CS 123
Introduction to Software Engineering

01: Introduction to Software Engineering

DISCS

SY 2013-2014

Learning Objectives

- To discuss characteristics of software
- To understand software quality
- To explain why software engineering is important
- To understand most probable reasons for software development success and failure
- To introduce software engineering
- To distinguish classical and modern approach in software engineering

What is software?

- **Computer programs** and **associated documentation**

- **Software products** may be developed for a particular customer or may be developed for a general market
- **Software products** may be
  - Generic – developed to be sold to a range of different customers
  - Custom – developed for a single customer according to their specification

Product

Goods

- E.g. bread, cloth, car, book
- Characteristics:
  - Tangible (physical): touch, smell, see, heard, taste
  - Consumable
  - Discrete manufactured

Is software a good or a service?

Services

- E.g. entertainment, consultation, tourism, care, teaching, delivery
- Characteristics:
  - Intangible (non-physical)
  - Inseparable: customer should be present to enjoy the service
  - Not transferable
  - Continuum: always improved

Discussion

- What does the statement “this software is licensed, not sold…” in the EULA?
- What is the difference between selling and license?
- Can you use or reuse your own code after you sell it?

Discussion

What factors make you buy a software?

Factors Not to buy

- Available open source
- Available pirated version
- Can be developed by ourselves
- ….

Factors to buy

- Nice features: functionality
- Fast: performance
- Reuse: component
- Productivity: reduce cost
- Support
- Update

Discussion

- What factors can make your software sellable and unique
  - Necessity
  - Productivity
  - Security
  - Reliability

Software Nature (1)

- Intangible production cost very high → hidden cost
  - High cost of research
  - High cost of development
- Cheap cost of tangible product: CD, disk
- Construction is Intellectual Value Intensive (high content of 'brain')
- Non-consumable: can be used "forever" (as long as life time of software)
- Short life time: 1-5 years, max 5 years

Software Nature (2)

- What we sell is actually information (not the CD, not even the program itself)
- Solution is complex: requires unusual rigor
  - UNIX contains 4 million lines of code
  - Windows 2000 contains 10^8 lines of code
- Malleable: easily shaped ("soft") to do almost anything → difficult to plan, monitor & control
- Dynamic Environment: adapts to better hardware & software over time

Creating New Software

- New software can be created by
  - Developing new programs,
  - Configuring generic software systems (e.g. component), or
  - Reusing existing software

Example Build/Build Decision

- Estimate software cost to build Php 220,000 (about 11,000 LOC at cost Php 20/LOC)
- Vendor offers most part of functionality, opens its source (9500 LOC), excellent documentation at Licensing fee of Php 75,000 and Php 5000 updating fee/year up to 5 years. You need to add functionality about 500 LOC (use double cost due to enhancing) and need to modify about 1500 LOC (at triple cost because it does not come from scratch)
- Will you buy or build from scratch?

Example Build/Build Decision (cont.)

- Cost of build from scratch P220,000
- Cost of Buy from vendor:

PARTICULARS	AMOUNT
Purchase price	Php 75,000
Modification cost (1,500 LOC x Php 60/LOC)	Php 60,000
Added Functionality (500 LOC x Php 40/LOC)	Php 20,000
5 years updating fee (5 x Php 5,000)	Php 25,000
TOTAL	Php 210,000

- Very close (other factor other than cost may be used for decision)
  - Risk
  - Maintenance

Discussion

When should you Buy or Build software?

When to buy?

- Already exists
- Reuse
- No testing
- Cheaper(?)
- ….

When to build?

- Request from client
- For fun
- Accumulate libraries
- Cheaper (?)
- ….

Analogy

- Dog House
  - Lumber, nail, & basic tools
  - Leaking is okay
  - If dog is not happy, get a less demanding dog
- Family House
  - Suits the needs of the family and building code
  - Detailed plan
  - Teams of workers
- High-Rise Building
  - Satisfies tenant requirements
  - Extensive plan and blue print
  - Professional team

What type of software are you building?

Activities in Software Development

City Analogy

- Progressive Activities
  - Increase living standard
  - Increase quality of life
- Anti/Regressive Activities
  - Garbage collection
  - Maintain status quo

Software

- Progressive Activities
  - Generating new code
  - Change existing code
- Anti/Regressive Activities
  - Write documentation
  - Improve code structure
  - Maintain comments

Activities in Software Development (contd)

- Neglecting Anti/Regressive Activities may not be harmful in the short term but will surely be in the long term
- Look for proper balance between both activities

Software Engineering ≠ Programming

- Software programming
  - Single developer
  - “Toy” applications
  - Short lifespan
  - Single or few stakeholders
    - Architect = Developer = Manager = Tester = Customer = User
  - One-of-a-kind systems
  - Built from scratch
  - Minimal maintenance

Software Engineering ≠ Programming

- Software engineering
  - Teams of developers with multiple roles
  - Complex systems
  - Indefinite lifespan
  - Numerous stakeholders
    - Architect ≠ Developer ≠ Manager ≠ Tester ≠ Customer ≠ User
  - System families
  - Reuse to amortize costs
  - Maintenance accounts for over 60% of overall development costs

Software Quality

1) Deliver on time
2) Within budget
3) Functional: does what it is supposed to do
4) High performance (efficient in speed, memory, and space)
5) Usable: User friendly
6) Understandable: well documented
7) Flexible: to be utilized as users need
8) Maintainable: evolves to meet changing needs
9) Reliable (dependable): no crash, works when it's needed, error free
10) Portable (adaptable): compatible with other systems or new environment

Discussion

- What are your personal goals when you develop a piece of software? Why? Do you need to re-examine these?

Software Engineering

- A discipline which is concerned with all aspects of software production
- Scope
  - study of software process, development principles, techniques, and models
- Goal
  - production of quality software that is delivered on time, within budget, satisfying customers' requirements, and users' needs

Software Engineering vs. Computer Science

Computer Science	Software Engineering
is concerned with	
• Theory • Fundamentals Algorithms, data structures, complexity theory, numerical methods	• The practicalities of developing • Delivering useful software SE deals with practical problems in complex software products

Computer science theories are currently insufficient to act as a complete underpinning for software engineering, BUT it is a foundation for practical aspects of software engineering

What software engineering is and is not..

