SENG 637
Dependability and Reliability of Software Systems

Chapter 1: Introduction

Department of Electrical & Software Engineering, University of Calgary

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

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

Introduction

Instructor

- **Name**: Behrouz Far, Professor, PhD., PEng.
- **Homepage**: http://people.ucalgary.ca/~far/
- **E-mail**: far@ucalgary.ca

  Email subjects should start with "SENG 637 – " followed by your own title
- **Office hours**: Tuesday and Thursday mornings

Course

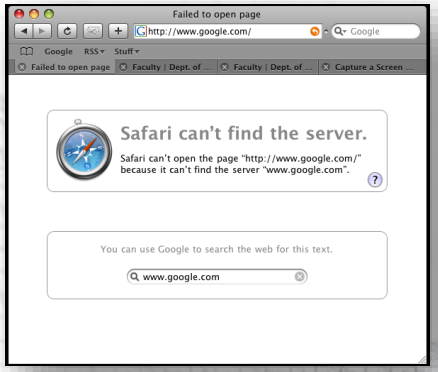
- **Name:** Dependability & Reliability of Software Systems
- **Homepage:** http://d2l.ucalgary.ca/
- **Course TA:** Yousef Mehrdad
- **Class hours:** Lecture: Mon 17:30-20:15 PM

Contents

Shorter version:

- **How to avoid these?**

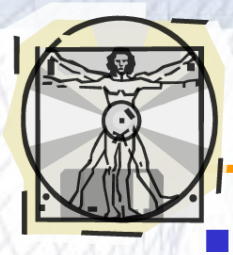
glitches/
failures/
defects/
bugs are
everywhere!



Contents

Longer version:

- What is software testing, reliability and quality?
- What is software testing and reliability engineering process?
- How to test a software system and assess test adequacy?
- How to assess quality of a software system?



Objectives

- **Understanding fundamental of software testing, reliability and quality**
  - Understand software testing and reliability engineering process
  - Understand software design with quality mind set
  - Have an overview of the careers outlooks of software testing and quality assessment
  - Gain experience working in a software testing team
- **Designing test cases and improving test suites**
  - Understand test planning process, test selection and prioritization strategies
  - Assess test adequacy using test coverage metrics
- **Assessing reliability and quality of software**
  - Use test tools and frameworks and track quality during development and post-release

CEAB Graduate Attributes

- Each course has its unique signature
- In SENG637 we focus on 3 main GAs

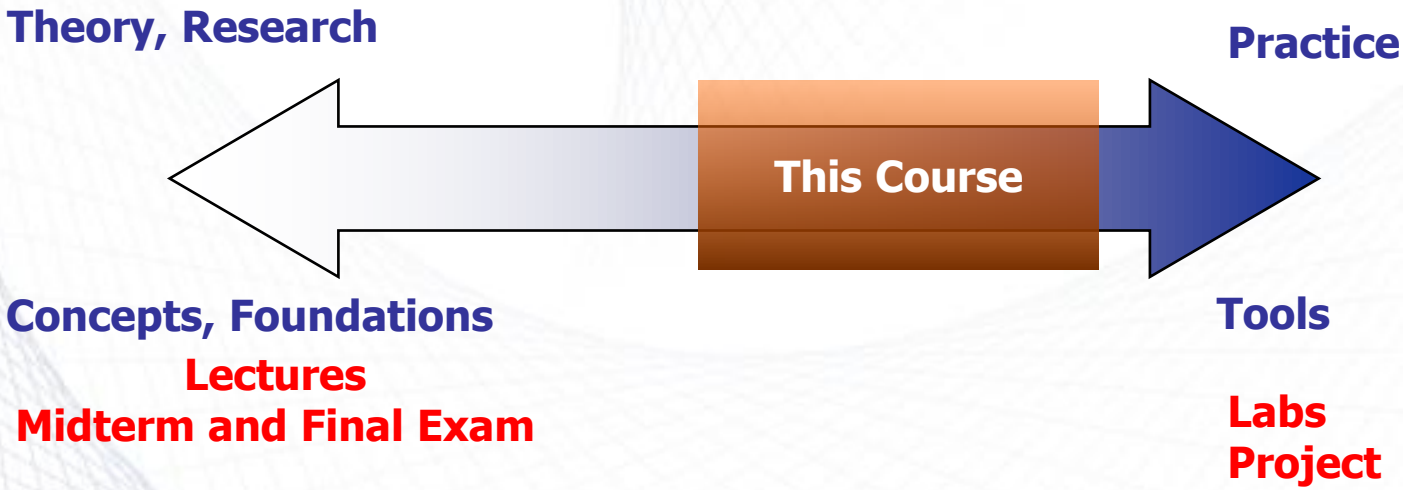
A1. A knowledge base for engineering
A2. Problem analysis
A3. Investigation
A4. Design
A5. Use of engineering tools
A6. Individual and team work
A7. Communication skills
A8. Professionalism
A9. Impact of engineering on society/environment
A10. Ethics and equity
A11. Economics and project management
A12. Life-long learning

The level at which the learning outcome is addressed in this course:

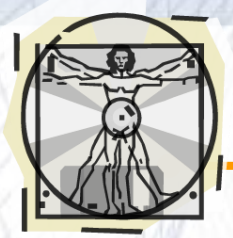
I (Introduced):	Introductory level
D (Developed):	Intermediate development level
A (Applied):	Advanced application level

Theory + Practice

- We need to mix Theory and Practice in this course



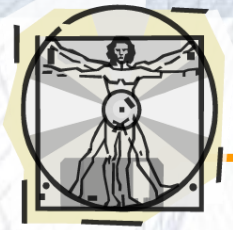
- Practical Engineering course
  - Novelty of problem; multiplicity of solution



Course Delivery

- GoF (Good Old Fashion) style Lectures
- Interactive and competitive learning environment
- No flip, flop, summersault, etc.!
- No non-sense entertainment, no info-tainment, no monkey business!

- Attend lectures and labs: there is more to lectures than to the slides and notes
- Yes, I have more than 1,000 slides and I don't go through all one by one. I emphasize on the important ones depending on the class discussion



What Do You Need to Know?

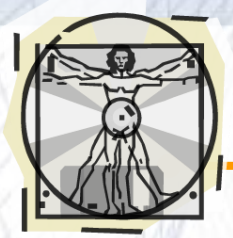
- Java programming language
  - You will test Java applications in the Lab
  - You will develop test cases in the Java environment
- Basics of software engineering
- Teamwork
  - You are going work as a team in the Lab, so be a team player

Scope of the Course

Course Study Material

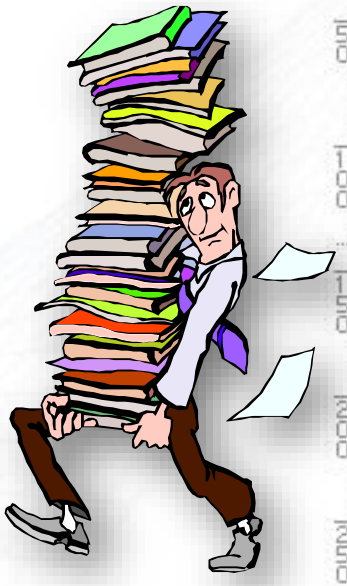
Lecture notes (slides ch1-12)

