# Mbarara University of Science and Technology

## Faculty of Computing and Informatics

Fundamentals of Software Engineering

**By**

**Richard Ssembatya, PhD**

# Legacy Software

- Many programs still provide a valuable business benefit, even though they are one or even two decades old.
- Software systems need to be continually updated if they are to remain useful to their customers.
- These programs must be maintained and this creates problems because their design is often not amenable to change.

- Referencing a system as "legacy" often implies that the system is out of date or in need of replacement.

# Legacy Software

- Why must it change?
  - Software must be **adapted** to meet the needs of new computing environments or technology.
  - Software must be **enhanced** to implement new business requirements.
  - Software must be **extended to make it interoperable** with other more modern systems or databases.
  - Software must be **re-architected** to make it viable within a network environment.

# Software Evolution

- **Software evolution** is the term used in **software** engineering to refer to the process of developing **software** initially, then repeatedly updating it for various reasons.
- **Continuing Change**
  - A program that is used in a real-world environment changes or become less and less useful in that environment
- **Increasing Complexity**
  - As an evolving program changes, its structure becomes more complex unless active efforts are made to avoid this phenomenon

# Software Evolution

- Implications
  - Change is inevitable, plan for it
  - Don’t attempt to make very big changes in a single increment
  - Adding staff to a project will have a limited effect on project recovery

