For Success, Build Record/Playback into Your Application

Gerard Meszaros

StarEast2008@gerardm.com

Presenter Background

Gerard Meszaros is a Calgary-based consultant specializing is agile development processes. Gerard built his first unit testing framework in 1996 and has been doing automated unit testing ever since. He is an expert in test automation patterns, refactoring of software and tests, and design for testability. Gerard has applied automated unit and acceptance testing on projects ranging from full-on eXtreme Programming to traditional waterfall development and technologies ranging from Java, Smalltalk and Ruby to PLSQL stored procedures and SAP's ABAP. He is the author of the book xUnit Test Patterns – Refactoring Test Code.

Gerard Meszaros

Outline

1. Test Automation Strategies
2. Test Automation Challenges
3. Built-in Test Recording Case Studies
4. Techniques for Mitigating Challenges
5. Implementing Built-In Test Automation

Purpose of Tests

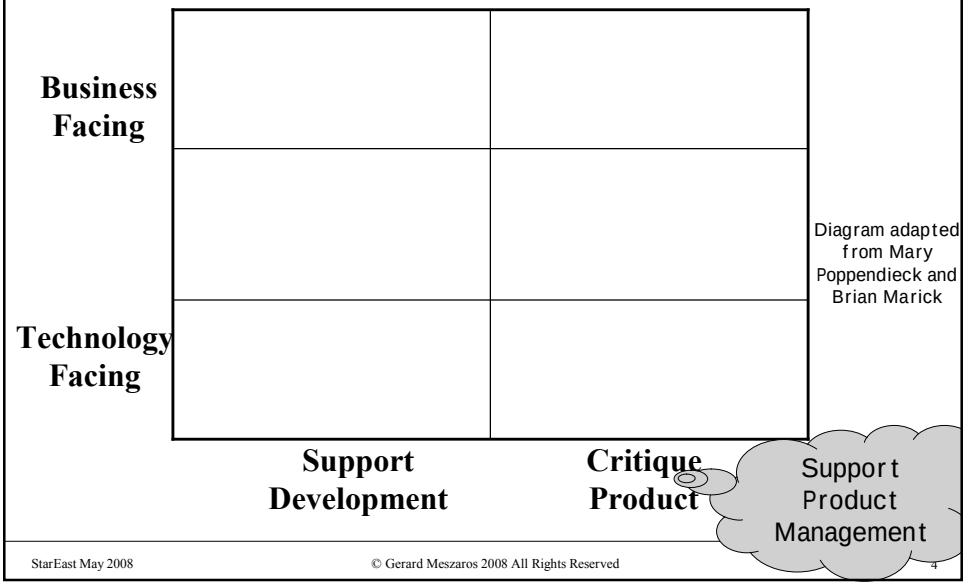


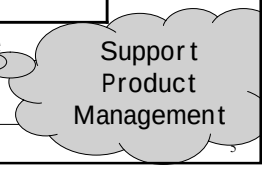
Diagram adapted from Mary Poppendieck and Brian Marick

Build Record/Playback into Your Application

Purpose of Tests

Business Facing	Acceptance Tests	Usability Testing
	Component Tests	Exploratory Testing
Technology Facing	Unit Tests	Property Testing
	Support Development	**Critique Product**

Diagram adapted from Mary Poppendieck and Brian Marick



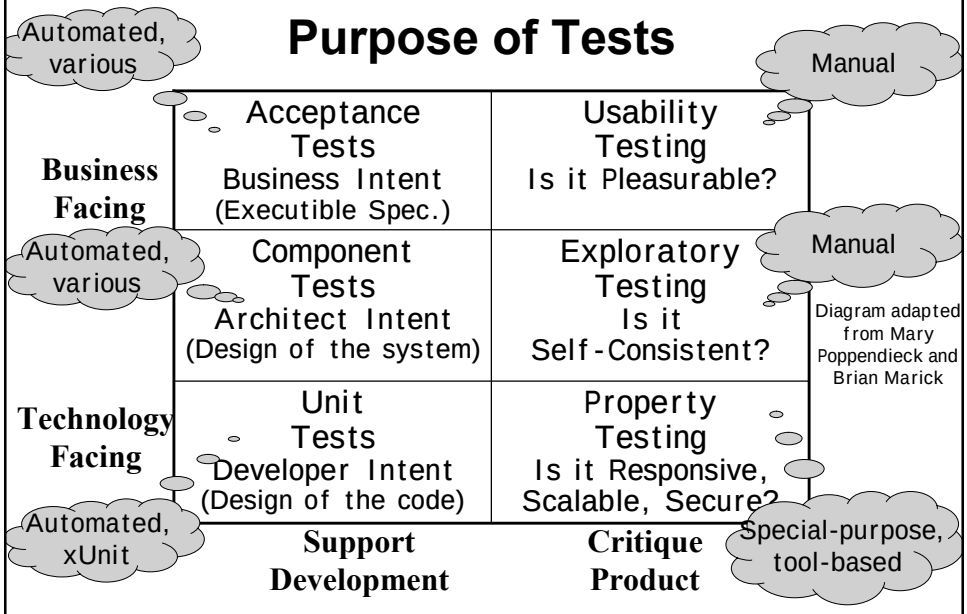
Build Record/Playback into Your Application

Purpose of Tests

Business Facing	Acceptance Tests Business Intent (Executable Spec.)	Usability Testing Is it Pleasurable?
	Component Tests Architect Intent (Design of the system)	Exploratory Testing Is it Self-Consistent?
Technology Facing	Unit Tests Developer Intent (Design of the code)	Property Testing Is it Responsive, Scalable, Secure?
	Support Development	**Critique Product**

Diagram adapted from Mary Poppendieck and Brian Marick

Purpose of Tests



Goals of Automated Tests

- **Fully Automated & Self-Checking**
  - Test can be executed without any human effort
  - Test verifies it own results without human inspection
- **Repeatable**
  - Can run same test many times in a row with same results
- **Robust**
  - Same results today, tomorrow, next month, next year
- **Maintainable**
  - Understandable
  - Easily changed

