

What your Mother Didn't Tell you about Testing

Meeting the Testing Challenges of the 21st Century

Gary P. Cort, Ph.D.

Vice President, Software Quality

Research In Motion, Limited

April 23, 2008

Agenda

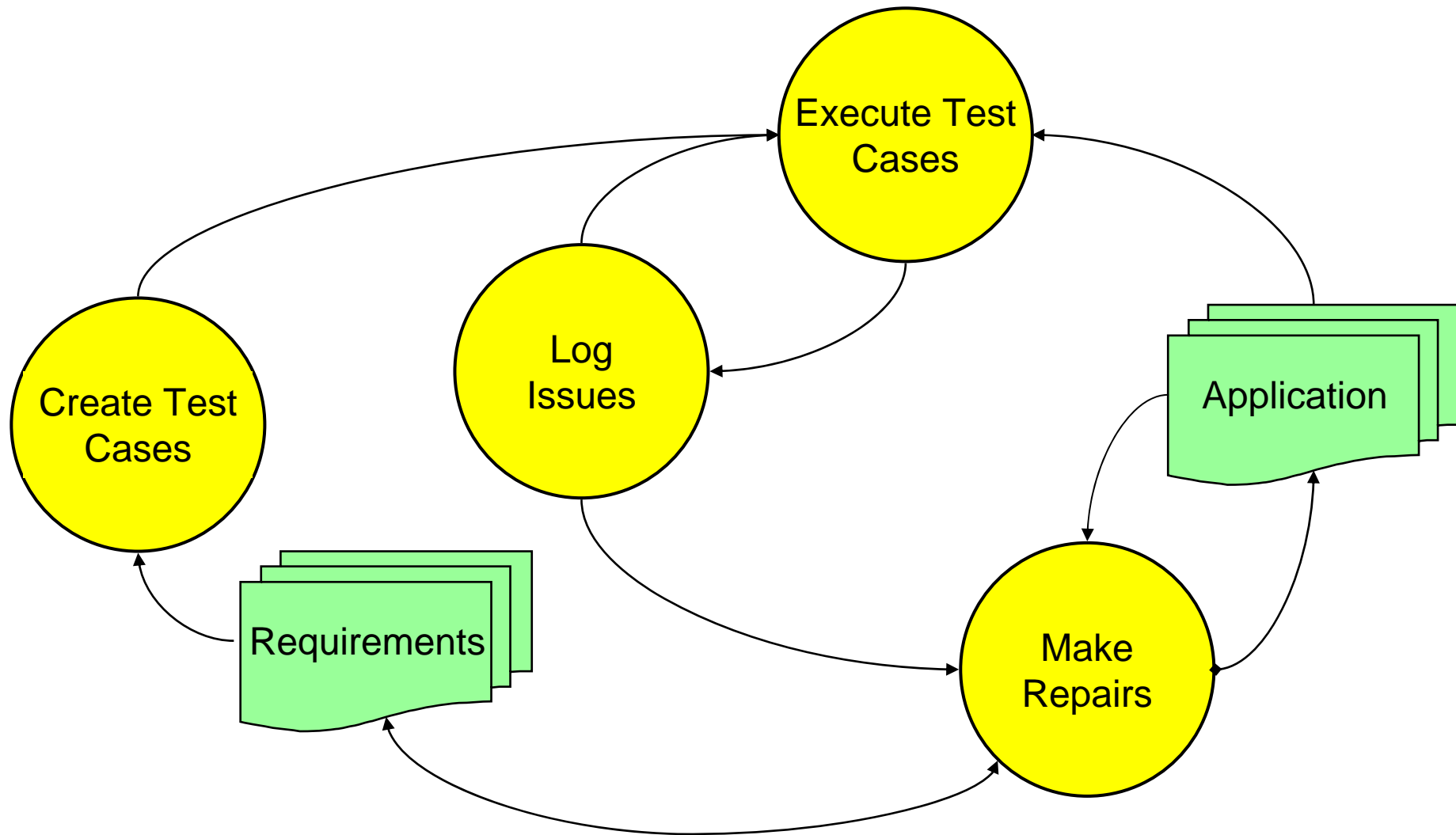
- The Challenges of Testing
- The Tyranny of Typical Practice
- Derivative Issues & Limiting Factors
- Stepping out of the Box
- Understanding the Problem
- Crafting an Effective Test Plan
- Keeping your Bearings: Earned Value
- Setting Testing Priorities: Cyclomatic Complexity
- Knowing when to Quit: Reliability Growth
- Summary

