Lecture 11 – Dependability Design (Sommerville Ch. 18)

Karl R. Wilcox
K.R.Wilcox@reading.ac.uk

Software dependability

- 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, however, have very high dependability requirements and special programming techniques must be used to achieve this

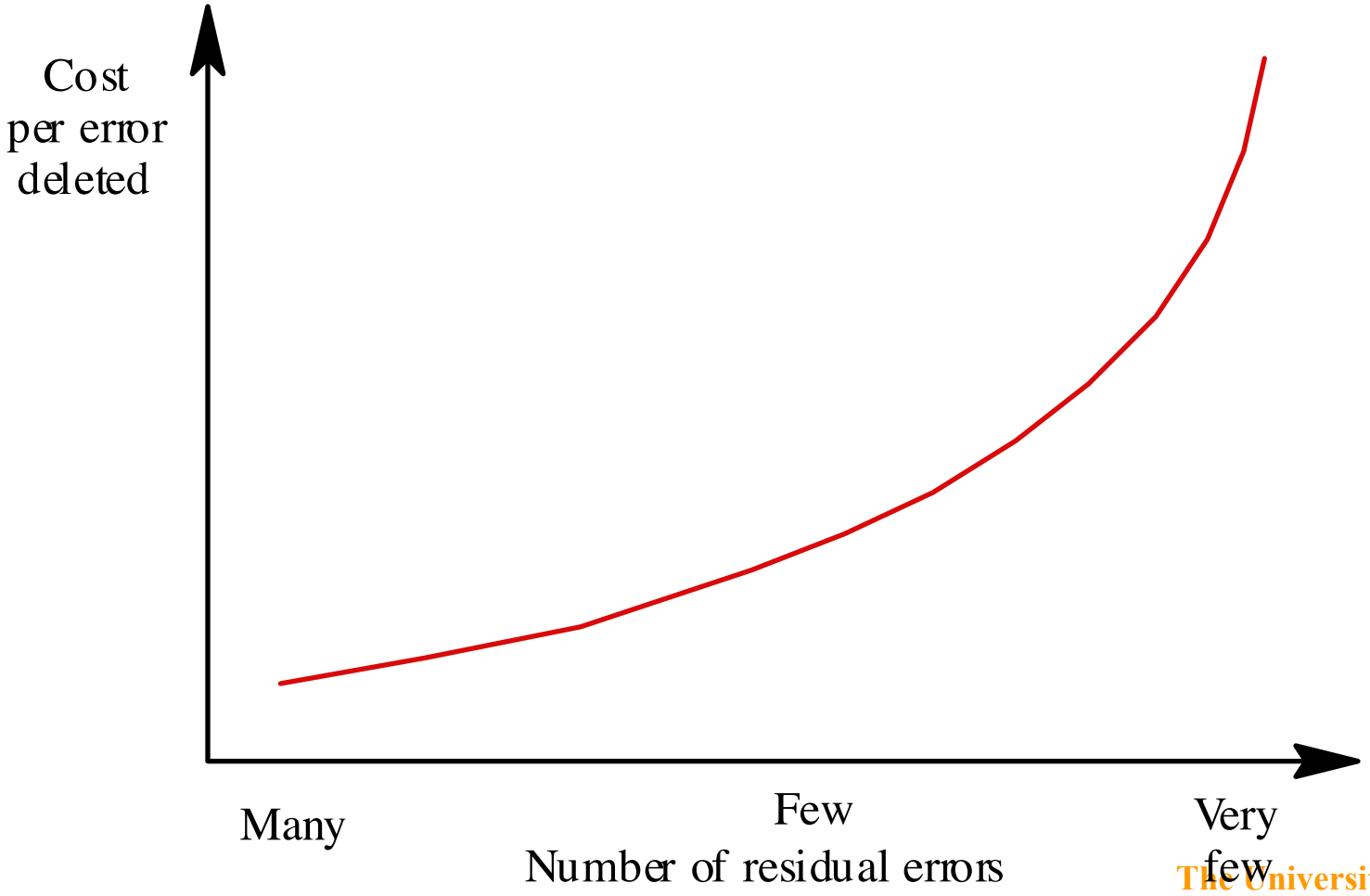
Dependability achievement

- **Fault avoidance**
  - **The software is developed in such a way that human error is avoided and thus system faults are minimised**
  - **The development process is organised so that faults in the software are detected and repaired before delivery to the customer**
- **Fault tolerance**
  - **The software is designed so that faults in the delivered software do not result in system failure**

