Non-Functional Testing
SFWR ENG 3S03: Software Testing

Alicia Marinache

Department of Computing and Software, McMaster University
Canada L8S 4L7, Hamilton, Ontario

Acknowledgments: Material based on slides Dr. R.Paige & Dr. R. Khedri and [Per00, Chapter 4]

Objectives

- Understand technology evolution impact on testing
- Explore different NFT Techniques
- Analyse the difference between the NFT Techniques

Where are we? We have...

- ... A range of widely applicable testing techniques
- ... Basic ways to measure our testing effectiveness
- ... Some idea how they could fit into a development process
- ... Some ideas about how to plan testing and develop a testing strategy

Next

- How can we adapt these to cope with all the different technologies and architecture styles we may encounter?
- A forty-plus (British) narrative

London, 1981

- Margaret Thatcher's government says that the future of the British economy is in financial services
- You want to be part of that

TradeMaster

- Take a day's trading data as a structured data file

TradeMaster

- Take a day's trading data as a structured data file
- Adds it to historical database

TradeMaster

- Take a day's trading data as a structured data file
- Adds it to historical database
- Tells you what trades to do tomorrow

TradeMaster

- Take a day's trading data as a structured data file
- Adds it to historical database
- Tells you what trades to do tomorrow
- Plain text output

TradeMaster

- Take a day's trading data as a structured data file
- Adds it to historical database
- Tells you what trades to do tomorrow
- Plain text output
- Written in C in a procedural style

TradeMaster

- Take a day's trading data as a structured data file
- Adds it to historical database
- Tells you what trades to do tomorrow
- Plain text output
- Written in C in a procedural style

How could we test this?

- Brainstorm for 1 minute
- One testing technique or testing process decision
- One design principle for testability

Technology Driver: Batch Processing

- Put in historical data, run it once, then check output is good "prediction" of what actually happened
- Validity checks
  - This is an uncertain world of predictions, so requirements may be hard to define
- Lots of coverage considerations
- Testing vs theory: does behaviour match the share-price model in the textbook?
- Testing vs reality: does behaviour match observed historical share price changes?

1986: Library Integration

- UK government significantly deregulates the London Stock Exchange
- Great times for your business, lots of demand for your product
- Customers want access to lots of external data feeds in arcane formats
- Luckily, you can buy software libraries that parse the feeds into common representations

Library Integration: Testing Challenges closed source

- You get printed instructions, C header files, and (obfuscated) object code
- Don't know any implementation details
- TradeMaster will only work if the libraries behave as you expect

How could we test this?

- Brainstorm for 1 minute
- One testing technique or testing process decision
- One design principle for testability

