Reliability

Objectives

- To explain how system reliability can be measured and how reliability growth models can be used for reliability prediction
- To describe safety arguments and how these are used
- To discuss the problems of safety assurance
- To introduce safety cases and how these are used in safety validation

Validation costs

- Because of the additional activities involved, the validation costs for critical systems are usually significantly higher than for non-critical systems.
- Normally, V & V costs take up more than 50% of the total system development costs.

Reliability validation

- Reliability validation involves exercising the program to assess whether or not it has reached the required level of reliability.
- This cannot normally be included as part of a normal defect testing process because data for defect testing is (usually) atypical of actual usage data.
- Reliability measurement therefore requires a specially designed data set that replicates the pattern of inputs to be processed by the system.

Statistical testing

- Testing software for reliability rather than fault detection.
- Measuring the number of errors allows the reliability of the software to be predicted. Note that, for statistical reasons, more errors than are allowed for in the reliability specification must be induced.
- An acceptable level of reliability should be specified and the software tested and amended until that level of reliability is reached.

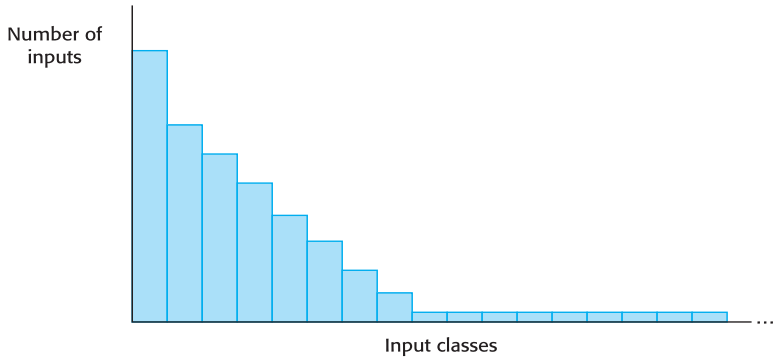
Operational profiles

- An operational profile is a set of test data whose frequency matches the actual frequency of these inputs from 'normal' usage of the system. A close match with actual usage is necessary otherwise the measured reliability will not be reflected in the actual usage of the system.
- It can be generated from real data collected from an existing system or (more often) depends on assumptions made about the pattern of usage of a system.

Reliability measurement problems

- Operational profile uncertainty
  - The operational profile may not be an accurate reflection of the real use of the system.
- High costs of test data generation
  - Costs can be very high if the test data for the system cannot be generated automatically.
- Statistical uncertainty
  - You need a statistically significant number of failures to compute the reliability but highly reliable systems will rarely fail.

An operational profile



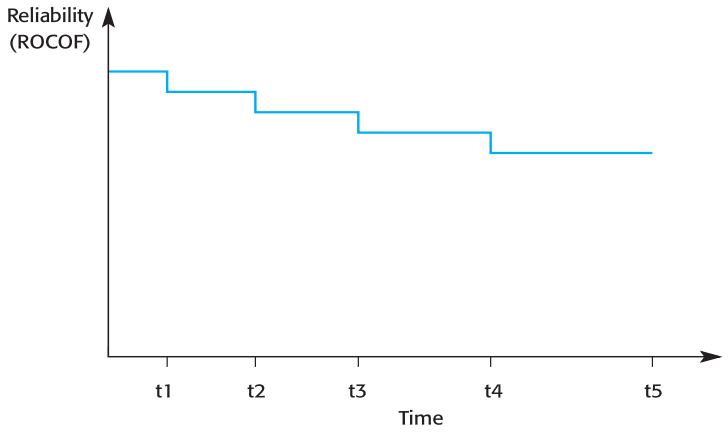
Operational profile generation

- Should be generated automatically whenever possible.
- Automatic profile generation is difficult for interactive systems.
- May be straightforward for 'normal' inputs but it is difficult to predict 'unlikely' inputs and to create test data for them.

