

Using Failure Modes to Power Up Your Testing



Dawn Haynes

Senior Trainer & Consultant



STARWest - October 2008

What inspired this talk?

The observation that we seem to write (and release) the same bugs over and over

Research that led to the “How to Break Software” series of books

The notion that we can learn from our mistakes

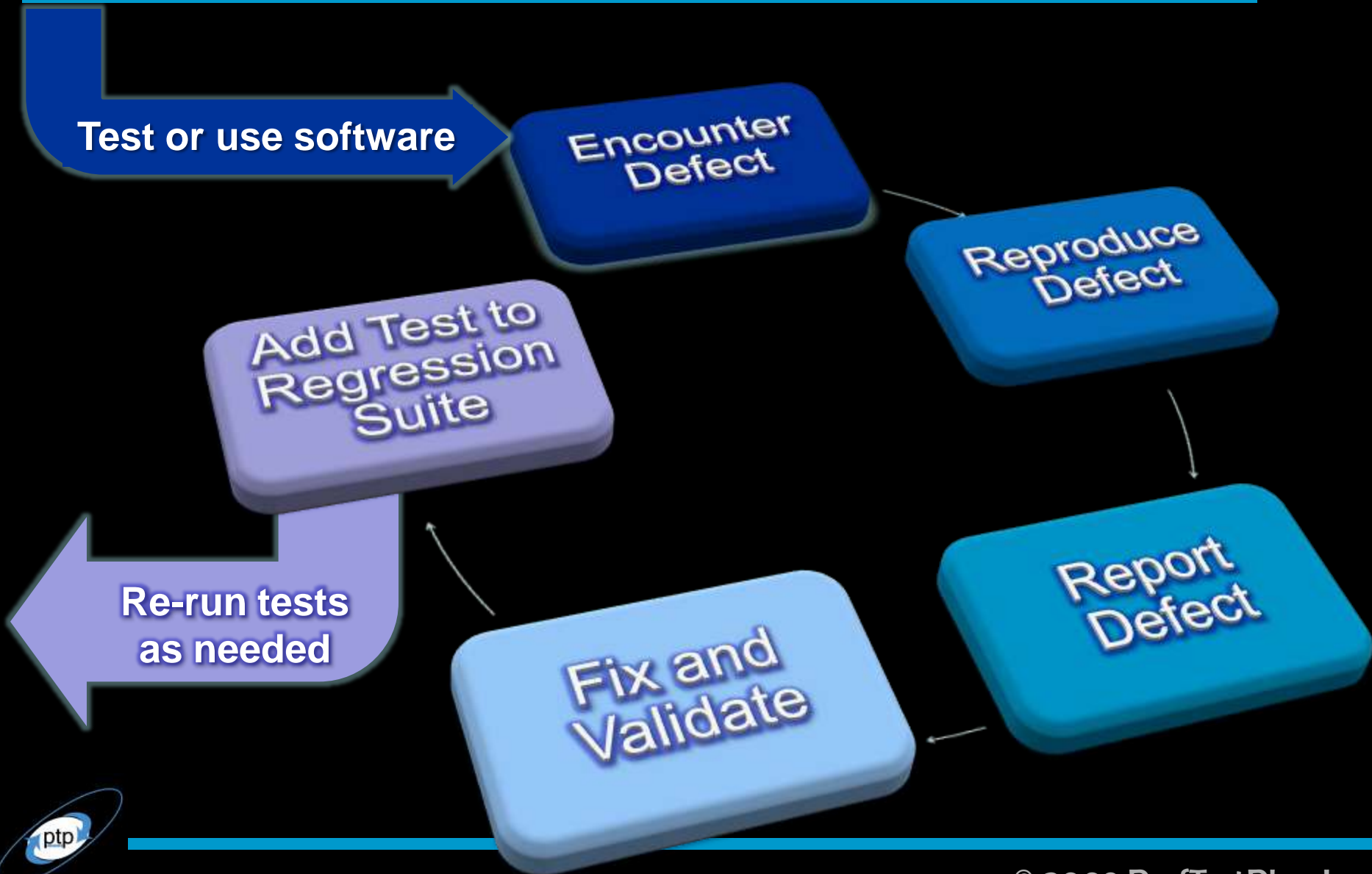


Attributions & Credits

- Common software errors and analysis tools
 - ◆ Glenford Myers, James Whittaker, Elisabeth Hendrickson, Hugh Thompson, James Bach, Cem Kaner
- Other sources
 - ◆ SQE, IEEE, ISO, ASQ
- Other references - disclaimer
 - ◆ Any missing references to the original author or source is unintentional if pointed out will be promptly corrected
 - ◆ Some errors and omissions may be due to a lack of proper citing on some Web sites, books, articles and other materials I have used