Library Integration: Testing Solutions

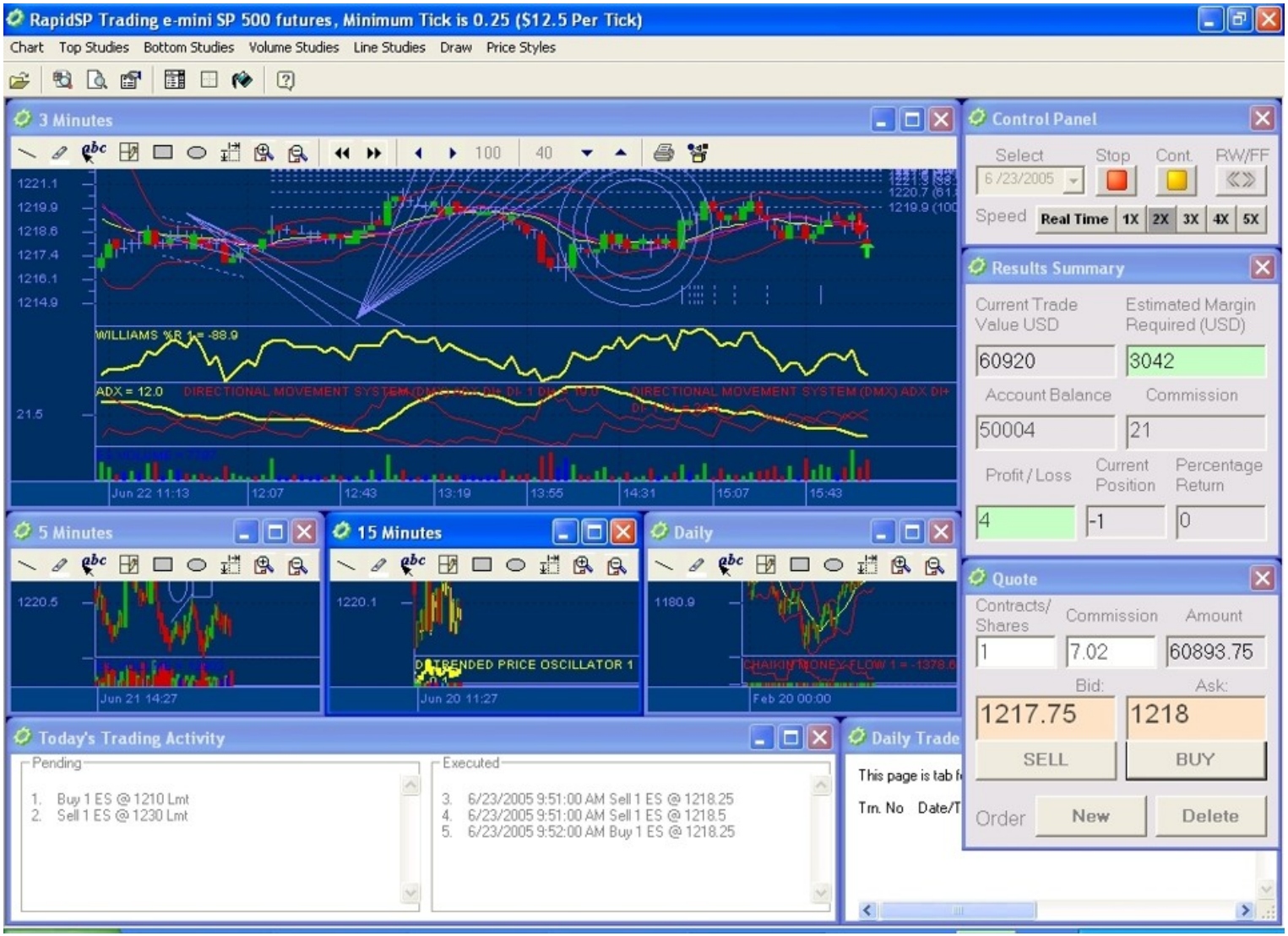
- Wrap library in a single module (e.g., "SkyfeedSharePriceParser") and test that
- Use a range of black-box techniques e.g., input partitioning
- Ammann & Offutt chapter 4 or 6 (1st or 2nd ed)

Library Integration Testing Challenge: Hidden State

- The libraries probably allocate internal data
- Different functions will interact to modify that data and respond to it: How? When?
- OO testing: this problem becomes massive

1991: GUI

- "Day trading" is getting more popular
  - And pace is increasing
- Customers want to trade from their desktop computers
  - Even PCs running Windows 3.0
- You decide to provide a GUI application with live data displays and trading actions



GUI: Testing Challenges

- Many possible actions (some of them equivalent)
- Whole extra layer of states and hence sequences
- Difficult to automate

How could we test this?

- Brainstorm for 1 minute
- One testing technique or testing process decision
- One design principle for testability

GUI: Testing Solutions

- Use models to represent user interactions (easy because GUIs are event-driven)
- Use MVC separation, and test the Model, the Controller, and the View separately, using stubs
- Look for inputs that might not be guarded (e.g., by reading the code)
- Use automation tools (e.g., "capture & replay")

GUI Testing: Other Challenge

- Fragility of GUI Test Automation (e.g., What happens if we move or rename a button?)

