Exception Handling

18-849b Dependable Embedded Systems

Charles P. Shelton

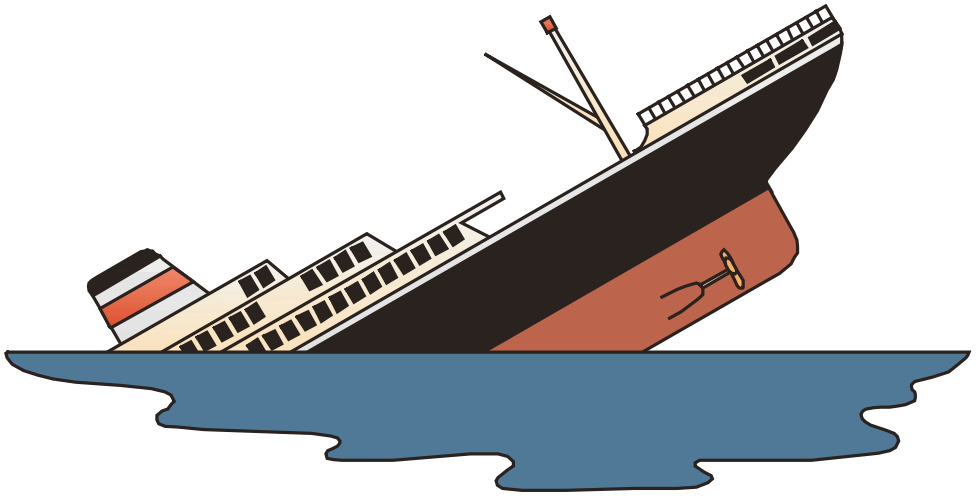
March 9, 1999

Required Reading: **Romanovsky, Alexander; Xiu, Jie; Randell, Brian;** ***Exception Handling in Object-Oriented Real-Time Distributed Systems***