- Slides Chapter 1-12



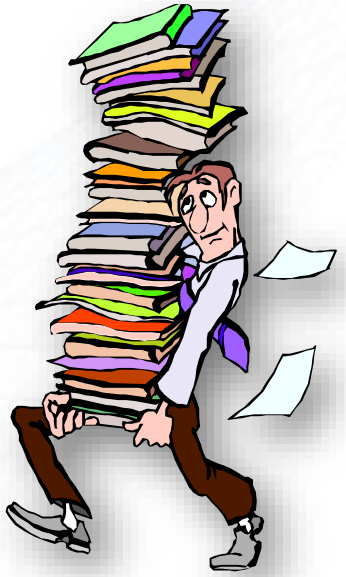
Detailed Contents

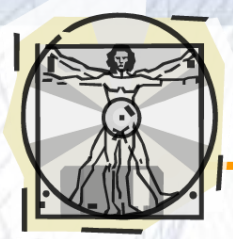
- **Ch1:** Introduction & terminology
- **Ch2:** Foundations of testing
- **Ch3:** Unit testing
- **Ch4:** Black box & combinatorial testing
- **Ch5:** White box testing: control and data flow coverage
- **Ch6:** Mutation testing
- **Ch7:** GUI/Web testing and test automation



Contents (cont'd)

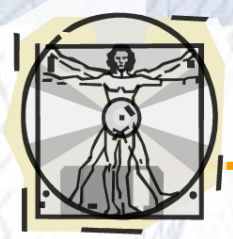
- **Ch8:** Reliability engineering concepts and process
- **Ch9:** System reliability
- **Ch10:** Reliability assessment tools and techniques
- **Ch11:** Integration, system, acceptance testing
- **Ch12:** Post release activities and maintenance





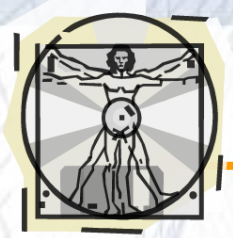
What Will Be Covered?

- Test management
  - How to design tests (ad-hoc) ← Ch.2
  - How to automate tests (Junit) ← Ch.3
  - How to run tests automatically ← Ch.3
  - How to systematically create and manage tests
    - Black-box ← Ch.4
    - White-box ← Ch.5A and Ch.5B
      - Control flow based approach
      - Data flow based approach



What Will Be Covered?

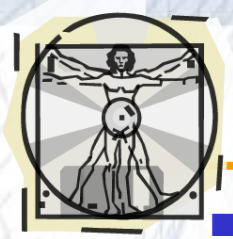
- More test management
  - How to design fault-based test (mutation) ← Ch.6
  - How to design repeatable tests (automation) ← Ch.7
- Test wrap up
  - Integration, system, acceptance tests ← Ch.11, Ch.12
- Test assessment
  - How to define quality of software ← Ch.8
  - How to set and manage reliability of a system ← Ch.9
  - How to assess reliability of software system ← Ch.10



Assignments

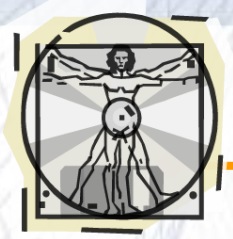
- 5 Assignments (one weeks each)

- A1: Bug tracking
- A2: Unit testing using Junit
- A3: Test coverage and adequacy
- A4: Mutation, GUI and Web testing
- A5: Reliability assessment



Assignments - Policies

- 5 Assignments (you have one week to complete each)
- Each assignment will be discussed first in class
  - In lecture time
  - The descriptions will be online before each assignment
  - **Come to class prepared:** you are expected to read the Assignment document before coming to the class
- Reports for each assignment by the due date of each assignment (exact dates will be posted)
- One mini-project: A small developed software will be given
  - You do the test, assessment and report (details later)

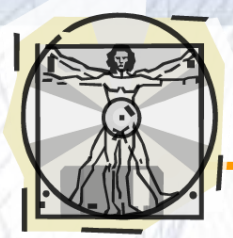


Learning Activities

- This is a practical course
- Usually, you need to spend a few hours out of the scheduled time to finish each assignment
  - Note that we care (and are very careful) about your workload
  - The goal: to prepare you for the software industry
- Other than the lecture times, if you have questions, contact TA, post on D2L, or email instructor
- Watch for office hours posted

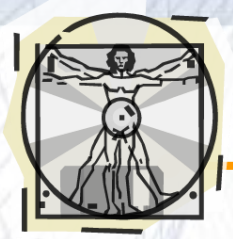
Late Reports

- Carefully check the schedule for report due dates. **Late reports will not get any marks**
- We (the TA and myself) are here to work with and help you!
- Rest assured that the testing knowledge you gain in these assignments prepare you for a real testing job
- The assignments have been designed in consultation with the software industry, and with the help of a few recent top students



Learning Activities: Groups

- All assignments and the project will be done in groups
  - Please form your groups of 2 or 4
  - You can do them individually if you want but don't expect to get less workload for doing the job alone!

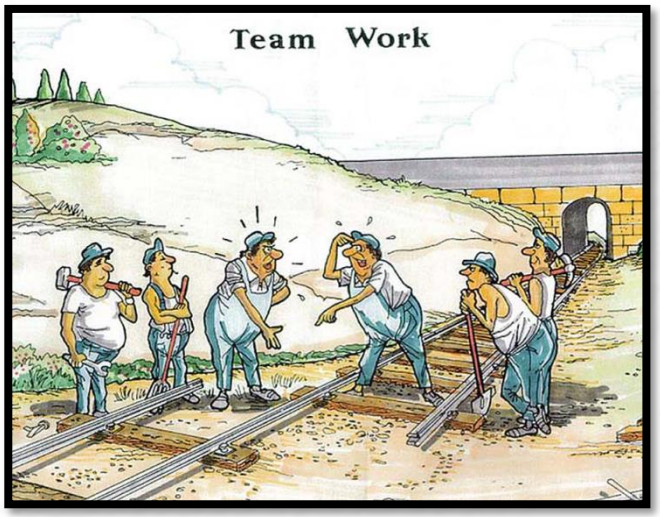


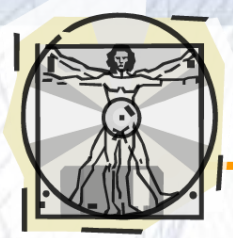
Group work: Very Important

- It is the team's responsibility to distribute their tasks evenly and deliver high quality reports and demos
- If you face any problem with individuals not working as they should, you must report it to the TAs and me, ASAP.
  - We assume everything is fine and everybody contributes the same, if you don't report
  - We can not help if you don't report on-time
  - **If a student does not contribute enough their project/assignment mark may be less than their teammates**

Why Group Work?

