Design for Testability

Bret Pettichord

bret@pettichord.com
www.pettichord.com

Testability is Key

- Three Keys to Test Automation
  - Partnership with Development
  - Commitment to Automation
  - Early Involvement

Success with test automation depends on testability!

Agenda

- What is Testability?
- Testability Features
- Making the Case for Testability
- History and Prospects

Acknowledgements

- **This material borrows from discussions with colleagues**
  - **Austin Workshops on Test Automation (AWTA)**

    AWTA 2 (Jan 2001) focused on software design for testability. Participants were Alan Jorgensen, Allen Johnson, Al Lowenstein, Barton Layne, Bret Pettichord, Brian Tervo, Harvey Deutsch, Jamie Mitchell, Cem Kaner, Keith Zambelich, Linda Hayes, Noel Nyman, Ross Collard, Sam Guckenheimer and Stan Taylor.
  - **Los Altos Workshops on Software Testing (LAWST)**

    LAWST 6 (Feb 1999) focused on testability assessment. Participants were Ned Young, Jack Falk, James Bach, Jeff Payne, Doug Hoffman, Noel Nyman, Melora Svoboda, Chris Agruss, Brian Marick, Payson Hall, Bret Pettichord, Johanna Rothman, Dave Gelperin, Bob Johnson, Sandra Love, Cem Kaner, Brian Lawrence and III.

What is Testability?

- Visibility
  - Our ability to observe the states and outputs of the software under test
- Control
  - Our ability to provide inputs and reach states in the software under test

A Broader View of Testability

- **Operability —** The better it works, the more efficiently it can be tested.
- **Controllability —** The better we can control it, the more the testing can be automated and optimized.
- **Observability —** What we see is what we test.
- **Simplicity —** The less there is to test, the more quickly we can test it.
- **Understandability —** The more information we have, the smarter we test.
- **Suitability —** The more we know about the intended use of the software, the better we can organize our testing to find important bugs.
- **Stability —** The fewer the changes, the fewer the disruptions to testing.

"Heuristics of Software Testability" James Bach, 1999.

Testability Features

Features that have been used to improve the visibility or controllability of software

Visibility Basics

- Developers must provide full viewing access to testers
  - Code
  - Change records
  - Design documents
- Someone on the testing team needs to know how to read code

Verbose Modes Aid Visibility

Example: Unix mail

```
> mail -v bret@pettichord.com
Subject: testability example
Sample text.
.
Cc:
bret@pettichord.com... Connecting to
    mx.io.com. via relay...
220-deliverator.io.com ESMTP Sendmail
    8.9.3/8.9.3; Fri, 12 Jan 2001
    15:34:36 -00
220 Welcome to Illuminati Online,
    Fnord!
>>> EHLO eris.io.com
250-deliverator.io.com Hello
    IDENT:wazmo@eris.io.com
    [199.170.88.11], pleased tu
250-8BITMIME
250-SIZE 5000000
250-DSN
250-ONEX
250-ETRN
250-XUSR
250 HELP
>>> MAIL From:<wazmo@eris.io.com>
    SIZE=67
250 <wazmo@eris.io.com>... Sender ok
>>> RCPT To:<bret@pettichord.com>
250 <bret@pettichord.com>... Recipient
    ok
>>> DATA
354 Enter mail, end with "." on a line
    by itself
>>> .
250 PAA07752 Message accepted for
    delivery
bret@pettichord.com... Sent (PAA07752
    Message accepted for delivery)
Closing connection to mx.io.com.
>>> QUIT
221 deliverator.io.com closing
    connection
```

Monitoring and Logging

- Diagnostic features can allow you to
  - View history and details of transactions
  - View the state of running methods, threads or processes
  - Attach a debugger and get a stack trace
  - Get reports of unusual behavior
- Learn about the logging available from web servers, databases and other standard system components

What to Log

- Error messages
- Source of error
- Usage profiles
- Resource utilization
- System messages

What Can You Do With Logs?