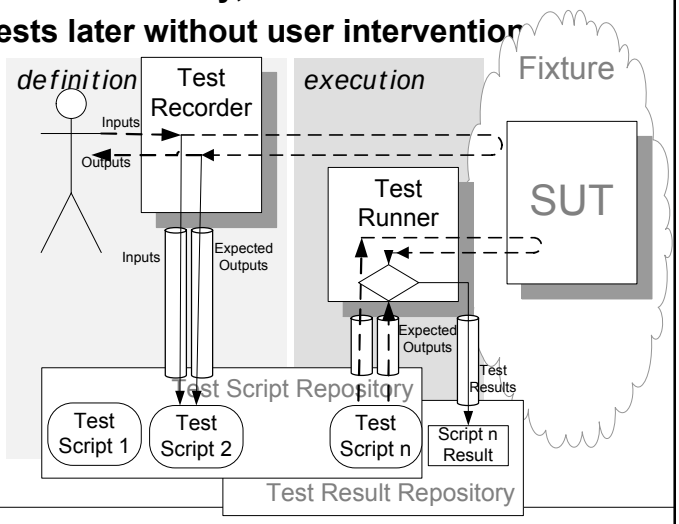
Test Preparation Strategies

- **Recorded Tests**
- **Scripted Tests**
- **Data-Driven Tests**

Recorded Tests

- **User executes tests manually; tool records as tests**
- **Tool replays tests later without user intervention**

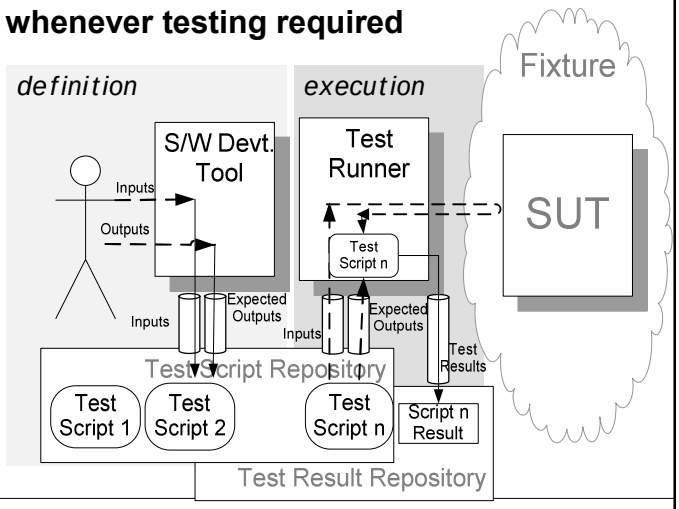
The tests are data interpreted by the test tool.



Scripted Tests

- **Tester writes test code to exercise the software**
- **Test code run whenever testing required**

The tests are a program that tests the program

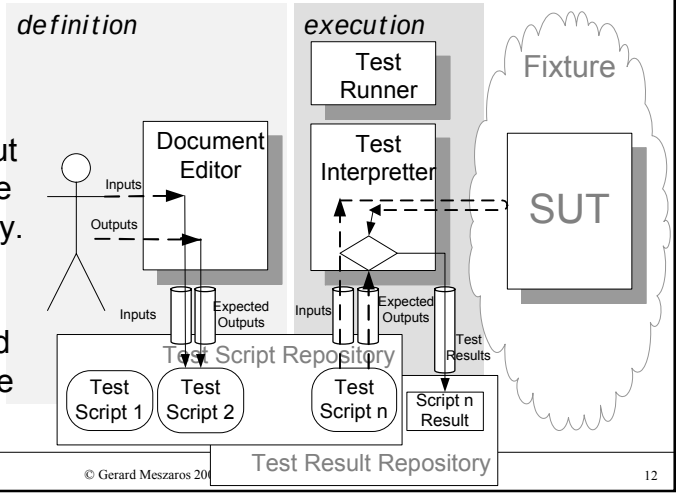


Data-Driven Tests

- **The tests are expressed as tabular data by users.**
- **The tests are read & executed by a test interpreter written by techies.**

Prepared like Scripted Tests but with a much more limited vocabulary.

"Keyword"-based testing is just one example.



Outline

1. **Test Automation Strategies**
2. **Test Automation Challenges**
3. **Built-in Test Recording Case Studies**
4. **Techniques for Mitigating Challenges**
5. **Implementing Built-In Test Automation**

Common Pitfalls of Recorded Tests

- **The tests cannot be rerun reliably due to**
  - Synchronization problems with the UI
  - Differences in system/database state
  - Differences in how system behaves at different times
- **The tests cannot be easily understood**
  - Tests are very long because they are focussed on UI details
  - Business rules are lost in all the UI details
- **The tests take a long time to run**
  - Asynchronous interaction via the UI requires frequent "waits" resulting in Slow Tests
  - Extensive interaction with databases doesn't help

Common Pitfalls of Recorded Tests (2)

- **The tests only verify a subset of what the tester verified**
  - Tester has a brain and eyes that look for things
    - » **Commercial R&PB tools don't know what to look for**
  - Adding checkpoints increases cost/complexity
    - » **of recording tests**
    - » **executing tests**
- **The tests are very sensitive to changes in the application**
  - They need to be rerecorded very often

Root Cause Analysis

- **Caused by trying to test the application after it has been built**
  - Test automation done on best effort basis,
  - using whatever interfaces are available (UI!)
- **Test automation doesn't influence design of system**
  - Little collaboration between Test and Development
  - Too little contact, too late in development cycle
- **This results in Slow, Erratic, Fragile Tests**
  - that consume a lot of resources to maintain

Common Symptoms or "Smells"