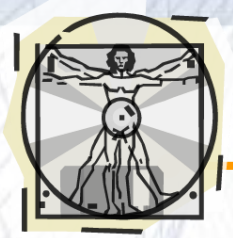
- This is how you will work eventually in real life
  - You will always have people in your team that are not either as good as you or they are better than you
  - You still need to manage the situation and deliver high quality material on-time





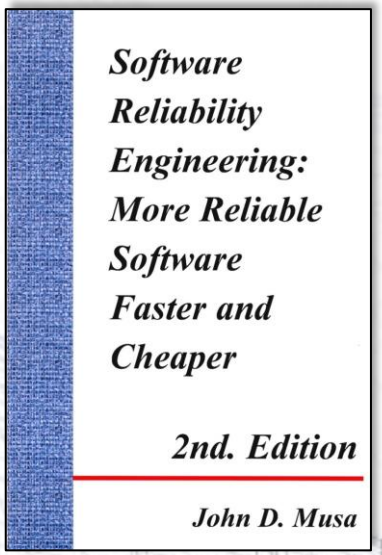
Missing & Deferrals

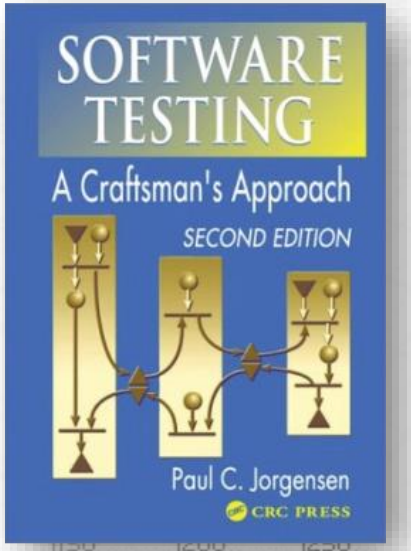
- University rules will be followed strictly
- If you miss a deadline or need more time to submit your report for a legitimate reason
  - such as illness, or a serious personal issue
- Please email and provide the instructor with proper documentation ASAP to arrange an adjustment



Books and Textbooks

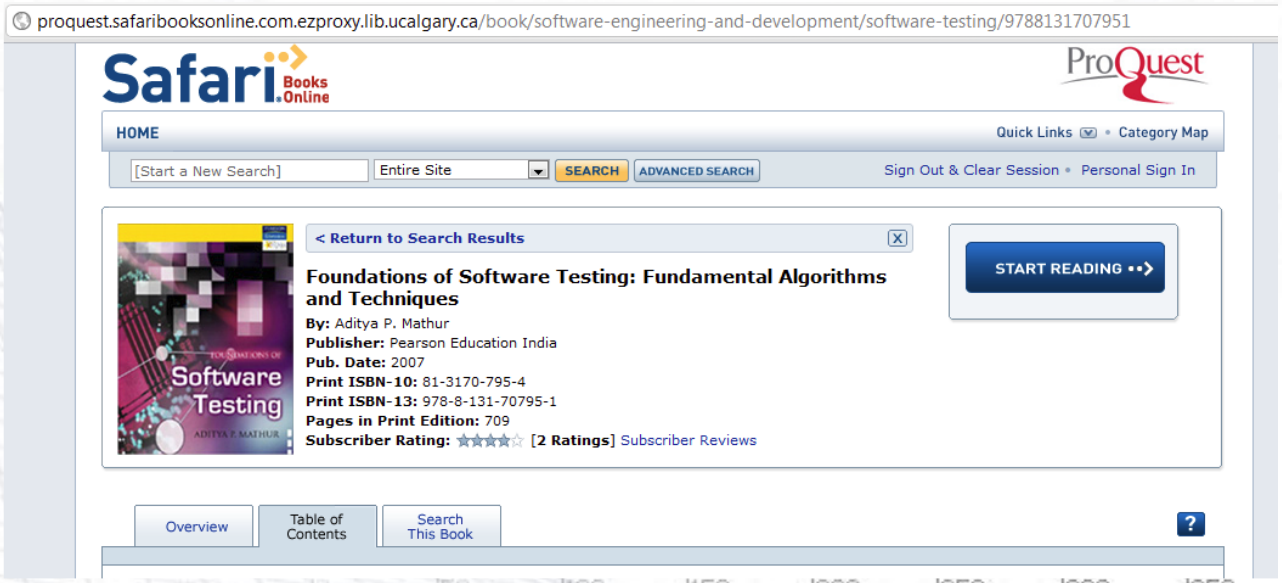
- Good news! There are NO required textbooks for this course
- We will learn from a selection of books
- Some recommended books:





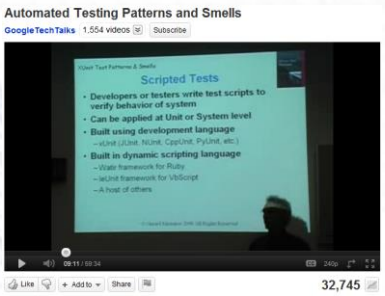
Books and Textbooks

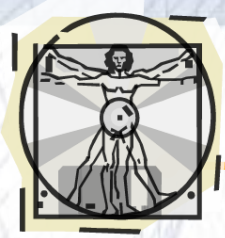
- The book is available full-text for FREE (from UofC IP addresses).
- http://proquest.safaribooksonline.com.ezproxy.lib.ucalgary.ca/book/software-engineering-anddevelopment/software-testing/9788131707951



Other Study Materials

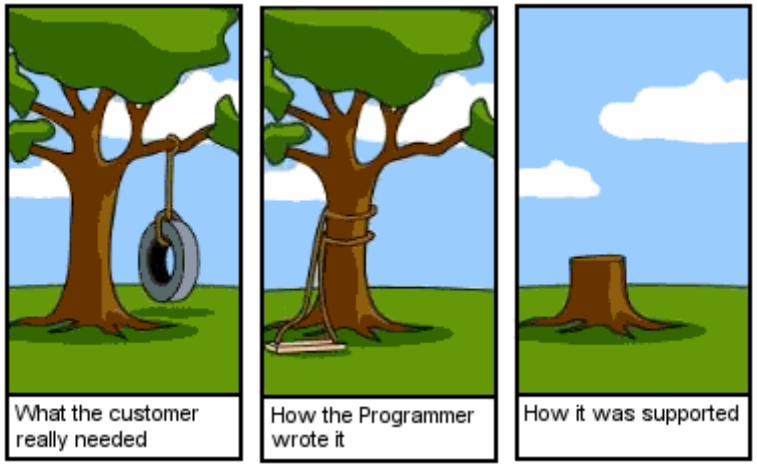
- Lots of online courses and videos

		
How To Recruit, Motivate, and Energize Superior Test engineers	Software testing Fundamentals, Microsoft virtual academy	Google Test Automation Conference (GTAC)
		
Automated Testing Patterns and Smells	Software Test Professionals (STP) Conference	



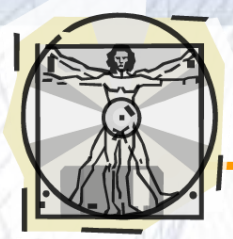
Question to Ask

- Do I really need to take this course?
- Answer depend on you!
- Take this course if you want to avoid these in your career as a designer, tester and quality controller



Pre-Assessment Question

- We test software to:
  - Demonstrate conformance to requirements
  - Find faults in software
  - Reduce costs in software life cycle
  - Show system meets user needs
  - Assess software quality
  - ~~Prove that the software has no defect~~



Now Your Turn

- Which program you are in?
  - SENG? Comp Science? Other?
- How much testing you have learned in courses so far?
- How much software testing have you done on your own?
  - Testing your own code
  - Testing others' code
- How much software testing have you learned in workplace?
  - Internship/co-op?
- What do you expect from this course? (besides a good mark!)

