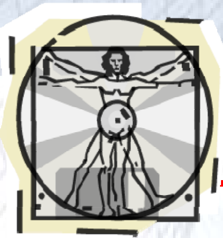
SENG 637
Dependability, Reliability & Testing of Software Systems

Chapter 1: Overview

Department of Electrical & Computer Engineering, University of Calgary

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

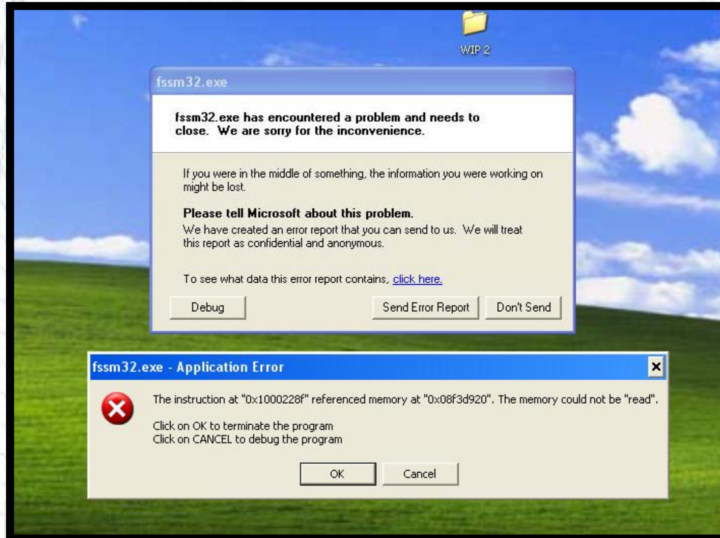
http://www.enel.ucalgary.ca/People/far/Lectures/SENG637/

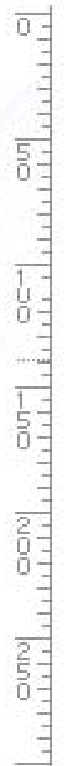


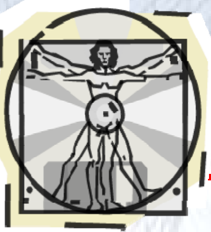
Contents

Shorter version:

- How to avoid these?



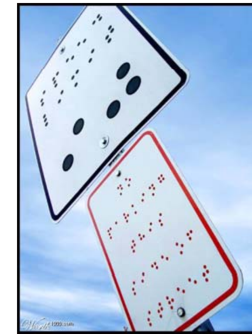


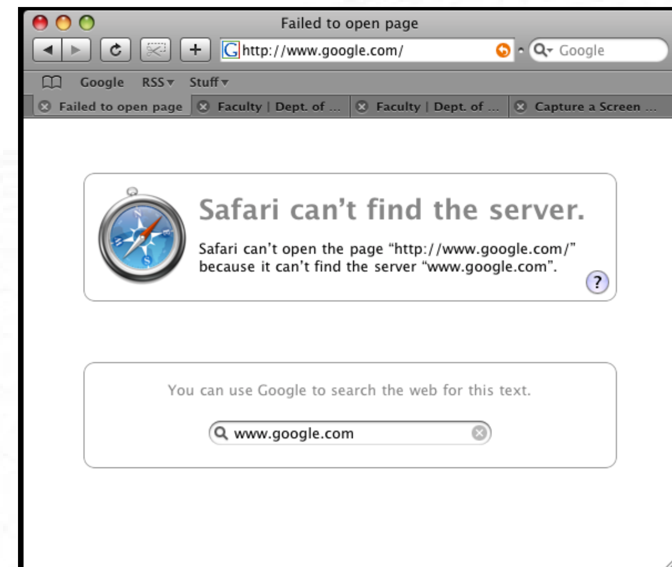


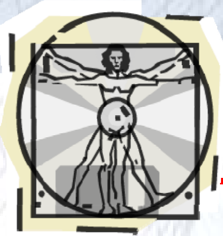
Contents

Longer version:

- **What is this course about?**
- **What factors affect software quality?**
- **What is software reliability?**
- **What is software reliability engineering?**
- **What is software reliability engineering process?**

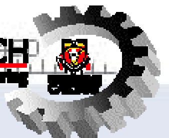


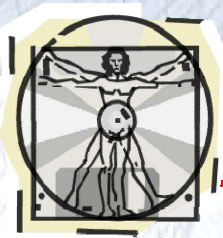




About This Course …

- The topics discussed include:
  - Concepts and relationships
  - Analytical models and supporting tools
  - Techniques for software reliability improvement, including:
    - Fault avoidance, fault elimination, fault tolerance
    - Error detection and repair
    - Failure detection and retraction
    - Risk management

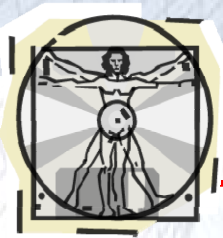




Terminology & Scope

- Dependability
  - Treats
    - Failures
    - Faults
    - Errors
  - 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

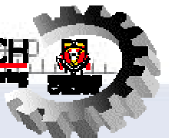
The ability of a system to deliver service that can justifiably be trusted.

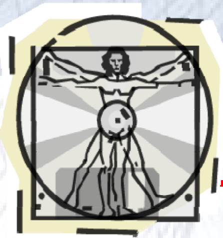


Software Reliability

Software Reliability

- Concept
  - Reliability
  - Availability
  - Failure rate
  - MTTF
  - Failure density
  - Etc.
- Model
  - Single Failure Model
  - Multiple Failure Model
    - Reliability Growth Model
- Process
  - SRE Process





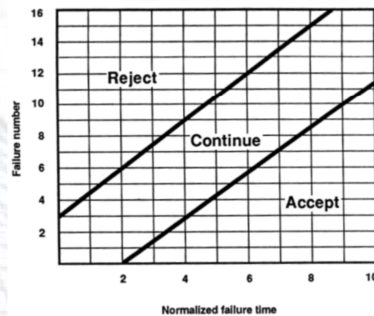
Software Testing

Software Testing

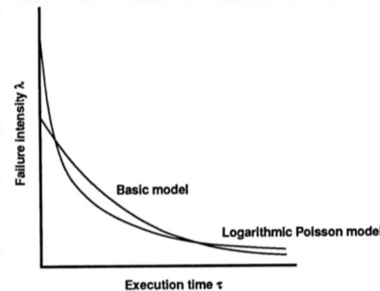
Techniques

Other:
- Black-box
- White-box
- Alpha
- Beta
- Big-bang
- Stress
- Etc.

Certification Test

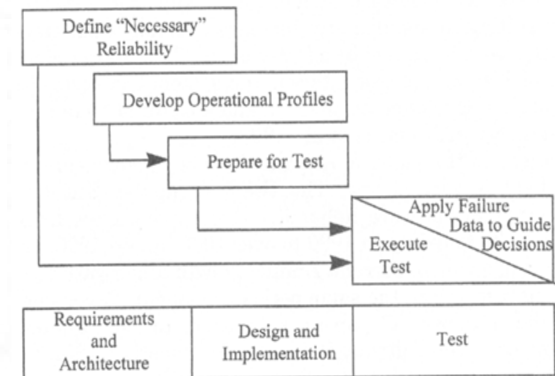


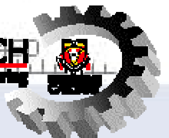
Reliability Growth Test

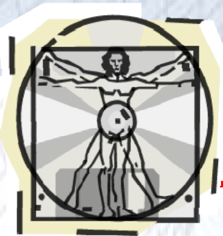


Process

SRE Process

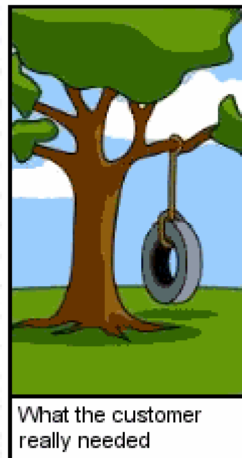


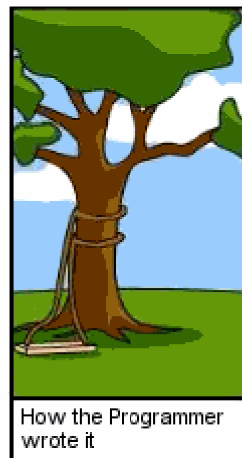


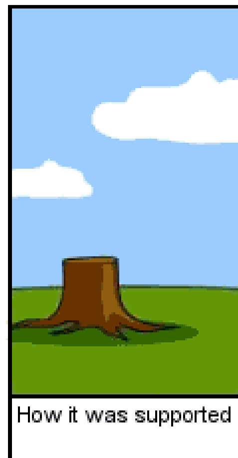


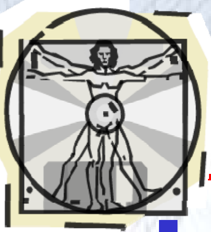
Question to Ask

- 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 software designer, tester and quality controller:



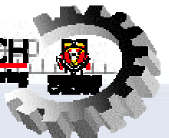






At The End …

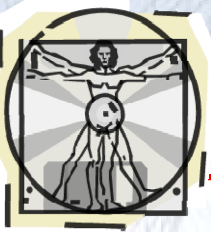
- What is software reliability engineering (SRE)?
- Why SRE is important? How does it affect software quality?
- What are the main factors that affect the reliability of software?
- Is SRE equivalent to software testing? What makes SRE different from software testing?
- How can one determine how often will the software fail?
- How can one determine the current reliability of the software under development?
- How can one determine whether the product is reliable enough to be released?
- Can SRE methodology be applied to the current ways of software development such as component-based and agile development?
- What are challenges and difficulties of applying SRE?
- What are current research topics of SRE?



UNIVERSITY OF CALGARY

Chapter 1 Section 1

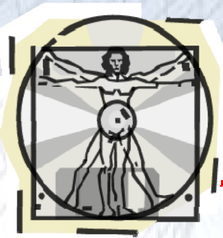
From Software Quality to Software Reliability Engineering



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)



Quality: Various Views

HCI

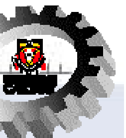
Aesthetic View

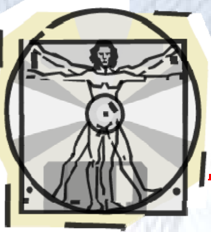
Developer View

SQA

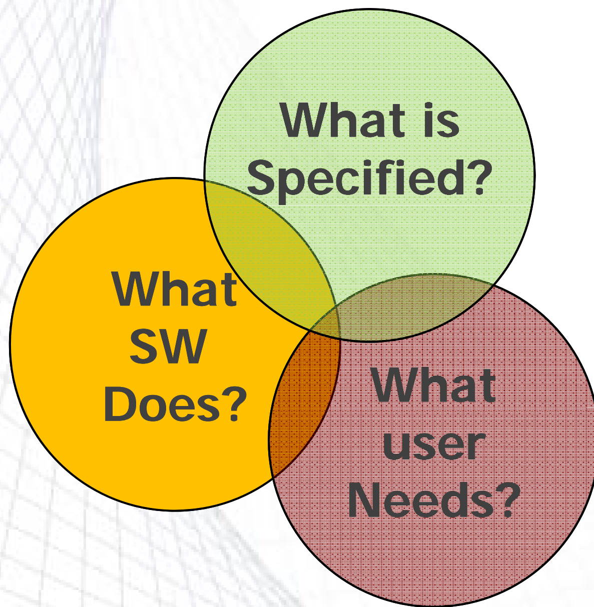
Customer View

SRE



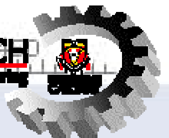


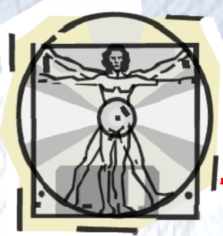
What is Software Quality?



