Schedule Updated (May 25)!

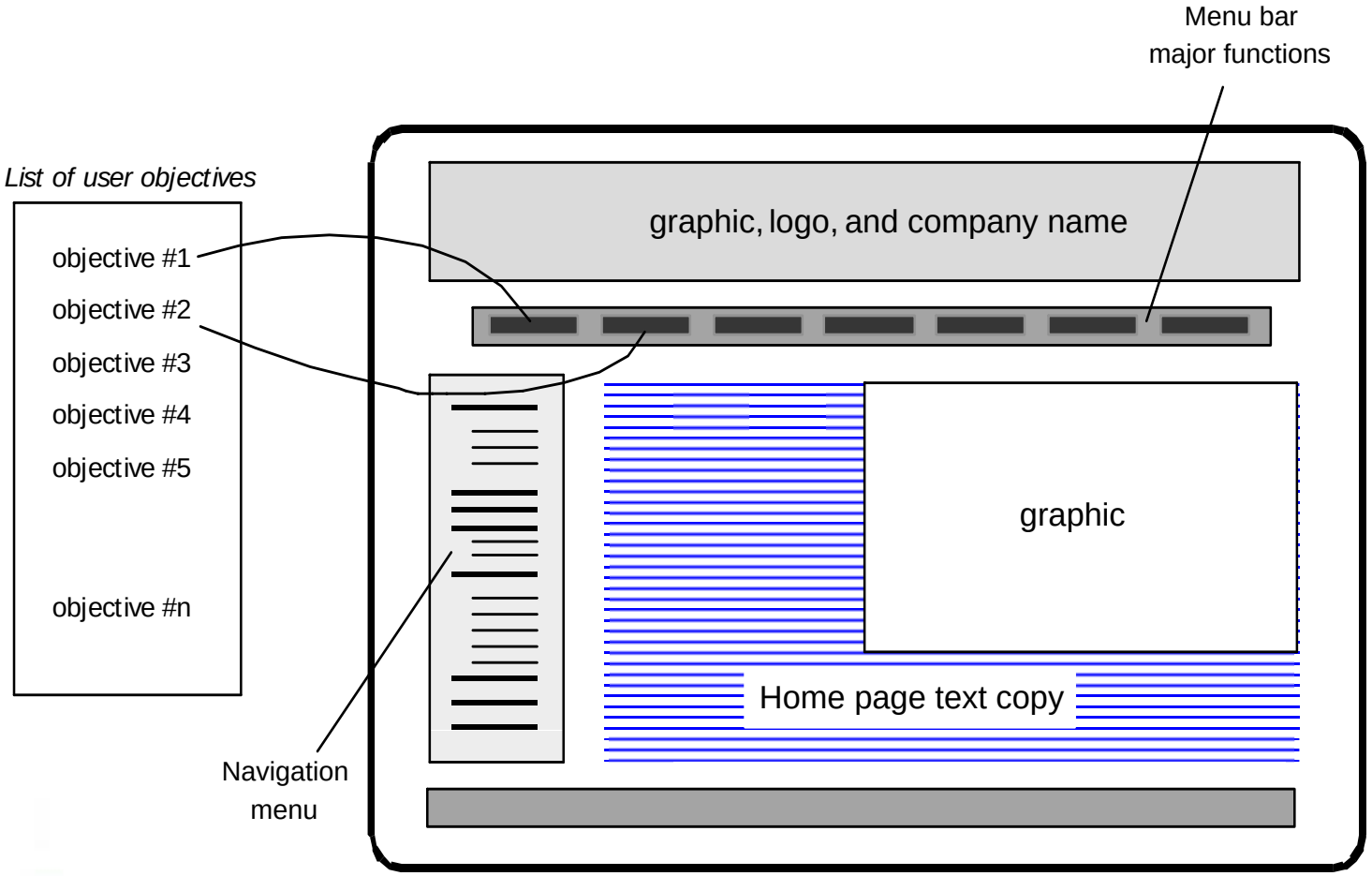
周次	教学内容	课时	课外作业	课外课时
1	Ch.0-1 The Nature of Software	2	习题集1	2
2	Ch.1-2 The Process	2	习题集2	2
3	Ch.3-4 The Process(2)	2	习题集3-4	2
4	Ch.31 Project Management； Ch.5 Agile； Ch.6 Human Aspects；	2	习题集31,5-6	2
5	Ch.7 Modeling Principles; Ch.8-9 Requirements: Concepts & Scenario(4.13)	2	习题集7-8-9, 布置需求报告	2
6	Ch.10-11 Requirements: Class & others (4.20)	2	习题集10-11	2
7	Ch.19 Quality Concepts；Ch.12 Design Concepts (4.26,周日，补清明节课)	2	习题集19-12	2
8	Ch.13 Architectural Design(4.27)	2	习题集13，收需求报告; 布置设计报告	2
9	Ch.17 WebApp Design；Ch.18 MobileApp Design；Ch.20-21 Review & SQA （5.4)）	2	习题集17-18，20-21	2
9B	总体设计报告演讲（5.10,周日,补!)		收设计报告	2
10	Ch.14 Component-level Design；Ch.15 UI Design； （5.18)	2	习题集14-15	2
11	Ch.16 Pattern-based Design Ch.29 Configuration Management； Ch.22 Testing Strategies （5.25)	2	习题集16, 22，29,布置设计模式研究报告, 布置测试报告	2
12	Ch.23-24 Testing Conventional & OO Apps （6.1)	2	习题集23-24，布置v1.0；收设计模式研究报告	2
13	Ch.25-26 Testing for WebApp & Mobile App （6.8)	2	习题集25-26，收测试	2
14	Ch.27 Security Engineering；Ch.28 Formal Methods*；Ch.36 Maintenance （6.15)	2	习题集27-28，36；收v1.0; 布置v2.0	2
15	Ch.34 Scheduling；Ch.35 Risk；Ch.30 Product Metrics （6.22)	2	习题集34-35，30,收v2.0；布置合并版	2
16	Ch.32 Project Process Metrics；Ch.33 Estimation （6.29)	2	习题集32-33；展示合并版	2

Ch.15 User Interface Design(Cont.)

May 25, 2015

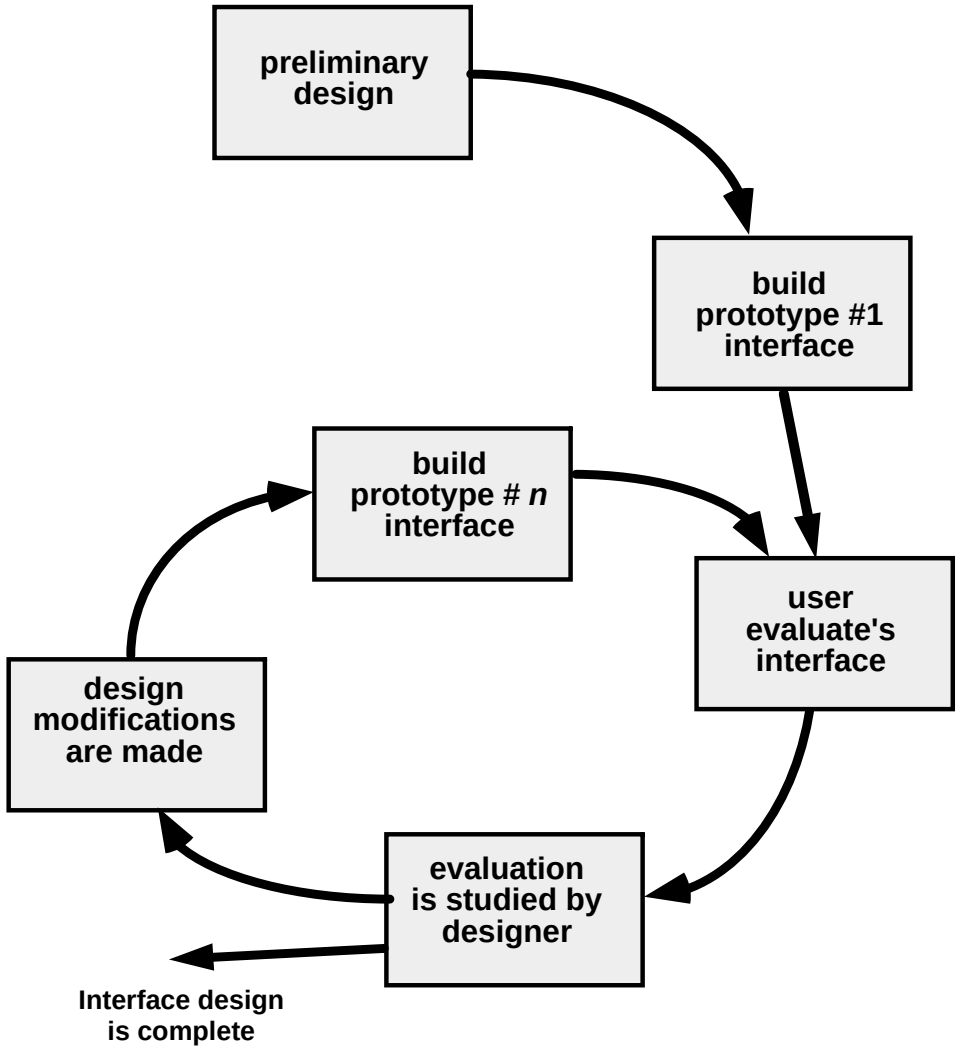
- **Interface Design Workflow - I**
  - Review information contained in the analysis model and refine as required.
  - Develop a rough sketch of the Web or Mobile App interface layout.
  - Map user objectives into specific interface actions.
  - Define a set of user tasks that are associated with each action.
  - Storyboard screen images for each interface action.
  - Refine interface layout and storyboards using input from aesthetic design.

- **Mapping User Objectives**



- **Interface Design Workflow - II**
  - Identify user interface objects that are required to implement the interface.
  - Develop a procedural representation of the user's interaction with the interface.
  - Develop a behavioral representation of the interface.
  - Describe the interface layout for each state.
  - Refine and review the interface design model.

- **Design Evaluation Cycle**



Ch.16 Pattern-Based Design

16.1 Design Patterns

I wonder if anyone has developed a solution to for a design problem ?

- Design patterns are a codified method for describing problems and their solution allows the software engineering community to capture design knowledge in a way that enables it to be reused.

Please read Christopher Alexander 's description in 1977

16.1 Design Patterns

- Basic Concepts

  "a three-part rule which expresses a relation between a certain context, a problem, and a solution."

  - Context: allows the reader to understand
    - the environment in which the problem resides
    - what solution might be appropriate within that environment.
  - a system of forces: a set of requirements, including limitations and constraints, influences
    - how the problem can be interpreted within its context
    - how the solution can be effectively applied.