SCHULICH School of Engineering

UNIVERSITY OF CALGARY

Section 2

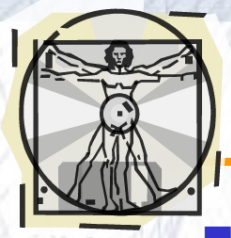
Terminology

What Affects Quality?

Cost

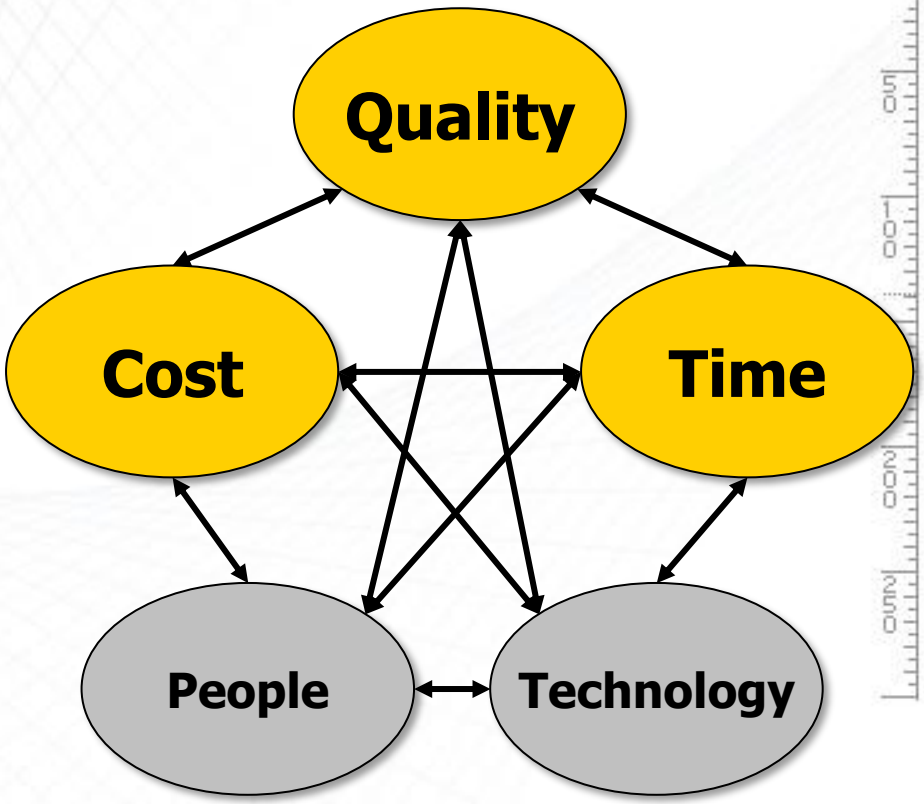
Quality

Time



What Affects Software Quality?

- **Time:**
  - Meeting the project deadline.
  - Reaching the market at the right time.
- **Cost:**
  - Meeting the anticipated project costs.
- **Quality (reliability):**
  - Working fine for the designated period on the designated system.



Terminology & Scope

The ability of a system to avoid failures that are more frequent or more severe, and outage durations that are longer, than is acceptable to the users.

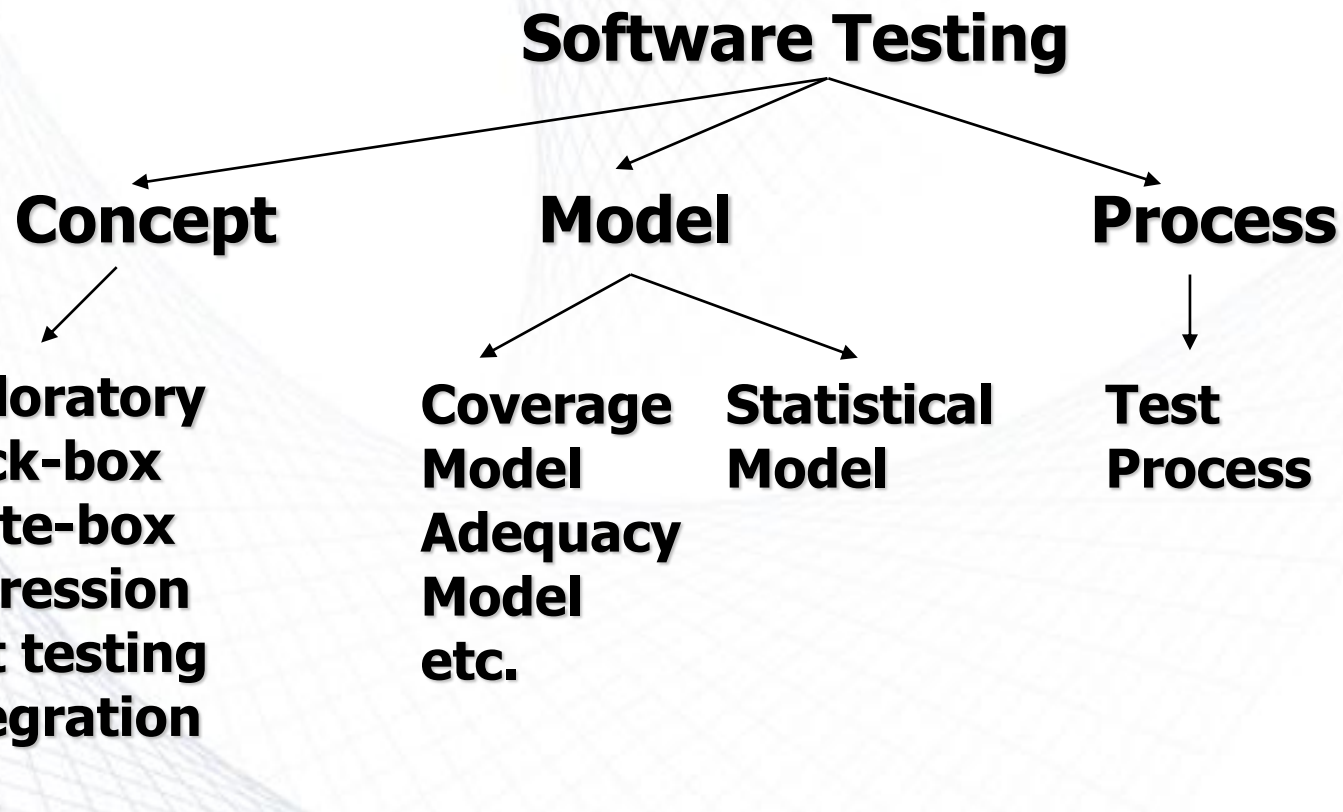
Dependability

- **Threats**
  - Failures
  - Faults
  - Errors

  bug
  defect
- **Attributes**
  - Availability
  - Reliability
  - Safety
  - Confidentiality
  - Integrity
  - Maintainability
- **Means**
  - Fault prevention
  - Fault tolerance
  - Fault removal
  - Fault forecasting
- **Models**
  - Reliability Block Diagram
  - Fault Tree model
  - Reliability Graph

The ability of a system to deliver service that can justifiably be trusted.