- **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

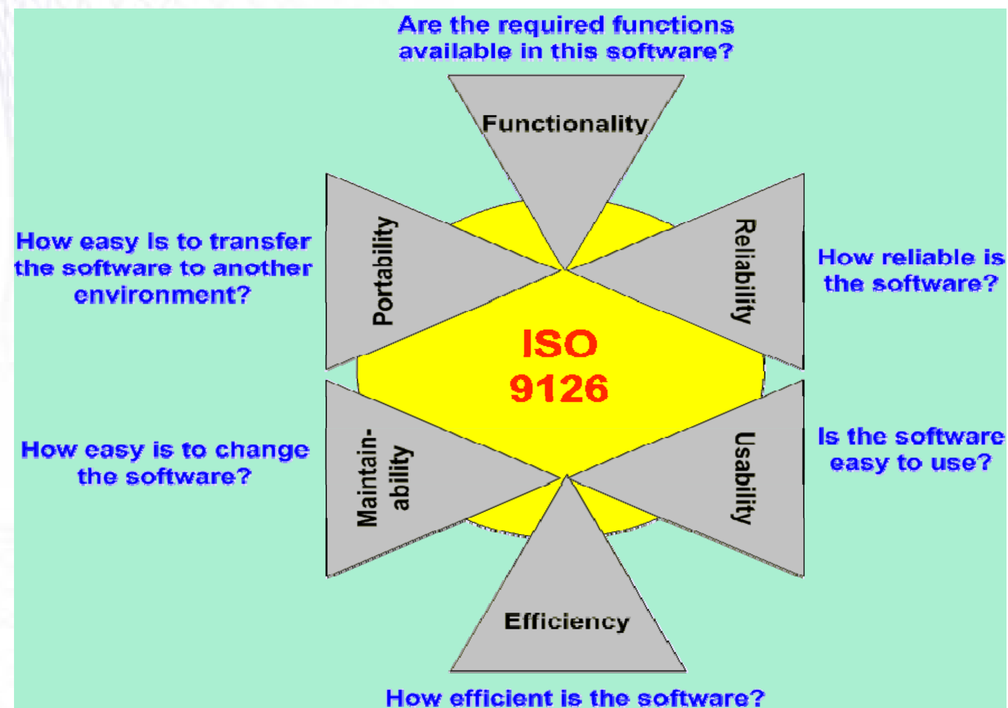




Definition: Software Quality

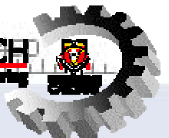
ISO 8402 definition of QUALITY:

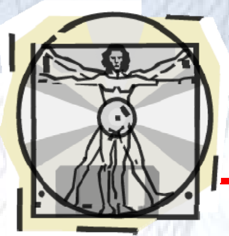
The totality of features and characteristics of a product or a service that bear on its ability to satisfy stated or implied needs



Reliability **Maintainability**

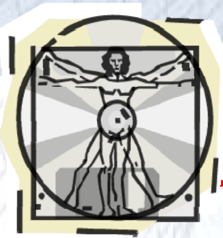
two major components of Quality





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		



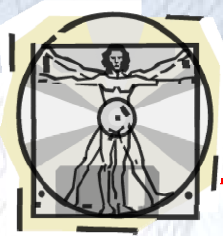
Quality Model – Structure

User oriented

Software oriented

- SW Quality
  - Quality Factor 1
  - Quality Factor 2
  - ...
  - Quality Factor n
- Quality Criterion 1
- Quality Criterion 2
- Quality Criterion 3
- ...
- Quality Criterion m
- Measures

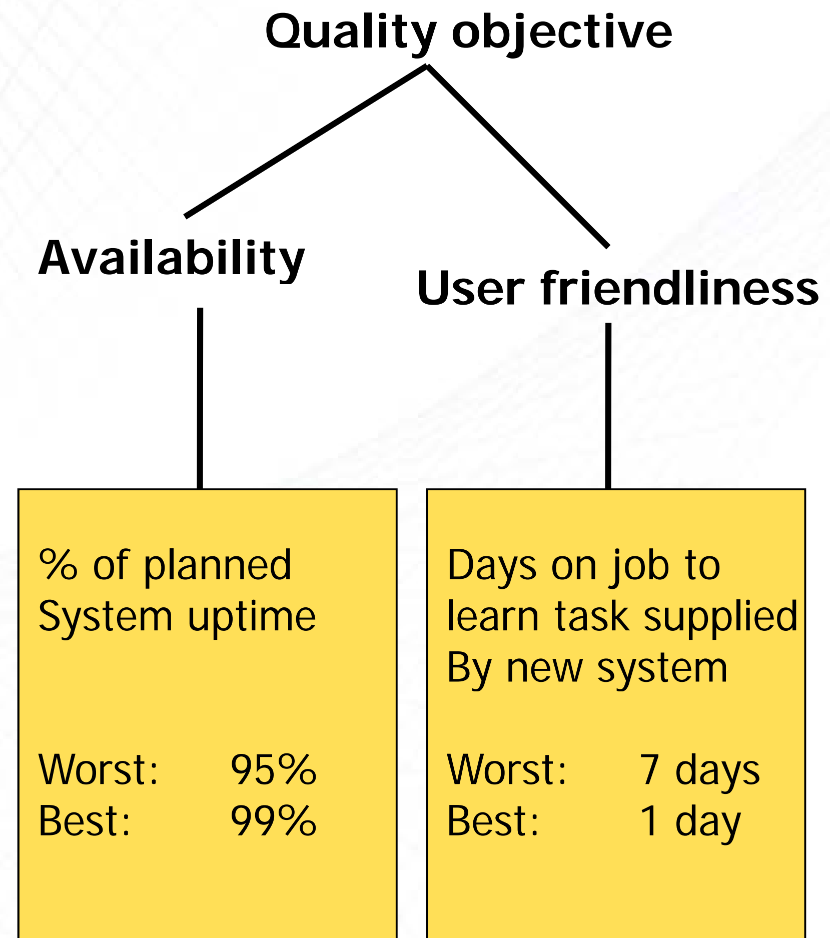
far@ucalgary.ca

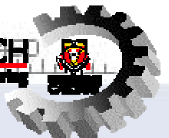


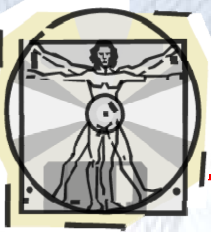
Example: Attribute Expansion

- **Design by measurable objectives:**

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







What Affects Quality?

Cost

Quality

Time

We offer three kinds of service:

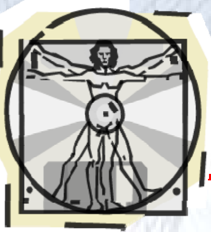
GOOD - CHEAP - FAST

You can pick any two

GOOD service CHEAP won't be FAST

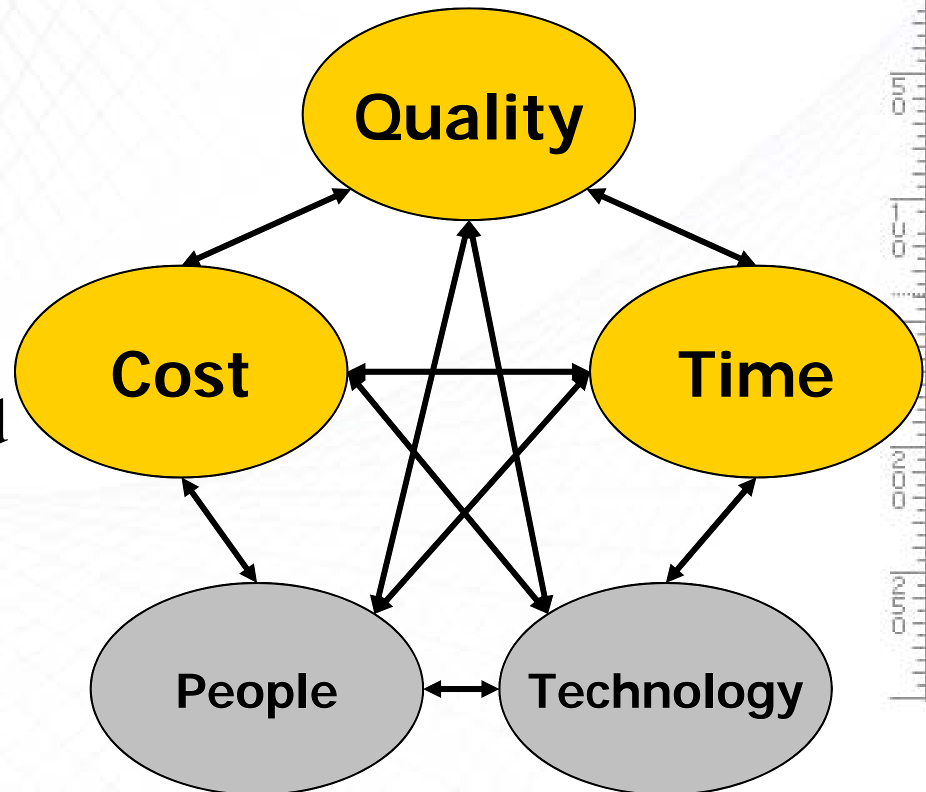
GOOD service FAST won't be CHEAP

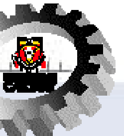
FAST service CHEAP won't be GOOD

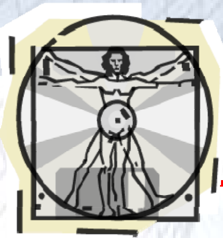


What Affects Software Quality?

- **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.

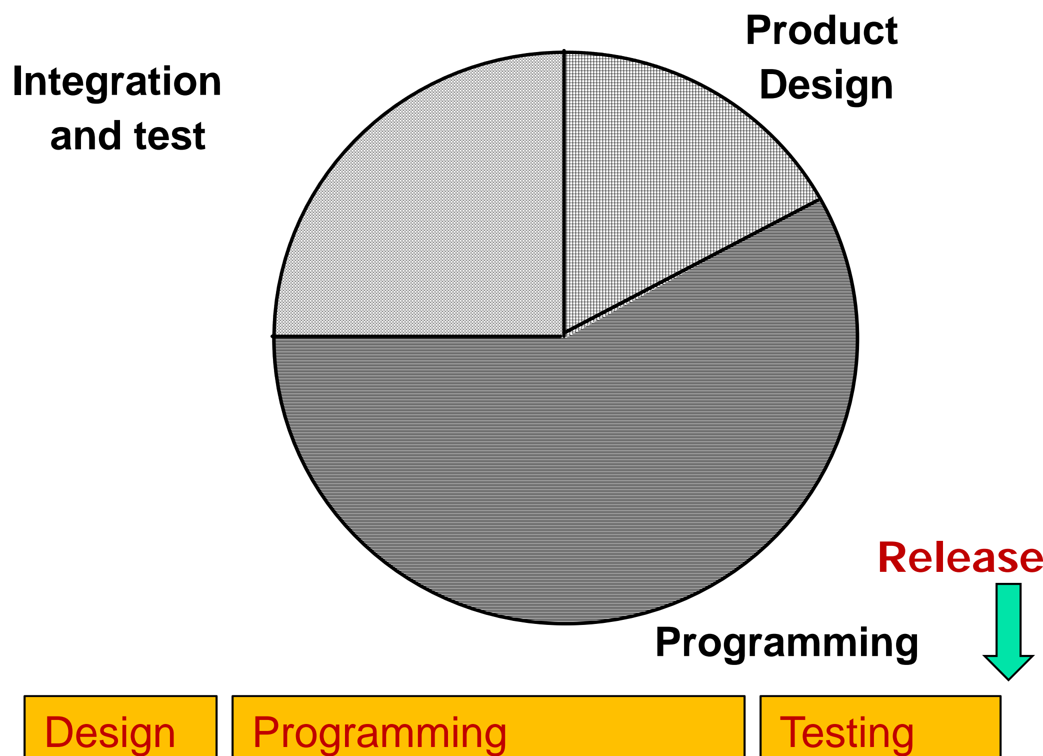






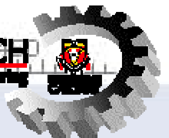
Quality vs. Project Costs

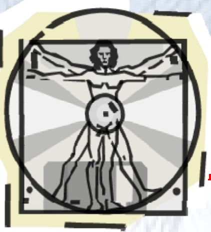
Cost distribution for a typical software project



Design	Programming	Testing

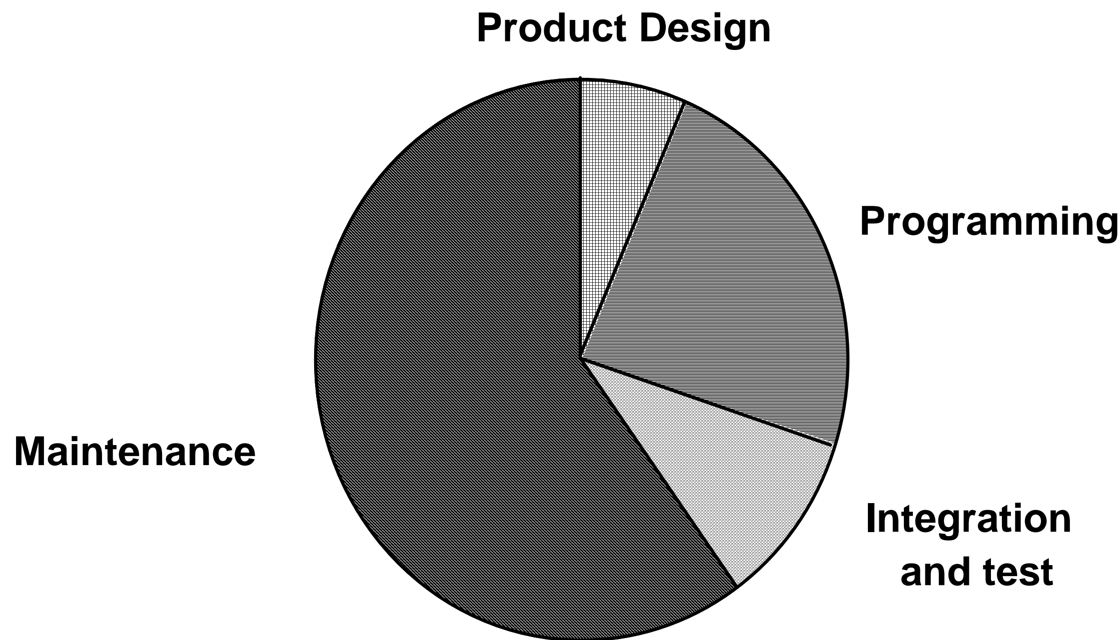
What is wrong with this picture?





