Review: Introduction to Software Engineering

Unit 0 Overview

0. History of computer technology

The raise of Software Engineering

1. The Nature of Software

⭐What is Software?

Software is a set of items or objects that form a “configuration” that includes

- programs
- documents
- data structure

Traditional Programing

Software = Algorithms + data structure

OO Programing

Software = Objects + Messages

Component Oriented Programming

Software = Components + Architecture

⭐The nature of software

> Tip
> 软硬件开发的区别是什么?

Hardware	Software
Manufactured	Developed/engineered
Wears out	Deteriorates
Built using components	Custom built
Relatively simple	Complex

Manufacturing vs. Development

Software products are routinely modified and upgraded.
Software costs are concentrated in design rather than production.

Wears out vs. Deterioration

Software deteriorates over time.

Component Based vs. Custom Built

Most software continues to be custom built.
The software industry does move toward component-based construction.
SaaS(Software-as-a-Service)

Complex

Cloud computing

Cloud computing provides distributed data storage and processing resources to networked computing devices.
Computing resources reside outside the cloud and have access to a variety of resources inside the

cloud.
Cloud computing requires developing an architecture containing both frontend and backend services.

2. Concepts of SE

⭐Software crisis

⭐Software Process

⭐Why we need software processes?

⭐Software Lifecycle

Umbrella / Framework Activities

Unit 1 Software Processes

3. Software Process Structure

Process Patterns

A process pattern

- describes a process-related problem that is encountered during software engineering work,
- identifies the environment in which the problem has been encountered, and
- suggests one or more proven solutions to the problem.

Stated in more general terms, a process pattern provides you with a template — a consistent method for describing problem solutions within the context of the software process.

Process Pattern Types

- **Stage patterns**—defines a problem associated with a framework activity for the process.
- **Task patterns**—defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice
- **Phase patterns**—define the sequence of framework activities that occur with the process, even when the overall flow of activities is iterative in nature.

⭐CMMI

CMMI (Capability Maturity Model Integration)

- A process meta-model proposed by the SEI (Software Engineering Institute).
- CMMI is a collection of best practices that meet the needs of organizations in different areas of interest.

CMMI Stages

1. Initial
2. Managed
3. Defined
4. Quantitatively Managed
5. Optimized

Adapting a Process Model

Tip
为什么要进行过程模型裁剪？如何裁剪？

The tasks (and degree of rigor) for each activity will vary based on:

- the type of project
- characteristics of the project
- common sense judgment; concurrence of the project team

4. Process Models

⭐Prescriptive Process Model

A Prescriptive Process Model defines a distinct set of :

- Framework activities
- SE actions
- Tasks
- Work products
- Quality assurance
- Change Control

  Which are required to engineer high-qualitysoftware

Note: The Prescriptive Process Models are useful but not perfect!

The Framework Activities of a Prescriptive Process Model

- **Communication**: Application Requirements determination，Software Requirements specification
- **Planning**: Planning，estimation，scheduling
- **Modeling**: Architectural design，Detailed design
- **Construction**: Implementation，Integration，Testing
- **Deployment**

⭐The Waterfall Model & the V Model

The Waterfall Model

- Communication: project initiation, requirements gatherings
- Planning: estimating, scheduling, tracking
- Modeling: analysis, design
- Construction: code, test
- Deployment: delivery, support, feedback

Also called “The classic lifecycle”

Although the original waterfall model is added “feedback loops”, the vast majority of organizations treat it as if it was strictly linear.

Characteristics

- Accept the work product (work product) of the activity from the previous activity as input;
- Implement the activity on the basis of this input;
- The work product of this activity will be transmitted to the next activity;
- Before the transmission, the work product will be reviewed and confirmed.

优点：

- 强调开发的阶段性；
- 强调早期计划及需求调查；

- 强调产品测试。

缺点：

- 过于依赖早期进行的唯一一次需求调查，不能适应需求的变化；
- 是单一流程，开发中的经验教训不能反馈应用于本产品的过程。

The V-Model (A variant of the waterfall model

⭐Evolutionary Models

- Business requirements often change
- Tight market deadlines, only allow a limited version
- Only core requirements are defined

Communication → Quick plan → Modeling Quick design → Construction of prototype → Deployment delivery & feedback → Communication → ...

- Modern computer software is characterized by continual change, by very tight timelines, and by an emphatic need for customer/user satisfaction.
- The intent of evolutionary models is to develop high-quality software in an interactive or incremental manner.
- It is possible to use an evolutionary process to emphasize flexibility, extensibility, and speed of development.

⭐Prototyping

- 探索型 (exploratory prototyping) 其目的是要弄清目标系统的要求，确定所希望 的特性，并探讨多种方案的可行性 。
- 实验型 (experimental prototyping) 其目的是验证方案或算法的合理性，它是在大规模开发和实现前，用于考核方案是否合适， 规格说明是否可靠。
- 演化型 (evolutionary prototyping) 其目的是将原型作为目标系统的一部分，通过 对原型的多次改进，逐步将原型演化成最终的 目标系统。

原型 (prototype) 特征：

软件原型是软件的最初版本，以最少的费用、最短的时间开发出的、反映最后软件的主要特征的系统。它具有以下特征：

1. 它是一个可实际运行的系统。
2. 它没有固定的生存期。它可能被扔掉，或者作为最终产品的一部分。
3. 从需求分析到最终产品都可作原型，即可为不同目标作原型。
4. 它必须快速、廉价。
5. 它是迭代过程的集成部分，即每次经用户评价后修改、运行，不断重复双方认可。

Advantages of the prototyping

1. It can help guide high-quality product requirements.
2. Relevant data of the project can be obtained at theearly stage of the project. Risk management andconfiguration management can be carried out assoon as possible.
3. It can encourage the developers.
4. Users can participate in the verification and providevaluable feedback.
5. It can make the sales work possible to be carried out in advance。

Disadvantages of the prototyping

1. Without strict process management, prototypingis likely to degenerate into an original unplanned"try-error-fix" cycle mode.
2. Psychologically, there may be an idea of “although we can't complete all functions, we have created a product with some functions. ”

3. If users are exposed to unstable functionswithout control, it may have a negative impact on developers and users.

Although prototyping can be used as astandalone process model, it is more commonly used as a technique that can be implemented within the context of any one of the process models noted in this chapter.