16.1 Design Patterns

- Effective Patterns (Coplien, 2005)
  - ➢ It solves a problem
    not just abstract principles or strategies.
  - ➢ It is a proven concept
    not theories or speculation.
  - ➢ The solution isn't obvious
    generate a solution to a problem indirectly
    --a necessary approach for the most difficult problems of design.
  - ➢ It describes a relationship
    don't just describe modules, but describe deeper system structures and mechanisms.
  - ➢ The pattern has a significant human component
    (minimize human intervention(干预))
    explicitly appeal to aesthetics and utility（功用）.

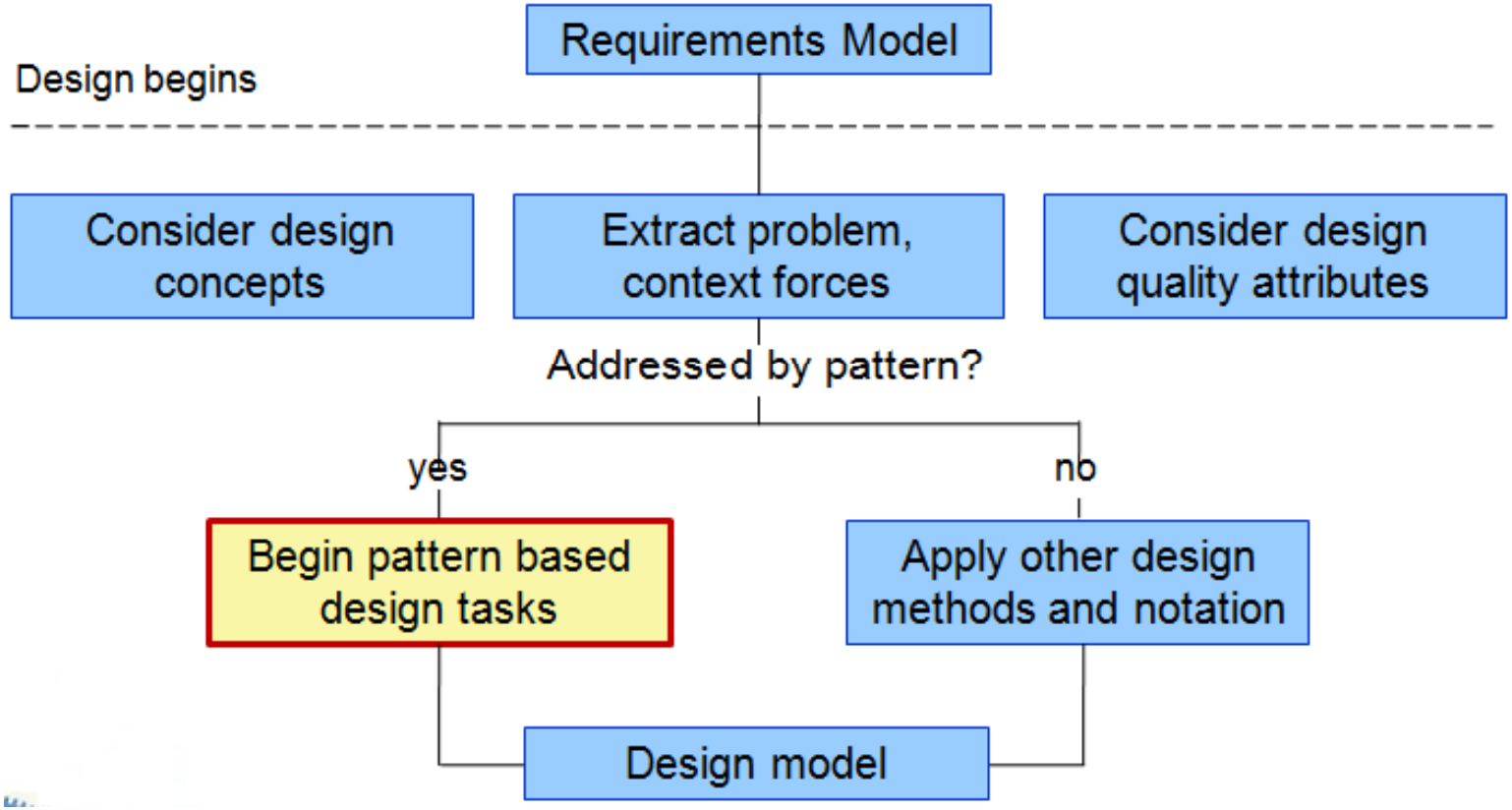
16.1 Design Patterns

- Frameworks
  - A "reusable mini-architecture that provides the generic structure and behavior for a family of software abstractions, along with a context … which specifies their collaboration and use within a given domain."
  - Not an architectural pattern, but rather a skeleton with a collection of "plug points" (also called hooks and slots（插槽）) that enable it to be adapted to a specific problem domain.

    - Design patterns are more *abstract* than frameworks.
    - Design patterns are *smaller* architectural elements than frameworks
    - Design patterns are *less specialized* than frameworks

16.2 Pattern-Based Software Design

- Pattern-Based Design in Context



16.2 Pattern-Based Software Design

- Thinking in Patterns and Design Tasks

➢ Shalloway and Trott [Sha05] suggest the following approach：

– Step1：Be sure you understand the big picture—the context in which the software to be built resides（所处）. The requirements model should communicate this to you.

– Step2： Examining the big picture, extract the patterns that are present at that level of abstraction.

– Step3： Begin your design with ‘big picture’ patterns that establish a context or skeleton for further design work.

– Step4：“Work inward from the context” looking for patterns at lower levels of abstraction that contribute to the design solution.

– Step5： Repeat steps 1 to 4 until the complete design is fleshed out（充实）.

– Step6： Refine the design by adapting each pattern to the specifics of the software you’re trying to build.

16.2 Pattern-Based Software Design

- Design Tasks

1. Examine the requirements model and develop a problem hierarchy.
2. Determine if a reliable pattern language has been developed for the problem domain.
3. Beginning with a broad problem, determine whether one or more architectural patterns are available for it.
4. Using the collaborations provided for the architectural pattern, examine subsystem or component level problems and search for appropriate patterns to address them.
5. Repeat steps 2 through 4 until all broad problems have been addressed.
6. If user interface design problems have been isolated (this is almost always the case), search the many user interface design pattern repositories for appropriate patterns.
7. Regardless of its level of abstraction, if a pattern language and/or patterns repository or individual pattern shows promise（有指望）, compare the problem to be solved against the existing pattern(s) presented.
8. Be certain to refine the design as it is derived from patterns using design quality criteria as a guide.

16.2 Pattern-Based Software Design

- Common Design Mistakes
  - Not enough time has been spent to understand the underlying problem, its context and forces, and as a consequence, you select a pattern that looks right, but is inappropriate for the solution required.
  - Once the wrong pattern is selected, you refuse to see your error and force fit the pattern.
  - In other cases, the problem has forces that are not considered by the pattern you've chosen, resulting in a poor or erroneous fit.
  - Sometimes a pattern is applied too literally and the required adaptations for your problem space are not implemented.

16.3 Architectural Patterns

Example：House & Kitchen pattern

- The Kitchen pattern and patterns it collaborates with address problems associated with the storage and preparation of food, the tools required to accomplish these tasks, and rules for placement of these tools relative to workflow in the room.
- The Kitchen pattern might address problems associated with countertops（厨房台面）, lighting, wall switches, a central island, flooring, and so on.
- There is more than a single design for a kitchen, but every design can be conceived within the context of the ‘solution’ suggested by the Kitchen pattern.

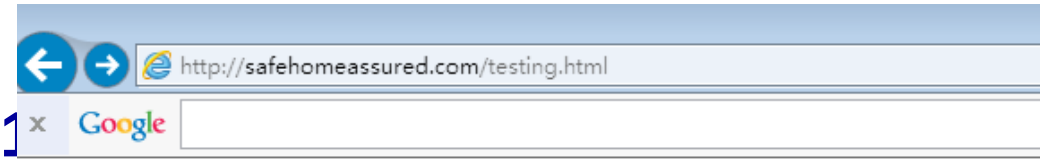
- Architectural Patterns address issues such as concurrency, persistence, and distribution, et al.

16.4 Component-Level Patterns

- Component-level design patterns provide a proven solution that addresses one or more sub-problems extracted from the requirement model.
- In many cases, design patterns of this type focus on some functional element of a system.

Example：SafeHomeAssured.com

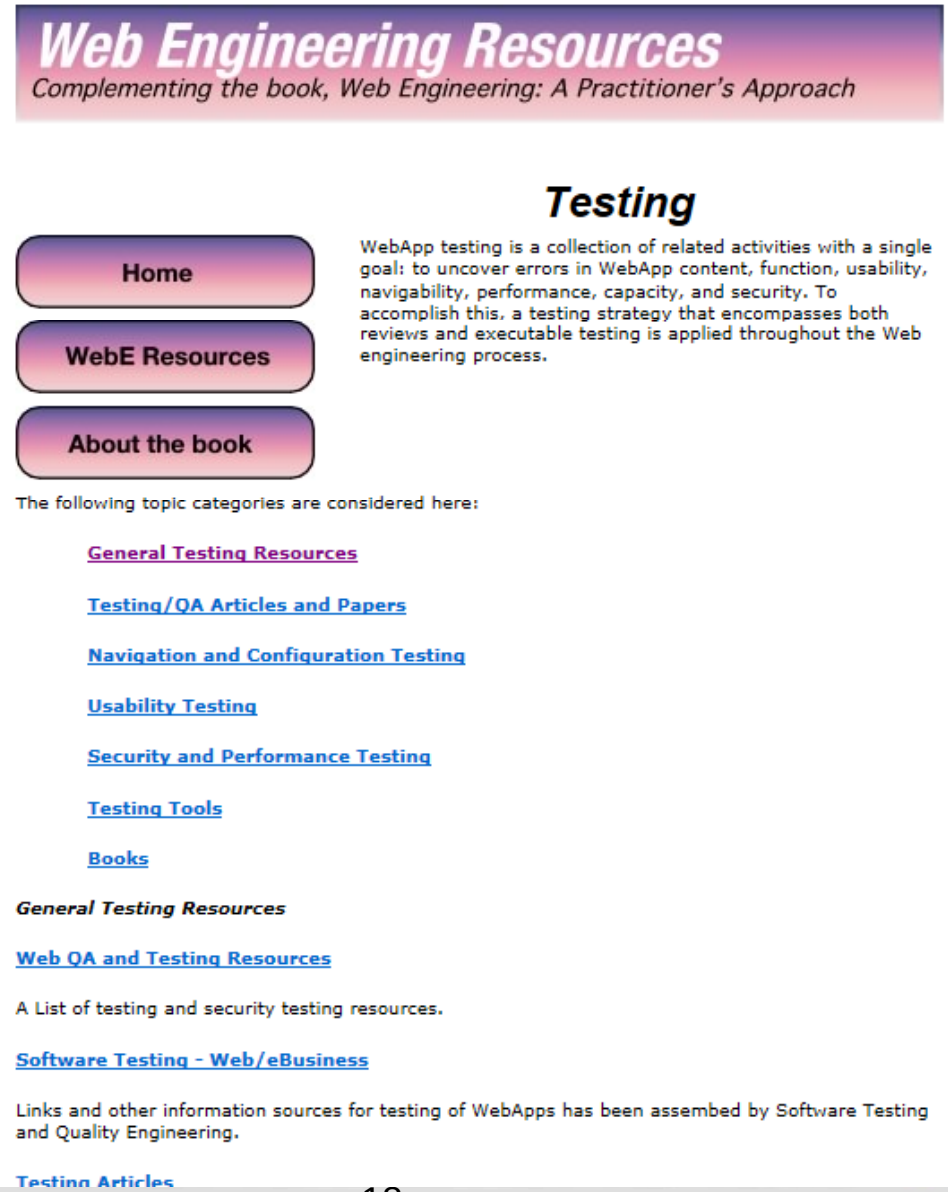
- The design sub-problem: How can we get product specifications and related information for any SafeHome device?
- Search-related patterns：counter AdvancedSearch, HelpWizard, SearchArea, SearchTips, SearchResults, SearchBox, ……