Total Cost Distribution

Maintenance is responsible for more that 60% of total cost for a typical software project

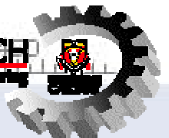


Questions:

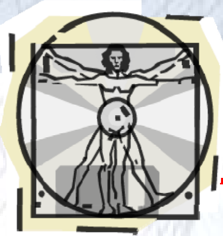
1) How to build quality into a system?

2) How to assess quality of a system?

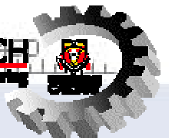
Developing better quality system will contribute to lowering maintenance costs

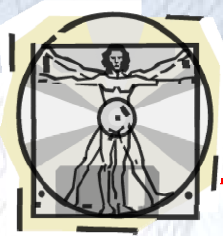


1) How to Build Quality into a System?



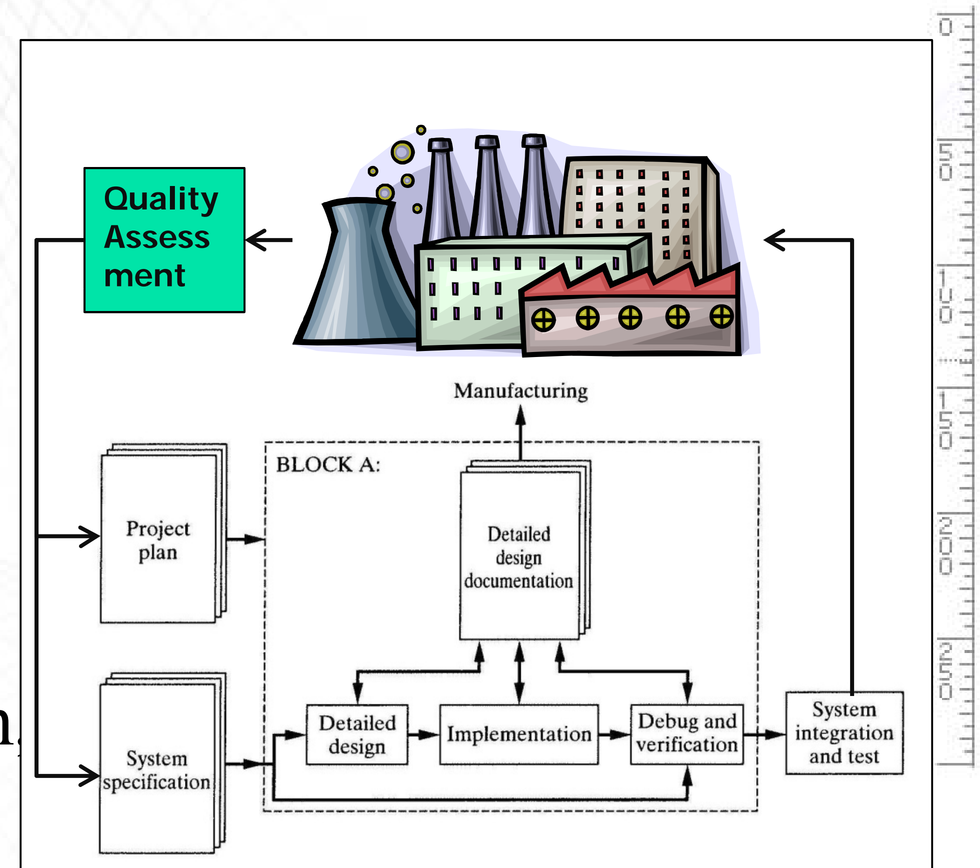
- Developing better quality systems requires:
  - Establishing **Quality Assurance (QA)** programs
  - Establishing **Reliability Engineering (SRE)** process

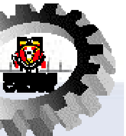


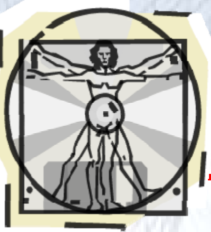


2) How to Assess Quality of a System?

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



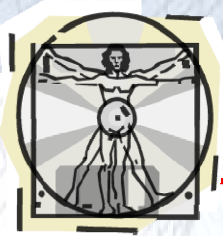




How Do We Assess Quality?

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

Our focus in this course

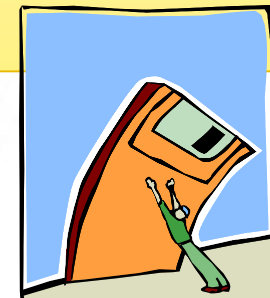


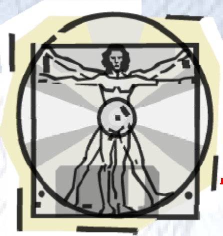
Pre-release Quality

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

Facts:

- **About 20% of the software projects are canceled. (missed schedules, etc.)**
- **About 84% of software projects are incomplete when released (need patch, etc).**
- **Almost all of the software projects costs exceed initial estimations. (cost overrun)**

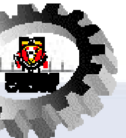


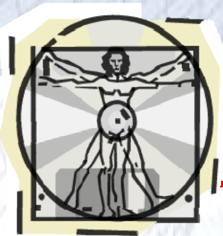


Fatal Software Examples

Fatal software related incidents [Gage & McCormick 2004]

Date	Casualties	Detail
2003	3	Software failure contributes of power outage across North-eastern U.S. and Canada.
2001	5	Panamanian cancer patients die following overdoses of radiation, determined by the use of faulty software.
2000	4	Crash of marine corps osprey tilt-rotor aircraft, partially blamed on software anomaly.
1997	225	Radar that could have prevented Korean jet crash hobbled by software problem.
1995	159	American airlines jet, descending into Cali, Columbia crashes into a mountain. A cause was that the software presented insufficient and conflicting information to the pilots, who got lost.
1991	28	Software problem prevents Patriot missile battery from picking up SCUD missile, which hits US Army barracks in Saudi Arabia.



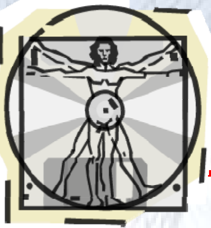


Cost of a Defect ...

	Require-ments	Design	Coding	Functional Test	System Test	Field Use
Fault Origin	10 %	40 %	50 %			
Fault Detection	3 %	5 %	7 %	25 %	50 %	10 %
Cost per Fault	1 KDM	1 KDM	1 KDM	6 KDM	12 KDM	20 KDM

1 KDM = 1,000 Deutsch Marks

CMU. Software Engineering Institute



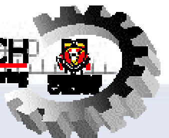
A Central Question

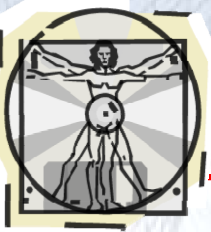
- In spite of having many development methodologies, central questions are:

1. Can we remove all bugs before release?

2. How often will the software fail?

These are exactly the questions that we are going to answer in this course!





Two Extremes

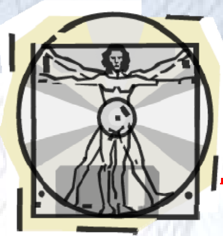
- Craftsman SE: fast, cheap, buggy
- Cleanroom SE: slow, expensive, zero defect
- ***Is there a middle solution?***

Craftsman Software Develop-ment

YES!

Using Software Reliability Engineering (SRE) Process

Cleanroom Software Develop-ment

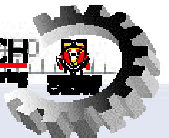


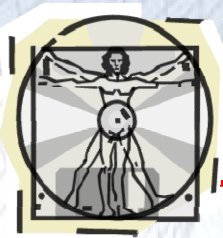
Can We Remove All Bugs?

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**

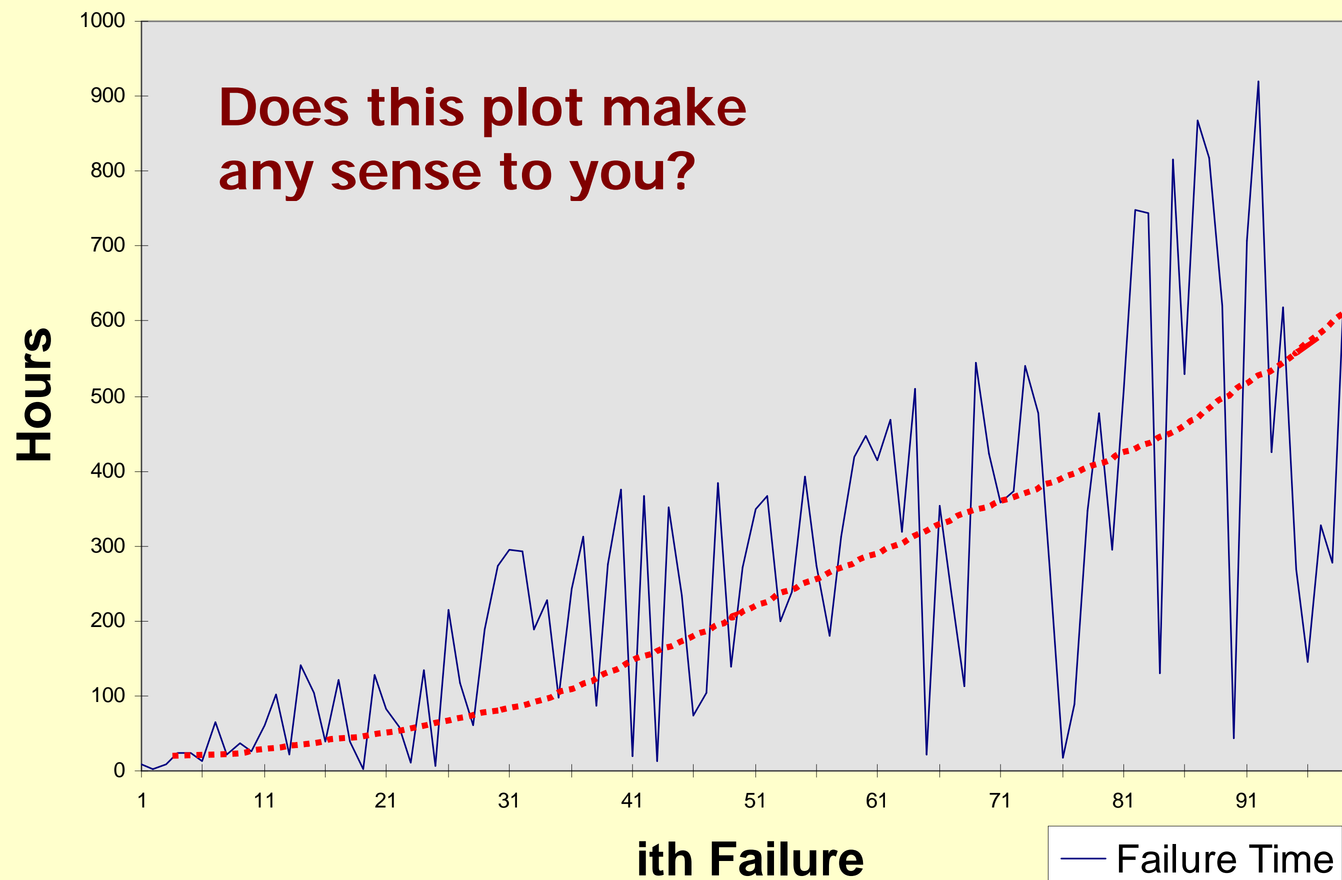
Defect potential and density are expressed in terms of defects per function point

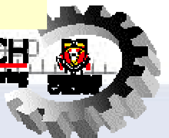
The answer is usually NO!