1995: Object Oriented (OO) Paradigm

- Complexity of our software is increasing
- Competitors have been talking about how their new products use "modern object-oriented design principles"
- You decide to reimplement your core software using C++ in an OO style

OO: Testing Challenges

- Inheritance: changes to parents have impacts on children
- State: methods on the same object (same class!) may interact; sequences matter a lot
- Interlinked nature of objects: their state matters too

How could we test this?

- Brainstorm for 1 minute
- One testing technique or testing process decision
- One design principle for testability

OO: Testing Solutions

- Use the class as the unit and have JUnit-style regression tests for it, use stubs and mocks in place of other classes
- Consider small clusters of classes as units
- Use assertions (Ch 17 [Bin00])
  - e.g., have methods check that the class is in a valid state at the beginning and end of their method body (debug builds only)

Assertions Use Example

```
1   public void withdraw(double amount) {
    // Preconditions: amount must be positive
        and less than current balance
3   assert amount > 0 : "Withdraw amount must
        be positive";
    assert amount <= this.balance : "Withdraw
        amount must be less or equal than
        balance;
5
    // Postcondition: The balance should
        decerase by the deposited amount
7   this.balance -= amount;

9   // Postcondition: balance must remain
        positive or zero
    assert this.balance >= 0 : "Balance cannot
        be negative after withdrawal";
11  }
```

OO Testing: Other Challenges

- Polymorphism (e.g., call subtype method through supertype pointer): code being called isn't known at compile-time (see next slide)
  - Maybe can't be known from the code fragment you're testing
- Errors can come from superclasses or subclasses that you don't control
- Frame problem - How do we specify or verify what methods preserve vs. mutate, especially when side effects are hidden behind abstractions?

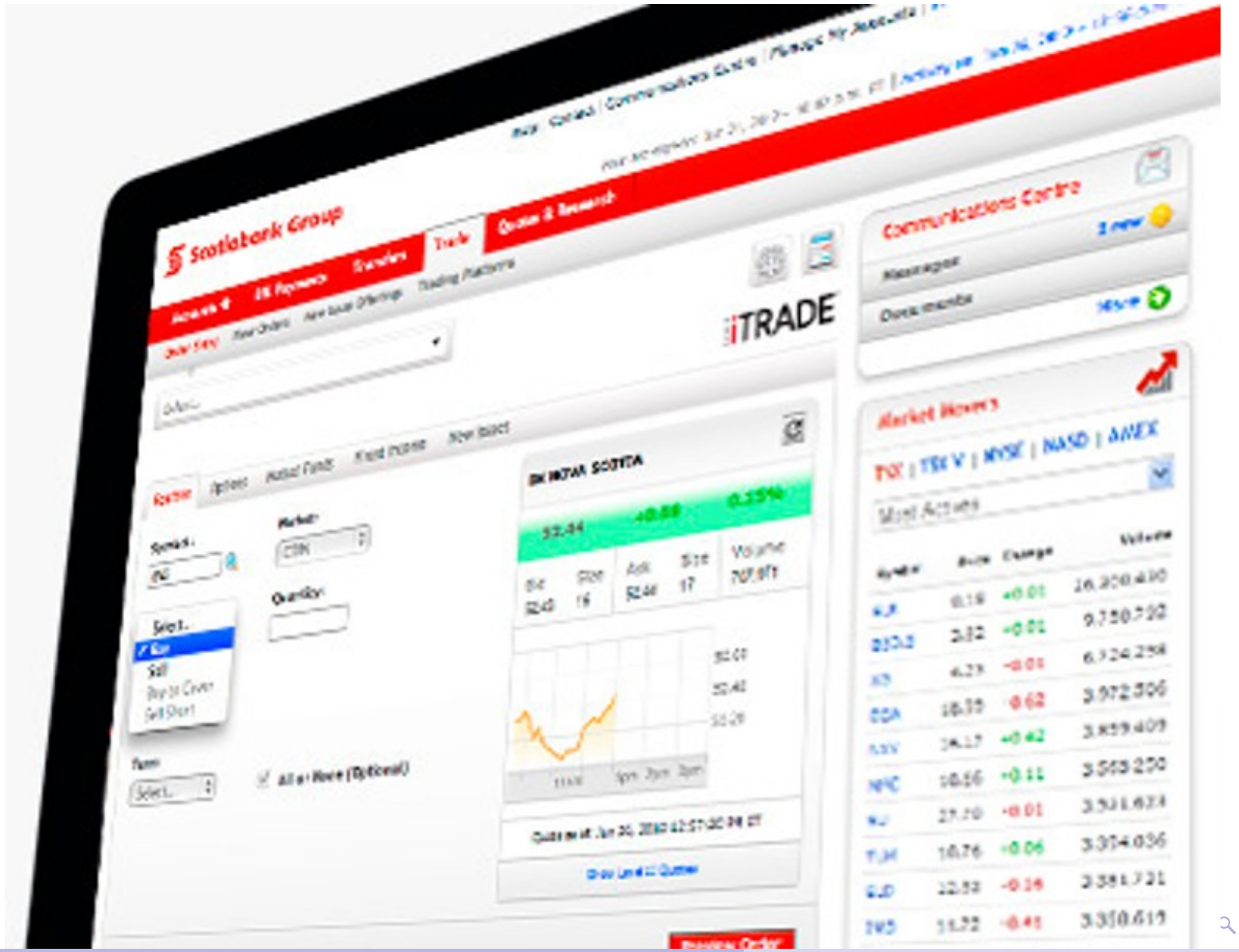
```
1  class Animal {
     virtual void speak() { std::cout << "Animal
         sound"; }
3  };

5  class Dog : public Animal {
     void speak() override { std::cout <<
         "Bark"; }
7  };

9  Animal* a = new Dog();
   a->speak();  // Output: Bark
```