Defects – Typical Approach



Results of this Approach

Perceived Software Insurance
(prevent known bug escapes)

Find
ONE
bug



Fix
ONE
bug



Create
ONE
test



*Run the ONE
test over and
over again*



Repeated Test Syndrome

Potential Issues

- Tests may get stale (pesticide paradox)
- Will we have the time to run these tests?

Potential Benefits

- Can these tests be re-used to help expose other defects?
- Can we use failures instead of specs as guides to expand testing?

Can we do better?

- Through defect analysis, can we discover
 - ◆ Other ways to make the same bug happen?
 - ◆ Other places where the same bug happens?
- Adjust regression tests to be more powerful
 - ◆ Apply variations (data, speed, sequence, etc.)
 - ◆ Create a generic test instead of a specific version that blocks only one instance of a failure



Power Up Strategies

- Create new specific tests using
 - ◆ Failure modes and effects analysis (FMEA)
 - ◆ Cause-effect mapping / graphing
- Create new generic tests using
 - ◆ Research on technologies, functionality or processes
 - ◆ Reported defect patterns and trends



Failure Modes and Effects Analysis

- FMEA is intended to document:
 - ◆ a Failure
 - ◆ its Mode
 - ◆ its Effect
 - ◆ by Analysis
- in a cause-effect manner.



