Structuring Redundancy for Fault Tolerance

CSE 598D: Fault Tolerant Software

What do we want to achieve?

Robust Software

- Robust software approach does not use redundancy.
- Robustness: extent to which software can continue to operate correctly despite the introduction of invalid inputs – as defined in program specification.
- Handles:
  - Out of range inputs
  - Inputs of the wrong type
  - Inputs in the wrong format
- What happens when invalid input is detected?
Self Checking Software

- Testing input data by, for example, error detecting code and data type checks
- Testing the control sequences by, for example, setting bounds on loop iterations
- Testing the function of the process by, for example, performing reasonableness check on the output

Assertions

- An assertion is a statement that enables you to test your assumptions about your program at specific points.
  - For example, if you write a function that calculates the speed of a particle, you might assert that the calculated speed is less than the speed of light.
  - Each assertion contains a Boolean expression that you believe will be true when the assertion executes. If it is not true, the system will throw an error.
  - By verifying that the Boolean expression is indeed true, the assertion confirms your assumptions about the behavior of your program, increasing your confidence that the program is free of error.

- Why do we need assertions if we have an exception mechanism?
  - Assertions are primarily used to handle unusual conditions arising during program execution. Assertions are used to specify conditions that a program is assumed to be true. When programming, if a programmer can say that the value being passed into a particular method is positive no matter what else is going on, passing it can be documented using an assertion to state it. Exceptions handle abnormal conditions arising in the course of program execution, but they do not guarantee successful correct execution of the program. Assertions help state scenarios that ensure the program is running smoothly. Assertions can be efficient tools to ensure correct execution of a program. They improve the confidence about the program.

- Not very new; see Bob Floyd’s original paper “Assigning meanings to programs” (1967)

Robust Software

- Errors detection of errors in the development and test process
- Cannot detect and tolerate less specific errors

Design Diversity

- Redundant, exact copies of software components alone cannot increase reliability
- Diversity: Provision of identical services through separate design and implementations – called modules, versions, variants, alternatives

- Goal: Make the variants as diverse and independent as possible, with the ultimate objective being the minimization of identical error causes
  - When the variants fail, we want them fail on disjoint subsets of the input space.
  - We want the reliability of variants as high as possible (at least one variant will be operational at all times)
Design Diversity

- Begins with an initial requirements specification
- Specifications may also employ diversity (as long as functional equivalency is maintained)
- Each developer or development organization implements the variant to the specification and provides the outputs required by the specification

Variants and Adjudicator and Cost

- When significant independence in the variants’ failure profile can be achieved, a simple and efficient adjudicator can be used, and design diversity provides effective error recovery from design faults
- It is likely, however, that completely independent development cannot be achieved in practice
- Is design diversity costly?

Case Study

- Bishop presents a useful review of the research in this area
- Summarized findings:
  - A significant proportion of the faults found in the experiments were similar
  - The major cause of the common faults was the specification (any solution?)
  - The major deficiencies in the specifications were incompleteness and ambiguity. This caused forced programmer to make some incorrect and potentially common design choices
  - Diverse design specifications can potentially reduce specification related common faults
  - In general, fewer faults seem to occur in strongly typed, tightly structured languages such as Modula-2 and Ada, whilst low-level assembler has the worst performance in terms of fault tolerance
  - A significant improvement in the reduction of identical and very similar faults was found by using the N-version design paradigm
Levels of Diversity

- Two aspects of the level of fault tolerance to consider
  - Determining at what level of detail to decompose the system into modules that will be diversified
  - Determination of which layers of the system to diversify (hardware, application software, system software, operators, and interfaced between these components)
- Multi-layer diversity?
  - Problems: cost and speed

Systematic Diversity

- One way to add diversity at a potentially lower cost is systematic diversity, although it is typically used as a software technique for tolerating hardware faults
  - Utilization of different processor registers in the variants
  - Transformation of mathematical expressions
  - Different implementation of programming structures
  - Different memory usages
  - Using complementary branching conditions in the variants by transforming the branch statements
  - Different compilers, libraries, and linkers
  - Different optimization and code-generation options

Data Diversity

- Limitations of some design diverse techniques led to the development of data diverse software fault tolerance techniques
- Data diverse techniques are meant to complement, rather than replace, design diverse techniques
- Steps
  - Obtain a related set of points in the program data space, executing the same software on those points
  - Use a decision algorithm to determine the resulting output

Failure Domain and Failure Region

- Failure Domain: set of input points that cause program failure
- Failure Region: geometry of the failure domain
- Input space of most programs is a hyperspace of many dimensions
  - E.g., if a program reads and processes a set of 25 floating-point numbers, its input space has 25 dimensions
- The valid program space is defined by the specifications and by tested values and ranges
Basic Data Re-expression

The program, P, and R determine the relationship between P(x) and P(y).

Re-expression with Postexecution Adjustment

Re-expression via Decomposition and Recombination

Sets in the Output Space

These sets are important in the development of data re-expression Algorithms.
Temporal Diversity

- Temporal diversity involves the performance or occurrence of an event at different times
- E.g., beginning software execution at different times = effective for transient faults
- Temporal diversity by using data produced at different times can also provide inputs to a data diverse technique – temporal skewing of data

Architectural Structure for Diverse Software

- To aid in avoidance of faults in the first place and the tolerance of those remaining faults, the system complexity must be controlled
- Structuring the hardware and software components that comprise these systems is a key factor to controlling the complexity
- Laprie and colleagues describe two structuring mechanisms
  - Layering: We went each layer to have the fault tolerance mechanisms to handle the errors produced in that layer
  - Error confinement areas: described in terms of the system hardware and software architecture elements

Data Re-expression

- Intersection of line segments (exact)
- Sort function (exact)
- Sensor data (approximate)
- What about “re-expression” via decomposition and recombination?
  - $\sin(a+b) = \sin(a)\cos(b) + \cos(a)\sin(b)$
  - $\cos(a) = \sin(\sqrt{2}\cdot a)$
  - Data re-expression can be used on numeric data, character strings, differential equations, and other representations. For example, combining tree transformations, data storage re-ordering, and code storage re-ordering provide considerable diversity in the data processed by large fractions of a conventional compiler
- Caution: Exact re-expression algorithms may have the defect of preserving precisely those aspects of the data that cause program failure

Examples of Data Re-expression

"*$R(x)=y$"
Xu and Randell Framework

Adjudicator

Voter?1
Voter?2
Variant?1
Variant?2

Complex
Variant?1

Two abstract classes

Targets at developing fault-tolerant application

User-defined adjudicators
User-defined variant hierarchy

Target at developing fault-tolerant applications.