UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

TOPIC 10

SYSTEM CONSTRUCTION, TESTING AND INSTALLATION

ANALISIS DAN PERANCANGAN SISTEM INFORMASI

CSIM603183

Learning Objectives

1. Able to explain various types of system testing and the usage
2. Able to create system test plan
3. Able to explain how to deploy an information systems

Introduction

- Programming can be **the largest single component** of any systems development project **in terms of time and cost.**
- However, it also can **be the best understood component** and therefore—except in rare circumstances — **offers the fewest problems** of all aspects of system development.
- When projects fail, it is usually not because the programmers were unable to write the programs, but because the analysis, design, installation, and/or project management were done poorly.

Introduction

- ***System construction*** is the development of all parts of the system, including the software itself, documentation, and new operating procedures.
- Construction phase of the Enhanced Unified Process deals predominantly with:
  - Implementation,
  - Testing, and
  - Configuration and Change Management.

Implementation & Testing Phase

- Implementation obviously deals with **programming**. Programming is often seen as the focal point of systems development.
  - After all, systems development is writing programs.
  - It is the reason we do all the analysis and design.
- Many beginning programmers see testing and documentation as bothersome afterthoughts.
- **Testing and documentation** aren't fun, so they often **receive less attention** than the creative activity of writing programs.

Implementation & Testing Phase

- Programming and testing are very similar to writing and editing.
- Thorough testing is the hallmark of professional software developers.
- Most professional organizations devote more time and money to testing (and the subsequent revision and retesting) than to writing the programs in the first place.

The Importance of Testing

- **The reasons are simple economics**: Downtime and failures caused by software bugs are extremely expensive.
- Many large organizations estimate the costs of downtime of critical applications at $50,000 to $200,000 per hour.
- One serious bug that causes an hour of downtime can cost more than one year's salary of a programmer—and how often are bugs found and fixed in one hour?
- **Testing is, therefore, a form of insurance.**
- Organizations are willing to spend a lot of time and money to prevent the possibility of major failures after the system is installed.

The Configuration and Change Management

- The Configuration and Change Management workflow **keeps track of the state of the evolving system.**
  - The evolving information system comprises a set of artifacts that include, for example, diagrams, source code, and executable.
  - During the development process, these artifacts are modified.
  - The amount of work, and hence dollars, that goes into the development of the artifacts is substantial.
  - Therefore, the artifacts themselves should be handled as any expensive asset would be handled: Access controls must be put into place to safeguard the artifacts from being stolen or destroyed.

The Configuration and Change Management

- The **traceability** of the artifacts back through the various artifacts developed, such as data management layer designs, class diagrams, package diagrams, and use-case diagrams, to the specific requirements is also very important.
- Without this traceability, we will not know which aspects of a system to modify when —not if— the requirements change

UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

MANAGING PROGRAMMING

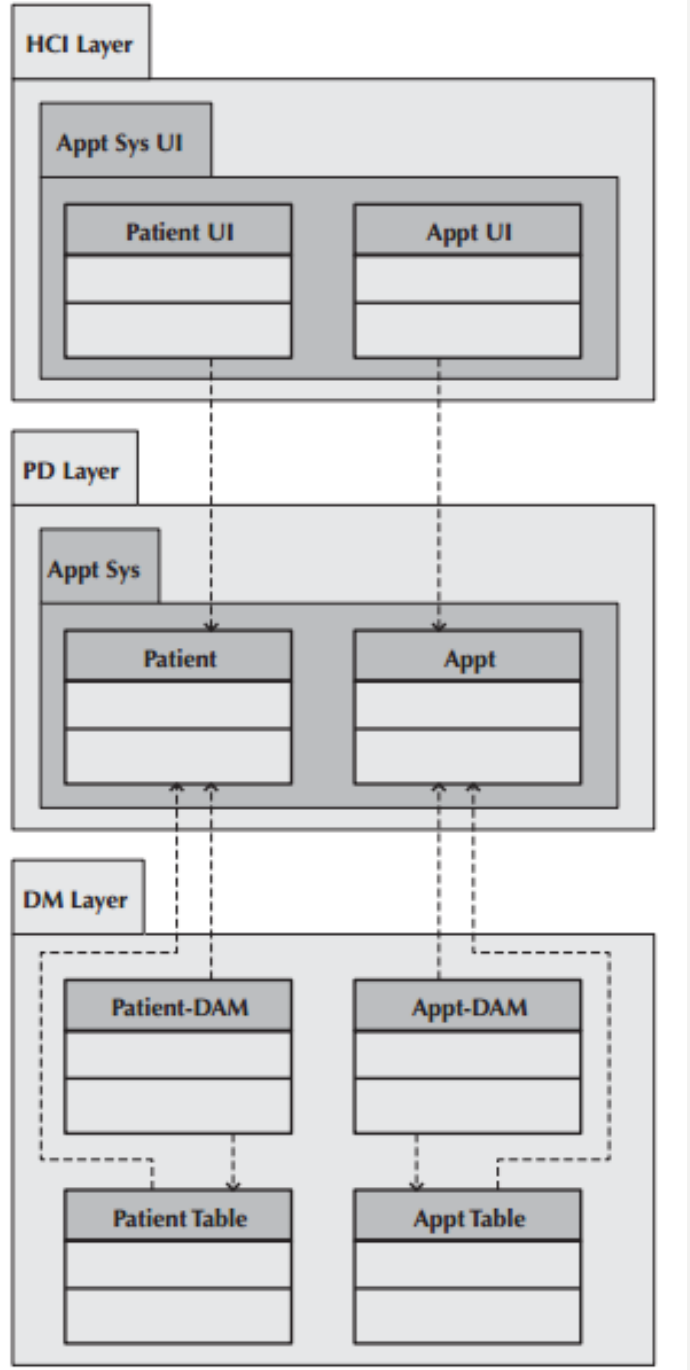
What does project manager do?

- Assigning Programmer
- Coordinating Activities
- Managing the Schedule
- Managing Cultural Issues