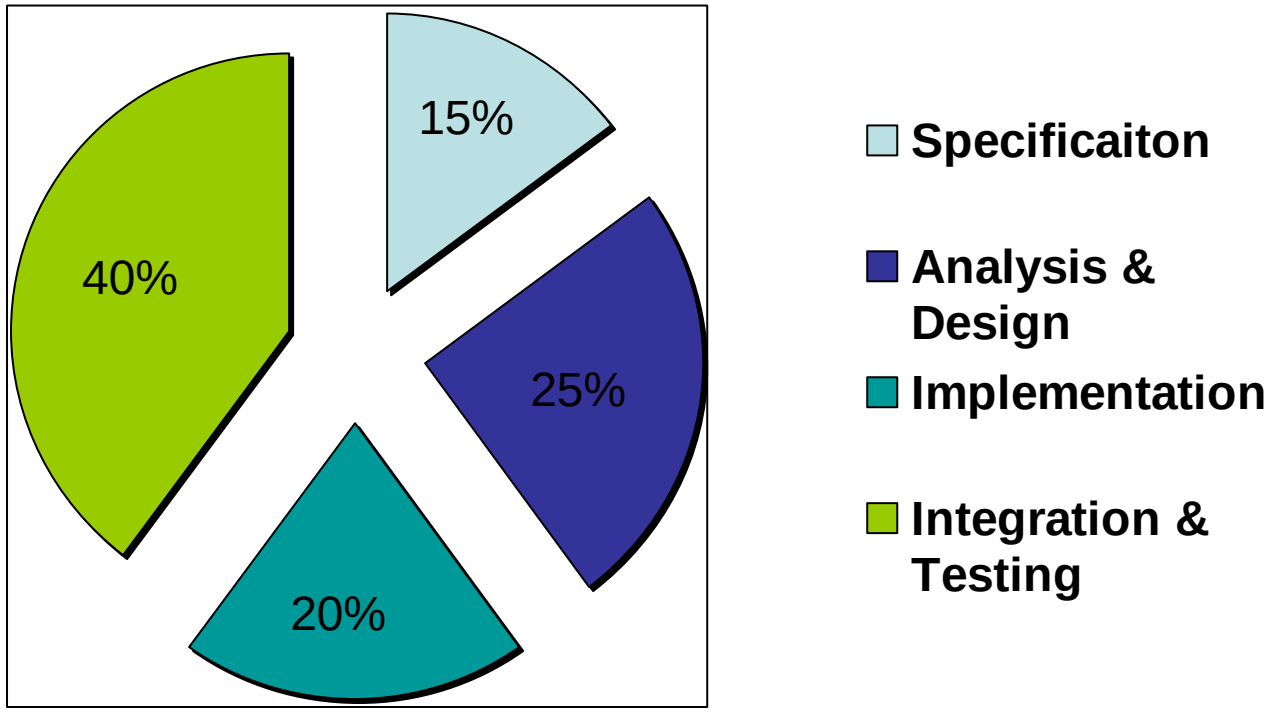
# Software Engineering Costs

- Distribution of costs in computing projects is changing - software rather than hardware is the largest single cost item.
- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs
- Software engineering is concerned with cost-effective software development

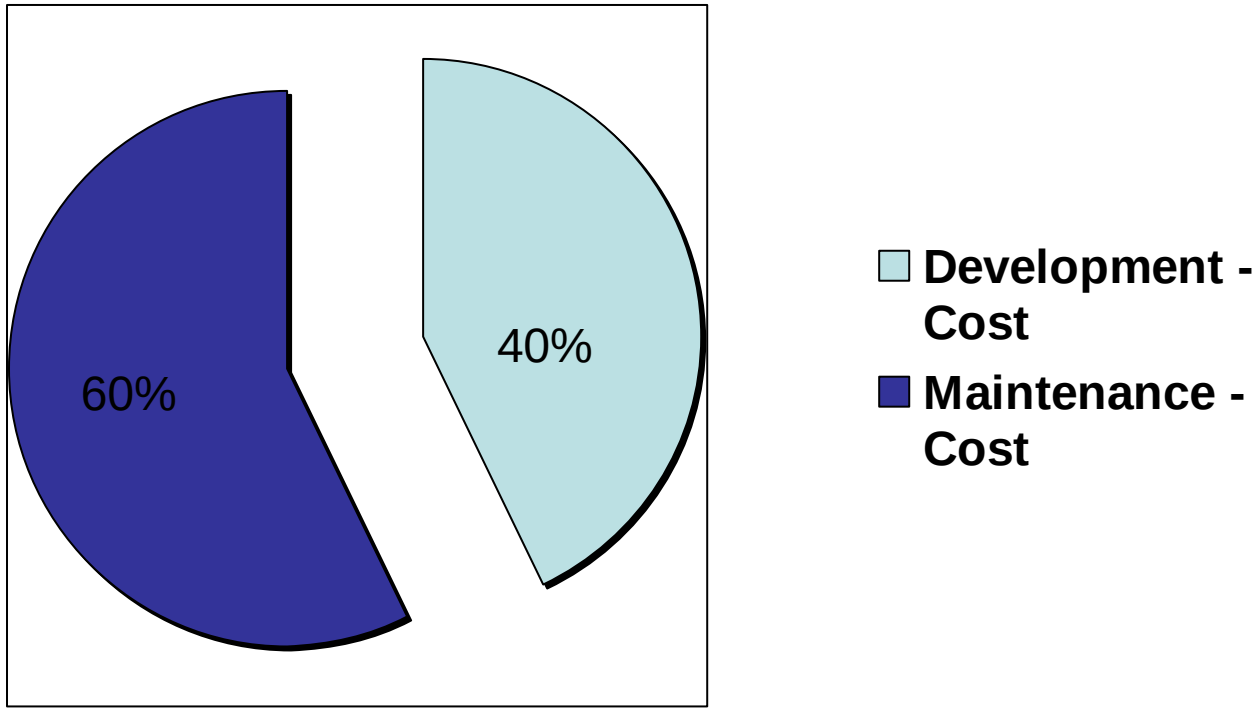
# Software Engineering Costs

- Distribution of costs depends on the development model.
- Costs vary depending on
  - the type of system being developed and;
  - the requirements of system attributes such as performance and system reliability.