⭐The Spiral

Using the spiral model, software is developed in a series of evolutionary releases.

- The early release might be a paper model or prototype.
- The latter release can be more complete versions.

Unlike other process models that end when software is delivered, the spiral model can be adapted to apply throughout the life of the software.

在螺旋模型结构中，维护只是螺旋模型的另一个周期，在维护和开发之间本质上并没有区别，从而解决了做太多测试或未作足够测试所带来的风险。

优点：

1. a.强调严格的全过程风险管理。
2. b.强调各开发阶段的质量。
3. c.提供机会检讨项目是否有价值继续下去。

   缺点：

   必须引入非常严格的风险识别，风险分析和风险控制，这对风险管理的技能水平提出了很高的要求。也需要人员，资金和时间的大量投入。

⭐Concurrent

It defines a series of events that will trigger transitions from state to state for each of the software engineering activities, actions, or tasks

⭐The Incremental Model

- 增量模型又称产品改进模型
- 从给定需求开始，通过构造一系列中间版本来实施开发活动，依次类推，直到系统完成。
- 每一个中间版本都是需求分析、设计、编码和测试的过程。
- 某些中间版本的开发可以并行进行。

增量模型特征

- 融合了线性顺序模型的基本成分和原型的迭代特征。
- 是随着日程时间的进展而交错的线性序列。
- 与原型不一样的地方是强调每个增量均发布一个可操作产品。
- 最典型的应用是同一个产品的不同项目 (合同、用户) 版本

增量模型特别适用于：

- 需求经常变化的软件开发
- 市场急需而开发人员和资金不能在设定的市场期限之前实现一个完善的产品的软件开发
  增量模型能有计划地管理技术风险，如早期增量版本中避免采用尚未成熟的技术

Other Process Models

- **Component based development**—the process to apply when reuse is a development objective
- **Formal methods**—emphasizes the mathematical specification of requirements
- **AOSD**—provides a process and methodological approach for defining, specifying, designing, and constructing aspects

⭐The Unified Process (UP)

- Use-case driven
- Architecture-centric
- Iterative and incremental
- Software process closely aligned with the Unified Modeling Language (UML)

RUP 的时间轴被分解为四个顺序的阶段：

- 初始阶段 (Inception) 为系统建立业务案例并确定项目的边界。
- 细化阶段 (Elaboration) 分析问题领域，建立健全的体系结构基础，编制项目计划，淘汰项目中最高风险的元素。
- 构造阶段 (Construction) 所有剩余的构件和应用程序功能被开发并集成为产品，所有的功能被详细测试。
- 交付阶段 (Transition) 确保软件对最终用户是可用的。

RUP 的核心工作流：分为 6 个核心过程工作流 (Core Process Workflows) 和 3 个核心支持工作流 (Core Supporting Workflows)。

- 核心过程工作流：业务建模，需求工作流，分析和设计工作流，实现，测试，部署
- 核心支持工作流：配置和变更管理，软件项目管理，环境

RUP 的优点：

- 提高了团队生产力，在迭代的开发过程、需求管理、基于组件的体系结构、可视化软件建模、验证软件质量及控制软件变更等方面，针对所有关键的开发活动为每个开发成员提供了必要的准则、 模板和工具指导，并确保全体成员共享相同的知识基础。它建立了简洁和清晰的过程结构，为开发过程提供较大的通用性。

RUP 的缺点

- RUP 在理论上，是比较理想的，但在实际应用上， 还需要更多的工具的支持和普及推广工作。

5. Agile Development

⭐Manifesto for Agile Software Development

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions over processes and tools** 个体和互动 高于 流程和工具
- **Working software over comprehensive documentation** 工作的软件 高于 详尽的文档
- **Customer collaboration over contract negotiation** 客户合作 高于 合同谈判
- **Responding to change over following a plan** 响应变化 高于 遵循计划

  That is, while there is value in the items on the right, we value the items on the left more.”

Agility and the Cost of Change

见课件的图

⭐An Agile Process

- Is driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities
- Delivers multiple ‘software increments’
- Adapts as changes occur

Agility Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face–to–face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self–organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

价值驱动 - 敏捷与传统瀑布型模式的最大区别

- 敏捷基于这种方式，可以实现研发过程的持续高可视性、高可适应性，更早且持续产出业务价值，更早发现和解决风险。

⭐Extreme Programming (XP) / Scrum/ Kanban

Extreme Programming (XP)

- **XP Planning**
  - Begins with the creation of user stories
  - Agile team assesses each story and assigns a cost
  - Stories are grouped to for a deliverable increment
  - A commitment is made on delivery date
  - After the first increment project velocity is used to help define subsequent delivery dates for other increments
- **XP Design**
  - Follows the KIS principle(Keep It Simple)
  - Encourage the use of CRC cards
  - For difficult design problems, suggests the creation of spike solutions — a design prototype
  - Encourages refactoring — an iterative refinement of the internal program design
- **XP Coding**
  - Recommends the construction of a unit test for a story before coding commences (测试驱动编程)
  - Encourages pair programming(结对编程)
- **XP Testing**
  - All unit tests are executed daily(强调 Daily Building，冒烟测试)
  - Acceptance tests are defined by the customer and executed to assess customer visible functionality(强调验收测试，回归测试)

Scrum

Scrum—distinguishing features

- Development work is partitioned into “packets”
- Testing and documentation are on-going as the product is constructed
- Work occurs in “sprints” and is derived from a “backlog” of existing requirements
- Meetings are very short and sometimes conducted without chairs
- “demos” are delivered to the customer with the time-box allocated

Scrum 是一个轻量级的项目管理的框架，它的核心在于迭代。
Scrum 团队模型
三种角色：Scrum Master，Product Owner，团队
三种工件：产品 Backlog，Sprint Backlog，产品增量
过程模型：5 个活动 +1 个合约

敏捷开发管理方法 - 看板

看板使得项目管理最大的可视化，但是看板更可以将研发的过程进行管理，记录下用户故事研发过程中的细节和历程。

6. Human Aspects of Software Engineering

Traits of Successful Software Engineers:

- Sense of individual responsibility
- Acutely aware of the needs of team members and stakeholders
- Brutally honest about design flaws and offers constructive criticism
- Resilient under pressure
- Heightened sense of fairness
- Attention to detail
- Pragmatic

Unit 2 Overview

7. Understanding Requirements

System Engineering

⭐Requirements Engineering

- Inception / Elicitation / Elaboration /NegotiationSpecification / Validation / Requirements management
- ⭐Non-Functional Requirements
- ⭐Diagrams (Use-case / Class / State / Activity)

8-10. Requirements Modeling

这里主要是 UML 图

Domain Analysis

⭐Scenario-Based Modeling

⭐Class-Based Modeling

⭐The definition of classes

- Nouns&verbs / Potencial classes / Analysis Classes
- Attributes&Operations / CRC card

⭐Behavioral Modeling

11. Design Concepts

- Data/Class design – transforms analysis classes into implementation classes and data structures
- Architectural design – defines relationships among the major software structural elements
- Interface design – defines how software elements, hardware elements, and end-users communicate
- Component-level design – transforms structural elements into procedural descriptions of software components

Design and quality

- **The design must implement all of the explicit requirements** contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.
- **The design must be a readable, understandable guide** for those who generate code and for those who test and subsequently support the software.
- **The design should provide a complete picture of the software**, addressing the data, functional, and behavioral domains from an implementation perspective.

Fundamental Concepts

- ⭐Abstraction / Architecture / Patterns / Separation of concerns / Modularity / Hiding / Functional independence / Refinement / Aspects / Refactoring
- OO design concepts: Design classes / Inheritance / Messages / Polymorphism
- Design Classes

> Tip
> 内聚耦合要会排序，但不要求背完整顺序

模块独立性的指标

- 内聚 (cohesion)：一个模块内部各个元素彼此结合的紧密程度 — 尽量高
- 耦合 (coupling)：模块之间相互关联的程度 — 尽量低

内聚度 （由高到低排列）：功能内聚，分层内聚，通信内聚，顺序内聚，过程内聚，时间内聚，实用内聚
耦合度 （由高到低排列）：非直接耦合，数据耦合，标记耦合，控制耦合，外部耦合，共用耦合，内容耦合

12-14. Design Methods

⭐Architecture Design

> Tip
> 这部分详细内容可以见 SA

⭐What is Architecture

The software architecture of a program or computing system is the structure or structures of the system, which comprise the **software components**, **the externally visible properties** of those components, and **the relationships** among them.

⭐Architectural Styles

Each style describes a system category that encompasses:

1. a set of components (e.g., a database, computational modules) that perform a function required by a system,
2. a set of connectors that enable “communication, coordination and cooperation” among components,
3. constraints that define how components can be integrated to form the system, and
4. semantic models that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

- Data flow architectures
  - batch processing, pipes and filters
- Call and return architectures
  - main program / subroutines, object-oriented system, layered system;
- Independent component architectures
  - event system
- Virtual machine architectures
  - interpreter, rule-based system
- Repository architectures
  - database system, blackboard system, etc.

Architectural Design

- The software must be placed into context
- A set of architectural archetypes should be identified
- The designer specifies the structure of the system by defining and refining software components that implement each archetype

⭐Agility and Architecture

- To avoid rework, user stories are used to create and evolve an architectural model (walking skeleton) before coding.
- Hybrid models which allow software architects contributing users stories to the evolving storyboard.
- Well run agile projects include delivery of work products during each sprint.
- Reviewing code emerging from the sprint can be a useful form of architectural review.

⭐Component Level Design

What is a Component

OO view: a component contains a set of collaborating classes
Conventional view: a component contains processing logic, the internal data structures that are required to implement the processing logic, and an interface that enables the component to be invoked and data to be passed to it.

⭐Basic Design Principles

- **⭐The Open-Closed Principle (OCP)**

- ⭐**The Liskov Substitution Principle (LSP)**
- ⭐**Dependency Inversion Principle (DIP)**
- ⭐**The Interface Segregation Principle (ISP)**
- The Release Reuse Equivalency Principle (REP)
- The Common Closure Principle (CCP)
- The Common Reuse Principle (CRP)

⭐Cohesion & Coupling

Cohesion

Conventional view: the “single-mindedness” of a module
OO view: cohesion implies that a component or class encapsulates only attributes and operations that are closely related to one another and to the class or component itself
Functional > Layer > Communicational > Sequential > Procedural > Temporal > Utility

Coupling

Conventional view: The degree to which a component is connected to other components and to the external world
OO view: a qualitative measure of the degree to which classes are connected to one another
Content > Common > Control > Stamp> Data > Routine call > Type use > Inclusion or import > External

⭐Component-Based Software Engineering

- Step 1. Identify all design classes that correspond to the problem domain.
- Step 2. Identify all design classes that correspond to the infrastructure domain.
- Step 3. Elaborate all design classes that are not acquired as reusable components.
  - Step 3a. Specify message details when classes or component collaborate.
  - Step 3b. Identify appropriate interfaces for each component.
  - Step 3c. Elaborate attributes and define data types and data structures required to implement them.
  - Step 3d. Describe processing flow within each operation in detail.
- Step 4. Describe persistent data sources (databases and files) and identify the classes required to manage them.
- Step 5. Develop and elaborate behavioral representations for a class or component.
- Step 6. Elaborate deployment diagrams to provide additional implementation detail.
- Step 7. Factor every component-level design representation and always consider alternatives.