The Challenges of Testing

A program of testing is intended to effectively and efficiently identify faults in a system so that those faults can be eliminated before the system is deployed. To accomplish these goals the testing program must overcome

- Explosive size and complexity in the application
- High costs incurred by human-intensive execution
- Deficiency of critical information about the application
- Late-stage intervention resulting in high cost of repair
- Compressed time schedules

The Tyranny of Typical Practice



Derivative Issues & Limiting Factors

- Over-reliance on functional testing dilutes testing effectiveness
 - Easiest to pass
 - Hardest for which to write good test cases
 - Most difficult for which to obtain authoritative basis documentation
- Poor-quality requirements exacerbate risks
 - Usually arrive late in the development cycle
 - Often are inconsistent with the actual application
 - May have to be reconstructed from the application by the test team
- Software bugs severely limit testing effectiveness
- Expectation of comprehensive testing drives dysfunctional resource allocation

Stepping out of the Box

To make the problem more tractable, we must reject the assumptions that serve us poorly

- Abandon the expectation of comprehensive testing
 - Reduce testing scope to the essential few
- Re-focus test specification on the code – not the requirements
 - Scale your testing efforts to address the highest risk software components
- Emphasize testing performance over test execution
 - Focus on what you are trying to *accomplish* – not what you are trying to *do*
 - Ensure that you always know where you are

Understanding the Problem

- Test Strategy
 - Identifies and prioritizes key test activities
- Test Plan
 - Specifies the requirements for conducting the testing process and assigns roles and responsibilities for test-process execution
- Test Specification
 - Describes the test cases for each test activity, including test case ID, purpose, inputs, procedure, and expected outputs
- Test Report
 - Interprets the results of the testing and makes a recommendation regarding whether the software should be accepted on the basis of the test results

Crafting an Effective Test Plan

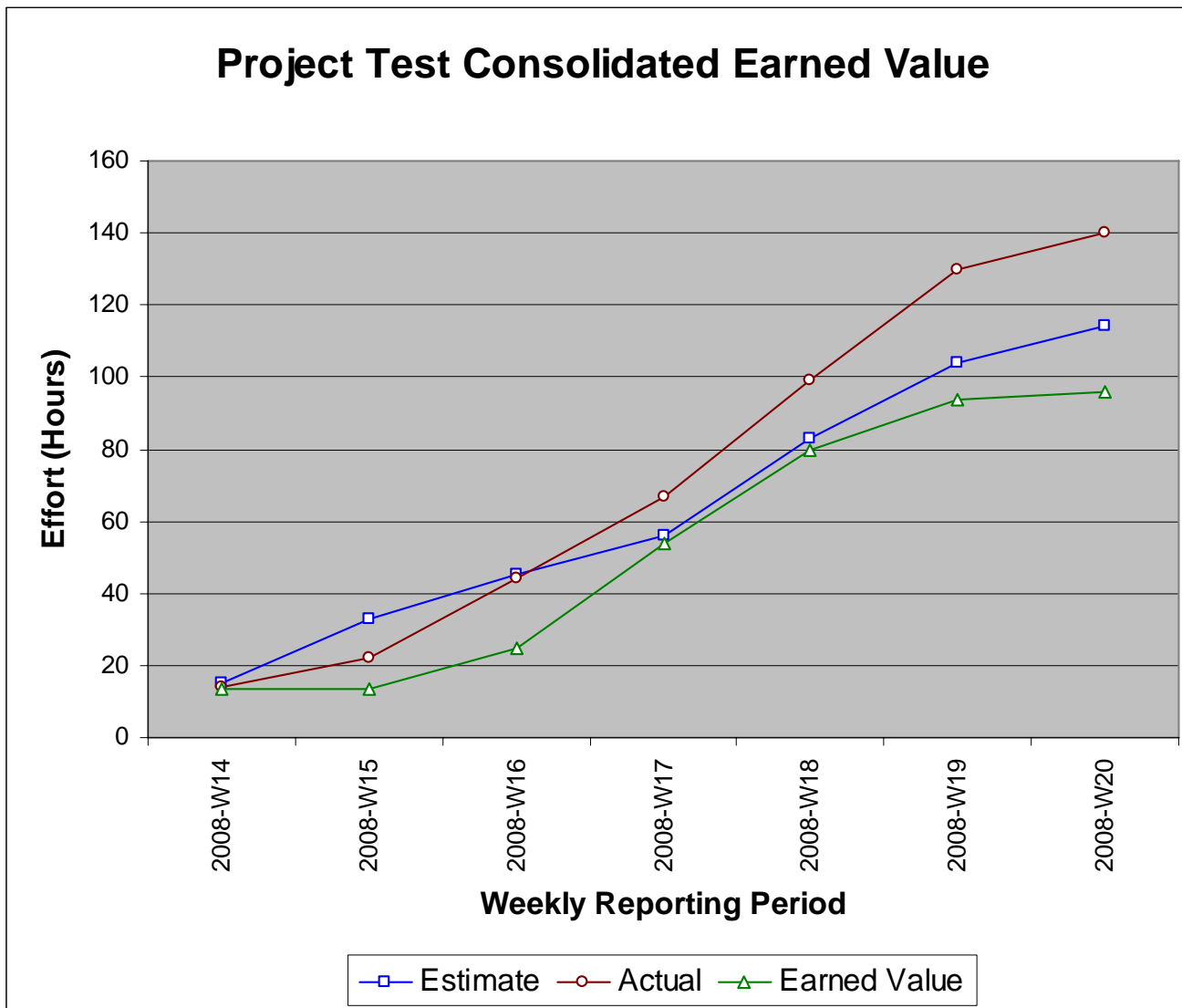
*As for any planning document, the **test plan** must address:*