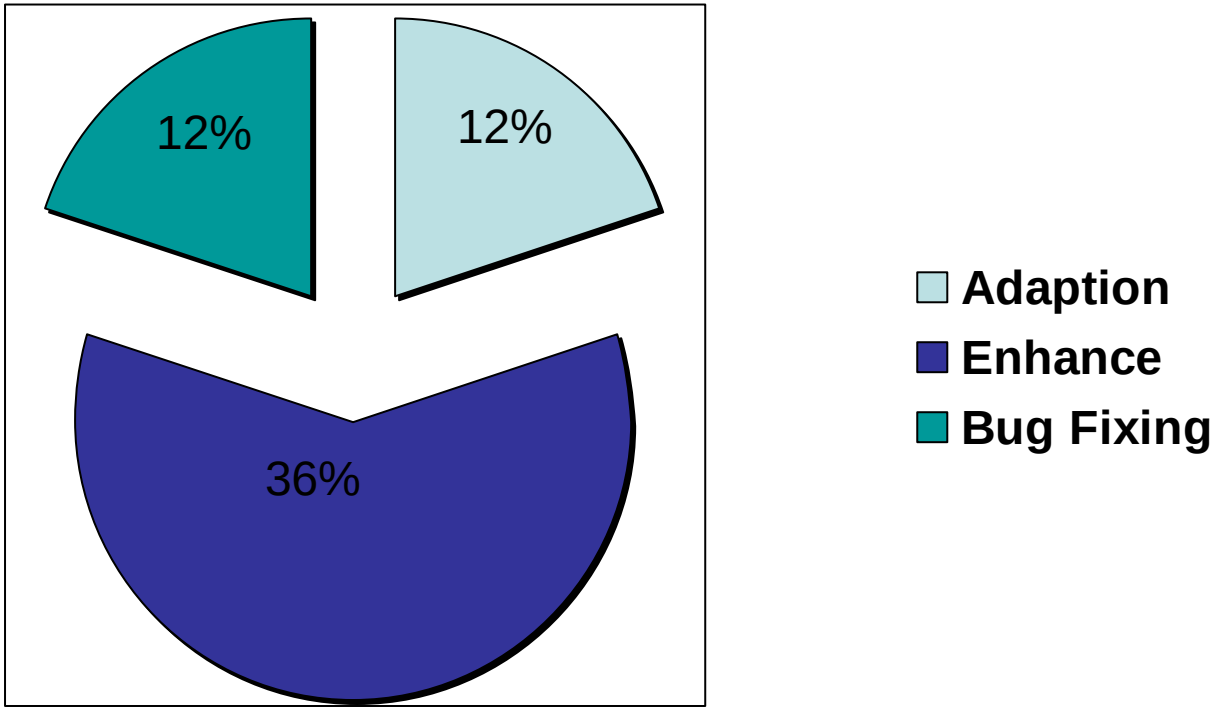
# Activity Cost Distribution



# Development - Maintenance Cost Ratios



# Maintenance Cost Ratios



# Software Engineering Challenges.

- Heterogeneity
  - Developing techniques for building software that can cope with heterogeneous platforms and execution environments;
- Delivery
  - Developing techniques that lead to faster delivery of software;
- Trust
  - Developing techniques that demonstrate that software can be trusted by its users.

# How a Project Starts?

- Every software project is precipitated by some business need
  - Need to correct a defect in an existing application.
  - Need to extend the functions and features of an existing application.
  - Need to create a new product or system.

# Software Development

- A set of activities that results in software products. Software development may include new development, modification, reuse, re-engineering, maintenance, or any other activities that result in software products.

# Software development

- There are 5 main parts to any software development effort
- Problem Analysis and Specification - What to do.
- Design - How to do it.
- Coding - Just do it.
- Verification - Does it do what it is supposed to do. Does it not do what it isn't supposed to.
- Maintenance - Change what it does (usually by repeating 1-4). Bug fixes and modifications.
- This is known as the software life cycle.