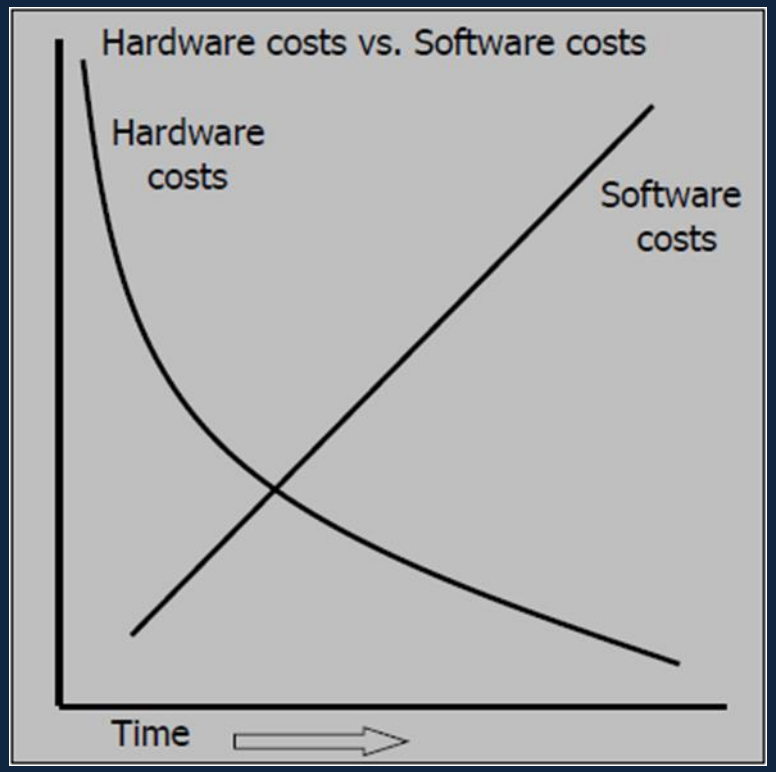
- Software engineering is concerned with “engineering” software systems, that is, building and modifying software systems:
  - on time,
  - within budget,
  - meeting quality and performance standards,
  - delivering the features desired/expected by the customer.
- Software engineering is not…
  - Just building small or new systems.
  - Hacking or debugging until it works.
  - Easy, simple, boring or even pointless!

What is Software Engineering?

- Software Engineering is a set of methods, tools, documents, practices, standards and procedures applied to definition, development and maintenance of computer software (General Electric, 1986)

Software development costs

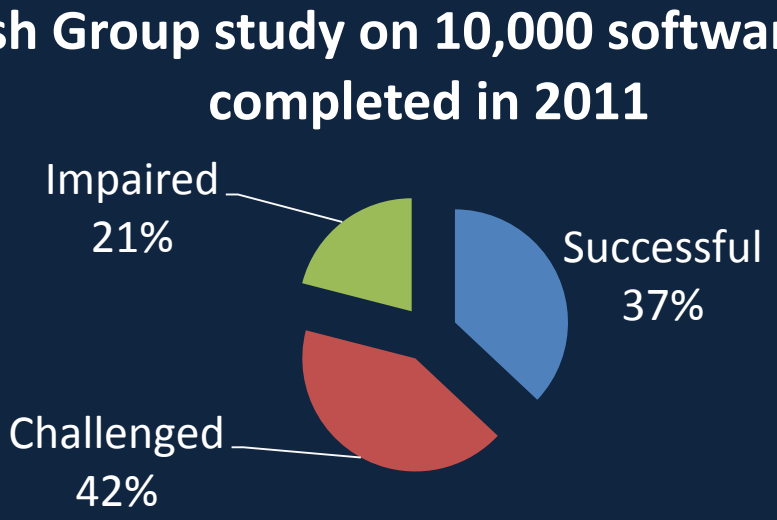
What can you infer from the graph?



Software costs are increasing as hardware costs continue to decline.

- Hardware technology had made great advances
- Simple and well understood tasks are encoded in hardware
- Least understood tasks are encoded in software
- Demands of software are growing
- Size of the software applications are also increasing
- Hence “the software crisis”

What can you infer from this chart?



- Successful projects (16.2%)
  - Delivered fully functional on time and on budget
- Challenged (52.7%)
  - Delivered with less functionality, over-budget, and late
- Impaired (31.1%)
  - Cancelled during development

Software Myths

- **Management myths**
  - Standards and procedures for building software
  - Add more programmers if behind the schedule
- **Customer myths**
  - A general description of objectives is enough to start coding
  - Requirements may change as the software is flexible
- **Practitioner myths**
  - Task is accomplished when the program works
  - Quality is assessed when the program is running
  - Working program is the only project deliverable

Software Failures

Failures resulting from software errors have varied consequences ranging from minor inconveniences to catastrophic loss of life & property:

- **Therac-25 (1985-1987)**: six people were overexposed during treatments for cancer
- **Taurus (1993)**: the planned automatic transaction settlement system for London Stock Exchange was cancelled after five years of development
- **Ariane 5 (1996)**: rocket exploded soon after its launch due to error conversion (16 floating point into 16-bit integer)
- **The Mars Climate Orbiter** assumed to be lost by NASA officials (1999): different measurement systems (Imperial and metric)

However…

Important progress:

- Ability to produce more complex software has increased
- New technologies have led to new SE approaches
- A better understanding of the activities involved in software development
- Effective methods to specify, design and implement software have been developed
- Many aspects have been made more systematic
  - Methods/methodologies/techniques
  - Languages
  - Tools
  - Processes

Economic Aspects

Coding method CM_{new} is 10% faster than currently used method CM_{old}. Should it be used?

