SCHULICH School of Engineering


UNIVERSITY OF CALGARY

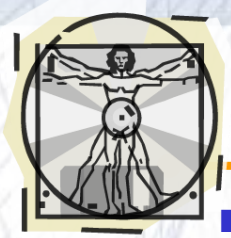
# SENG 637
# Dependability and Reliability of Software Systems

## Chapter 8: Software Quality & Reliability Engineering (SRE)

**Department of Electrical & Software Engineering, University of Calgary**

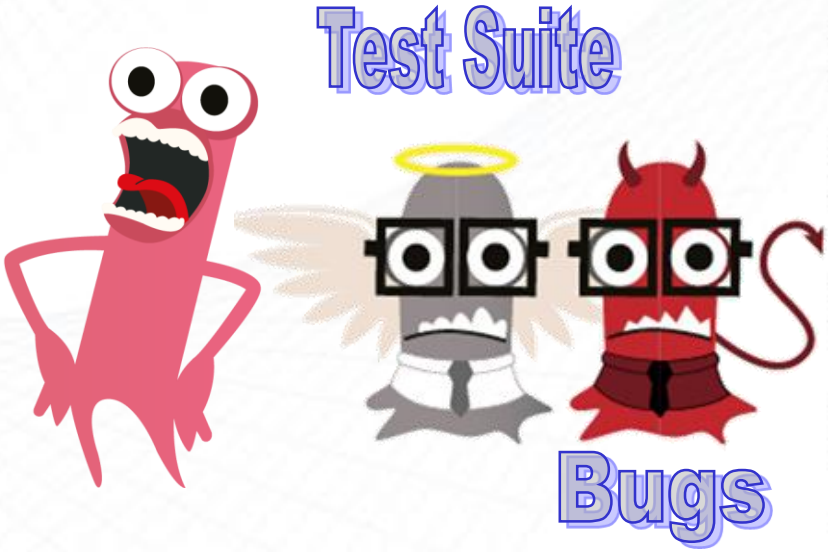
B.H. Far (far@ucalgary.ca)

http://people.ucalgary.ca/~far

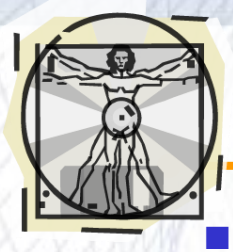


# Recap …

- We have learned techniques and tools to become test experts
- These techniques can be used by developers and test engineers
  - Why we need to test?
  - Exploratory testing
  - Unit testing and Junit
  - Black box, white box testing
  - Code coverage
  - Mutation testing
  - Web/GUI testing
  - Test automation (partially)

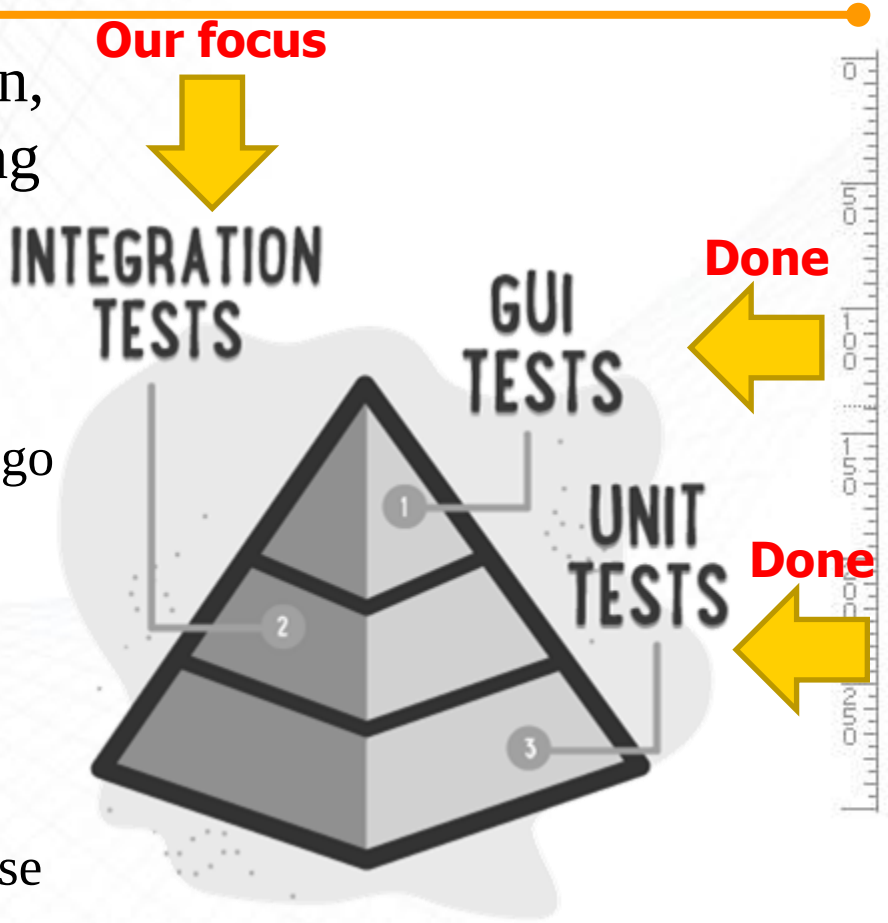


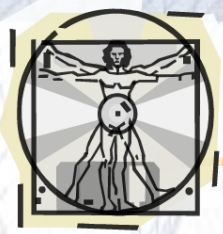

- All are mainly concerned with unit of job: Unit testing



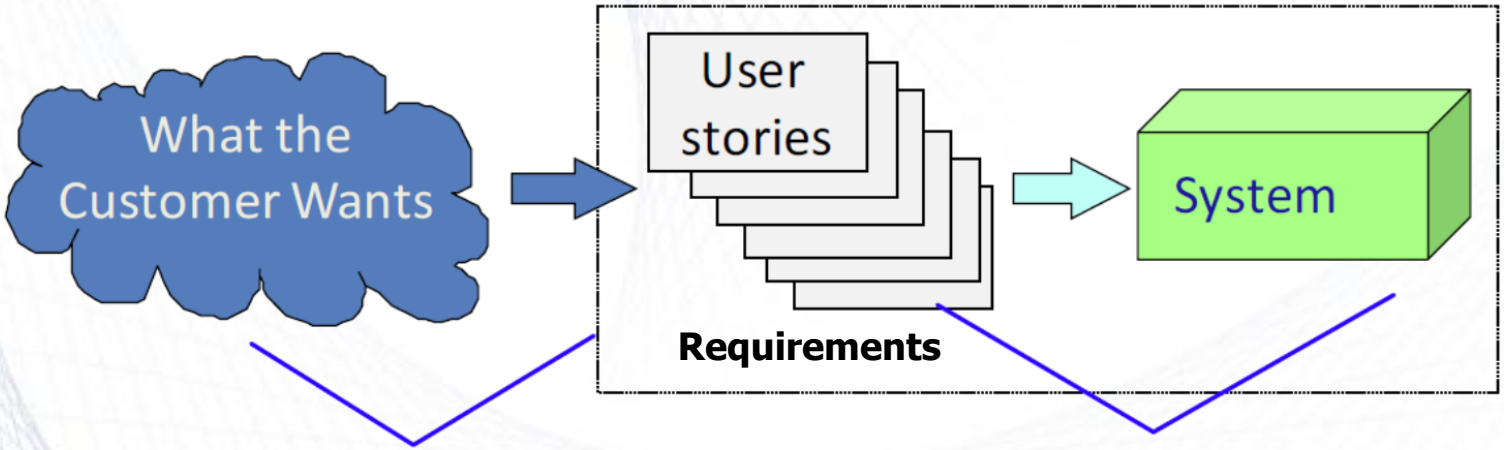
# This Week on …

- Roadmap towards integration, system and acceptance testing with focus on Quality
  - How to build quality into a software project?
  - How to manage quality as we go forward with software development?
  - How to measure quality of software at the end?
  - Decision making process and plan for release and post-release activities





# Validation & Verification

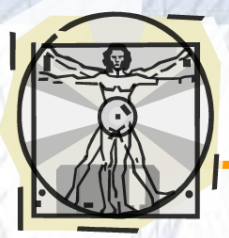



**Validation: checking whether the system meets customer's actual needs**

**Building the right product**

**Verification: whether the system is well-engineered, bug free, etc.**

**Building the product right**



# Contents

- What is software quality?
- Reliability concepts and process
- Pre and post release quality
- Engineering reliability in a software system