- Componen ven solution that addres acted from the require
- In many ca us on some functional e

Ex om

- The design cifications and related
- Search-rela elpWizard, SearchArea .....



16.5 User Interface (UI) Patterns

- Design Focus

- Whole UI. Provide design guidance for top-level structure and navigation throughout the entire interface.
- Page layout. Address the general organization of pages (for Websites) or distinct screen displays (for interactive applications)
- Forms and input. Consider a variety of design techniques for completing form-level input.
- Tables. Provide design guidance for creating and manipulating tabular data of all kinds.
- Direct data manipulation. Address data editing, modification, and transformation.
- Navigation. Assist the user in navigating through hierarchical menus, Web pages, and interactive display screens.
- Searching. Enable content-specific searches through information maintained within a Web site or contained by persistent data stores that are accessible via an interactive application.
- Page elements. Implement specific elements of a Web page or display screen.
- E-commerce. Specific to Web sites, these patterns implement recurring elements of e-commerce applications.

16.6 Webapp Patterns

- Categories

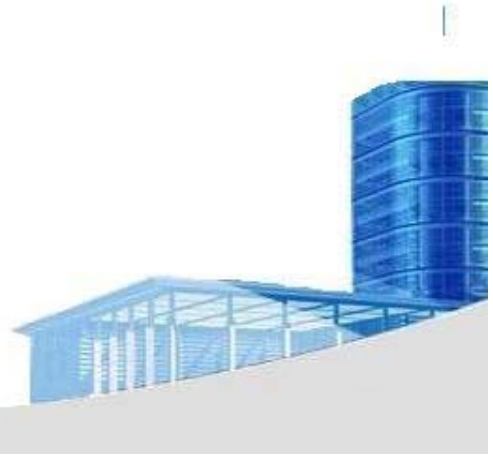
- Information architecture patterns relate to the overall structure of the information space, and the ways in which users will interact with the information.
- Navigation patterns define navigation link structures, such as hierarchies, rings, tours, and so on.
- Interaction patterns contribute to the design of the user interface. Patterns in this category address how the interface informs the user of the consequences of a specific action; how a user expands content based on usage context and user desires; how to best describe the destination that is implied by a link; how to inform the user about the status of an on-going interaction, and interface related issues.
- Presentation patterns assist in the presentation of content as it is presented to the user via the interface. Patterns in this category address how to organize user interface control functions for better usability; how to show the relationship between an interface action and the content objects it affects, and how to establish effective content hierarchies.
- Functional patterns define the workflows, behaviors, processing, communications, and other algorithmic elements within a WebApp.

16.7 Patterns for Mobile Apps

- Mobile User Interface Patterns
  - Check-in screens, Maps, Popovers(显示/隐藏提示), Sign-up(报名，签约) flows, Custom Tab Navigation, Invitations. Ex. 格林豪泰酒店

- Mobile User Interface Patterns
  - Active Objects
  - Applications Controller
  - Communicator
  - Data Transfer Object
  - Domain Model
  - Entity Translator
  - Lazy Acquisition
  - Model-View-Controller
  - Pagination(页码标注)
  - Reliable Sessions
  - Synchronization
  - Transaction Script

Ch.22 Software Testing Strategies



Example 1: **In 1963, USA, the rocket to Mars failed, losing $ 10 million. The cause was a mistake in the FORTRAN program –**

DO 5 I = 1, 3 was written as DO 5 I = 1.3

Example 2: **[Washington Post, 1996]**

Dallas, Aug. 23 — The captain of an American Airlines jet that crashed in Colombia last December entered an incorrect one-letter computer command that sent the plane into a mountain, the airline said today.

The crash killed all but four of the 163 people aboard.

American's investigators concluded that the captain of the Boeing 757 apparently thought he had entered the coordinates for the intended destination, Cali.

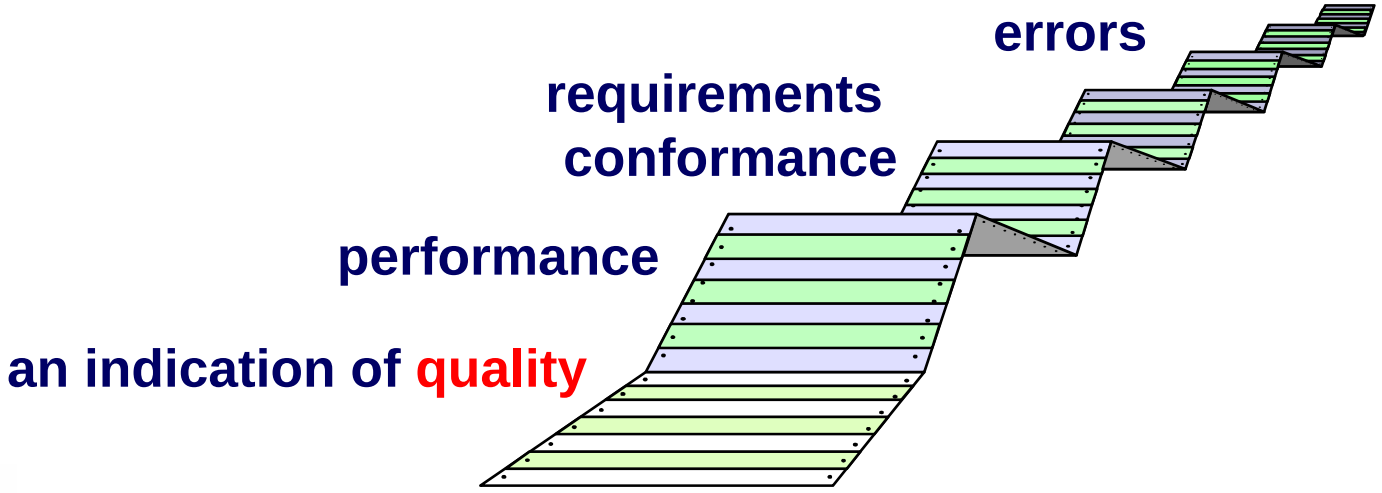
But on most South American aeronautical charts, the one-letter code for Cali is the same as the one for Bogota, 132 miles in the opposite direction.

The coordinates for Bogota directed the plane toward the mountain, according to a letter by Cecil Ewell, American's chief pilot and vice president for flight. The codes for Bogota and Cali are different in most computer databases, Ewell said.

- **What is testing for?**

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

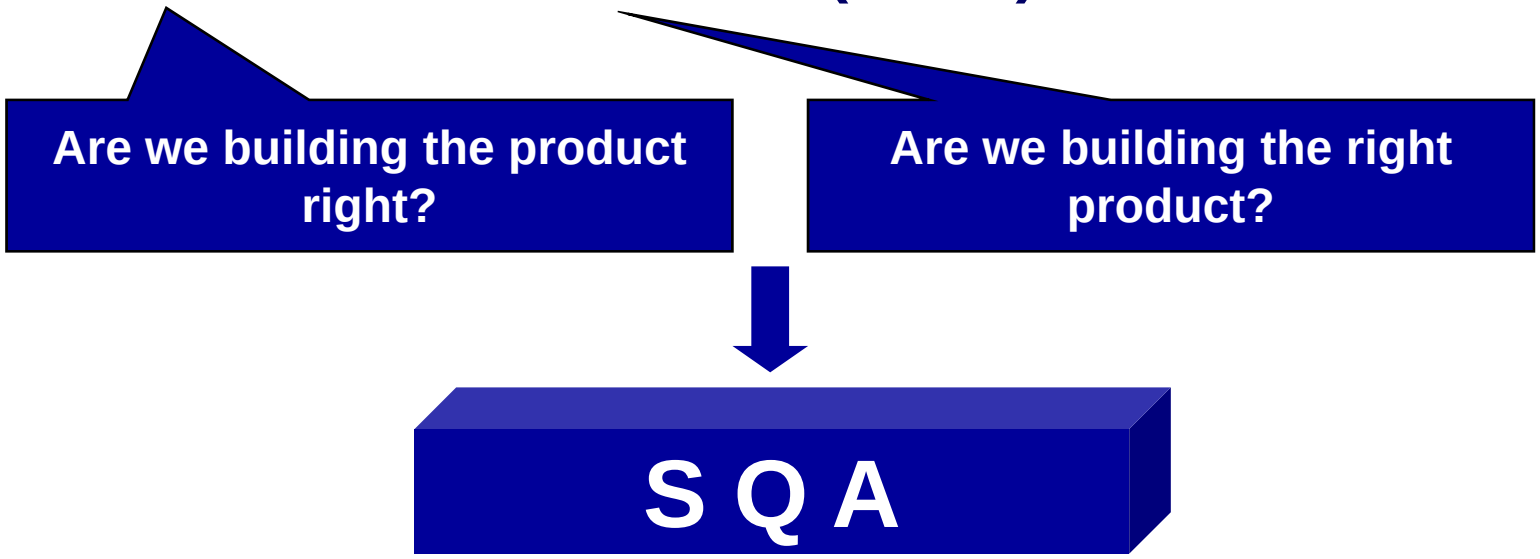
- **What testing shows?**



- **Strategic Approach**
  - To perform effective testing, you should conduct effective technical reviews. By doing this, many errors will be eliminated before testing commences.
  - Testing begins at the component level and works "outward" toward the integration of the entire computer-based system.
  - Different testing techniques are appropriate for different software engineering approaches and at different points in time.
  - Testing is conducted by the developer of the software and (for large projects) an independent test group.
  - Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.