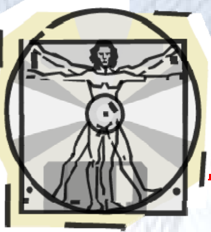




What Can We Learn from Failures?





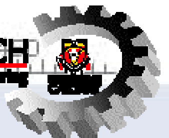


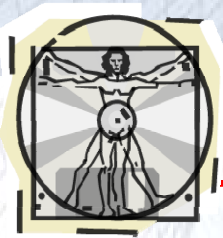
How to Handle Defects?

- 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?

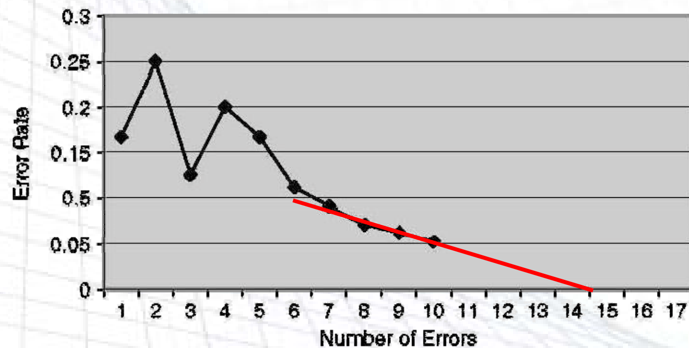


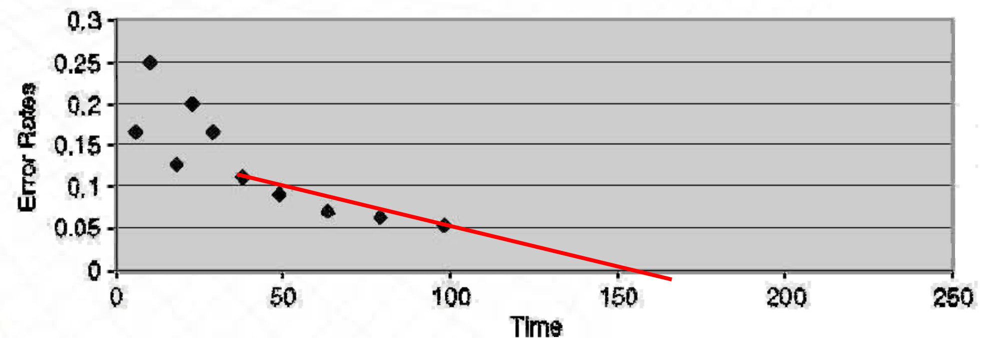


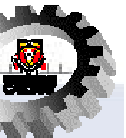
What to Learn from Data?

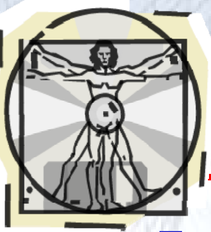
- The inverses of the inter-failure times are the ***failure intensity*** (= failure per unit of time) data points

Error 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*









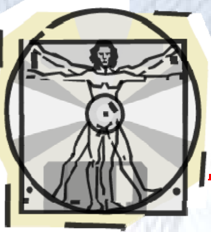
What to Learn from Data?

- Mean-time-to-failures MTTF (or 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

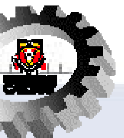
$$R = e^{-\lambda t} = e^{-t/MTTF} = e^{-1/9.8} = 0.90299$$

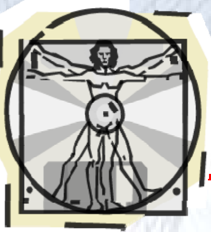
- Fitting a straight line to the graph in (a) would show an x-intercept of about 15. Using this as an estimate of the total number of original failures, we estimate that there are still five bugs in the software.
- Fitting a straight line to the graph in (b) would give an x-intercept near 160. This would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



A Typical Problem: Question

- Failure intensity (failure rate) of a system is usually expressed using FIT (Failure-In-Time) unit which is 1 failure per 10**9 device hours.
- Failure intensity of an electric pump system used for pumping crude oil in Northern Alberta's oil field is constant and is 10,000 FITs and 100 such pumps are operational.
- If for continuous operation all failed units are to be replaced immediately, what shall be the minimum inventory size of pumps for one year of operation?





A Typical Problem: Answer

Pump's Mean-Time-To-Failure (MTTF)

λ = 10,000 FITs = 10,000 / 10**9 hour = 1×10**-5 hour

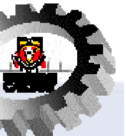
= 1 failure per 100,000 hours

The 12-month reliability is: (1 year = 8,760 hours)

R(8,760 hours) = exp{-8,760/100,000} = 0.916 and "unreliability" is,

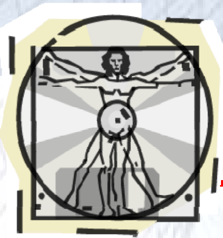
F(8,760) = 1 - 0.916 = 0.084

Therefore, inventory size is 8.4% or minimum 9 pumps should be at stock in the first year.



Chapter 1 Section 2

Definitions

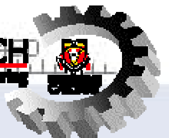


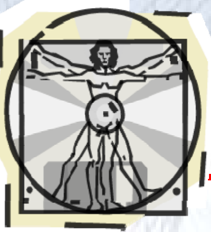
Terminology

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.

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

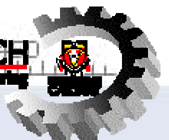
- **Dependability**
  - **Treats**
    - Failures
    - Faults
    - Errors
  - **Attributes**
    - Availability
    - Reliability
    - Safety
    - Confidentiality
    - Integrity
    - Maintainability
  - **Means**
    - Fault prevention
    - Fault tolerance
    - Fault removal
    - Fault forecasting
  - **Models**
    - Reliability Block Diagram
    - Fault Tree model
    - Reliability Graph

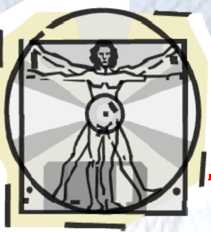




Dependability: Treats

- 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.

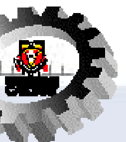


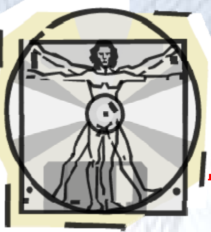


Definition: Failure

- **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.

Not all failures are caused by a bug





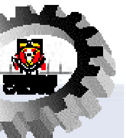
Failure Intensity & Density

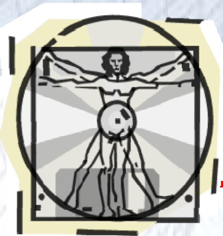
- **Failure Intensity (failure rate):** the rate failures are happening, i.e., number of failures per natural or time unit. Failure intensity is way of expressing system reliability, e.g., 5 failures per hour; 2 failures per 1000 transactions.

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.

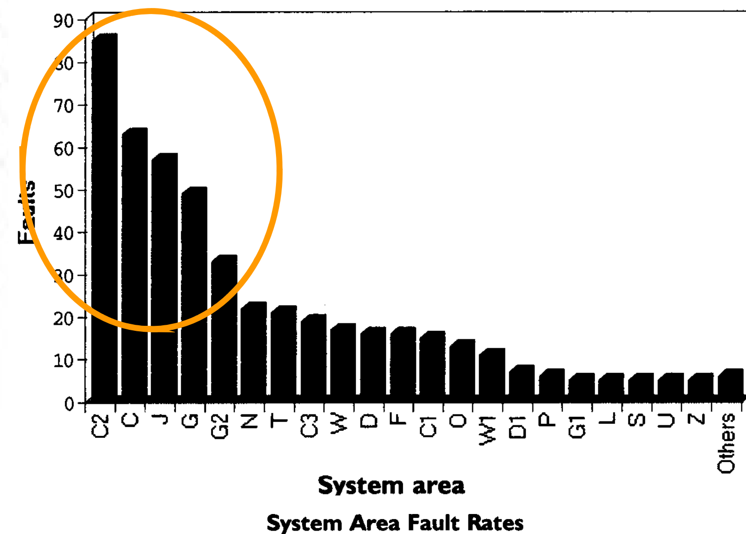
For system developers

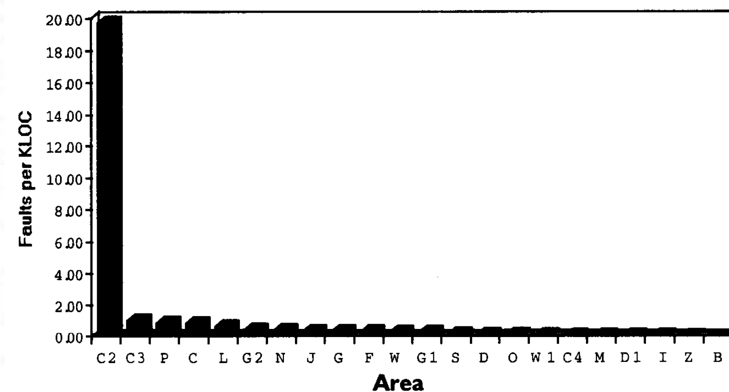




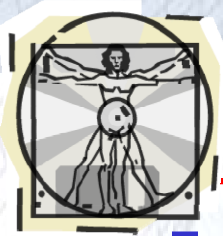
Example: Failure Density

- 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.



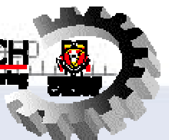


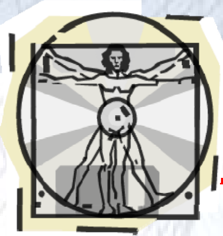
Example from Fenton's Book



Definition: Fault

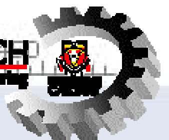
- **Fault:** A fault is a cause for either a failure of the program or an internal error (e.g., an incorrect state, incorrect timing)
  - A fault must be detected and then removed
  - Fault can be removed without execution (e.g., code inspection, design review)
  - Fault removal due to execution depends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

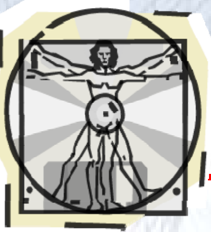




Definition: Error

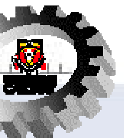
- Error has two meanings:
  - A discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition.
  - A human action that results in software containing a fault.
- Human errors are the hardest to detect.

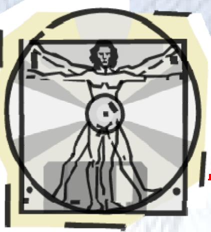




Dependability: Attributes /1

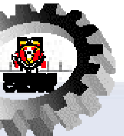
- **Availability:** readiness for correct service
- **Reliability:** continuity of correct service
- **Safety:** absence of catastrophic consequences on the users and the environment
- **Confidentiality:** absence of unauthorized disclosure of information
- **Integrity:** absence of improper system state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

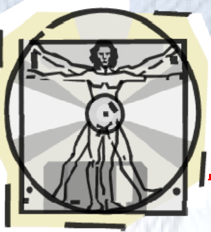




Dependability: Attributes /2

- Dependability attributes may be emphasized to a greater or lesser extent depending on the application: availability is always required, whereas confidentiality or 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.



