuration Item (CI)

- *"An aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process."*

- Not only program code segments but all type of documents according to development, e.g
  - all type of code files
  - drivers for tests
  - analysis or design documents
  - user or developer manuals
  - system configurations (e.g. version of compiler used)

- In some systems, not only software but also hardware configuration items (CPUs, bus speed frequencies) exist!

Finding CIs

- Large projects typically produce thousands of entities (files, documents, data ...) which must be uniquely identified.
- Any entity can potentially be brought under configuration management control
- But not all the time.

- Two Issues:
  - What: Selection of Configuration Items
    - What should be under configuration control?
  - When: to start to place entities under configuration control?

- Conflict for the Project Manager:
  - Starting with CIs too early introduces too much bureaucracy
  - Starting with CIs too late introduces chaos

Finding CIs (cont'd)

- □ Some items must be maintained for the lifetime of the software.
- □ Sometimes after the software is no longer developed but still in use;
- □ Who expects proper support for lots of years.
- □ An entity naming scheme should be defined

Which of these Entities should be CIs?

- Problem Statement
- Software Project Management Plan (SPMP)
- Requirements Analysis Document (RAD)
- System Design Document (SDD)
- Project Agreement
- Object Design Document (ODD)
- Dynamic Model
- Object model
- Functional Model
- Unit tests
- Integration test strategy
- Source code
- API Specification
- Input data and data bases
- Test plan
- Test data
- Support software (part of the product)
- Support software (not part of the product)
- User manual
- Administrator manual

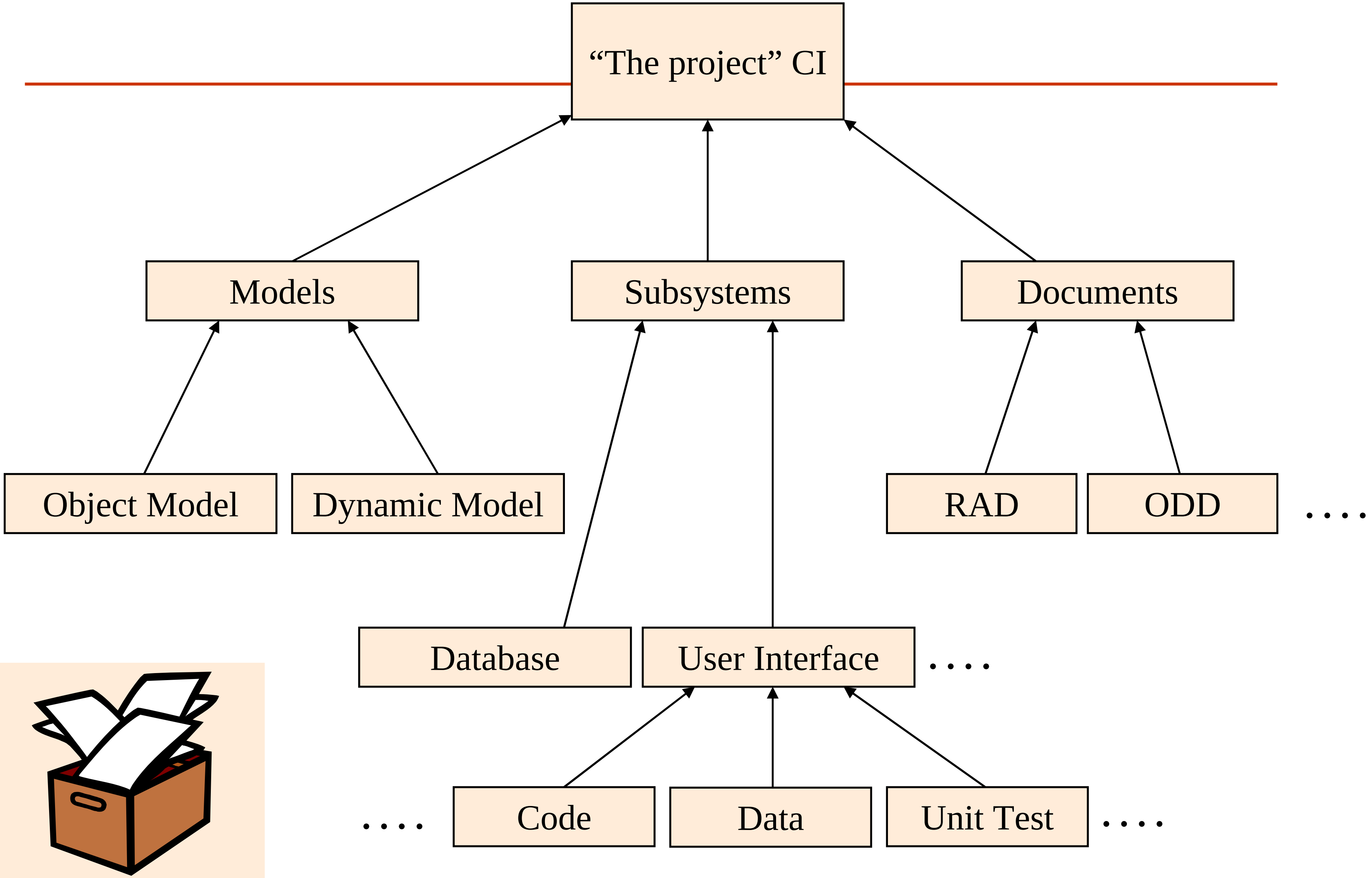
Possible Selection of CIs

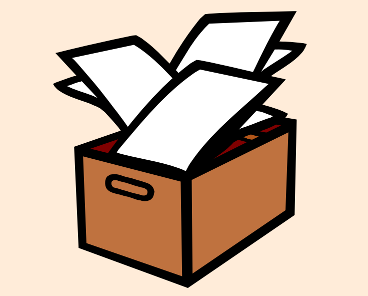
- Problem Statement
- Software Project Management Plan (SPMP)
- Requirements Analysis Document (RAD)
- S
  (
- Project Agreement
- Object Design Document (ODD)
- Dynamic Model
- Object model
- Functional Model
- Unit tests
- Integration test strategy

- Source code
- API Specification
- Input data and databases
- Test plan
- Support software (part of the product)
- Support software (not part of the product)
- User manual
- Administrator manual

Once the Configuration Items are selected, they are usually organized in a tree

CI Tree (Example)





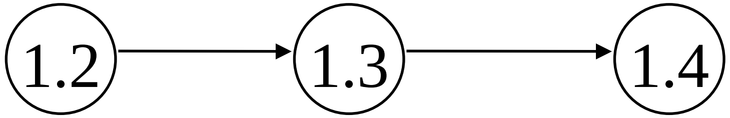
“The project”

Version, Variant, and Revision

- **Version**: a CI at one point in its development, includes revision and variant

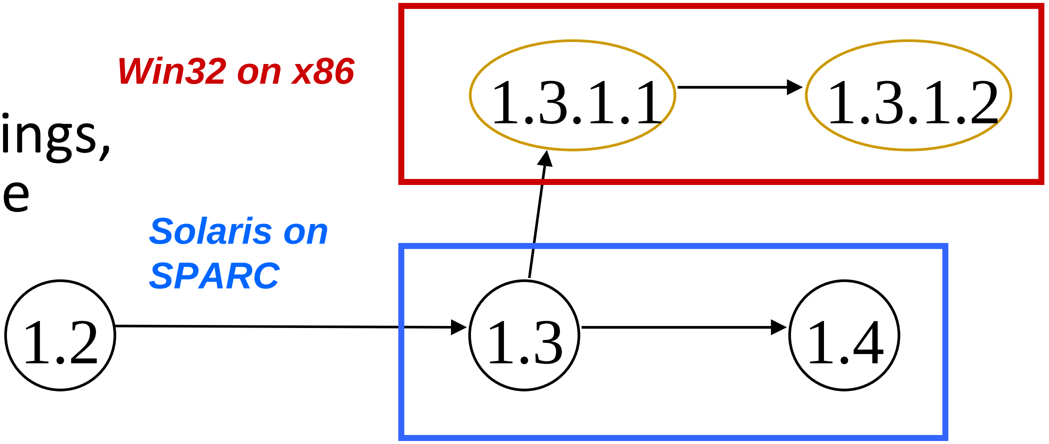
Question: Is Windows 7 a new version or a new revision compared to Windows XP?