验证　　确认

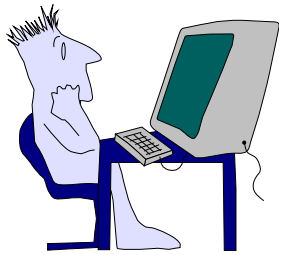
- **Verification and Validation (V & V)**



Software testing is only one element of SQA.

Quality must be built in to the development process, one **cannot** use testing to **add** quality **after** the fact.

- **Organizing for Software Testing**



Developer

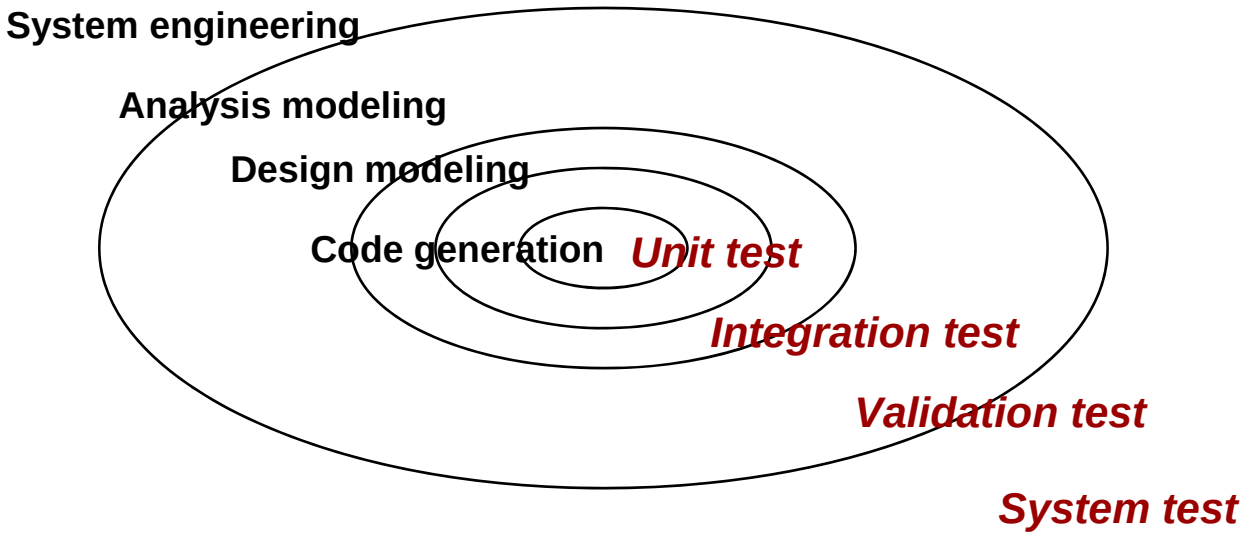
Understands the system, but will test "gently" and, is driven by "delivery"

Independent Test Group

Must learn about the system, but will attempt to break it and, is driven by quality

WRONG	RIGHT
The developer should do no testing at all.	Unit + Integration. Then ITG gets in.
Software is tossed "over the wall" to ITG.	Work closely and remove errors.
Testers are not involved with the project until it is time for it to be tested.	Becomes involved during analysis and design and stays involved.

- **Testing Strategy**



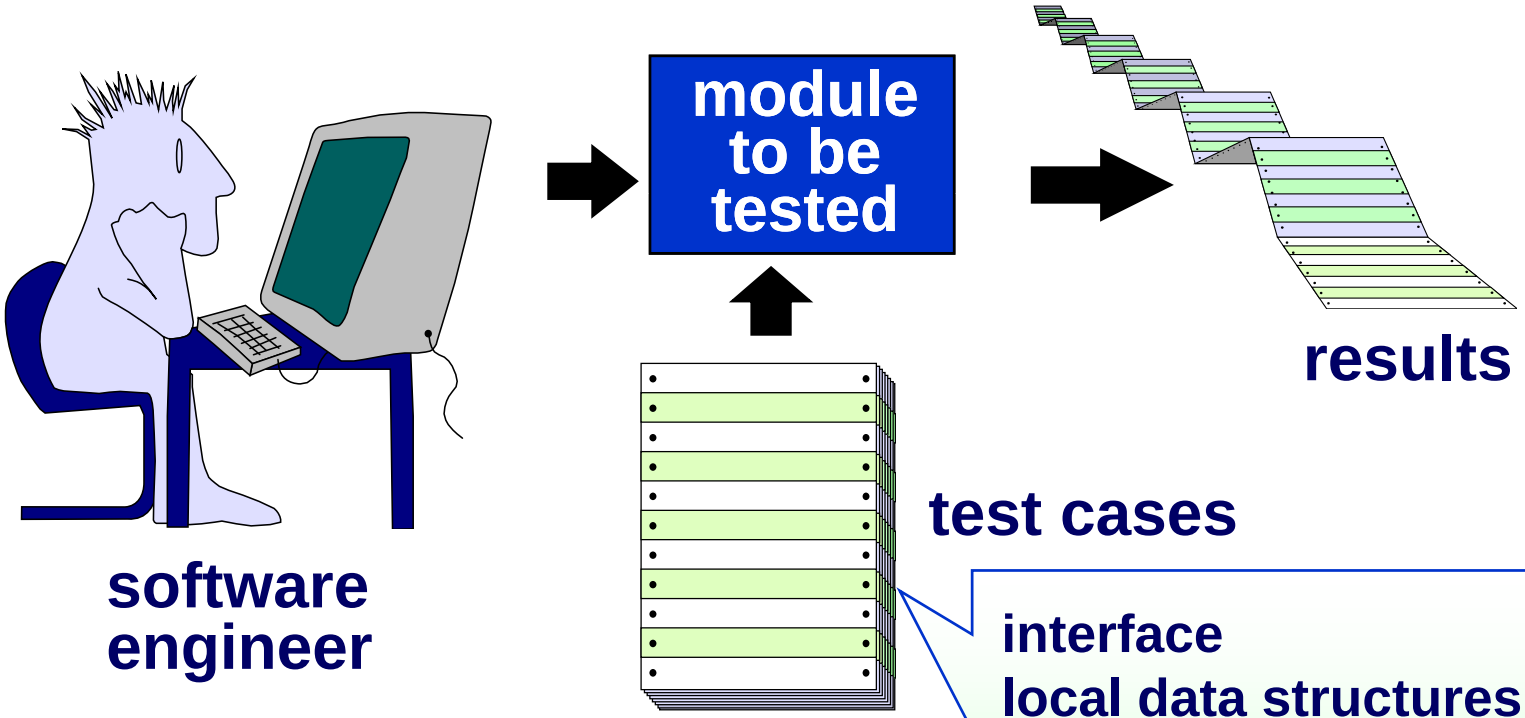
- **Testing Strategy**
  - We begin by *'testing-in-the-small'* and move toward *'testing-in-the-large'*
  - For conventional software
    - The module (component) is our initial focus
    - Integration of modules follows
  - For O-O software
    - our focus when "testing in the small" changes from an individual module (the conventional view) to an O-O class that encompasses attributes and operations and implies communication and collaboration

- **Strategic Issues**

1. Specify product requirements in a quantifiable manner long before testing commences.
2. State testing objectives explicitly.
3. Understand the users of the software and develop a profile for each user category.
4. Develop a testing plan that emphasizes “rapid cycle testing.”
5. Build “robust” software that is designed to test itself
6. Use effective technical reviews as a filter prior to testing
7. Conduct technical reviews to assess the test strategy and test cases themselves.
8. Develop a continuous improvement approach for the testing process.