User Interface Design

Golden Rules for UI Design

- Place the user in control
- Reduce the user’s memory load
- Make the interface consistent

User Interface Design Process

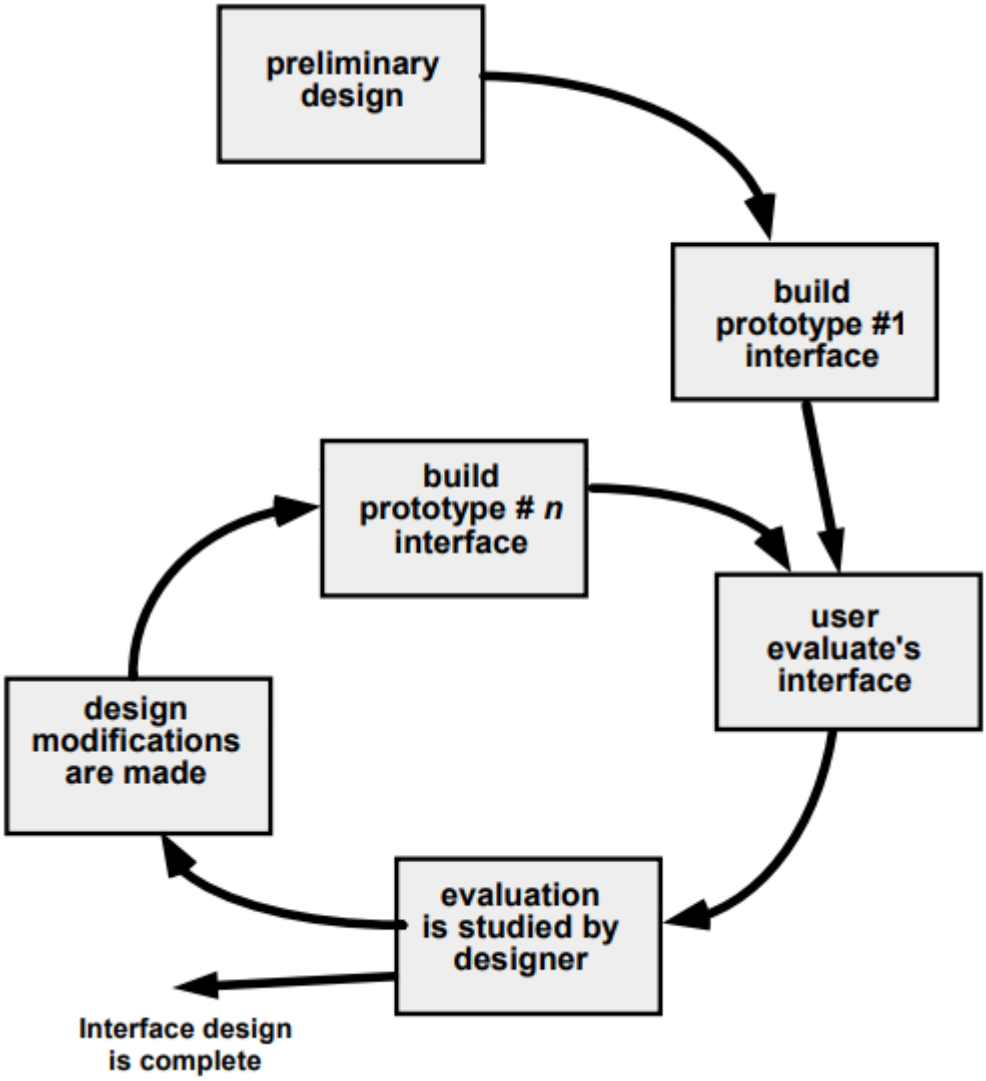
- Interface Analysis
- User Analysis

- Task Analysis and Modeling
- Analysis of Display Content

Interface Analysis

- Using information developed during interface analysis, **define interface objects and actions (operations)**.
- **Define events (user actions)** that will cause the state of the user interface to change. Model this behavior.
- **Depict each interface state** as it will actually look to the end-user.
- **Indicate how the user interprets the state of the system** from information provided through the interface.

Design Evaluation Cycle



Unit 3 Quality Assurance

15-16. Software Quality

⭐What is Quality?

An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.

⭐McCall's Triangle of Quality

⭐The Cost of Quality

- Prevention Costs (COP):
  - training,
  - standard, procedures
  - planning
  - quality improvement
  - audits
  - analysis
- Appraisal Costs (COA)
  - reviews
  - walkthroughs
  - testing
  - supplier monitoring
- Internal Failure Costs
  - Correction and rework
  - Regression testing
  - Wasted tester time
  - Cost of late shipment
  - Opportunity cost of late shipment
- External Failure Costs
  - Technical support calls
  - Investigate customer complaints
  - Refunds and recalls
  - Lost sales
  - Lost goodwill
  - Penalties
  - Liability costs

    - Legal fees

The relative costs to find and repair an error or defect increase dramatically as we go from prevention to detection to internal failure to external failure costs.

Achieving Software Quality

- Software quality is the result of good project management and solid engineering practice.
- To build high quality software you must understand the problem to be solved and be capable of creating a quality design the conforms to the problem requirements.
- Eliminating architectural flaws during design can improve quality.
- **Project management** - project plan includes explicit techniques for quality and change management.
- **Quality control** - series of inspections, reviews, and tests used to ensure conformance of a work product to its specifications.
- **Quality assurance** - consists of the auditing and reporting procedures used to provide management with data needed to make proactive decisions.

Software Quality assurance

Elements of SQA

- Standards
- Reviews and Audits
- Testing
- Error/defect collection and analysis
- Change management
- Education
- Vendor management
- Security management
- Safety
- Risk management

SQA Goals

- Requirements quality.
- Design quality.
- Code quality.
- Quality control effectiveness.

17-19. Testing Strategy & Techniques

⭐Testing concepts

- Error: a mistake in the design or programming; occurs when the system does not produce the expected output
- Fault: a mistake in the code; the result of an error
- Failure: the occurrence of asoftwarefault

Testing is the process of exercising a program with the specific intent of finding errors and faults prior to delivery to the end user.