- a WBS-based, resource-loaded schedule
- risks associated with the test program
- a plan for data management
- a resource management plan, including roles and responsibilities
- a plan for needed knowledge and skills
- a plan for involving relevant stakeholders
- a mechanism for insuring that the plan is realistic
- positive means for obtaining commitment to the plan

Keeping your Bearings – Earned Value (1)

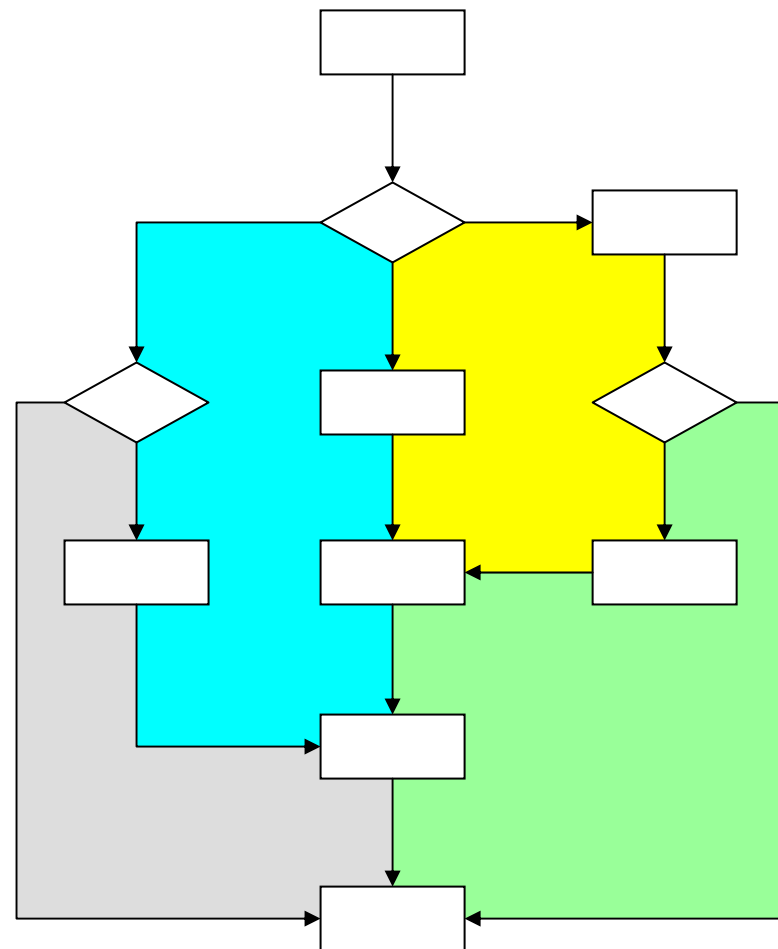
Week:	14	15	16	17	18	19	20	
Test Planning	15	10	2					27
	14	8	4					
Unit Test		8	6	1				15
			10	7				
Integration I			4	3	1			8
			8	4	2			
Integration II				7	6	3		16
				12	5	3		
Integration III					8	4		12
					14	7		
System Test					12	14	6	32
					11	21	7	
Test Reporting							4	4
							4	
Cumulative Plan	15	33	45	56	83	104	114	
Cumulative Actual	14	22	44	67	99	130	141	

Keeping your Bearings – Earned Value (2)



Setting Testing Priorities – Cyclomatic Complexity

- With finite testing resources and inflexible delivery deadlines, you cannot test everything.
- Use formal methods and measures to focus testing on the modules that need it the most.
- Drive the prioritization process from within the software being tested.
- The McCabe Cyclomatic Complexity metric has many advantages
 - Simple to understand
 - Easy to compute



$$C = A + 1$$

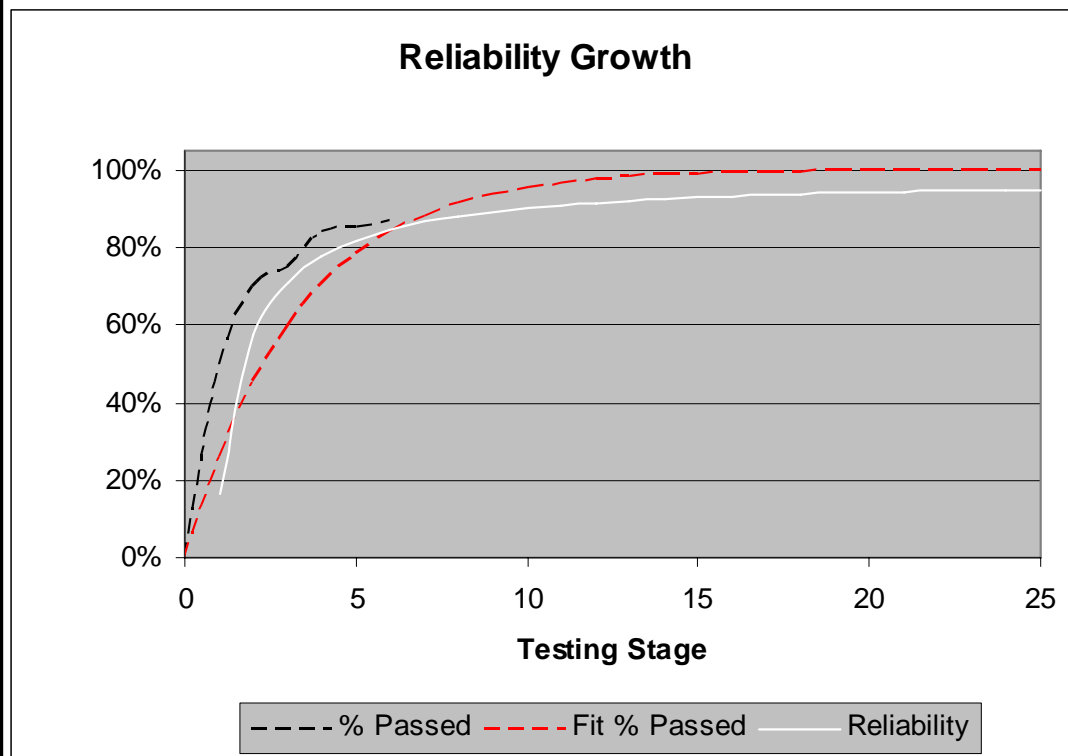
Knowing when to Quit – Reliability Growth

Test Stage	% Passed	Reliability
1	50%	16%
2	70%	57%
3	75%	71%
4	84%	78%
5	85%	82%
6	87%	85%
7		
8		
9		

- Computes relative reliability as a function of percentage of test cases passed
- Based on a well known statistical technique: Weibull analysis.
- Provides insight into the “repairability” of the system.
- Can be used in three modes:
 - Software reliability target
 - Test-sufficiency indicator
 - Development process capability indicator

Knowing when to Quit – Reliability Growth

Test Stage	% Passed	Reliability
1	50%	16%
2	70%	57%
3	75%	71%
4	84%	78%
5	85%	82%
6	87%	85%
7		
8		
9		



Summary

In today's high-pressure, fast-paced world, we must realign our thinking to test effectively

- Abandon the expectation of comprehensive testing
- De-emphasize functional testing
- Focus on the code as the source of information to drive test specification
- Invest in layered, up-front test planning
- Monitor the performance of the test project
- Prioritize testing with risk-based objective measures
- Recognize the point of diminishing returns

Discussion?