- Common sense answer
  - Of course!
- Software Engineering answer
  - Consider the cost of training
  - Consider the impact of introducing a new technology
  - Consider the effect of CM_{new} on maintenance

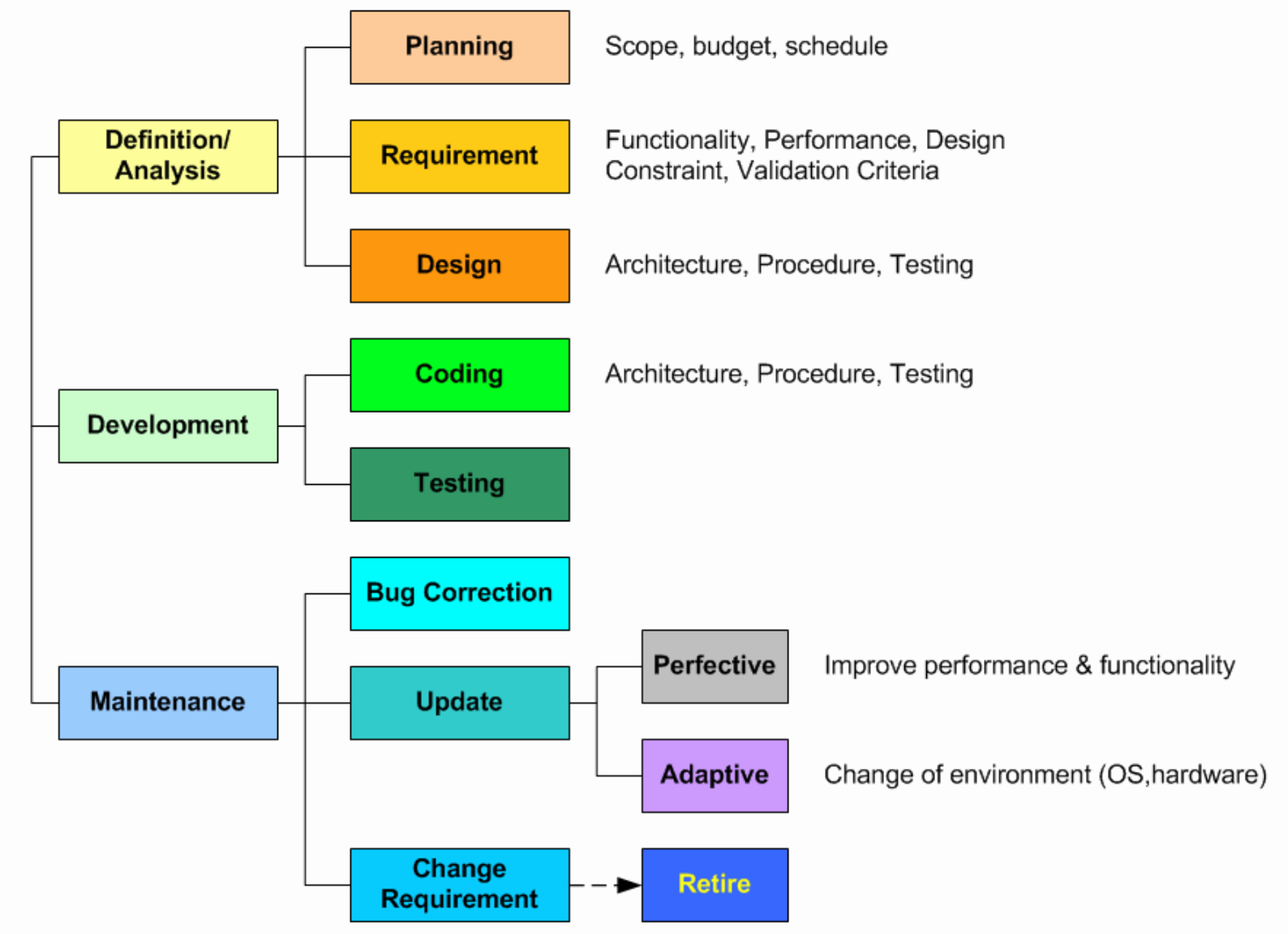
Economic and Management Aspects

- Software production =
  development + maintenance (evolution)
- Maintenance costs > 60% of all development costs
  - 20% corrective (bug fixing)
  - 30% adaptive (change of environment)
  - 50% perfective (improve performance & functionality)
- Quicker development is not always preferable
  - higher up-front costs may defray downstream costs
  - poorly designed/implemented software is a critical cost factor