2003: The Web

- The web is ubiquitous
- You want to offer your main product as a web app



The Web: Testing Challenges

- All the "shrinkwrap" problems of wide userbase and high expectations of quality
- Need to test client, server and their interactions
- Diversity of browsers and, increasingly, devices – so you may have a mobile web version as well)

How could we test this?

- Brainstorm for 1 minute
- One testing technique or testing process decision
- One design principle for testability

The Web: Testing Solutions

- Simple websites can be subjected to coverage of the graph of links between pages
  - This often breaks down once pages are dynamically created
- Tools (such as: Selenium IDE) can record interactions and check results (e.g., check that some string appears in the page that you end up with)
- Treat web browser as just one more configuration variable (like OS, user category, etc)

Web Testing: Other Challenges

- Security breaches!

2008: Reactive Systems

- Humans are too slow
- The future is automated: "High Frequency Trading"
- So you build a version that doesn't just recommend, it buys and sells autonomously
  - Often trades several times a second
  - This is now a reactive system

Reactive Systems: Testing Challenges

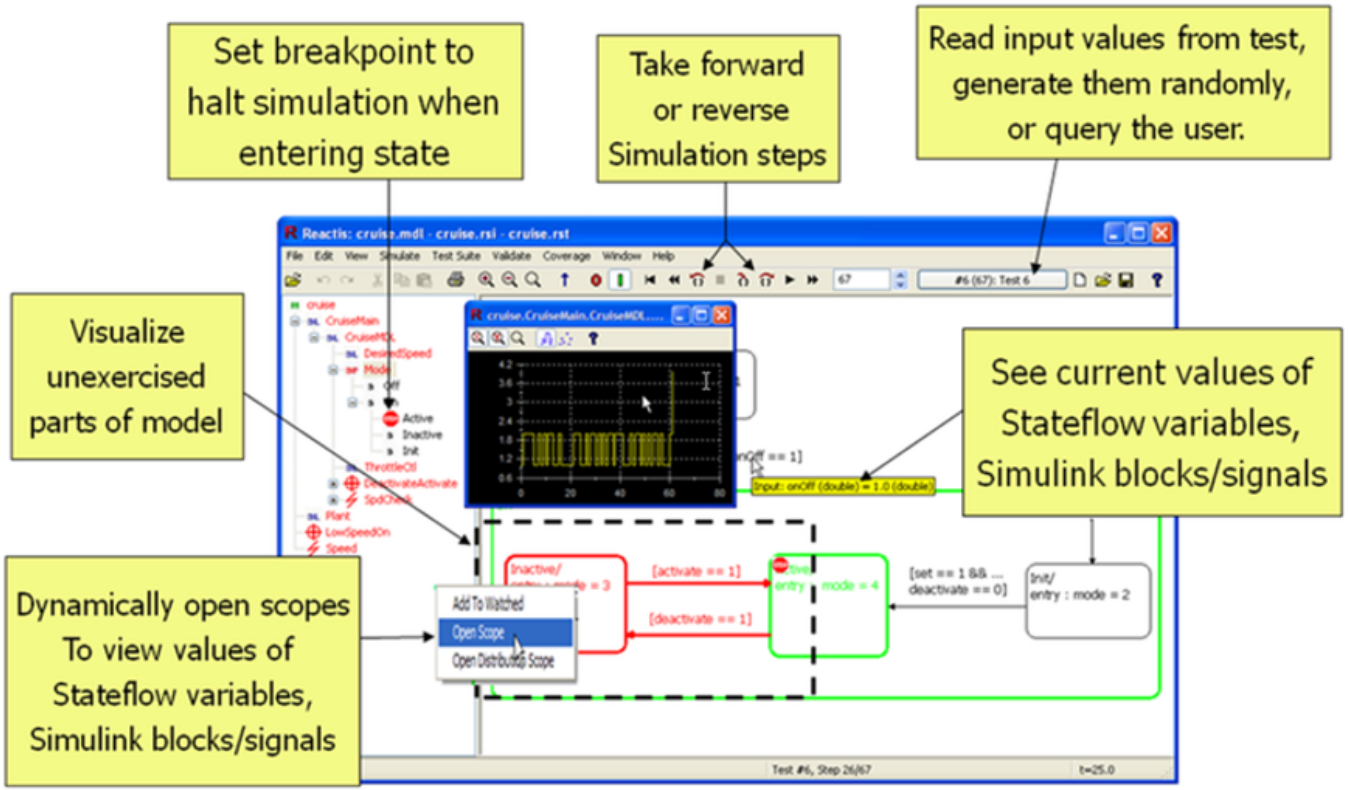
- Reactive: behaviour of system depends a lot on behaviour of its environment, which in turn is influenced by what the system does
  - Feedback loops
- Real-time performance is important
  - Often Worst Case Execution Time (WCET)

How could we test this?

- Brainstorm for 1 minute
- One testing technique or testing process decision
- One design principle for testability

Reactive Systems: Testing Solutions

- Try simulating it



Source: `http://www.reactive-systems.com/simulink-debug.html`

Reactive Systems: Other Challenges

- Making valid simulations is hard, often impossible
- Particularly in a competitive situation (e.g., financial trading) where other actors may react cleverly to what you're doing

2010: Concurrency

- Your autonomous trading engine is getting more sophisticated
  - e.g., Neural networks, Bayesian learning, ...
- CPU cores are not getting faster any more
- Your programmers are increasingly having to exploit concurrency