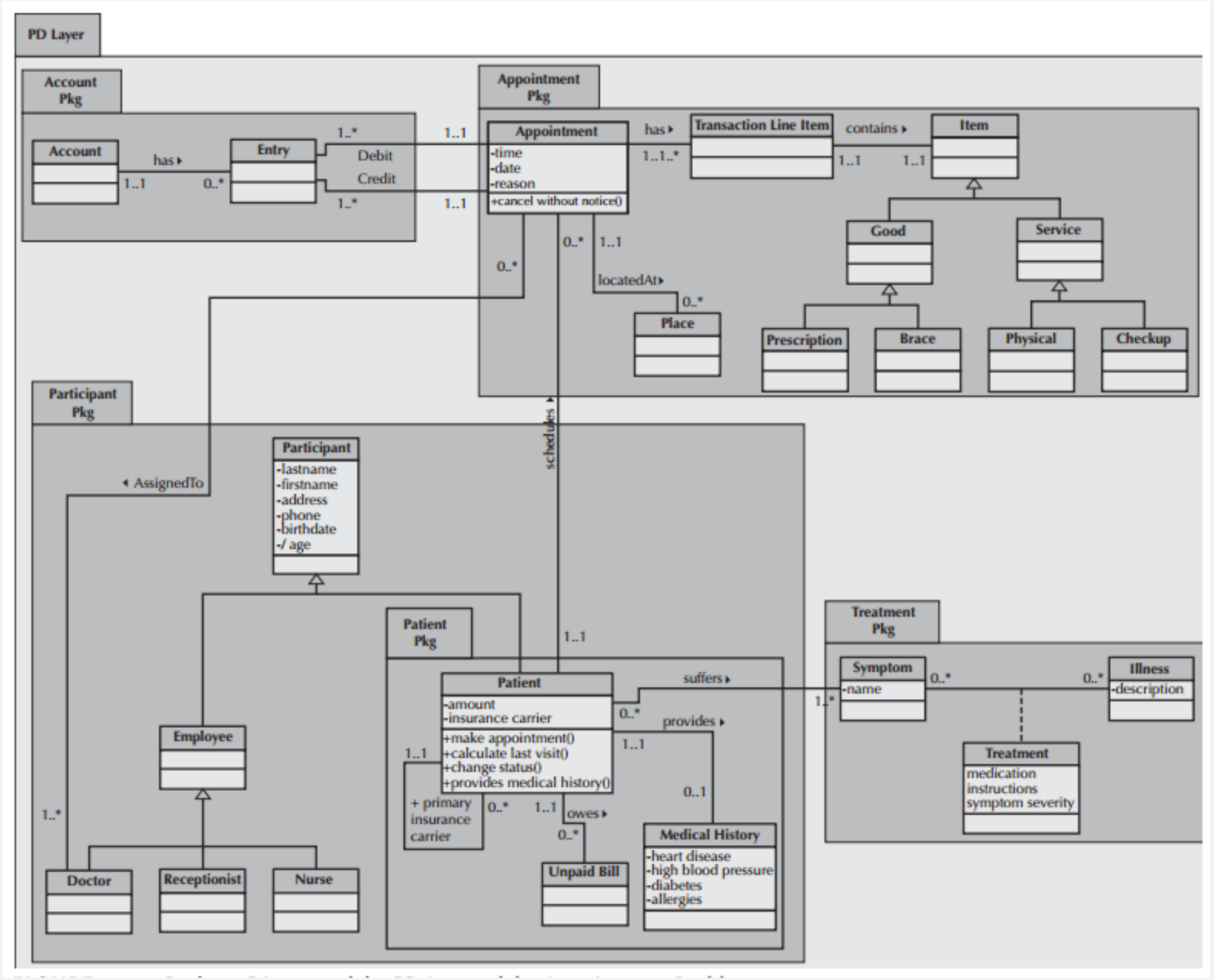
Assigning Programming

- The project manager first **groups together classes** that are **related** so that each programmer is working on related classes.
  - cohesion should be maximized and coupling should be minimized
- These groups of classes are then assigned to programmers.
- A good place to start is to look at the **package diagrams.**
  - **The idea of packages as a way to group UML artifact (use case, class diagram, etc.) together to make them easier to read and to keep the models at a reasonable level of complexity.**

Example Package Diagram



Example Package Diagram



Assigning Programming

- One of the rules of systems development is that the **more programmers** who are involved in a project, **the longer the system will take to build.**
- This is because as the size of the programming team increases, the need for **coordination** increases exponentially, and the more coordination required, the less time programmers can spend actually writing systems.
- The best size is the smallest possible programming team.
- When projects are so complex that they require a large team, the best strategy is to try to **break the project into a series of smaller parts** that can function as independently as possible.

Coordinating Activities

1. Have a **weekly project meeting** to discuss any changes to the system that have arisen during the past week—or any issues that have come up.
2. Create and follow **standards** that can range from formal rules for naming fi les, to forms that must be completed when goals are reached, to programming guidelines.
3. Set up **three areas** in which programmers can work: a development area, a testing area, and a production area.

Coordinating Activities

4. Manage **change control**, the action of coordinating a system as it changes through construction.
   - By keeping track of which programmer changes which classes and packages by using a program log.
   - The log is merely a form on which programmers sign out classes and packages to write and sign in when they are completed.
   - Both the programming areas and program log help the analysts understand exactly who has worked on what and the system's current status.
   - Without these techniques, files can be put into production without the proper testing (e.g., two programmers can start working on the same class or package at the same time).
5. Utilize **CASE tool**.

Managing the Schedule

- The time estimates that were produced during project identification and refined during analysis and design almost always need to be refined as the project progresses during construction because it is virtually impossible to develop an exact assessment of the project's schedule.
  - If a program module takes longer to develop than expected, then the prudent response is to move the expected completion date later by the same amount.
  - Manage **scope creep**

Managing Cultural Issues (Hall & Hofstede)

1. Speed of messages
2. Context
3. Time
4. Power Distance
5. Uncertainty avoidance
6. Individualism vs collectivism
7. Masculinity vs femininity
8. Long vs short term orientation

UNIVERSITAS
INDONESIA

Veritas, Probitas, Iustitia

FAKULTAS
ILMU
KOMPUTER

DEVELOPING DOCUMENTATION

Basic Types of Documentation

There are two fundamentally different types of documentation:

1. **system documentation**
2. **and user documentation.**

System Documentation

- System documentation is intended to help **programmers and systems analysts** understand the application software and enable them to build it or maintain it after the system is installed.
- System documentation is largely a by-product of the systems analysis and design process and is created as the project unfolds.

User Documentation

- **User documentation** (such as user's manuals, training manuals, and online help systems) is designed to help the user operate the system.
  - User documentation is often left until the end of the project, which is a dangerous strategy.
  - Developing good documentation takes longer than many people expect because it requires much more than simply writing a few pages → requires designing the documents, writing the text, editing the documents, and testing them.
  - For good-quality documentation, this process usually takes about three hours per page (single-spaced) for paper-based documentation or two hours per screen for online documentation.