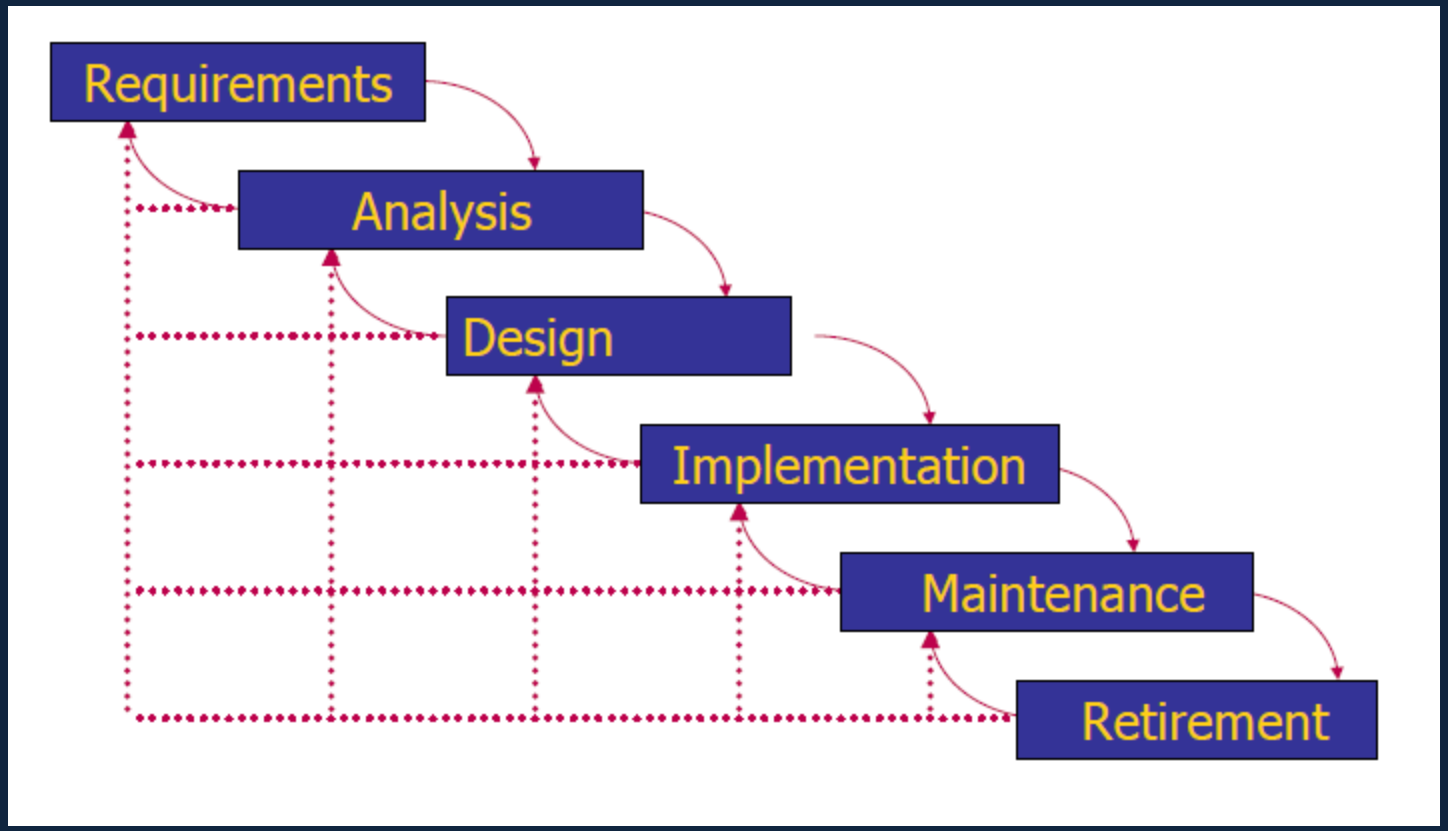
Software Development Life cycle

- Life-cycle model
  - The steps (phases) to follow when building software
  - A theoretical description of what should be done
- Life cycle
  - The actual steps performed on a specific product

Classical Life Cycle Model (1970)



Modern Waterfall Life-Cycle Model



Typical Phases

- Requirements phase
  - Explore the concept
  - Elicit the client's requirements
- Analysis (specification) phase
  - Analyze the client's requirements
  - Draw up the specification document
  - Draw up the software project management plan
  - "What the product is supposed to do"

Typical Phases (contd.)

- Design phase
  - Architectural design, followed by
  - Detailed design
  - “How the product does it”
- Implementation phase
  - Coding
  - Unit testing
  - Integration
  - Acceptance testing

Typical Phases (contd.)

- Postdelivery maintenance
  - Corrective maintenance
  - Perfective maintenance
  - Adaptive maintenance
- Retirement

Classical and Modern Views of Maintenance

- Classical maintenance
  - Development-then-maintenance model
- This is a temporal definition
  - Classification as development or maintenance depends on **when** an activity is performed

Classical Maintenance Definition — Consequence 1

- A fault is detected and corrected one day after the software product was installed
  - Classical maintenance
- The identical fault is detected and corrected one day before installation
  - Classical development

Classical Maintenance Definition — Consequence 2

- A software product has been installed
- The client wants its functionality to be increased
  - Classical (perfective) maintenance
- The client wants the identical change to be made just before installation (“moving target problem”)
  - Classical development

Classical Maintenance Definition

- The reason for these and similar unexpected consequences
  - Classically, maintenance is defined in terms of the time at which the activity is performed
- Another problem:
  - Development (building software from scratch) is rare today
  - Reuse is widespread

Modern Maintenance Definition

- In 1995, the International Standards Organization and International Electrotechnical Commission defined maintenance *operationally*
- Maintenance is nowadays defined as
  - The process that occurs when a software artifact is modified because of a problem or because of a need for improvement or adaptation

Modern Maintenance Definition (contd.)

- In terms of the ISO/IEC definition
  - Maintenance occurs whenever software is modified
  - Regardless of whether this takes place before or after installation of the software product
- The ISO/IEC definition has also been adopted by IEEE and EIA

Modern Maintenance Terminology

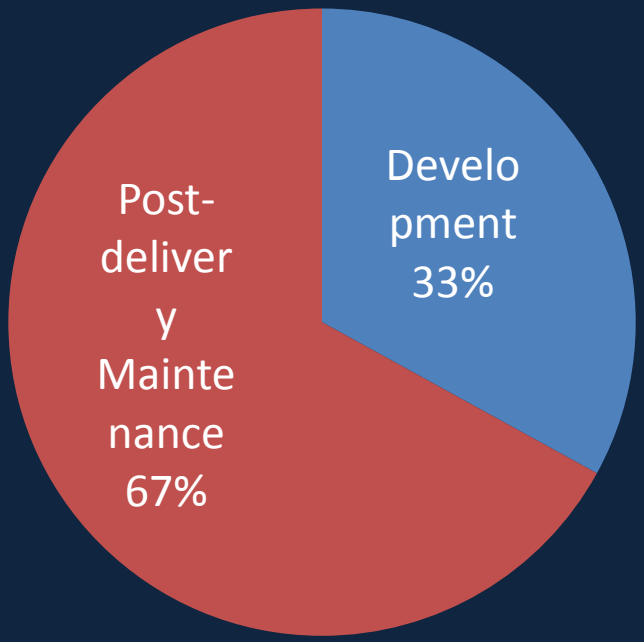
- Post-delivery maintenance
  - Changes after delivery and installation [IEEE 1990]
- Modern maintenance (or just maintenance)
  - Corrective, perfective, or adaptive maintenance performed at any time [ISO/IEC 1995, IEEE/EIA 1998]

The Importance of Post-delivery Maintenance

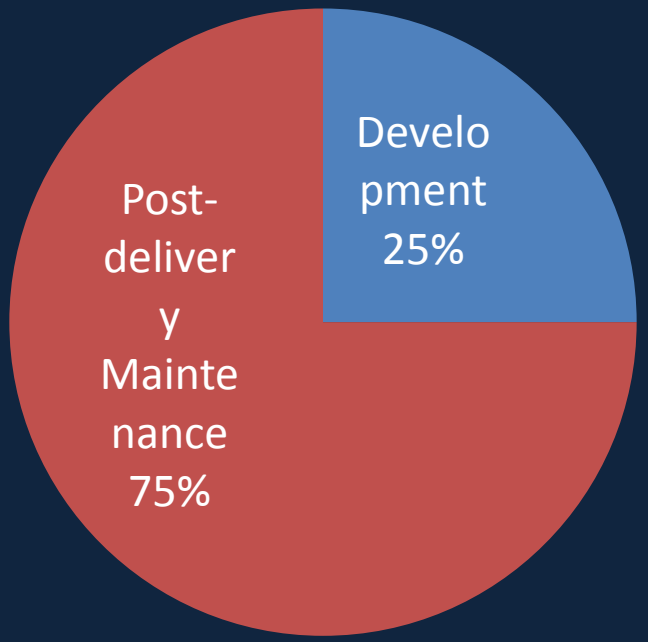
- Bad software is discarded
- Good software is maintained for 10, 20 years or more
- Software is a model of reality, which is constantly changing