⭐V model / V&V

Verification vs. Validation

- **Verification** refers to the set of tasksthatensure that software correctlyimplementsaspecific function. Are we building the product right?
- **Validation** refers to a different setoftasksthat ensure that the softwarethathasbeenbuilt is traceable to customerrequirements. Are we building the right product?
- Both verification and validation use the sametechniques(such as testing, review, inspection, etc.)
- The key difference between themis what is usedasthe“reference for correctness”
  - In verification, we use the output fromthepreviousstepas the reference of correctness (for example, whenweverify a code, we check it against the design)
  - In validation, we always use the “user needs”asthereference of correctness (for example, whenwevalidateacode, we check it against the user needs andexpectation)

⭐Testing Strategies

- Unit testing
- Integration testing
  - Top down;
  - Bottom up;
  - Regression;
  - Smoke
- System testing
  - Recovery testing;
  - Security testing;
  - Stress testing;
  - Performance testing
- Validation testing
  - α testing;
  - ß testing

⭐Testing techniques

White-Box Testing

The test case should be able to

- guarantee that all independent paths within a module have been exercised at least once,
- exercise all logical decisions on their true and false sides,
- execute all lopes at their boundaries and within their operational bounds, and
- exercise internal data structures to ensure their validity

Black Box Testing

- Also called behavioral testing.
- Focuses on the functional requirements of the software.
- It is a complementary approach that is likely to uncover a different class of errors than White box testing.
- It tends to be applied during later stages of testing.
- To find these errors:

- § Incorrect or missing functions
  - Interface errors
  - Errors in data structures or external database access
  - Behavior or performance errors
  - Initialization and termination errors.

Black Box vs. White Box testing

Black Box Methods	White Box Methods
Equivalence Partitioning （等价类划分）	Desk Checking （桌面检查）
Boundary Value Analysis （边界值分析）	Walkthrough （代码走查）
Graph-Based Methods （因果图分析）	Inspection （代码审查）
Orthogonal Array Testing （正交数组法）	Basis Path Testing （基本路径测试）
.....	.....

Static Testing

Others

- Static Testing
- Dynamic Testing

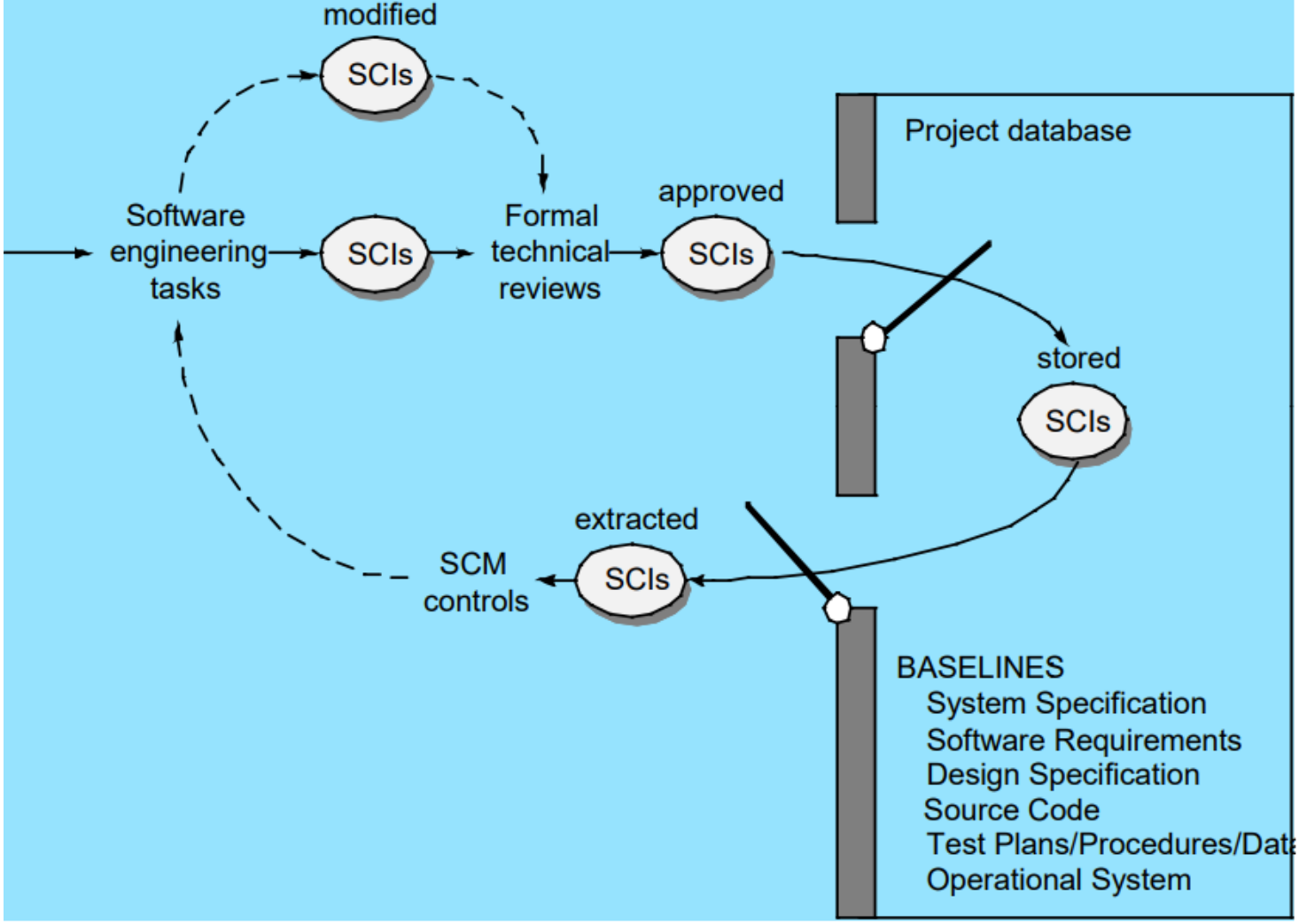
20-21. Security Engineering & SCM

⭐Why Security Engineering

- Security is a perquisite to system integrity, availability, reliability, and safety.
- Security provides the mechanism that enable a systems to protect its assets from attack.
- Assets are system resources (information, files, programs, storage, processor capacity) that have value to its stakeholders.
- Attacks take advantage of vulnerabilities that allow unauthorized system access.
- It is difficult to make a system more secure by responding to bug reports, security must be designed in from the beginning.

⭐SCM concepts: Baselines / SCI / SCM Process

Baselines



SCI

Repository Features

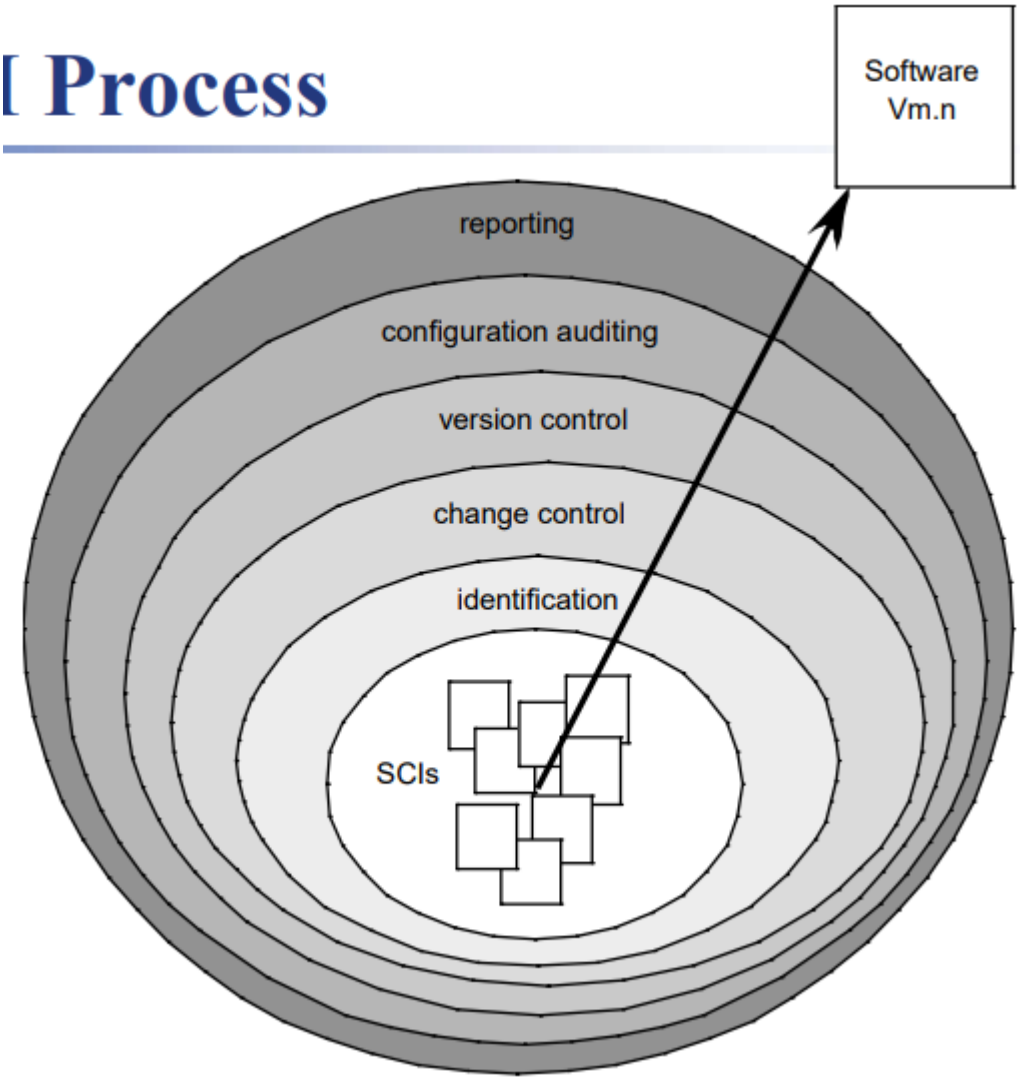
- Versioning
- Dependency tracking and change management
- Requirements tracing
- Configuration management
- Audit trails

SCM Elements

- Component elements
- Process elements
- Construction elements
- Human elements

⭐SCM process

[Process



Unit 4 Software Project Management

22. Project Management Concepts

⭐4P

- **People** — the most important element of a successful project
- **Product** — the software to be built
- **Process** — the set of framework activities and software engineering tasks to get the job done
- **Project** — all work required to make the product a reality

Stakeholders

- Senior managers
- Project (technical) managers
- Practitioners
- Customers
- End-users

The Product Scope

- Context
- Information objectives
- Function and performance

Software project scope must be unambiguous and understandable at the management and technical levels.

Problem Decomposition

Sometimes called **partitioning** or **problem elaboration**.
Decomposition process continues until all functions or problem classes have been defined.

The Process

Once a process framework has been established

- Consider project characteristics
- Determine the degree of rigor required
- Define a task set for each software engineering activity
  - Task set =
    - Software engineering tasks
    - Work products
    - Quality assurance points
    - Milestones

Common-Sense Approach to Projects

- Start on the right foot
- Maintain momentum
- Track progress
- Make smart decisions
- Conduct a postmortem analysis

⭐W5HH

To Get to the Essence of a Project

- Why is the system being developed?
- What will be done?
- When will it be accomplished?
- Who is responsible?
- Where are they organizationally located?
- How will the job be done technically and managerially?
- How much of each resource (e.g., people, software, tools, database) will be needed?

23. Process and Project Metrics

⭐Why Do We Measure?

- Assess the status of an ongoing project.
- Track potential risks.
- Uncover problem areas before they go “critical”.
- Adjust work flow or tasks.
- Evaluate the project team’s ability to control quality of software work products.

⭐Process Measurement and Metrics

Process Measurement

We measure the efficacy of a software process indirectly.

- That is, we derive a set of metrics based on the outcomes that can be derived from the process.
- Outcomes include
  - measures of errors uncovered before release of the software
  - defects delivered to and reported by end-users
  - work products delivered (productivity)
  - human effort expended
  - calendar time expended
  - schedule conformance
  - other measures.
- We also derive process metrics by measuring the characteristics of specific software engineering tasks.

Process Metrics

- **Quality-related** - focus on quality of work products and deliverables
- **Productivity-related** - Production of work-products related to effort expended
- **Statistical SQA data** - error categorization & analysis
- **Defect removal efficiency** - propagation of errors from process activity to activity
- **Reuse data** - The number of components produced and their degree of reusability

⭐Project Metrics

- Used to minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks
- Used to assess product quality on an ongoing basis and, when necessary, modify the technical approach to improve quality.
- Every project should measure:
  - inputs—measures of the resources (e.g., people, tools) required to do the work.
  - outputs—measures of the deliverables or work products created during the software engineering process.
  - results—measures that indicate the effectiveness of the deliverables.

Typical Types of Metrics

- errors per KLOC (thousand lines of code)
- defects per KLOC
- $ per LOC
- pages of documentation per KLOC
- errors per person-month
- errors per review hour
- LOC per person-month
- $ per page of documentation

24-25. Project Estimation & Scheduling

⭐Scope

Scope refers to all the work involved in creating the products of the project and the processes used to create them. It defines what is or is not to be done.
Software scope describes

- the functions and features that are to be delivered to end-users
- the data that are input and output
- the "content" that is presented to users as a consequence of using the software
- the performance, constraints, interfaces, and reliability that bound the system.

⭐Work Breakdown Structure (WBS)

A work breakdown structure is a deliverableoriented grouping of the work involved in a project that defines the total scope of the project.
It provides the basis for

- Planning and managing project schedules, costs, and changes
- Risk analysis
- Organizational structure
- Measurement

A WBS may be a diagram (graphical tree) or a text list (outline format).

WBS Types

- WBS is usually organized around
  - project products (tools, tangible outputs, .)
  - project phases (concepts, steps,..)
  - project management process groups (initializing, developing, executing, closing)
- Product WBS
  - Entity-oriented
    - E.g., Financial engine, Interface system, DB
    - Typically used by engineering manager
- Process WBS
  - Activity-oriented
    - E.g., Requirements, Analysis, Design, Testing
    - Typically used by project managers

Approaches to Developing WBS

1. Using guidelines
2. Analogy approach
3. Top-down approach
4. Bottom-up approach
5. Mind-mapping approach

⭐LOC / FP / COCOMO estimation

- Compute LOC/FP using estimates of information domain values
- Use historical data to build estimates for the project

LOC Approach

- Advantages
  - Simple to measure

    - Easy to automate
- Disadvantages
    - Only defined for code, not the design
    - Bad design may cause excessive LOC
    - Language dependent
    - Does not account for functionality or complexity

FP Approach

- Advantages
    - Usable in the earliest requirements phases
    - Independent of programming language, product design, or development style
    - User's view, rather than implementation view
    - Can be used to measure the non-coding activities
    - There exists a large body of historical data A well documented method
- Disadvantages
    - Cannot directly count an existing product's (source code) FP content
    - Hard to automate
    - FP do not reflect language, design, or style differences
    - FP are designed for estimating commercial data processing applications
    - Subjective counting

COCOMO (Constructive Cost Modeling)

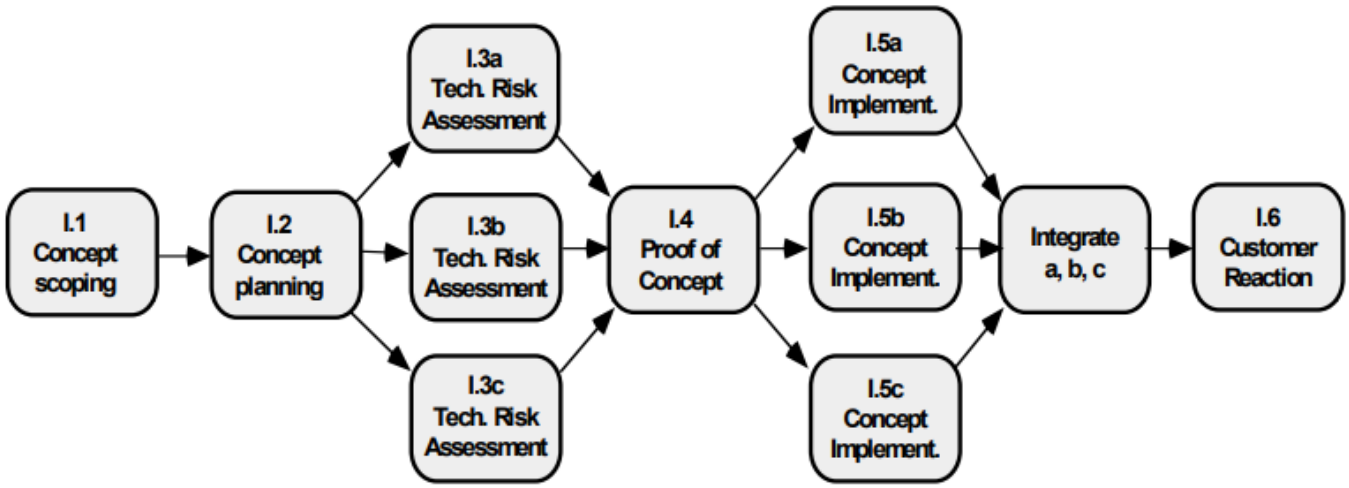
An empirical model based on project experience

Scheduling Steps

1. Defining Task Sets——WBS
2. Sequencing Activities
3. Drawing Project network diagrams
4. Critical path analysis
5. Using Gantt charts for scheduling
6. Schedule Tracking