- **Revision**: a CI linked to another via revision-of relationship, and ordered in time



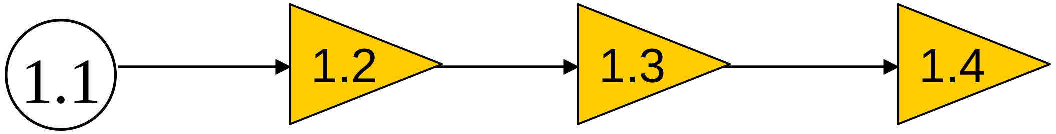
- **Variant**: functionally equivalent versions, but designed for different settings, e.g. hardware and software

- **Branch**: a sequence of versions in the time line

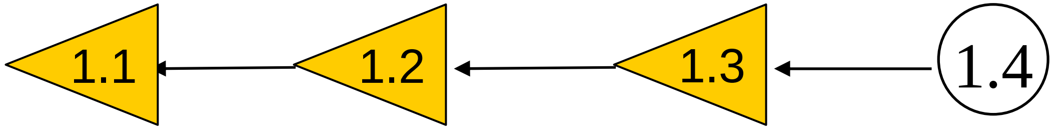


How Versions are Stored

- Full copy of each version
- **Delta** (differences between two versions)
- Forward delta



- Reverse delta



- Mixed delta

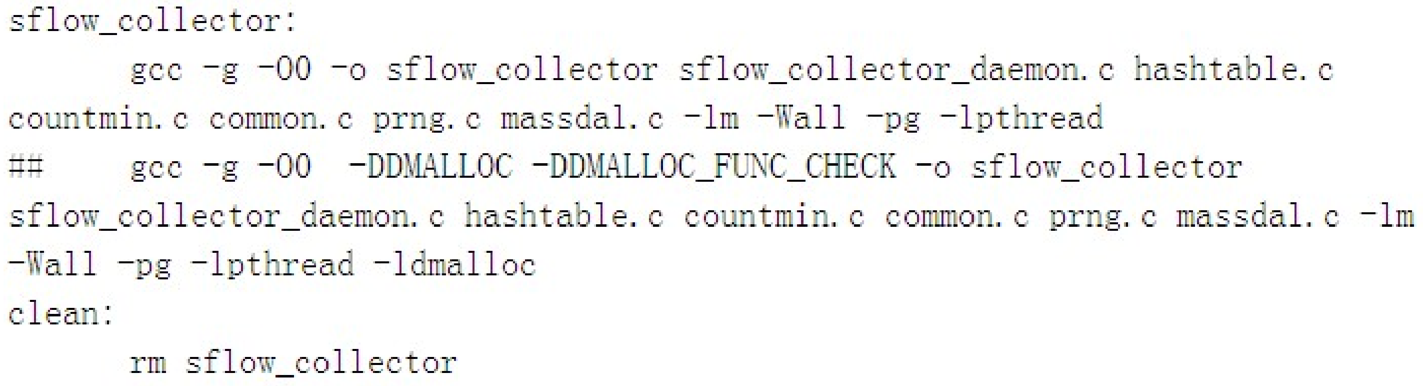
Configuration

- An arrangement of functional CIs according to their nature, version and other characteristics

- Guaranteed to recreate configurations with quality and functional assurance

- Sometimes, configuration needs to record environment details, e.g. compiler version, library version, hardware platform, etc.

- Simple examples
  - Ant buildfile, Makefile



```
sflow_collector:
        gcc -g -O0 -o sflow_collector sflow_collector_daemon.c hashtable.c
countmin.c common.c prng.c massdal.c -lm -Wall -pg -lpthread
##      gcc -g -O0  -DDMALLOC -DDMALLOC_FUNC_CHECK -o sflow_collector
sflow_collector_daemon.c hashtable.c countmin.c common.c prng.c massdal.c -lm
-Wall -pg -lpthread -ldmalloc
clean:
        rm sflow_collector
```

Baseline

- A collection of item versions that have been formally reviewed and agreed on, a version of configuration
- Marks milestones and serves as basis for further development
- Can only be changed via formal change management process
- Baseline + change sets to create new baselines

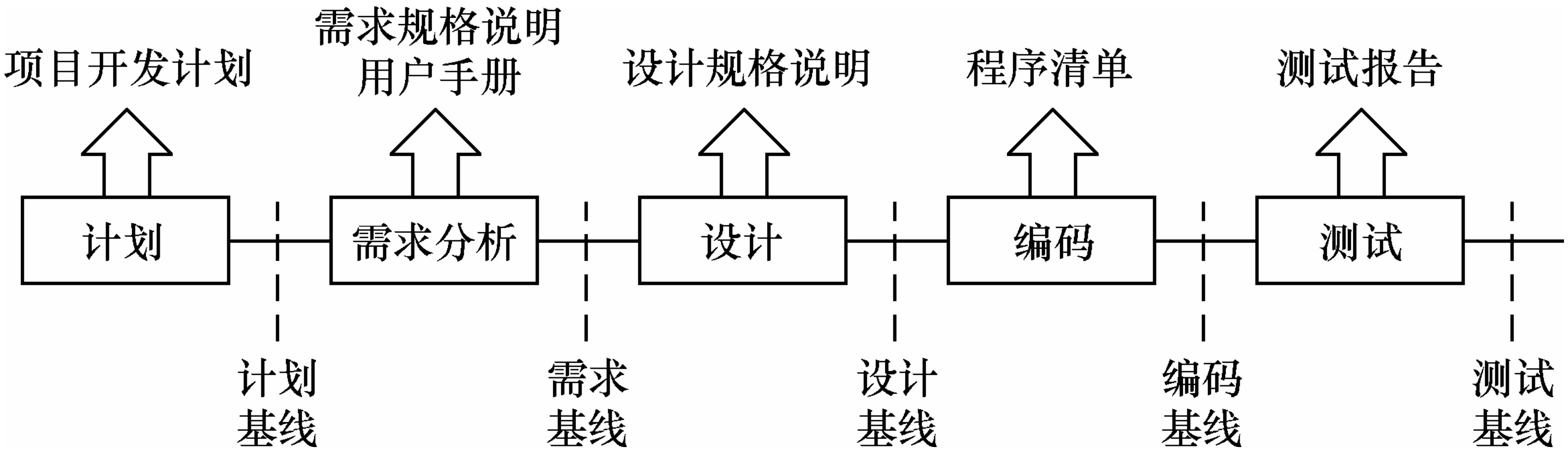
Examples:

Baseline A: All t**he API have completely been defined; the bodies of the methods are empty**.

Baseline B: **All data access methods are implemented and tested**.