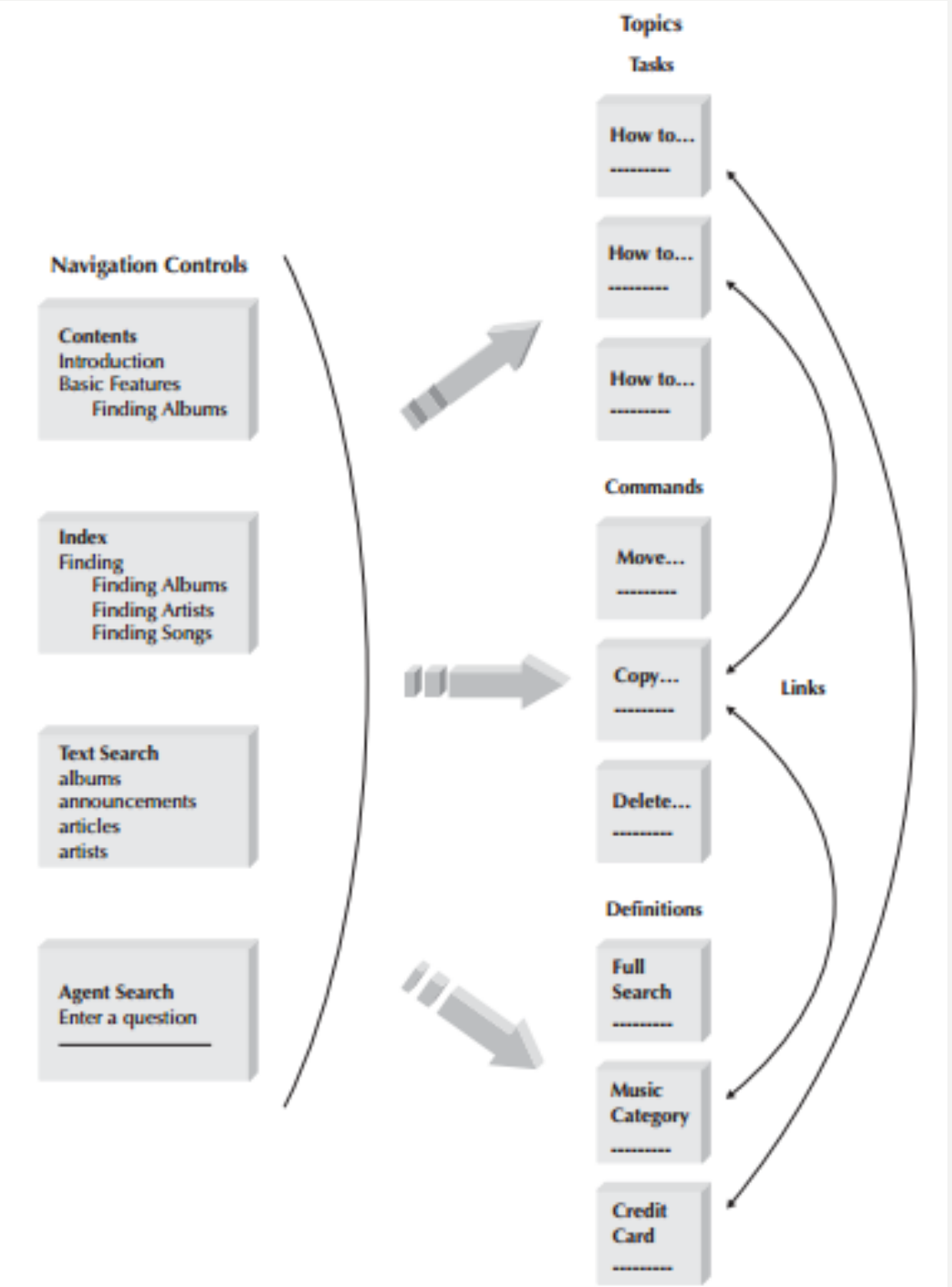
Types of User Documentation

1. **Reference documents** also called the help system) are designed to be used when the user needs to learn how to perform a specific function (e.g., updating a field, adding a new record).
   - Often people read reference information when they have tried and failed to perform the function; writing reference documents requires special care because the user is often impatient or frustrated when he or she begins to read them.
2. **Procedures manuals** describe how to perform business tasks (e.g., printing a monthly report, taking a customer order).
   - Each item in the procedures manual typically guides the user through a task that requires several functions or steps in the system. Therefore, each entry is typically much longer than an entry in a reference document.

Types of User Documentation

3. **Tutorials** — obviously — teach people how to use major components of a system (e.g., an introduction to the basic operations of the system).
    - Each entry in the tutorial is typically longer still than the entries in procedures manuals, and the entries are usually designed to be read in sequence (whereas entries in reference documents and procedures manuals are designed to be read individually).

Designing Documentation Structure



UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

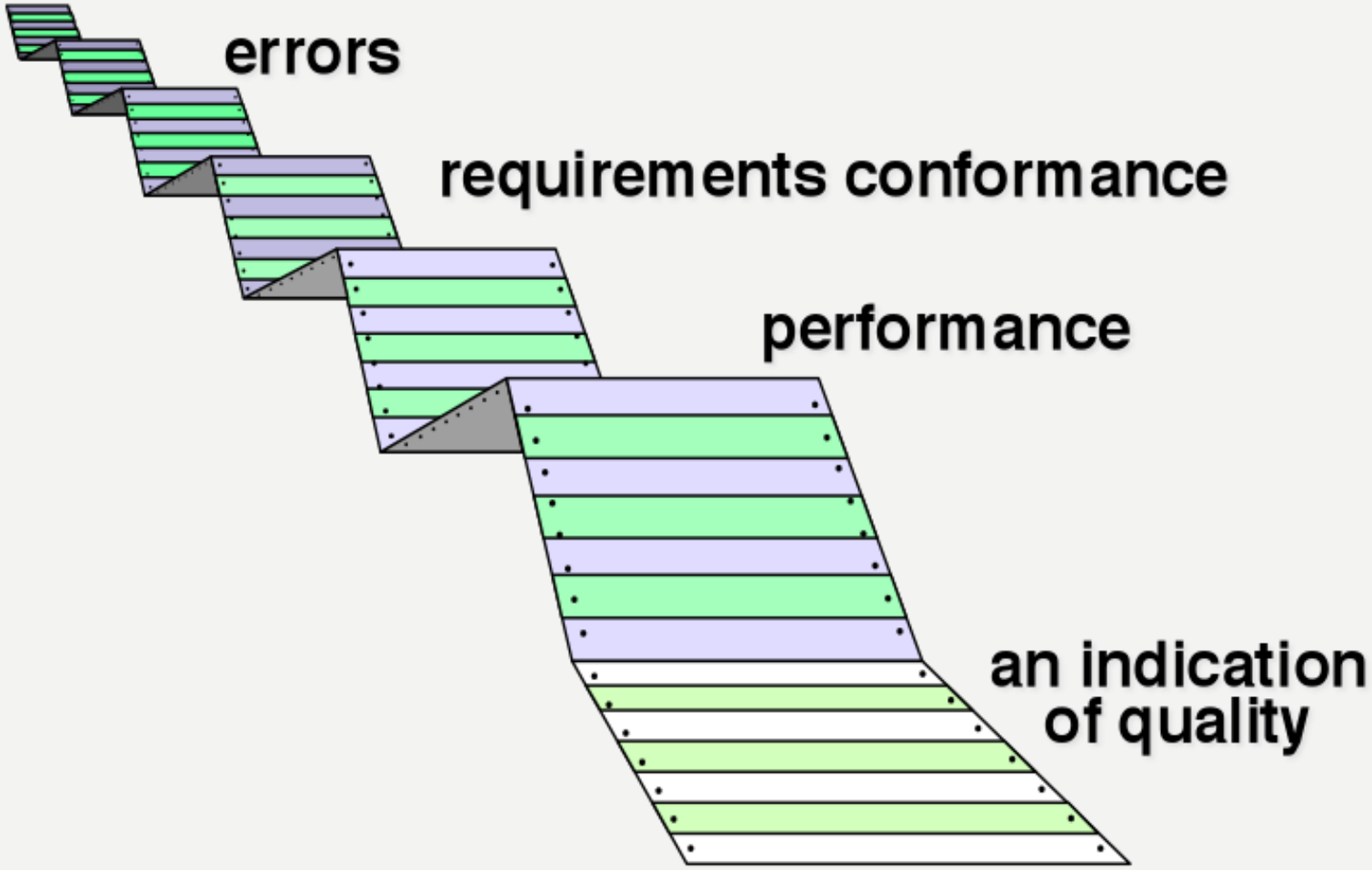
FAKULTAS ILMU KOMPUTER

DESIGNING SYSTEM TESTS

Designing Tests

- **Testing** is the *process of exercising* a program with the *specific intent of finding errors* prior to delivery to the end user.
- The purpose of testing is **not** to demonstrate that the system is **free of errors**, but **to uncover as many errors as feasible.**
- Testing is more critical to object-oriented systems than to systems developed in the past. Based on encapsulation (and information hiding), polymorphism (and dynamic binding), inheritance, reuse, and the actual object-oriented products, thorough testing is much more difficult and critical.