- Logs may report internal errors before they appear at the system level.
- Analyzing logs can help you reproduce and isolate defects.
- Logged information provides context than can be very helpful to developers.
- Logging can give you information about your tests. It can also help give information about customer usage.
- Logs can give you valuable information about test coverage.

User Interface Automation

Testability issues that commonly arise with the use of popular test tools

- Compatibility with application development platform
- Duplicate window and screen names
- Detached labels for controls and fields
- Windows with non-standard close methods
- Custom controls
- Dynamic class names

Custom Controls

- Custom user interface controls often raise serious testability problems with GUI test drivers.
- Ensuring testability usually requires:
  - Adding methods to report necessary information.
  - Customizing test tools to make use of these methods.

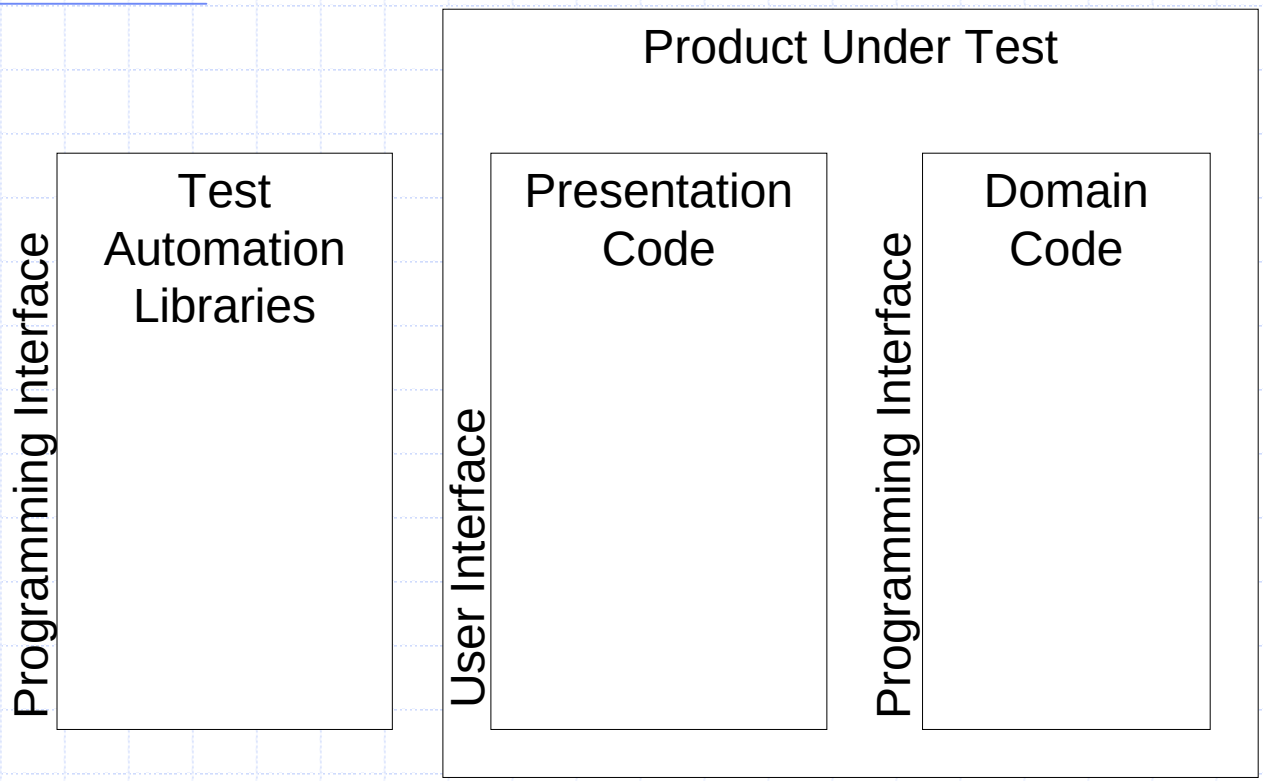
Test Interfaces

- Interfaces may be provided specifically for testing.
  - Excel
  - Xconq

Any interface is easier to automate than a GUI.