Baseline C: **The GUI is implemented.**

Example


项目开发计划
需求规格说明
用户手册
设计规格说明
程序清单
测试报告
计划
需求分析
设计
编码
测试
计划
基线
需求
基线
设计
基线
编码
基线
测试
基线

Workspace

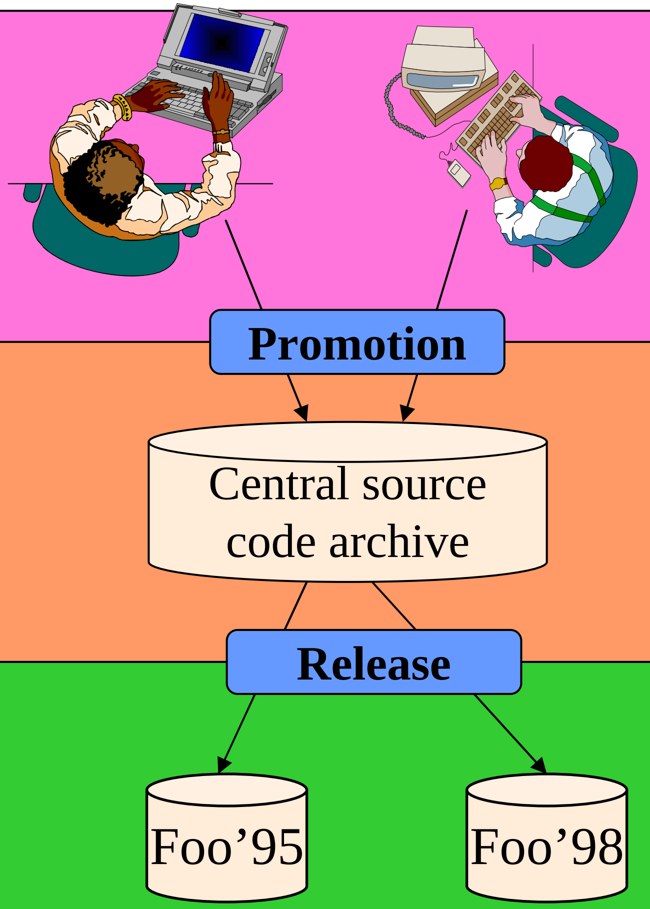
- An isolated environment where a developer can work (edit, change, compile, test) without interfering other developers

- Examples
  - Local directory under version control
  - Private workspace on the server

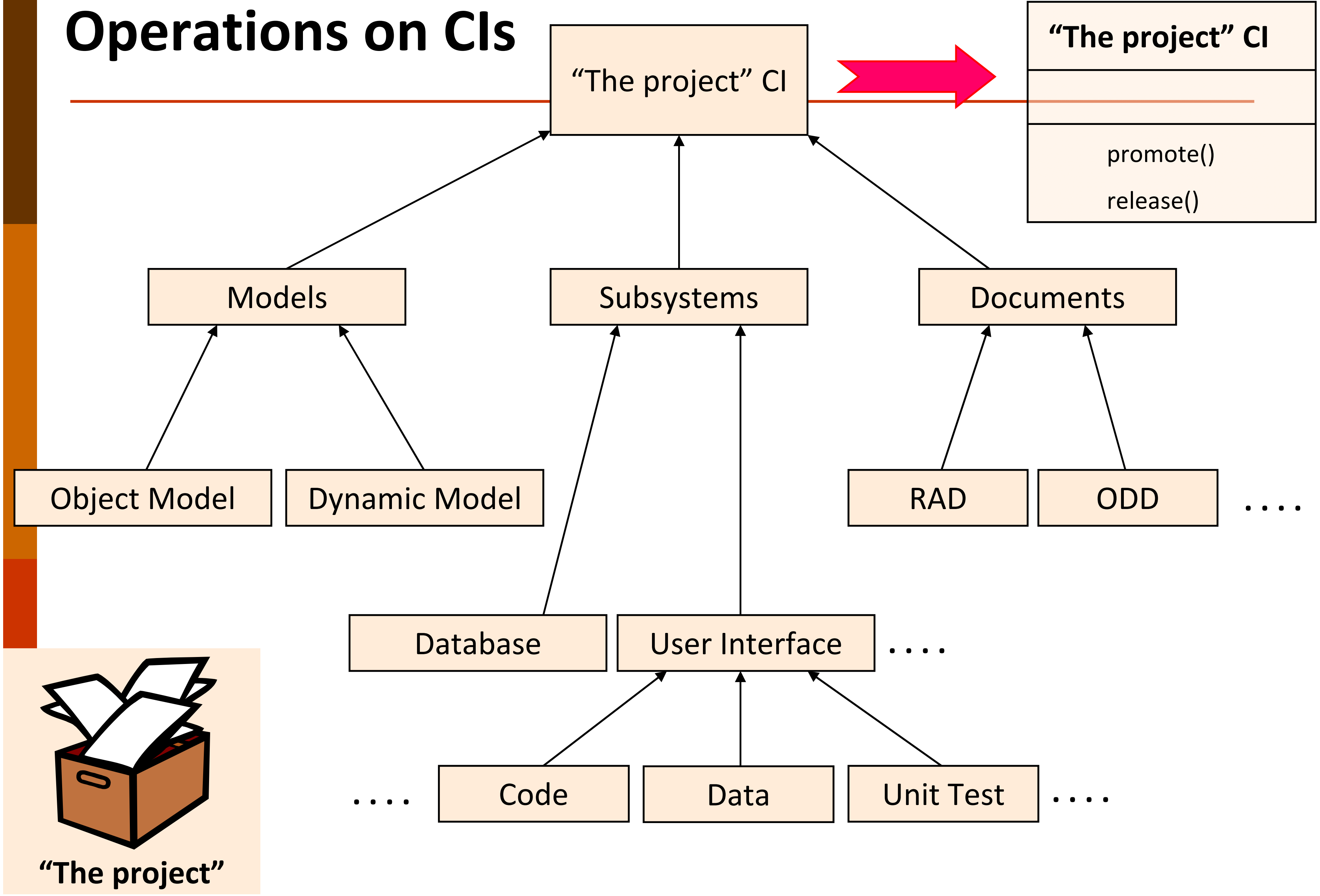
- Common Operations
  - Import: put resources into version control in repository
  - Update: get latest version on the default branch
  - Checkout: get a version into workspace
  - Checkin: commit changes to the repository

Standard SCM Directories

- Programmer's Directory
  - (IEEE Std: "Dynamic Library")
  - Completely under control of one programmer.
- Master Directory
  - (IEEE Std: "Controlled Library")
  - Central directory of all promotions.
- Software Repository
  - (IEEE Std: "Static Library")
  - Externally released baselines.



Promotion and Release are Operations on CIs

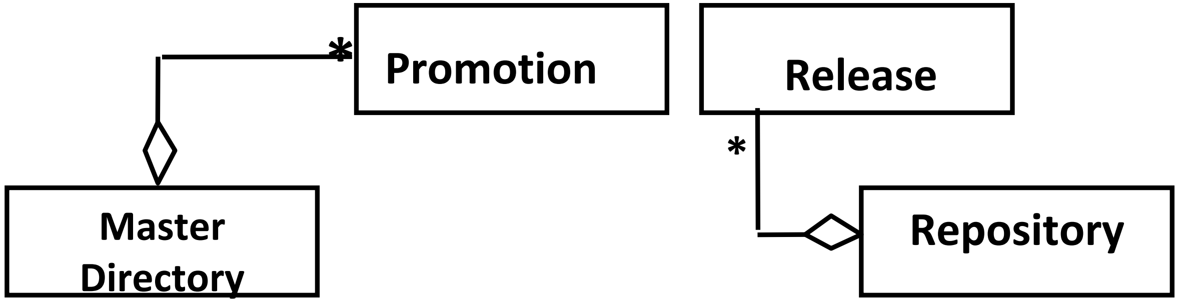


Let‘s Create a Model for Configuration Management

- We just learned that promotions are stored in the master directory and releases are stored in the repository

