

# Software Testing

## Introduction Lecture

# What is Software?

- Not just about computer programs
- It is also associated with documentation and configuration data that is needed to make these programs operate correctly

# Early Days of Software

- In the early days of computing, computer-based systems were developed using hardware-oriented management
- Project managers focused on hardware
- Project managers applied the controls, methods, and tools that we recognize as hardware engineering

# Early Days of Software

- Programming was viewed as an art form
- The programmer often learned his craft by trial and error
- The software world was undisciplined

# The Crisis in Software Engineering

- In the 1970's there were a number of problems with software:
  - Projects were running over-budget
  - Projects were running over-time
  - The Software products were of low quality
  - The Software products often did not meet their requirements
  - Projects were chaotic
  - Software maintenance was very difficult

# Software Engineering

- The actual term 'software engineering' was first proposed as far back as 1968 at a conference held to discuss what was then called the 'software crisis'.
- It was becoming clear by then that individual approaches to program development did not scale up to large and complex software systems. These were unreliable, cost more than expected, and were delivered late.

# Software Engineering

- To overcome these problems throughout 1970s and 1980s, a variety of new software engineering techniques and methods were developed
- These include structured programming, information hiding and object-oriented development.
- Tools and standard notations that were developed at that time are now extensively used

# Problems with Software

- These can be broken down into three elements of Errors, Faults and Failures



# Errors, Faults and Failures

1. Errors: these are mistakes made by software developers. They exist in the mind, and can result in one or more faults in the software.
2. Faults: these consist of incorrect material in the source code, and can be the product of one or more errors. Faults can lead to failures during program execution.
3. Failures: these are symptoms of a fault, and consist of incorrect, or out-of specification behaviour by the software. Faults may remain hidden until a certain set of conditions are met which reveal them as a failure in the software execution.

# Software Faults - Categories

- Algorithmic faults
  - Algorithmic faults are the ones that occurs when a unit of the software does not produce an output corresponding to the given input under the designated algorithm
- Syntax Faults
  - These occur when code is not in conformance to the programming language specification, (i.e. source code compiled a few years back with older versions of compilers may have syntax that does not conform to present syntax checking by compilers (because of standards conformity).

# Software Faults - Categories

- Documentation faults
  - Incomplete or incorrect documentation will lead to Documentation faults
- Stress or overload faults
  - Stress or Overload faults happens when data structures are filled past their specific capacity where as the system characteristics are designed to handle no more than a maximum load planned under the requirements
- Capacity and boundary faults
  - Capacity or Boundary faults occur when the system produces an unacceptable performance because the system activity may reach its specified limit due to overload
- Computation and precision faults
  - Computation and Precision faults occur when the calculated result using the chosen formula does not conform to the expected accuracy or precision

# Software Faults - Categories

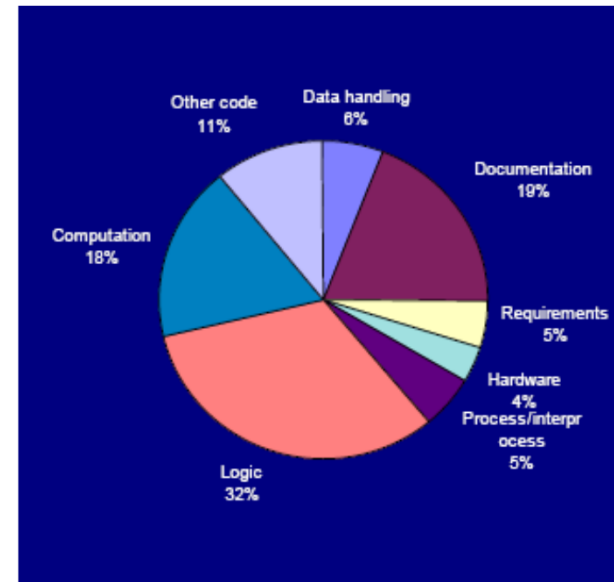
- Throughput or performance faults
  - This is when the developed system does not perform at the speed specified under the stipulated requirements
- Recovery faults
  - This happens when the system does not recover to the expected performance even after a fault is detected and corrected
- Timing or coordination faults
  - These are typical of real time systems when the programming and coding are not commensurate to meet the co-ordination of several concurrent processes or when the processes have to be executed in a carefully defined sequence

# Software Faults - Categories

- Standards and Procedure Faults
  - Standards and Procedure faults occur when a team member does not follow the standards deployed by the organization which will lead to the problem of other members having to understand the logic employed or to find the data description needed for solving a problem.