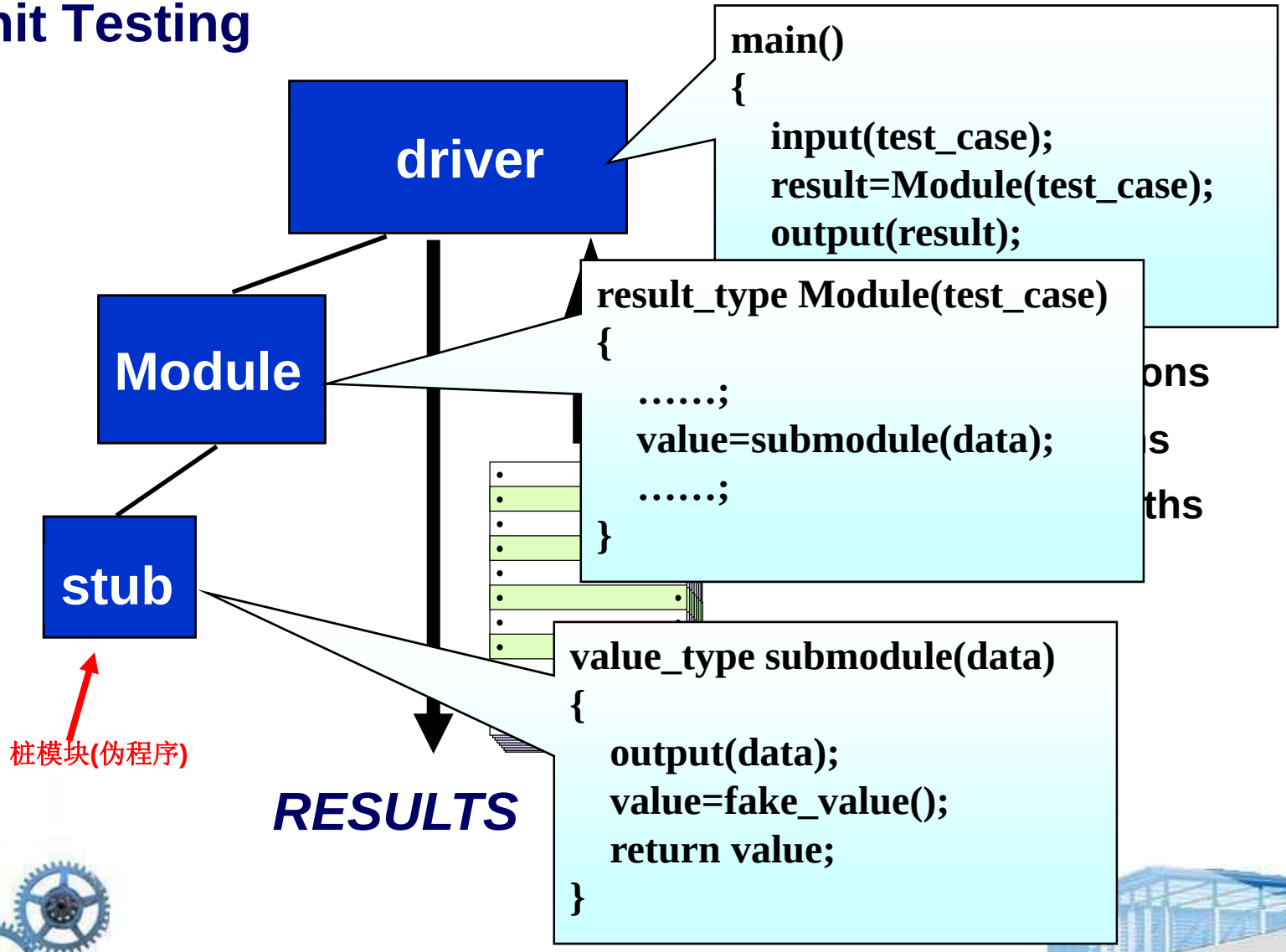
For Conventional Software

- **Unit Testing**



For Conventional Software

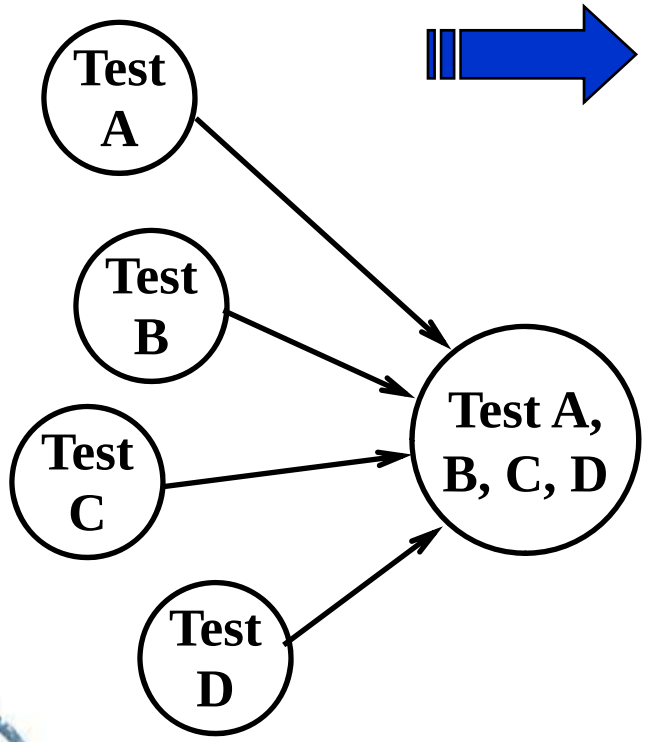
- **Unit Testing**



For Conventional Software

- **Integration Testing**

Big-bang testing

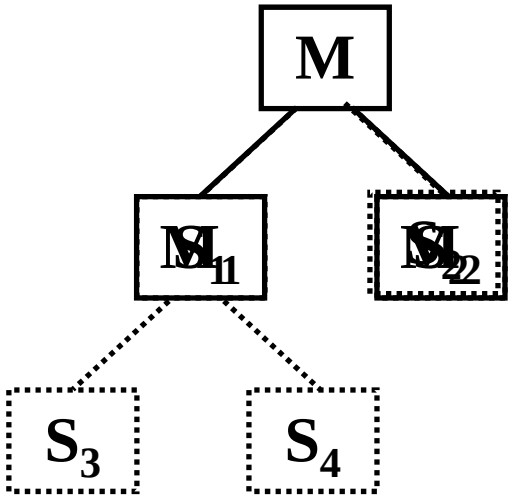


Chaos !

Isolation of causes is complicated.

For Conventional Software

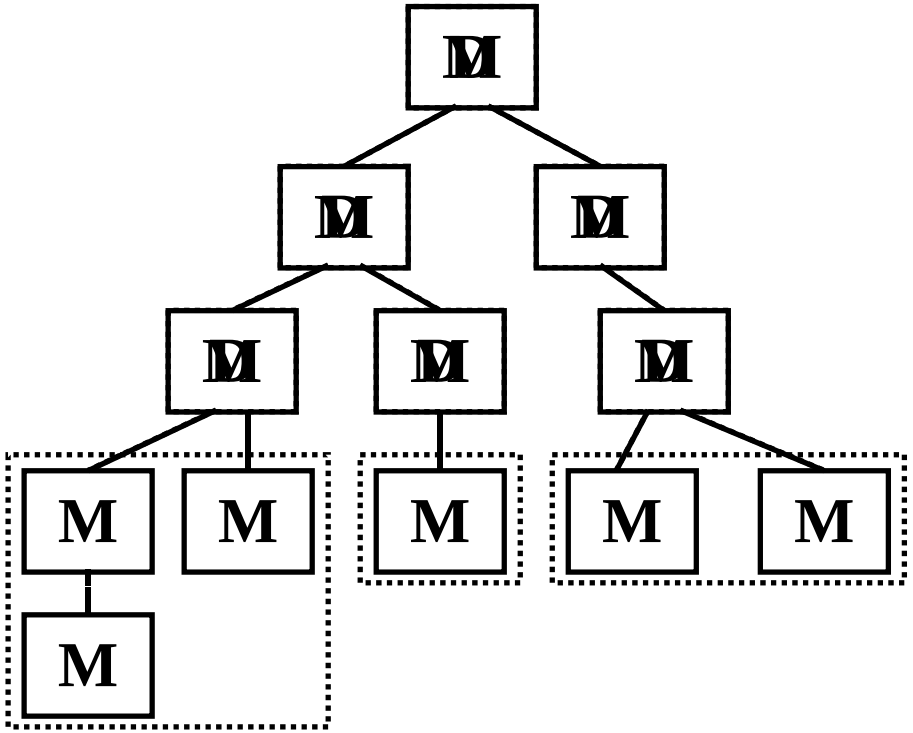
Top-down



verify major control or decision points early.

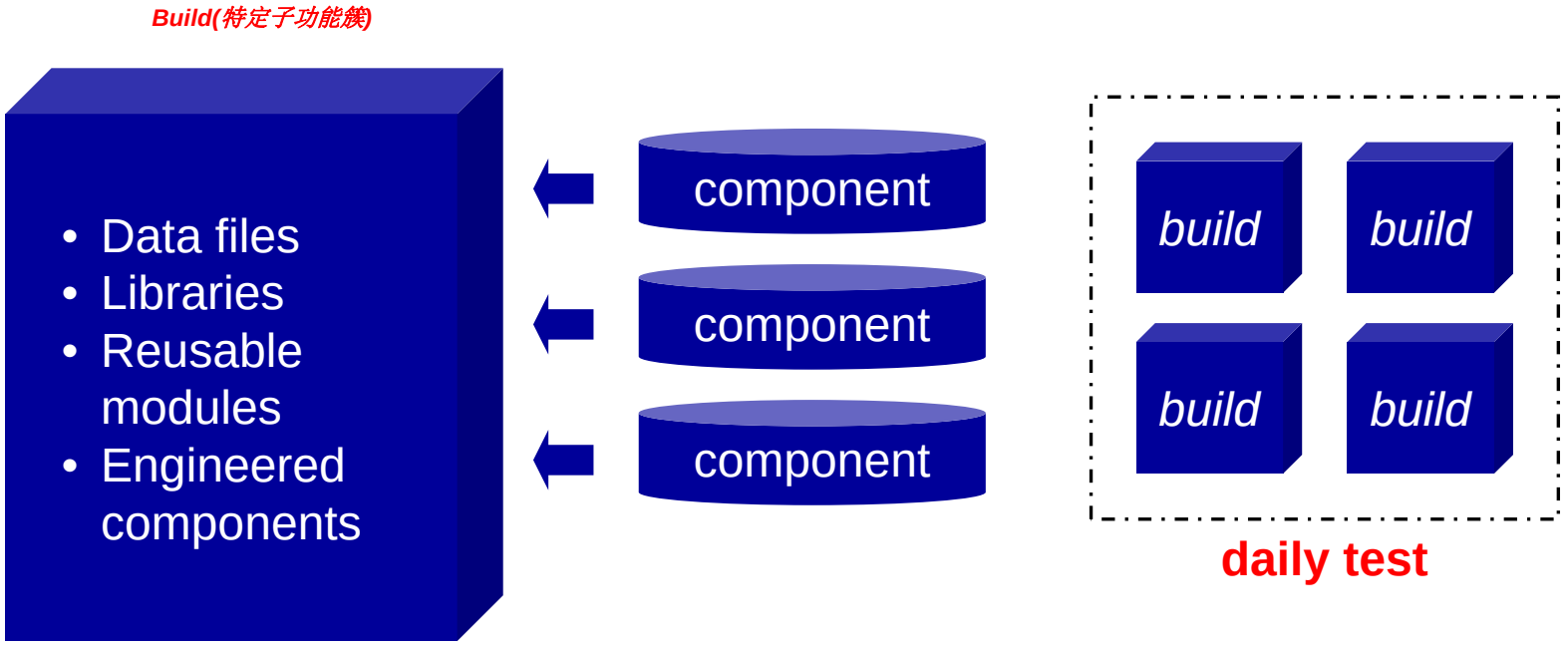
no significant data can flow upward.

Bottom-up



Regression（回归） testing:
to re-execute some subset of tests to ensure that ***changes*** have not propagated unintended side effects.

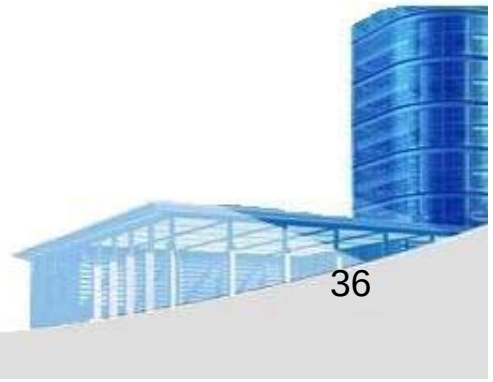
- **Smoke testing** (Why is it called “smoke” testing?)



Good for *time-critical* projects.