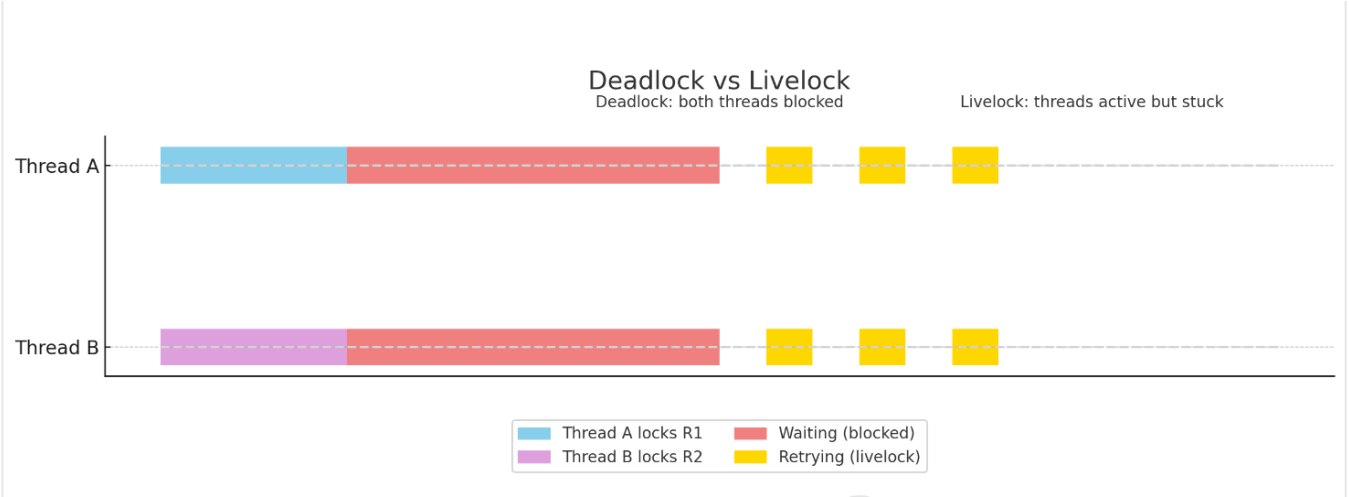
Concurrency: Testing Challenges

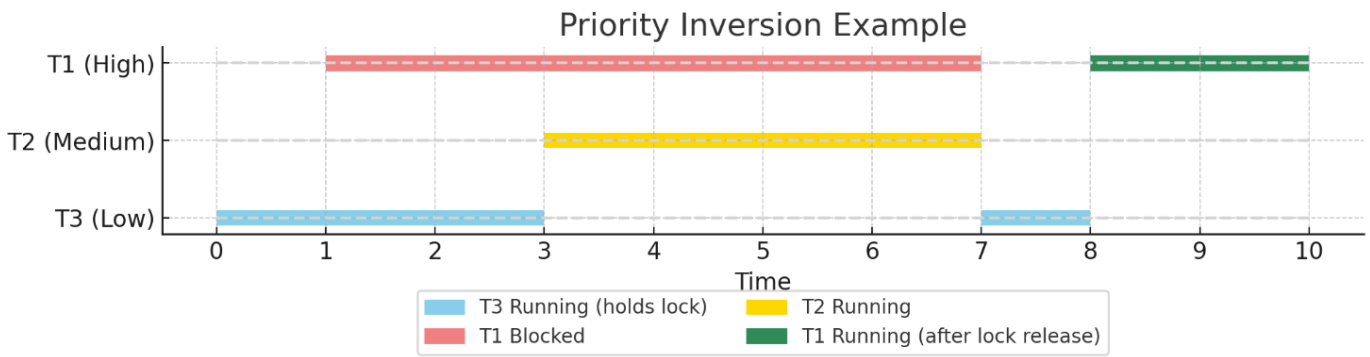
- New classes of failure mechanisms
  - e.g., Livelocks, deadlocks, starvation, priority inversion, data corruption due to unexpected interleaving
- New system states
  - e.g., Backup thread is down, but rest of system is up

How could we test this?

- Brainstorm for 1 minute
- One testing technique or testing process decision
- One design principle for testability

Concurrency: Failure Mechanisms Examples





Concurrency: Testing Solutions

- Test parts in isolation, as single-threaded as possible
- Build autotesters that stress-test the system
- Use models (at least semi-formal) to understand how locks are taken and released
  - TLA+ might be useful for this

Concurrency: Testing Tools

- Detect data race
  - e.g., `https://clang.llvm.org/docs/ThreadSanitizer.html`
  - Tracks reads, writes, memory barriers, lock acquisitions, thread creation etc.
  - Detects lock inversions, overlapping reads/writes without synchronisation etc.
  - Memory and speed overheads in region of 10x

Concurrency: Testing Tools (cont'd)

- Track effects of locks
  - e.g., `https://github.com/couchbase/phosphor`
  - How long held for?
  - Which threads were affected?