# Software Faults

- A study by Hewlett Packard on the frequency of occurrence of various fault types, found that 50% of the faults were either Algorithmic or Computation and Precision



# Software Failures

1. Failure causes a system crash and the recovery time is extensive; or failure causes a loss of function and data and there is no workaround
2. Failure causes a loss of function or data but there is manual workaround to temporarily accomplish the tasks
3. Failure causes a partial loss of function or data where user can accomplish most of the tasks with a small amount of workaround
4. Failure causes cosmetic and minor inconveniences where all the user tasks can still be accomplished

# Comparing Software with Hardware

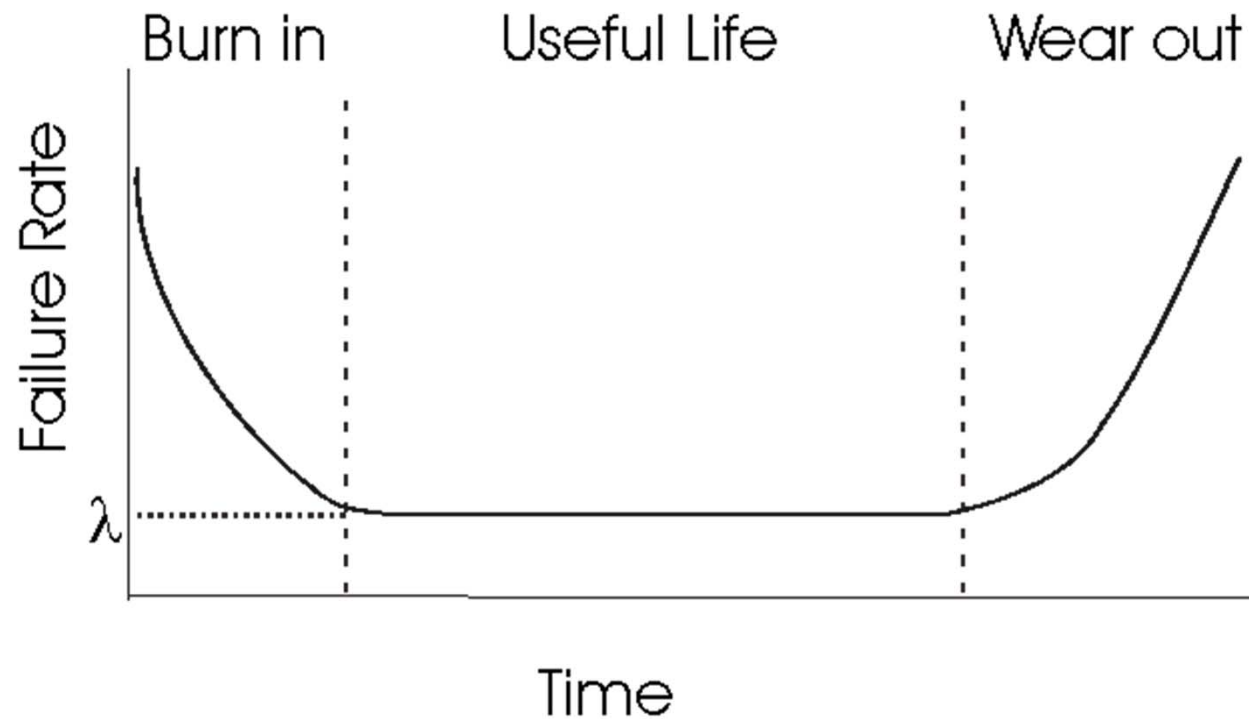
- To gain an understanding of SW and how to approach testing it, it is important to examine the characteristics of software that make it different from other things that human beings built.
- When hardware is built, the human creative process (analysis, design, construction, testing) is ultimately translated into a physical form.
- Software is a logical rather than a physical system element.
- Therefore, software has characteristics that differ considerably from those of hardware