SCHULICH School of Engineering

UNIVERSITY OF CALGARY

# Section 1


## Software Quality

# Review: Software Quality

**Software Quality**

**Size & Effort**

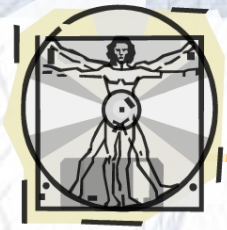
**Predict size**
**Predict effort**
**Predict time**

**Standards & Models**

**ISO 9126**
**ISO 25K**
**etc.**

**Assessment techniques**

**Pre-release**
**Post-release**



# Review: Software Reliability

**Software Reliability**

**Concept**

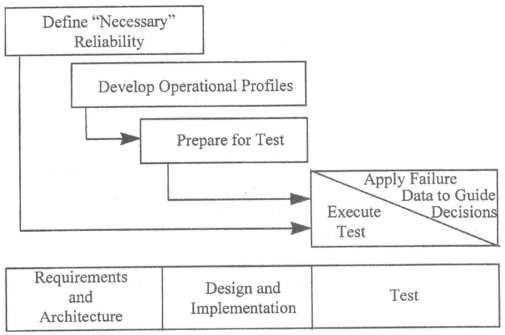
- Reliability
- Availability
- Failure rate
- MTTF
- Failure density
- Etc.

**Model**

- Single Failure Model
- Multiple Failure Model
  - Reliability Growth Model

**Process**

- SRE Process

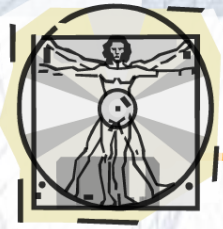


# What is Quality?

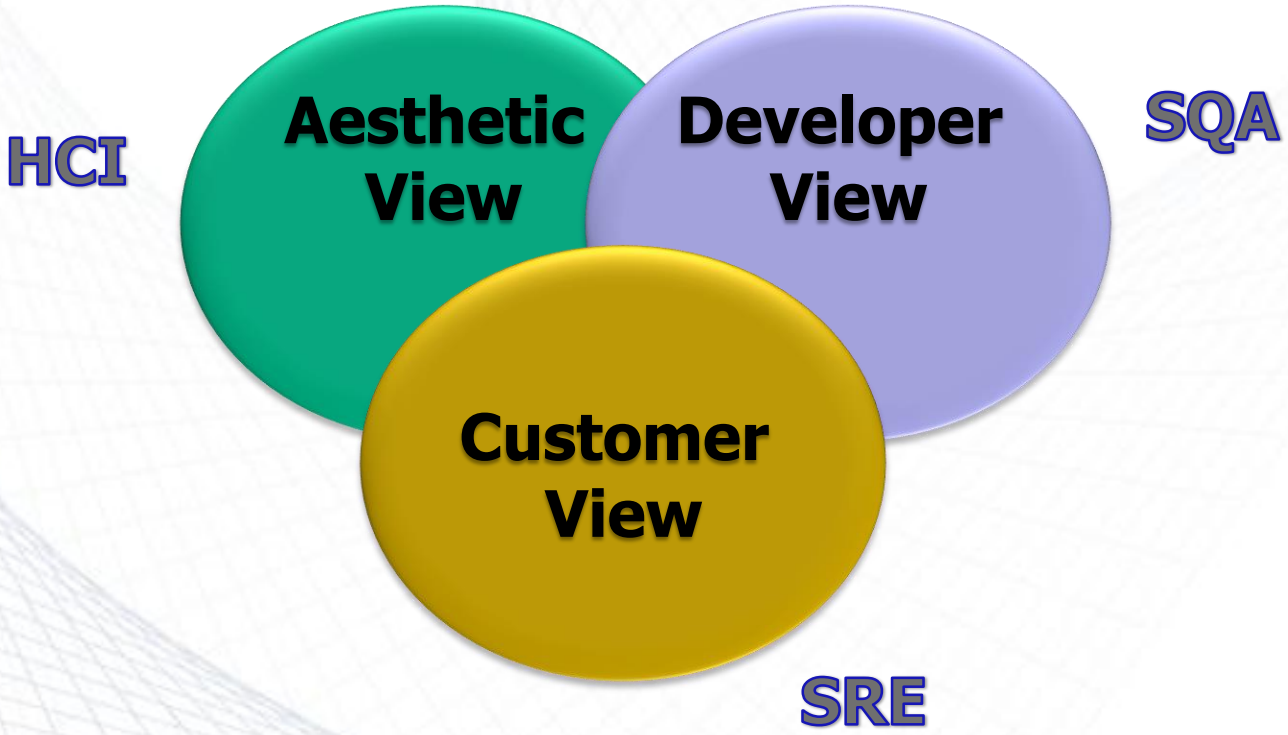
- **Quality popular view:**
  - Something "good" but not quantifiable
  - Something luxury and classy

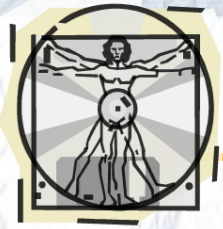
- **Quality professional view:**
  - Conformance to requirement (Crosby, 1979)
  - Fitness for use (Juran, 1970)

**Which one has higher "quality" as a mode of transportation?**

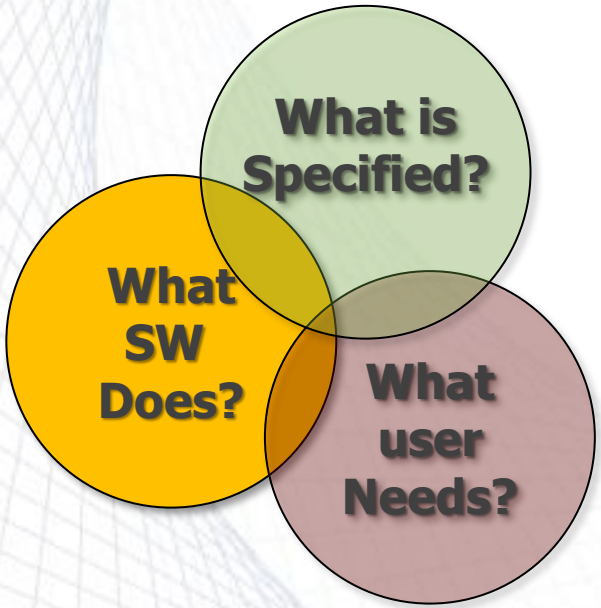


# Quality: Various Views



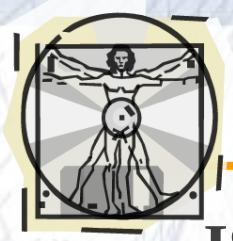


# What is Software Quality?




- **Conformance to requirement**
  - The requirements are clearly stated and the product must conform to it
  - Any deviation from the requirements is regarded as a defect
  - A good quality product **contains fewer defects**
- **Fitness for use**
  - Fit to user expectations: meet user's needs
  - A good quality product **provides better user satisfaction**