Reliability prediction

- A reliability growth model is a mathematical model of the system reliability change as it is tested and faults are removed.
- It is used as a means of reliability prediction by extrapolating from current data
  - Simplifies test planning and customer negotiations.
  - You can predict when testing will be completed and demonstrate to customers whether or not the reliability growth will ever be achieved.
- Prediction depends on the use of statistical testing to measure the reliability of a system version.

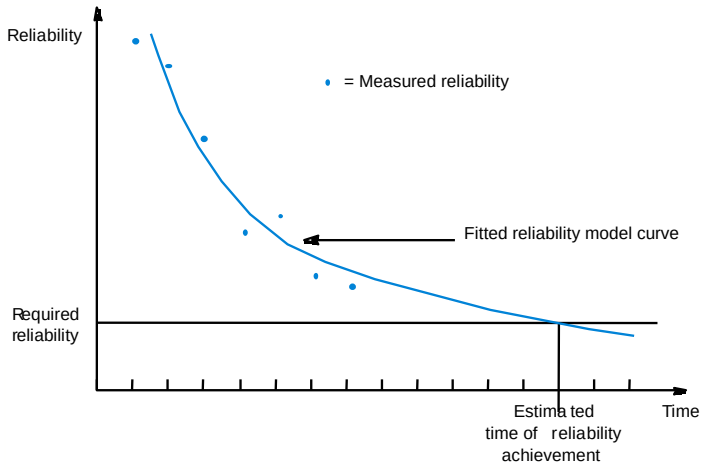
Equal-step reliability growth



Growth model selection

- Many different reliability growth models have been proposed.
- There is no universally applicable growth model.
- Reliability should be measured and observed data should be fitted to several models.
- The best-fit model can then be used for reliability prediction.

Reliability prediction



Reliability measures

- TIME DEPENDENT APPROACHES
  - TIME BETWEEN FAILURES (Musa Model)
  - FAILURE COUNTS IN SPECIFIED INTERVALS (Goel/Okumoto)
- TIME-INDEPENDENT APPROACHES
  - ERROR SEEDING
  - INPUT DOMAIN ANALYSIS
- PROBLEMS WITH USE OF RELIABILITY MODELS
  - LACK OF CLEAR UNDERSTANDING OF INHERENT STRENGTHS AND WEAKNESSES
  - UNDERLYING ASSUMPTIONS AND OUTPUTS NOT FULLY UNDERSTOOD BY USER
  - NOT ALL MODELS APPLICABLE TO ALL TESTING ENVIRONMENTS

Reliability measures - Musa

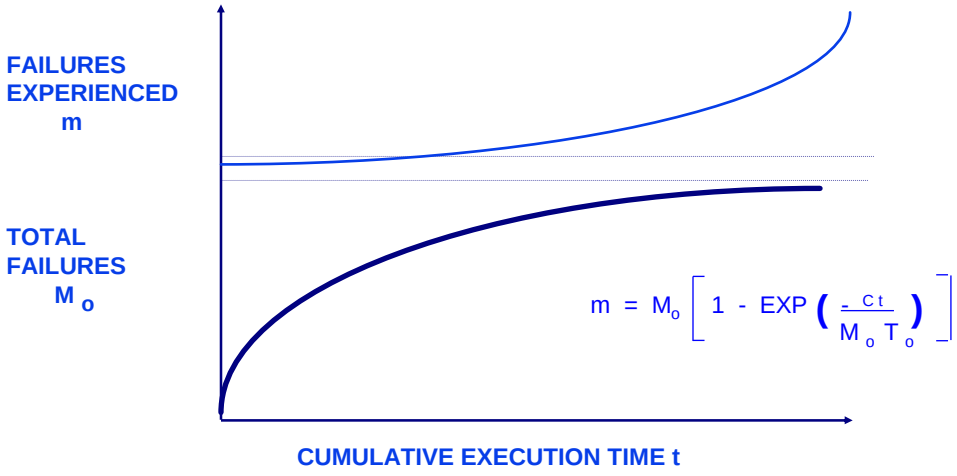
ASSUMPTIONS:

1. ERRORS ARE DISTRIBUTED RANDOMLY THROUGH THE PROGRAM
2. TESTING IS DONE WITH REPEATED RANDOM SELECTION FROM THE ENTIRE RANGE OF INPUT DATA
3. THE ERROR DISCOVERY RATE IS PROPORTIONAL TO THE NUMBER OF ERRORS IN THE PROGRAM
4. ALL FAILURES ARE TRACED TO THE ERRORS CAUSING THEM AND CORRECTED BEFORE TESTING RESUMES
5. NO NEW ERRORS ARE INTRODUCED DURING DEBUGGING

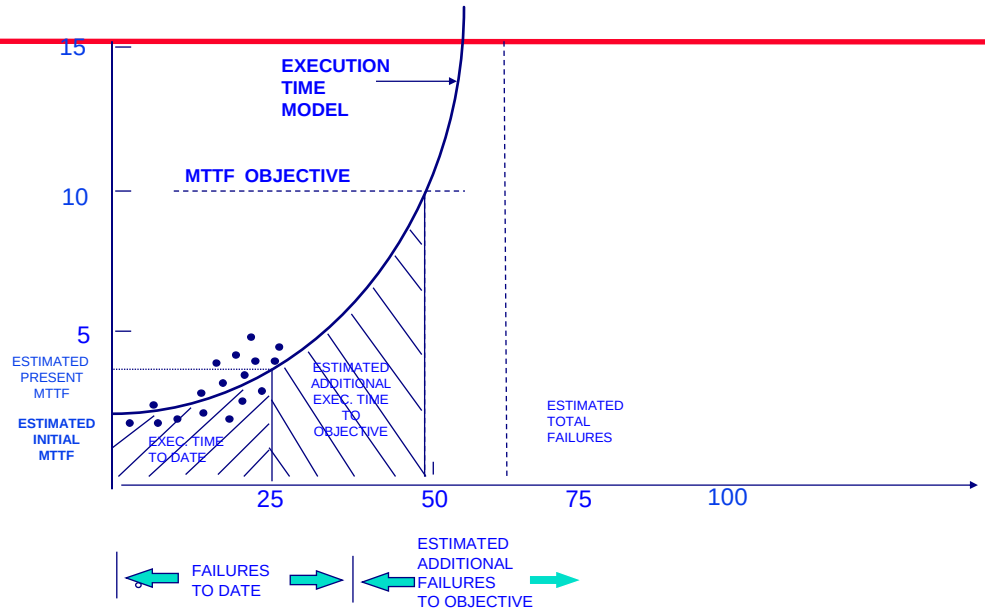
$$T = \frac{1}{KE} e^{K t}$$

WHERE **E** IS TOTAL ERRORS IN THE SYSTEM
t IS THE ACCUMULATED RUN TIME (STARTS at 0)
T IS THE MEAN TIME TO FAILURE

Reliability measures – Failure rate



Reliability measures – Using model



Capture/Recapture Reliability

- Based on fishing industry
  - Set free a number of tagged fish (e.g. 100)
  - See how many are later caught (e.g., 5)
  - Assume uniform distribution
  - Assume catch 1000 fish with 5 tagged.
  - Result says that 1000 is 5% of total population or 20,000 total population
- Apply to reliability of software
  - Seed N defects (unknown to testers)
  - Let them test. Assume they find K defects of which M were seeded
  - Assumes K*(N/M) total defects in software

Reliability – Combining approaches

CLEAN ROOM

- DEVELOPER USES READING TECHNIQUES, TOP DOWN DEVELOPMENT
- TESTING DONE BY INDEPENDENT ORGANIZATION AT INCREMENTAL STEPS
- RELIABILITY MODEL USED TO PROVIDE DEVELOPER WITH QUALITY ASSESSMENT