What Testing Shows



Testing Myths

1. Testing is too expensive.
2. Testing is time consuming.
3. Testing cannot be started if the product is not fully developed.
4. Complete Testing is Possible.
5. If the software is tested then it must be bug free.
6. Missed defects are due to Testers.
7. Testers should be responsible for the quality of a product.
8. Any one can test a Software application.
9. A tester’s task is only to find bugs.

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.
- Testing is **conducted by the developer of the systems and (for large projects) an independent test group**.
- **Testing and debugging are different activities**, but debugging must be accommodated in any testing strategy.

Verification and Validation

Verification refers to the set of tasks that ensure that the systems correctly implements a specific function.

Verification: "Are we building the product right?"

Validation refers to a different set of tasks that ensure that the systems that has been built is traceable to customer requirements. Boehm [Boe81] states this another way:

Validation: "Are we building the right product?"

Who Tests the Software?

Developer

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

Independent Tester

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

Testing Strategy

- We begin by **"testing-in-the-small"** and move toward **"testing-in-the-large"**
- For OO systems
  - Our focus when "testing in the small" changes from an individual module (the conventional view) to an OO class that encompasses attributes and operations and implies communication and collaboration

Testing Strategy

1. Specify product requirements in a quantifiable manner long before testing commences.
2. State testing objectives explicitly.
3. Understand the users of the systems and develop a profile for each user category.
4. Develop a testing plan that emphasizes “rapid cycle testing.”

Testing Strategy

1. Build “robust” system that is designed to test itself
2. Use effective technical reviews as a filter prior to testing
3. Conduct technical reviews to assess the test strategy and test cases themselves.
4. Develop a continuous improvement approach for the testing process.

Testing and Object Oriented Systems Issue

1. Encapsulation and Information Hiding
2. Polymorphism and Dynamic Binding
3. Inheritance
4. Re-use

Testing and Object Oriented Systems Issue

- Encapsulation and Information Hiding
  - Encapsulation: Mechanism that combines the processes and data into a single object.
  - Information hiding: suggests only the information required to use an object be available outside the object
- Polymorphism and Dynamic Binding
  - Polymorphism: having the ability to take several forms, so OO systems can send the same message to a set of objects, which can be interpreted differently by different classes of objects.
  - Dynamic binding: the ability of OO systems to defer the data typing of objects to run time

Testing and Object Oriented Systems Issue

- Inheritance
  - Allows developers to define classes incrementally by reusing classes defined previously as the basis for new classes.
- Re-use

Encapsulation and Information Hiding (1)

- Encapsulation and information hiding allow processes and data to be combined to create holistic entities (i.e. Objects).
- However, business process is distributed over a set of collaborating classes and contained in the methods of those classes.
- So, testers need to know the effect that business process has on a system by looking at the state changes that take place in the system

Encapsulation and Information Hiding (2)

- A second issue is the definition of “unit” for unit testing.
  - It might be package, class, or method
- What is “the unit” to be tested?
  - In traditional approaches, the answer would be the process that is contained in a function.
  - However, the process in OO systems is distributed over a set of classes.
  - Therefore, testing individual methods makes no sense, as the answer is the class.

Encapsulation and Information Hiding (3)

- Third issue raised is the impact on integration testing
- In this case, objects can be aggregated to form aggregate object.
- **Classes can be grouped together in defferent ways to form collaborations.**
  - **Class A + B + C → Business Process A**
  - **Class A + B + D → Business Process B**
  - **Class A + C + D → Buisiness Process C**
- **How does we effectively do integration testing?**

Polymorphism and Dynamic Binding (1)

- Impact on both unit and integration testing.
- Because an individual business process is implemented through a set of methods distributed over a set of objects, the unit test makes no sense at the method level.
- However, with polymorphism and dynamic binding, the same method (a small part of the overall business process) can be implemented in many different object.
- Therefore, testing individual implementations of methods makes no sense.
- So, the unit test that make sense to test is the class

Polymorphism and Dynamic Binding (2)

- Furthermore, dynamic binding makes it impossible to know which implementation is going to be executed until the system does it.
- Therefore, integration testing becomes very challenging.

Inheritance

- Through the use of inheritance, bugs can be propagated instantaneously from a superclass to all its direct and indirect subclasses.
- However, the tests that are applicable to a superclass are also applicable to all its subclasses.
- All these issues impact unit and integration testing.