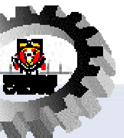


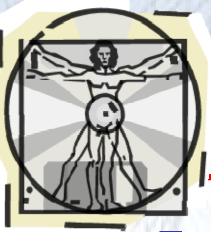
Definition: Availability

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

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

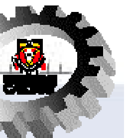
$$Availability = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF}$$

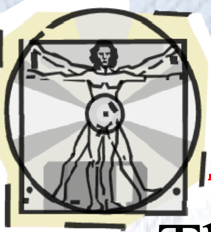




Definition: Reliability /1

- 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

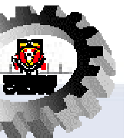
Three key points:

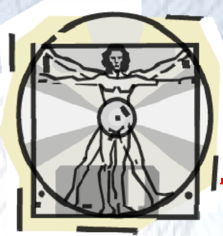
- Reliability depends on how the software is used

  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

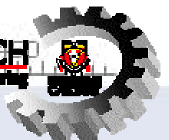
  Therefore a ***probabilistic model of failure*** is needed

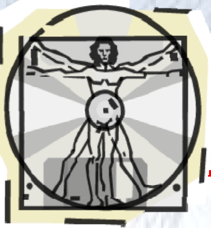




Definition: Safety

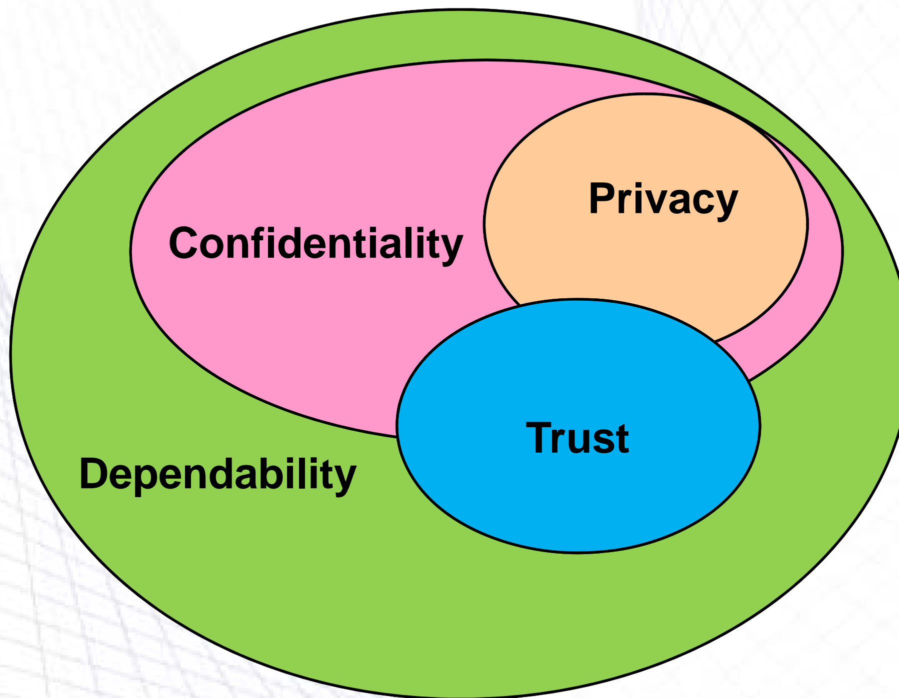
- **Safety:** absence of catastrophic consequences on the users and the environment
- Safety is an extension of reliability: safety is reliability with respect to catastrophic failures.
- When the state of correct service and the states of incorrect service due to non-catastrophic failure are grouped 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.





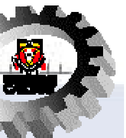
Definition: Confidentiality

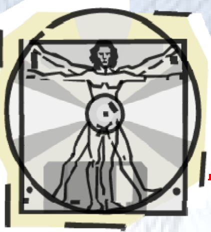
- **Confidentiality:** absence of unauthorized disclosure of information



Privacy: Preventing the release of unauthorized information about individuals considered sensitive

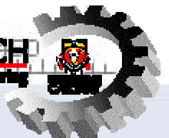
Trust: Confidence one has that an individual will give him/her correct information or an individual will protect sensitive information

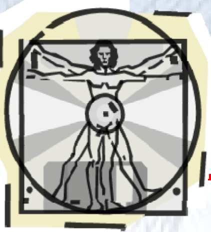




Definition: Integrity

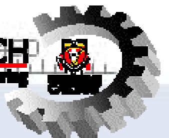
- **Integrity:** absence of improper system state alterations

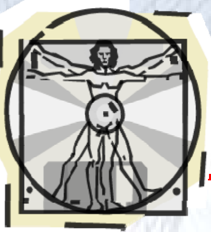




Definition: Maintainability

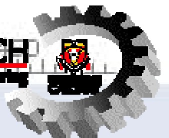
- **Maintainability:** ability to undergo repairs and modifications
- Maintainability is a measure of the time to service restoration since the last failure occurrence, or equivalently, measure of the continuous delivery of incorrect service.

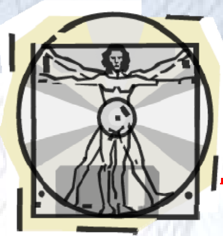




Dependability: Means

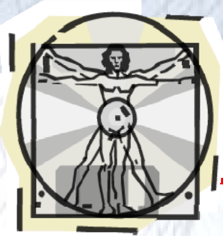
- **Fault prevention:** how to prevent the occurrence or introduction of faults
- **Fault tolerance:** how to deliver correct service in the presence of faults
- **Fault removal:** how to reduce the number or severity of faults
- **Fault forecasting:** how to estimate the present number, the future incidence,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





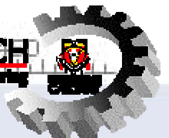
Definition: Fault Prevention

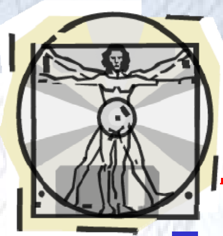
- To avoid fault occurrences by construction.
- Fault prevention is attained by quality control techniques employed during the design and manufacturing of software.
- Fault prevention intends to prevent ***operational physical*** faults.
- **Example techniques:** design review, modularization, consistency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



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***

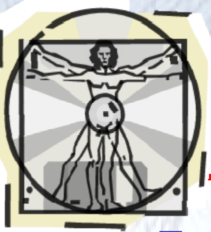




Definition: Fault Removal /1

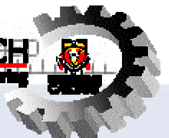
- Fault removal is performed both during the development phase, and during the operational life of a system.
- Fault removal during the development phase of a system life-cycle consists of three steps:

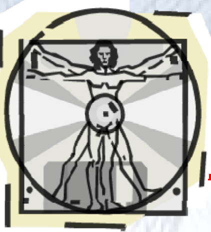
  verification → diagnosis → correction
- *Verification* is the process of checking whether the system adheres to given properties, called the *verification conditions*. If it does not, the other two steps follow: *diagnosing* the faults that prevented the verification conditions from being fulfilled,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



Definition: Fault Removal /2

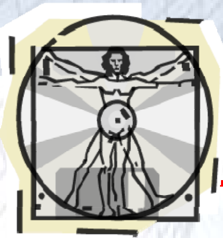
- After correction, the verification process should be repeated in order to check that fault removal had no undesired consequences; the verification performed at this stage is usually called non-regression verification.
- Checking the specification is usually referred to as ***validation***.
- Uncovering 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

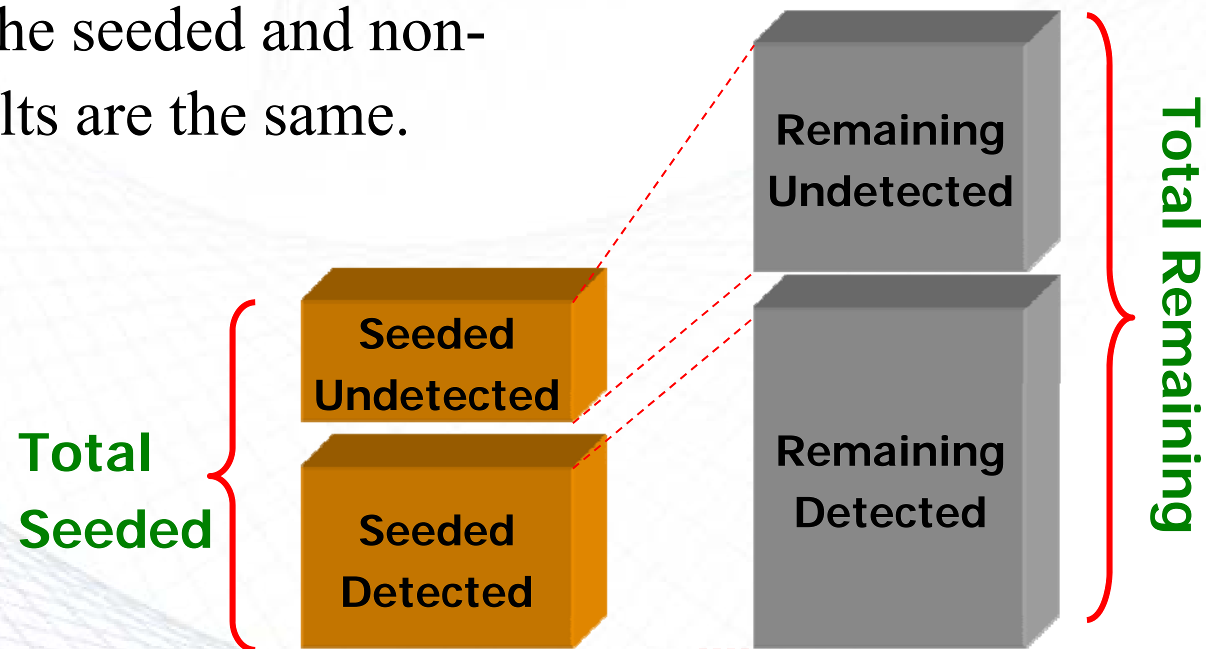
- Fault forecasting is conducted by performing an evaluation of the system behaviour with respect to fault occurrence or activation

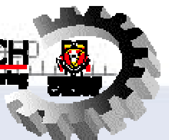


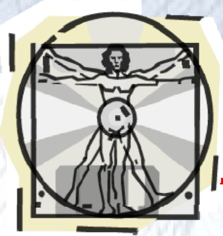
Fault Forecasting : How to /1

Q: How to determine number of remaining bugs?

The idea is to inject (seed) some faults in the program and calculate the remaining bugs based on detecting the seeded faults [Mills 1972]. Assuming that the probability of detecting the seeded and non-seeded faults are the same.







Fault Forecasting : How to /2

$$\frac{n_s}{N_s} = \frac{n_d}{N_d} \quad \text{or} \quad N_d = \frac{n_d}{n_s} \times N_s$$

n_s detected seeded faults

N_s total seeded faults

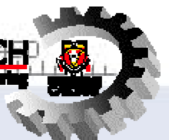
n_d detected remaining faults

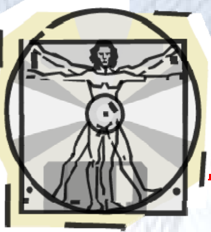
N_d total remaining faults

N_r undetected remaining faults

$$N_r = (N_d - n_d) + (N_s - n_s)$$

- The total injected faults (N_s) is already known; n_d and n_s are measured for a certain period of time.
- **Assumption:** all faults should have the same probability of being detected.





Example

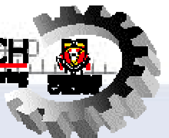
- Assume that

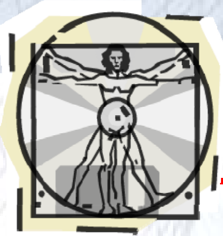
$$N_s = 20 \qquad n_s = 10 \qquad n_d = 50$$

$$N_d = \frac{n_d}{n_s} \times N_s = \frac{50}{10} \times 20 = 100$$

$$N_r = (N_d - n_d) + (N_s - n_s)$$

$$N_r = (100 - 50) + (20 - 10) = 60$$





Comparative Remaining Defects /1

- Two testing teams will be assigned to test the same product.

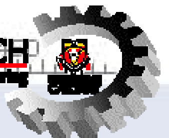
$$N_d = \frac{d_1 d_2}{d_{12}} \qquad N_r = N_d - (d_1 + d_2 - d_{12})$$

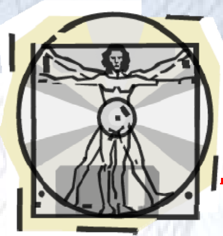
Defects detected by Team 1 : d_1 ; by Team 2 : d_2

Defects detected by both teams: d_{12}

N_d total remaining defects

N_r undetected remaining defects





Example

Defects detected

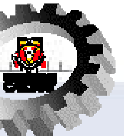
by Team 1 : $d_1 = 50$; by Team 2 : $d_2 = 40$

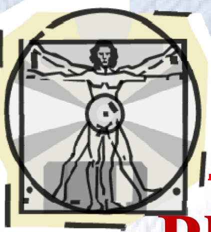
Defects detected by both teams: $d_{12} = 20$

$$N_d = \frac{d_1 d_2}{d_{12}} = \frac{50 \times 40}{20} = 100$$

$$N_r = N_d - (d_1 + d_2 - d_{12})$$

$$N_r = 100 - (50 + 40 - 20) = 30$$





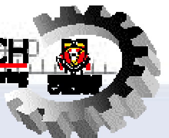
Fault Forecasting: PCE

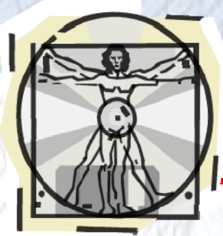
Phase containment effectiveness” (PCE)

- According to Dr. Stephen Kan the “phase containment effectiveness” (PCE) in the software development process is:

$$PCE = \frac{\text{Defects removed (at the step)} \times 100\%}{\text{Defects existing on step entry} + \text{Defects injected during the step}}$$

- Higher PCE is better because it indicates better response to the faults within the phase. A higher PCE means that less faults are pushed forward to later phases.