Time (= Cost) of Post-delivery Maintenance

Between 1976 and 1981

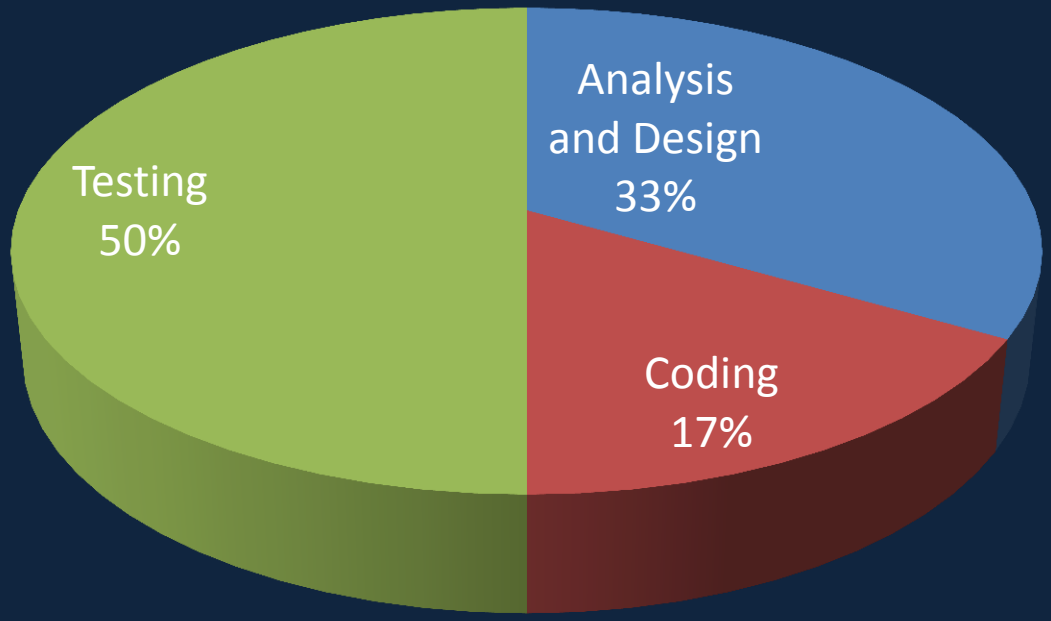


Between 1976 and 1981



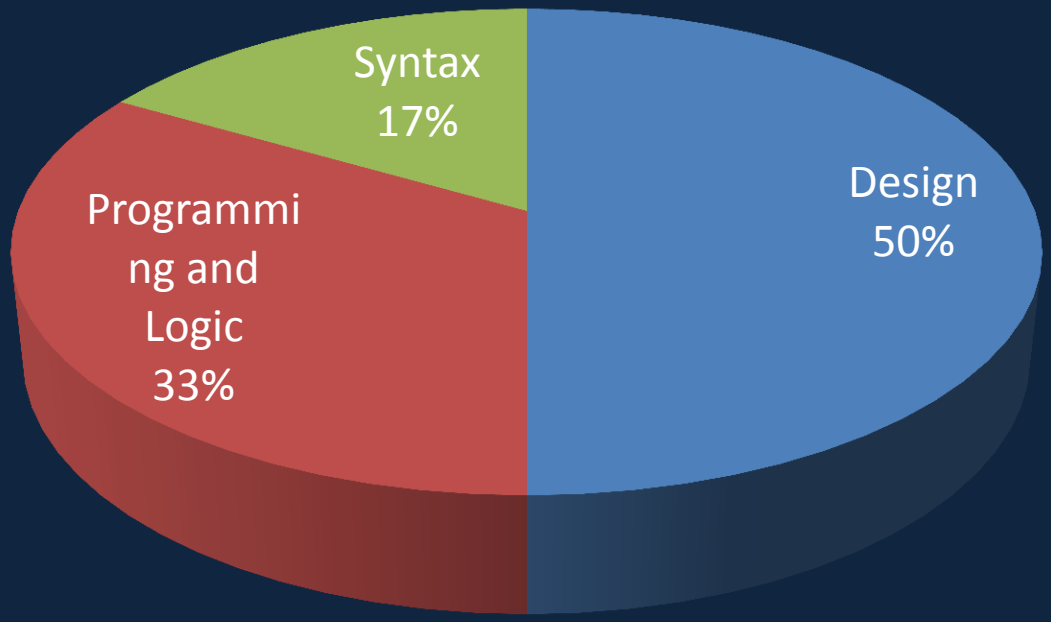
Relative Cost

- Relative costs of the stages of software development
- Analysis and Design: 1/3
- Coding: 1/6
- Testing: 1/2