Software Testing



Software Reliability

Software Reliability

Concept

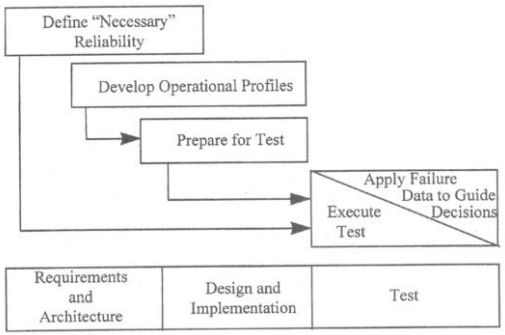
- Reliability
- Availability
- Failure rate
- MTTF
- Failure density
- Etc.

Model

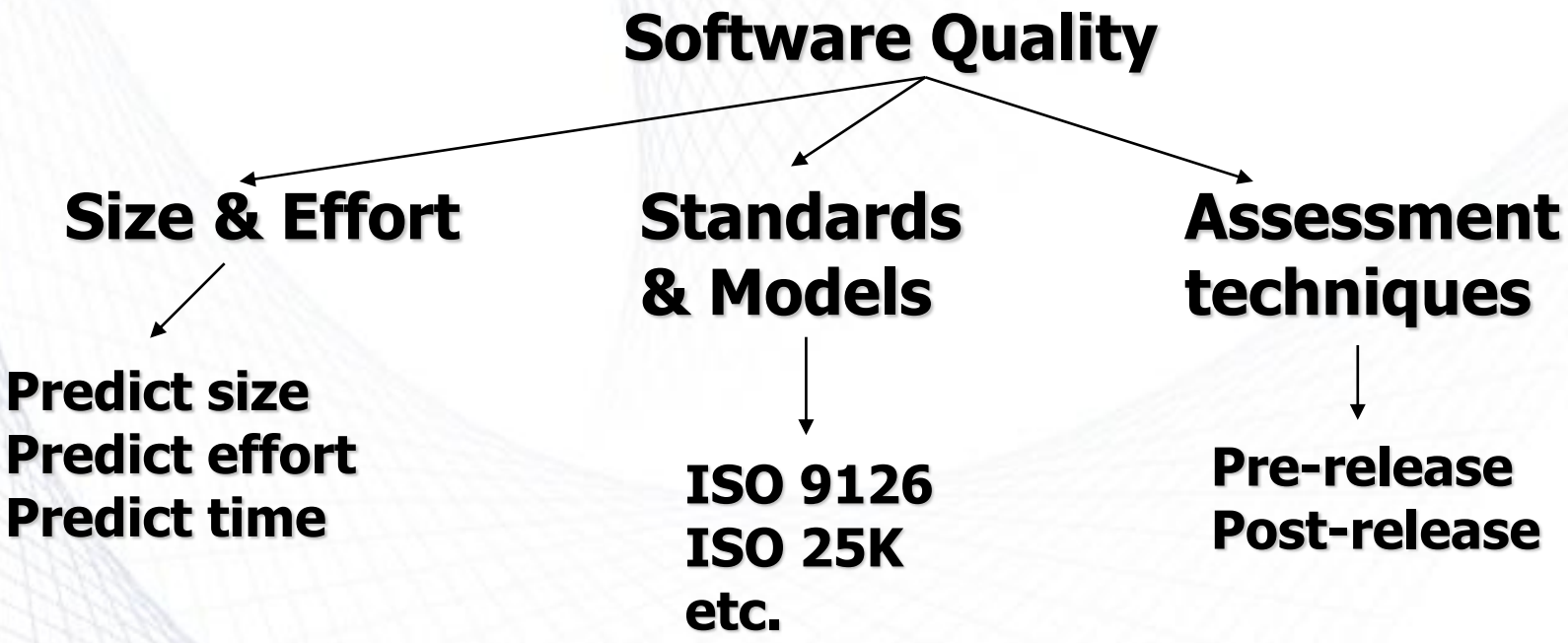
- Single Failure Model
- Multiple Failure Model
  - Reliability Growth Model

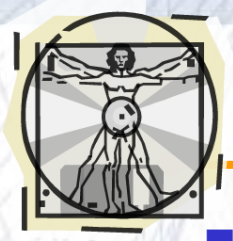
Process

- SRE Process



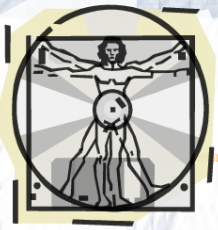
Software Quality





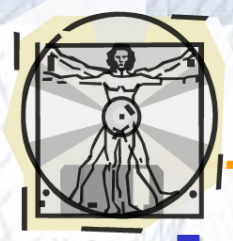
Definition: Service

- The service delivered by a system is its behaviour as it is perceived by its users; a user is another system (physical, human) that interacts with the former at the service interface.
- The function of a system is what the system is intended to do, and is described by the functional specification.
- Correct service is delivered when the service implements the system function.
- The delivery of incorrect service is a system outage.
- A transition from incorrect service to correct service is service restoration.



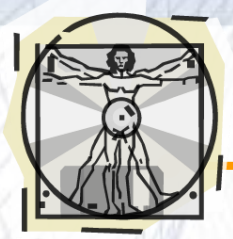
Dependability: Threats

- An error is a human action that results in software containing a fault.
- A fault (bug) is a cause for either a failure of the program or an internal error (e.g., an incorrect state, incorrect timing). It must be detected and removed.
- Among the 3 factors only failure is observable.



Definition: Failure

- **Failure:**
  - A system failure is an event that occurs when the delivered service deviates from correct service. A failure is thus a transition from correct service to incorrect service, i.e., to not implementing the system function.
  - Any departure of system behavior in execution from user needs. A failure is caused by a fault and the cause of a fault is usually a human error.
- **Failure Mode:**
  - The manner in which a fault occurs, i.e., the way in which the element faults.
- **Failure Effect:**
  - The consequence(s) of a failure mode on an operation, function, status of a system/process/activity/environment. The undesirable outcome of a fault of a system element in a particular mode. ← **associated Risk**



Failure Intensity & Density

- **Failure Intensity (failure rate):** the rate failures are happening, i.e., number of failures per natural or time unit. Failure intensity is way of expressing system reliability, e.g., 5 failures per hour; 2 failures per 1000 transactions.