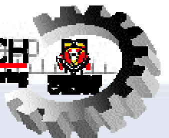
Example 2 (cont'd)

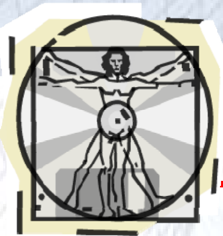
- Using the data from the table below, calculate the phase containment of the requirement, design and coding phases.

Phase	*Number of defects*		
	Introduced	***Found***	***Removed***
Requirements	12	9	9
Design	25	16	12
Coding	47	42	36

$$PCE_{req} = \frac{9 \times 100\%}{0 + 12} = \%75 \qquad PCE_{design} = \frac{12 \times 100\%}{3 + 25} = \%42.85$$

$$PCE_{coding} = \frac{36 \times 100\%}{(13+3) + 47} = \%57.14$$





Quality Models: CUPRIMDA

- Quality parameters (parameters for fitness):
  - **C**apability
  - **U**sability
  - **P**erformance
  - **R**eliability
  - **I**nstallability
  - **M**aintainability
  - **D**ocumentation
  - **A**vailability

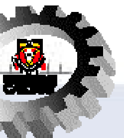
	CAPABILITY	USABILITY	PERFORMANCE	RELIABILITY	INSTALLABILITY	MAINTAINABILITY	DOCUMENTATION	AVAILABILITY
CAPABILITY								
USABILITY								
PERFORMANCE	●	●						
RELIABILITY	●	○	●					
INSTALLABILITY		○	○	○				
MAINTAINABILITY	●	○	●	○				
DOCUMENTATION	●	○				○		
AVAILABILITY	●	○		○	○			

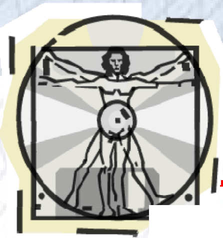
● CONFLICTIVE

○ SUPPORT ONE ANOTHER

BLANK NONE

Reference: S.H. Kan (1995)





Quality Models: Boehm's

Primary uses | **Intermediate constructs** | **Primitive constructs**

General utility

As is utility

Maintainability

Portability

Reliability

Efficiency

Human engineering

Testability

Understandability

Modifiability

Device Independence

Completeness

Accuracy

Consistency

Device efficiency

Acessibility

Communicativeness

Structuredness

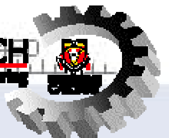
Self descriptiveness

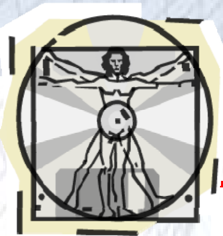
Conciseness

Legibility

Augmentability

Metrics

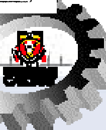


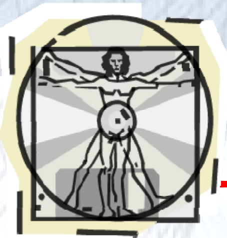


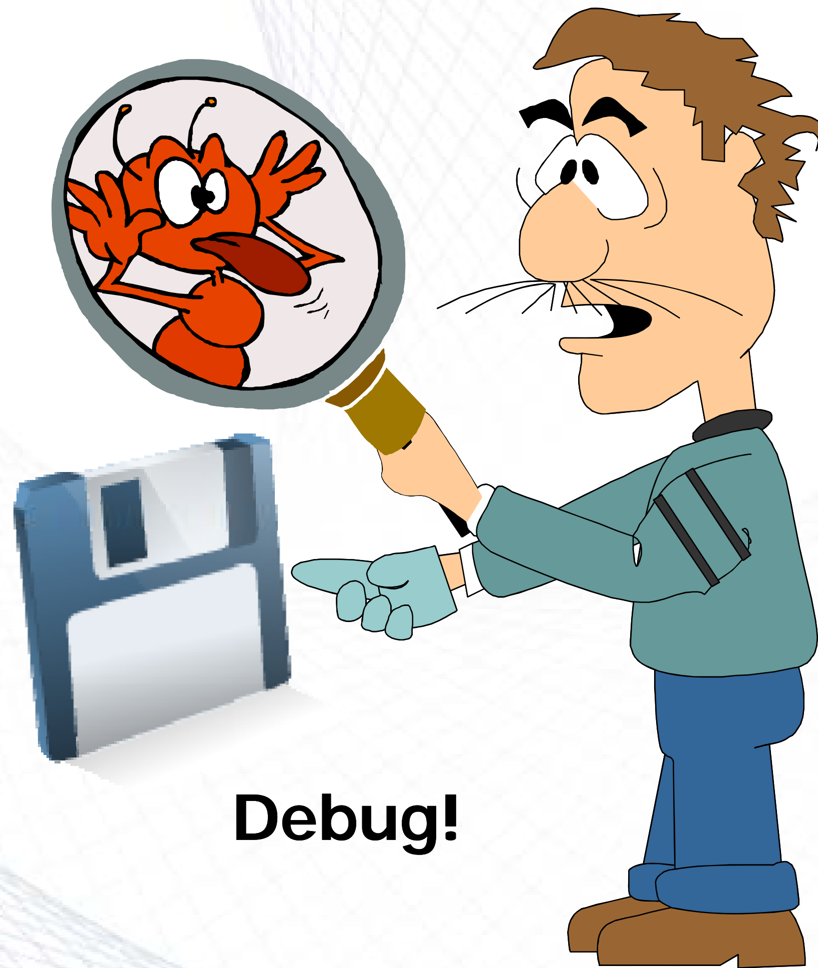
Quality Models: McCall's

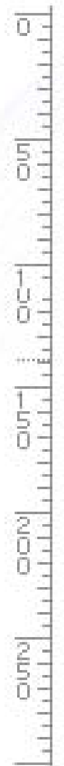
Use	Factor	Criteria
Product operation	Usability	Operability
	Integrity	Training
	Efficiency	Communicativeness
	Correctness	I/O volume
	Reliability	I/O rate
Product revision	Maintainability	Access control
	Testability	Access audit
	Flexibility	Storage efficiency
Product transition	Reusability	Execution efficiency
	Portability	Traceability
	Interoperability	Completeness
		Accuracy
		Error tolerance
		Consistency
		Simplicity
		Conciseness
		Instrumentation
		Expandability
		Generality
		Self-descriptiveness
		Modularity
		Machine independence
		S/w system independence
		Comms commonality
		Data commonality

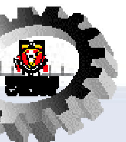
Metrics





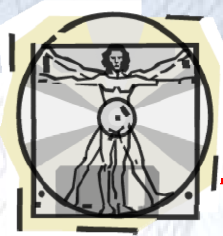

Debug!





Chapter 1 Section 3

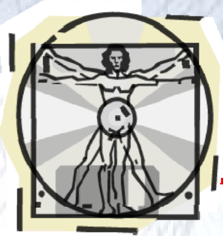
Software and Hardware Reliability



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.

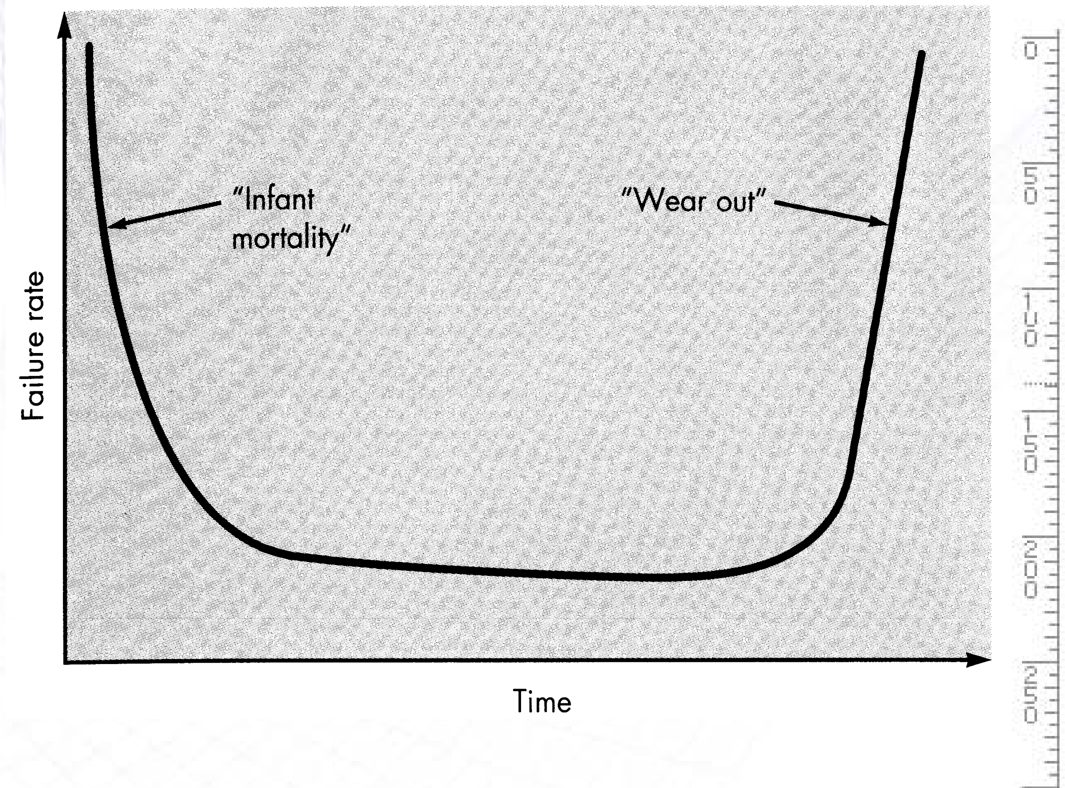
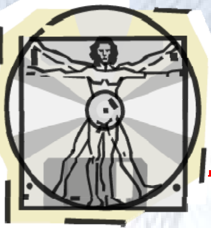


Figure from Pressman's book



Reliability: Hardware

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

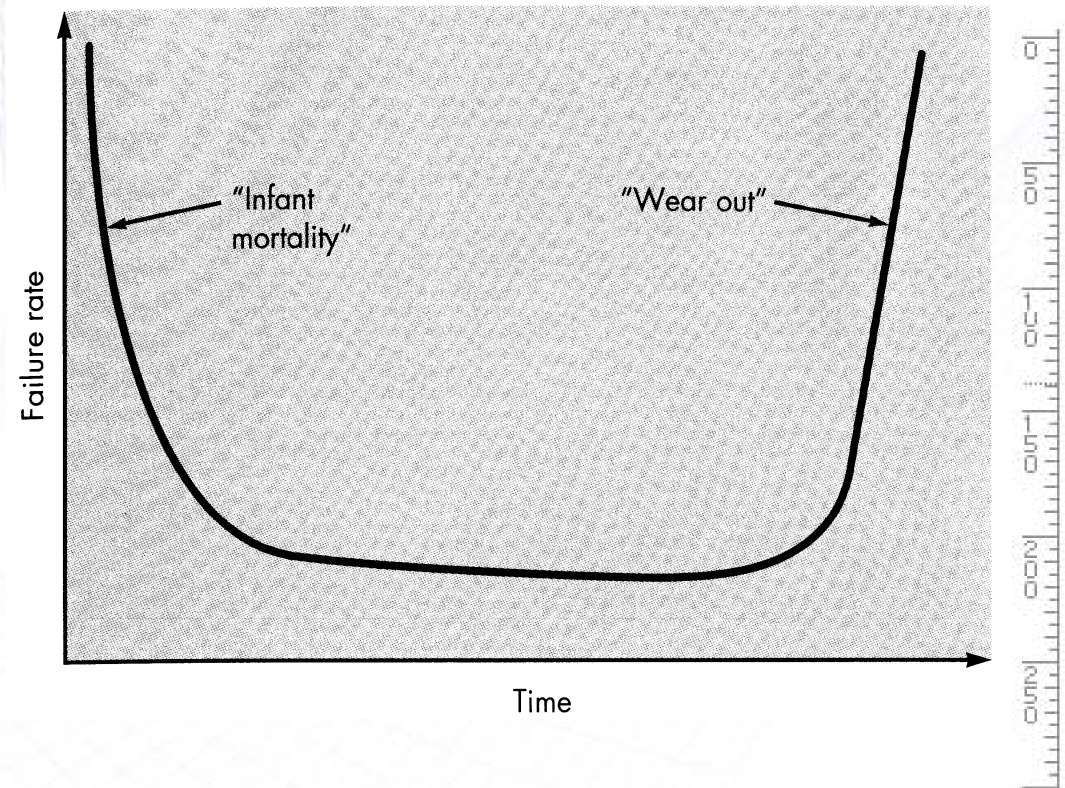
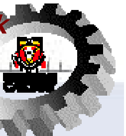
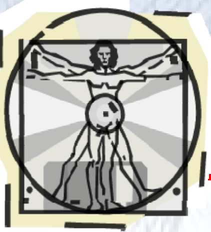


Figure from Pressman's book





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.