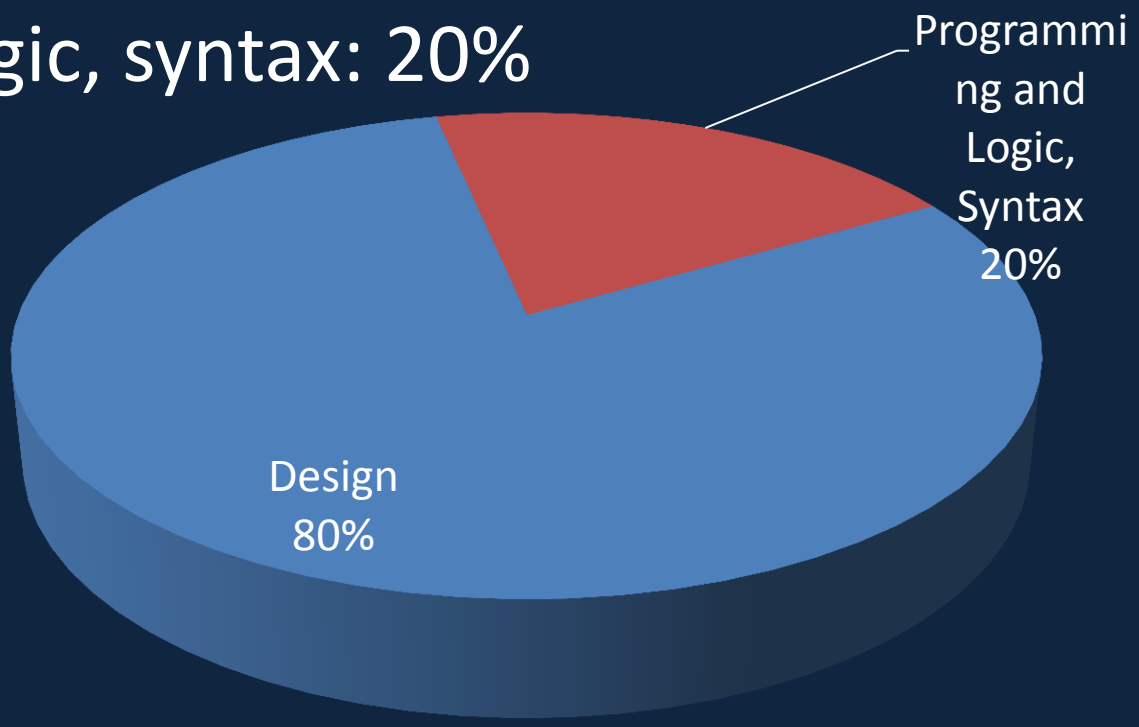
Relative Errors

- Relative number of errors made during the stages of software development
- Design: 1/2
- Programming and Logic: 1/3
- Syntax: 1/6



Relative Cost of Fixing

- Relative cost of fixing different types of software error
- Design: 80%
- Programming logic, syntax: 20%



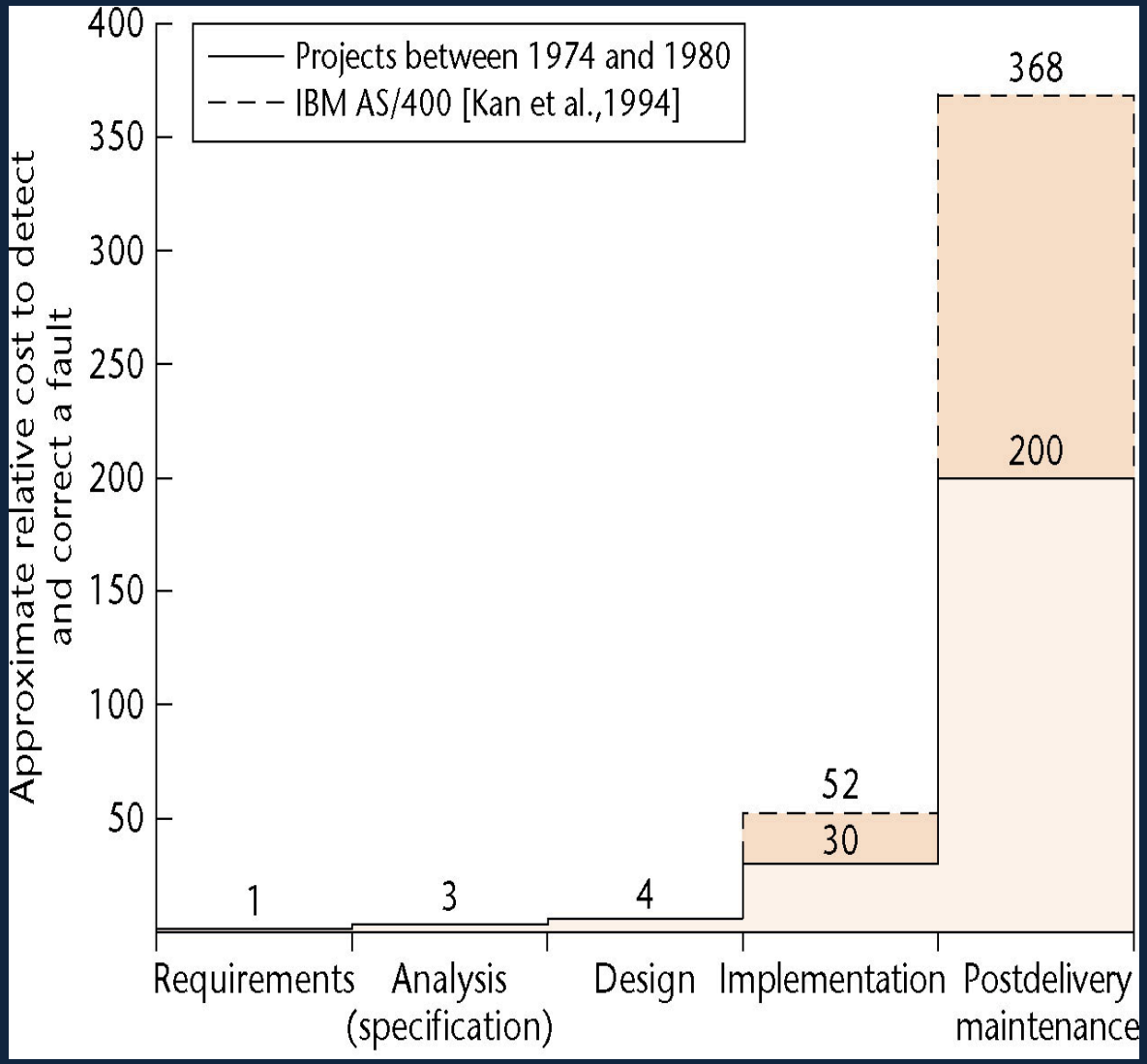
The Cost of Development Phases

- Surprisingly, the costs of the development phases have hardly changed over time

	Various Projects between 1976 and 1981	132 More Recent Hewlett-Packard Projects
Requirements and analysis (specification) phases	21%	18%
Design phase	18	19
Implementation phase		
Coding (including unit testing)	36	34
Integration	24	29