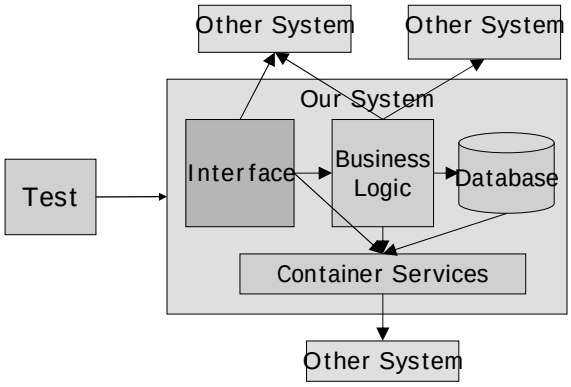
- **Fragile Tests**
  - Interface Sensitivity
  - Behavior Sensitivity
  - Data/Context Sensitivity
- **Erratic/Unrepeatable Tests**
  - Shared Fixtures
- **Lack of Test Maintainability**
  - Obscure/Verbose Test
  - Hard-Coded Values
- **Slow Test Execution**
  - Asynchronous Interfaces
  - Database dependencies

All Described in book: xUnit Test Patterns

The Fragile Test Problem

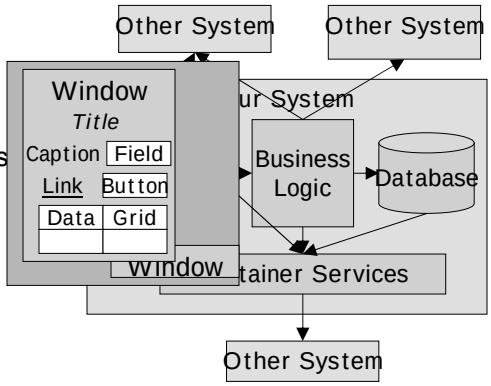
What, when changed, may break our tests accidentally:

- Behavior Sensitivity
  - » **Business logic**
- Interface Sensitivity
  - » **User or system**
- Data Sensitivity
  - » **Database contents**
- Context Sensitivity
  - » **Other system state**



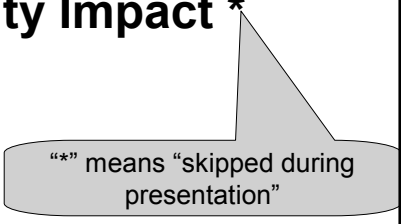
Interface Sensitivity

- **Tests must interact with the SUT through some interface**
- **Any changes to interface may cause tests to fail.**
  - New/renamed/deleted windows or messages
  - New/renamed/deleted fields
  - New/renamed/deleted data values in lists
  - Renamed functions in API
  - Changed function parameters



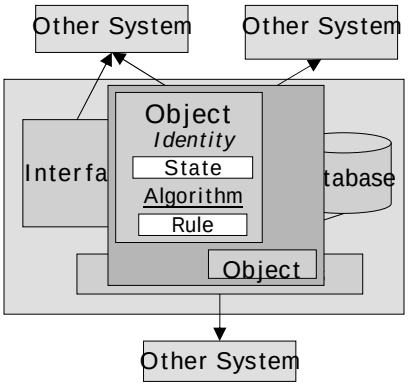
Interface Sensitivity Impact *

- **Symptoms:**
  - Unrecognized interaction
    - » **e.g. window or message**
  - New unrecognized field
  - Wrong data
- **Impact:**
  - Many of these will be fatal.
  - Non-fatals ones need manual inspection
  - Every test that uses modified interface may fail
    - » **Large maintenance impact**



Behavior Sensitivity

- **Tests must verify the behavior of the system.**
  - Behavior also involved in fixture setup
- **Any changes to business logic may cause tests to fail.**
  - New/renamed/deleted states
  - New/changed/removed business rules
  - Changes to business algorithms
  - Additional data requirements

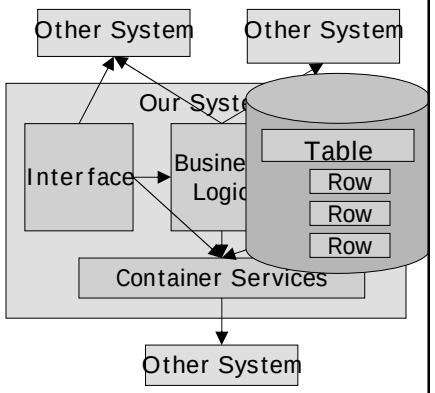


Behavior Sensitivity Impact *

- **Symptoms:**
  - Tests block due to invalid inputs
  - More/fewer items found
- **Impact:**
  - Tests that depend on this behavior to setup the test fixture may also fail unexpectedly (fragile test)

Data Sensitivity

- **Tests depend on a test fixture**
  - Consists largely of data
- **Changing the contents of the database may cause tests to fail.**
  - Added/changed/deleted records

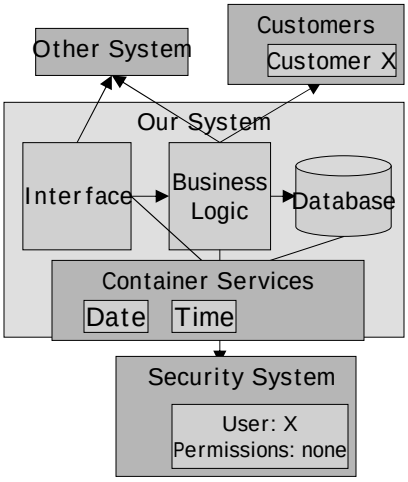


Data Sensitivity Impact *

- **Symptoms:**
  - Too many records found
  - Too few records found
  - Unique key violation
- **Impact:**
  - Any test that depends on database contents may fail
  - Tests that add to database may fail

Context Sensitivity

- **Tests may depend on another systems' state**
  - Stored outside the application being tested
- **Changing the state of the context may cause tests to fail.**
  - State of the container
    - » **e.g. time/date**
  - State of related systems
    - » **Availability, data contents**



Context Sensitivity Impact *

- **Symptoms:**
  - Same system "build" behaves differently during different test runs
- **Impact:**
  - Unrepeatable tests
  - Nondeterministic tests

Recorded Tests Can Work Well …

Satisfied Customer Says:

"… Not only did this save 10's of man-years of testing effort, but it even uncovered before unknown bugs in the legacy system which we considered to be the gold standard. …"

… in the right circumstances:

- We retrofitted the capability to record and playback right into the application.
- We came up with strategies to mitigate each of the challenges

Why Build R&PB into your Application?