- **General Testing Criteria**

  - *Interface integrity* – internal and external module interfaces are tested as each module or cluster is added to the software
  - *Functional validity* – test to uncover functional defects in the software
  - *Information content* – test for errors in local or global data structures
  - *Performance* – verify specified performance bounds are tested

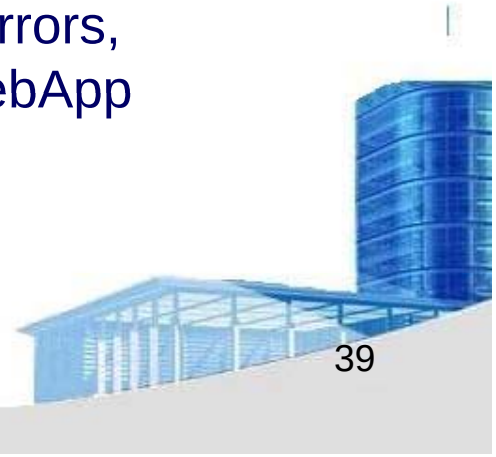


- **For OO (Object-Oriented) Software**

  - **Unit Testing == Class Testing**
    - must not test operations in isolation (***hierarchy***)
    - driven by class ***operations*** and ***state behavior***, not algorithmic detail and data flow across module interface
  - **Integration Testing**
    - **thread-based testing** - testing all classes required to respond to one system input or event
    - **use-based testing** - begins by testing independent classes (classes that use very few server classes) first and then dependent classes that make use of them
    - **cluster testing** - groups of collaborating classes are tested for interaction errors

- **WebApp Testing - I**
  - The content model for the WebApp is reviewed to uncover errors.
  - The interface model is reviewed to ensure that all use cases can be accommodated.
  - The design model for the WebApp is reviewed to uncover navigation errors.
  - The user interface is tested to uncover errors in presentation and/or navigation mechanics.
  - Each functional component is unit tested.

- **WebApp Testing - II**
  - Navigation throughout the architecture is tested.
  - The WebApp is implemented in a variety of different environmental configurations and is tested for compatibility with each configuration.
  - Security tests are conducted in an attempt to exploit vulnerabilities in the WebApp or within its environment.
  - Performance tests are conducted.
  - The WebApp is tested by a controlled and monitored population of end-users. The results of their interaction with the system are evaluated for content and navigation errors, usability concerns, compatibility concerns, and WebApp reliability and performance.

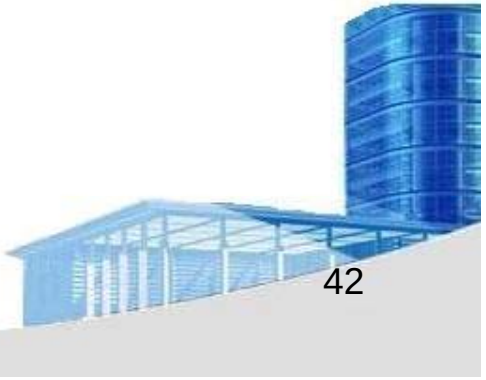


- **MobileApp Testing**

  - *User experience testing* – ensuring app meets stakeholder usability and accessibility expectations
  - *Device compatibility testing* – testing on multiple devices
  - *Performance testing* – testing non-functional requirements
  - *Connectivity testing* – testing ability of app to connect reliably
  - *Security testing* – ensuring app meets stakeholder security expectations
  - *Testing-in-the-wild* – testing app on user devices in actual user environments
  - *Certification testing* – app meets the distribution standards

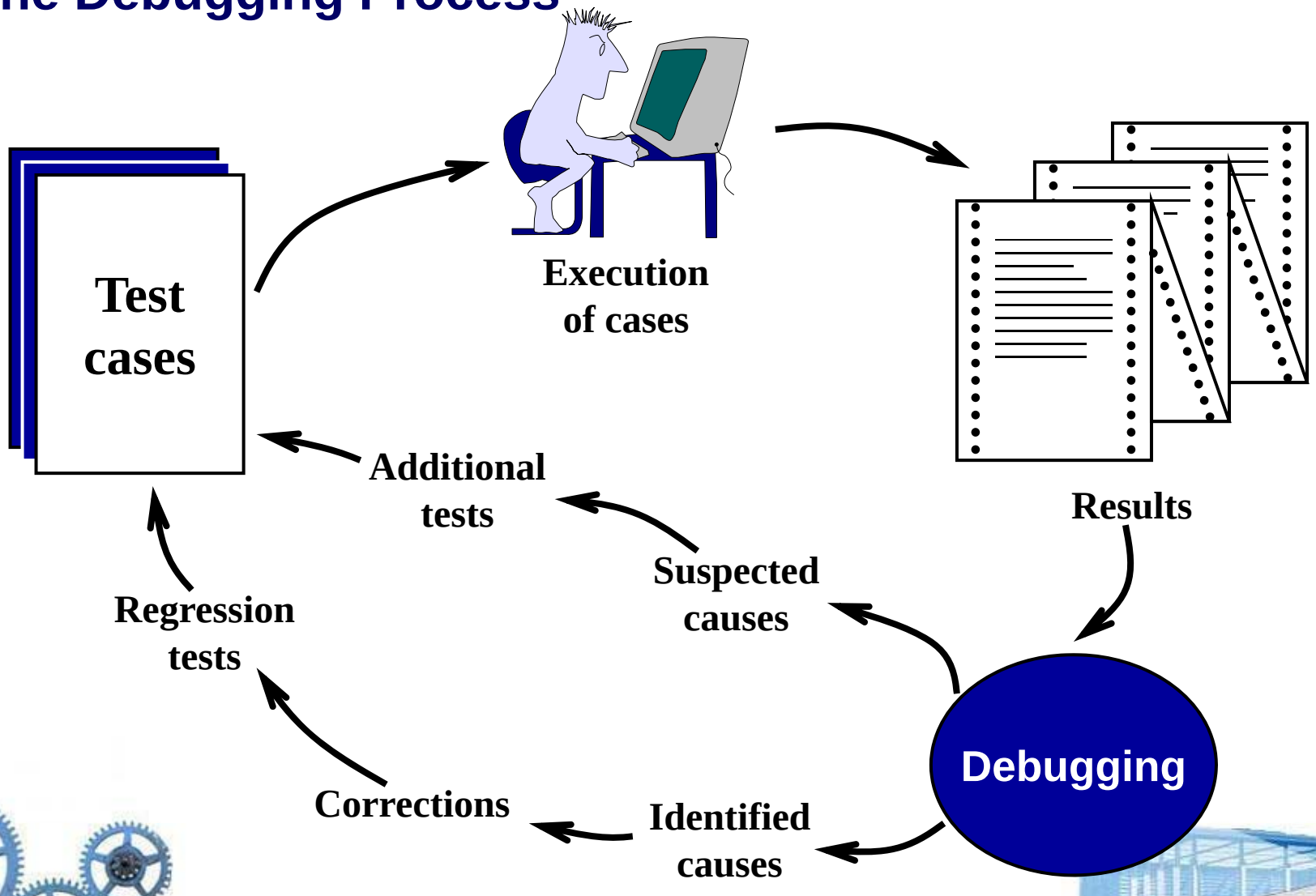
- **High Order Testing**
  - *Validation testing*
    - Focus is on software requirements
  - *System testing*
    - Focus is on system integration
  - *Alpha/Beta testing*
    - Focus is on customer usage
  - *Recovery testing*
    - forces the software to fail in a variety of ways and verifies that recovery is properly performed
  - *Security testing*
    - verifies that protection mechanisms built into a system will, in fact, protect it from improper penetration
  - *Stress testing*
    - executes a system in a manner that demands resources in abnormal quantity, frequency, or volume
  - *Performance Testing*
    - test the run-time performance of software within the context of an integrated system

- **Debugging: A Diagnostic Process**

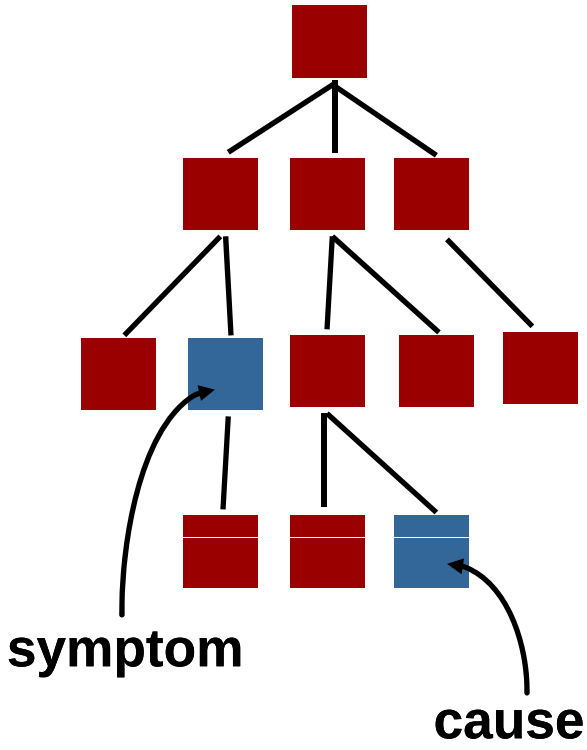


The Art of Debugging

- The Debugging Process



- **Symptoms & Causes**

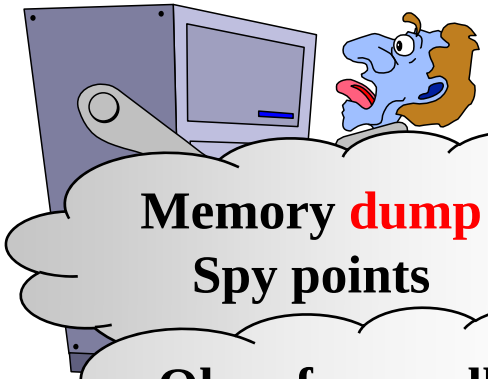


- **symptom and cause may be geographically separated**
- **symptom may disappear when another problem is fixed**
- **cause may be due to a combination of non-errors**
- **cause may be due to a system or compiler error**
- **cause may be due to assumptions that everyone believes**
- **symptom may be intermittent(间歇性)**

The Art of Debugging

- The Debugging Techniques

- brute force（蛮力） testing
- Backtracking（回溯法）
- Induction(归纳)
- Deduction（推演）



Memory dump
Spy points

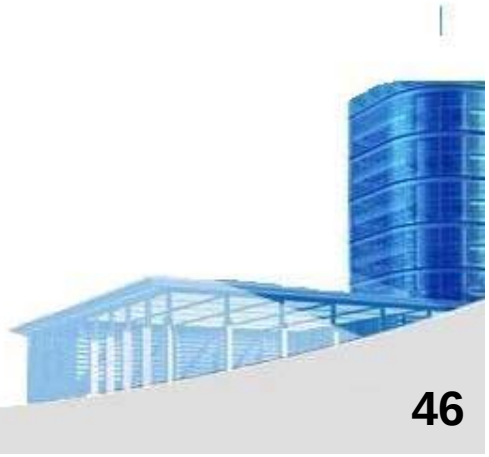
Okay for small programs

Cause elimination

	Yes	No
What		
Where		
When		
How		

3W1H Table

- **Bug Removal Considerations**
  - Is the cause of the bug reproduced in another part of the program?
  - What "next bug" might be introduced by the fix that is being proposed?
  - What could have been done to prevent this bug in the first place?



