Dependability and Reliability

Idaho State University | Computer Science

Isaac Griffith

CS 3321
Department of Computer Science
Idaho State University

ROAR

Topics Covered

- Dependable processes
- Formal methods and dependability
- Availability and reliability

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

- **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.
- **Standarized** - 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.

Formal methods and dependability

Formal specification

- Formal methods are approaches to software development that are based on mathematical representation and analysis of software.
- Formal methods include
  - Formal specification;
  - Specification analysis and proof;
  - Transformational development;
  - Program verification.
- 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.
- Refinement-based approaches
  - A representation of a system is systematically transformed into another, lower-level represention e.g. a specification is transformed automatically into an implementation.
  - This means that, if the transformation is correct, the representations are equivalent.

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 analyzing 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.
- Inconsistencies 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.

Benefits of formal specification

- Developing a formal specification requires the system requirements to be analyzed in detail. This helps to detect problems, inconsistencies and incompleteness in the requirements.
- 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.
  - Formal specification is not really compatible with agile development methods.

Software Reliability

Software reliability

- In general, software customers expect all software to be dependable. However, for non-critical applications, they may be willing to accept some system failures.
- Some applications (critical systems) have very high reliability requirements and special software engineering techniques may be used to achieve this.
  - Medical systems
  - Telecommunications and power systems
  - Aerospace systems

Faults, errors and failures

- **Human error or mistake** - Human behavior that results in the introduction of faults into a system. For example, in the wilderness weather system, a programmer might decide that the way to compute the time for the next transmission is to add 1 hour to the current time. This works except when the transmission time is between 23.00 and midnight (midnight is 00.00 in the 24-hour clock).
- **System fault** - A characteristic of a software system that can lead to a system error. The fault is the inclusion of the code to add 1 hour to the time of the last transmission, without a check if the time is greater than or equal to 23.00.

Faults, errors and failures

- **System error** - An erroneous system state that can lead to system behavior that is unexpected by system users. The value of transmission time is set incorrectly (to 24.XX rather than 00.XX) when the faulty code is executed.
- **System failure** - An event that occurs at some point in time when the system does not deliver a service as expected by its users. No weather data is transmitted because the time is invalid.

Faults and failures

- Failures are a usually a result of system errors that are derived from faults in the system
- However, faults do not necessarily result in system errors
  - The erroneous system state resulting from the fault may be transient and 'corrected' before an error arises.
  - The faulty code may never be executed.
- Errors do not necessarily lead to system failures
  - The error can be corrected by built-in error detection and recovery
  - The failure can be protected against by built-in protection facilities. These may, for example, protect system resources from system errors

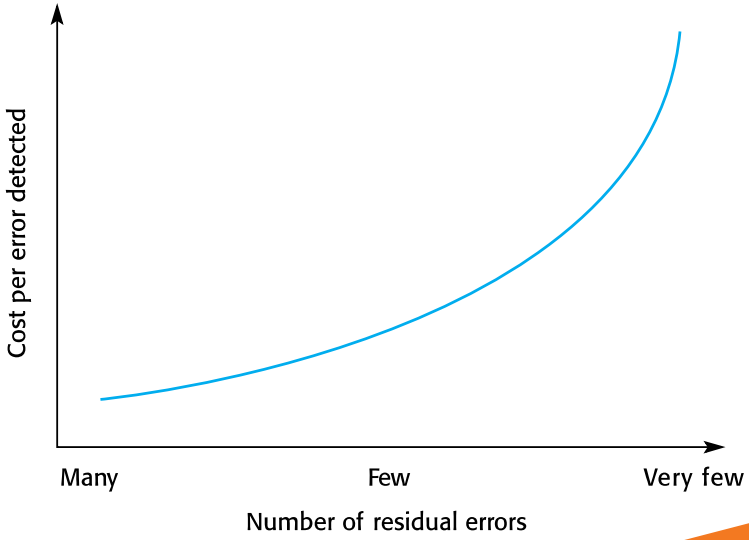
Fault management

- Fault avoidance
  - The system is developed in such a way that human error is avoided and thus system faults are minimized.
  - The development process is organized so that faults in the system are detected and repaired before delivery to the customer.
- Fault detection
  - Verification and validation techniques are used to discover and remove faults in a system before it is deployed.
- Fault tolerance
  - The system is designed so that faults in the delivered software do not result in system failure.

Reliability achievement

- Fault avoidance
  - Development technique are used that either minimize the possibility of mistakes or trap mistakes before they result in the introduction of system faults.
- Fault detection and removal
  - Verification and validation techniques are used that increase the probability of detecting and correcting errors before the system goes into service are used.
- Fault tolerance
  - Run-time techniques are used to ensure that system faults do not result in system errors and/or that system errors do not lead to system failures.

The increasing costs of residual fault removal



Availability and reliability

Availability and reliability

- Reliability
  - The probability of failure-free system operation over a specified time in a given environment for a given purpose
- Availability
  - The probability that a system, at a point in time, will be operational and able to deliver the requested services
- Both of these attributes can be expressed quantitatively e.g. availability of 0.999 means that the system is up and running for 99.9% of the time.

Reliability and specifications

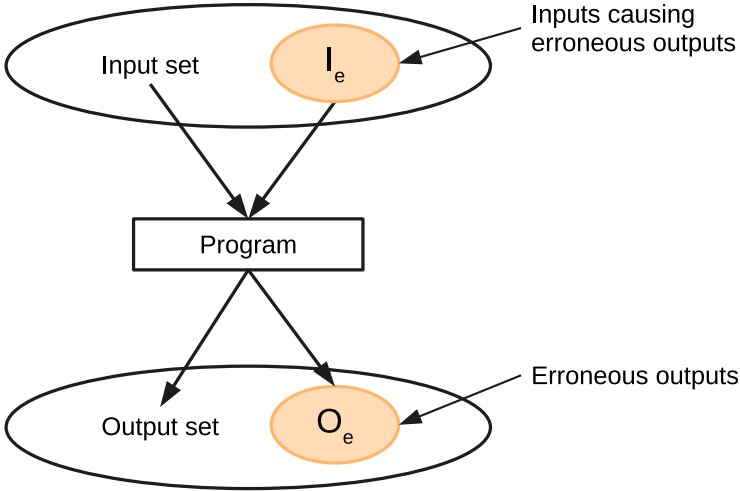
- Reliability can only be defined formally with respect to a system specification i.e. a failure is a deviation from a specification.
- However, many specifications are incomplete or incorrect – hence, a system that conforms to its specification may 'fail' from the perspective of system users.
- Furthermore, users don't read specifications so don't know how the system is supposed to behave.
- Therefore perceived reliability is more important in practice.

Perceptions of reliability

- The formal definition of reliability does not always reflect the user's perception of a system's reliability
  - The assumptions that are made about the environment where a system will be used may be incorrect
    - Usage of a system in an office environment is likely to be quite different from usage of the same system in a university environment
  - The consequences of system failures affects the perception of reliability
    - Unreliable windscreen wipers in a car may be irrelevant in a dry climate
    - Failures that have serious consequences (such as an engine breakdown in a car) are given greater weight by users than failures that are inconvenient

Idaho State University
Computer Science

An input/output mapping

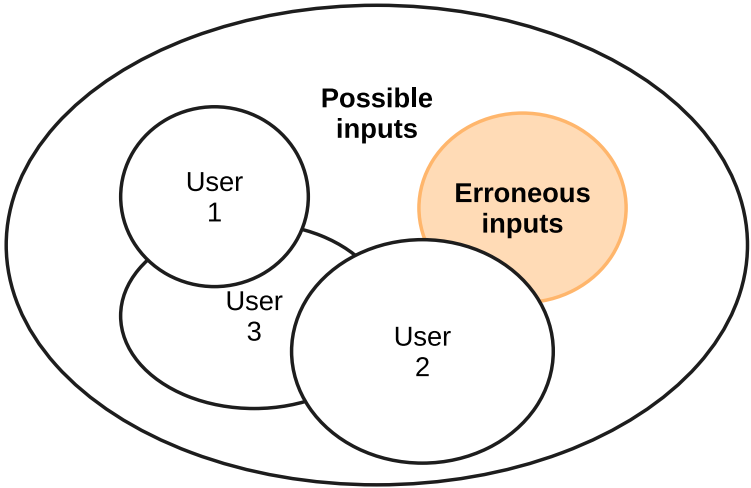


ROAR

Availability perception

- Availability is usually expressed as a percentage of the time that the system is available to deliver services e.g. 99.95%.
- However, this does not take into account two factors:
  - The number of users affected by the service outage. Loss of service in the middle of the night is less important for many systems than loss of service during peak usage periods.
  - The length of the outage. The longer the outage, the more the disruption. Several short outages are less likely to be disruptive than 1 long outage. Long repair times are a particular problem.

Software usage patterns



Reliability in use

- Removing X% of the faults in a system will not necessarily improve the reliability by X%.
- Program defects may be in rarely executed sections of the code so may never be encountered by users. Removing these does not affect the perceived reliability.
- Users adapt their behavior to avoid system features that may fail for them.
- A program with known faults may therefore still be perceived as reliable by its users.

Key points

- 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.
- Functional reliability requirements are requirements for system functionality, such as checking and redundancy requirements, which help the system meet its non-functional reliability requirements.

Are there any questions?