Can make it much easier for your business people to use:

- **Domain-specific terminology**
  - **Business concepts, not technical**
  - **E.g. Select Direction South**
    vs. Select value "south" from dialog box with title="Direction"
- **Testing rules built in**
  - **No need to fiddle with generic sensitivity settings in each recorded test**
  - **E.g., Always ignoring time fields**
- **Simplified test setup**
  - **Knows what starting state consists of and how to cause it to be**
  - **E.g., Managing Database Snapshots**

Why Build R&PB into your Application?

- **Gives you complete control over what to pay attention to and what to ignore!**
  - E.g. Ignore the time/date fields
- **Tests will run much faster bypassing the User Interface**
- **May be less expensive than commercial tools, if:**
  - Large numbers of users, or
  - Low cost to build
    - » **I.e., simple design or easy-to-use framework**

Why Not:

- **Can be a significant investment of time and money**
  - **Expect it to cost twice what you first estimate**
- **Can require a fair bit of expertise**

Why Use a Commercial Tool?

- **Available Right Away**
  - Free trial downloads
  - A lot of test automation expertise is built in
- **May Be Lower Risk**
  - If you can address the 4 sensitivities

Why Not?

- **Expensive to buy**
  - **Per-user licensing cost can make it very expensive to deploy**
- **May force you to work at too low a level**
  - **E.g., screen widgets**
  - **May cause you to loose focus on the big picture**
- **Giving up control of your destiny**
  - **Bound to a tools vendor and their vision of the future**

Outline

1. Test Automation Strategies
2. Test Automation Challenges
3. Built-in Test Recording Case Studies
4. Techniques for Mitigating Challenges
5. Implementing Built-In Test Automation

Case Studies

For each case study:

- **Test Architecture**
  - Test Hooks vs. Injected Decorator
- **Sample Tests**
  - Abstraction Level of Test Recording
- **Result Verification Strategy**
- **Dependency Management Strategy**

Case Study: MS Excel

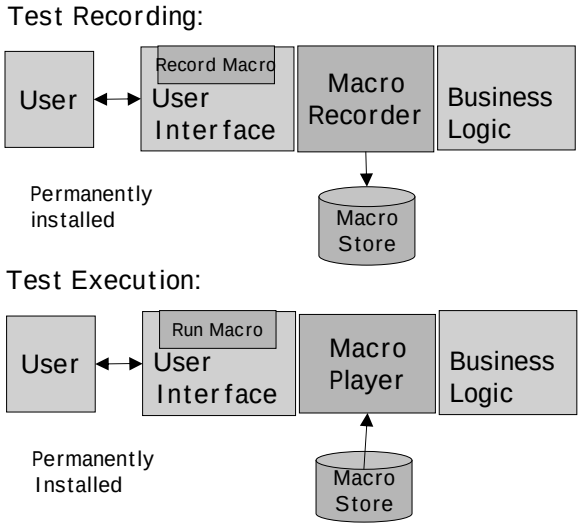
- **Microsoft Office applications include a "Macro recording" capability.**
- **Rumour has it that this was introduced initially to automate testing of Excel**
- **Let's look at how this can be used to record tests.**

Sample Recording – MS Excel

```
Sub OriginalSortByDayAndTimes()
'
'   SortByDayAndTimes Macro
'    Macro recorded 20/02/2008 by Gerard Meszaros
'
    Rows("3:8").Select
    Selection.Sort Key1:=Range("F4"), Order1:=xlAscending, _
        Key2:=Range("D4"), Order2:=xlAscending,  _
        Key3:=Range("E4"), Order3:=xlAscending,  _
        Header:=xlGuess, OrderCustom:=1, MatchCase:=False, _
        Orientation:=xlTopToBottom,  DataOption1:=xlSortNormal, _
        DataOption2:=xlSortNormal, DataOption3:= xlSortNormal
End Sub
```

Note: Recording contains results of Dialog Box, *not* user actions *in* Dialog Box.

Test Architecture – MS Excel



Goals Scorecard – Excel Macros

- **Fully Automated & Self-Checking**
  - Not self-checking, unless we record explicit checkpoints into the test
- **Repeatable**
  - Yes, running macro invokes key functionality
- **Robust**
  - Depends on what we depend on.
  - Bad: Hard-coded row & column references will break when data layout modified
- **Maintainable**
  - Good: No UI interactions
  - Bad: Hard-coded row & column references

Record, Refactor, Playback

- **Replace duplicated sequences of steps**
  - Calls to utility methods/functions/macros
  - Parameterize if necessary
- **Replace obtuse literals with symbols**
  - Make them intent revealing
  - Symbol Constants, Variables or Calculated Values

If you find yourself refactoring every test the same way, modify your test recorder to record it that way!