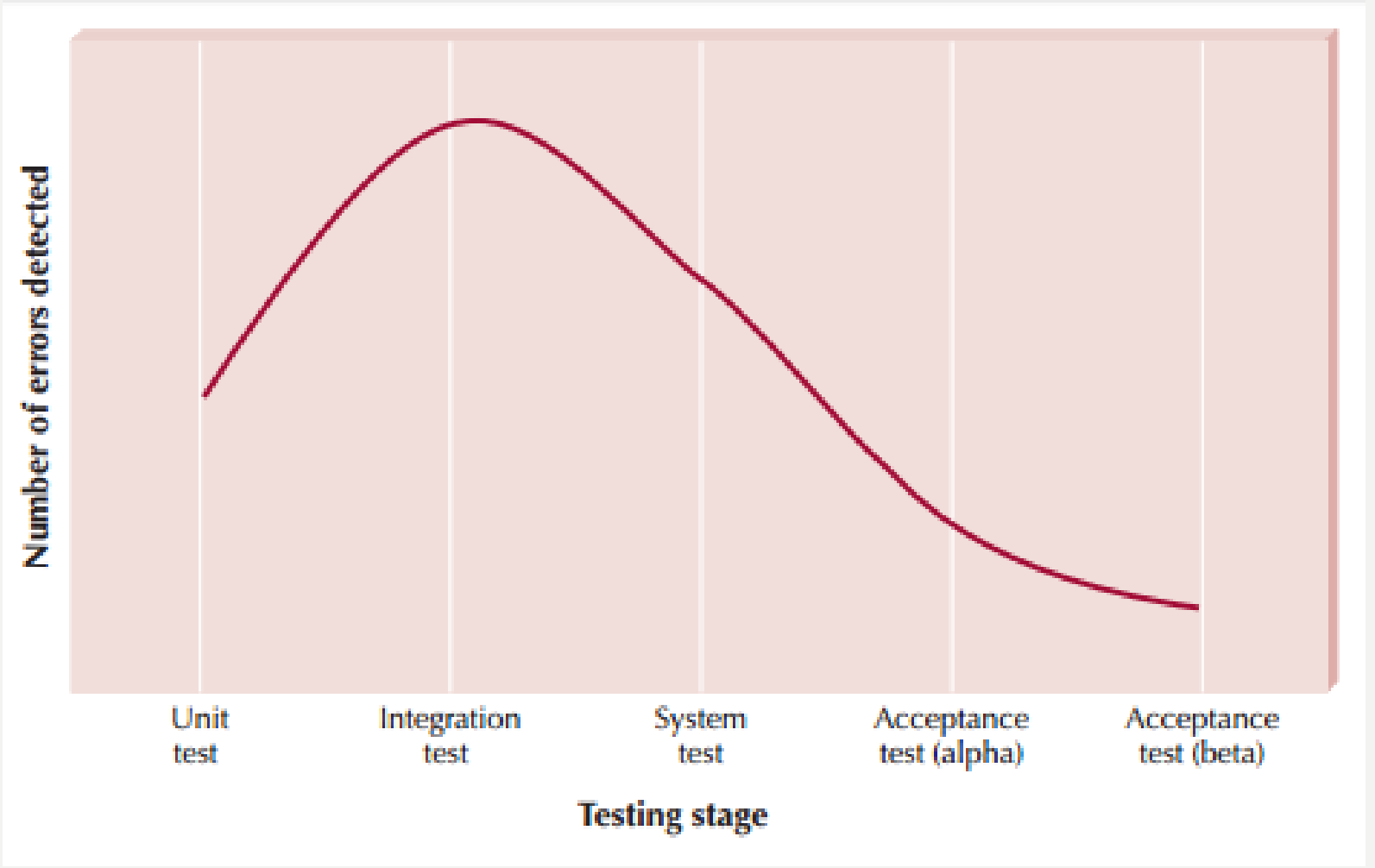
Re-use

- On the surface, reuse should decrease the amount of testing required.
- However, each time a class is used in a different context, the class must be tested again.
- Therefore, anytime a class library, framework, or component is used, unit testing and integration testing are important.

Four General Stages of Tests

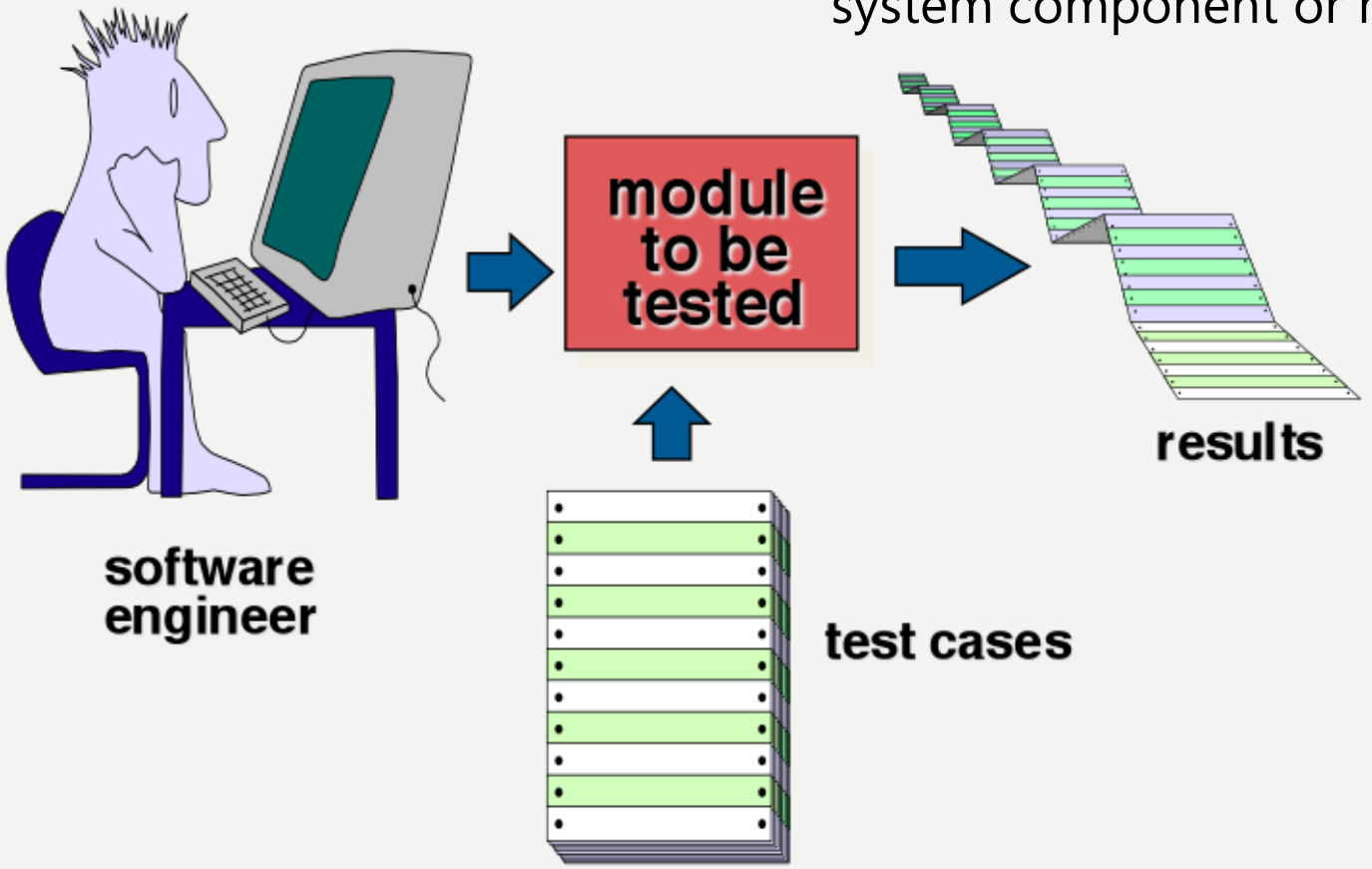
1. Unit tests
2. Integration tests
3. System test
4. Acceptance test