FUNCTIONAL TESTING/COVERAGE METRICS

- USE FUNCTIONAL TESTING APPROACH
- COLLECT ERROR DISTRIBUTIONS, E.G., OMISSION vs COMMISSION
- OBTAIN COVERAGE METRICS
- KNOWING NUMBER OF ERRORS OF OMISSION, EXTRAPOLATE

ERROR ANALYSIS AND RELIABILITY MODELS

- ESTABLISH ERROR HISTORY FROM PREVIOUS PROJECTS
- DISTINGUISH SIMILARITIES AND DIFFERENCES TO CURRENT PROJECT
- DETERMINE PRIOR ERROR DISTRIBUTIONS FOR CURRENT PROJECT
- SELECT CLASS OF STOCHASTIC MODELS FOR CURRENT PROJECT
- UPDATE PRIOR DISTRIBUTIONS AND COMPARE ACTUAL DATA WITH THE PRIORS FOR THE CURRENT PROJECT

Approaches toward reliability: Fault tolerance

- Fault detection
  - The system must detect that a fault (an incorrect system state) has occurred.
- Damage assessment
  - The parts of the system state affected by the fault must be detected.
- Fault recovery
  - The system must restore its state to a known safe state.
- Fault repair
  - The system may be modified to prevent recurrence of the fault. As many software faults are transitory, this is often unnecessary.

Fault detection

- Preventative fault detection
  - The fault detection mechanism is initiated before the state change is committed. If an erroneous state is detected, the change is not made.
- Retrospective fault detection
  - The fault detection mechanism is initiated after the system state has been changed. This is used when a incorrect sequence of correct actions leads to an erroneous state or when preventative fault detection involves too much overhead.

Type system extension

- Preventative fault detection really involves extending the type system by including additional constraints as part of the type definition.
- These constraints are implemented by defining basic operations within a class definition.

Damage assessment

- Analyze system state to judge the extent of corruption caused by a system failure.
- The assessment must check what parts of the state space have been affected by the failure.
- Generally based on 'validity functions' that can be applied to the state elements to assess if their value is within an allowed range.

Damage assessment techniques

- Checksums are used for damage assessment in data transmission.
- Redundant pointers can be used to check the integrity of data structures.
- Watch dog timers can check for non-terminating processes. If no response after a certain time, a problem is assumed.

Fault recovery and repair

- Forward recovery
  - Apply repairs to a corrupted system state.
- Backward recovery
  - Restore the system state to a known safe state.
- Forward recovery is usually application specific - domain knowledge is required to compute possible state corrections.
- Backward error recovery is simpler. Details of a safe state are maintained and this replaces the corrupted system state.

Forward recovery

- Corruption of data coding
  - Error coding techniques which add redundancy to coded data can be used for repairing data corrupted during transmission.
- Redundant pointers
  - When redundant pointers are included in data structures (e.g. two-way lists), a corrupted list or filestore may be rebuilt if a sufficient number of pointers are uncorrupted
  - Often used for database and file system repair.

Backward recovery

- Transactions are a frequently used method of backward recovery. Changes are not applied until computation is complete. If an error occurs, the system is left in the state preceding the transaction.
- Periodic checkpoints allow system to 'roll-back' to a correct state.

Safety assurance

- Safety assurance and reliability measurement are quite different:
  - Within the limits of measurement error, you know whether or not a required level of reliability has been achieved;
  - However, quantitative measurement of safety is impossible. Safety assurance is concerned with establishing a confidence level in the system.
- Confidence is developed through:
  - Past experience with the company developing the software;
  - The use of dependable processes and process activities geared to safety;
  - Extensive V & V including both static and dynamic validation techniques.

Safety reviews

- Review for correct intended function.
- Review for maintainable, understandable structure.
- Review to verify algorithm and data structure design against specification.
- Review to check code consistency with algorithm and data structure design.
- Review adequacy of system testing.