Carnegie Mellon

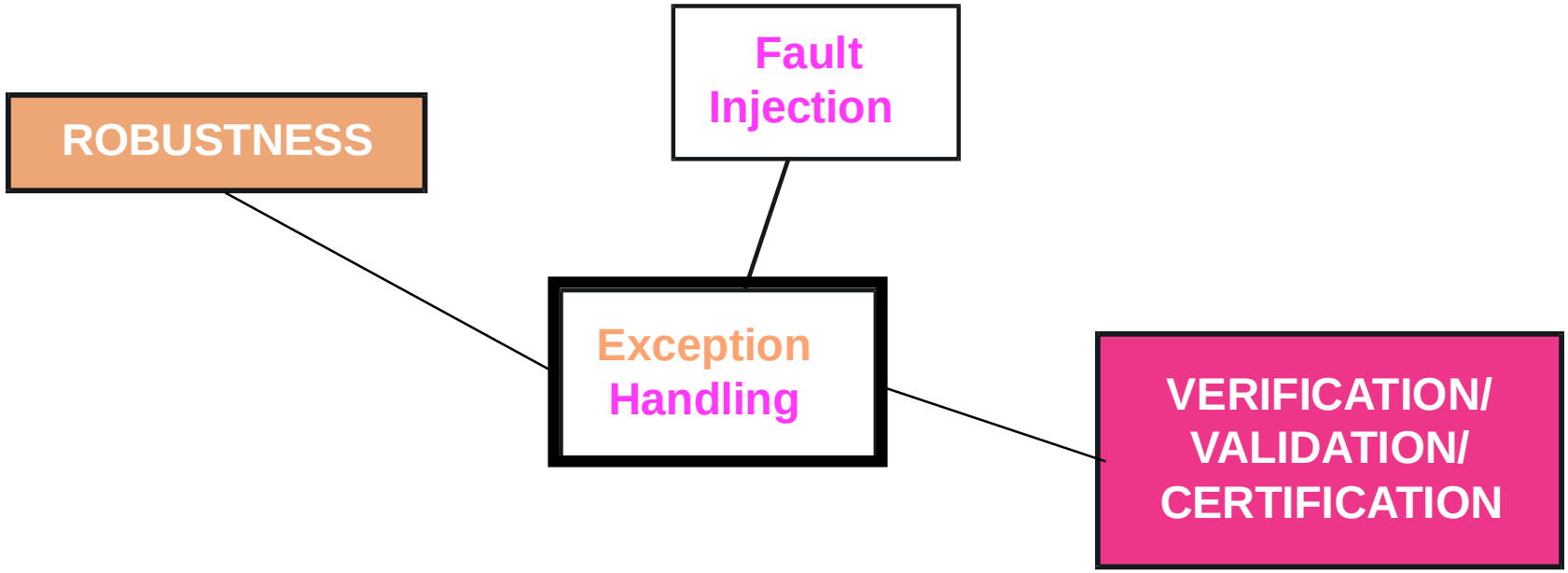
Overview: Exception Handling

- **Introduction**
- **Key concepts**
  - Known versus Unknown exceptions
  - Forward and Backward error recovery
  - Robust Exception Handling versus Real-Time System Constraints
- **Tools / techniques**
  - Dependability Cases
  - Xept
- **Metrics**
  - Ballista
- **Relationship to other topics**
- **Conclusions & future work**



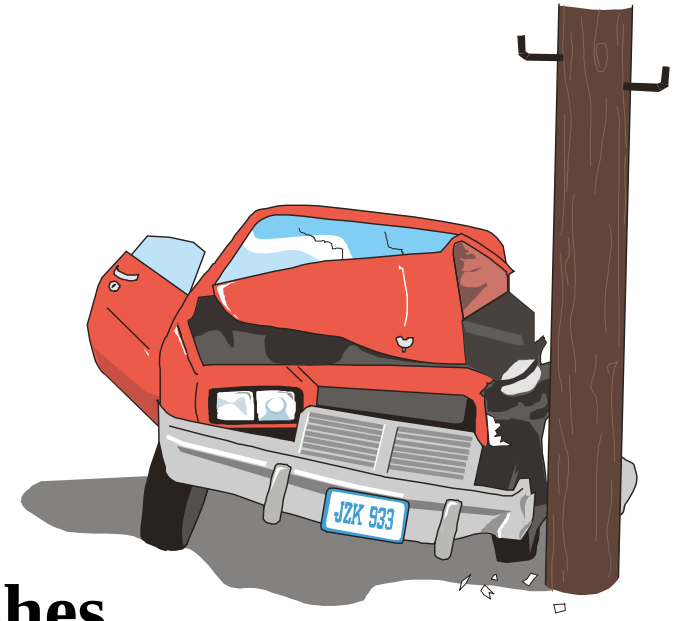
YOU ARE HERE

- **Exception Handling is a method of achieving Robustness:**



Introduction: Exception Handling

- **Exception Handling is the method of building a system to detect and recover from exceptional conditions**
  - Instances of things occurring outside the specifications of normal operation
  - Incorrect input
  - Memory/Data corruption
  - Software defects
  - Environmental anomalies, etc.



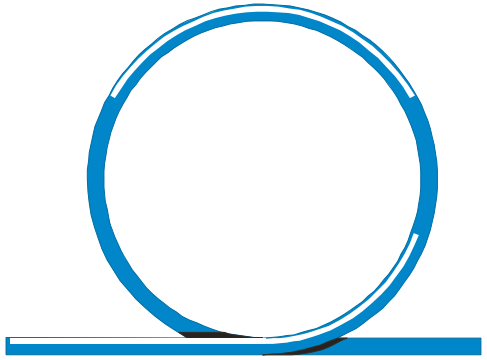
- **Exception failures are estimated to account for up to 2/3 of system crashes and 50% of security vulnerabilities**

Known versus Unknown Exceptions

- **Known exceptions**
  - Exception handlers can be written for exceptional conditions the designers know are likely to occur
  - Code reviews, walkthroughs, and testing can illuminate more conditions that can be accounted for
  - e.g. checking for null pointers, validating inputs to modules, assuring files exist before attempting to read/write to them, etc.
- **Unknown exceptions**
  - Designers cannot achieve complete coverage of all exceptional conditions
  - What about complex situations no one could anticipate?
  - Build in graceful degradation to exception handlers to minimize damage