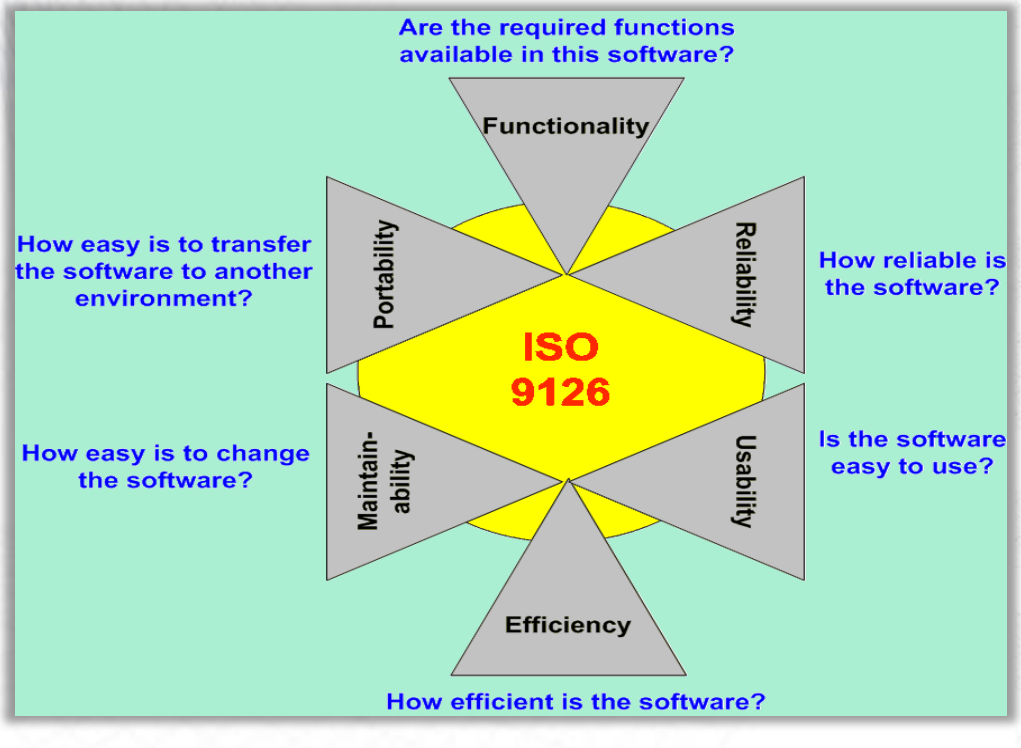
**Both → Dependable computing system**



# Definition: Software Quality

**ISO 8402 definition of QUALITY:**

*The totality of features and characteristics of a product or a service that bear on its ability to satisfy stated or implied needs*




**Reliability** **Maintainability**

**two major components of Quality**

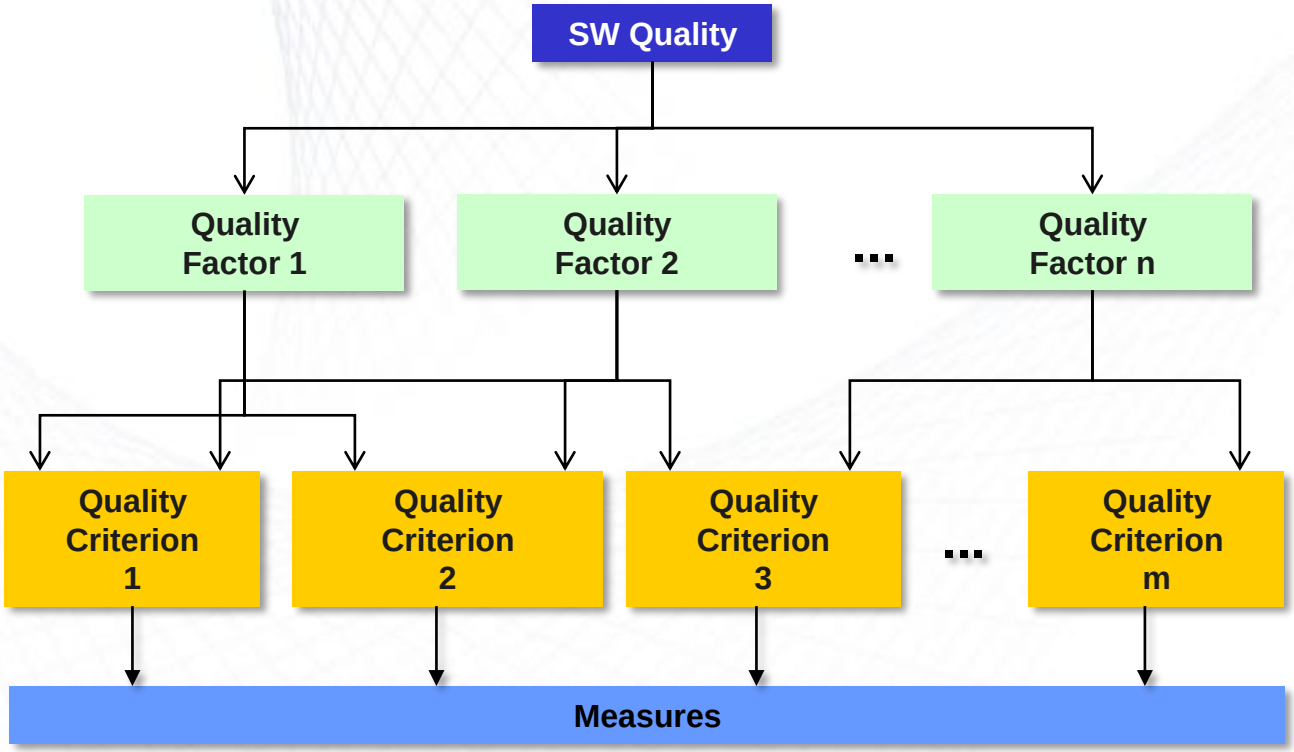
# Quality Model: ISO 9126

| Characteristics | Attributes | | |
|---|---|---|---|
| **1. Functionality** | Suitability | Interoperability | Accuracy |
| | Compliance | Security | |
| **2. Reliability** | Maturity | Recoverability | Fault tolerance |
| | Crash frequency | | |
| **3. Usability** | Understandability | Learnability | Operability |
| **4. Efficiency** | Time behaviour | Resource behaviour | |
| **5. Maintainability** | Analyzability | Stability | Changeability |
| | Testability | | |
| **6. Portability** | Adaptability | Installability | Conformance |
| | Replacability | | |

# Quality Model – Structure

User oriented

Software oriented



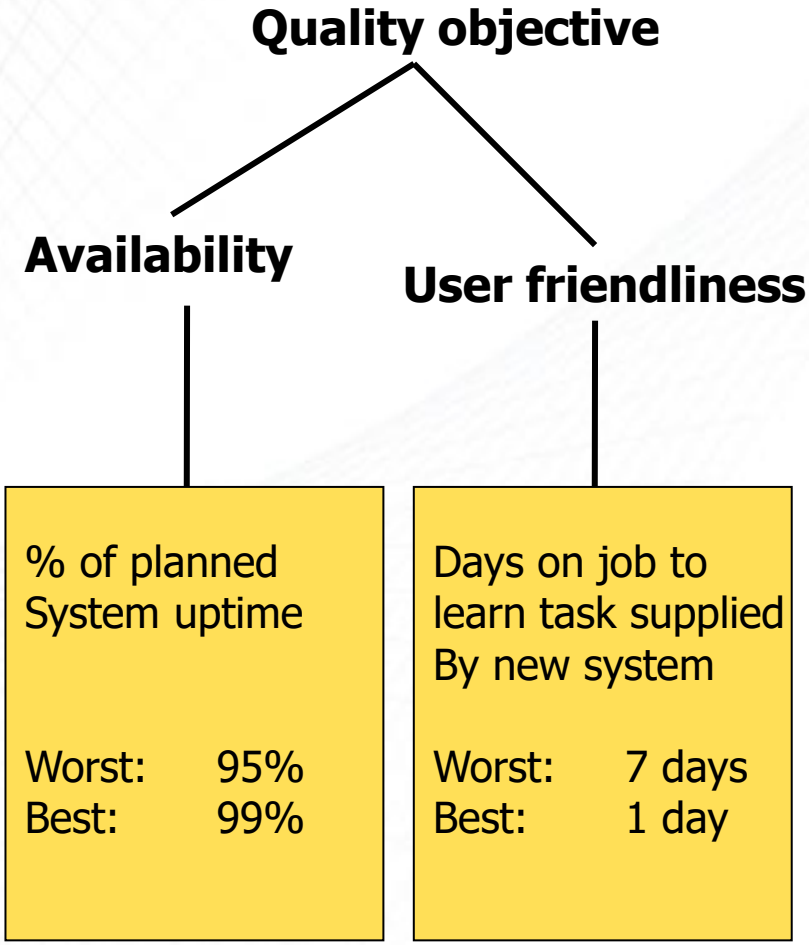

UNIVERSITY OF CALGARY

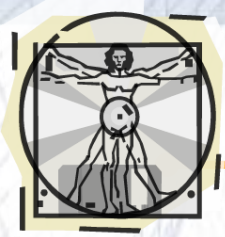
far@ucalgary.ca

# Example: Attribute Expansion

- **Design by measurable objectives:**

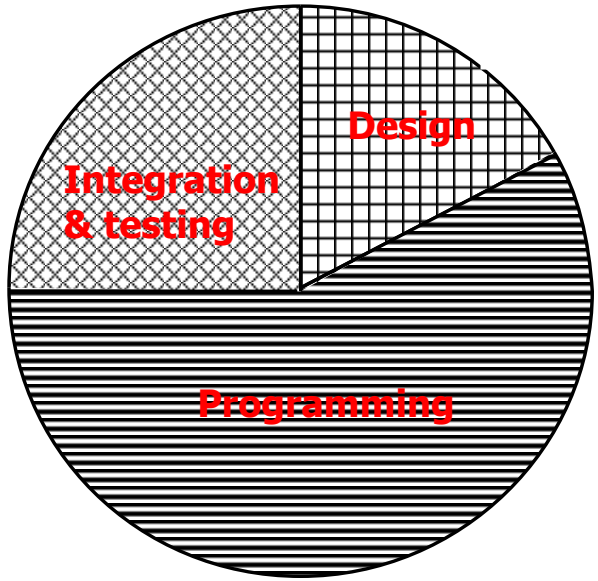
  Incremental design is evaluated to check whether the goal for each increment was achieved