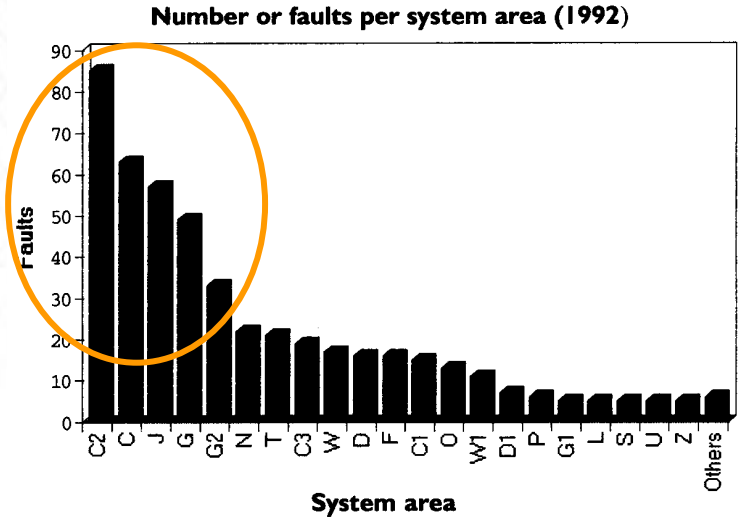
For system end users

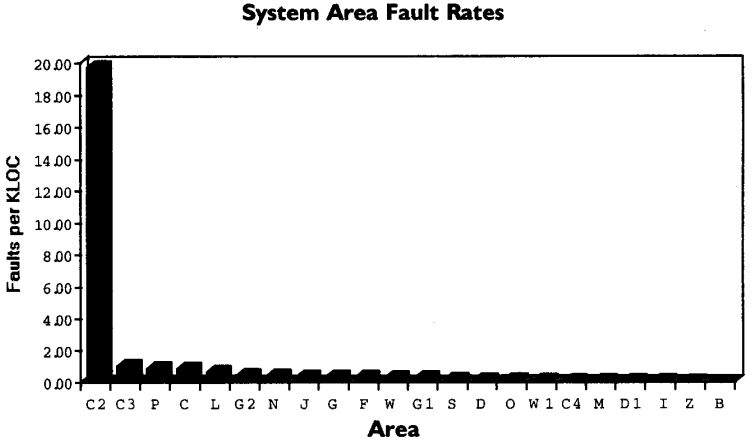
- **Failure Density:** failure per KLOC (or per FP) of developed code, e.g., 1 failure per KLOC, 0.2 failure per FP, etc.

For system developers

Example: Failure Density

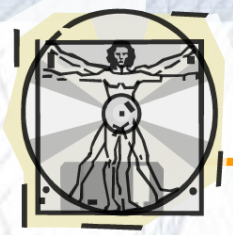
- In a software system, measuring number of failures lead to identification of 5 modules.
- However, measuring failures per KLOC (Failure Density) leads to identification of only one module.





Example from Fenton's Book

Failure Density vs. Inspection Effort



- Is failure density a good metrics for software quality?
- The more bugs found and fixed doesn't necessarily imply better software quality because the fault injection rate and the effort to fix them may be different.

		Inspection Effort	
		Higher	Lower
Failure Density	Higher	Good/ Not bad	Worst Case
	Lower	Best Case	Unsure

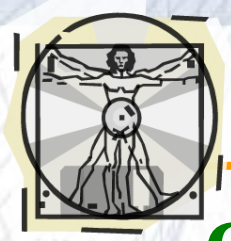
Testing vs. Inspection

Inspections are strict and close examinations conducted on specifications, design, code, test, and other artifacts.

- Inspections allow for defect detection, prevention, and isolation
- Start early in life cycle

- Testing allows for defect detection
- Start later in life cycle

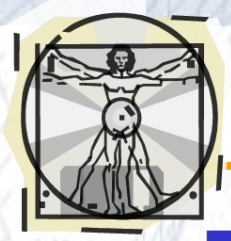
- Inspections are up to 20 times more efficient than testing
- Code reading detects twice as many defects/hour as testing
- 80% of development errors are usually found by inspections
- Inspections resulted in a 10x reduction in cost of finding errors



Inspections or Testing?

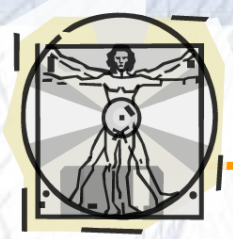
Q. *Can inspection replace testing?*

- Basically no. Inspections could replace testing if and only if all information gleaned through testing could be obtained through inspection. ← usually impossible
  - Complex interactions in large systems
  - Software reliability indicator
  - Nonfunctional requirements: performance, usability, etc.



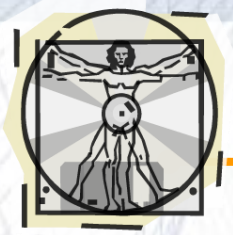
Definition: Fault

- **Fault:** A fault is a cause for either a failure of the program or an internal error (e.g., an incorrect state, incorrect timing)
  - A fault must be detected and then removed
  - Fault can be removed without execution (e.g., code inspection, design review)
  - Fault removal due to execution depends on the occurrence of associated "failure".
  - Occurrence depends on length of execution time and operational profile.
- **Defect:** refers to either fault (cause) or failure (effect)



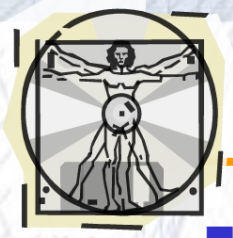
Definition: Error

- Error has two meanings:
  - A discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition.
  - A human action that results in software containing a fault.
- Human errors are the hardest to detect.



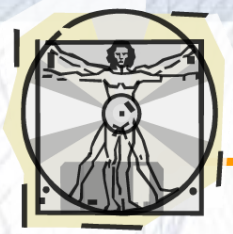
Dependability: Attributes /1

- **Availability:** readiness for correct service
- **Reliability:** continuity of correct service
- **Safety:** absence of catastrophic consequences on the users and the environment
- **Confidentiality:** absence of unauthorized disclosure of information
- **Integrity:** absence of improper system state alterations
- **Maintainability:** ability to undergo repairs and modifications



Dependability: Attributes /2

- Dependability attributes may be emphasized to a greater or lesser extent depending on the application: availability is always required, whereas reliability, confidentiality, safety may or may not be required.
- Other dependability attributes can be defined as combinations or specializations of the six basic attributes.
- **Example:** Security is the concurrent existence of
  - Availability for authorized users only;
  - Confidentiality; and
  - Integrity with improper taken as meaning unauthorized.

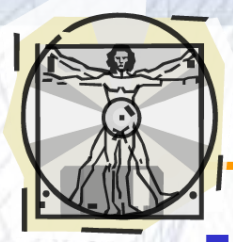


Definition: Availability

- **Availability:** a measure of the delivery of correct service with respect to the alternation of correct and incorrect service

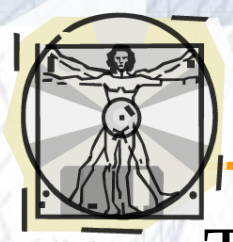
$$Availability = \frac{Uptime}{Uptime + Downtine}$$

$$Availability = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF}$$



Definition: Reliability /1

- Reliability is a measure of the continuous delivery of correct service
- Reliability is the probability that a system or a capability of a system functions without failure for a "specified time" or "number of natural units" in a specified environment. (Musa, et al.) *Given that the system was functioning properly at the beginning of the time period*
- Probability of failure-free operation for a specified *time* in a specified *environment* for a given *purpose* (Sommerville)
- A recent survey of software consumers revealed that reliability was the most important quality attribute of the application software



Definition: Reliability /2

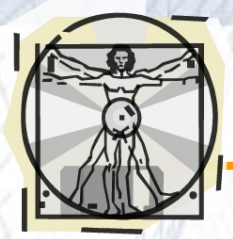
Three key points:

- Reliability depends on how the software is used

  Therefore a ***model of usage*** is required
- Reliability can be improved over time if certain bugs are fixed (reliability growth)

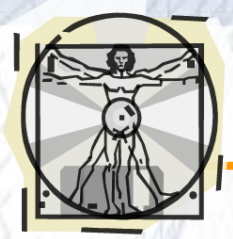
  Therefore a ***trend model*** (aggregation or regression) is needed
- Failures may happen at random time

  Therefore a ***probabilistic model of failure*** is needed



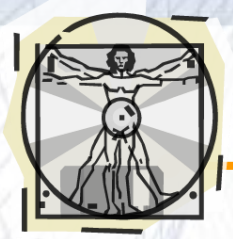
Definition: Safety

- **Safety:** absence of catastrophic consequences on the users and the environment
- Safety is an extension of reliability: safety is reliability with respect to catastrophic failures.
- When the state of correct service and the states of incorrect service due to non-catastrophic failure are grouped into a safe state (in the sense of being free from catastrophic damage, not from danger), safety is a measure of continuous safeness, or equivalently, of the time to catastrophic failure.



