ECE 651 – Software Engineering
Week Seven: Dependable Systems
March 2nd, 2016

Ric Telford
Adjunct Associate Professor

Founder, Telford Ventures

Duke | PRATT SCHOOL OF ENGINEERING

Extra Credit – 2 points

- Write in 2 topics that you want to make sure we cover this semester. It can be anything related to the practice of Software Engineering you want – does not have to be from the list below.
- Some of the remaining topics in the book (and others) include:

Safety Engineering
Resilience Engineering
Component-based software eng.
Service-oriented (SOA, REST, etc)
Systems of systems
Project Management
Quality Management
Agile deep dive
Maintenance and Reengineering
Mobile App Design and Test

Security Engineering
Software Reuse
Distributed Software engineering
Systems engineering
Real-time software engineering
Project Planning
Configuration Management
Process and Project Metrics
Risk Management
Web App Design and Test

03/02/16 – Week 7 Overview

- Recap / Announcements
- Lecture – Software Testing
- Homework Review
- Break
- Midterm Review
- Lecture – Dependable Systems
- About Week 8

Recap of Week 6

- Testing is intended to show that a program does what it is intended to do (Validation Testing) and to discover program defects before it is put into use (Defect Testing)
- There are 3 stages of Testing:
  - Development Testing is the responsibility of the software development team and includes Unit Test, Component Test and System Test. Know all three.
  - Release Testing is where a separate testing team tests a complete version of the system before it is released to users
  - User Testing is where users or potential users of a sytem test it in the own environment
- Verification (building product right) and Validation (building the right product)
- Inspections (static analysis of a system) and Testing (dynamic verification)
- Wherever possible, write automated tests using JUnit or something similar. They can be run every time a change is made to the system.

Announcements

- Sprint 1 code has been made public if you want to see how others have coded something
  - If you reuse code from someone else, make sure to acknowledge in the comments from who you got the code!
  - Otherwise you can use it just to see how someone did something then write your own code.
- Second half of the semester there won't be homework, but...
  - The expectation is that you are working on your project.
  - You need to set up checkpoint meeting with me – sometime in late March
  - You need to set up 2 checkpoint meetings with your TA – mid March, early April
  - Everyone should be owning code, checking in code, running test cases, etc as laid out in your Project Plan
- In small team development, it is important that everyone can contribute everywhere.
  - Make sure you know Git, Eclipse, JUnit (as applicable), etc
  - If you don't understand something, please ask for help on Piazza, from your TA, or from me.

Homework 6 Review

- Very nice job by everyone!
- I have not graded yet, but right now I don't see anyone getting below the baseline.
- There was something to like in everyone's – some had nice Gantt charts, one had defined their critical path, good structure, overviews, etc
- Some teams are still filename challenged…
- Keep the Project Plan updated to match your actual plan as it changes. We will continue to review in checkpoint meetings

Software Test, continued

System testing

- System testing during development involves integrating components to create a version of the system and then testing the integrated system.
- The focus in system testing is testing the interactions between components.
- System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.
- System testing tests the emergent behaviour of a system.

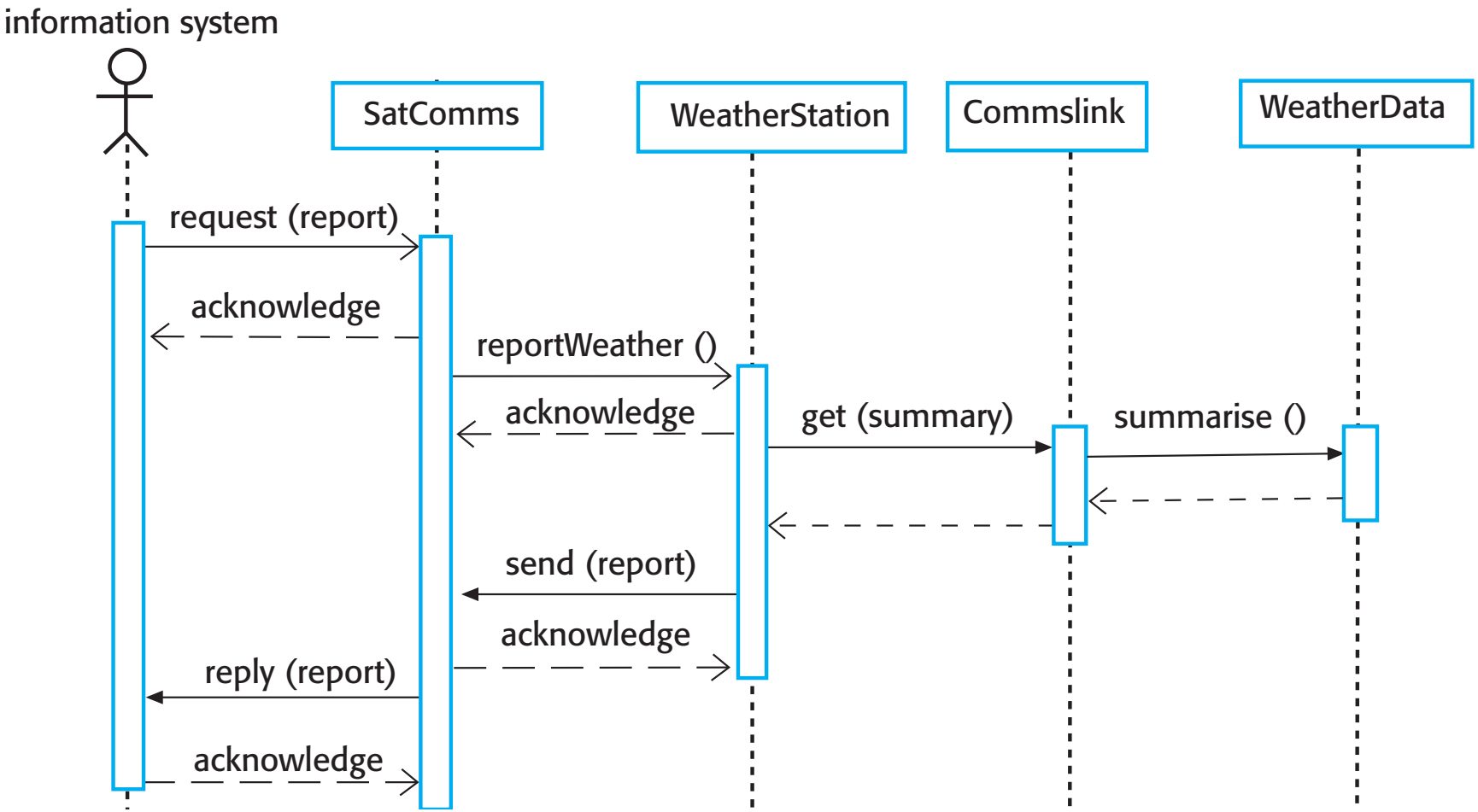
System and component testing

- During system testing, reusable components that have been separately developed and off-the-shelf systems may be integrated with newly developed components. The complete system is then tested.
- Components developed by different team members or sub-teams may be integrated at this stage. System testing is a collective rather than an individual process.
  - In some companies, system testing may involve a separate testing team with no involvement from designers and programmers.

Use-case testing

- The use-cases developed to identify system interactions can be used as a basis for system testing.
- Each use case usually involves several system components so testing the use case forces these interactions to occur.
- The sequence diagrams associated with the use case documents the components and interactions that are being tested.

Collect weather data sequence chart



Test cases derived from sequence diagram

- An input of a request for a report should have an associated acknowledgement. A report should ultimately be returned from the request.
  - You should create summarized data that can be used to check that the report is correctly organized.
- An input request for a report to WeatherStation results in a summarized report being generated.
  - Can be tested by creating raw data corresponding to the summary that you have prepared for the test of SatComms and checking that the WeatherStation object correctly produces this summary. This raw data is also used to test the WeatherData object.

Testing policies

- Exhaustive system testing is impossible so testing policies which define the required system test coverage may be developed.
- Examples of testing policies:
  - All system functions that are accessed through menus should be tested.
  - Combinations of functions (e.g. text formatting) that are accessed through the same menu must be tested.
  - Where user input is provided, all functions must be tested with both correct and incorrect input.

Release testing

Release testing

- Release testing is the process of testing a particular release of a system that is intended for use outside of the development team.
- The primary goal of the release testing process is to convince the supplier of the system that it is good enough for use.
  - Release testing, therefore, has to show that the system delivers its specified functionality, performance and dependability, and that it does not fail during normal use.
- Release testing is usually a black-box testing process where tests are only derived from the system specification.

Release testing and system testing

- *Release testing is a form of system testing.*
- Important differences:
  - A separate team that has not been involved in the system development, should be responsible for release testing.
  - System testing by the development team should focus on discovering bugs in the system (defect testing). The objective of release testing is to check that the system meets its requirements and is good enough for external use (validation testing).

Requirements based testing

- Requirements-based testing involves examining each requirement and developing a test or tests for it.
- Mentcare system requirements:
  - If a patient is known to be allergic to any particular medication, then prescription of that medication shall result in a warning message being issued to the system user.
  - If a prescriber chooses to ignore an allergy warning, they shall provide a reason why this has been ignored.

Requirements tests

- Set up a patient record with no known allergies. Prescribe medication for allergies that are known to exist. Check that a warning message is not issued by the system.
- Set up a patient record with a known allergy. Prescribe the medication to that the patient is allergic to, and check that the warning is issued by the system.
- Set up a patient record in which allergies to two or more drugs are recorded. Prescribe both of these drugs separately and check that the correct warning for each drug is issued.
- Prescribe two drugs that the patient is allergic to. Check that two warnings are correctly issued.
- Prescribe a drug that issues a warning and overrule that warning. Check that the system requires the user to provide information explaining why the warning was overruled.

A usage scenario for the Mentcare system

George is a nurse who specializes in mental healthcare. One of his responsibilities is to visit patients at home to check that their treatment is effective and that they are not suffering from medication side effects.

On a day for home visits, George logs into the Mentcare system and uses it to print his schedule of home visits for that day, along with summary information about the patients to be visited. He requests that the records for these patients be downloaded to his laptop. He is prompted for his key phrase to encrypt the records on the laptop.

One of the patients that he visits is Jim, who is being treated with medication for depression. Jim feels that the medication is helping him but believes that it has the side effect of keeping him awake at night. George looks up Jim's record and is prompted for his key phrase to decrypt the record. He checks the drug prescribed and queries its side effects. Sleeplessness is a known side effect so he notes the problem in Jim's record and suggests that he visits the clinic to have his medication changed. Jim agrees so George enters a prompt to call him when he gets back to the clinic to make an appointment with a physician. George ends the consultation and the system re-encrypts Jim's record.

After, finishing his consultations, George returns to the clinic and uploads the records of patients visited to the database. The system generates a call list for George of those patients who He has to contact for follow-up information and make clinic appointments.

Features tested by scenario

- Authentication by logging on to the system.
- Downloading and uploading of specified patient records to a laptop.
- Home visit scheduling.
- Encryption and decryption of patient records on a mobile device.
- Record retrieval and modification.
- Links with the drugs database that maintains side-effect information.
- The system for call prompting.

Performance testing

- Part of release testing may involve testing the emergent properties of a system, such as performance and reliability.
- Tests should reflect the profile of use of the system.
- Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance becomes unacceptable.
- Stress testing is a form of performance testing where the system is deliberately overloaded to test its failure behaviour.

User testing

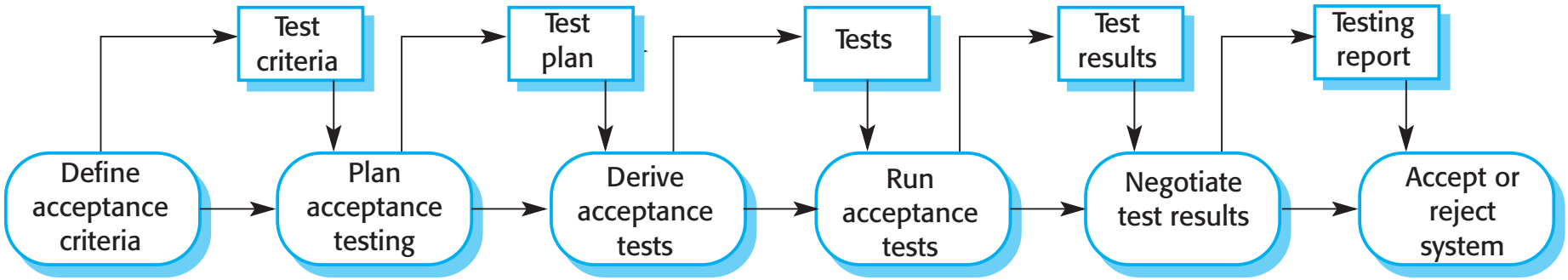
User testing

- User or customer testing is a stage in the testing process in which users or customers provide input and advice on system testing.
- User testing is essential, even when comprehensive system and release testing have been carried out.
  - The reason for this is that influences from the user's working environment have a major effect on the reliability, performance, usability and robustness of a system. These cannot be replicated in a testing environment.

Types of user testing

- Alpha testing
  - Users of the software work with the development team to test the software at the developer's site.
- Beta testing
  - A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.
- Acceptance testing
  - Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

The acceptance testing process



Stages in the acceptance testing process

- Define acceptance criteria
- Plan acceptance testing
- Derive acceptance tests
- Run acceptance tests
- Negotiate test results
- Reject/accept system

Agile methods and acceptance testing

- In agile methods, the user/customer is part of the development team and is responsible for making decisions on the acceptability of the system.
- Tests are defined by the user/customer and are integrated with other tests in that they are run automatically when changes are made.
- There is no separate acceptance testing process.
- Main problem here is whether or not the embedded user is 'typical' and can represent the interests of all system stakeholders.

Key points

- System Test is the final phase of Development Test and involves integration of all components into a candidate “system”
- Release testing is the process of testing a particular release of a system that is intended for use outside of the development team.
  - Scenario testing involves inventing a typical usage scenario and using this to derive test cases.
- User or customer testing is a stage in the testing process in which users or customers provide input and advice on system testing.
  - Acceptance testing is a user testing process where the aim is to decide if the software is good enough to be deployed and used in its operational environment.

BREAK

Midterm Study Guide

- I have broken each section of the book down by Top Priority, Priority, and Lower Priority. These correlate to things you Must Know, Should Know, and Not as Important out of book and lectures. This should help you prioritize your studying.

Chapter	Top Priority	Priority	Lower Priority
1 Intro	1.0, 1.1, 1.1.1, 1.3, 1.3.2, Key Points	1.1.2, 1.3.1, 1.3.3	1.1.3, 1.3.4
2 Process	2.0, 2.1, 2.1.1, 2.1.2, 2.2, 2.2.1, 2.2.2, 2.2.3, Key Points	2.1.3, 2.4	2.2.4, 2.3, 2.3.1, 2.3.2
3 Agile	3.0, 3.1, 3.2, 3.2.1, 3.3, Key Points	3.2.2, 3.2.3, 3.4.1, 3.4.2	3.2.4, 3.4, 3.4.3, 3.4.4
4 Rqmts	4.0, 4.1, 4.1.1, 4.1.2, 4.2, 4.3, 4.3.2, 4.4, 4.4.1, 4.4.2, 4.4.3, 4.4.4, Key Points	4.3.1, 4.5, 4.6, 4.6.1, 4.6.2	
5 Modeling	5.0, 5.1, 5.2, 5.2.1, 5.3, 5.3.1, 5.3.2, 5.3.3, Key Points	5.2.2, 5.4, 5.4.1, 5.4.2	5.4.3, 5.5
6 Arch	6.0, 6.1, 6.2, 6.3, 6.3.1, 6.3.2, 6.3.3, 6.3.4, Key Points	6.4, 6.4.1, 6.4.2, 6.4.3	
7 Design & Impl	7.0, 7.1, 7.1.1, 7.1.2, 7.1.3, 7.1.4, Key Points	7.2, 7.3, 7.3.1, 7.3.2, 7.3.3, 7.3.4	7.4, 7.4.1
8 Testing	8.0, 8.1, 8.1.1, 8.1.2, 8.1.3, 8.1.4, 8.3, 8.3.1, 8.3.2, Key Points	8.4	8.2
OTHER	My lecture on Project Management, Agile Library, Git commands, Eclipse / Java Basics		

Example Midterm Questions

- Any teams want to volunteer to answer a question or should I just choose someone?

Q1. Software Processes are interleaved sequences of activities across 3 domains. Which is these is NOT one of those domains?

A. Technical Activities
B. Marketing Activities
C. Collaborative Activities
D. Managerial Activities

Q2. What do you call an early version of a software system that is used to demonstrate concepts and try out design options?

A. Beta Version

B. Alpha Version

C. Prototype

D. Storyboard

Q3. What do you call the Agile practice of constantly looking for code improvements and implementing them immediately?

A. Refactoring

B. Test-first Development

C. Sustainable Pace

D. Pair Programming

Q4. Regulatory requirements and Legislative requirements are specific examples of what class of non-functional requirements?

A. Product Requirements

B. Organizational Requirements

C. External Requirements

D. Usability Requirements

Q6. What do you call the official statement of what the system developers should implement?

A. Architectural Overview

B. Programmer's Bible

C. Software Requirements Document

D. System Test Plan

Chapter 10 – Dependable Systems
Topics covered

- Dependability properties
- Sociotechnical systems
- Redundancy and diversity
- Dependable processes
- Formal methods and dependability

System dependability

- For many computer-based systems, the most important system property is the dependability of the system.
- The dependability of a system reflects the user's degree of trust in that system. It reflects the extent of the user's confidence that it will operate as users expect and that it will not 'fail' in normal use.
- Dependability covers the related systems attributes of reliability, availability and security. These are all inter-dependent.

Importance of dependability

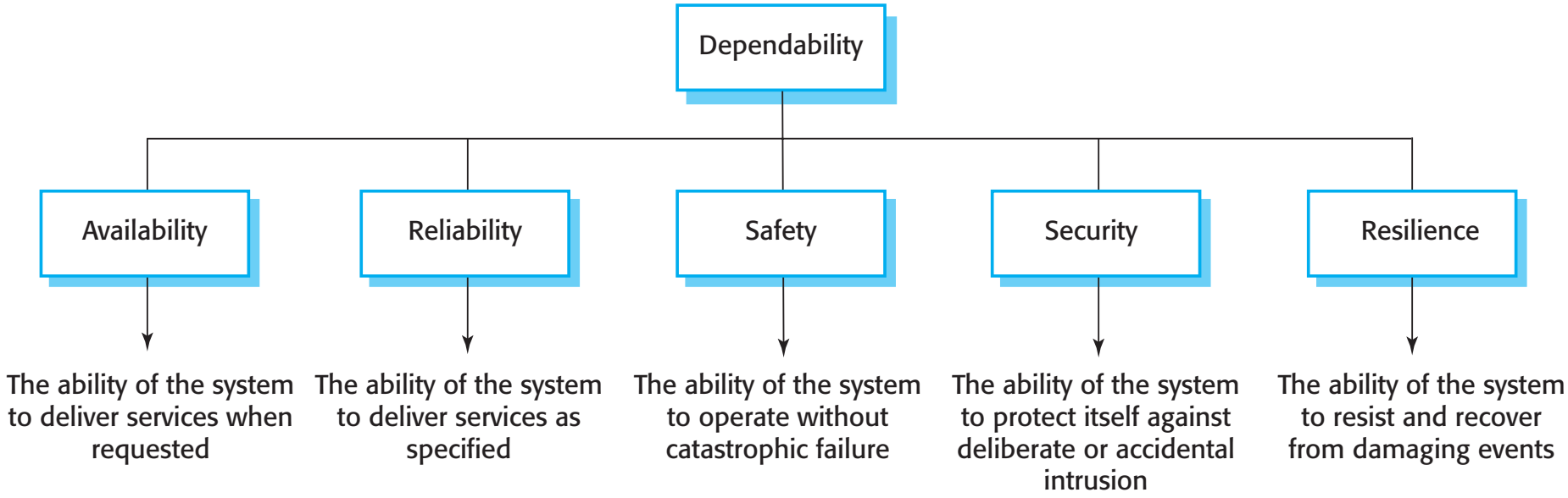
- System failures may have widespread effects with large numbers of people affected by the failure.
- Systems that are not dependable and are unreliable, unsafe or insecure may be rejected by their users.
- The costs of system failure may be very high if the failure leads to economic losses or physical damage.
- Undependable systems may cause information loss with a high consequent recovery cost.

Causes of failure

- Hardware failure
  - Hardware fails because of design and manufacturing errors or because components have reached the end of their natural life.
- Software failure
  - Software fails due to errors in its specification, design or implementation.
- Operational failure
  - Human operators make mistakes. Now perhaps the largest single cause of system failures in socio-technical systems.

Dependability properties

The principal dependability properties



Principal properties

- Availability
  - The probability that the system will be up and running and able to deliver useful services to users.
- Reliability
  - The probability that the system will correctly deliver services as expected by users.
- Safety
  - A judgment of how likely it is that the system will cause damage to people or its environment.

Principal properties

- Security
  - A judgment of how likely it is that the system can resist accidental or deliberate intrusions.
- Resilience
  - A judgment of how well a system can maintain the continuity of its critical services in the presence of disruptive events such as equipment failure and cyberattacks.

Other dependability properties

- Repairability
  - Reflects the extent to which the system can be repaired in the event of a failure
- Maintainability
  - Reflects the extent to which the system can be adapted to new requirements;
- Error tolerance
  - Reflects the extent to which user input errors can be avoided and tolerated.

Dependability attribute dependencies

- Safe system operation depends on the system being available and operating reliably.
- A system may be unreliable because its data has been corrupted by an external attack.
- Denial of service attacks on a system are intended to make it unavailable.
- If a system is infected with a virus, you cannot be confident in its reliability or safety.

Dependability achievement

- Avoid the introduction of accidental errors when developing the system.
- Design V & V processes that are effective in discovering residual errors in the system.
- Design systems to be fault tolerant so that they can continue in operation when faults occur
- Design protection mechanisms that guard against external attacks.

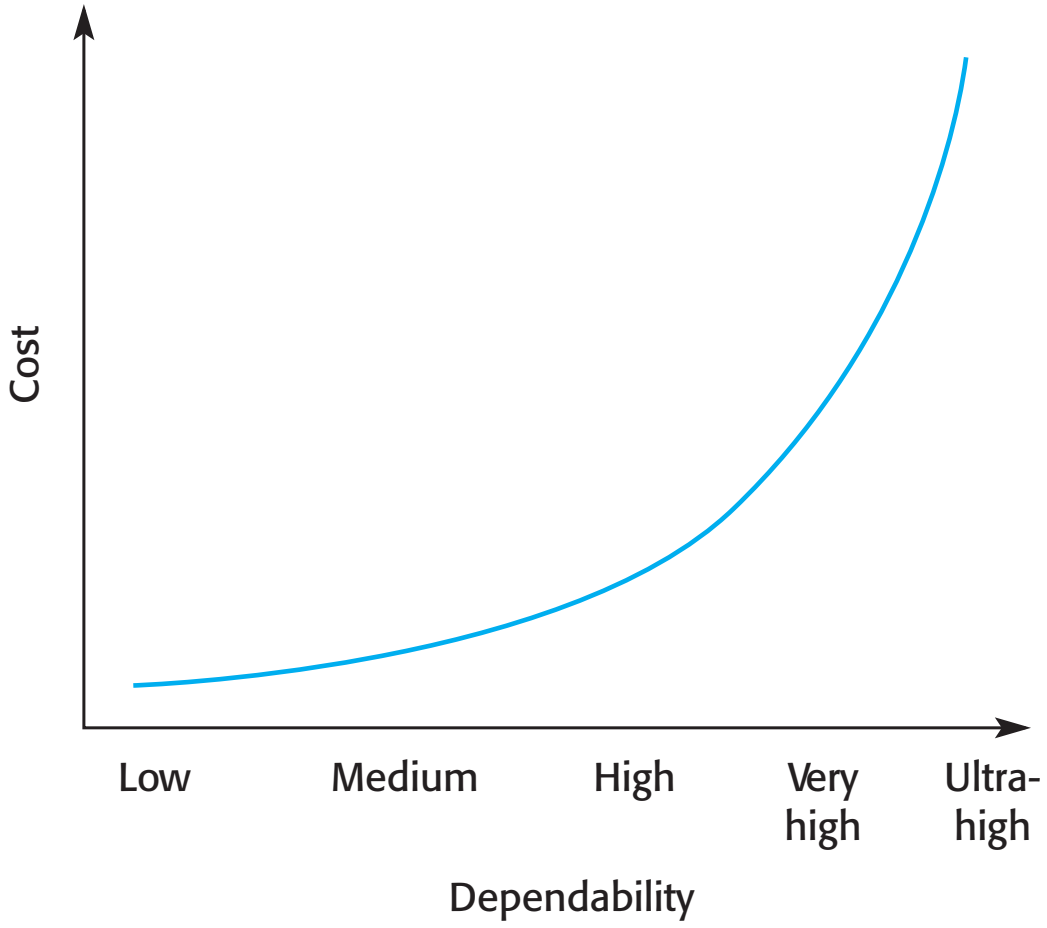
Dependability achievement

- Configure the system correctly for its operating environment.
- Include system capabilities to recognise and resist cyberattacks.
- Include recovery mechanisms to help restore normal system service after a failure.

Dependability costs

- Dependability costs tend to increase exponentially as increasing levels of dependability are required.
- There are two reasons for this
  - The use of more expensive development techniques and hardware that are required to achieve the higher levels of dependability.
  - The increased testing and system validation that is required to convince the system client and regulators that the required levels of dependability have been achieved.

Cost/dependability curve



Dependability economics

- Because of very high costs of dependability achievement, it may be more cost effective to accept untrustworthy systems and pay for failure costs
- However, this depends on social and political factors. A reputation for products that can't be trusted may lose future business
- Depends on system type - for business systems in particular, modest levels of dependability may be adequate

Redundancy and diversity

Redundancy and diversity

- Redundancy
  - Keep more than a single version of critical components so that if one fails then a backup is available.
- Diversity
  - Provide the same functionality in different ways in different components so that they will not fail in the same way.
- Redundant and diverse components should be independent so that they will not suffer from 'common-mode' failures
  - For example, components implemented in different programming languages means that a compiler fault will not affect all of them.

Diversity and redundancy examples

- Redundancy. Where availability is critical (e.g. in e-commerce systems), companies normally keep backup servers and switch to these automatically if failure occurs.
- Diversity. To provide resilience against external attacks, different servers may be implemented using different operating systems (e.g. Windows and Linux)

Process diversity and redundancy

- Process activities, such as validation, should not depend on a single approach, such as testing, to validate the system.
- Redundant and diverse process activities are important especially for verification and validation.
- Multiple, different process activities the complement each other and allow for cross-checking help to avoid process errors, which may lead to errors in the software.

Problems with redundancy and diversity

- Adding diversity and redundancy to a system increases the system complexity.
- This can increase the chances of error because of unanticipated interactions and dependencies between the redundant system components.
- Some engineers therefore advocate simplicity and extensive V & V as a more effective route to software dependability.
- Airbus FCS architecture is redundant/diverse; Boeing 777 FCS architecture has no software diversity

Dependable processes

Dependable processes

- To ensure a minimal number of software faults, it is important to have a well-defined, repeatable software process.
- A well-defined repeatable process is one that does not depend entirely on individual skills; rather can be enacted by different people.
- Regulators use information about the process to check if good software engineering practice has been used.
- For fault detection, it is clear that the process activities should include significant effort devoted to verification and validation.

Dependable process characteristics

- Explicitly defined
  - A process that has a defined process model that is used to drive the software production process. Data must be collected during the process that proves that the development team has followed the process as defined in the process model.
- Repeatable
  - A process that does not rely on individual interpretation and judgment. The process can be repeated across projects and with different team members, irrespective of who is involved in the development.

Attributes of dependable processes

Process characteristic	Description
Auditable	The process should be understandable by people apart from process participants, who can check that process standards are being followed and make suggestions for process improvement.
Diverse	The process should include redundant and diverse verification and validation activities.
Documentable	The process should have a defined process model that sets out the activities in the process and the documentation that is to be produced during these activities.
Robust	The process should be able to recover from failures of individual process activities.
Standardized	A comprehensive set of software development standards covering software production and documentation should be available.

Dependable process activities

- Requirements reviews to check that the requirements are, as far as possible, complete and consistent.
- Requirements management to ensure that changes to the requirements are controlled and that the impact of proposed requirements changes is understood.
- Formal specification, where a mathematical model of the software is created and analyzed.
- System modeling, where the software design is explicitly documented as a set of graphical models, and the links between the requirements and these models are documented.

Dependable process activities

- Design and program inspections, where the different descriptions of the system are inspected and checked by different people.
- Static analysis, where automated checks are carried out on the source code of the program.
- Test planning and management, where a comprehensive set of system tests is designed.
  - The testing process has to be carefully managed to demonstrate that these tests provide coverage of the system requirements and have been correctly applied in the testing process.

Dependable processes and agility

- Dependable software often requires certification so both process and product documentation has to be produced.
- Up-front requirements analysis is also essential to discover requirements and requirements conflicts that may compromise the safety and security of the system.
- These conflict with the general approach in agile development of co-development of the requirements and the system and minimizing documentation.

Dependable processes and agility

- An agile process may be defined that incorporates techniques such as iterative development, test-first development and user involvement in the development team.
- So long as the team follows that process and documents their actions, agile methods can be used.
- However, additional documentation and planning is essential so ‘pure agile’ is impractical for dependable systems engineering.

Key points

- System dependability is important because failure of critical systems can lead to economic losses, information loss, physical damage or threats to human life.
- The dependability of a computer system is a system property that reflects the user's degree of trust in the system. The most important dimensions of dependability are availability, reliability, safety, security and resilience.

Key points

- The use of a dependable, repeatable process is essential if faults in a system are to be minimized. The process should include verification and validation activities at all stages, from requirements definition through to system implementation.
- The use of redundancy and diversity in hardware, software processes and software systems is essential to the development of dependable systems.

About Week 8 – March 9th

- TEAM Homework Assignments
  - HW7 - Test Plan. Deliver a .pdf document that clearly defines your test **methodologies** for your product and test **plans**. 50 points / 43 base
    - **Methodology** – approach to how you are going to do a particular test (Unit, Component, System). Example – use of JUnit, testing of component / class design info, use case system test or requirements-based system test, etc.
    - **Plans** – Expansion of your project plan specifically about all your test activities – WHO, WHAT and WHEN of all test-related work.
    - You don't need to have all your specific test cases defined yet but at least an outline of what they will be. We will review the actual test cases in checkpoint meetings and final project deliverable.
    - You should explicitly define how you are approaching Unit Test, Component Test, System / Release Test, and others if applicable (like Performance Test or Usability Test). These should be the major sections of your Test Plan.
    - Grade will be based on completeness and granularity of test design, clarity of everyone's role and responsibilities, and creativeness in design of your test plan
    - Due March 21st
  - HW 8 – Checkpoint Meetings. It is up to you to schedule. Up to ten points earned for each meeting held on time. 3 meetings for total of 30 points.
    - First checkpoint with TA – DUE March 22nd
    - Checkpoint with Me – AFTER March 15th but DUE March 31st
    - Second checkpoint with TA – DUE April 15th
    - No need to respond in Sakai – we will track and grade.