Error-Discovery Rates for Different Stages of Tests

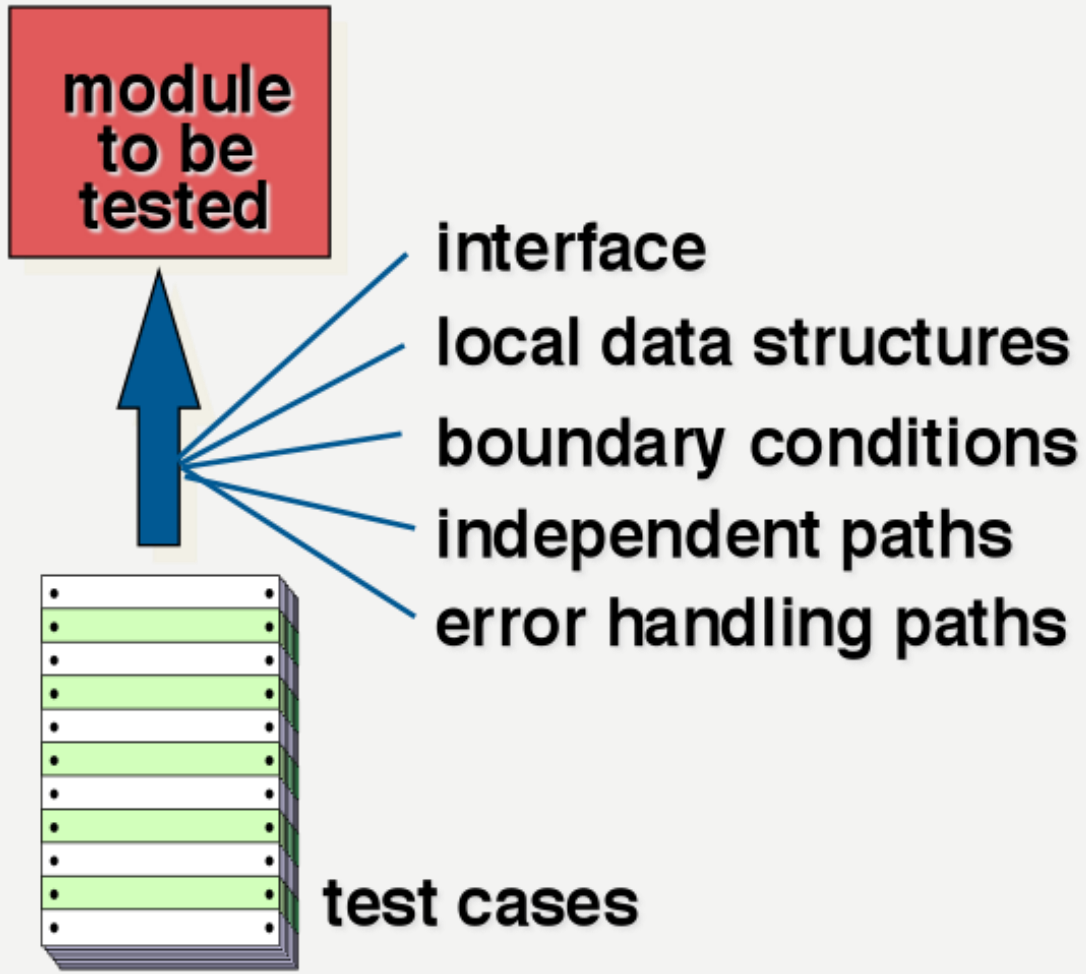


Unit Tests

focuses verification effort on the smallest unit of system design—the system component or module.



Unit Tests


module to be tested
interface
local data structures
boundary conditions
independent paths
error handling paths
test cases

Unit Tests

Stage	Types of Tests	Test Plan Source	When to Use	Notes
Unit Testing	**Black-Box Testing** Treats class as a black box **White-Box Testing** Looks inside the class to test its major elements	CRC Cards Class Diagrams Contracts Method Specifications	For normal unit testing When complexity is high	• Tester focuses on whether the class meets the requirements stated in the specifications. • By looking inside the class to review the code itself the tester may discover errors or assumptions not immediately obvious to someone treating the class as a black box.

Integration tests

- "If they all work individually, why do you doubt that they'll work when we put them together?"
- Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing.

Integration tests

- The *"Big Bang"* approach
  - All components are combined in advance. The entire program is tested as a whole.
  - Chaos usually becomes the result! A set of errors is encountered.
- An incremental construction strategy
  - The program is constructed and tested in small increments, where errors are easier to isolate and correct

Integration tests

Stage	Types of Tests	Test Plan Source	When to Use	Notes
Integration Testing	**User Interface Testing** The tester tests each interface function	Interface Design	For normal integration testing	• Testing is done by moving through each and every menu item in the interface either in a top-down or bottom-up manner.
	Use-Case Testing The tester tests each use case	Use Cases	When the user interface is important	• Testing is done by moving through each use case to ensure they work correctly. • Usually combined with user interface testing because it does not test all interfaces.
	Interaction Testing Tests each process in step-by-step fashion	Class Diagrams Sequence Diagrams Communication Diagrams	When the system performs data processing	• The entire system begins as a set of stubs. Each class is added in turn and the results of the class compared to the correct result from the test data; when a class passes, the next class is added and the test rerun. This is done for each package. Once each package has passed all tests, then the process repeats integrating the packages.
	System Interface Testing Tests the exchange of data with other systems	Use-Case Diagram	When the system exchanges data	• Because data transfers between systems are often automated and not monitored directly by the users it is critical to design tests to ensure they are being done correctly.

System tests

- System tests are usually conducted by the systems analysts to ensure that all classes work together without error.
- System testing is similar to integration testing but is much broader in scope.
- Whereas integration testing focuses on whether the classes work together without error, system tests examine how well the system meets business requirements and its usability, security, and performance under heavy load.

System tests

Stage	Types of Tests	Test Plan Source	When to Use	Notes
System Testing	**Requirements Testing** Tests to whether original business requirements are met	System Design, Unit Tests, and Integration Tests	For normal system testing	• Ensures that changes made as a result of integration testing did not create new errors. • Testers often pretend to be uninformed users and perform improper actions to ensure the system is immune to invalid actions (e.g., adding blank records).
	Usability Testing Tests how convenient the system is to use	Interface Design and Use Cases	When user interface is important	• Often done by analyst with experience in how users think and in good interface design. • Sometimes uses formal usability testing procedures discussed in Chapter 10.
	Security Testing Tests disaster recovery and unauthorized access	Infrastructure Design	When the system is important	• Security testing is a complex task, usually done by an infrastructure analyst assigned to the project. • In extreme cases, a professional firm may be hired.
	Performance Testing Examines the ability to perform under high loads	System Proposal Infrastructure Design	When the system is important	• High volumes of transactions are generated and given to the system. • Often done by using special purpose testing software.
	Documentation Testing Tests the accuracy of the documentation	Help System, Procedures, Tutorials	For normal system testing	• Analysts spot check or check every item on every page in all documentation to ensure the documentation items and examples work properly.

Acceptance tests

- Acceptance test is done primarily by the users with support from the project team.
- The goal is to confirm that system is complete, meets the business needs that prompted the system to be developed, and is acceptable to users.

Acceptance tests

Stage	Types of Tests	Test Plan Source	When to Use	Notes
Acceptance Testing	**Alpha Testing** Conducted by users to ensure they accept the system	System Tests	For normal acceptance testing	• Often repeats previous tests but are conducted by users themselves to ensure they accept the system.
	Beta Testing Uses real data, not test data	System Requirements	When the system is important	• Users closely monitor system for errors or useful improvements.

UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

CHANGE MANAGEMENT

Implementing Change

- Managing the change to a new system—whether it is computerized or not—is one of the most difficult tasks in any organization (Machiavelli).
- Because of the challenges involved, most organizations begin developing their conversion and change management plans while the programmers are still developing the software.

Cultural Issues and Information Technology Adoption

1. Speed of messages
2. Context
3. Time
4. Power Distance
5. Uncertainty avoidance
6. Individualism vs collectivism
7. Masculinity vs femininity
8. Long vs short term orientation

1. Speed of messages

- Speed of messages has implications for the development of documentation and training approaches.
- In a culture that values “deep” content, so that members of the culture can take their time to thoroughly understand the new system, simply providing an online help system is not going to be sufficient to ensure the successful adoption of the new information system.
- However, in a culture that prefers “fast” messages, an online help system could be sufficient.

2. Context

- Context also affects the adoption and deployment of a new system.
- In high-context cultures, it is expected that the new information system will be placed into the entire context of the enterprise wide system.
- Members of this type of society expect to be able to understand exactly where the system fits into the firm's overall picture.
- Like the speed of messages dimension, this affects the training approach used and the documentation developed

3. Time

- Time can also effect the adoption and deployment of a new system.
- In a polychronic time culture, the training could need to be spread out over a longer period of time, when compared to a monochronic time culture.
- In a monochromic time culture, interruptions would be considered rude. Consequently, training could be accomplished in a small set of intense sessions.
- However, with a polychronic time culture, because interruptions may occur frequently, maximum flexibility in setting up the training sessions may be necessary

4. Power distance

- Power distance addresses how power issues are dealt with in the culture.
  - For example, if a superior in an organization has an incorrect belief about an important issue, can a subordinate point out this error?
  - In some cultures, the answer is a resounding no.
- Consequently, this dimension could have major ramifications for the successful deployment of an information system.
  - For example, in a culture with a high power distance, the deployment of a new information system is dependent on the impression of the most important stakeholder. Therefore, much care must be taken to ensure that this stakeholder is pleased with the system.
  - Otherwise, it might never be used.