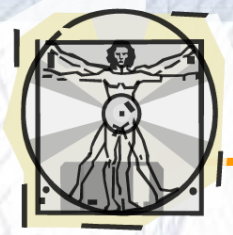
Definition: Confidentiality

- **Confidentiality:** absence of unauthorized disclosure of information



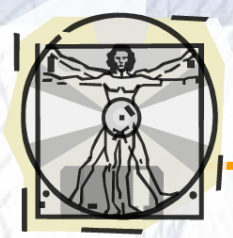
Definition: Integrity

- **Integrity:** absence of improper system state alterations



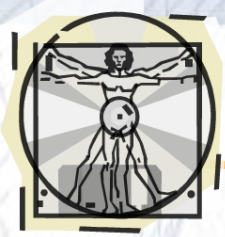
Definition: Maintainability

- **Maintainability:** ability to undergo repairs and modifications
- Maintainability is a measure of the time to service restoration since the last failure occurrence, or equivalently, measure of the continuous delivery of incorrect service.



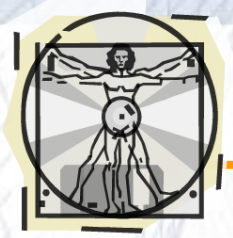
Dependability: Means

- **Fault prevention:** how to prevent the occurrence or introduction of faults
- **Fault tolerance:** how to deliver correct service in the presence of faults
- **Fault removal:** how to reduce the number or severity of faults
- **Fault forecasting:** how to estimate the present number, the future incidence, and the likely consequences of faults



Definition: Fault Prevention

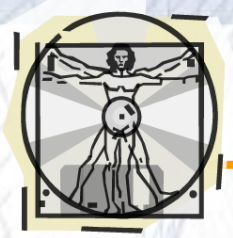
- To avoid fault occurrences by construction.
- Fault prevention is attained by quality control techniques employed during the design and manufacturing of software.
- Fault prevention intends to prevent ***operational physical*** faults.
- **Example techniques:** design review, modularization, consistency checking, structured programming, etc.



Fault Prevention

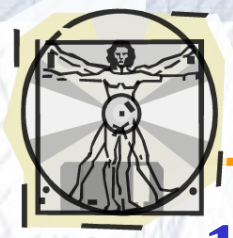
- Activities:
  - Requirement review
  - Design review
  - Clear code
  - Establishing standards (ISO 9000-3, etc.)
  - Using CASE tools with built-in check mechanisms

All these activities are included in ISO 9000-3: Guidelines for application of ISO 9001 to the development, supply and maintenance of software (1991)



Definition: Fault Tolerance

- A fault-tolerant computing system is capable of providing specified services in the presence of a bounded number of failures
- Use of techniques to enable continued delivery of service during system operation
- It is generally implemented by ***error detection*** and subsequent ***system recovery***
- Based on the principle of:
  - *Act during operation* while
  - *Defined during specification and design*



Fault Tolerance Process

1. Detection

- Identify faults and their causes (errors)

2. Assessment

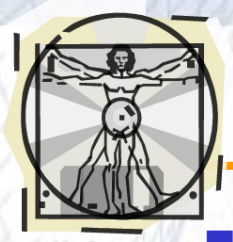
- Assess the extent to which the system state has been damaged or corrupted.

3. Recovery

- Remain operational or regain operational status

4. Fault treatment and continued service

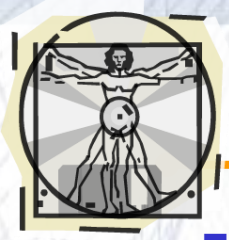
- Locate and repair the fault to prevent another occurence



Definition: Fault Removal /1

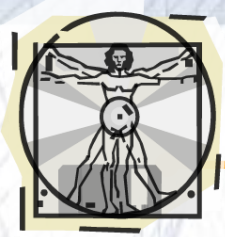
- Fault removal is performed both during the development phase, and during the operational life of a system.
- Fault removal during the development phase of a system life-cycle consists of three steps:

  verification → diagnosis → correction
- *Verification* is the process of checking whether the system adheres to given properties, called the *verification conditions*. If it does not, the other two steps follow: *diagnosing* the faults that prevented the verification conditions from being fulfilled, and then performing the necessary *corrections*.



Definition: Fault Removal /2

- After correction, the verification process should be repeated in order to check that fault removal had no undesired consequences; the verification performed at this stage is usually called non-regression verification.
- Checking the specification is usually referred to as ***validation***.
- Uncovering specification faults can happen at any stage of the development, either during the specification phase itself, or during subsequent phases when evidence is found that the system will not implement its function, or that the implementation cannot be achieved in a cost effective way.



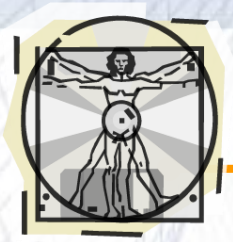
Definition: Fault Forecasting

- Fault forecasting is conducted by performing an evaluation of the system behaviour with respect to fault occurrence or activation
- Evaluation has two aspects:
  - qualitative, or ordinal, evaluation, which aims to identify, classify, rank the failure modes, or the event combinations (component failures or environmental conditions) that would lead to system failures
  - quantitative, or probabilistic, evaluation, which aims to evaluate in terms of probabilities the extent to which some of the attributes of dependability are satisfied; those attributes are then viewed as measures of dependability

UNIVERSITY OF CALGARY

Section 3

What is Coming Next?



Chapter 1

- Introduction & terminology
- **Reference:**
  - Slides ch.1
  - Chapter 1 of the Muthur's book
- **Contents:**
  - Introducing the course
  - Terminology
  - What is software quality?