# Difference between Software and Hardware

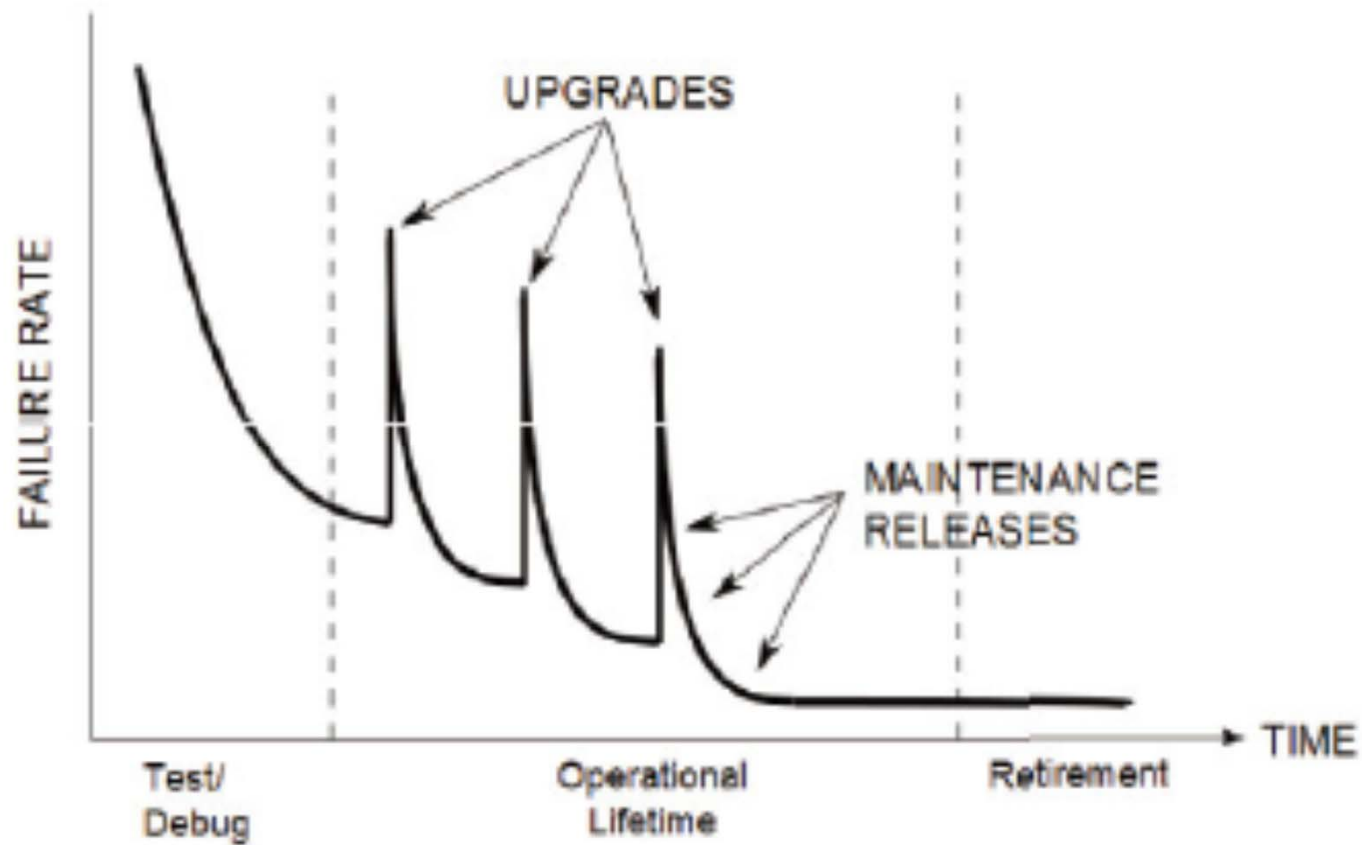
- Software is developed or engineered, it is not manufactured in the classical sense.
- Software does not wear out
- See failure curves for hardware and software
- Most software is custom-built, rather than being assembled from existing components

# Failure Curve for Hardware



Burn-in: Procedure used in spotting weak parts or circuits of an electronic device (such as a computer) by running it at full power in a hot and humid environment for extended periods (up to 30 days for critical systems). This practice is based on the experience that semiconductor devices often show their defects in the first few days or weeks of operation.