Simple Software FMEA Example

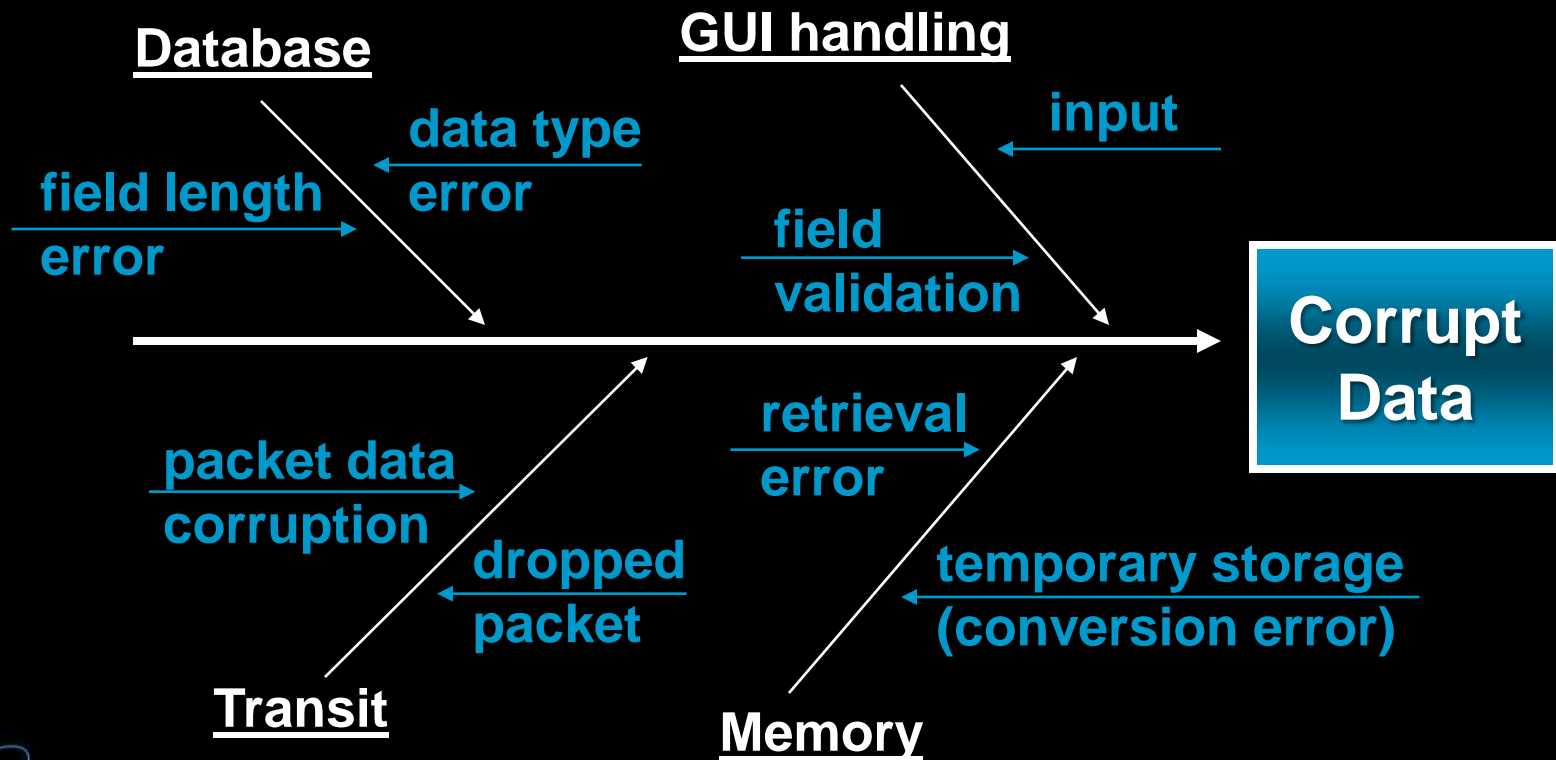
FUNCTION: View Search Results

Failure Mode	Undesired behavior	Sluggish response to client/UI
Causes	Failure causes or conditions	<ul style="list-style-type: none">• Low memory condition (client)• Slow network connection• Overloaded server
Effects	Possible results of failure occurring	<ul style="list-style-type: none">• User repeats operation• User abandons operation• User closes program during processing



Fishbone (Ishikawa) Diagram

- Map out (relate/group/list) possible ways for data corruption to occur



Cause-Effect Graphing

- Typically used to illustrate relationships between
 - ◆ Causes (inputs)
 - ◆ Effect (outputs)
- ... for the purpose of extracting test cases from detailed specifications
- Let's use this to analyze and map out actual or potential application behavior



Extracting Causes and Effects


Sendfile Command

Outcomes: if all arguments are correct, file is sent; otherwise error message is generated

Arg1= an existing file in the sender's home directory

Arg2 = name of the receiver's file server

Arg3 = the userid of the receiver



Causes	Effects
1. The first argument is the name of an existing file in the sender's home directory.	100. The file is successfully sent.
2. The second argument is the name of receiver's file server.	101. The sender obtains an error message.
3. The third argument is the receiver's userid.	

Table 2.1 - List of causes and effects

Cause-Effect Graph – Example

The first argument is the name of an existing file in the sender's home directory.

The second argument is the name of receiver's file server.

The third argument is the receiver's userid.

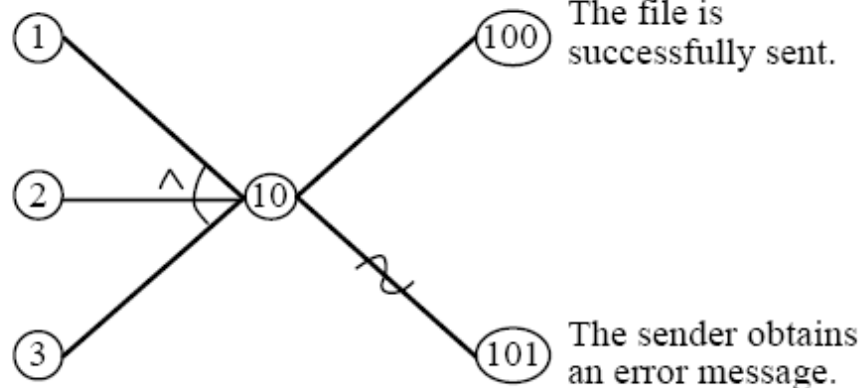


Figure 2.2 - The Cause-Effect Graph obtained

Mapped to a
decision table

	1	2	3	4	5	6	7	8
Causes								
1	1	0	1	0	0	1	1	0
2	1	0	0	1	0	1	0	1
3	1	0	0	0	1	0	1	1
Effects								
100	1	0	0	0	0	0	0	0
101	0	1	1	1	1	1	1	1

Figure 2.3 - Decision Table produced from CEG given in Figure 2.2 using Myers' rules

Cause-Effect "Style" Exploration

Cause	Category	Effect
Start program	Preferred	Normal start
	Sub-par operation	<ul style="list-style-type: none">• Starts with first-run operations in effect (wizard, help, etc.)• Starts with info/warning/error message(s)• Starts very slowly
	Failure case	<ul style="list-style-type: none">• Does not start but reports error message• Does not start or report error message

- What interacts with the primary cause (input, trigger, state, etc.) to create various effects?