Concurrency: Other Challenges

- Concurrency failures can be very hard to reproduce

2024: Constraining Emergence

- In the wake of the catastrophic stock market behaviour of 2024, coming so soon on the heels of that of 2008, radical new legislation is imposed Europe-wide
- Financial trading software providers directly responsible for the emergent effects of their applications
- They have to be able to show that their software has respected, in all its decisions, a set of strict "globally responsible trading constraints"

Constraining Emergence: Testing Challenges

- How do you predict emergent effects when testing?
- "Globally responsible trading constraints" may be not so hard to obey . . . if you don't mind crippling your trading performance
- Competition are busy working out exactly where the gaps are
- Crude safety-first approach will see your product left behind

How **could** we test this?

Constraining Emergence: Testing Solutions

- I have no idea!
- Good thing they have you as a test engineer, right?

Era	Tech Driver	Testing Challenge
1981 – 1986	Batch + Library integration	Static correctness, input/output validation
1991	GUIs	State explosion, automation difficulties
1995	Object Orientation	Stateful interaction, polymorphism
2003	Web applications	Multi-tier interaction, environment variability
2008	Reactive systems	Real-time, environment coupling
2010	Concurrency	Non-determinism, hard-to-reproduce bugs
2024+	Constraint Emergence	Global constraints, accountability, socio-technical impact

More Reading

- On Libraries: Security Audit [Sec14]
- On GUI: Section 7.3 (ed.1 only) [AO16], [Mem02], [MSP01]
- On OO: [Fow07], [Bin00], [PY08], Section 7.1 (ed.1 only) [AO16]
- On Web: Section 7.2 (ed.1 only)) [AO16]
- On Reactive Systems: Section 7.4 (ed.1 only)) [AO16], [RAO92]
- On Concurrency: Chapter 12 [Goe06]

- The objective is to ensure that the product designed is structurally sound and will function within parameters
- Determines that the technology has been used properly and that when all the component parts are assembled they function as a cohesive unit
- Focus is on process, not so much on correctness
- Validation & Verification

- The structural system testing techniques are:
  - Stress testing, Execution testing
  - Recovery testing
  - Compliance testing
  - Security testing
  - etc.

- Stress testing determines if the system can handle heavy loads
- The areas that are stressed include
  - input transactions, accessing users,
  - disk space, communications,
  - memory capacity, etc.
- Emphasis on robustness, availability, and error handling above the break threshold

How to Use Stress Testing

- Simulate as closely as possible the production environment
- Enter heavier than expected volumes of data
- Simulated larger than expected number of users
- Error conditions should be included in tested transactions

What are we looking for?

- Load balancing problems
- Bandwidth issues
- System capacity issues
- Poor response time
- etc.

- Execution testing determines if the system achieves the desired level of proficiency in a production status
- It can verify response times, turnaround times, as well as design performance
- The execution of a system can be tested in whole or in part, using the actual system or a simulated model of a system

Specific objectives of execution testing include:

- Determining the performance of the system structure
- Verifying the optimum use of hardware and software
- Determining response time to on-line use requests
- Determining transaction processing turnaround time

- Recovery testing determines the ability to restart operations after the integrity of the application has been lost
- Revert to a point where the integrity of the system is known, reprocess transactions up to the point of failure
- The time required to recover operations is affected by
  - the number of restart points,
  - the volume of applications run on the computer center,
  - the training and skill of the people conducting the recovery operation,
  - and the tools available for recovery

Specific objectives of recovery testing include the following:

- Adequate backup data is preserved
- Backup data is stored in a secure location
- Recovery procedures are documented
- Recovery personnel have been assigned and trained
- Recovery tools have been developed and are available

- How to Use Recovery Testing?
  - Asses the procedures, methods, tools, and techniques to evaluate their adequacy
  - Introduce a failure in the system and evaluate the ability to recover the system
  - Perform recovery testing by people who would execute it in the real situation
- When to use? when the continuity of operation of the application is essential