Fault minimisation

- Current methods of software engineering now allow for the production of fault-free software.
- Fault-free software means software which conforms to its specification. It does NOT mean software which will always perform correctly as there may be specification errors.
- The cost of producing fault free software is very high. It is only cost-effective in exceptional situations. May be cheaper to accept software faults

Fault removal costs



Fault-free software development

- Needs a precise (preferably formal) specification.
- Requires an organizational commitment to quality.
- Information hiding and encapsulation in software design is essential
- A programming language with strict typing and run-time checking should be used
- Error-prone constructs should be avoided
- Dependable and repeatable development process

Structured programming

- First discussed in the 1970's
- Programming without gotos
- While loops and if statements as the only control statements.
- Top-down design.
- Important because it promoted thought and discussion about programming
- Leads to programs that are easier to read and understand

Error-prone constructs

- **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**
- **Parallelism**
  - **Can result in subtle timing errors because of unforseen interaction between parallel processes**

Error-prone constructs

- Recursion
  - Errors in recursion can cause memory overflow
- Interrupts
  - Interrupts can cause a critical operation to be terminated and make a program difficult to understand. they are comparable to goto statements.
- Inheritance
  - Code is not localised. This can result in unexpected behaviour when changes are made and problems of understanding
- These constructs don't have to be avoided but they must be used with great care.

Information hiding

- Information should only be exposed to those parts of the program which need to access it. This involves the creation of objects or abstract data types which maintain state and operations on that state
- This avoids faults for three reasons:
  - the probability of accidental corruption of information
  - the information is surrounded by 'firewalls' so that problems are less likely to spread to other parts of the program
  - as all information is localised, the programmer is less likely to make errors and reviewers are more likely to find errors

A queue specification in Java

```
interface Queue {

        public void put (Object o) ;
        public void remove (Object o) ;
        public int size () ;

} //Queue
```

Reliable software 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 minimisation, it is clear that the process activities should include significant verification and validation

Process validation activities

- Requirements inspections
- Requirements management
- Model checking
- Design and code inspection
- Static analysis
- Test planning and management
- Configuration management is also essential

Fault tolerance

- In critical situations, software systems must be fault tolerant. Fault tolerance is required where there are high availability requirements or where system failure costs are very high..
- Fault tolerance means that the system can continue in operation in spite of software failure
- Even if the system seems to be fault-free, it must also be fault tolerant as there may be specification errors or the validation may be incorrect

Fault tolerance actions

- **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.

Approaches to fault tolerance

- **Defensive programming**
  - **Programmers assume that there may be faults in the code of the system and incorporate redundant code to check the state after modifications to ensure that it is consistent.**
  - **Fault-tolerant architectures**
  - **Hardware and software system architectures that support hardware and software redundancy and a fault tolerance controller that detects problems and supports fault recovery**
  - **These are complementary rather than opposing techniques**

Exception management

- 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 in a sequence of nested procedure calls needs many additional statements to be added to the program and adds a significant timing overhead.

Exceptions in Java

```
class SensorFailureException extends Exception {
        SensorFailureException (String msg) {
                super (msg) ;
                Alarm.activate (msg) ;
        }
} // SensorFailureException

class Sensor {
        int readVal () throws SensorFailureException {
        try {
                int theValue = DeviceIO.readInteger () ;
                if (theValue < 0)
                        throw new SensorFailureException ("Sensor failure") ;
                return theValue ;
        }
        catch (deviceIOException e)
                { throw new SensorFailureException (“ Sensor read error ”) ; }
        } // readVal
} // Sensor
```

Fault detection

- Languages such as Java and Ada have a strict type system that allows many errors to be trapped at compile-time
- However, some classes of error can only be discovered at run-time
- Fault detection involves detecting an erroneous system state and throwing an exception to manage the detected fault

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. Used when a incorrect sequence of correct actions leads to an erroneous state or when preventative fault detection involves too much overhead**

Damage assessment

- Analyse system state to judge the extent of corruption caused by a system failure
- Must assess what parts of the state space have been affected by the failure
- Generally based on ‘validity functions’ which 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

- 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 filesystem 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

Key points

- Fault tolerant software can continue in execution in the presence of software faults
- Fault tolerance requires failure detection, damage assessment, recovery and repair
- Defensive programming is an approach to fault tolerance that relies on the inclusion of redundant checks in a program
- Exception handling facilities simplify the process of defensive programming