Problem: There can be many promotions and many releases

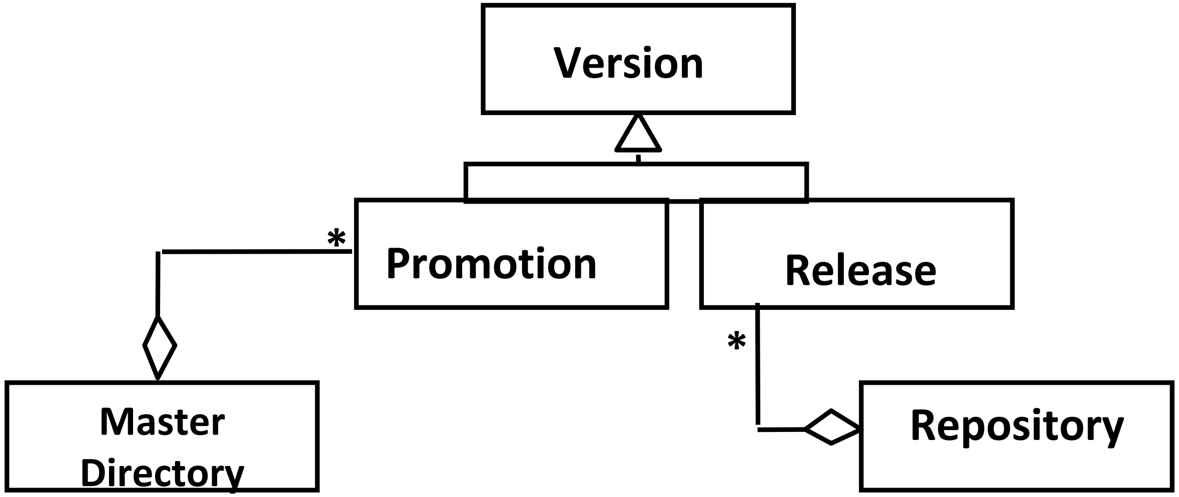
Solution: Use Multiplicity



Let‘s Create a Model for Configuration Management

- Insight: Promotions and Releases are both versions

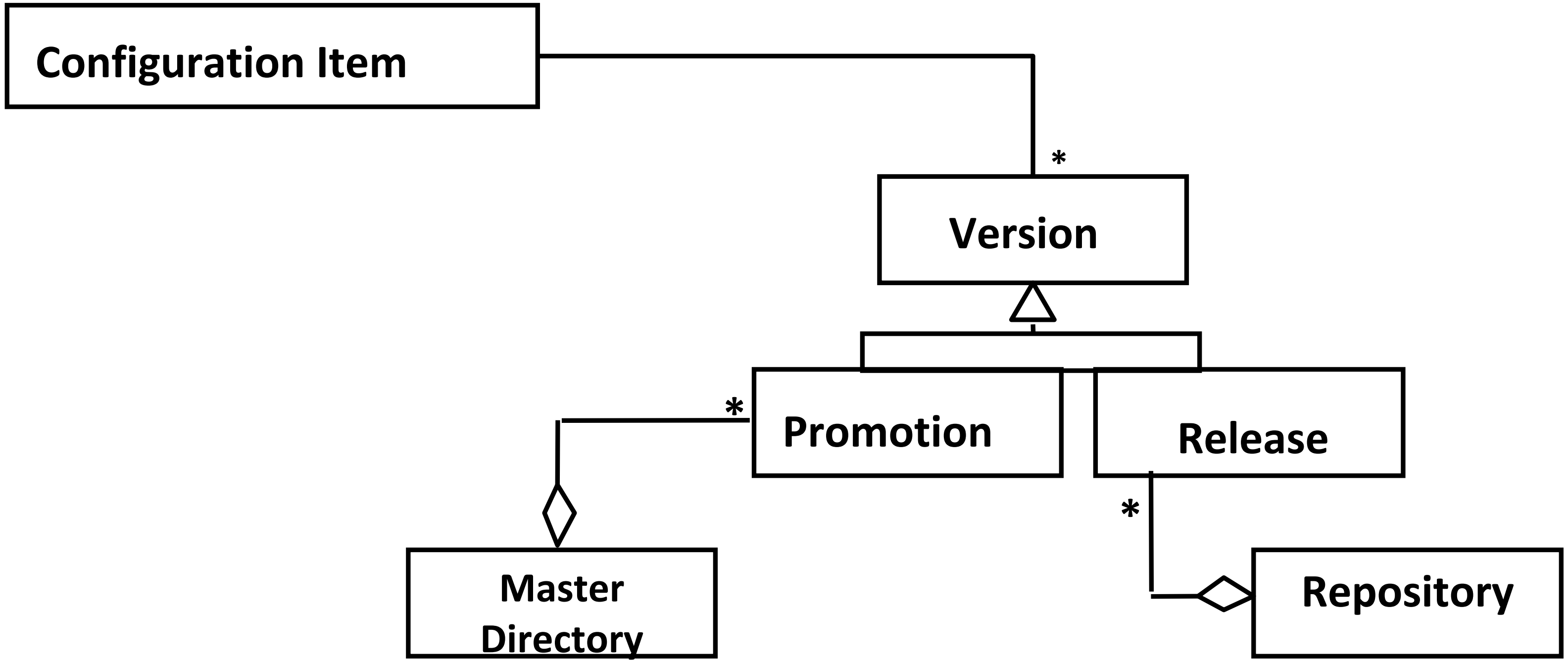
Solution: Use Inheritance



Let‘s Create a Model for Configuration Management

- Problem: A configuration item has many versions

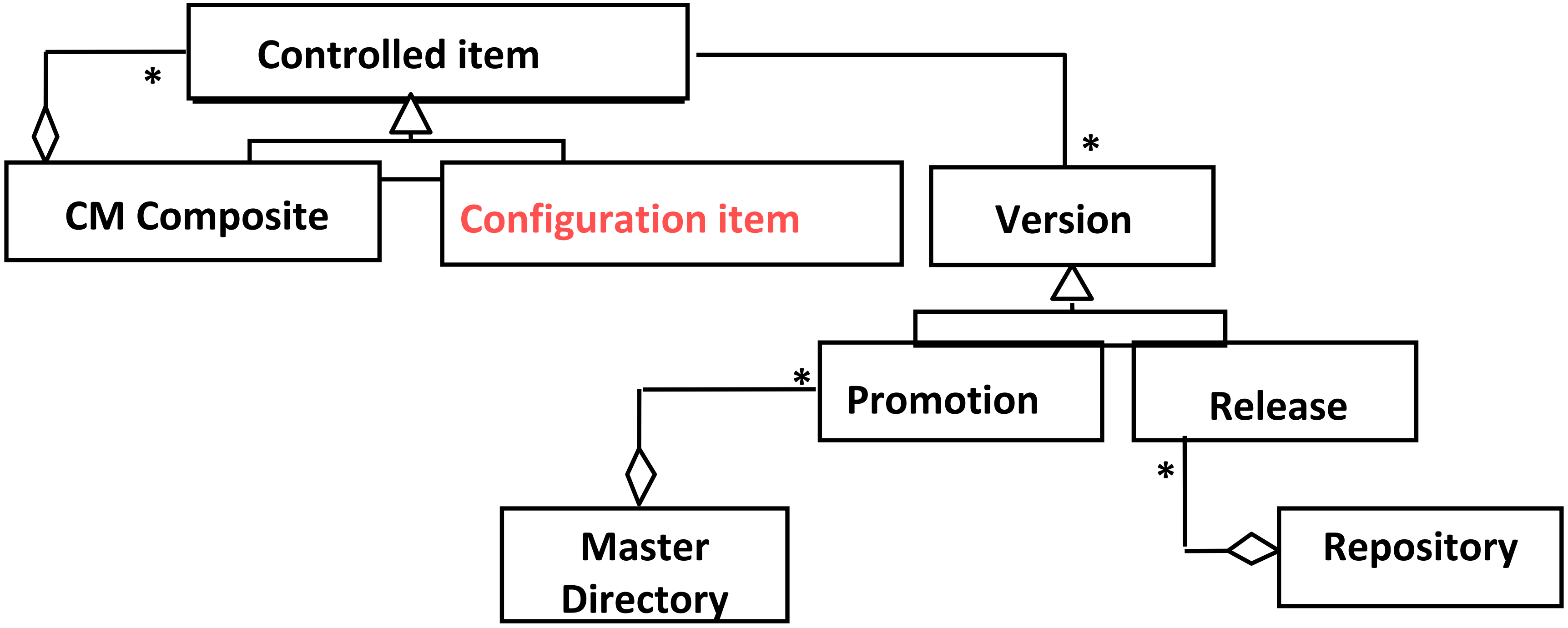
Solution: Create a 1-many association between Configuration Item and Version