# Quality vs. Project Costs

**Cost distribution for a typical software project**

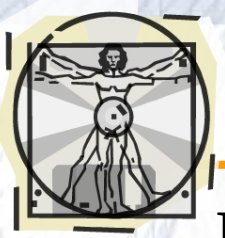




Release

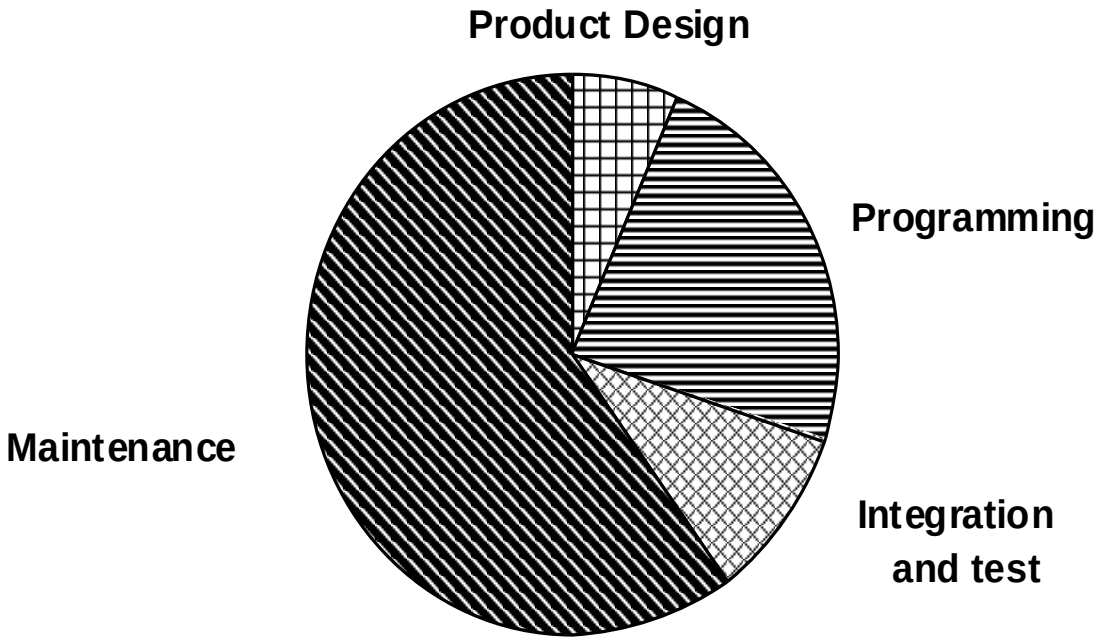
| 15% | 60% | 25% |
|---|---|---|
| Design | Programming | Testing |

**What is wrong with this picture?**



# Total Cost Distribution

**Maintenance is responsible for about 60% of total cost for a typical software project**



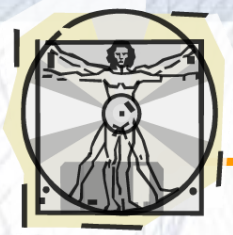

**Questions:**

**1) How to build quality into a system?**

**2) How to assess quality of a system?**

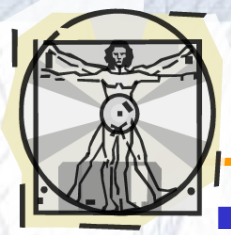
**Developing better quality system will contribute to lowering maintenance costs**

# 1) How to Build Quality into a System?



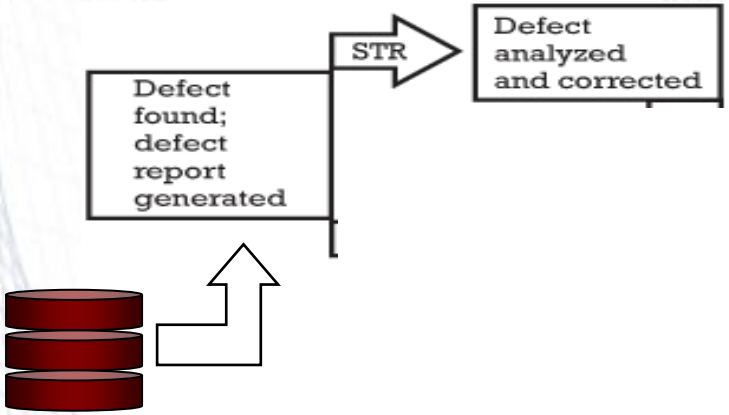
- Developing better quality systems requires:
  - Establishing **Software Quality Assurance (SQA)** programs
  - Establishing **Reliability Engineering (SRE)** process

Later this chapter



# What is SQA?

- Mechanism for bug reporting, tracking and closure
- Plus: development process improvement



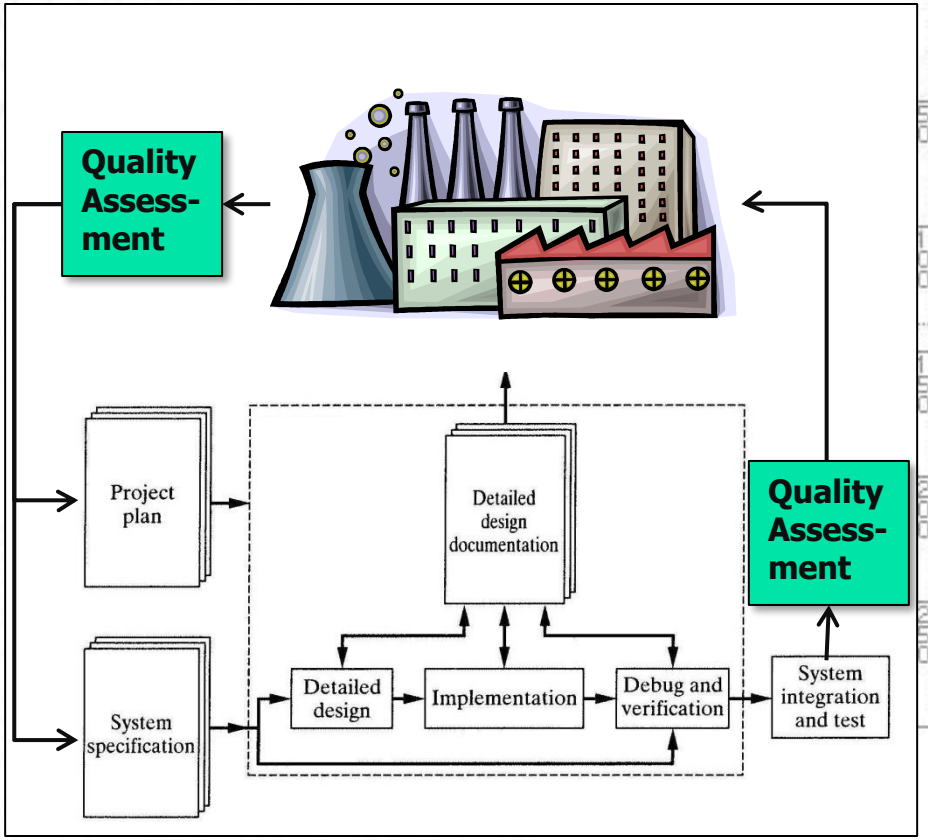

Bug reports DB

SCN: software change notice

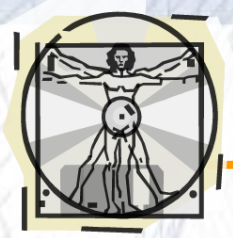
STR: software trouble report

# 2) How to Assess Quality of a System?

- Relevant to both pre-release and post-release
- Pre-release: SRE, certification, standards ISO9001
- Post-release: evaluation, validation, RAM