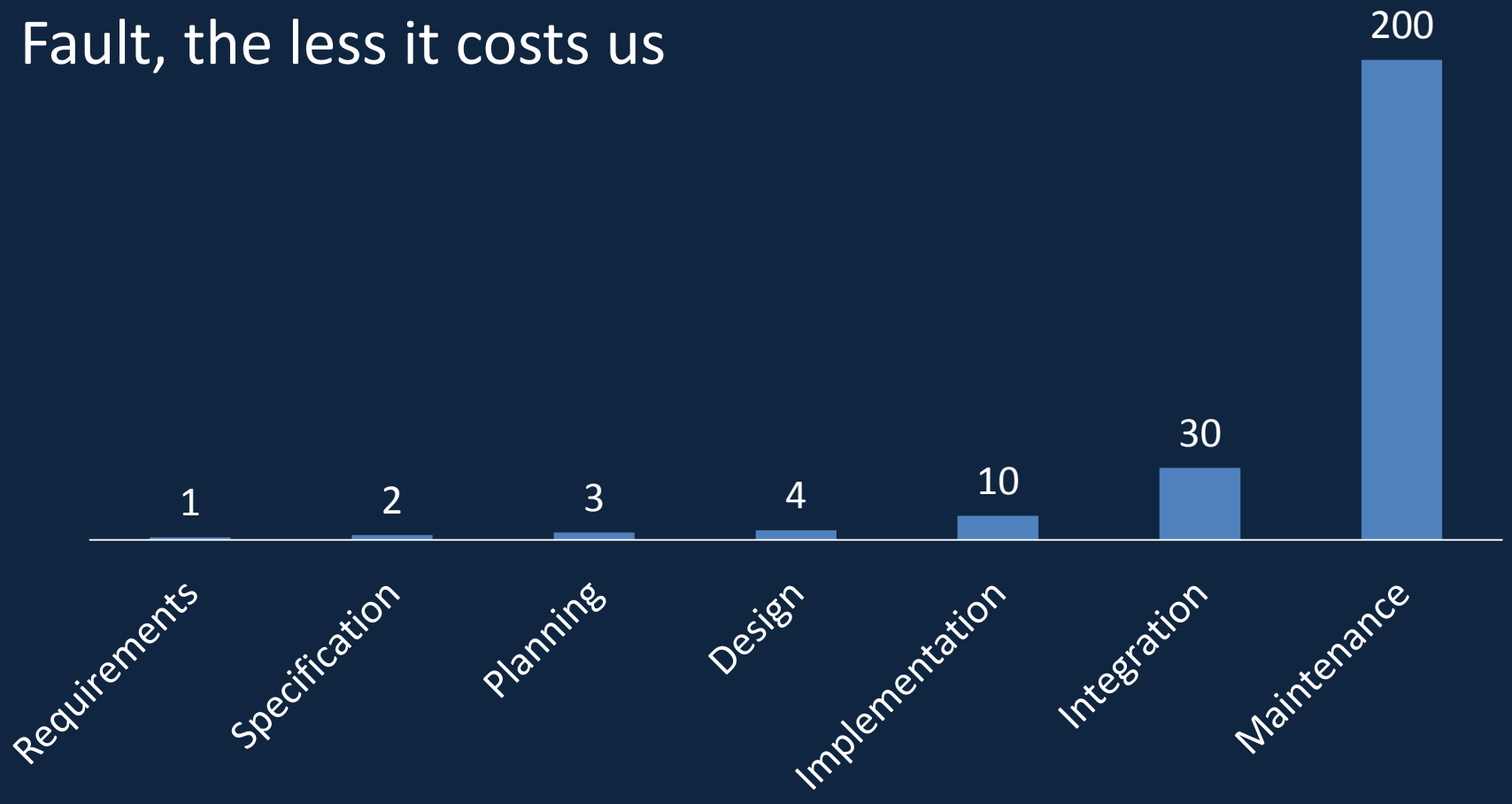
Requirements, Analysis, and Design Aspects

- The cost of detecting and correcting a fault at each phase



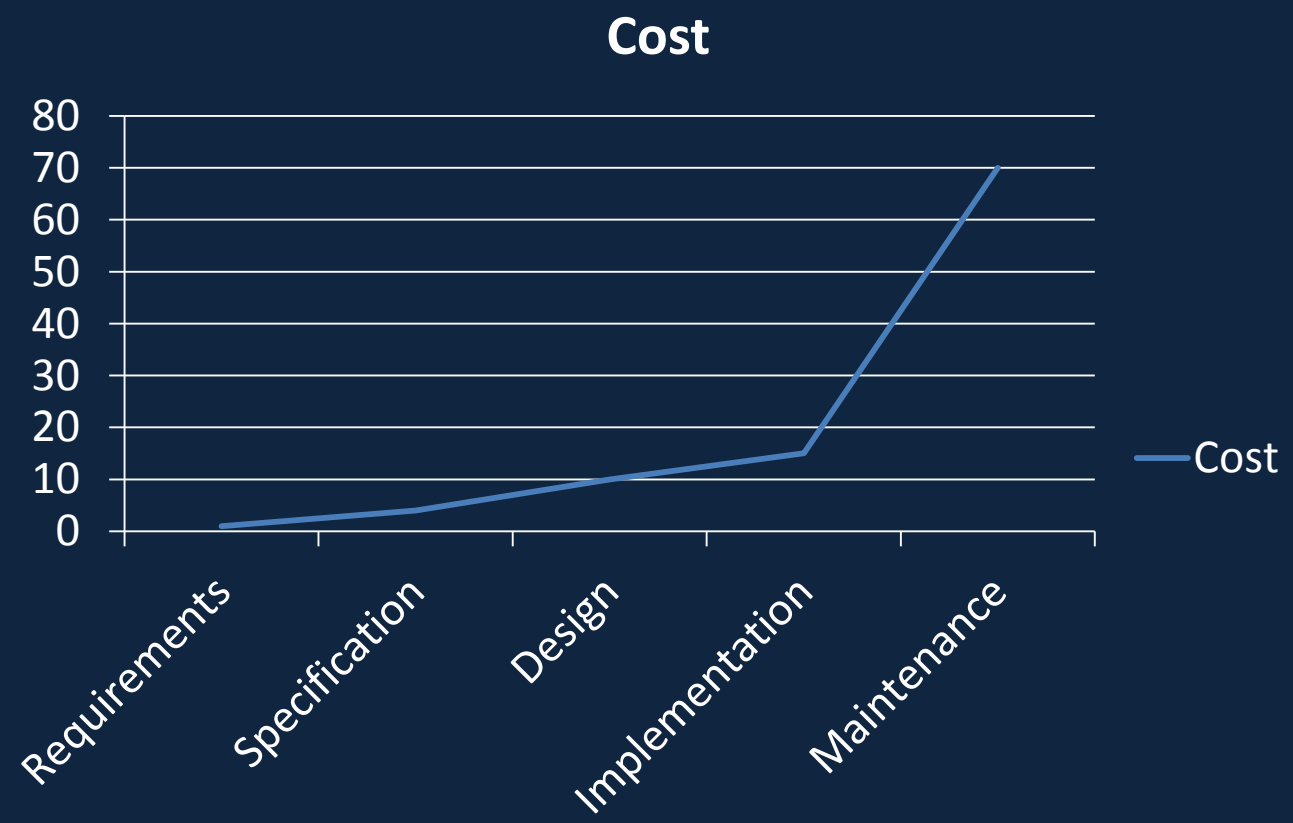
Relative Costs of Fixing Software Faults