What are some generic failures?

- Incorrect result returned
- Data corruption or truncation
- Slow response or hang
- Display errors
- Memory leak
- Crash or unexpected shutdown
- Incorrect error messages



Historical Failures

- Internal sources
 - ◆ Defects from formal testing groups
 - ◆ Developer submitted issues
 - ◆ Issues reported from other staff
 - ◆ Anomalies and intermittent failures
- External sources
 - ◆ Customer submissions (defects, requests)
 - ◆ Issues reported by beta testers
 - ◆ Reports from consultants or field engineers



Historical Failure Analysis

- Target defects or other incorrect behavior
 - ◆ Perform root-cause analysis if possible
 - ◆ Group similar items (by guess or by cause)
 - ◆ Look for patterns
- Try to create your own generic tests based on these patterns
 - ◆ Seek to create test ideas instead of single tests
 - ◆ Make lists (or search for lists/heuristics/guides)



Generic Failure Analysis

- Error guessing / bug hunting
- Software behavior and interfaces
- Technologies and architectures
- Processes, standards, SDLC used



Error Guessing

- Using experience as a guide
- Remembering frequent or spectacular failures and going on a hunt
- Every software tester has a bucket of these
 - ♦ Zero, null, blank, space, -1
 - ♦ Special characters, delimiters
 - ♦ ... and more



Software Behavior & Interfaces

- Generic capabilities
 - ◆ Input
 - ◆ Output
 - ◆ Data handling (storage & retrieval)
 - ◆ Computation
- Interfaces and integration points
 - ◆ GUIs, APIs, devices
 - ◆ Operating system, registry, file system
 - ◆ Other supporting software



Use Generic Software "Attacks"

- How to Break Software book
 - ◆ 17 user interface attacks
 - Long strings
 - Repeating tasks or inputs
 - Use default values
 - Interacting inputs or features
 - ◆ 6 file system interface attacks
 - ◆ Generic methods for attacking the operating system and supporting software
- Build your own list of Whack-a-Mole tests!



Technology Areas

- Programming languages
 - ◆ Java
 - ◆ C / C++
- Architectures
 - ◆ SOA
 - ◆ Web
 - ◆ Client/server
- Technical implementations
 - ◆ Security



Some Resources for Analysis

- Web

- ◆ How to Break Web Software (*Whittaker & Andrews*)

- Security

- ◆ How to Break Software Security
(*Whittaker & Thompson*)
 - ◆ Common vulnerabilities database
 - <http://cve.mitre.org>
 - ◆ CERT
 - <http://www.us-cert.gov/cas/alldocs.html>



Processes

- Software development lifecycle & methods
 - ◆ Agile, waterfall, iterative, V-model, etc.
 - ◆ Test driven development, unit testing
 - ◆ Coding standards, code reviews
- Formal processes and standards
 - ◆ Six Sigma, CMMI
 - ◆ ISO, IEEE
 - ◆ Regulatory and others (FDA, SEC, 508, i18N ...)



Process Analysis

- Seek out standards and processes
- Understand what they will catch or prevent
- Identify what they are not good at
- Look to reduce redundancy and duplication of effort
 - ◆ Employ the “trust but verify” technique
 - ◆ Reduce system test emphasis on covered areas
- Develop tests to cover the gaps



Power Up: *Create New Tests*

- Explore root causes and discover new tests that exhibit the same behaviors or failures
- Search for multiple outcomes from single actions to expose subtle variables not currently considered in testing scope
- Reveal interactions and connections between inputs, conditions and components to target tests in new areas



Power Up: *Create Generic Tests*

- Find repeated defects and genericize
- Use experience and go bug hunting
- Borrow existing generic tests
- Expose and exploit generic flaws
 - ◆ Related to technology implementations
 - ◆ Rooted in process implementations
 - ◆ Caused by narrowly focused standards



Wrap Up

- THANK YOU for joining me today!
- Questions? Feedback?
- Any relevant experiences to share?

- Contact info
 - Dawn Haynes
 - dhaynes@perftestplus.com
 - www.perftestplus.com