# Curve for Failure Rate over Product lifecycle



# Curve for Failure Rate over Product lifecycle

- Unlike hardware software does not physically wear out.
- It is subject to ongoing changes after release and is subject to changes in the external environment (such as an OS upgrade).
- These changes can introduce new faults or expose latent faults.

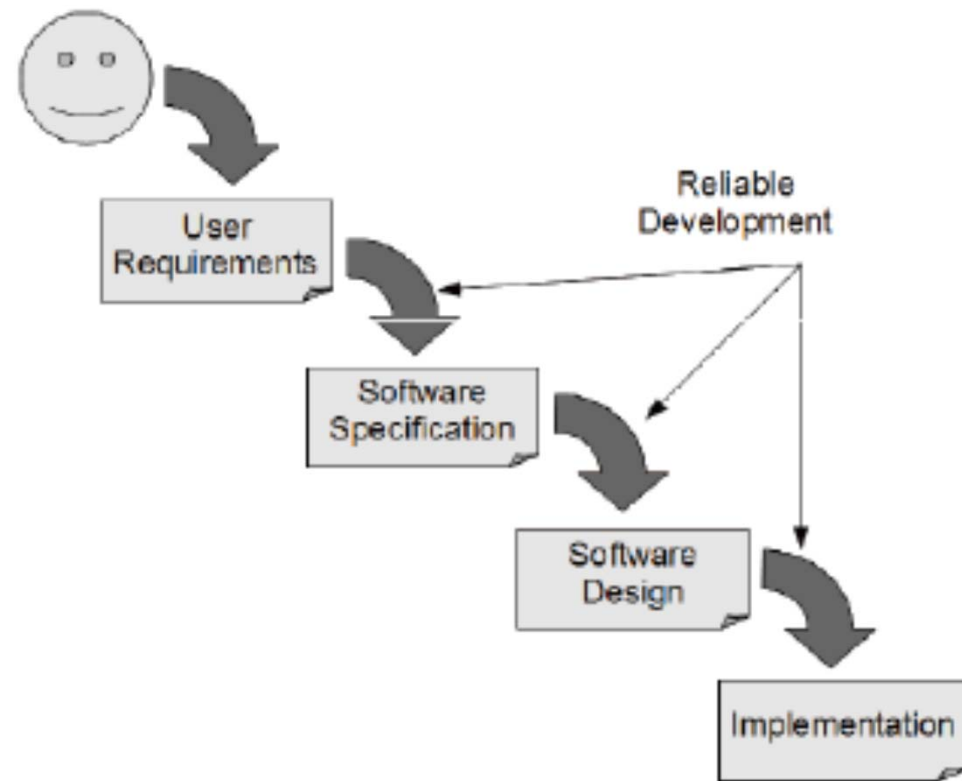
# Curve for Failure Rate over Product lifecycle

- Initially, the first version has a high failure rate. With debugging and testing, as these are found and corrected the failure rate decreases until it reaches an acceptable level.
- Upgrades introduce new faults in upgrades that are then fixed via maintenance releases
- Once the software is no longer supported the failure rate levels off until the product becomes obsolete.