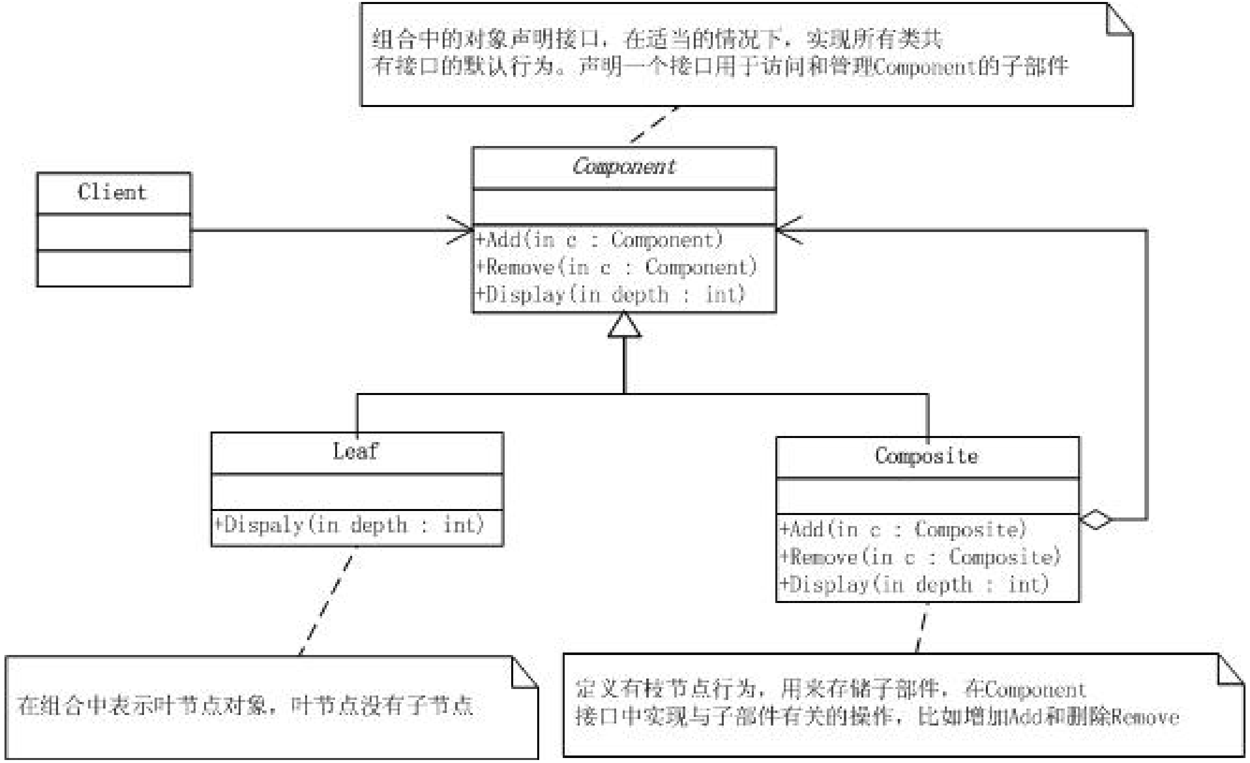
Let's Create a Model for Configuration Management

- Problem: Configuration items can themselves be grouped

  Solution: Use ***Composite Design Pattern***



Revisit: Composite Design Pattern



组合中的对象声明接口，在适当的情况下，实现所有类共有接口的默认行为。声明一个接口用于访问和管理Component的子部件

在组合中表示叶节点对象，叶节点没有子节点

定义有枝节点行为，用来存储子部件，在Component接口中实现与子部件有关的操作，比如增加Add和删除Remove

- Component：组合中的对象声明接口，在适当情况下实现所有类共有的默认行为
- Leaf: 在组合中表示叶节点，叶节点没有子节点，定义对象的基本行为。
- Composite：定义有子部件的那些部件的行为，存储子部件并在Component接口实现与子部件有关的操作。
- Client:通过Component接口操作组合部件的对象。

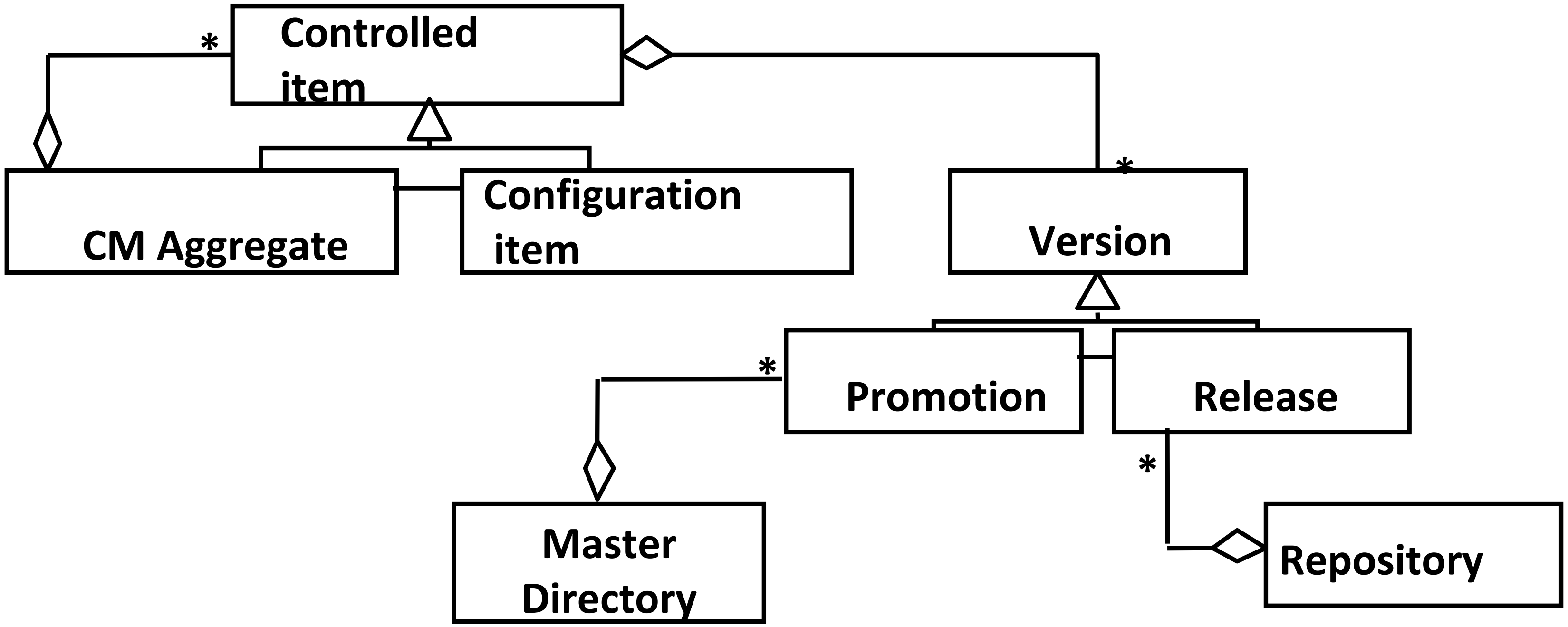
```
public abstract class Component
{
    protected string name;
    public Component(string name)
    {          this.name = name;}

    public abstract void Add(Component c);
    public abstract void Remove(Component c)
    public abstract void Display(int depth);
}
```

```
public class Leaf : Component
{
    public Leaf(string name) : base(name) { }

    public override void Add(Component c)
    {  Console.WriteLine("Cannot add to a leaf"); }

    public override void Remove(Component c)
    { Console.WriteLine("Cannot remove from a leaf"); }

    public override void Display(int depth)
    { Console.WriteLine(new String('-', depth) + name); }
}
```

```
public class Composite : Component
 {
    private List<Component> children = new Li

    public Composite(string name) : base(nam

    public override void Add(Component c)
    {  children.Add(c);}

    public override void Remove(Component c)
    { children.Remove(c);}

    public override void Display(int depth)
    {
        Console.WriteLine(new String('-', depth) + name);

        foreach (Component component in children)
        { component.Display(depth + 2);}
    }  }
```