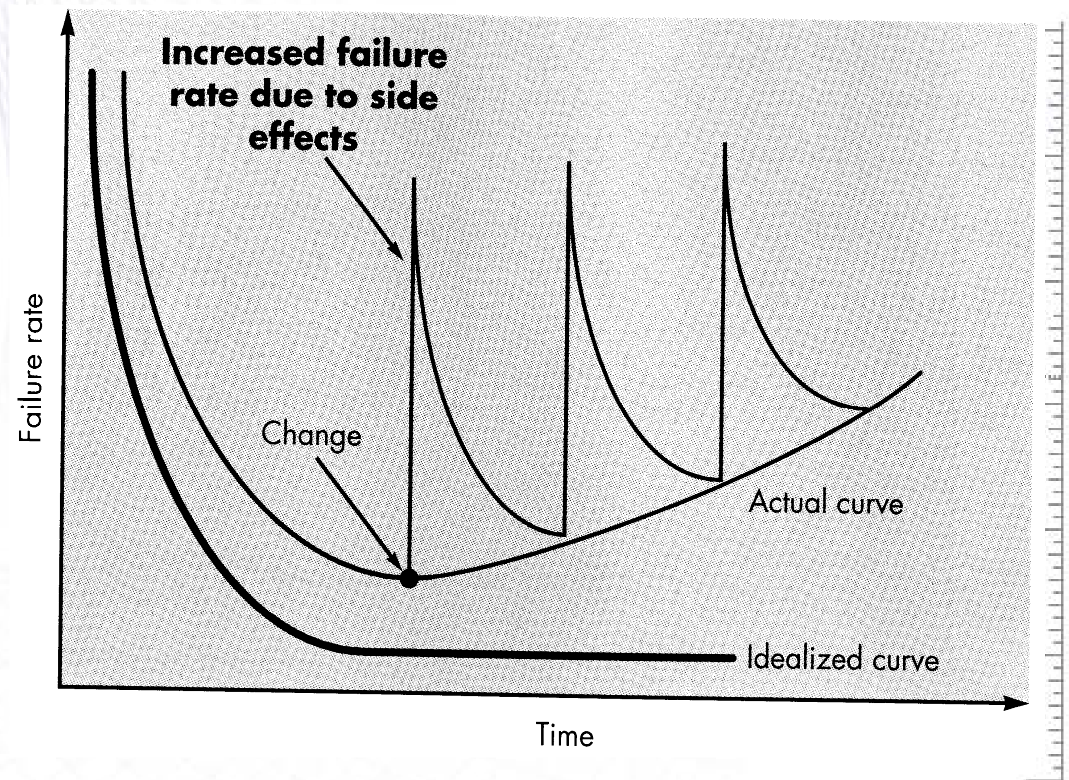
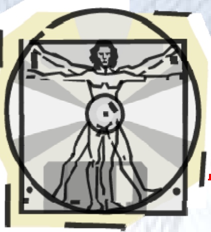
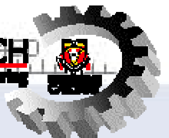


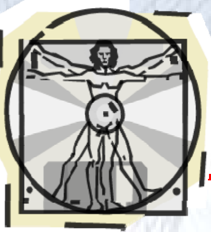
Figure from Pressman's book



Software vs. Hardware

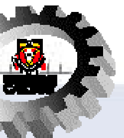
- Software reliability doesn't decrease with time, i.e., software doesn't wear out.
- Hardware faults are mostly ***physical faults***, e.g., fatigue.
- Software faults are mostly ***design faults*** which are harder to measure, model, detect and correct.

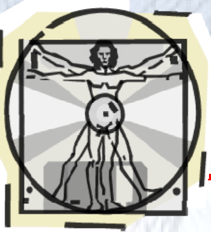




Software vs. Hardware

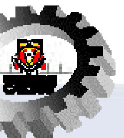
- Hardware failure can be “fixed” by replacing a faulty component with an identical one, therefore no reliability growth.
- Software problems can be “fixed” by changing the code in order to have the failure not happen again, therefore reliability growth is present.
- Software does not go through production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

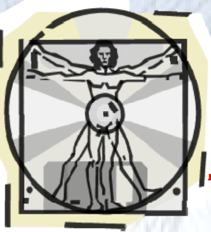




Reliability: Science

- 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.

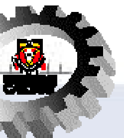


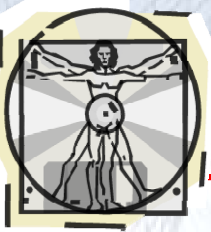


Reliability: Engineering /1

- Engineering of "reliability" in software products.
- Reliability Engineering's goal:

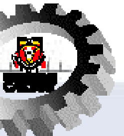
  developing software to reach the *market*
  - With "minimum" development time
  - With "minimum" development cost
  - With "maximum" reliability
  - With "minimum" expertise needed
  - With "minimum" available technology

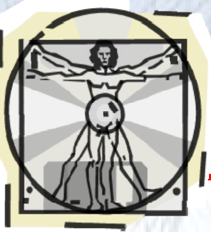




What is SRE? /1

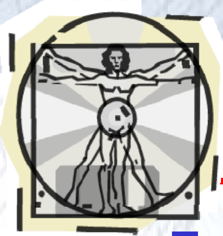
- Software Reliability Engineering (SRE) is a multi-faceted discipline covering the software product lifecycle.
- It involves both ***technical*** and ***management*** activities in three basic areas:
  - Software Development and Maintenance
  - Measurement and Analysis of reliability data
  - Feedback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





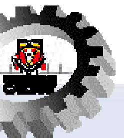
What is SRE ? /2

- 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.



However …

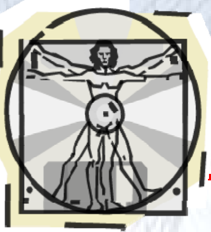
- Practical implementation of an effective SRE program is a non-trivial task.
- Mechanisms for collection and analysis of data on software product and process quality must be in place.
- Fault identification and elimination techniques must be in place.
- Other organizational abilities such as the use of reviews and inspections, reliability based testing and software process improvement are also necessary for effective SRE.



UNIVERSITY OF
CALGARY

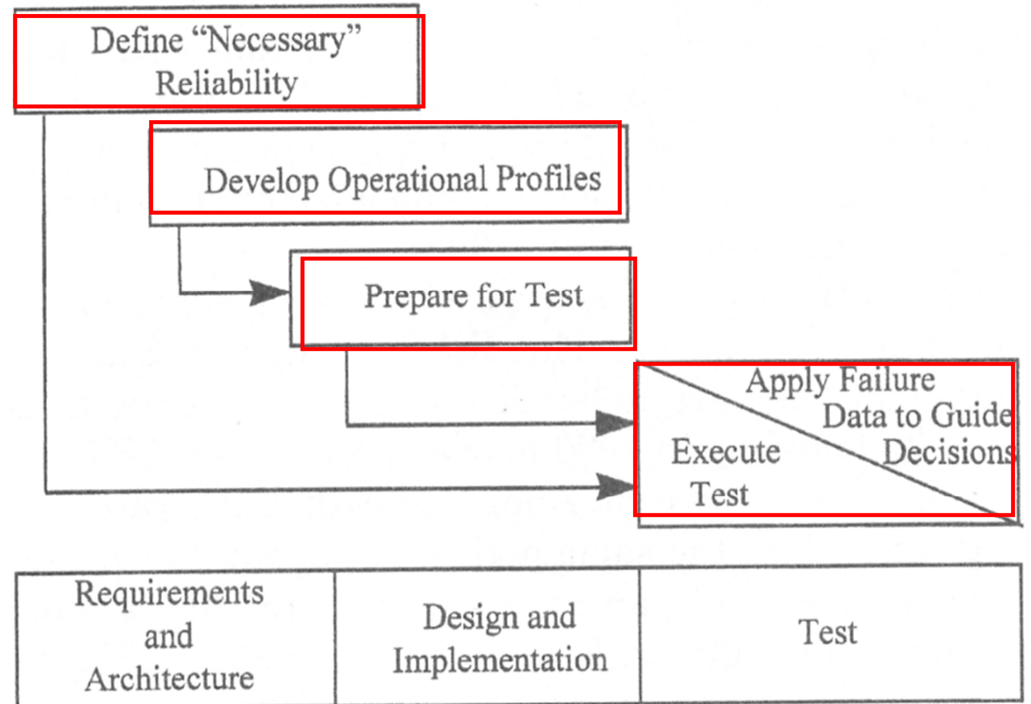
Chapter 1 Section 4

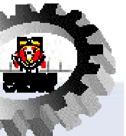
Software Reliability Engineering (SRE) Process

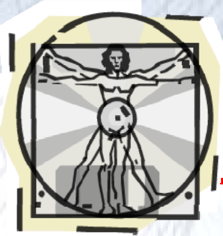


SRE: Process /1

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

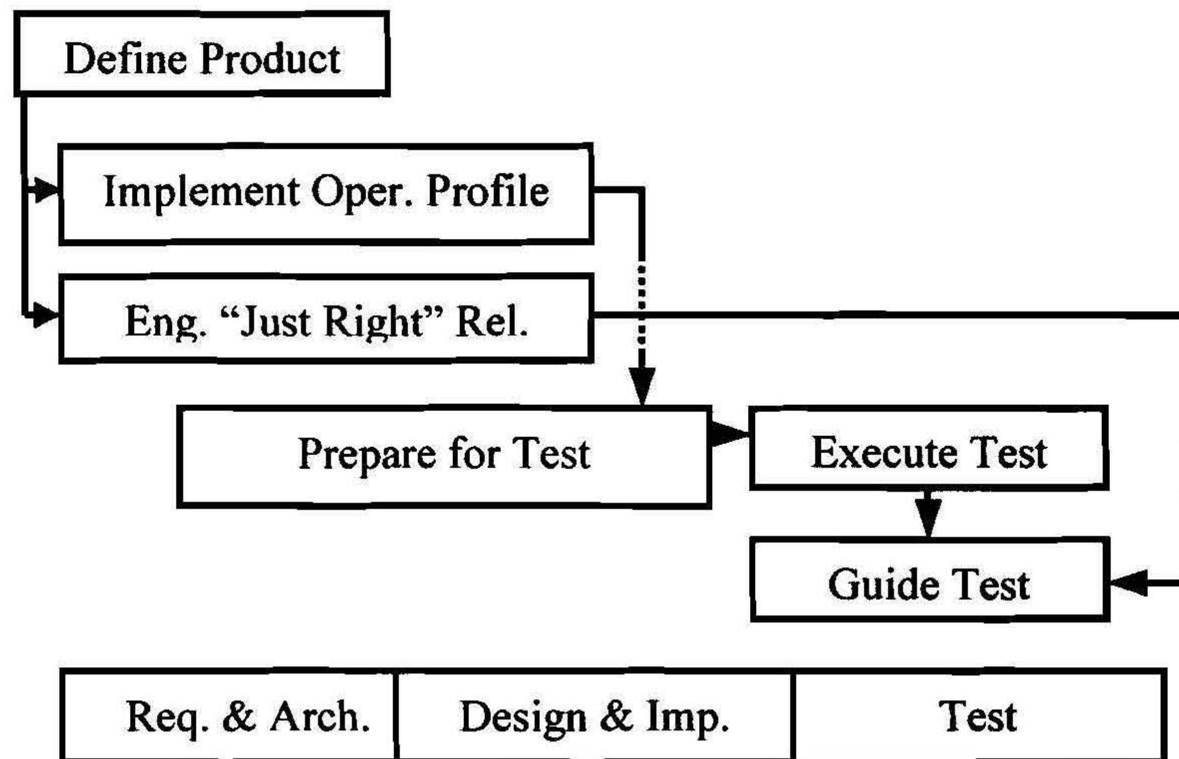




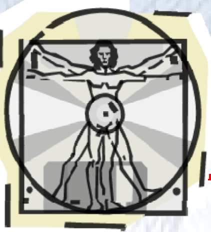


SRE: Process /2

- Modified version of the SRE Process

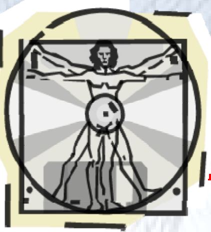


Ref: Musa's book 2nd Ed



SRE: Necessary Reliability

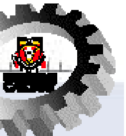
- Define what "failure" means for the software product.
- Choose a common measure for all failure intensities, either failures per some natural unit or failures per hour.
- Set the total system failure intensity objective (FIO) for the software/hardware system.
- Compute a developed software FIO by subtracting the total of the FIOs of all hardware and acquired software components from the system FIOs.
- Use the developed software FIOs to track the reliability growth during system test (later on).