- **Final Thoughts**

  - Think -- before you act to correct;
  - Use tools （Automated Debugging） to gain additional insight;
  - If you’re at an impasse(僵局), get help from someone else;
  - Once you correct the bug, use regression testing to uncover any side effects.

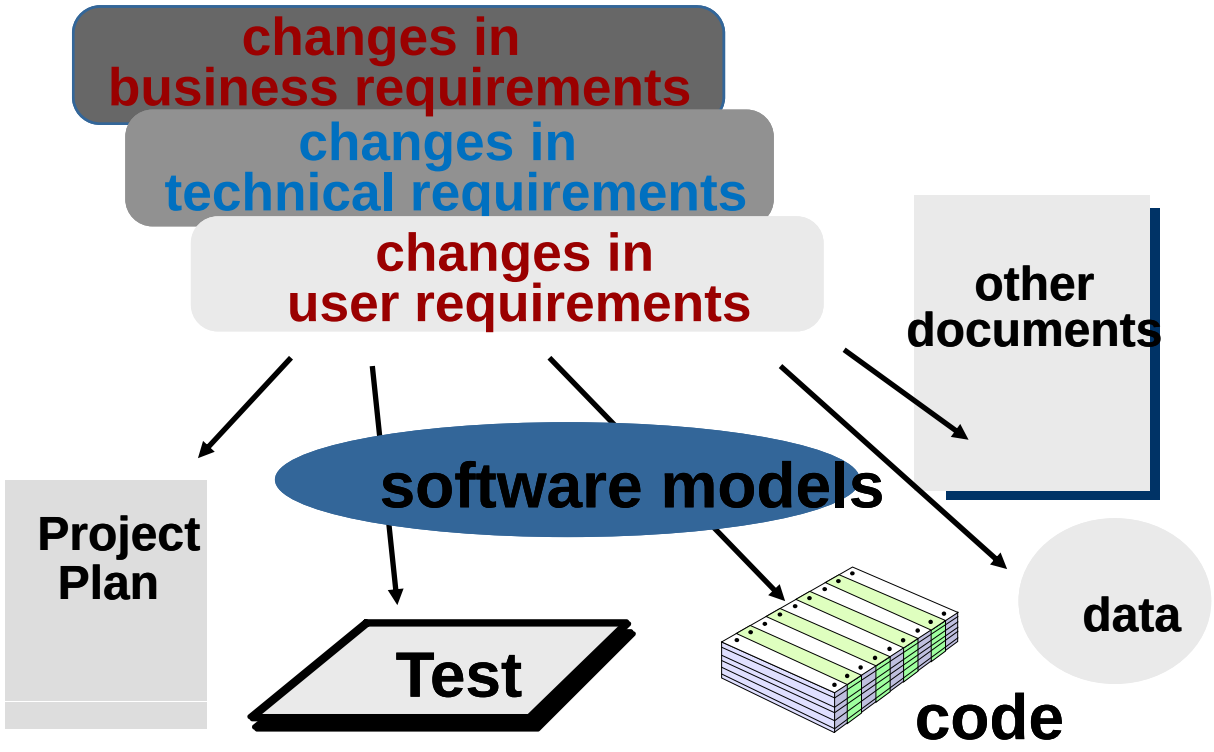
Ch.29 Software Configuration Management

• The “First Law”

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

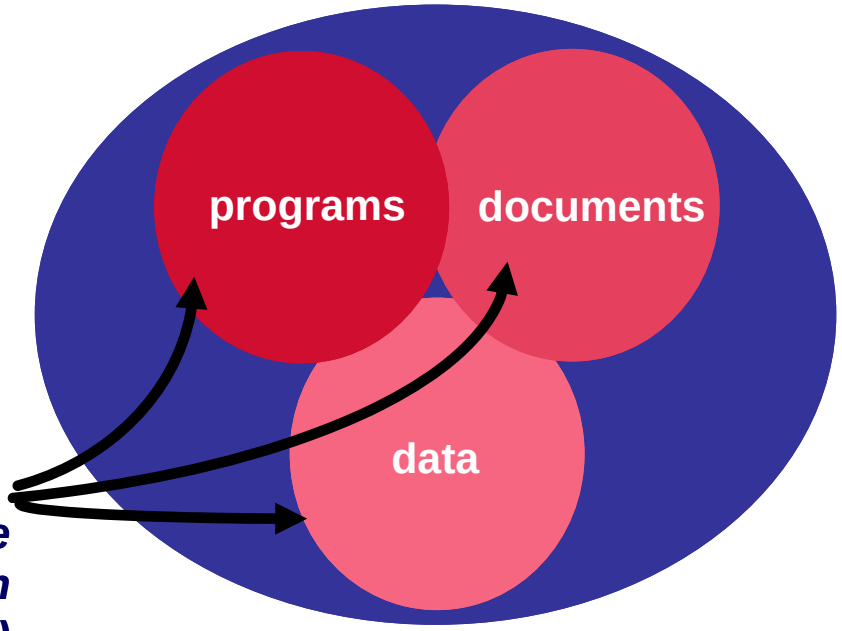
Bersoff, et al, 1980

- **What Are These Changes?**



Software Configuration Management [SCM]

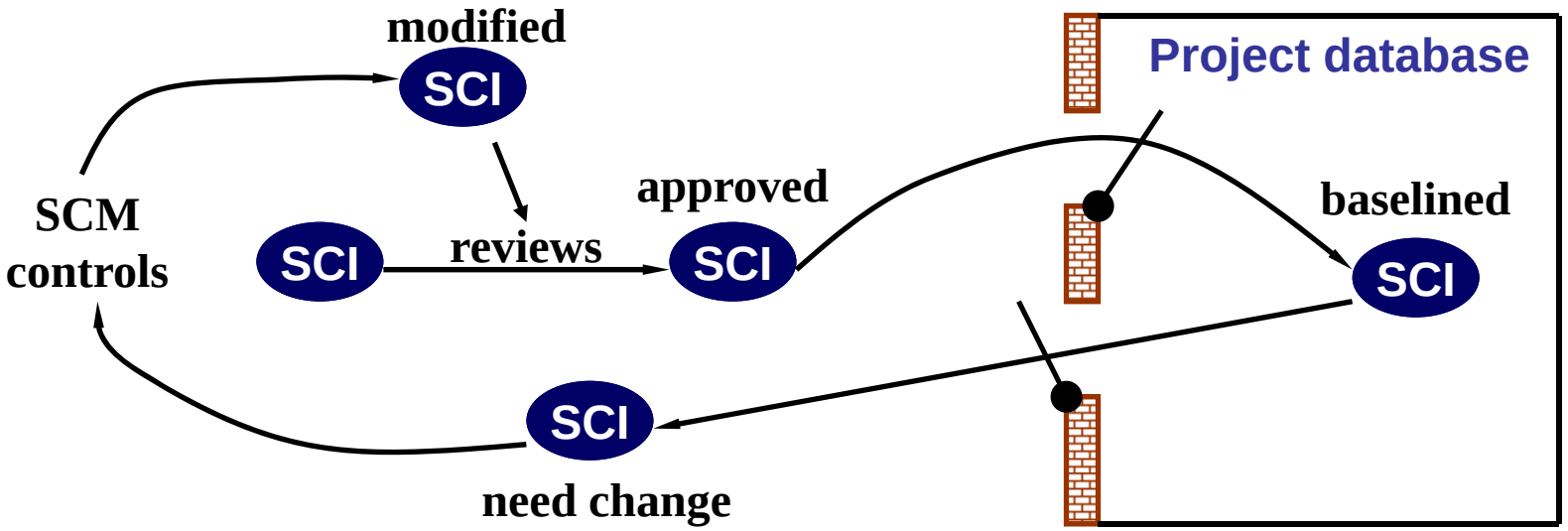
- **Software Configuration:**
  The items that comprise all information produced as part of the software process (programs, data, documents, and more …) are collectively called a software configuration.



Software Configuration Items (SCI)

- **Baselines (IEEE Std. No.610.12-1990):** 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.

Software Configuration Management (SCM)



Baselines:

System Specification
Design Specification
Test Plans / Procedures / Data
Software Requirements
Source Code
Operational System

• The SCM Process

1. **Identification:** 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?
2. **Version Control:** How does an organization control changes before and after software is released to a customer?
3. **Change Control:** Who has responsibility for approving and ranking changes?
4. **Configuration Auditing:** How can we ensure that changes have been made properly?
5. **Reporting:** What mechanism is used to appraise others of changes that are made?

- **SCM Repository(中心存储库)**
  - The SCM repository is the set of mechanisms and data structures that allow a software team to manage change in an effective manner
  - The repository performs or precipitates the following functions [For89]:
    - Data integrity
    - Information sharing
    - Tool integration
    - Data integration
    - Methodology enforcement(实施)
    - Document standardization