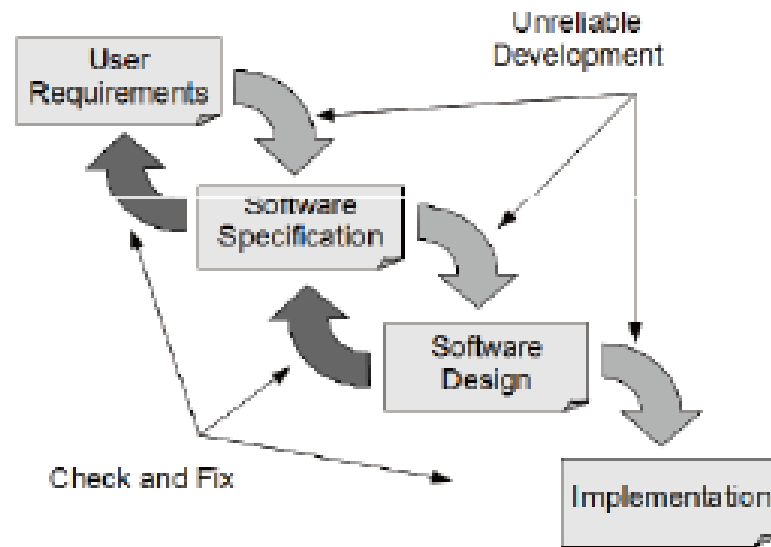
# Forward Engineering

- This is an approach to designing correct systems. It starts with the user and ends with the correct implementation. The end product matches the specification.

# Forward Engineering



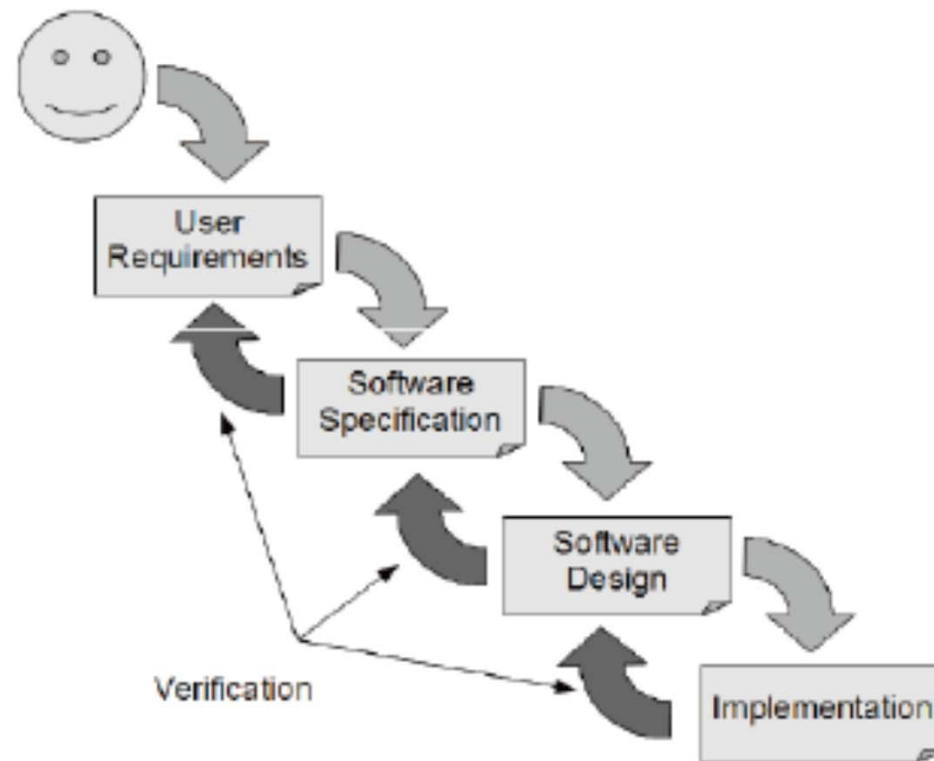
# Check and Fix



In practice, all development steps must be subject to a check to ensure it has been carried out properly and any mistakes encountered are fixed.