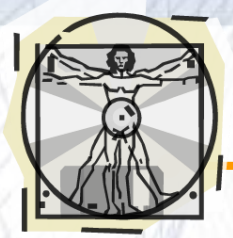

**RAM: Reliability And Maintainability**



# How Do We Assess Quality?

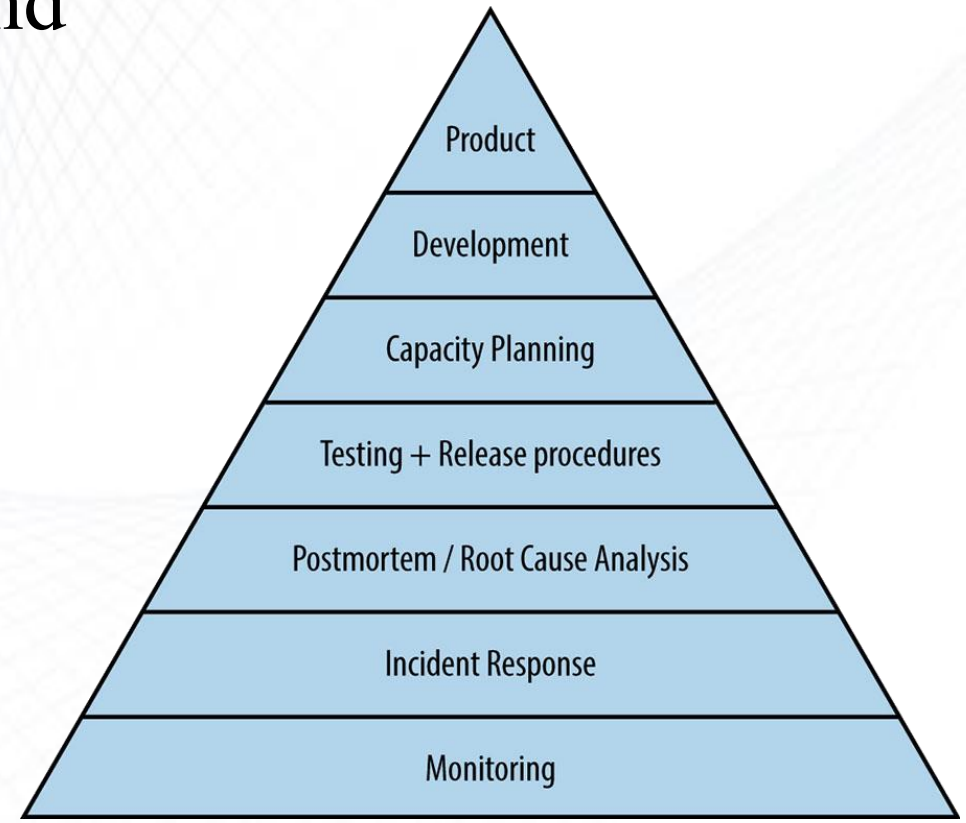
- Ad-hoc (trial and error) approach!
- Systematic approach


Our focus here

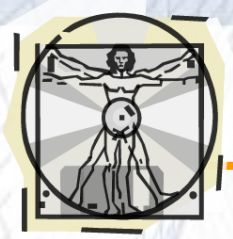


# Post Release Quality

- Google's story and practice

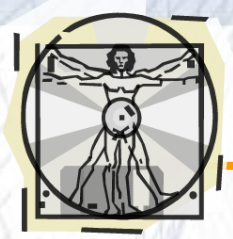



https://landing.google.com/sre/sre-book/toc/



# Pre-release Quality

- Software **inspection** and **testing**
- Methods:
  - SRE
  - Certification
  - Standards ISO9001, 9126, 25000



# Two Main Questions …

- In spite of having many development methodologies, two main questions are:

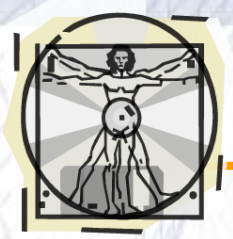
***1. Can we remove all bugs before release?***

***2. How often will the software fail?***

# Two Extremes

- Craftsman (ad-hoc) SENG: fast, cheap, buggy
- Cleanroom SENG: slow, expensive, zero defect
- ***Is there a middle solution?***

**Craftsman Software Develop-ment**



# Two Extremes

- Craftsman (ad-hoc) SENG: fast, cheap, buggy
- Cleanroom SENG: slow, expensive, zero defect
- ***Is there a middle solution?***

**Craftsman Software Develop-ment**

**YES!**

**Using Software Reliability Engineering (SRE) Process**

**Cleanroom Software Develop-ment**

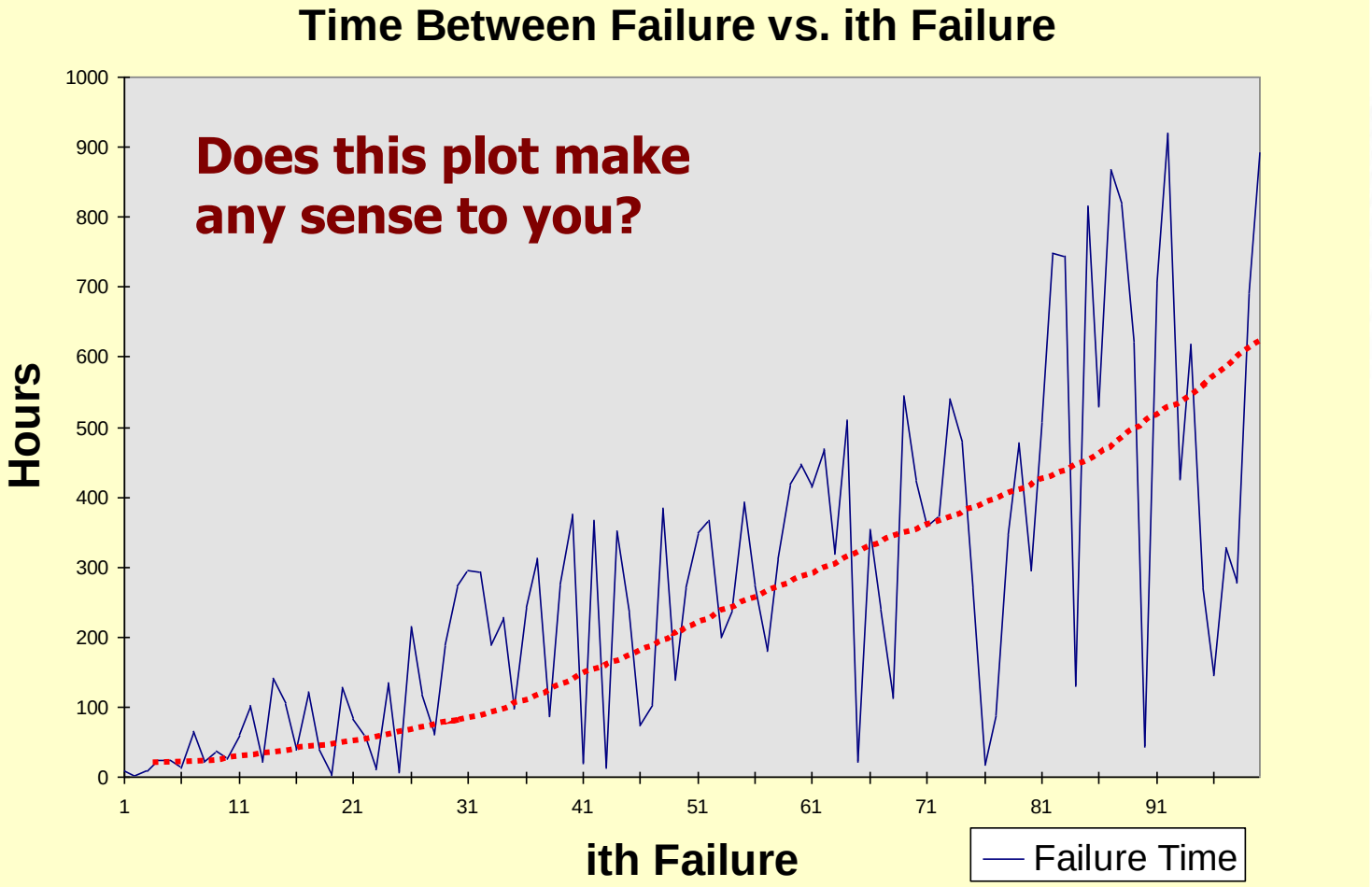
# Can We Remove All Bugs?