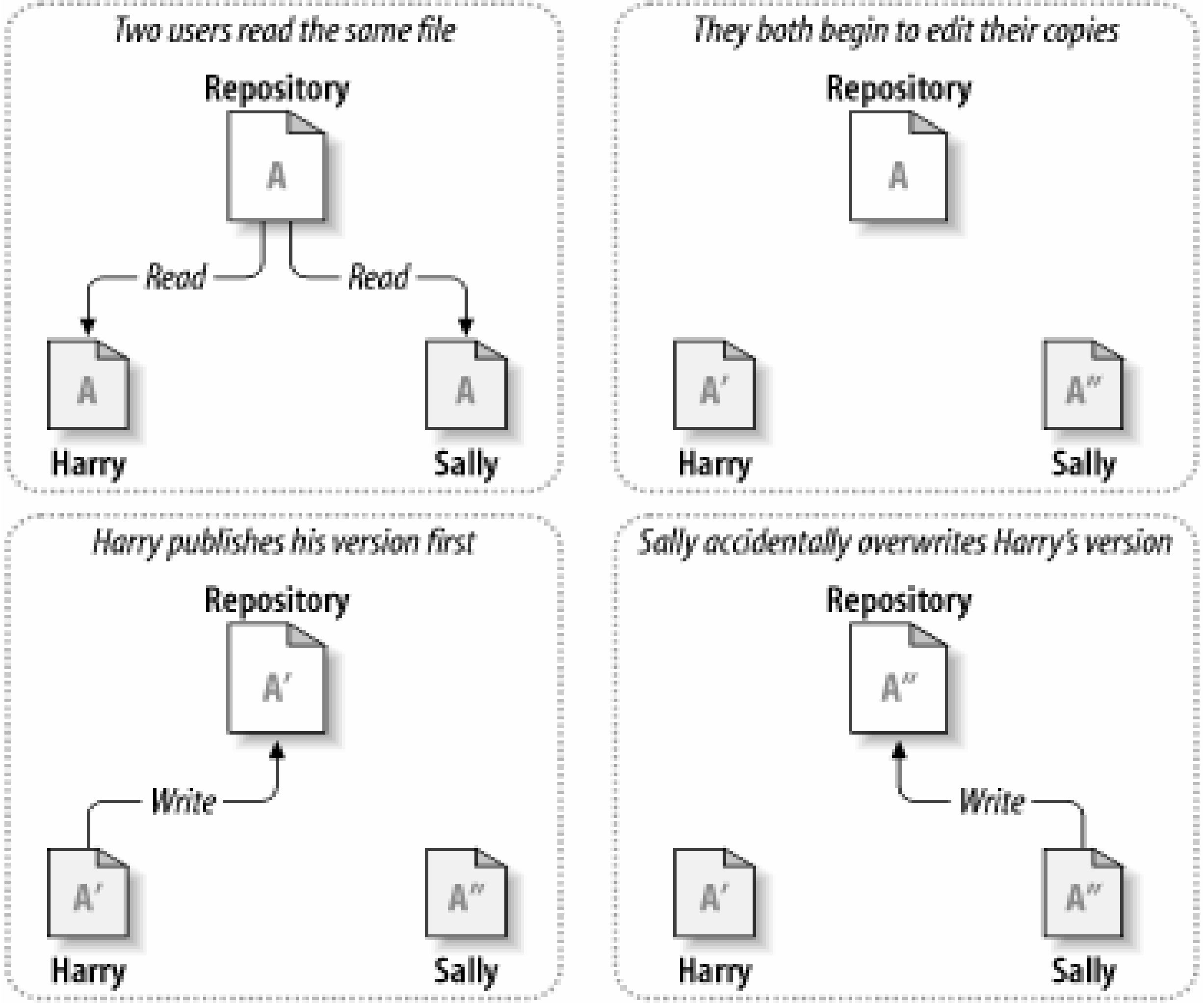
```
public class Program
{
    static void Main(string[] args)
    {
        Composite root = new Composite("root");
        root.Add(new Leaf("Leaf A"));
        root.Add(new Leaf("Leaf B"));

        Composite comp = new Composite("Composite X");
        comp.Add(new Leaf("Leaf XA"));
        comp.Add(new Leaf("Leaf XB"));

        root.Add(comp);

        Composite comp2 = new Composite("Composite XY");
        comp2.Add(new Leaf("Leaf XYA"));
        comp2.Add(new Leaf("Leaf XYB"));

        comp.Add(comp2);

        root.Add(new Leaf("Leaf C"));

        Leaf leaf = new Leaf("Leaf D");
        root.Add(leaf);
        root.Remove(leaf);

        root.Display(1);
        Console.Read();
    }    }
```

CI Model (UML Class Diagram)



Version Control Models (1/3)

