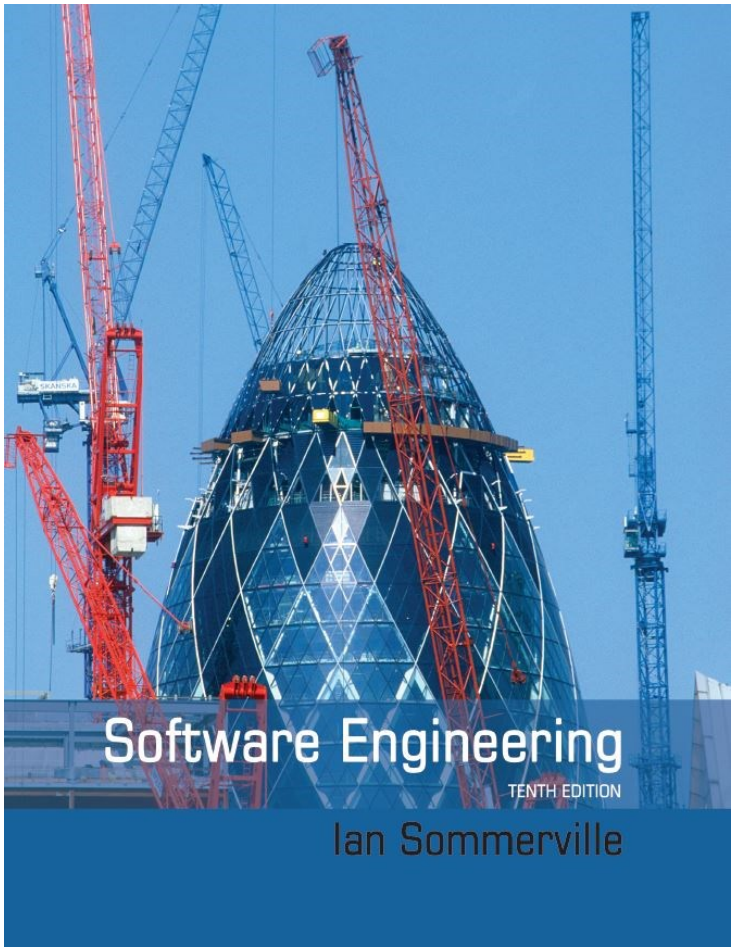
Software Engineering

Tenth Edition



Chapter 10

Dependable systems

Learning Objectives

10.1 Dependability properties

10.2 Sociotechnical systems

10.3 Redundancy and diversity

10.4 Dependable processes

10.5 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, safety and security. These are all inter-dependent.

Importance of Dependability

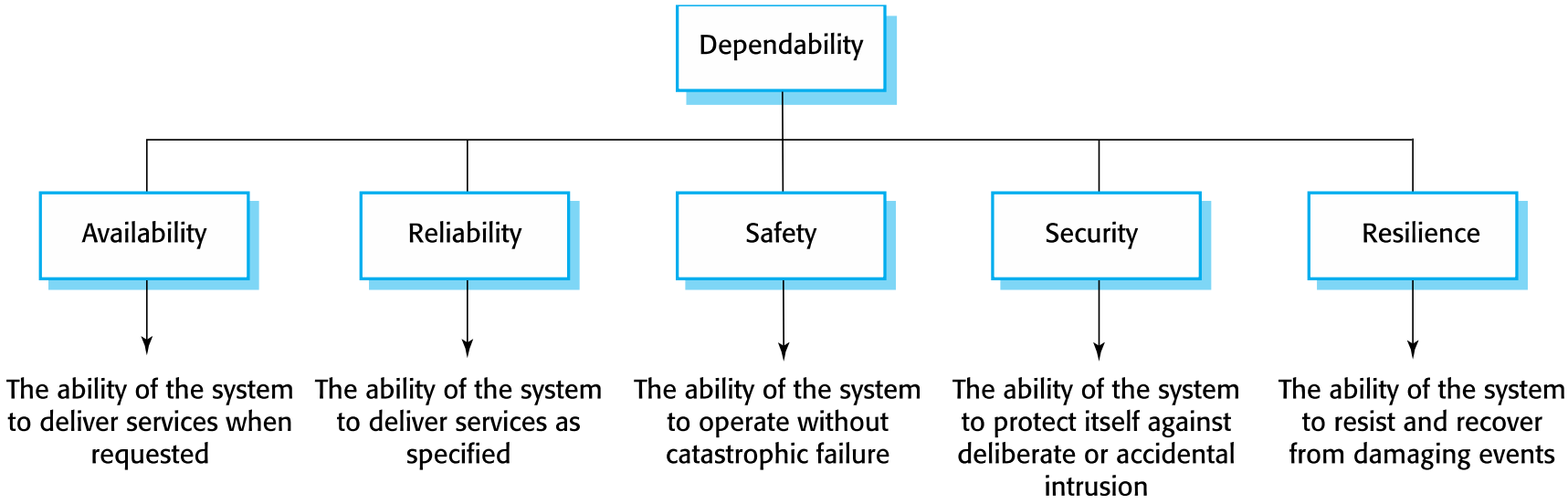
- System failures may have widespread effects with large numbers of people affected by the failure.
- Systems that are not dependable and are unreliable, unsafe or insecure may be rejected by their users.
- The costs of system failure may be very high if the failure leads to economic losses or physical 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 users may fail to use or operate the system as intended by its designers. 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.

Principal Properties

- The dependability properties are complex properties that can be broken down into several simpler properties:
  - security includes “integrity” (ensuring that the systems program and data are not damaged) and “confidentiality” (ensuring that information can only be accessed by people who are authorized).
  - Reliability includes “correctness” (ensuring the system services are as specified), “precision” (ensuring information is delivered at an appropriate level of detail), and “timeliness” (ensuring that information is delivered when it is required).

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 recognize and resist cyberattacks.
- Include recovery mechanisms to help restore normal system service after a failure or cyberattacks without the loss of critical data.

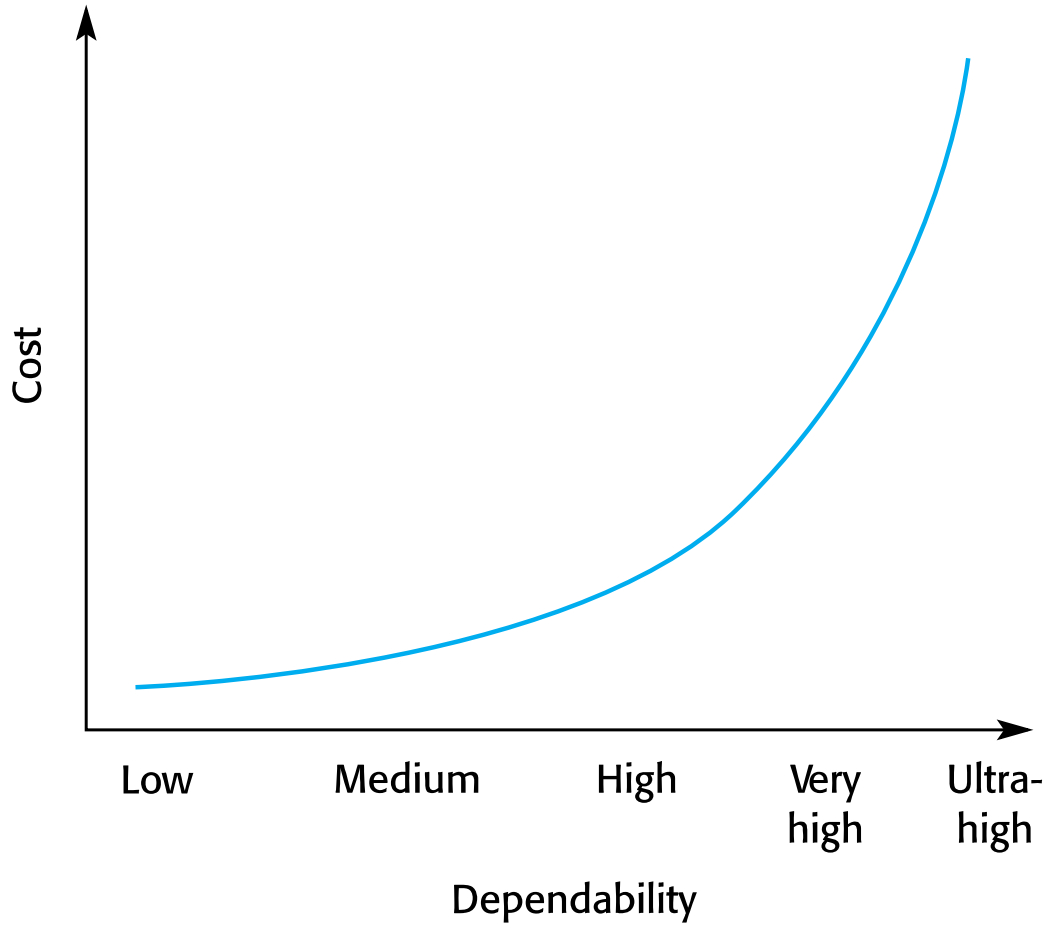
Dependable Systems

- The need for fault tolerance means that dependable systems have to include redundant code to help them monitor themselves, detect erroneous states, and recover from faults before failures occur.
- Therefore, designers usually have to trade off performance and dependability.
- You may need to leave checks out of the system because these slow the system down.

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

Sociotechnical Systems

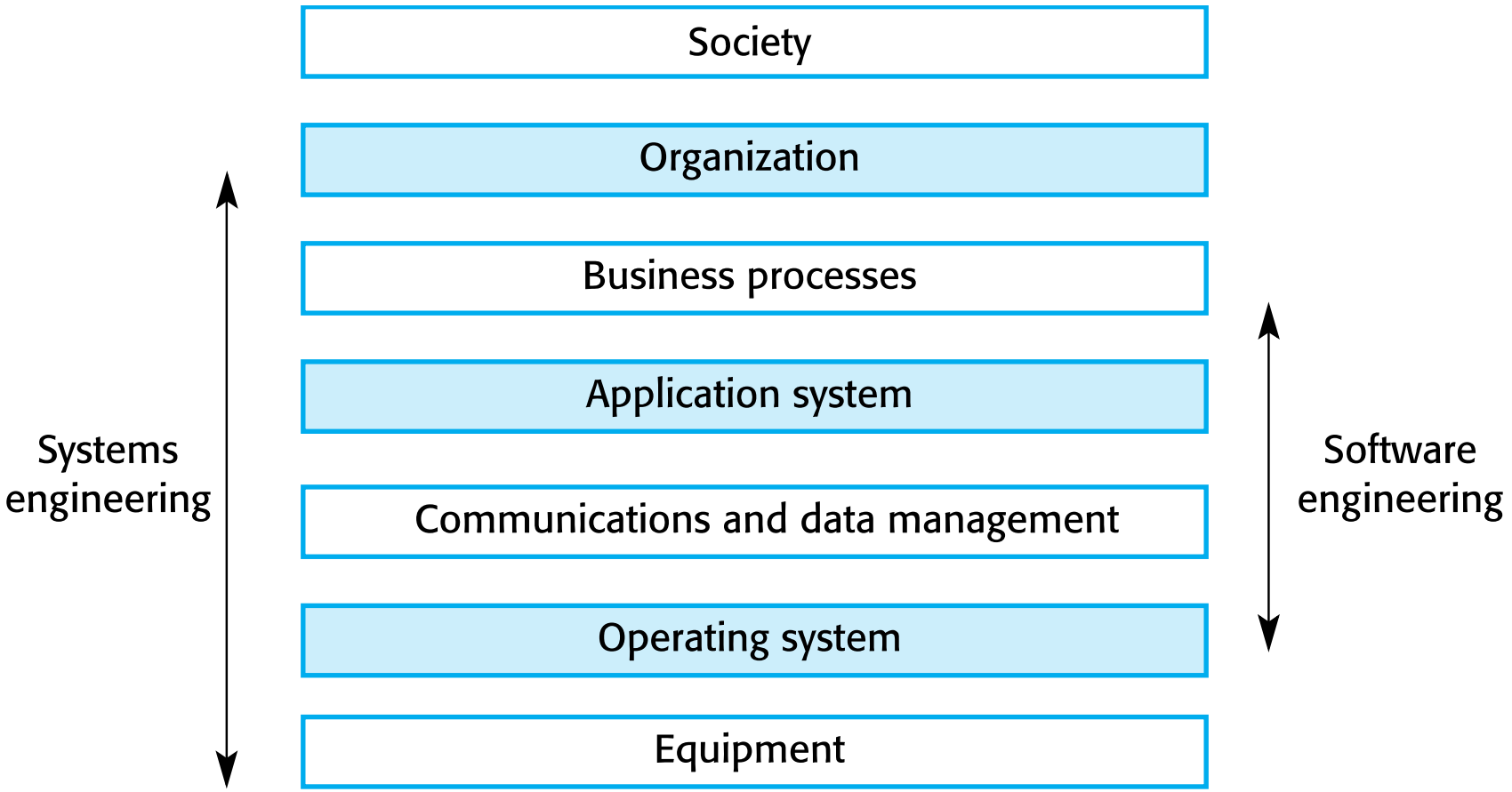
Systems and Software

- Software systems are not isolated systems but are essential components of broader systems that have a human, social or organizational purpose.
- Software engineering is not an isolated activity but is part of a broader systems engineering process.
- Example
  - The wilderness weather system is part of broader weather recording and forecasting systems
  - These include hardware and software, forecasting processes, system users, the organizations that depend on weather forecasts, etc.

Sociotechnical Systems

- Sociotechnical Systems include nontechnical elements such as people, processes, and regulations, as well as technical components such as computers, software, and other equipment.
- System dependability is influenced by all of the elements in a sociotechnical system—hardware, software, people, and organizations.

The Sociotechnical Systems Stack



Layers in the STS Stack

- Equipment
  - Hardware devices, some of which may be computers. Most devices will include an embedded system of some kind.
- Operating system
  - Provides a set of common facilities for higher levels in the system.
- Communications and data management
  - Middleware that provides access to remote systems and databases.
- Application systems
  - Specific functionality to meet some organization requirements.

Layers in the STS Stack

- Business processes
  - A set of processes involving people and computer systems that support the activities of the business.
- Organizations
  - Higher level strategic business activities that affect the operation of the system.
- Society
  - Laws, regulation and culture that affect the operation of the system.

Holistic System Design

- Thinking holistically about systems, rather than simply considering software in isolation, is essential when considering software security and dependability.
- There are interactions and dependencies between the layers in a system and changes at one level ripple through the other levels.
- You must, therefore, take a system-level view when you are designing software that has to be dependable and secure.

Holistic System Design

- You must therefore examine how the software interacts with its immediate environment to ensure that:
  - Software failures are, as far as possible, contained within the enclosing layer of the system stack and do not seriously affect the operation of other layers in the system.
  - You understand how faults and failures in the other layers of the systems stack may affect the software.

Holistic System Design

- As software is inherently flexible, unexpected system problems are often left to software engineers to solve.
- Software engineers are left with the problem of enhancing software capabilities without increasing hardware cost, is very common.
- Many so-called software failures are not a consequence of inherent software problems but rather are the result of trying to change the software to accommodate modified system engineering requirements.

Regulation and Compliance

- The general model of economic organization that is now almost universal in the world is that privately owned companies offer goods and services and make a profit on these.
- To ensure the safety of their citizens, most governments regulate (limit the freedom of) privately owned companies so that they must follow certain standards to ensure that their products are safe and secure.

Regulated Systems

- Many critical systems are regulated systems, which means that their use must be approved by an external regulator before the systems go into service.
  - Nuclear systems
  - Air traffic control systems
  - Medical devices
- A safety and dependability case has to be approved by the regulator. Therefore, critical systems development has to create the evidence to convince a regulator that the system is dependable, safe and secure.

Safety Regulation

- Regulation and compliance (following the rules) applies to the sociotechnical system as a whole and not simply the software element of that system.
- Safety-related systems may have to be certified as safe by the regulator.
- To achieve certification, companies that are developing safety-critical systems have to produce an extensive safety case that shows that rules and regulations have been followed.
- It can be as expensive develop the documentation for certification as it is to develop the system itself.

Redundancy and Diversity

Redundancy and Diversity

- Component failures in any system are inevitable.
- We use a range of strategies to reduce the number of human failures.
- However, we cannot be sure that these will eliminate component failures. We should therefore design systems so that individual component failures do not lead to overall system failure.
- Strategies to achieve and enhance dependability rely on both 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 that 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.
- For fault detection, it is clear that the process activities should include significant effort devoted to verification and validation.
- Regulators use information about the process to check if good software engineering practice has been used.

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 Processes

- Dependable processes make use of redundancy and diversity to achieve reliability.
- They often include different activities that have the same aim.
- The activities that are used in dependable processes obviously depend on the type of software that is being developed.

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 Process Activities

- Process activities must also be well-defined quality management and change management processes.
- Quality management processes establish a set of process and product standards. They also include activities that capture process information to demonstrate that these standards have been followed.
- Change management is concerned with managing changes to a system, ensuring that accepted changes are actually implemented, and confirming that planned releases of the software include the planned changes.

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.

Formal Methods and Dependability

Formal Specification

- Formal methods are approaches to software development that are based on mathematical representation and analysis of software.
- You may then formally analyze this model to search for errors and inconsistencies
- There is a claim that the use of formal methods can lead to “faultless systems”.
- Formal methods significantly reduce some types of programming errors and can be cost-effective for dependable systems engineering.

Formal Approaches

- Verification-based approaches
  - Different representations of a software system such as a specification and a program implementing that specification are proved to be equivalent.
  - This demonstrates the absence of implementation errors.
  - However, developing the proof obligations for theorem provers is a difficult and specialized task, so formal verification is not widely used.

Formal Approaches

- Refinement-based approaches
  - A representation of a system is systematically transformed into another lower-level representation e.g. a specification is transformed automatically into an implementation.
  - This means that, if the transformation is correct, the representations are equivalent.
  - Because these are trusted transformations, you can be confident that the generated program is consistent with its formal specification.

Use of Formal Methods

- The principal benefits of formal methods are in reducing the number of faults in systems.
- Consequently, their main area of applicability is in dependable systems engineering. There have been several successful projects where formal methods have been used in this area.
- In this area, the use of formal methods is most likely to be cost-effective because high system failure costs must be avoided.

Classes of Error

- Specification and design errors and omissions.
  - Developing and analysing a formal model of the software may reveal errors and omissions in the software requirements. If the model is generated automatically or systematically from source code, analysis using model checking can find undesirable states that may occur such as deadlock in a concurrent system.
- Inconsistences between a specification and a program.
  - If a refinement method is used, mistakes made by developers that make the software inconsistent with the specification are avoided. Program proving discovers inconsistencies between a program and its specification.

Formal Methods

- The starting point for all formal methods is a mathematical system model, which acts as a system specification.
- To create this model, you translate the system's user requirements.
- The formal specification is an unambiguous description of what the system should do.
- Formal specifications are the most precise way of specifying systems and so reduce the scope for misunderstanding.

Benefits of Formal Specification

- Developing a formal specification requires the system requirements to be analyzed in detail. Requirements problems are discovered early.
- As the specification is expressed in a formal language, it can be automatically analyzed to discover inconsistencies and incompleteness.
- If you use a formal method such as the B method, you can transform the formal specification into a ‘correct’ program.
- Program testing costs may be reduced if the program is formally verified against its specification.

Acceptance of Formal Methods

- Formal methods have had limited impact on practical software development:
  - Problem owners cannot understand a formal specification and so cannot assess if it is an accurate representation of their requirements.
  - It is easy to assess the costs of developing a formal specification but harder to assess the benefits. Managers may therefore be unwilling to invest in formal methods.
  - Software engineers are unfamiliar with this approach and are therefore reluctant to propose the use of FM.
  - Formal methods are still hard to scale up to large systems.
  - Tool support for formal methods is limited.
  - Formal specification is not really compatible with agile development methods.

Key Points (1 of 2)

- 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.
- Sociotechnical systems include computer hardware, software and people, and are situated within an organization. They are designed to support organizational or business goals and objectives.

Key Points (2 of 2)

- 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.
- Formal methods, where a formal model of a system is used as a basis for development help reduce the number of specification and implementation errors in a system.

Copyright