The earlier we detect and correct a Fault, the less it costs us



Relative costs to fix errors

Notice that planning, testing, and integration Phases were removed



Cost to fix an error increases as it is found later and later in the software lifecycle

Discussion Example

- You are in charge of a software development.
- The cost of developing the software is estimated to be P 400,000. Approximately, how much additional money would be needed for post delivery maintenance of the software?

Discussion Answer

Answer: Php 400,000 = 40%

Additional 60% = Php 600,000 may be needed for post-delivery maintenance

Thus the total cost could reach Php 1 million

If maintenance is 75%, then

Php 400,000 = 25%

Additional Php 1.2 M is needed for maintenance

Total cost is Php 1.6 M

Requirement, Analysis, and Design Aspects (contd.)

- To correct a fault early in the life cycle
  - Usually just a document needs to be changed
- To correct a fault late in the life cycle
  - Change the code and the documentation
  - Test the change itself
  - Perform regression testing
  - Reinstall the product on the client's computer(s)
- Between 60% and 70% of all faults in large-scale products are requirements, analysis, and design faults

Discussion Example

6 months after delivery, a fault is detected in the software. Cost of fixing is Php 2,000,000. The cause of faults is an ambiguous sentence in the specification document. Approximately, how much would it cost to have corrected the fault during analysis phase?

Consequence of Relative Costs of Phases

- Comparison of Coding Techniques: CT_{new} = 10% faster than CT_{old}. Will you use?
- Reducing the coding cost by 10% yields at most a 0.85% reduction in total costs
  - Consider the expenses and disruption incurred
- Comparison of Maintenance Technique: MT_{new} = 10% cheaper than MT_{old}. Will you use?
  - Reducing post-delivery maintenance cost by 10% yields a 7.5% reduction in overall costs

Planning Phase

- It is vital to improve our requirements, analysis, and design techniques
  - To find faults as early as possible
  - To reduce the overall number of faults (and, hence, the overall cost)
- Planning activities are carried out throughout the life cycle
- There is no separate planning phase in modern life cycle

Testing Phase

- It is far too late to test after development and before delivery

Testing Activities of the Classical Paradigm

- Verification
  - Testing at the end of each phase (too late)
- Validation
  - Testing at the end of the project (far too late)

Testing

- Continual testing activities must be carried out throughout the life cycle
- This testing is the responsibility of
  - Every software professional, and
  - The software quality assurance group
- There is no separate testing phase

Documentation

- It is far too late to document after development and before delivery

Documentation Must Always be Current

- Key individuals may leave before the documentation is complete
- We cannot perform a phase without having the documentation of the previous phase
- We cannot test without documentation
- We cannot maintain without documentation

Documentation

- Documentation activities must be performed in parallel with all other development and maintenance activities
- There is no separate documentation phase

Summary

- Software is an information service
- Software nature is intangible, intellectual intensive, complex, malleable, exists in dynamic environment
- Software engineering is not only programming
- Software development quality
- What is software engineering

Summary

- Critical aspects of our day to day lives depend on software, yet software development lacks the rigor and discipline of mature engineering disciplines:
  - Too many projects get delayed, costs and schedules slip
- Software engineering seeks to bring discipline and rigor to the building and maintenance of software systems
- Why is it important to consider alternative models of the software development process?

Group Project 01: Initial Project Proposal

- Choose the members of your team. Each team should consist of 3-5 members
- List down the names of the members and 1 or more options for course project, described in general terms (and sorted by priority if more than 1)
- Some examples of initial project proposals:
  - online product listing and promo management
  - desktop application for scheduling DISCS classes and assigning teachers
  - mobile application for plate number recording
  - reverse bidding online site
- Software projects may be in any form (desktop, mobile, web, open source, etc.)

Group Project 02: Prototype Presentation

- Start to define your project
- Report 1 to 2 paragraphs of problem statement & simple feasibility study (you may also refer to the sample in the succeeding slide)
- Prepare for a prototype presentation of your project

Example of problem statement

FIGURE 2.14
An initial problem statement for The Appliance Store's inventory problem.

PROJECT PROPOSAL:	April 2, 1994
PREPARED BY:	W. S. Davis
THE CLIENT:	Tina Temple, The Appliance Store
THE PROBLEM:	A recent audit indicated that inventory is $50,000 too high, given sales.
OBJECTIVES:	1. To reduce store inventory by $50,000 by providing accurate, daily inventory status data to support reorder and sale item decisions. 2. To maintain the new inventory level (as a percentage of sales) into the future.
SCOPE:	The cost of this system will not exceed $10,000.
PRELIMINARY IDEAS:	A microcomputer-based inventory system.
KEY CONTACTS:	Tina Temple; Ted Temple.
FEASIBILITY STUDY:	In order to more fully investigate this project, a feasibility study lasting approximately two weeks and costing no more than $1,000 is suggested. The cost of the study is included in the scope. Following the feasibility study, the client will have the option of continuing with or terminating the project.

ACCEPTED BY:_____ DATE: _____

William Davis, Business Systems Analysis & Design

Discussion Answer

Relative costs of fixing software faults during specification is 2 compare to 200 during maintenance. Thus, if the fault was corrected during analysis phase, it would cost only Php 20,000

References

- CS 123 Lectures: Kardi Teknomo, PhD
- Schach. Object-Oriented and Classical Software Engineering, 8th Ed.