- Compliance testing verifies that the application was developed in accordance with information technology standards, procedures, and guidelines

- The methodologies are used to
  - increase the probability of success
  - enable the transfer of people in and out of the project with minimal cost,
  - increase the maintainability of the application system

Specific objectives of compliance testing include:

- Determining that systems development and maintenance methodologies are followed
- Ensuring compliance to standards, procedures, and guidelines
- Evaluating the system documentation to verify it is complete and reasonable

- Security testing is designed to ensure secure confidential information and for competitive purposes to assure third parties that their data will be protected
- The amount of security provided will be dependent upon the risks associated with compromise or loss of information
- Security defects do not become as obvious as other types of defects

Specific objectives of security testing include:

- Determining that adequate attention has been devoted to identifying security risks
- Determining that a realistic definition and enforcement of access to the system has been implemented
- Determining that sufficient expertise exists to perform adequate security testing
- Conducting reasonable tests to ensure that the implemented security measures function properly

- Security testing is a highly specialized part of the test process
- Physical security deals with the penetration by people in order to physically gather information
- Logical security deals with the use of computer processing and/or communication capabilities to improperly access information

- The type of tests to be conducted will vary upon the condition being tested
- The testing should be performed both prior to the system going into an operational status and after the system is placed into an operational status
- Extent of testing should depend on the security risks
- The individual assigned to conduct the test should be selected based on the estimated sophistication that might be used to penetrate security

Technique	Objective (determine that...)
Stress	... system behaves outside expected volumes and load
Execution	... system achieves desired level of proficiency (performance)
Recovery	... system can be returned to operational status after a disaster / failure
Compliance	... system is developed in accordance with standards and procedures
Security	... system is protected

When to use?

Technique	Rqmt.	Design	Dev	Pre-Prod.	Maint.
Stress					

When to use?

Technique	Rqmt.	Design	Dev	Pre-Prod.	Maint.
Stress	X	X	X	X	X
Execution					

When to use?

Technique	Rqmt.	Design	Dev	Pre-Prod.	Maint.
Stress	X	X	X	X	X
Execution	X	X	X	X	X
Recovery					

When to use?

Technique	Rqmt.	Design	Dev	Pre-Prod.	Maint.
Stress	X	X	X	X	X
Execution	X	X	X	X	X
Recovery	X	X	X	X	X
Compliance					

When to use?

Technique	Rqmt.	Design	Dev	Pre-Prod.	Maint.
Stress	X	X	X	X	X
Execution	X	X	X	X	X
Recovery	X	X	X	X	X
Compliance	X	X	X	X	X
Security					

When to use?

Technique	Rqmt.	Design	Dev	Pre-Prod.	Maint.
Stress	X	X	X	X	X
Execution	X	X	X	X	X
Recovery	X	X	X	X	X
Compliance	X	X	X	X	X
Security	X	X	X	X	X

References I


Paul Ammann and Jeff Offutt, *Introduction to software testing*, Cambridge University Press, 2016.


Robert Binder, *Testing object-oriented systems: models, patterns, and tools*, Addison-Wesley Professional, 2000.


Martin Fowler, *Mocks aren't stubs*, 2007.


Brian Goetz, *Java concurrency in practice*, Pearson Education, 2006.


Atif M Memon, *GUI testing: Pitfalls and process*, IEEE Computer Society (2002), no. 08, 87–88.

References II


Atif M Memon, Mary Lou Soffa, and Martha E Pollack, *Coverage criteria for GUI testing*, Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, 2001, pp. 256–267.


William E. Perry, *Effective Methods for Software Testing*, 2nd ed., John Wiley & Sons, Inc., 2000.


Mauro Pezzè and Michal Young, *Software testing and analysis: process, principles, and techniques*, John Wiley & Sons, 2008.

References III

 Debra J Richardson, Stephanie Leif Aha, and T Owen O'Malley, *Specification-based test oracles for reactive systems*, Proceedings of the 14th international conference on Software engineering, 1992, pp. 105–118.

 Contrast Security, *The unfortunate reality of insecure libraries*, 2014.