- Existing interfaces may be able to support significant testing.
  - InstallShield
  - Autocad
  - Interleaf
  - Tivoli
  - *Any defined client/server interface*

Test Interfaces



- Which interface will be easier to test?

Installation and Setup

- Notification when installation has completed successfully.
- Ability to verify installation
- Ability to programmatically create sample records
- Ability to run multiple clients, daemons, or servers on a single machine.
- *Ease of installation can also help*

Diagnostic Features

These make the software behavior more visible.

- **Assertions**. These logical statements in the code make assumptions explicit. If false, there must be a bug. Typically assertion checking is only done during testing and debugging.
- **Event triggers**. Provide identifying signals when tasks begin and end.
- **Database Integrity Checks**. Ongoing tests for database corruption, making corruption quickly visible to the tester.
- **Code Integrity Checks**. Quick check to see whether code has been overwritten.
- **Memory Integrity Checks**. Check for wild pointers, other corruption.
- **Access methods**. Special access methods allow viewing component state.

Control Features

- **Exception seeding**. Instrument low level I/O code to simulate errors. This can make it much easier to test error handling.
- **Test points**. These allow data to be inspected, inserted or modified at points in the software. Useful for dataflow applications.
- **Automatically filling memory**.

Instrumenting Test Automation

- **Add Logging**. This can be used to track which interface elements have been tested and in what combinations.
- **Modify Methods**. This can be used to change the standard methods for accomplishing tasks. This is useful when there are multiple ways of doing something and you suspect that a bug may be lying in one of them.

Making the Case for Testability

Convincing Developers and Managers

Convincing Developers

- Developers want to solve problems. This is what they love to do.
- Talk their language.
  - They don't know what we want until we ask.
  - We need put our requests in terms they can understand.
- Ask early.
  - The earlier you talk to them, the more realistic they are about the need and challenges of testing.
  - Developers are open to alternative designs early in the design process.
- Give them a chance to do their magic.

Convincing Managers

- What are we asking for?
  - Some technical parity – staff, training
    - So we can talk development's language
  - Early involvement
- Three approaches
  - Reduce costs
  - Improve quality
  - Facilitate control

At the Design Review

- Things to ask for
  - Testability features
  - Access to code
    - Also: configuration management records
  - Access to existing information
    - Design documents, interface specifications, algorithm descriptions, error catalogs
  - Adherence to interface standards
  - Explanations of existing architecture and features
    - E.g. diagnostic features
  - Agreement to treat test-block defects with priority

At the Design Review (2)

- Asking for these can lead to trouble
  - Better design
  - Better discipline
  - Better process
  - Better documentation
  - Better communication
  - Hand holding

History and Prospects

Testability initiatives in hardware design and possibilities for improved software testability

Component-based Architectures

- Build systems out of components. Components have well-defined interfaces and are tested in isolation. Components are typically provided by separate companies or divisions.
- Emerging software component frameworks.
  - J2EE – Java based
  - Dot-Net – Windows based