Refactored Recording

```
Sub SortByDayAndTimes_Refactored()
   Call SortDataBy3Columns( "DayColumn", "StartTimeColumn", _
         "EndTimeColumn")
End Sub

Sub SortDataBy3Columns(ColumnRange1, ColumnRange2, _
                       ColumnRange3)
    Range("SortData").Select
    Selection.Sort _
      Key1:=Range(ColumnRange1), Order1:=xlAscending, _
      Key2:=Range(ColumnRange2), Order2:=xlAscending, _
      Key3:=Range(ColumnRange3), Order3:=xlAscending, _
      Header:=xlGuess, OrderCustom:=1, MatchCase:=False, _
      Orientation:=xlTopToBottom, DataOption1:=xlSortNormal, _
      DataOption2:=xlSortNormal, DataOption3:=xlSortNormal
End Sub
```

Case Study – Project "Safety"

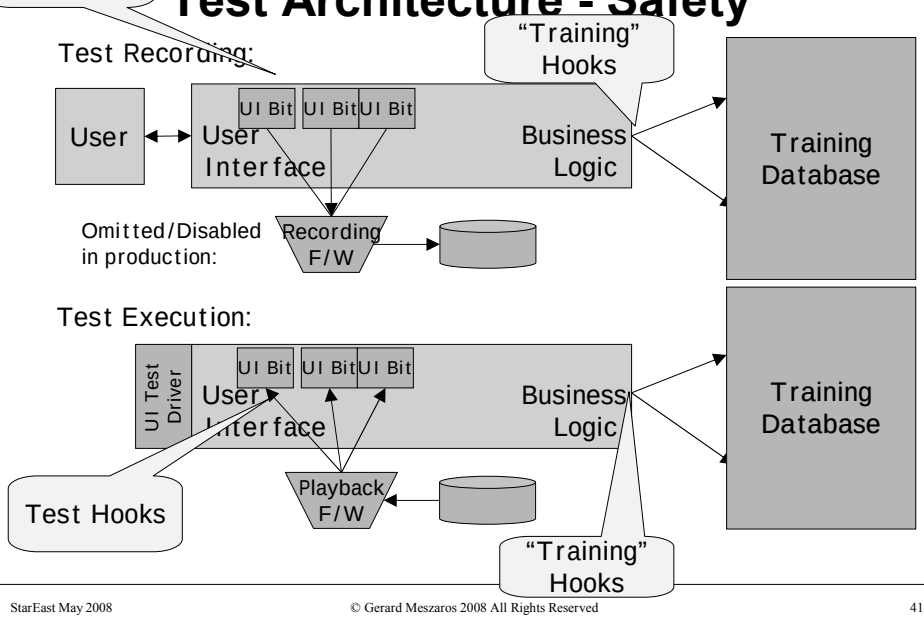
- **Re-engineering of an existing "safety involved" system**
  - Originally built in C; moving to C++
  - Porting from OS/2 to Windows 2000
  - Replacing home-grown with commercial middleware
- **Poor separation between presentation and business logic**
  - Automates very complex business rules
- **Antique UI Technology**
  - DOS-style text-based windows drawn with characters
  - No commercial R&PB tools available on both OS/2 and W2K

More Factors

- **Needed R&PB tool that worked on OS2 and Windows**
  - No commercial tools available across both platforms
- **Application had "Training Mode" that redirected I/O from production databases**
  - We leveraged this to make tests deterministic by controlling otherwise "random" inputs

Build Record/Playback into Your Application

Test Architecture - Safety



Build Record/Playback into Your Application

Sample R&PB Test Hooks

```
if (playback_is_on()) {
  choice = get_choice_for_playback(dialog_id,
             choices_list);
} else    {
  choice  = display_dialog(choices_list, row,
             col, title, key);
}

if (recording_is_on())  {
   record_choice(dialog_id, choice_list,
             choice, key);
}
```

Sample R&PB Test Hooks

```
if (playback_is_on()) {
  choice = get_choice_for_playback(dialog_id,
             choices_list);
} else    {
  choice  = display_dialog(choices_list, row,
             col, title, key);
}

if (recording_is_on())  {
   record_choice(dialog_id, choice_list,
             choice, key);
}
```


Sample R&PB Test Hooks

```
if (playback_is_on()) {
  choice = get_choice_for_playback(dialog_id,
             choices_list);
} else    {
  choice  = display_dialog(choices_list, row,
             col, title, key);
}

if (recording_is_on())  {
   record_choice(dialog_id, choice_list,
             choice, key);
}
```

Sample Test Recording – Raw XML

User Input

Previously recorded choices

Actual choices plus test results

```
<command seqno="2" id="Supply Create">
  <field name="select-supply" type="selection">
    <used-value>Create train</used-value>
    <expected>
      <value>Create train</value>
      <value>Create gang</value>
    </expected>
    <actual>
      <value status="ok">Create train</value>
      <value status="ok">Create gang</value>
    </actual>
  </field>
  <field name="rtc-initials" type="output">
    <expected>HDM</expected>
    <actual status="ok">HDM</actual>
  </field>
  <field name="engineno" type="input">
    <used-value>9595</used-value>
    <expected></expected>
    <actual status="ok"/>
  </field>
```


Raw XML for "Designation" Field *

```
<field name="designation" type="selection">
    <used-value>DIRECTIONAL</used-value>
    <expected>
      <value>DIRECTIONAL</value>
      <value>WORK</value>
      <value>PSGR</value>
      <value>PLOW</value>
      <value>PLOW WORK</value>
      <value>ENG</value>
    </expected>
    <actual>
      <value status="ok">DIRECTIONAL</value>
      <value status="ok">WORK</value>
      <value status="ok">ENG</value>
      <value status="ok">PSGR</value>
      <value status="surplus">MIXED</value>
      <value status="ok">PLOW</value>
      <value status="ok">PLOW WORK</value>
    </actual>
  </field>
```

Sample Test Recording - Formatted

2. Supply Create

Field Name	Type	Used Value	Default or Choices Value(s)	
select-supply	selection	Create train	Create train Create gang	ok ok
rtc-initials	output		HDM	ok
engineno	input	9595		ok
designation	selection	DIRECTIONAL	DIRECTIONAL WORK ENG PSGR MIXED PLOW PLOW WORK	ok ok ok ok surplus ok ok
direction	selection	NORTH	SOUTH NORTH	ok ok
shortname	output		X 9595 N	ignore