5. Uncertainty avoidance

- Uncertainty avoidance is based on the degree to which the culture depends on rules for direction, how well individuals in the culture handle stress, and the importance of employment stability.
- For example, in a high-uncertainty-avoidance culture, the use of detailed procedures manuals and good training can reduce the uncertainty in adopting the new system.

6. Individualism versus collectivism

- Individualism versus collectivism is based on the level of emphasis the culture places on the individual or the collective.
- The relationship between the individual and the group is important for the success of an information system.
- Depending on the culture's orientation, the success of an information system being transitioned into production can depend on whether the focus of the information system will benefit the individual or the group.

7. Masculinity versus femininity

- Masculinity versus femininity addresses how well masculine and feminine characteristics are valued by the culture.
- Some of the differences that could affect the adoption of an information system include employee motivational issues.
- In a masculine culture, motivation would be based on advancement, earnings, and training, whereas in a feminine culture, motivations would include friendly atmosphere, physical conditions, and cooperation.
- Depending on how the culture views this dimension, different motivations might need to be used to increase the likelihood of the information system being successfully deployed.

8. Long-versus short-term orientation

Long- versus short-term orientation, deals with how the culture views the past and the future. In East Asia, long-term thinking is highly respected, whereas in North America and Europe, short-term profits and the current stock price seem to be the only things that matter.

- For example, if the local culture views success only in a short-term manner, then any new information system that is deployed to support one department of an organization may give that department a competitive advantage over other departments in the short run.
- If only short-run measures are used to judge the success of a department, then it would be in the interest of the other departments to fight the successful deployment of the information system.
- However, if a longer-run perspective is the norm, then the other departments could be convinced to support the new information system because they could have new supportive information systems in the future

UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

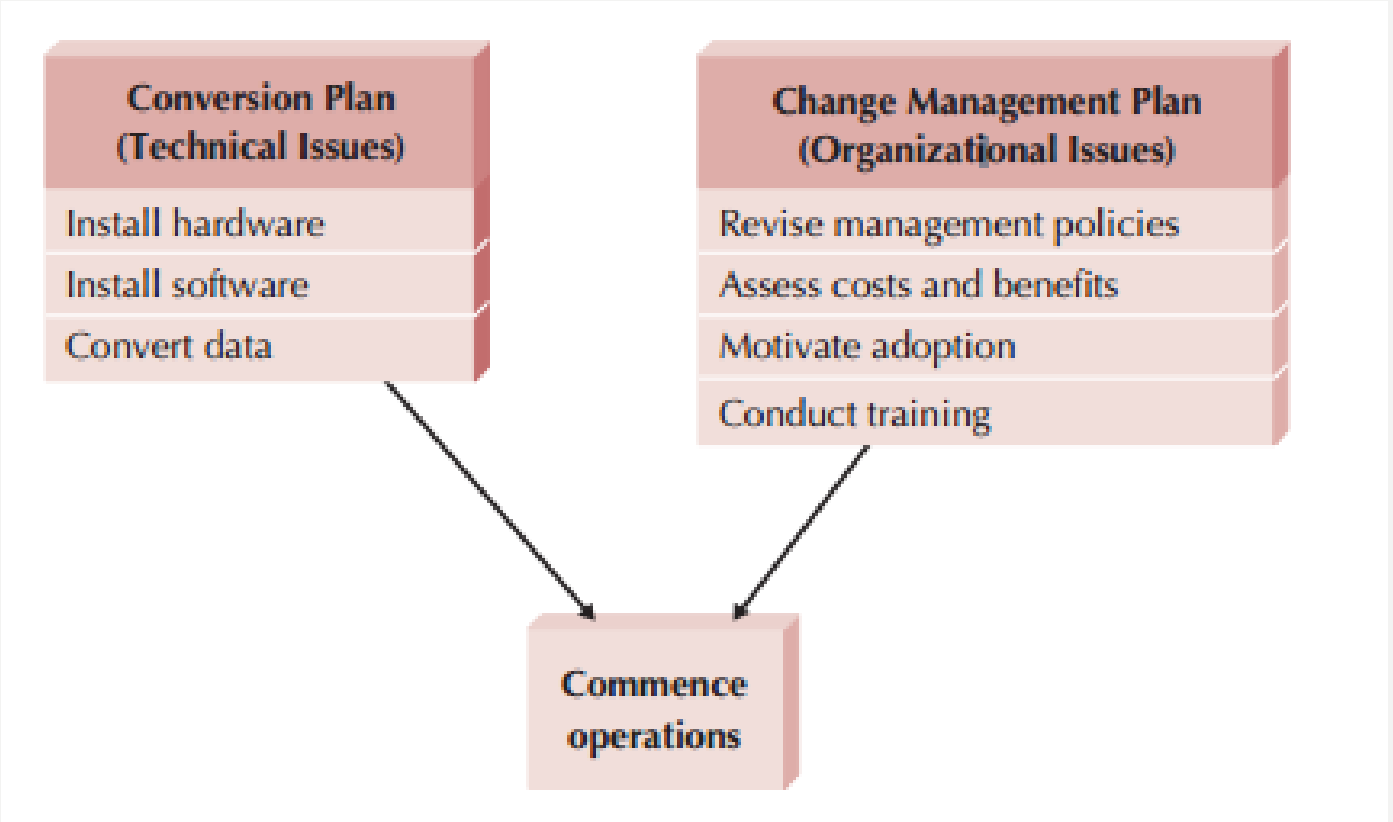
FAKULTAS ILMU KOMPUTER

MIGRATION PLAN

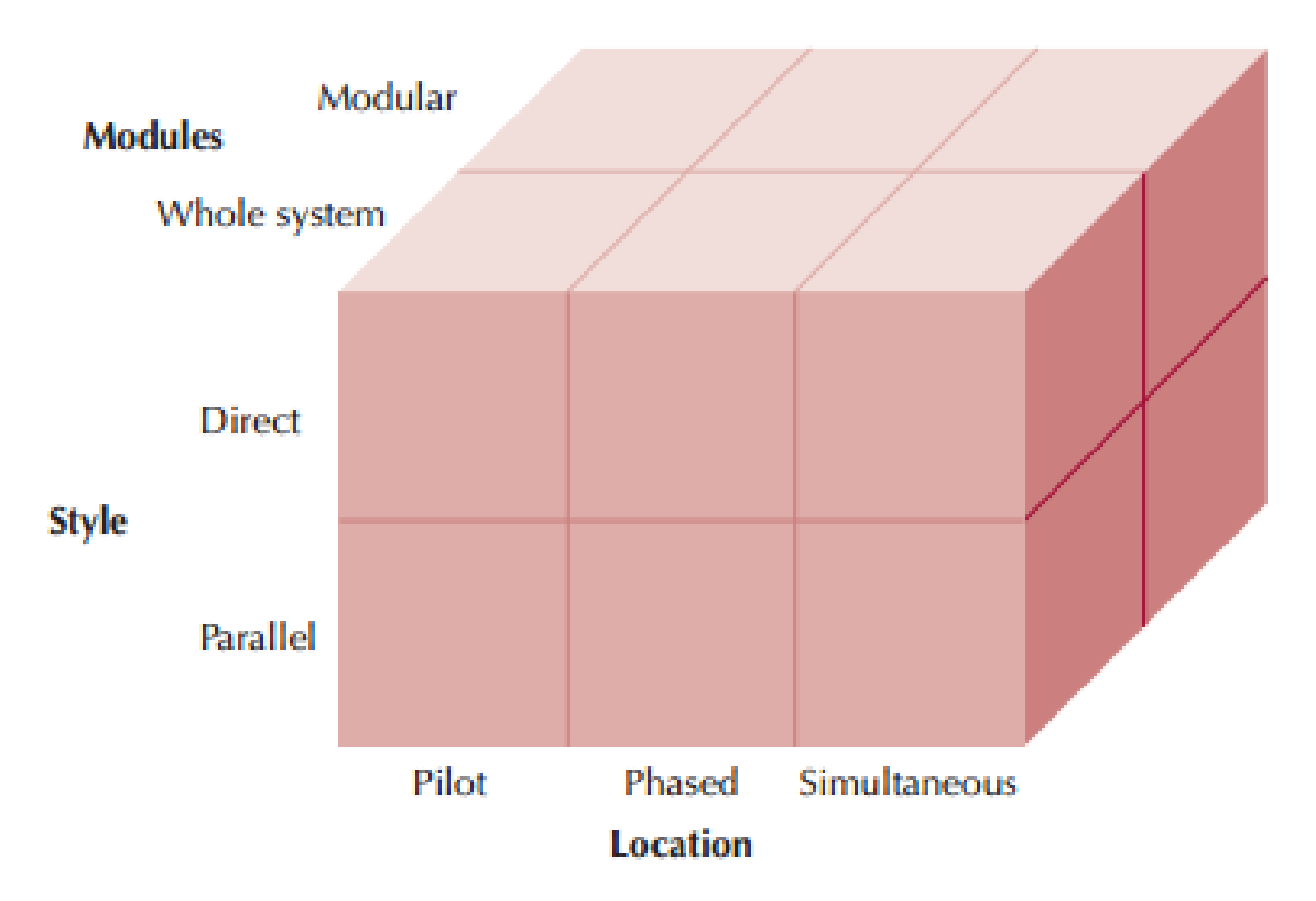
Migration Plan

- Users are moved from using the as-is business processes and computer programs to the to-be business processes and programs.
- The migration plan specifies what activities will be performed when and by whom and includes both technical aspects (such as installing hardware and software and converting data from the as-is system to the to-be system) and organizational aspects (such as training and motivating the users to embrace the new system).
- Conversion is the technical process by which a new system replaces an old system.

Migration Plan



Conversion Cube



Conversion Style (1)

- **Direct Conversion**
  - **sometimes called cold turkey, big bang, or abrupt cutover**
  - **the new system instantly replaces the old system.**
  - the new system is turned on and the old system is immediately turned off.
  - this is the approach that we are likely to use when we upgrade commercial software (e.g., Microsoft Word) from one version to another; we simply begin using the new version and stop using the old version.
  - direct conversion is the simplest and most straightforward.
  - However, it is the most risky because any problems with the new system that have escaped detection during testing can seriously disrupt the organization.

Conversion Style (2)

- **Parallel Conversion**
  - **the new system is operated side by side with the old system; both systems are used simultaneously.**
    - For example, if a new accounting system is installed, the organization enters data into both the old system and the new system and then carefully compares the output from both systems to ensure that the new system is performing correctly.
    - After some time period (often one to two months) of parallel operation and intense comparison between the two systems, the old system is turned off and the organization continues using the new system.

Conversion Style (2)

- **Parallel Conversion**
  - This approach is more likely to catch any major bugs in the new system and prevent the organization from suffering major problems.
  - If problems are discovered in the new system, the system is simply turned off and fixed and then the conversion process starts again.
  - The problem with this approach is the added expense of operating two systems that perform the same function.

Conversion Location (1)

- **Pilot Conversion**
  - **one or more locations or units or work groups within a location are selected to be converted first as part of a pilot test.**
  - The locations participating in the pilot test are converted (using either direct or parallel conversion).
  - If the system passes the pilot test, then the system is installed at the remaining locations (again using either direct or parallel conversion).

Conversion Location (1)

- **Pilot Conversion**
  - Pilot conversion has the advantage of providing an additional level of testing before the system is widely deployed throughout the organization, so that any problems with the system affect only the pilot locations.
  - However, this type of conversion obviously requires more time before the system is installed at all organizational locations.
  - Also, it means that different organizational units are using different versions of the system and business processes, which can make it difficult for them to exchange data.

Conversion Location (2)

- **Phased Conversion**
  - **the system is installed sequentially at different locations.**
  - A first set of locations is converted, then a second set, then a third set, and so on, until all locations are converted.
  - Sometimes there is a deliberate delay between the different sets (at least between the first and the second), so that any problems with the system are detected before too much of the organization is affected.
  - In other cases, the sets are converted back to back so that as soon as those converting one location have finished, the project team moves to the next and continues the conversion.

Conversion Location (2)

- **Phased Conversion**
  - Phased conversion has the same advantages and disadvantages of pilot conversion. In addition, it means that fewer people are required to perform the actual conversion (and any associated user training) than if all locations were converted at once.

Conversion Location (3)

- **Simultaneous Conversion**
  - **means that all locations are converted at the same time.**
  - The new system is installed and made ready at all locations; at a preset time, all users begin using the new system.
  - Simultaneous conversion is often used with direct conversion, but it can also be used with parallel conversion.
  - Simultaneous conversion eliminates problems with having different organizational units using different systems and processes.
  - However, it also means that the organization must have sufficient staff to perform the conversion and train the users at all locations simultaneously

Conversion Module (1)

- **Whole-System Conversion**
  - **The entire system is installed at one time, is the most common.**
  - It is simple and the easiest to understand. However, if the system is large and/or extremely complex (e.g., an enterprise resource-planning system such as SAP or PeopleSoft), the whole system can prove too difficult for users to learn in one conversion step.

Conversion Module (2)

- **Modular Conversion**
  - When the modules within a system are separate and distinct, organizations sometimes choose to convert to the new system one module at a time—that is, using modular conversion.
  - Modular conversion requires special care in developing the system (and usually adds extra cost).
  - Each module either must be written to work with both the old and new systems or object wrappers must be used to encapsulate the old system from the new.
  - When modules are tightly integrated, this is very challenging and therefore is seldom done.

Conversion Module (2)

- **Modular Conversion**
  - However, when there is only a loose association between modules, module conversion is easier.
  - Modular conversion reduces the amount of training required to begin using the new system.
  - Users need training only in the new module being implemented. However, modular conversion does take longer and has more steps than does the whole-system process.

Selecting the Appropriate Conversion Strategy

	Conversion Style		Conversion Location			Conversion Modules	
Characteristic	Direct Conversion	Parallel Conversion	Pilot Conversion	Phased Conversion	Simultaneous Conversion	Whole-System Conversion	Modular Conversion
Risk	High	Low	Low	Medium	High	High	Medium
Cost	Low	High	Medium	Medium	High	Medium	High
Time	Short	Long	Medium	Long	Short	Short	Long