- Basic problem of collaborative work

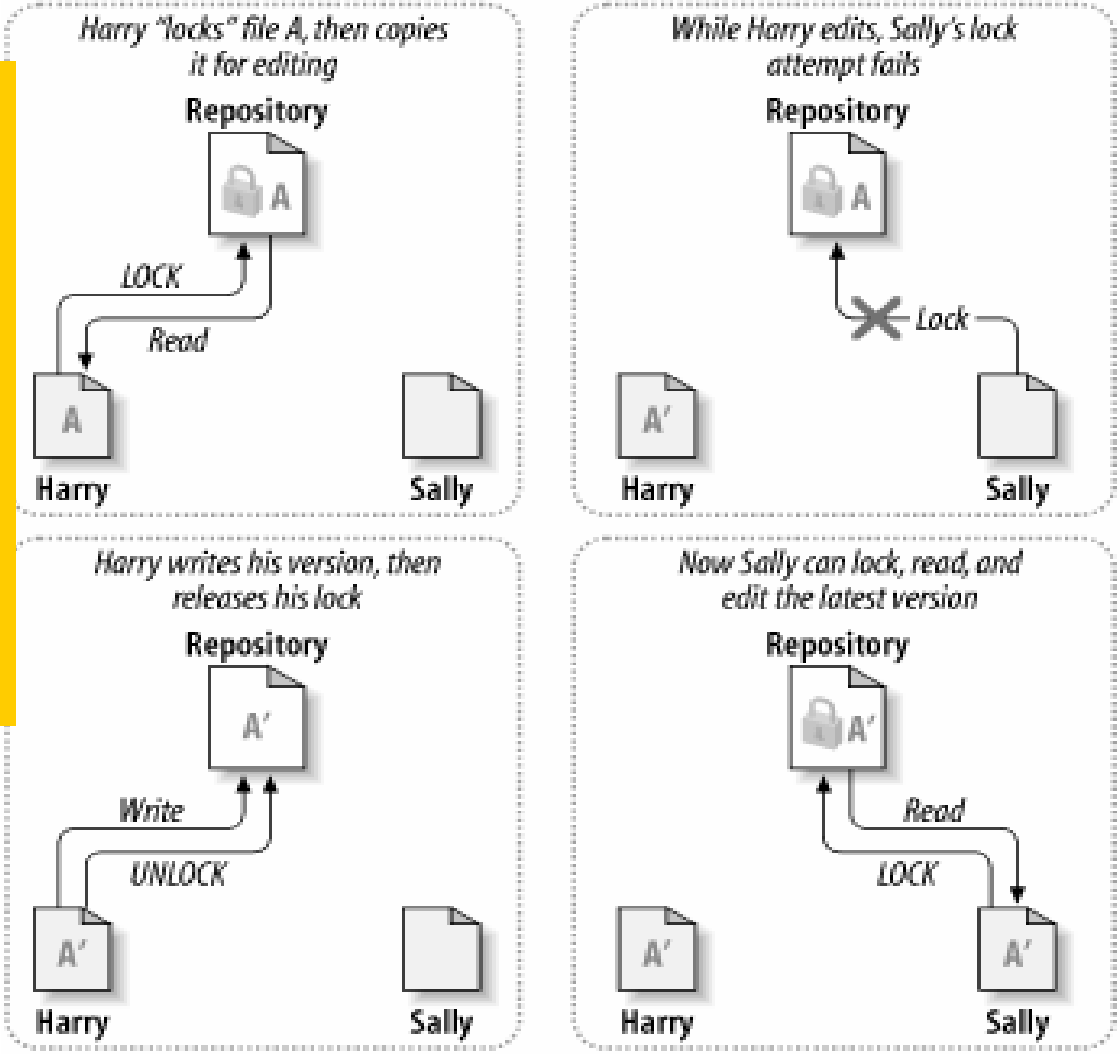

Two users read the same file
Repository
A
Read
Read
A
A
Harry
Sally
They both begin to edit their copies
Repository
A
A′
A″
Harry
Sally
Harry publishes his version first
Repository
A′
Write
A′
A″
Harry
Sally
Sally accidentally overwrites Harry's version
Repository
A″
Write
A′
A″
Harry
Sally

Version Control Models (2/3)

- Model 1-Pessimistic: lock-modify-unlock

Problems:

- Forget to unlock
- Parallel work not possible
- Deadlock


Harry "locks" file A, then copies it for editing
Repository
A
LOCK
Read
A
Harry
Sally
While Harry edits, Sally's lock attempt fails
Repository
A
Lock
A'
Harry
Sally
Harry writes his version, then releases his lock
Repository
A'
Write
UNLOCK
A'
Harry
Sally
Now Sally can lock, read, and edit the latest version
Repository
A'
Read
LOCK
A'
Harry
A'
Sally

svn-book

Version Control Models (3/3)

Model 2-Optimistic: copy-modify-merge

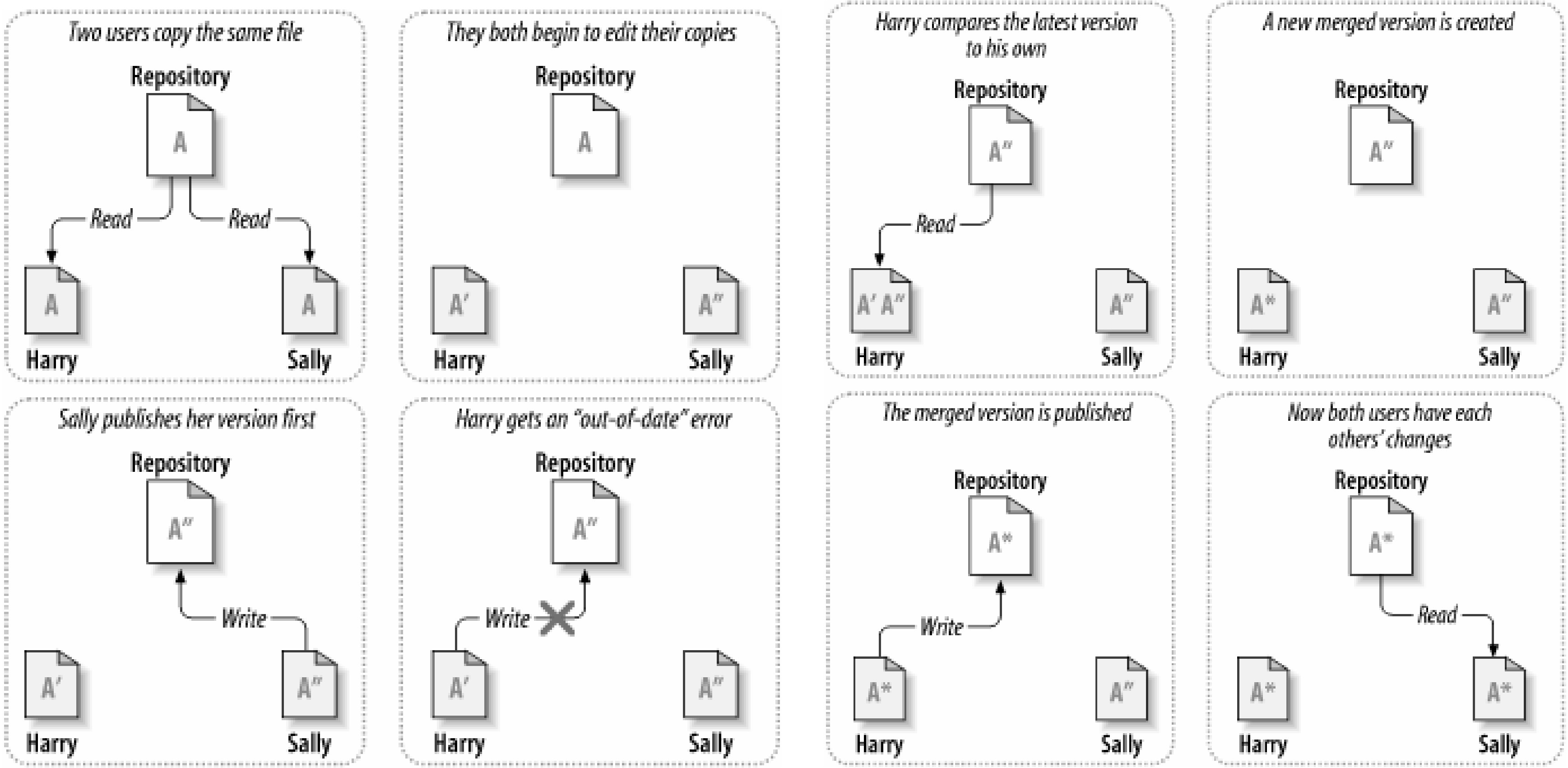

Two users copy the same file
Repository
A
Read
Read
A
A
Harry
Sally
They both begin to edit their copies
Repository
A
A′
A″
Harry
Sally
Harry compares the latest version to his own
Repository
A″
Read
A′ A″
A″
Harry
Sally
A new merged version is created
Repository
A″
A*
A″
Harry
Sally
Sally publishes her version first
Repository
A″
Write
A′
A″
Harry
Sally
Harry gets an "out-of-date" error
Repository
A″
Write
A′
A″
Harry
Sally
The merged version is published
Repository
A*
Write
A*
A″
Harry
Sally
Now both users have each others' changes
Repository
A*
Read
A*
A*
Harry
Sally