Sample Test Runner

```
// Check for recording output file
if (can_open_file( output_file)) {
  set_recording_is_on( output_file);
}
```

set up for recording

```
// Check for playback input file
if (can_open_file( input_file )) {
  set_playback_is_on( input_file );
  command = get_next_command( input_file );
  while ( command != nil ) {
    execute( command );
    command = get_next_command( input_file );
}
```

execute commands in playback file

```
// read user input from standard in
```

let user take over

Goals Scorecard – Project Safety

- **Fully Automated & Self-Checking**
  - Self-checking because expected outputs are verified
- **Repeatable**
  - Tests start from a known state in Training Mode
- **Robust**
  - Yes, Training Mode isolated SUT from Context Sensitivity
  - Playback tool customized to ignore time/date
- **Maintainable**
  - Tests are readable and easily rerecorded when business logic changes

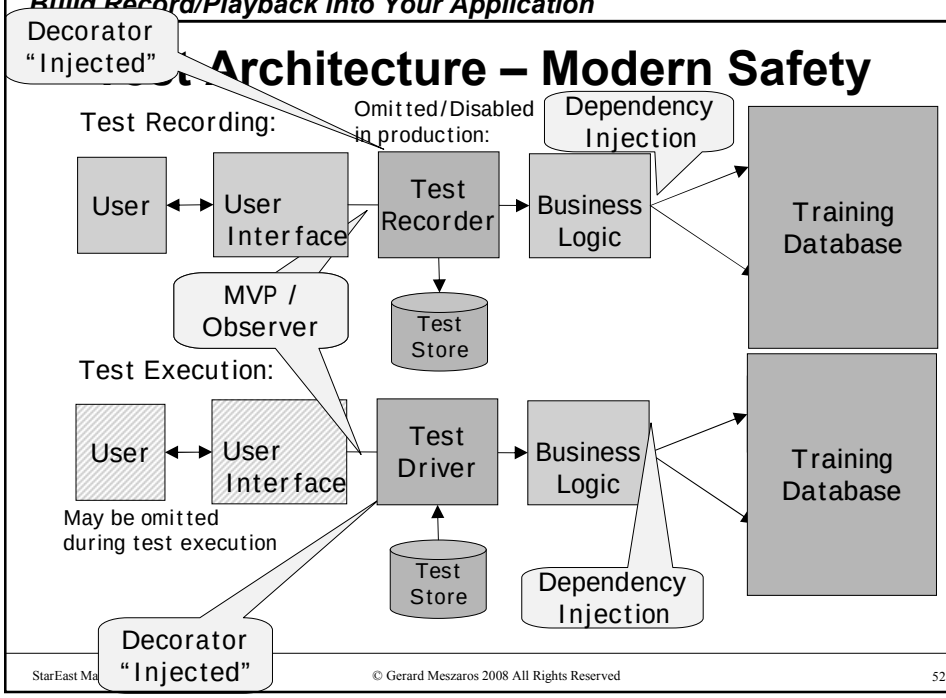
Test Sensitivity Strategy

Sensitivity	Strategy
Interface Sensitivity	•Prevented by building the R&PB capability directly into the application. •Visual changes to UI have no impact on tests.
Data Sensitivity	• Avoided by carefully controlling the data. • Playback tool starts system with a known starting point in the database.
Context Sensitivity	• Avoided by using "trainer mode" which "stubbed out" all the systems it interacted with. • Playback tool customized to ignore time/date
Behavior Sensitivity	• Changes in input validation or business rules likely to fail existing tests during playback. • Accommodated by freezing affected functionality for the duration of the re-engineering project. • Use a Test Regeneration strategy to improve lifetime.

Case Study – Modernized "Safety"

- **Hypothetical example of a modern application designed for testability**

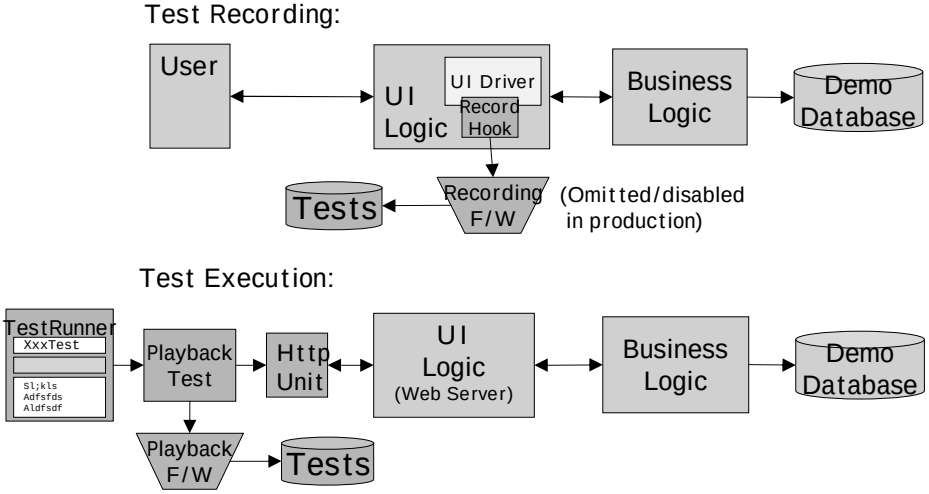
Test Architecture – Modern Safety



Case Study – Project "Billy"

- **A Billing System Demo Project**
  - Capability demo for client – cannot have failures!
- **Web-based presentation**
- **Transform View Architecture**
  - Façade generated XML
  - Servlet transforms XML to HTML using XSLT
- **Demo UI was built in a hurry without any tests**
  - Test Recording & Playback was retrofitted