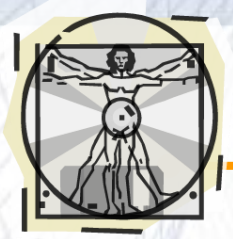
| Size [function points] | Failure potential [development] | Failure removal rate | Failure Density [at release] |
|---|---|---|---|
| 1 | 1.85 | 95% | 0.09 |
| 10 | 2.45 | 92% | 0.20 |
| 100 | 3.68 | 90% | 0.37 |
| 1000 | 5.00 | 85% | 0.75 |
| 10000 | 7.60 | 78% | 1.67 |
| 100000 | 9.55 | 75% | 2.39 |
| **Average** | **5.02** | **86%** | **0.91** |

Defect potential and density are expressed in terms of defects per function point (1FP is about 60 Java Loc)

**The answer is usually NO!**

# What Can We Learn from Failures?



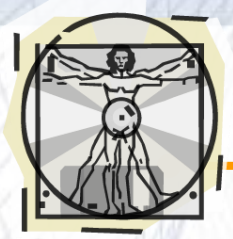


# How to Handle Defects?

- Table below gives the time between failures for a software system:

| ***Failure no.*** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| ***Time since last failure (hours)*** | 6 | 4 | 8 | 5 | 6 | 9 | 11 | 14 | 16 | 19 |

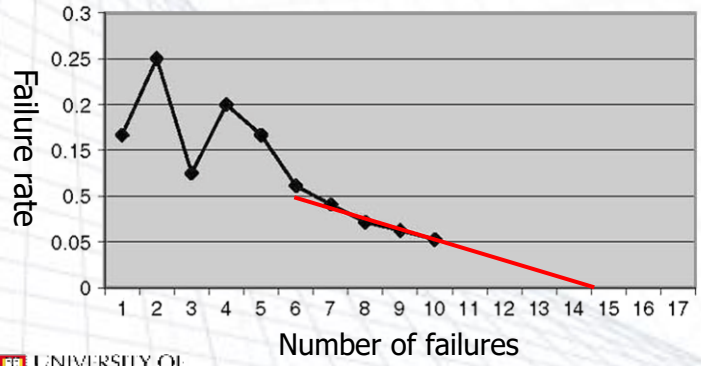
- What can we learn from this data?
  - System reliability?
  - Approximate number of bugs in the system?
  - Approximate time to remove remaining bugs?

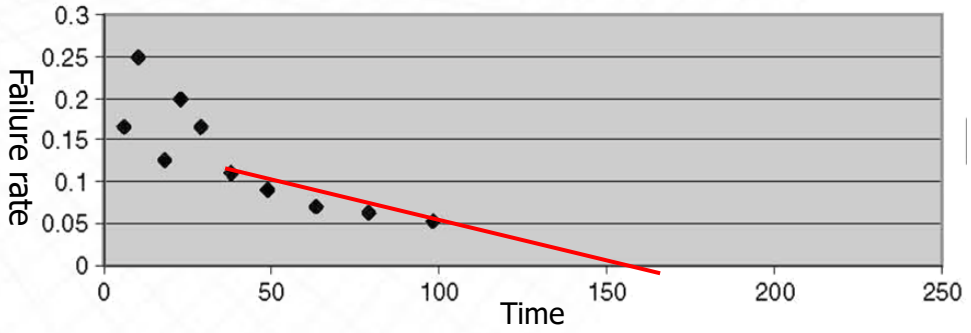


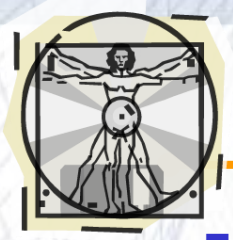
# What to Learn from Data?

- The inverse of the inter-failure time is the ***failure intensity*** (= failure per unit of time)

| *Failure no.* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |
|---|---|---|---|---|---|---|---|---|---|---|
| *Time since last failure (hours)* | *6* | *4* | *8* | *5* | *6* | *9* | *11* | *14* | *16* | *19* |
| *Failure intensity* | *0.166* | *0.25* | *0.125* | *0.20* | *0.166* | *0.111* | *0.09* | *0.071* | *0.062* | *0.053* |





# What to Learn from Data?

- Mean-time-to-failures MTTF (= average failure rate)
  MTTF = (6+4+8+5+6+9+11+14+16+19)/10 = 9.8 hour
- System reliability for 1 hour of operation

$$R = e^{-\lambda t} = e^{-t/MTTF} = e^{-1/9.8} = 0.90299$$

- Fitting a straight line to the graph in (a) would show an x-intercept of about 15. Using this as an estimate of the total number of original failures, we estimate that there are still five bugs in the software.
- Fitting a straight line to the graph in (b) would give an x-intercept near 160. This would give an additional testing time of 62 units to remove all bugs, approximately.

SCHULICH School of Engineering

UNIVERSITY OF CALGARY


# Section 2

## Reliability Concepts

# Reliability Theory

- Reliability theory developed apart from the mainstream of probability and statistics, and was used primarily as a tool to help nineteenth century maritime and life insurance companies compute profitable rates to charge their customers. Even today, the terms “failure rate” and “hazard rate” are often used interchangeably
- Probability of survival of merchandize after one MTTF is $R = e^{-1} = 0.37$

From Engineering Statistics Handbook

# Reliability: Natural System

- Natural system life cycle
- Aging effect: Life span of a natural system is limited by the maximum reproduction rate of the cells

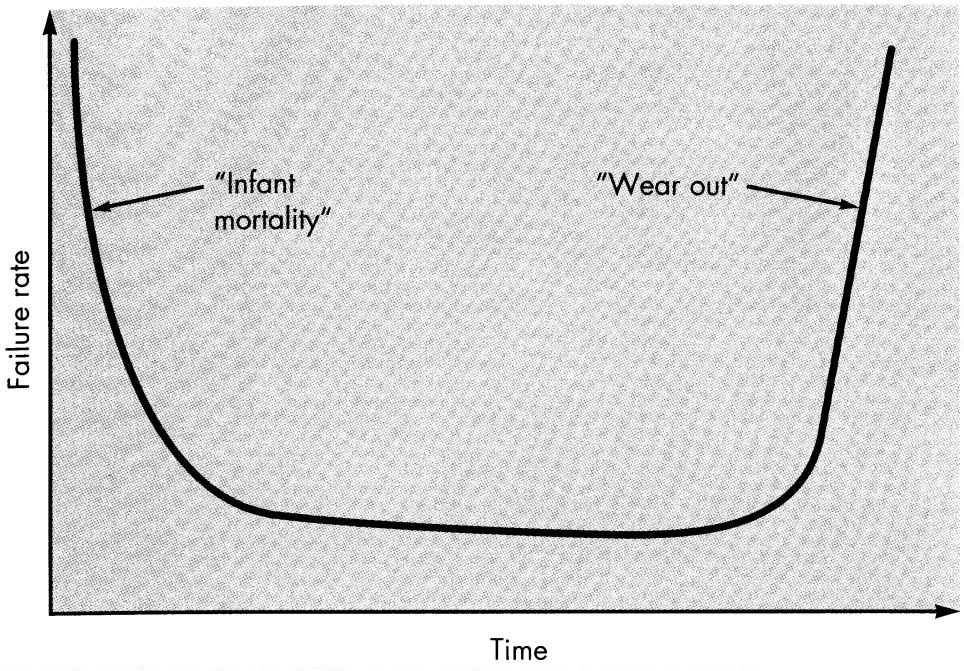
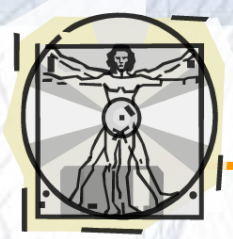



*Figure from Pressman's book*



# Reliability: Hardware

- Hardware life cycle
- Useful life span of a hardware system is limited by the age (wear out) of the system