Safety arguments

- Safety arguments are intended to show that the system cannot reach in unsafe state.
- These are weaker than correctness arguments which must show that the system code conforms to its specification.
- They are generally based on proof by contradiction
  - Assume that an unsafe state can be reached;
  - Show that this is contradicted by the program code.
- A graphical model of the safety argument may be developed.

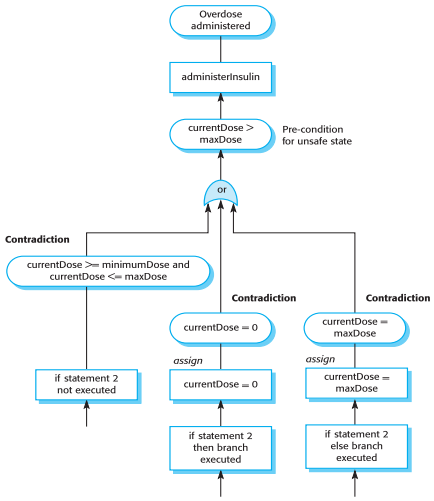
Construction of a safety argument

- Establish the safe exit conditions for a component or a program.
- Starting from the END of the code, work backwards until you have identified all paths that lead to the exit of the code.
- Assume that the exit condition is false.
- Show that, for each path leading to the exit that the assignments made in that path contradict the assumption of an unsafe exit from the component.

Example code: Delivery code

```
currentDose = computeInsulin () ;
// Safety check - adjust currentDose if necessary
// if st atement 1
if (previousDose == 0)
{
      if (currentDose > 16)
                  currentDose = 16 ;
}
else
      if (currentDose > (previousDose * 2) )
            currentDose = p reviousDose * 2 ;
// if st atement 2
if ( currentDose < m inimumDose )
            currentDose = 0 ;
else if ( c urrentDose > ma xDose )
            currentDose = maxDose ;
administerInsulin (currentDose) ;
```

Safety argument model



Program paths

- Neither branch of if-statement 2 is executed
  - Can only happen if CurrentDose is >= minimumDose and <= maxDose.
- then branch of if-statement 2 is executed
  - currentDose = 0.
- else branch of if-statement 2 is executed
  - currentDose = maxDose.
- In all cases, the post conditions contradict the unsafe condition that the dose administered is greater than maxDose.

Hazard analysis

- Hazard analysis involves identifying hazards and their root causes.
- There should be clear traceability from identified hazards through their analysis to the actions taken during the process to ensure that these hazards have been covered.
- A hazard log may be used to track hazards throughout the process.

Hazard log entry

Hazard Log. **Page 4: Printed 20.02.2(**

System: Insulin Pump System *File:* InsulinPump/Safety/HazardLog

Safety Engineer: James Brown *Log version:* 1/3

Identified Hazard	Insulin overdose delivered to patient				
Identified by	Jane Williams				
Criticality class	1				
Identified risk	High				
Fault tree identified	YES	*Date*	24.01.99	*Location*	Hazard Log, Page 5
Fault tree creators	Jane Williams and Bill Smith				
Fault tree checked	YES	*Date*	28.01.99	*Checker*	James Brown

System safety design requirements

1. The system shall include self-testing software that will test the sensor system, the clock and the insulin delivery system.
2. The self-checking software shall be executed once per minute
3. In the event of the self-checking software discovering a fault in any of the system components, an audible warning shall be issued and the pump display should indicate the name of the component where the fault has been discovered. The delivery of insulin should be suspended.
4. The system shall incorporate an override system that allows the system user to modify the computed dose of insulin that is to be delivered by the system.
5. The amount of override should be limited to be no greater than a pre-set value that is set when the system is configured by medical staff.

Run-time safety checking

- During program execution, safety checks can be incorporated as assertions to check that the program is executing within a safe operating 'envelope'.
- Assertions can be included as comments (or using an assert statement in some languages). Code can be generated automatically to check these assertions.