⭐Task Network / Gantt charts / milestone

Define a Task Network



Three I.3 tasks are applied in parallel to 3 different concept functions

Three I.3 tasks are applied in parallel to 3 different concept functions

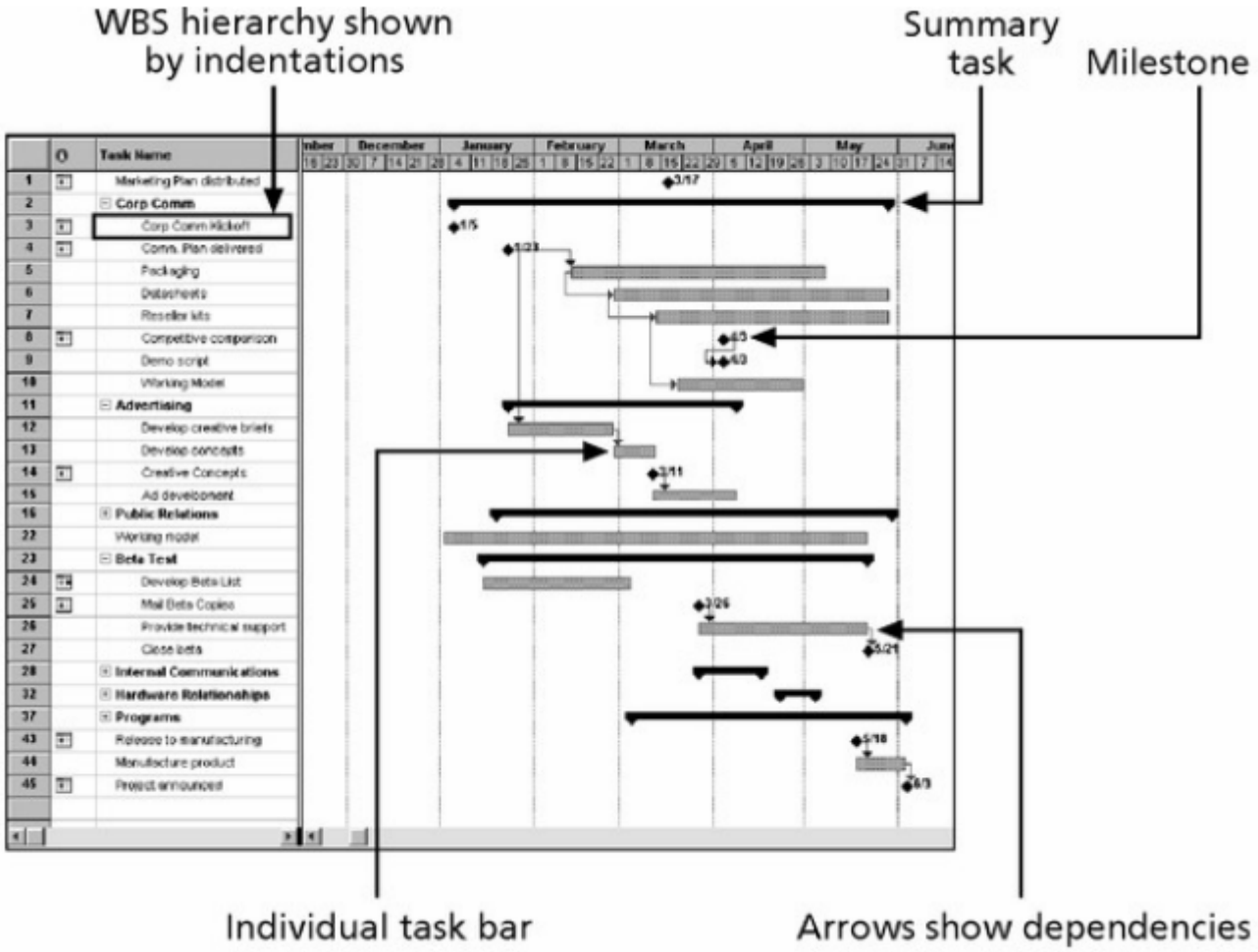
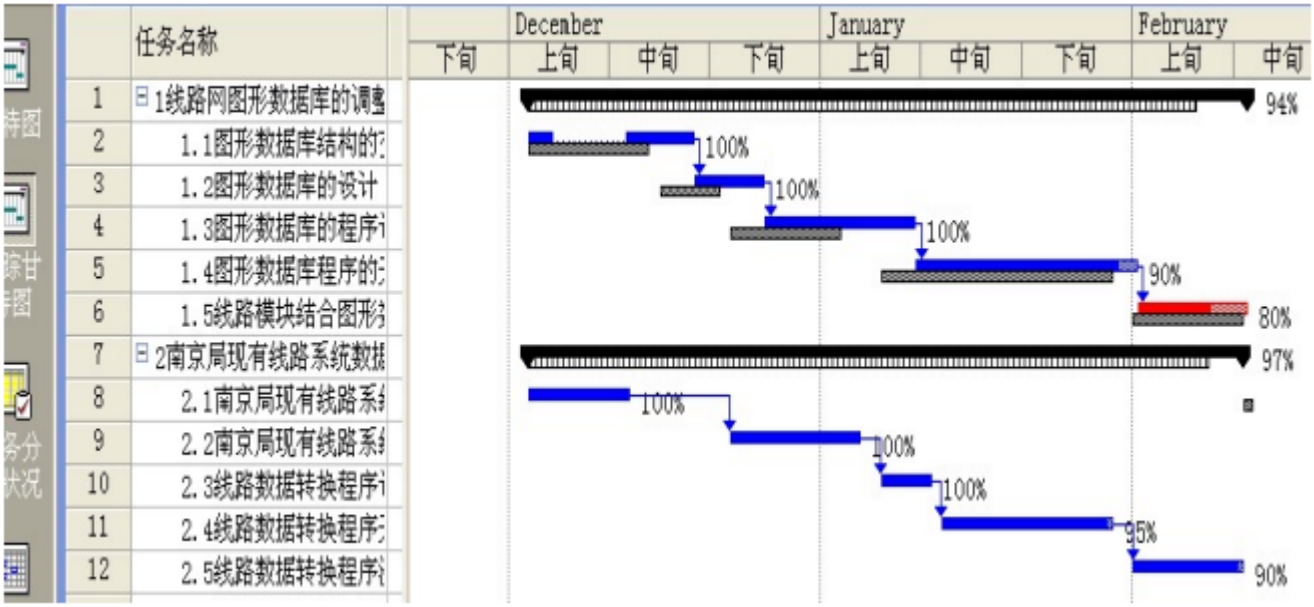


Figure 6-6. Gantt Chart for Software Launch Project



26. Risk Analysis

Project Risk

- What can go wrong?
- What is the likelihood?
- What will the damage be?
- What can we do about it?

Reactive Risk Management

Project team reacts to risks when they occur

- mitigation—plan for additional resources in anticipation of fire fighting
- fix on failure—resource are found and applied when the risk strikes
- crisis management—failure does not respond to applied resources and project is in jeopardy

Proactive Risk Management

- formal risk analysis is performed
- organization corrects the root causes of risk
  - TQM concepts and statistical SQA
  - examining risk sources that lie beyond the bounds of the software
  - developing the skill to manage change

⭐Risk Management Paradigm

- Risk identification
- Risk analysis
- Risk planning
- Risk monitoring

⭐RMMM

- Mitigation—how can we avoid the risk?
- Monitoring—what factors can we track that will enable us to determine if the risk is becoming more or less likely?
- Management—what contingency plans do we have if the risk becomes a reality?