# Software development

- Whether or not you formally do each part of the software life cycle, it is important to consider each aspect.
- They are all important.
- Often problems can be found earlier, saving time.
- Proper emphasis on each stage makes later stages easier and faster.
- Each of these steps may be done by the same person, or by several different people.
- These steps may be done for an operating system, a program, a subsystem (part of a program), and an individual function.

# Problem analysis and specification

- Can't begin without knowing what needs to be done
- **What it does:**
- Clearly defines problem - what is and is not being solved
- Constitutes an agreement on what is to be done
- May discover problems

# Problem analysis and specification

- **What it doesn't do**
- Specify how to solve the problem.
- Example:
  No: Should be fast.
  Yes: Should run in less than 10 seconds.
  No: Should use quick sort

# Important parts of a problem specification

- A list of inputs
- A list of constants
- A list of outputs

# Software Design

- Once we have a clear idea of what needs to be done, we can work out how to do it. This is the design phase of the software development process.

# Design

- **A Software Design:**
- Clearly specifies how the problem will be solved
- Allows developers to determine what resources will be needed to solve the problem
- Hopefully solves all problems that could arise in the development of the software component.
- A design is not code and does not contain any code.

# Design

- In this phases it is decided how the system will operate, in terms of the hardware, software, and network infrastructure; the user interface, forms, and reports that will be used; and the specific programs, databases, and files that will be needed.

# Five Design Steps

1. **Design Strategy**: This clarifies whether the system will be developed by the company or outside the company.
2. **Architecture Design**: This describes the hardware, software, and network infrastructure that will be used.
3. **Database and File Specifications**: These documents define what and where the data will be stored.
4. **Program Design**: Defines what programs need to be written and what they will do.

# Custom Development

| PROS | CONS |
| --- | --- |
| Allows flexibility and creativity | Requires significant time and effort |
| Consistent with existing technology and standards | May exacerbate existing backlogs |
| Builds technical skills and functional knowledge in-house | May require missing skills |
| | Often costs more |
| | Often takes more calendar time |
| | Risk of project failure |

# Packaged Software

- Available for many common business needs
- Tested, proven; cost and time savings
- Rarely a perfect fit with business needs
- May allow for customization
  - Manipulation of system parameters
  - Changing way features work
  - Synchronizing with other application interfaces
- May require workarounds

# Systems Integration

- Building systems by combining packages, legacy systems, and custom pieces
- Integrating data is the key

# Outsourcing

- Hiring an external vendor, developer, or service provider
- May reduce costs or add value
- Risks include possibly
  - Losing confidential information
  - Losing control over future development
  - Losing learning opportunities

# Outsourcing Guidelines

- Keep the lines of communication open between you and your outsourcer.
- Define and stabilize requirements before signing a contract.
- View the outsourcing relationship as a partnership.
- Select the vendor, developer, or service provider carefully.
- Assign a person to manage the relationship.
- Don't outsource what you don't understand.
- Emphasize flexible requirements, long-term relationships, and short-term contracts.

# Code

- Once a design is complete, coding can begin. Given a good design, this should be very straightforward. All hard problems should have been worked out in the design stage. New hard problems should send the project (temporarily) back to the design stage.

# Code

- Good code should be
- Correct
- Readable and Understandable
- Modifiable
- Reusable

# Programs

**Programs should be:**

- Well structured
- Break programs into meaningful parts
- Strive for simplicity and clarity
- Well documented (commented))
- Good comments before each program and/or function
- Good comments before each important part of a program/function
- Use meaningful identifiers (function and variable names)
- Space things out
- Use blank lines between logical blocks

# Verification

- Does the problem do all and only what it is supposed to do?
- Each program/function should be tested against its requirements to see that it does what it is supposed to do.
- Programs should also be tested to make sure that they don't do what they are not supposed to do.
- Tests should include correct and incorrect inputs, as well as nonsense inputs.

# Software Process

- Lecture Four