Insulin administration with assertions

```
static void administerInsulin ( ) throws SafetyException {

        int maxIncrements = InsulinPump.maxDose / 8 ;
        int increments = InsulinPump.currentDose / 8 ;

        // assert currentDose <= InsulinPump.maxDose

        if (InsulinPump.currentDose > InsulinPump.maxDose)
            throw new SafetyException (Pump.doseHigh);
        else
            for (int i=1; i<= increments; i++)
            {
                generateSignal ();
                if (i > maxIncrements)
                    throw new SafetyException ( Pump.incorrectIncrements);
            } // for loop

    } //administerInsulin
```

Will discuss this next time – formal specifications

Security assessment

- Security assessment has something in common with safety assessment.
- It is intended to demonstrate that the system cannot enter some state (an unsafe or an insecure state) rather than to demonstrate that the system can do something.
- However, there are differences
  - Safety problems are accidental; security problems are deliberate;
  - Security problems are more generic - many systems suffer from the same problems; Safety problems are mostly related to the application domain

Security validation

- Experience-based validation
  - The system is reviewed and analysed against the types of attack that are known to the validation team.
- Tool-based validation
  - Various security tools such as password checkers are used to analyse the system in operation.
- Tiger teams
  - A team is established whose goal is to breach the security of the system by simulating attacks on the system.
- Formal verification
  - The system is verified against a formal security specification.

Security checklist

1. Do all files that are created in the application have appropriate access permissions? The wrong access permissions may lead to these files being accessed by unauthorized users.
2. Does the system automatically terminate user sessions after a period of inactivity? Sessions that are left active may allow unauthorized access through an unattended computer.
3. If the system is written in a programming language without array bound checking, are there situations where buffer overflow may be exploited? Buffer overflow may allow attackers to send code strings to the system and then execute them.
4. If passwords are set, does the system check that password are 'strong'. Strong passwords consist of mixed letters, numbers and punctuation and are not normal dictionary entries. They are more difficult to break than simple passwords.

Safety and dependability cases

- Safety and dependability cases are structured documents that set out detailed arguments and evidence that a required level of safety or dependability has been achieved.
- They are normally required by regulators before a system can be certified for operational use.

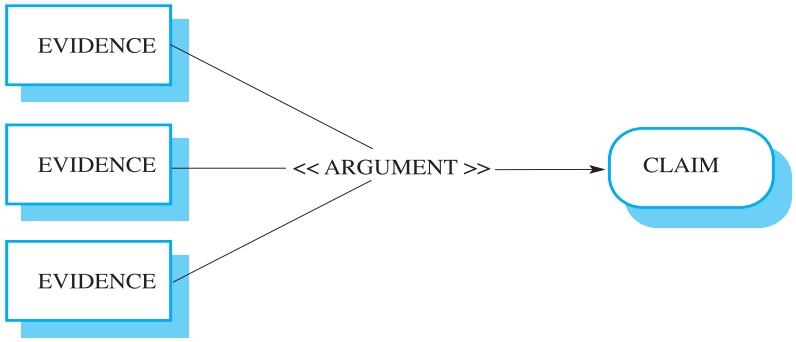
The system safety case

- It is now normal practice for a formal safety case to be required for all safety-critical computer-based systems e.g. railway signalling, air traffic control, etc.
- A safety case is:
  - A documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment.
- Arguments in a safety or dependability case can be based on formal proof, design rationale, safety proofs, etc. Process factors may also be included.

Components of a safety case

Component	Description
System description	An overview of the system and a description of its critical components.
Safety requirements	The safety requirements abstracted from the system requirements specification.
Hazard and risk analysis	Documents describing the hazards and risks that have been identified and the measures taken to reduce risk.
Design analysis	A set of structured arguments that justify why the design is safe.
Verification and validation	A description of the V & V procedures used and, where appropriate, the test plans for the system. Results of system V &V.
Review reports	Records of all design and safety reviews.
Team competences	Evidence of the competence of all of the team involved in safety-related systems development and validation.
Process QA	Records of the quality assurance processes carried out during system development.
Change management processes	Records of all changes proposed, actions taken and, where appropriate, justification of the safety of these changes.
Associated safety cases	References to other safety cases that may impact on this safety case.