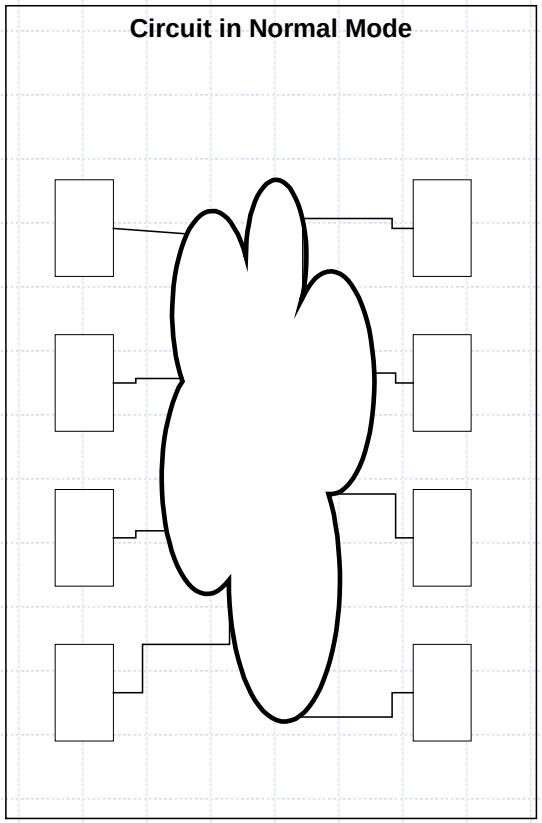
Design Rules

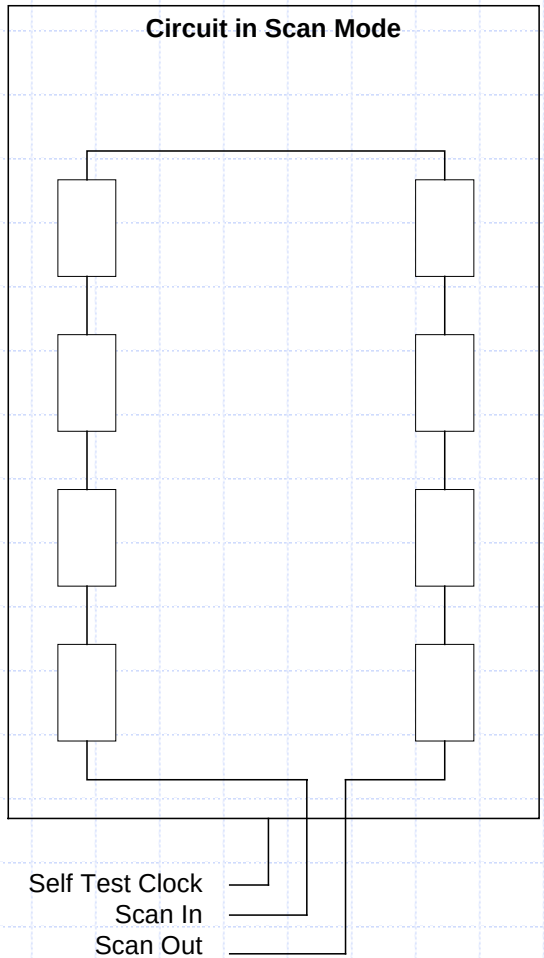
- Enable modularity by reducing the interdependency within designs.
  - **Architecture**. Identification and roles of modules.
  - **Interfaces**. How modules shall interact.
  - **Integration protocols and testing standards**. Procedures for determining whether the system works and whether modules adhere to the design rules.

Design for Testability

- An initiative in the computer hardware industry in the 1980's.
- Part of a larger effort called **Error Detection and Fault Isolation** (EDFI). This required that running systems be able to detect errors and isolate them to the originating components.
- It facilitated a component marketplace. Previously, edge vectors had been used for testing but the increasing size of chips made this no longer feasible.
- EDFI was an executive mandate. No comparable event has yet occurred in the software world... yet.

Boundary Scan Design





Allows hardware components to test themselves in place

Source: Al Lowenstein

Interface Contracts

- Each component was responsible for validating inputs and outputs. This reduced performance and took up valuable real estate (consequently reducing yield).
- **Design by Contract** provides a schema for doing the same thing with software.

Design by Contract

- This technique requires that assertions be defined for functions.
  - **Preconditions** apply to inputs. Violations implicate **calling** functions.
  - **Postconditions** apply to outputs. Violations implicate **called** functions.
- This effectively solves the oracle problem for testing. What remains is the task of selecting a sufficiently varied test inputs.

Language and Interfaces

- Test Interfaces
  - Test interfaces were provided. These were only to be used for testing.
  - How could we enforce the use of certain software interfaces for testing only?
- Resets
  - Hardware had to be able to reset itself.
  - With software, a true reset typical requires a reinstall.
- Standard Design Language
  - Common design language (HDL) allowed common tools to be developed.
  - Is UML the equivalent for software? Currently it is not sufficiently detailed to contain the complete design – rather only an overview.

Concluding Thoughts

- Engage your developers early.
- You may be better off than you realize.
- Things will get easier with time.