Forward and Backward Error Recovery

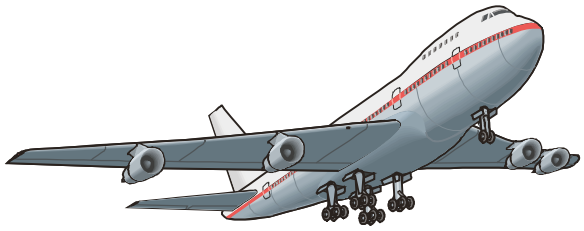
- **Forward Error Recovery: Programmed Exception Handling**
  - When an exceptional condition is reached, call exception handler to recover from error condition, but try to continue execution from error state back to normal operation
  - Implemented for known exceptional conditions at design stage
- **Backward Error Recovery: Default Exception Handling**
  - Catch-all for unanticipated exceptions and design defects
  - Exception handler halts execution and tries to return system to a previous known state



  - Good for protecting against transient and intermittent errors, where simply retrying the operation will fix the problem

Exception Handling vs. Real-Time Systems

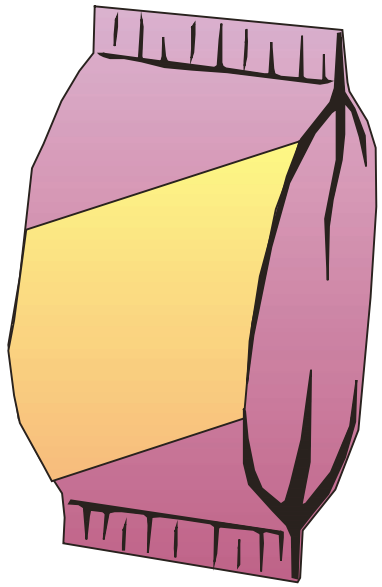
- **Robust Exception Handling may require extra processing time**
  - Transferring control from module to exception handling routine
  - Resetting system state and retrying an operation
- **Real-Time Systems may not tolerate delays due to exception handling**
  - Exception Handling routines may not be factorable into deadline constraints because of unpredictability of whether exceptions will occur



  - More bulletproof exception handling may require longer code and longer processing time to account for different execution paths

Tools / Techniques

- **No rigorous methods of exception handling design exist**
- **Major problem is covering all exceptional cases**
  - traditional software engineering techniques; code walkthroughs, code reviews, testing
  - Dependability cases develop taxonomies for improving coverage
- **Xept**
  - Method of automatically generating software wrappers correcting for exceptional inputs before passing them to the software module
  - Useful for COTS software where source code is not available for modification but you want more exception handling than module provides



Metrics

- **Measuring a system's level of exception handling is difficult**
  - How can we know a system handles all exceptional conditions if we cannot think of all possible exceptions?
  - Exhaustive testing is intractable
- **Ballista**
  - Black box method of testing software modules' responses to exceptional inputs
  - Measured relative robustness of POSIX operating systems
  - Limited to repeatable exceptions at the module level; exceptions occurring from complex interactions not covered
  - Exceptional inputs must be generated by developers

Relationship To Other Topic Areas

- **Robustness**
- **Fault Tolerant Computing**
- **Software Fault Tolerance**
- **Checkpoint/Recovery**

  - Method of handling exceptions by returning system to a known state
- **Security**
  - Robust exception handling will patch a lot of security holes
- **Human Interface/Human Error**

  - Humans are one of the biggest sources of exceptional inputs to a system
  - Exception Handling at the HCI level may prevent propagating faults

Conclusions & Future Work

- **Conclusions**
  - Coverage is a major problem. It is unrealistic to cover all exceptional conditions because they are not predictable
  - It is difficult to develop strategies to safely handle exceptions for unanticipated situations
  - Tradeoff between developing robust exception handlers and meeting real-time system deadline constraints
- **Future Work**
  - Xept and Ballista: Generating software wrappers for trapping exceptional inputs to COTS software modules
  - Using object-oriented techniques to structure designing exception handlers

Paper: Exception Handling in RT Systems

- **Trying to apply Object-Oriented techniques to exception handling in real-time distributed systems**
- **Uses coordinated atomic (CA) actions to encapsulate all operations and exception handling procedures**
  - CA actions coordinate and operate on system objects
  - CA actions manage real-time deadlines and confine scope of exception handlers
- **Developing a more structured approach to resolving exception handling and real-time constraints**
  - Addresses both timing constraints and exceptions as well as data and procedure exceptions