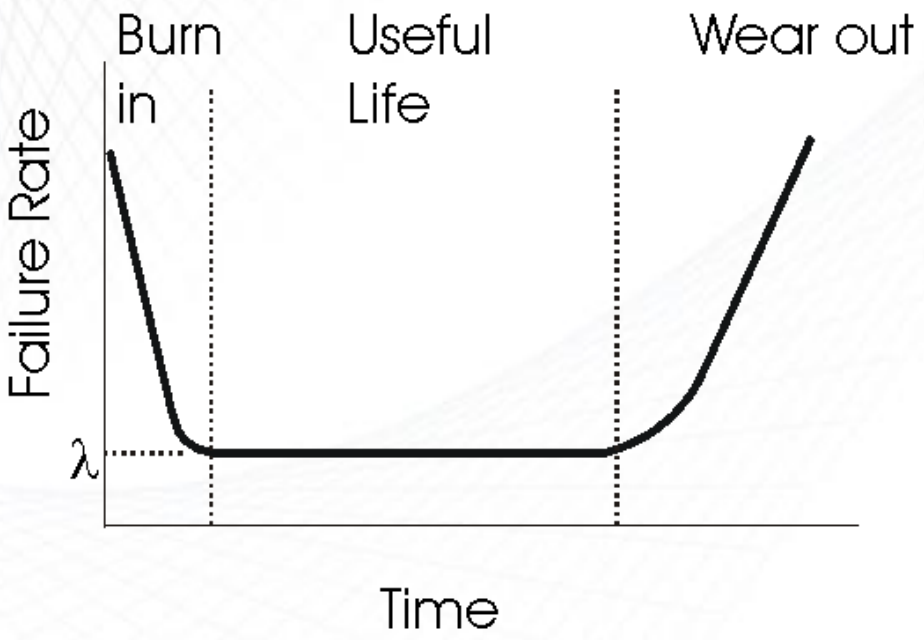
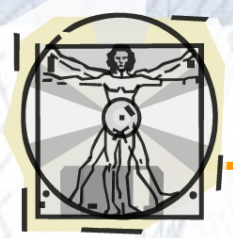



*Figure from Pressman's book*



# Reliability: Software

- Software life cycle.
- Software systems are changed (updated) many times during their life cycle
- Each update adds to the structural deterioration of the software system

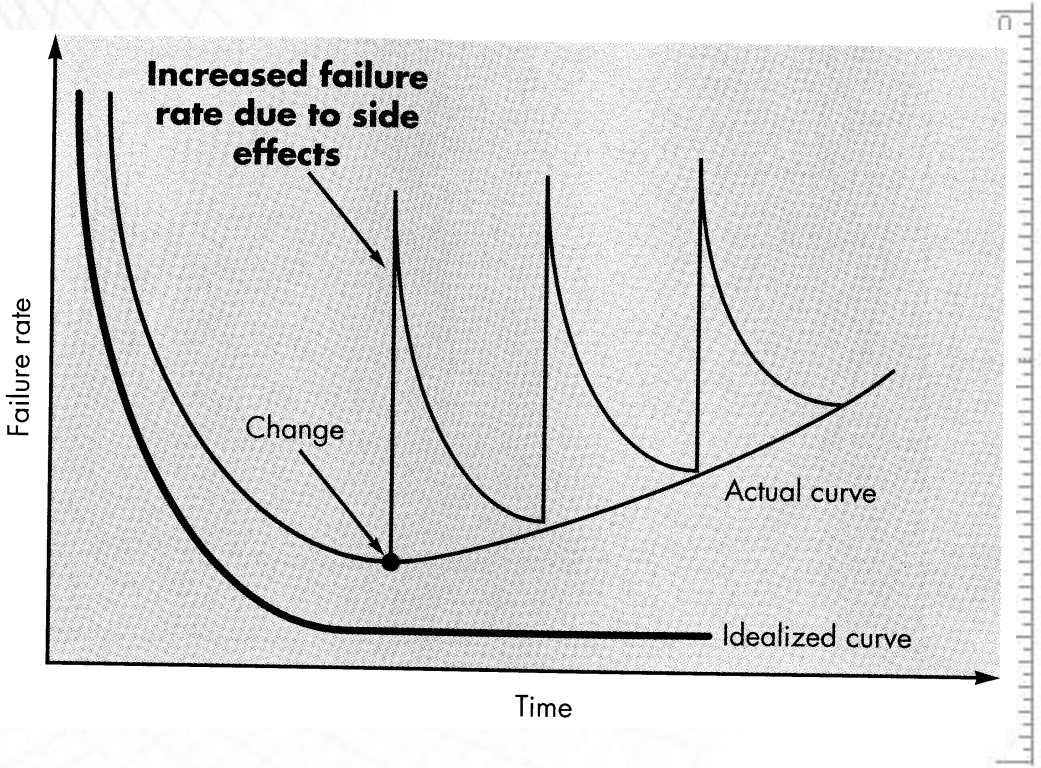
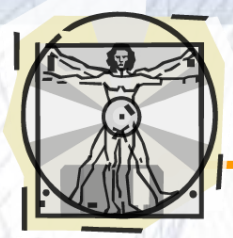



*Figure from Pressman's book*

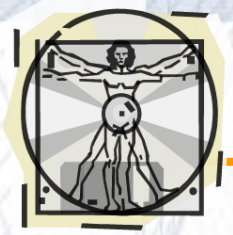
# Software vs. Hardware

- Software reliability doesn't decrease with time, i.e., software doesn't wear out
- Hardware faults are mostly ***physical faults***, e.g., fatigue
- Software faults are mostly ***design faults*** which are harder to measure, model, detect and correct



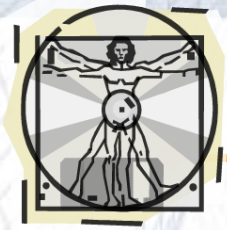
# Software vs. Hardware

- Hardware failure can be “fixed” by replacing a faulty component with an identical one, therefore no reliability growth
- Software problems can be “fixed” by changing the code in order to have the failure not happen again, therefore reliability growth is present
- Software does not go through production phase the same way as hardware does
- **Conclusion:** hardware reliability models may not be used identically for software



# Reliability: Science

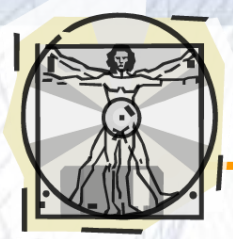
- Exploring ways of implementing "reliability" in software products
- Reliability Science's goals:
  - Developing "models" (regression and aggregation models) and "techniques" to build reliable software
  - Testing such models and techniques for adequacy, soundness and completeness



# Reliability: Engineering

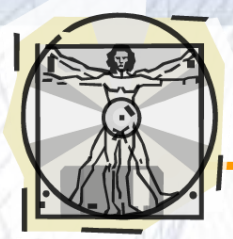
- Engineering of "reliability" in software products ← Managed quality
- Reliability Engineering's goal:

  developing software to reach the *market*
  - With "minimum" development time
  - With "minimum" development cost
  - With "maximum" reliability
  - With "minimum" expertise needed
  - With "minimum" available technology



# What is SRE? /1

- Software Reliability Engineering (SRE) is a multi-faceted discipline covering the software product lifecycle
- It involves both ***technical*** and ***management*** activities in three basic areas:
  - Software Development and Maintenance
  - Measurement and Analysis of reliability data
  - Feedback of reliability information into the software lifecycle activities