Figure from svn-book

SCM Processes

- □ Change control process
- □ Status accounting
- □ Configuration audit
- □ Release management
- □ CM planning

Change Control Process

- Submission of Change Request (CR)
- Technical and business evaluation and impact analysis
- Approval by Change Control Board (CCB)
- Engineering Change Order (ECO) is generated stating
  - changes to be made
  - criteria for reviewing the changed CI
- CI’s checked out
- Changes made and reviewed
- CI’s checked in

Status Accounting

- Administrative tracking and reporting of CIs in CM system

- Examples
  - Status of proposed changes
  - Status of approved changes
  - Progress of current version, on or behind schedule
  - Estimate of resources to finish one task
  - bugs identified by configuration audit

Configuration Audit

- Independent review or examination to assess if a product or process is in compliance with specification, standards, contractual agreement, or other criteria
- Examples
  - Verifies that CIs are tested to satisfy functional requirements
  - Verifies that baseline contains necessary and correct CI versions
  - Ensures that changes made to a baseline comply with the configuration status report

Release Management

- Creation and availability of a new version of software to the public

- Release format
  - Source code + build script + instructions
  - Executables packaged for specific platforms
  - Other portable formats: Java Web Start, plugins
  - Patches and updates: automatic, manual

- Release content
  - Source and/or binary, data files, installation scripts, libraries, user and/or developer documentation, feedback programs, etc.

Software CM Planning

- Software configuration management planning starts during the early phases of a project.

- The outcome of the SCM planning phase is the

  Software Configuration Management Plan (SCMP)

  which might be extended or revised during the rest of the project.

- Either follow a public standard like the IEEE 828, or an internal (e.g. company specific) standard.

SCMP

- □ Defines the *types of documents* to be managed and a document naming scheme.
- □ Defines *who takes responsibility* for the CM procedures and creation of baselines.
- □ Defines *policies for change* control and version management.
- □ Describes the *tools* which should be used to assist the CM process and any limitations on their use.
- □ Defines the *configuration management database* used to record configuration information.

Outline of a SCMP by IEEE 828-1990

1. Introduction
   - Describes purpose, scope of application, key terms and references
2. Management (WHO?)
   - Identifies the responsibilities and authorities for accomplishing the planned configuration management activities
3. Activities (WHAT?)
   - Identifies the activities to be performed in applying to the project.
4. Schedule (WHEN?)
   - Establishes the sequence and coordination of the SCM activities with project mile stones.
5. Resources (HOW?)
   - Identifies tools and techniques required for the implementation of the SCMP
6. Maintenance
   - Identifies activities and responsibilities on how the SCMP will be kept current during the life-cycle of the project.

SCMP Section 1: Introduction

- Simplified overview of the CM activities.
- Scope:
  - Overview description of the project
  - Identification of the CI(s) to which software CM will be applied.
- Identification of other software to be included as part of the SCMP (support software and test software)
- Relationship of SCM to hardware of system CM activities
- Degree of formality and depth of control for applying SCM to project.
- Limitations and time constraints for applying SCM to this project
- Assumptions that might have an impact on the cost, schedule and ability to perform defined SCM activities.

SCMP Section 2: Management

- Organization
  - Organizational context (technical and managerial) within which the SCM activities are implemented. Identifies
    - All organizational units (client, developers, managers) that participate in a SCM activity
    - Functional roles of these people within the project
    - Relationship between organizational units
- Responsibilities
  - For each SCM activity list the name or job title to perform this activity
  - For each board performing SCM activities, list
    - purpose and objectives
    - membership and affiliations
    - period of effectivity, scope of authority
    - operational procedures
- Applicable Policies
  - External constraints placed on the SCMP

SCMP Section 3: Activities

3.1 Configuration Identification

3.2 Configuration Control

3.3 Configuration Status Accounting

3.4 Configuration Audits and Reviews

3.5 Interface Control

3.2 Configuration Control

Defines the following steps

- 3.2.1 How to identify the need for a change (layout of change request form)
- 3.2.2 Analysis and evaluation of a change request (CR)
- 3.2.3 Approval or disapproval of a request
- 3.2.4 Verification, implementation and release of a change

3.2.1 Change Request

- Specifies the procedures for requesting a change to a baselined CI and the information to be documented:
  - Name(s) and version(s) of the CI(s) where the problem appears
  - Originator’s name and address
  - Date of request
  - Indication of urgency
  - The need for the change
  - Description of the requested change

3.2.2 Evaluation of a Change

- Specifies the analysis required to determine the impact of proposed changes and the procedure for reviewing the results of the analysis.

3.2.3 Change Approval or Disapproval

- This section of the SCMP describes the organization of the configuration control board (CCB).
- Configuration Control Board (CCB)
    - Can be an individual or a group.
    - Multiple levels of CCBs are also possible, depending on the complexity of the project
- Multiple levels of CCBs may be specified.
    - In small development efforts one CCB level is sufficient.
- This section of the SCMP also indicates the level of authority of the CCB and its responsibility.
    - In particular, the SCMP must specify when the CCB is invoked.

3.2.4 Implementing Change

- Specifies the activities for verifying and implementing an approved change.
- A completed change request must contain:
  - The original change request(s)
  - The names and versions of the affected configuration items
  - Verification date and responsible party
  - Identifier of the new version
  - Release or installation date and responsible party
- This section must also specify activities for
  - Archiving completed change requests
  - Planning and control of releases
  - How to coordinate multiple changes
  - How to add new CIs to the configuration
  - How to deliver a new baseline

3.3 Configuration Status Accounting

- What elements are to be tracked and reported for baselines and changes?
- What types of status accounting reports are to be generated? What is their frequency?
- How is information to be collected, stored and reported?
- How is access to the configuration management status data controlled?

3.4 Configuration Audits and Reviews

- Identifies audits and reviews for the project.
  - An audit determines for each CI if it has the required physical and functional characteristics.
  - A review is a management tool for establishing a baseline.

- For each audit or review the plan has to define:
  - Objective
  - The Configuration Items under review
  - The schedule for the review
  - Procedures for conducting the review
  - Participants by job title
  - Required documentation
  - Procedure for recording deficiencies and how to correct them
  - Approval criteria

Summary of SCMP

- Standards
  - IEEE Std 828 (SCM Plans), ANSI-IEEE Std 1042 (SCM), etc.
- CM plan components
  - What will be managed (list and organize CIs)
  - Who will be responsible for what activities (roles and tasks)
  - How to make it happen (design processes for change requests, task dispatching, monitoring, testing, release, etc.)
  - What records to keep (logs, notes, configurations, changes, etc.)
  - What resources and how many (tools, money, manpower, etc.)
  - What metrics to measure progress and success

CM Tools

- Version control
  - RCS, CVS, Subversion, Visual Source Safe, Rational ClearCase
- Bug tracking
  - Bugzilla, Mantis Bugtracker, Rational ClearQuest
- Build
  - GNU Make and many variants, Ant
- Project management
  - Sourceforge.net, freshmeat.net, GForge, DForge

Reference and Further Readings

Reference

- *Introduction to Configuration Management*, lecture slides for COMP3100/3500, Ian Barnes, the Australian National University.
- *Software Configuration Management*, Center for Development of Advanced Computing, Mumbai at Juhu, India.
- *Concepts in Configuration Management Systems*, Susan Dart, CMU.
- *Software Configuration Management: A Roadmap*, Jacky Estublier, CNRS, France.

Further Reading

- *Software Engineering, a Practitioner's Approach (6th), part 4*, Roger Pressman.
- *Code Complete (2nd)*, Steve McConnel.
- *http://cmcrossroads.com/*
- *Implementing and Integrating PDM and SCM*, Ivica Crnkovic et al.
- *Version Control with Subversion*, Ben Collins-Sussman et al.