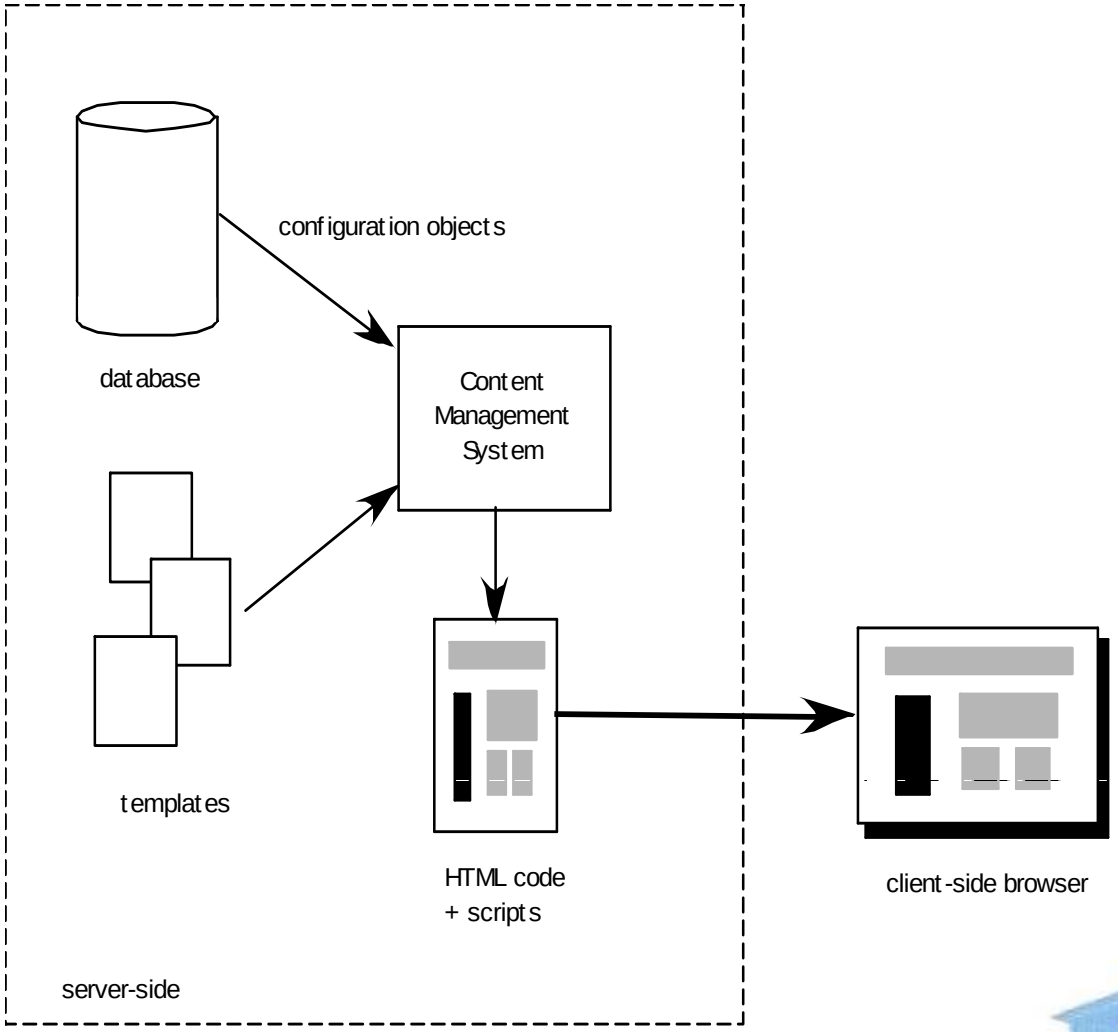
- **SCM for Web and Mobile Engineering - I**
  - *Content.*
    - A typical Web or Mobile App contains a vast array of content—text, graphics, applets（Java程序）, scripts, audio/video files, forms, active page elements, tables, streaming data, and many others.
    - The challenge is to organize this sea of content into a rational set of configuration objects (Section 29.2.1) and then establish appropriate configuration control mechanisms for these objects.
  - *People.*
    - Because a significant percentage of Web and Mobile App development continues to be conducted in an ad hoc（特别的） manner, any person involved in the App can (and often does) create content.

- **SCM for Web and Mobile Engineering - II**
  - *Scalability.*
    - As size and complexity grow, small changes can have far-reaching and unintended affects that can be problematic. Therefore, the rigor of configuration control mechanisms should be directly proportional to application scale.
  - *Politics.*
    - Who 'owns' a App?
    - Who assumes responsibility for the accuracy of the information displayed by the App?
    - Who assures that quality control processes have been followed before information is published to the site?
    - Who is responsible for making changes?
    - Who assumes the cost of change?

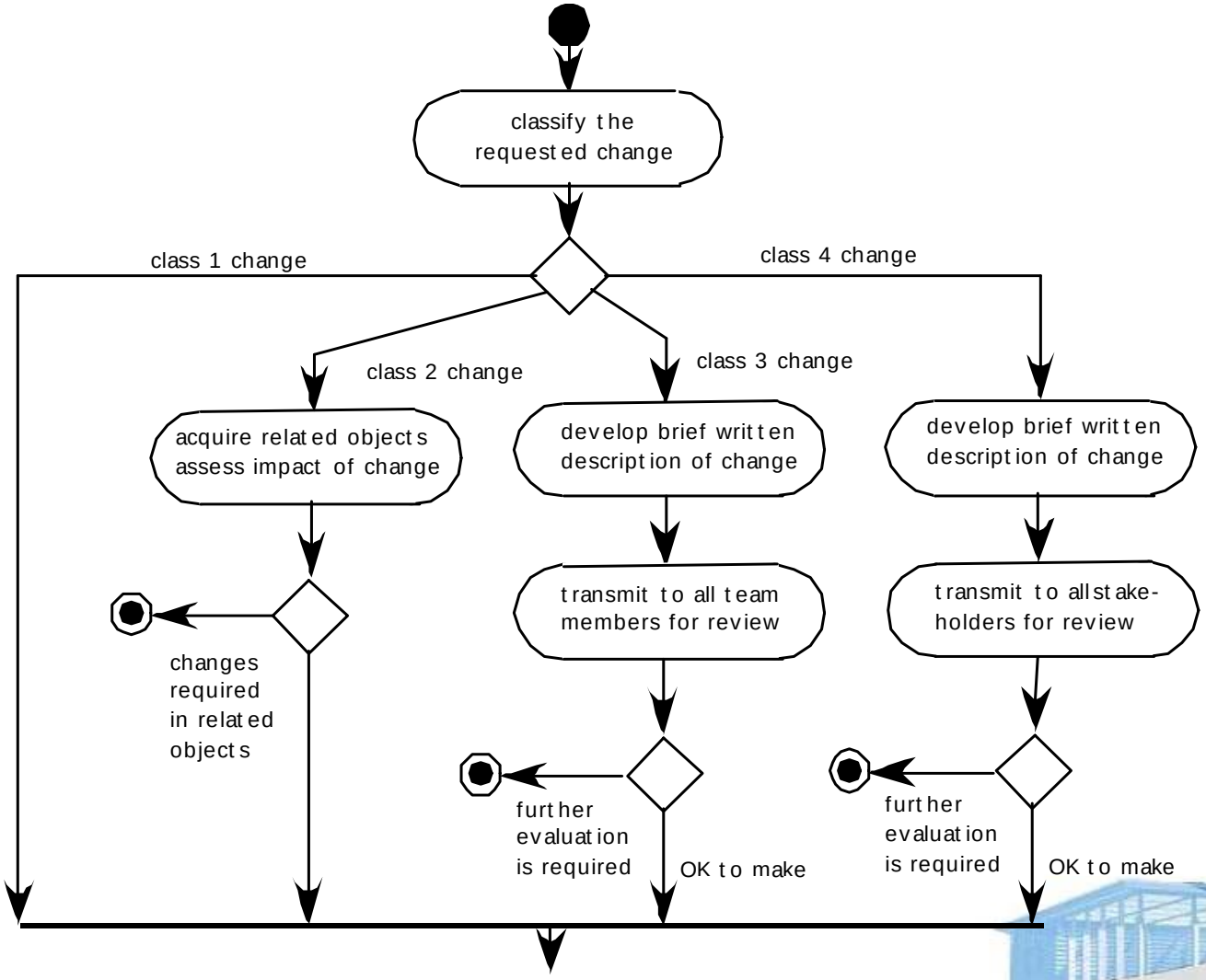
- **Content Management - I**
  - *The collection subsystem* encompasses all actions required to create and/or acquire content, and the technical functions that are necessary to
    - convert content into a form that can be represented by a mark-up language (e.g., HTML, XML)
    - organize content into packets that can be displayed effectively on the client-side.
  - *The management subsystem* implements a repository that encompasses the following elements:
    - **Content database**—the information structure that has been established to store all content objects
    - **Database capabilities**—functions that enable the CMS to search for specific content objects (or categories of objects), store and retrieve objects, and manage the file structure that has been established for the content
    - **Configuration management functions**—the functional elements and associated workflow that support content object identification, version control, change management, change auditing, and reporting.

- **Content Management - II**
  - The *publishing subsystem* extracts from the repository, converts it to a form that is amenable to publication, and formats it so that it can be transmitted to client-side browsers. The publishing subsystem accomplishes these tasks using a series of templates.
  - Each template is a function that builds a publication using one of three different components [BOI02]:
    - **Static elements**—text, graphics, media, and scripts that require no further processing are transmitted directly to the client-side
    - **Publication services**—function calls to specific retrieval（检索） and formatting services that personalize content (using predefined rules), perform data conversion, and build appropriate navigation links.
    - **External services**—provide access to external corporate information infrastructure such as enterprise data or “back-room” applications.

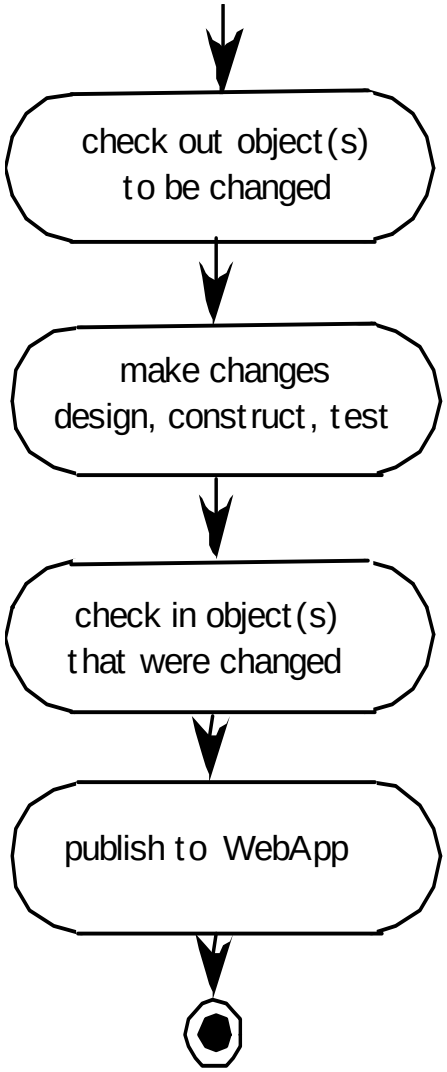
- **Content Management - III**



- **Change Management for Web and Mobile Apps - I**



- **Change Management for Web and Mobile Apps - II**



《 Research on Design Patterns 》

Due: June 1, 2015

Minimum requirement of contents:

Review of latest research papers (10 points);

Analysis on the architecture of your subsystem (15 points);

Design pattern applied to your subsystem (20 points);

Discussions on other design patterns which may also support your subsystem (10 points);

References (3 points).

Concerned points:

The pattern description must be complete.

The language and style of the document must be uniformed (2 points).

Grading: The full mark = 60 points * number of participants

《Test Specification》

Due: June 8, 2015

Minimum requirement of contents:

Overall Plan (10 points);
Functional Testing (10 points);
Boundary Testing (4 points);
Stress Testing (4 points);
Interface Communication Testing with Other Groups' Modules (5 points).
Each group is supposed to provide all the necessary testing databases and detailed test cases. (4 points)

Concerned points:

The plan must be complete and operable.
The language and style of the document must be uniformed (3 points).

Grading: The full mark = 40 points. Only partial participation is required. This part will be graded together with the subsystem version 1.0.

Tasks

- **Review** Ch.16, 22, 29
- **Finish** “Problems and points to ponder” in Ch. 16, 22, 29
- **Preview** Ch.23，24
- Prepare for Pattern-based Design Report (Due June 1)
- Prepare for Testing Plan (Due June 8)