Chapter 2

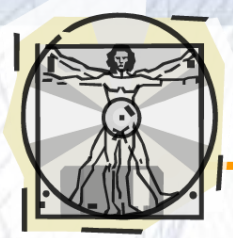
- Foundations of Testing
- **Reference:**
  - Slides ch.2
- **Contents:**
  - Software testing concepts
  - Software Testing Life Cycle (STLC)
  - Requirement Traceability Matrix (RTM)
  - Manual software testing
    - Exploratory testing
  - Bug report
  - Manual regression testing

Chapter 3

- Criteria-Based Test Design Unit Testing
- **Reference:**
  - Slides ch.3
- **Contents:**
  - Introduction to criteria-based test design
  - Unit testing
  - Automated test execution
    - JUnit
  - Dependencies
    - Stubbing – mocking

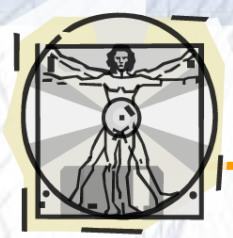
Chapter 4

- Black-box Testing
- **Reference:**
  - Slides Ch.4
  - Chapter 2 of the Muthur's book
- **Contents:**
  - Equivalence class partitioning
  - Boundary value analysis
  - Decision tables
  - Combinatorial testing
  - Profile based testing



Chapter 5A

- White-box Testing Control-flow Coverage
- **Reference:**
  - Slides ch.5A
  - Chapter 2 of the Muthur's book
- **Contents:**
  - Control flow
  - Statement/Node/Line coverage
  - Decision/Edge/Branch coverage
  - Condition coverage
  - Path coverage
  - Cyclomatic complexity



Chapter 5B

- White-box Testing data-flow Coverage
- **Reference:**
  - Slides ch.5B
  - Mathur's book: Section 6.3. Data-Flow Concepts
- **Contents:**
  - Data flow coverage

Chapter 6

- Mutation Testing
- **Reference:**
  - Slides ch.6
  - Chapters 6-7 of the Muthur's book
- **Contents:**
  - Mutation testing technique
  - Mutation testing tools

Chapter 7

- GUI testing and test automation
- **Reference:**
  - Slides ch.7
- **Contents:**
  - Test automation concept and techniques
  - GUI testing tools
  - Web testing tools

Chapter 8

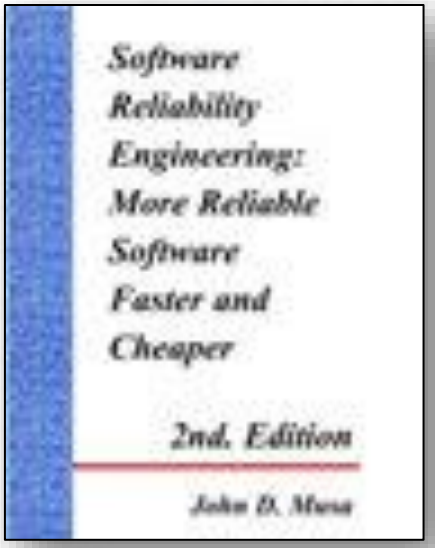
- Quality and Reliability engineering concepts and process
- **Reference:**
  - Slides ch.8
- **Contents:**
  - How to define quality for a system
  - How to measure quality
  - Engineering reliability in software development process

Chapter 9

- System reliability
- **Reference:**
  - Slides ch.9
- **Contents:**
  - Reliability Block Diagram (RBD)
  - Serial and parallel configuration
  - Hazard analysis: Fault tree analysis (FTA)

Chapter 10

- Reliability assessment tools and techniques
- **Reference:**
  - Slides ch.10
- **Contents:**
  - Reliability assessment tools
  - System reliability assessment
  - Assessment criteria:
    - Certification testing using RDC
    - Reliability growth
    - Zero failure testing

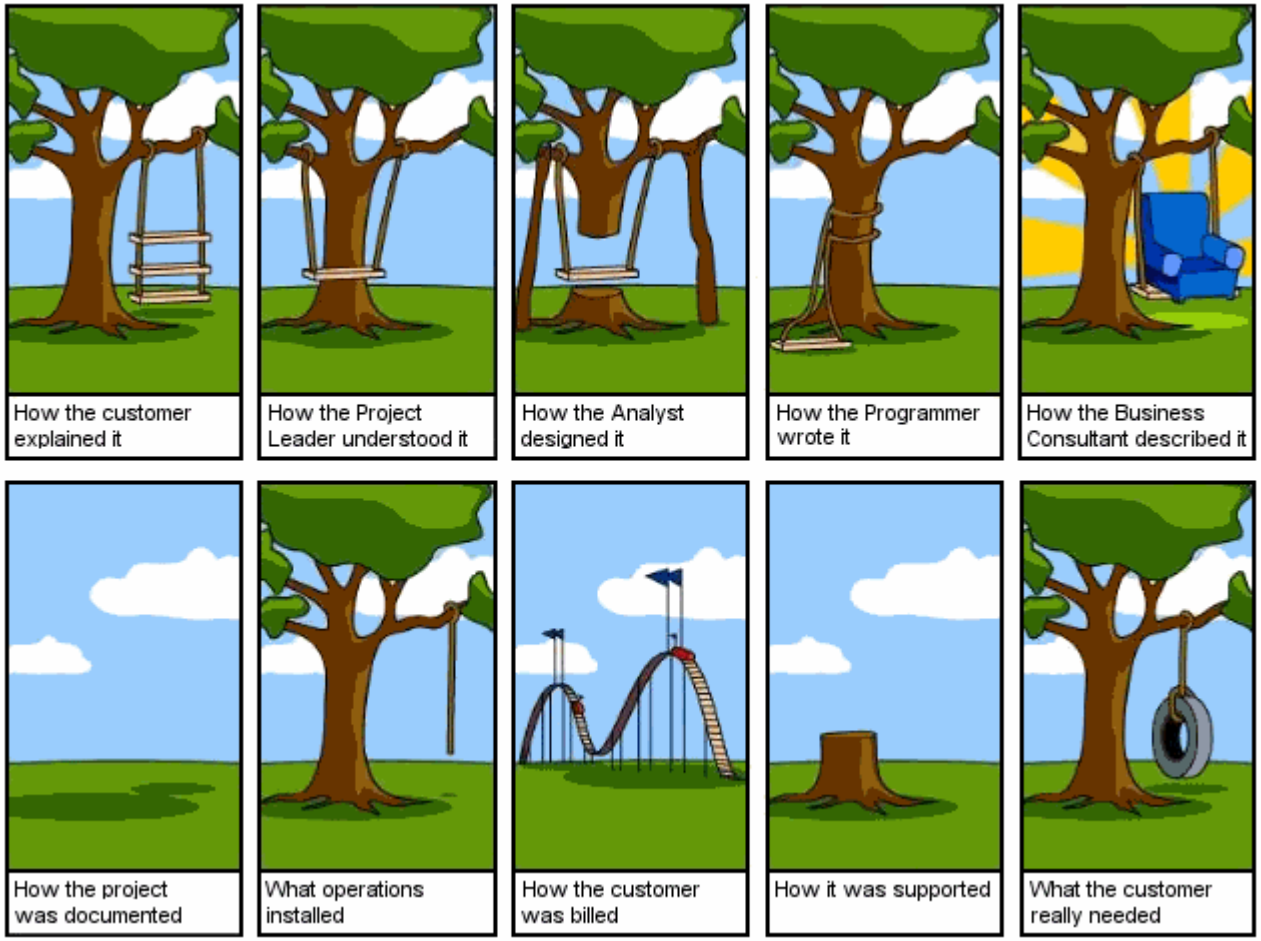


Chapter 11

- Integration, system and acceptance testing
- **Reference:**
  - Slides ch.11
- **Contents:**
  - Integration testing concept and techniques
  - System testing concept and techniques
  - Acceptance testing concept and techniques

Chapter 12

- Post-release activities
- **Reference:**
  - Slides ch.12
- **Contents:**
  - SQS, SQA, SRE processes and activities
  - Quality, test and data plan
  - Post release activities


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed