Reliability

Idaho State University | Computer Science

Isaac Griffith

CS 3321
Department of Computer Science
Idaho State University

ROAR

Topics Covered

- Programming for reliability
- Reliability measurement

Programming for reliability

Dependable programming

- Good programming practices can be adopted that help reduce the incidence of program faults.
- These programming practices support
  - Fault avoidance
  - Fault detection
  - Fault tolerance

Dependable programming guidelines

Dependable programming guidelines

❶ Limit the visibility of information in a program

❷ Check all inputs for validity

❸ Provide a handler for all exceptions

❹ Minimize the use of error-prone constructs

❺ Provide restart capabilities

❻ Check array bounds

❼ Include timeouts when calling external components

❽ Name all constants that represent real-world values

(1) Limit the visibility of information in a program

- Program components should only be allowed access to data that they need for their implementation.
- This means that accidental corruption of parts of the program state by these components is impossible.
- You can control visibility by using abstract data types where the data representation is private and you only allow access to the data through predefined operations such as get () and put ().

(2) Check all inputs for validity

- All program take inputs from their environment and make assumptions about these inputs.
- However, program specifications rarely define what to do if an input is not consistent with these assumptions.
- Consequently, many programs behave unpredictably when presented with unusual inputs and, sometimes, these are threats to the security of the system.
- Consequently, you should always check inputs before processing against the assumptions made about these inputs.

Validity checks

- Range checks
  - Check that the input falls within a known range.
- Size checks
  - Check that the input does not exceed some maximum size e.g. 40 characters for a name.
- Representation checks
  - Check that the input does not include characters that should not be part of its representation e.g. names do not include numerals.
- Reasonableness checks
  - Use information about the input to check if it is reasonable rather than an extreme value.

(3) Provide a handler for all exceptions

- A program exception is an error or some unexpected event such as a power failure.
- Exception handling constructs allow for such events to be handled without the need for continual status checking to detect exceptions.
- Using normal control constructs to detect exceptions needs many additional statements to be added to the program. This adds a significant overhead and is potentially error-prone.

Exception handling

- Three possible exception handling strategies
  - Signal to a calling component that an exception has occurred and provide information about the type of exception.
  - Carry out some alternative processing to the processing where the exception occurred. This is only possible where the exception handler has enough information to recover from the problem that has arisen.
  - Pass control to a run-time support system to handle the exception.
- Exception handling is a mechanism to provide some fault tolerance

(4) Minimize the use of error-prone constructs

- Program faults are usually a consequence of human error because programmers lose track of the relationships between the different parts of the system
- This is exacerbated by error-prone constructs in programming languages that are inherently complex or that don’t check for mistakes when they could do so.
- Therefore, when programming, you should try to avoid or at least minimize the use of these error-prone constructs.

Error-prone constructs

- Unconditional branch (goto) statements
- Floating-point numbers
  - Inherently imprecise. The imprecision may lead to invalid comparisons.
- Pointers
  - Pointers referring to the wrong memory areas can corrupt data. Aliasing can make programs difficult to understand and change.
- Dynamic memory allocation
  - Run-time allocation can cause memory overflow.

Error-prone constructs

- Parallelism
  - Can result in subtle timing errors because of unforeseen interaction between parallel processes.
- Recursion
  - Errors in recursion can cause memory overflow as the program stack fills up.
- Interrupts
  - Interrupts can cause a critical operation to be terminated and make a program difficult to understand.
- Inheritance
  - Code is not localized. This can result in unexpected behavior when changes are made and problems of understanding the code.

Error-prone constructs

- Aliasing
  - Using more than 1 name to refer to the same state variable.
- Unbounded arrays
  - Buffer overflow failures can occur if no bound checking on arrays.
- Default input processing
  - An input action that occurs irrespective of the input.
  - This can cause problems if the default action is to transfer control elsewhere in the program. In incorrect or deliberately malicious input can then trigger a program failure.

(5) Provide restart capabilities

- For systems that involve long transactions or user interactions, you should always provide a restart capability that allows the system to restart after failure without users having to redo everything that they have done.
- Restart depends on the type of system
  - Keep copies of forms so that users don’t have to fill them in again if there is a problem
  - Save state periodically and restart from the saved state

(6) Check array bounds

- In some programming languages, such as C, it is possible to address a memory location outside of the range allowed for in an array declaration.
- This leads to the well-known ‘bounded buffer’ vulnerability where attackers write executable code into memory by deliberately writing beyond the top element in an array.
- If your language does not include bound checking, you should therefore always check that an array access is within the bounds of the array.

(7) Include timeouts when calling external components

- In a distributed system, failure of a remote computer can be ‘silent’ so that programs expecting a service from that computer may never receive that service or any indication that there has been a failure.
- To avoid this, you should always include timeouts on all calls to external components.
- After a defined time period has elapsed without a response, your system should then assume failure and take whatever actions are required to recover from this.

(8) Name all constants that represent real-world values

- Always give constants that reflect real-world values (such as tax rates) names rather than using their numeric values and always refer to them by name
- You are less likely to make mistakes and type the wrong value when you are using a name rather than a value.
- This means that when these ‘constants’ change (for sure, they are not really constant), then you only have to make the change in one place in your program.

Reliability measurement

Reliability measurement

- To assess the reliability of a system, you have to collect data about its operation. The data required may include:
  - The number of system failures given a number of requests for system services. This is used to measure the POFOD. This applies irrespective of the time over which the demands are made.
  - The time or the number of transactions between system failures plus the total elapsed time or total number of transactions. This is used to measure ROCOF and MTTF.
  - The repair or restart time after a system failure that leads to loss of service. This is used in the measurement of availability. Availability does not just depend on the time between failures but also on the time required to get the system back into operation.

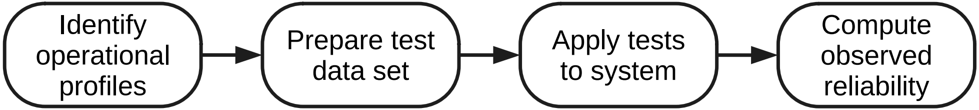
Reliability testing

- Reliability testing (Statistical testing) involves running 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.

Reliability measurement


Identify operational profiles

Prepare test data set

Apply tests to system

Compute observed reliability

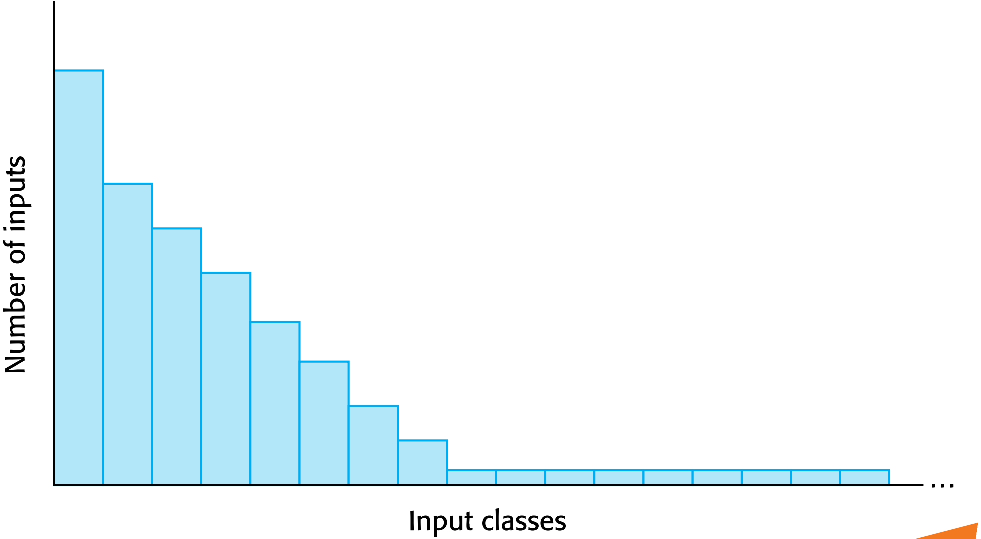
Reliability measurement problems

- 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.
- Recognizing failure
  - It is not always obvious when a failure has occurred as there may be conflicting interpretations of a specification.

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.

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.
- Pattern of usage of new systems is unknown.
- Operational profiles are not static but change as users learn about a new system and change the way that they use it.

Key points

- Dependable programming relies on including redundancy in a program as checks on the validity of inputs and the values of program variables.
- Statistical testing is used to estimate software reliability. It relies on testing the system with test data that matches an operational profile, which reflects the distribution of inputs to the software when it is in use.
- Software reliability can be achieved by avoiding the introduction of faults, by detecting and removing faults before system deployment and by including fault tolerance facilities that allow the system to remain operational after a fault has caused a system failure.
- Reliability requirements can be defined quantitatively in the system requirements specification.

Key points

- Reliability metrics include probability of failure on demand (POFOD), rate of occurrence of failure (ROCOF) and availability (AVAIL).
- Functional reliability requirements are requirements for system functionality, such as checking and redundancy requirements, which help the system meet its non-functional reliability requirements.
- Dependable system architectures are system architectures that are designed for fault tolerance.
- There are a number of architectural styles that support fault tolerance including protection systems, self-monitoring architectures and N-version programming.

Key points

- Software diversity is difficult to achieve because it is practically impossible to ensure that each version of the software is truly independent.
- Dependable programming relies on including redundancy in a program as checks on the validity of inputs and the values of program variables.
- Statistical testing is used to estimate software reliability. It relies on testing the system with test data that matches an operational profile, which reflects the distribution of inputs to the software when it is in use.

Are there any questions?