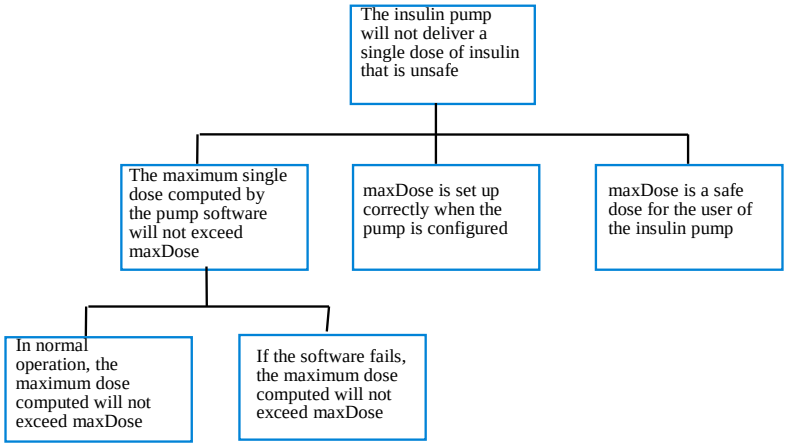
Argument structure



Insulin pump argument

Claim:	The maximum single dose computed by the insulin pump will not exceed maxDose.
Evidence:	Safety argument for insulin pump as shown in Figure 24.7
Evidence:	Test data sets for insulin pump
Evidence:	Static analysis report for insulin pump software
Argument:	The safety argument presented shows that the maximum dose of insulin that can be computed is equal to maxDose. In 400 tests, the value of Dose was correctly computed and never exceeded maxDose. The static analysis of the control software revealed no anomalies. Overall, it is reasonable to assume that the claim is justified.

Claim hierarchy



What is quality?

Besides reliability and fault tolerance, we also must consider quality:

- Quality, simplistically, means that a product should meet its specification.
- This is problematical for software systems
  - There is tension between customer requirements, such as between quality and cost.
  - There is a tension between customer quality requirements (efficiency, reliability, etc.) and developer quality requirements (maintainability, reusability, etc.);
  - Some quality requirements are difficult to specify in an unambiguous way;
  - Software specifications are usually incomplete and often inconsistent.

Process and product quality

- Basic assumption: The quality of a developed product is influenced by the quality of the production process.
  - This is important in software development as some product quality attributes are hard to assess.
  - However, there is a very complex and poorly understood relationship between software processes and product quality.
- Bad processes generally produce bad software, but it has not been adequately proven that good processes produce good software
  - But is assumption behind programs such as CMMI and ISO 9000.

Practical process quality

- Define process standards such as how reviews should be conducted, configuration management, etc.
- Monitor the development process to ensure that standards are being followed.
- Report on the process to project management and software procurer.
- Don't use inappropriate practices simply because standards have been established.

Example: Reliability, correctness, quality

- Consider 2 clocks:
  - Clock A doesn't work (it is stopped).
  - Clock B loses 1 minute per day
- How accurate is each clock? Which clock is more accurate?
- Which clock is more reliable? Why?
- Which clock has higher quality? (What is the specification for a clock and which clock best meets that specification?)

Key points

- Reliability measurement relies on exercising the system using an operational profile - a simulated input set which matches the actual usage of the system.
- Reliability growth modelling is concerned with modelling how the reliability of a software system improves as it is tested and faults are removed.
- Safety arguments or proofs are a way of demonstrating that a hazardous condition can never occur.

Key points

- It is important to have a dependable process for safety-critical systems development. The process should include hazard identification and monitoring activities.
- Security validation may involve experience-based analysis, tool-based analysis or the use of 'tiger teams' to attack the system.
- Safety cases collect together the evidence that a system is safe.