Test Architecture



Sample XML Recording

```
<CommandLog>
   <Exchange>
       <Request>action=generateInvoices&endDate=01/03/05</Request>
       <FinalResult>/demodashboard.jsp</FinalResult>
   </Exchange>
   <Exchange>
      <Request>action=getAllInvoices </Request>
      <IntermediateResult>
         <?xml version="1.0" encoding="UTF-8"?>
            <ArrayList>
               <invoice number="24"> ...  </invoice>
            </ArrayList>
      </IntermediateResult>
      <FinalResult>
         <html>
            <head>   <title>Display All Invoices</title> </head>
            <body bgcolor="#FFFFFF"> ...  </body>
         </html>
      </FinalResult>
   </Exchange>
</CommandLog>
```

Raw XML Output

Formatted Html Output

Test Sensitivity Strategy

Sensitivity	Strategy
Behavior Sensitivity	**Behavior was not being changed.**
Interface Sensitivity	**Accepted as the cost of automating these tests. We re-recorded when demo changed.**
Data Sensitivity	**Avoided because all the data for each run was loaded from files during the demo.**
Context Sensitivity	**Avoided by controlling the system clock stub via the user interface**

Outline

1. **Test Automation Strategies**
2. **Test Automation Challenges**
3. **Built-in Test Recording Case Studies**
4. **Techniques for Mitigating Challenges**
5. **Implementing Built-In Test Automation**

Dealing With Smells & Sensitivities

- **Must be addressed to ensure success of automated testing**
- **Choices will vary based on your context**
- **What has worked for us:**
  - Building R&PB into our application
  - Using Open Source test tools, mostly Scripted Test
    - » **e.g. Watir, Seleniumn**
  - Using commercial R&PB tools
    - » **e.g. QuickTest**
    - » **More for testing GUI than for business logic**

Avoiding Interface Sensitivity

- **Use Stable Interfaces**
  - Bypass Presentation Layer (UI)
    - » **Subcutaneous Test**
  - Backwards compatibility of changes to used interface
    - **e.g. Façade**
- **Use appropriate level of abstraction**
  - Write/Record tests at "intent revealing" level
    - » **Use Built-in Test Recording**
  - Hide non-essential parts of SUT API from test via
    - » **Creation Method**
    - » **Finder Method**
    - » **Verification Method**

Avoiding Data/Context Sensitivity

- **Minimal Fresh Fixture**
  - Build a Fresh Fixture for each test or test run
  - Custom design it for each test.
  - Avoid a Standard Fixture that could become a Fragile Fixture
- **Test Stubs**
  - Replace the need for real fixture by using a Test Stub to provide indirect inputs

Verifying Expected Results

- **Can also record the system's responses into the test script and verify them when running the tests.**
- **Excel is not a good example because no built-in support for checkpoints or assertions**
  - Need to hand-code them into macro or build assertion commands that can be recorded
- **Works best with Built-In Recording**
  - Because what needs to be checked vs ignored is application specific

Avoiding Unmaintainable Tests by Choosing the Recording Level

- **Commercial tools record tests at the UI dialog level.**
  - This is very verbose and obscures the intent of the test(er)
- **Picking the right level of abstraction for the interaction is key to making tests readable and editable.**
  - E.g. Create New Customer("Acme", "Tucson", "AZ", …)
- **vs.**
  - NavigateTo("New Customer Screen")
  - FillField("Customer Name", "Acme")
  - FillField("City", "Tucson")
  - FillField("State", "AZ")
  - PressButton("Add Customer")…

Upgrading Recorded Test Scripts

- **When SUT Behavior changes, we expect tests to fail.**
- **How can we minimize the effort/errors involved in recreating the test scripts?**

Solution: Record & Playback are not mutually exclusive:

1. Record new test script while replaying old script
2. Verify failures in original test script are valid
3. Check in new version of script replacing original

Semi-Automated Testing

- **Use a Recorded Test to put SUT into a known state**
- **Do manual / exploratory testing from that state**
- **Make it possible to invoke other test scripts at any time to**
  - FastForward to another state
  - Clean Up after a test

Test Recording Code Strategy

- **Need a way to capture the users actions and inputs when tests are being recorded. Three major options:**
- **Test Hooks**
  - Hard-coded test recording logic within application
  - Can leave them in (as "feature") or use conditional compilation to strip them out.
- **Test Recording Decorator**
  - Objects injected into application only when testing
- **Aspect-Oriented Programming (AOP)**
  - Use an AOP tool to "weave" the hooks into the code after the fact

Test Hook

```
choice  = display_dialog(choices_list, row,
            col, title, key);
}
if (recording_is_on())  {
   record_choice(dialog_id, choice_list,
            choice, key);
}
```

Test Hook

Pros & Cons:

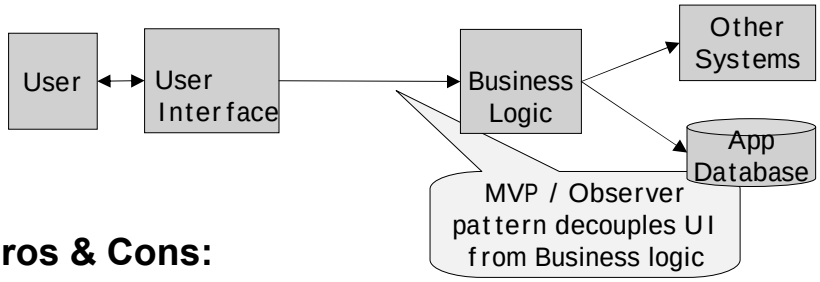
+ Very simple

– May be many Test Hooks to maintain

– Test Code in Production introduces risk

Build Record/Playback into Your Application

Test Recording Decorator



Pros & Cons:

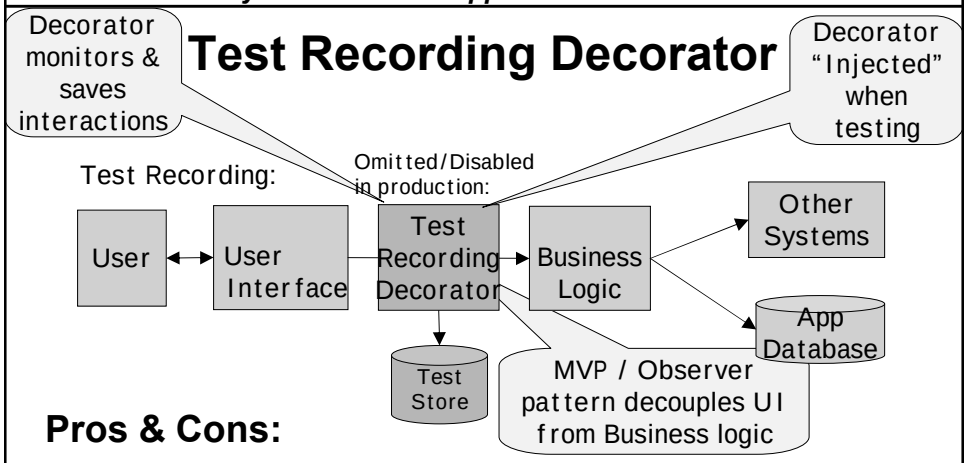
+ Very clean; No Test Code in Production

– Needs to be designed in

– Requires object-oriented coding experience

Build Record/Playback into Your Application

Test Recording Decorator



Pros & Cons:

+ Very clean; No Test Code in Production

– Needs to be designed in

– Requires object-oriented coding experience

Managing Dependencies

- **Need a way to deal with non-deterministic interactions with other systems or databases**
- **Make them deterministic**
  - Set them up from within the test itself
  - Doesn't address changes in code
- **Replace them with deterministic equivalents (Test Doubles)**
  - Fake Object with simple but equivalent behavior
  - Test Stub with canned responses
  - Mock Object with expectations loaded by test

For more details on Test Doubles see:
xUnit Test Patterns – Refactoring Test Code

Installing Test Doubles

- **Test Hook**
  - Hard-code hooks to call Test Double instead
  - if (testing) then …
- **Dependency Injection**
  - Test passes name of Test Double to application at runtime as an argument
- **Dependency Lookup**
  - Application looks up dependency at runtime via 3rd party
  - Test passes Test Double to 3rd party before running application

For more details on installing Test Doubles see:
xUnit Test Patterns – Refactoring Test Code

Outline

1. **Test Automation Strategies**
2. **Test Automation Challenges**
3. **Built-in Test Recording Case Studies**
4. **Techniques for Mitigating Challenges**
5. **Implementing Built-In Test Automation**

Implementing Built-In R&PB

- **Cannot be done by Test Organization alone**
  - Requires collaboration with Development throughout development lifecycle
- **Why should Development help Test?**
  - Quality is a joint deliverable
  - Test & Development need to collaborate to achieve quality
  - Automated regression testing acts as “safety net” for development

Win-Win Situation!

What Can Test Offer Development?

- **Help Development understand requirements**
  - By clarifying them via concrete examples (tests)
  - Providing these tests before development
    - » **Need not be automated yet**
    - » **Need to be concrete examples c/w sample data**
- **Help Development assess quality**
  - As soon as possible after software is built
    - » **By testing the code immediately**
  - Whenever changes are made
    - » **By providing automated regression tests**

This should give Development the necessary incentive to help

Recap of Smells

- **Fragile Tests**
  - Interface Sensitivity
  - Behavior Sensitivity
  - Data/Context Sensitivity
- **Unrepeatable Tests**
  - Shared Fixtures
- **Lack of Test Maintainability**
  - Verbose Test
  - Hard-Coded Values
- **Slow Test Execution**
  - Asynchronous Interfaces
  - Database dependencies

All Described in book: xUnit Test Patterns

Recap of Test Patterns

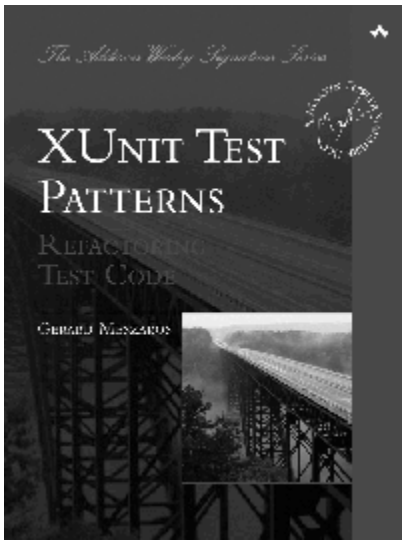
- **Built-In Record & Playback**
  - Record, Refactor, Playback
- **Test Hook – xUnit Test Patterns**
- **Decorator – Design Patterns**
- **Dependency Injection– xUnit Test Patterns**
- **Dependency Lookup– xUnit Test Patterns**

Conclusions

- **R&PB is a viable test automation strategy when circumstances permit**
- **Test Strategy must address the 4 Fragile Test "sensitivities"**
  - Interface, Behavior, Data, Context
- **Consider building R&PB into the application to reduce impact of the sensitivities**
- **Tests must still be planned to ensure test coverage**

More on xUnit Patterns & Smells

- **Book:**
  xUnit Test Patterns
  Refactoring Test Code
  by: Gerard Meszaros
  – published by Addison Wesley
  – available here!
- **Website:**
  http://xunitpatterns.com
  With handy links to purchase



Thank You!
Gerard Meszaros
StarEast2008@gerardm.com

Other Useful Books

- **Working Effectively with Legacy Code**
  – Michael Feathers
- **Fit for Software Development**
  – Rick Mugridge, Ward Cunningham
- **Refactoring - Improving the Design of Existing Code**
  – Martin Fowler plus contributors
- **Design Patterns: Reusable Elements of Design**
  – Erich Gamma, Richard Helm, Ralph Johnson, John Vlisides