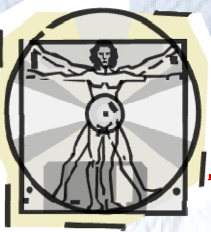


Failure Intensity Objective (FIO)

- Failure intensity (λ) is defined as failure per natural units (or time), e.g.
  - 3 alarms per 100 hours of operation.
  - 5 failures per 1000 transactions, etc.
- Failure intensity of a cascade (serial) system is the sum of failure intensities for all of the components of the system.
- For exponential model:

$$z_{system}(t) = \lambda_1 + \lambda_2 + \cdots + \lambda_n = \sum_{i=1}^{n} \lambda_i$$





How to Set FIO?

- Setting FIO in terms of system reliability (R) or availability (A):

$$\lambda = \frac{-\ln R}{t} \quad or \quad \lambda \approx \frac{(1-R)}{t} \quad for \quad R \geq 0.95$$

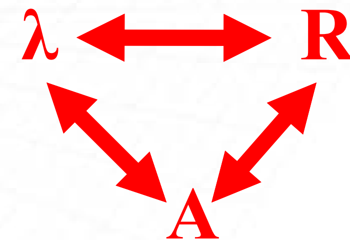
$$\lambda = \frac{1-A}{t_m A}$$

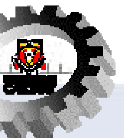
λ is failure intensity

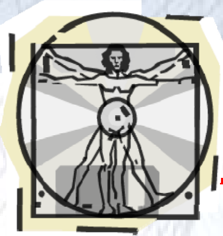
R is reliability

t is natural unit (time, etc.)

t_m is downtime per failure

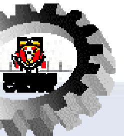


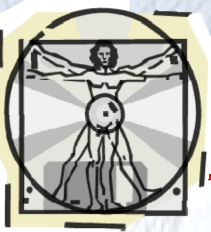




Reliability vs. Failure Intensity

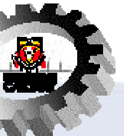
Reliability for 1 hour mission time	*Failure intensity*
0.36800	1 failure / hour
0.90000	105 failure / 1000 hours
0.95900	1 failure / day
0.99000	10 failure / 1000 hours
0.99400	1 failure / week
0.99860	1 failure / month
0.99900	1 failure / 1000 hours
0.99989	1 failure / year

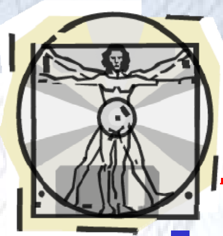




SRE: Operation

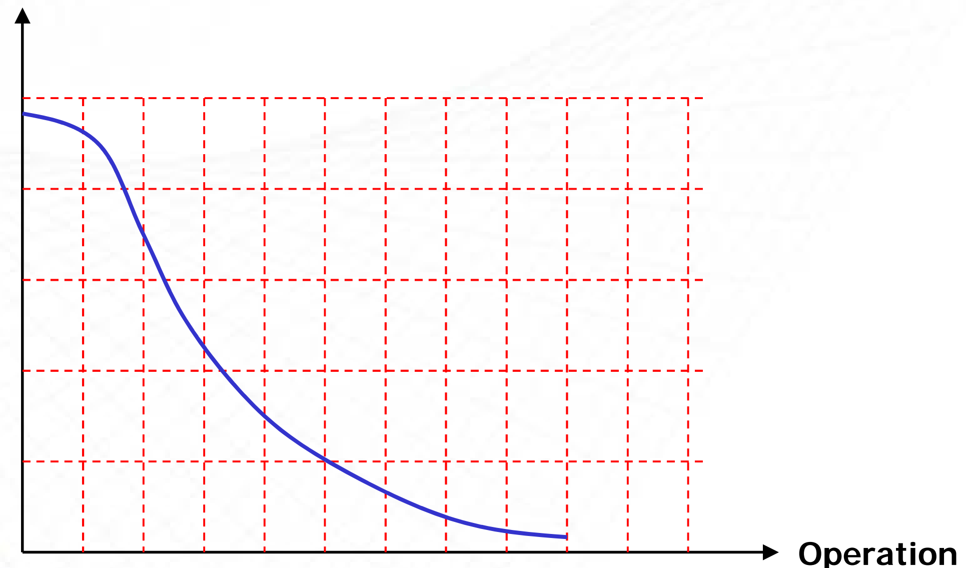
- An **operation** is a major system logical task, which returns control to the system when complete.
- An **operation** is a functionality together with its input event(s) that affects the course of behavior of software.
- **Example:** operations for a Web proxy server
  - Connect internal users to external Web
  - Email internal users to external users
  - Email external users to internal users
  - DNS request by internal users
  - Etc.

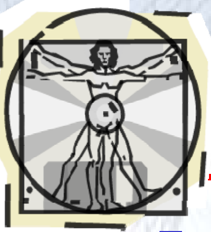




SRE: Operational Profile

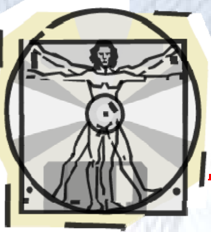
- An **operational profile** is a complete set of operations with their probabilities of occurrence (during the operational use of the software).
- An **operational profile** is a description of the distribution of input events that is expected to occur in actual software operation.
- The operational profile of the software reflects how it will be used in practice.
- **Operational mode**





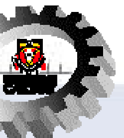
SRE: System Operational Profile

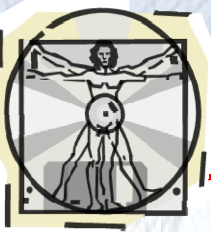
- **System operational profile** must be developed for all of its important operational modes.
- There are four principal steps in developing an operational profile:
  - Identify the operation initiators (i.e., user types, external systems, and the system itself)
  - List the operations invoked by each initiator
  - Determine the occurrence rates
  - Determine the occurrence probabilities by dividing the occurrence rates by the total occurrence rate



SRE: Prepare for Test

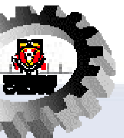
- The Prepare for Test activity uses the operational profiles to prepare test cases and test procedures.
- Test cases are allocated in accordance with the operational profile.
- Test cases are assigned to the operations by selecting from all the possible intra-operation choices with equal probability.
- The test procedure is the controller that invokes test cases during execution.

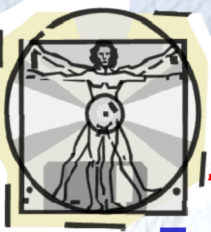




SRE: Execute Test

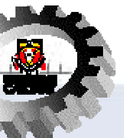
- Allocate test time among the associated systems and types of test (feature, load, regression, etc.).
- Invoke the test cases at random times, choosing operations randomly in accordance with the operational profile.
- Identify failures, along with when they occur.
- This information will be used in Apply Failure Data and Guide Test.

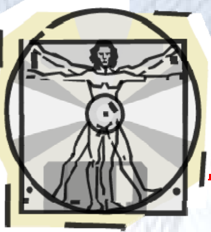




Types of Test

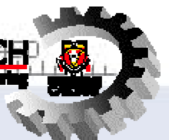
- **Certification Test:** Accept or reject (binary decision) an acquired component for a given target failure intensity.
- **Feature Test:** A single execution of an operation with interaction between operations minimized.
- **Load Test:** Testing with field use data and accounting for interactions
- **Regression Test:** Feature tests after every build involving significant change, i.e., check whether a bug fix worked.

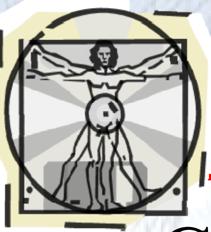




SRE: Apply Failure Data

- Plot each new failure as it occurs on a reliability demonstration chart.
- Accept or reject software (operations) using reliability demonstration chart.
- Track reliability growth as faults are removed.

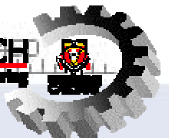


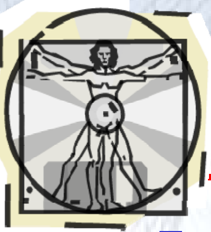


Release Criteria

Consider releasing the product when:

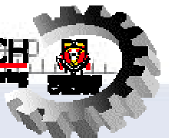
1. All acquired components pass certification test
2. Test terminated satisfactorily for all the product variations and components with the λ / λ_F ratios for these variations don't appreciably exceed 0.5 (Confidence factor)

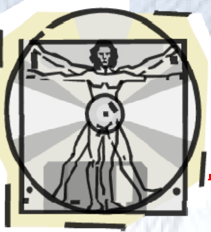




Collect Field Data

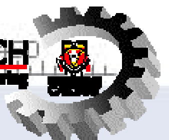
- SRE for the software product lifecycle.
- Collect field data to use in succeeding releases either using automatic reporting routines or manual collection, using a random sample of field sites.
- Collect data on failure intensity and on customer satisfaction and use this information in setting the failure intensity objective for the next release.
- Measure operational profiles in the field and use this information to correct the operational profiles we estimated.
- Collect information to refine the process of choosing reliability strategies in future projects.

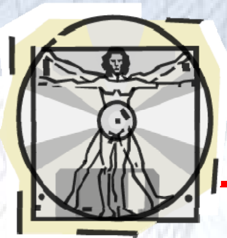


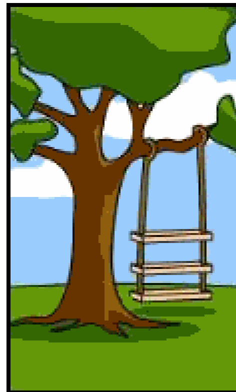


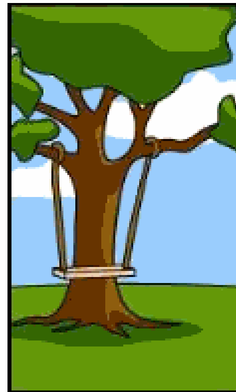
Conclusions

- Software Reliability Engineering (SRE) can offer metrics and measures to help elevate a software development organization to the upper levels of software development maturity.
- However, in practice effective implementation of SRE is a non-trivial task!

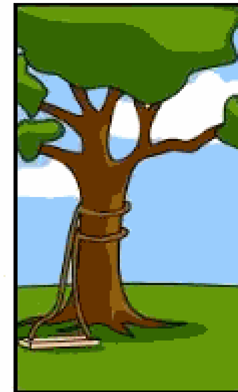


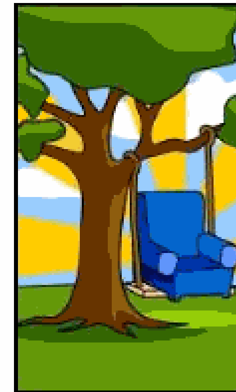


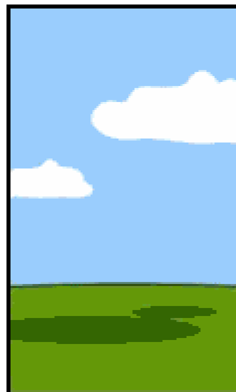

How the customer explained it

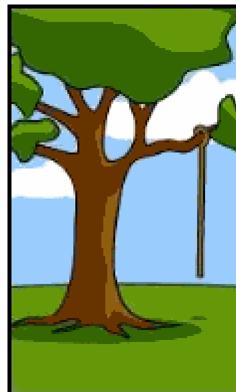

How the Project Leader understood it

How the Analyst designed it

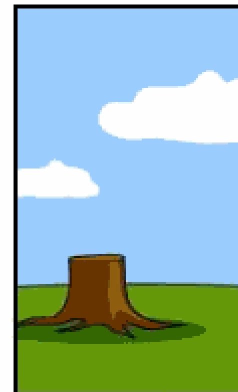

How the Programmer wrote it

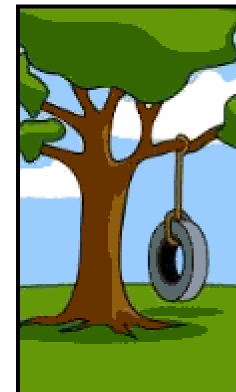

How the Business Consultant described it


How the project was documented


What operations installed

How the customer was billed


How it was supported


What the customer really needed