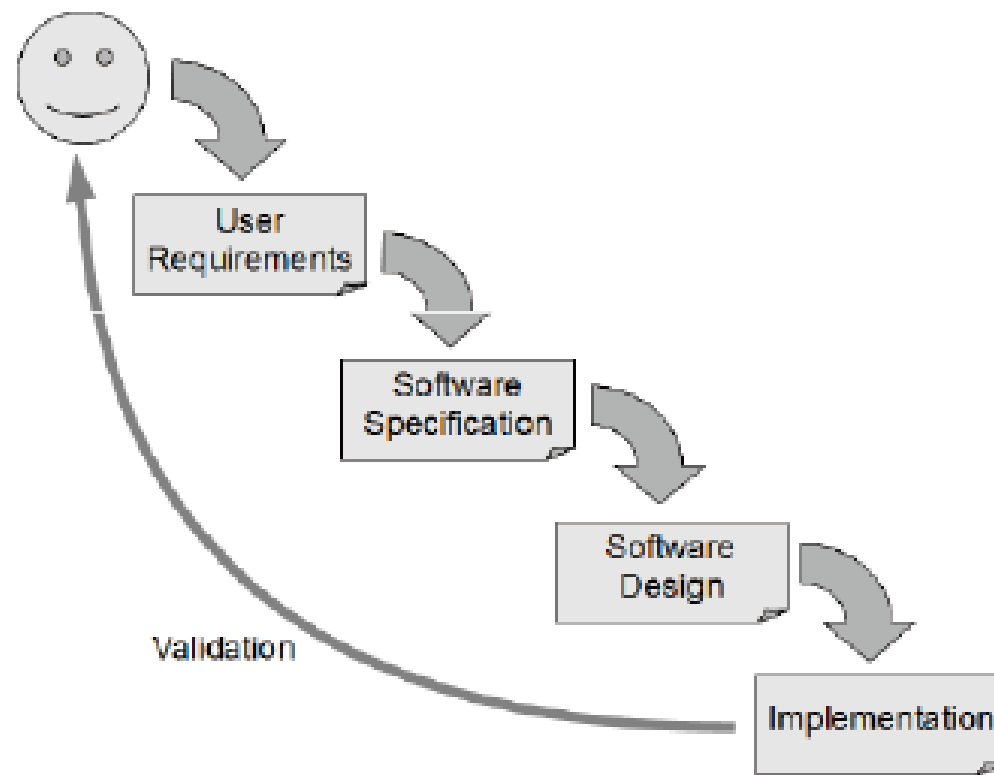


# Including verification in the development process



Check that each step meets its specifications

# Validation in the development process



Check that the software meets the users needs

# Specifications

- Specifications play a key role. Detailed specifications provide the correct behaviour of the software.
- They must describe normal and error behaviour.

# Testing in the development process

- Software has three key characteristics
  - User requirements that state the user's needs
  - A functional specification stating what the software must do
  - A number of modules that are integrated to form the final system
- These must be verified using the following four test activities

# Unit Testing

- An individual unit of software is tested to ensure that it works correctly. This may be a single component or a compound component.
- A component might be a method, a class, or subsystem. It could be a single GUI component (e.g. button) or a collection of them (e.g. a window).
- This makes use of the programming interface of the unit.

# Integration Testing

- Two or more units are tested to ensure that they interoperate correctly.
- This may use the programming interface or the System interface.
- Can be Top-Down, Bottom-up, or take an 'end-to-end user functionality' approach

# System Testing

- The entire software system is tested to make sure that it works correctly and that it meets/solves the user's needs/problem.
- This uses the system interface which may be a GUI, Network interface, web interface etc...

# Acceptance Testing

- The entire software system is tested to make sure that it meets the user's needs.
- Again, this uses the system interface.



# Theory of Testing

- The goal is to identify the ideal test – that is, the minimum test data required to ensure that the software works for all inputs.

# Exhaustive Testing

- This is generally not feasible as it would take too long or require too much memory space.
- A good test should have a high probability of finding faults, not duplicate another test, be independent of what it measures so no faults conceal one another, and test as much of the code as possible.

# Testing Types

- Black-box testing – generate input values that exercise the specification and compare the actual output with the expected output
- White-box testing – generate input values the exercise the implementation and compare the actual output with the expected output
- Fault insertion – insert faults into the code or data to measure the effectiveness of testing or ensure that the fault is detected and handled correctly

# Finishing Testing

- A budgetary point of view: when the time or budget allocated has expired
- An activity point of view: when the software has passed all of the planned tests
- A risk management point of view: when the predicted failure rate meets some quality criteria

# Criteria

- Usage-based criteria give priority to the most frequent sequences of program events.
- Risk of Release predicts the cost of future failures based on their chance of occurring.