# What is SRE ? /2

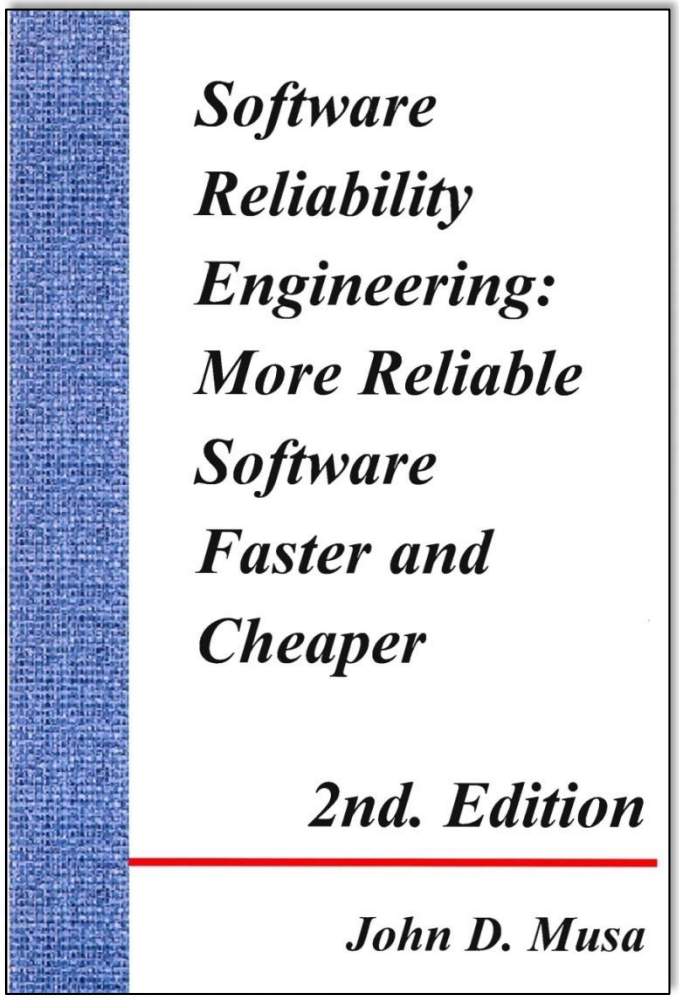
- SRE is a practice for quantitatively planning and guiding software development and test, with emphasis on reliability and availability
- SRE simultaneously does three things:
  - It ensures that product reliability and availability meet user needs
  - It delivers the product to market faster
  - It increases productivity, lowering product life-cycle cost
- In applying SRE, one can vary relative emphasis placed on these three factors


# Section 3

## Software Reliability Engineering (SRE) Process

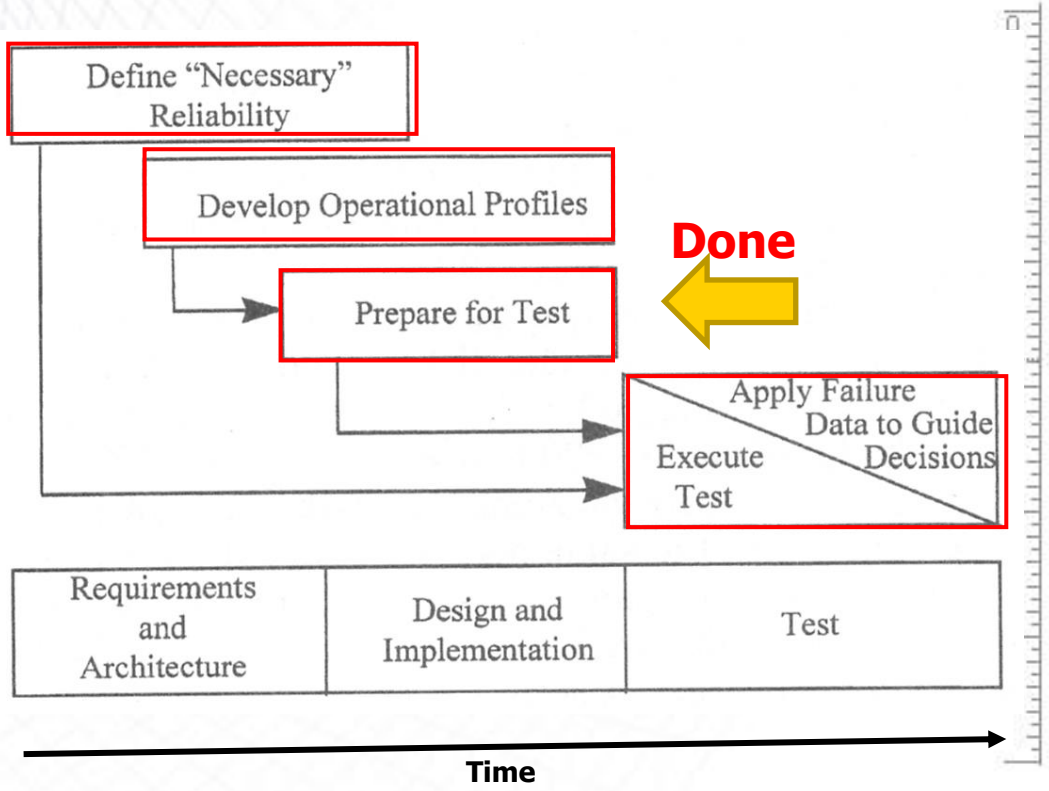
# Reference

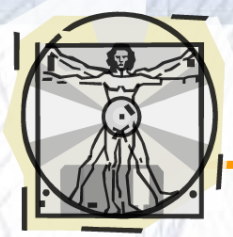
- Dr. Musa's Software Reliability Engineering, 2 Ed
- Chapter 1



# SRE: Process /1

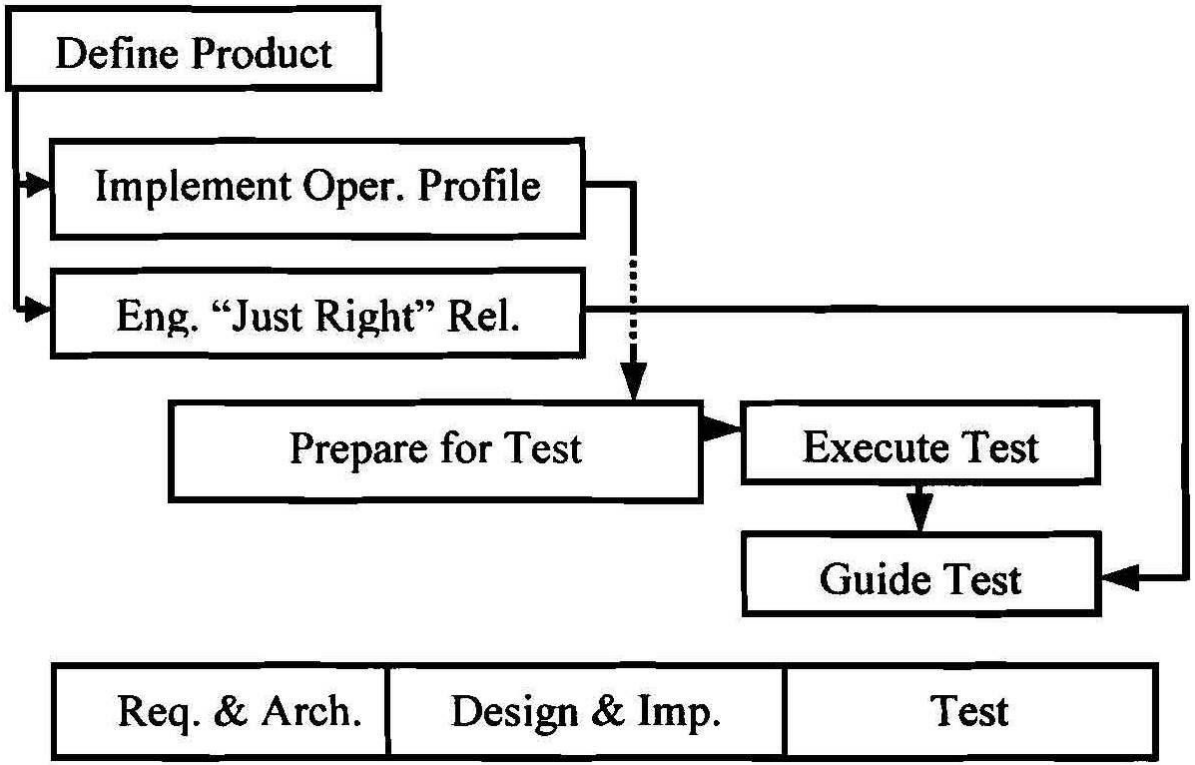
- There are 5 steps in SRE process:
  - Define necessary reliability
  - Develop operational profiles
  - Prepare for test
  - Execute test
  - Apply failure data to guide decisions



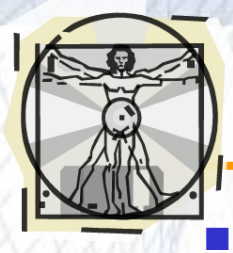


# SRE: Process /2

- Modified version of the SRE Process




*Ref: Musa's book 2nd Ed*



# Conclusions …

- Software testing and reliability assessment can offer metrics and measures to help elevate a software development organization to the upper levels of software development maturity
- Practical implementation of an effective SRE program is a non-trivial task
- Mechanisms for collection and analysis of data on software product and process quality must be in place
- Fault identification and elimination techniques must be in place
- Other organizational abilities such as the use of reviews and inspections, reliability based testing and software process improvement are also necessary for effective SRE