

Learnings from planning and executing performance testing for a large application

Abhijit Deshmukh

15 September 2006

Agenda

- Background – Product, Process, Environment, Constraints
- Overview of testing process
- Learnings – Planning Performance Testing
- Learnings – Executing Performance Testing
- Learnings – Managing timelines and people issues
- Conclusions
- Questions and Answers

Reference Product - Background

- Shrink Wrapped application
- Monitors IT Infrastructure – Performance and Availability Management for Networks, Systems, Databases, Middleware, Application Servers, etc
- Enables IT Service Management (ITSM)
- ITIL Aligned features and capabilities for Service Delivery
- Distributed data collection capabilities
- Classic 3 tier architecture & Web based user interface

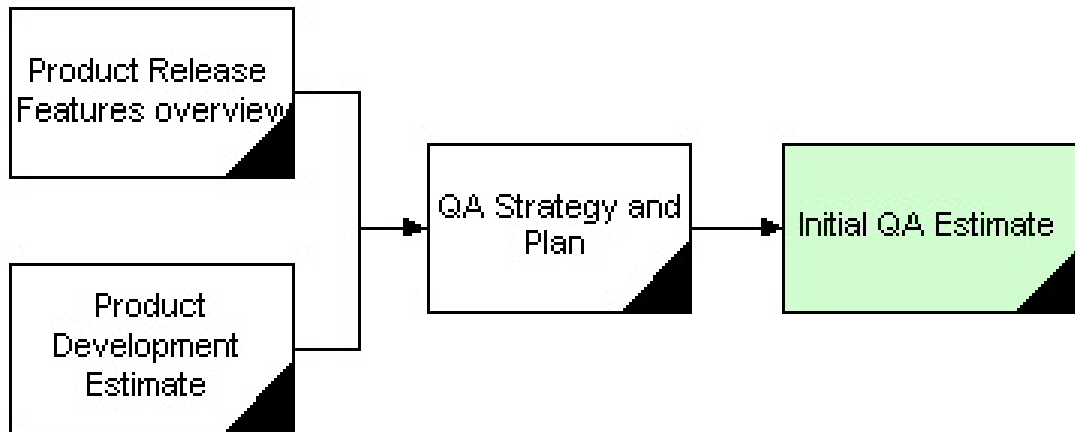
Testing Environment

- Network Elements – Routers, Switches, etc
- SNMP Simulators, Trap Generators, etc
- Databases – MS-SQL, Oracle, DB2, Sybase
- Systems – Linux, Windows, Solaris, AIX based servers
- Applications – MQ Series, WebSphere, WebLogic, etc
- Messaging – Lotus Notes, Microsoft Exchange
- Many More elements

Constraints

- Our constraints determine the way we plan and execute performance testing
- Constraints are normal - not unique to our environment
- QA is a shared service across multiple product lines,
- Aggressive deadlines
- Talent attrition
- Feature rich and complex product releases

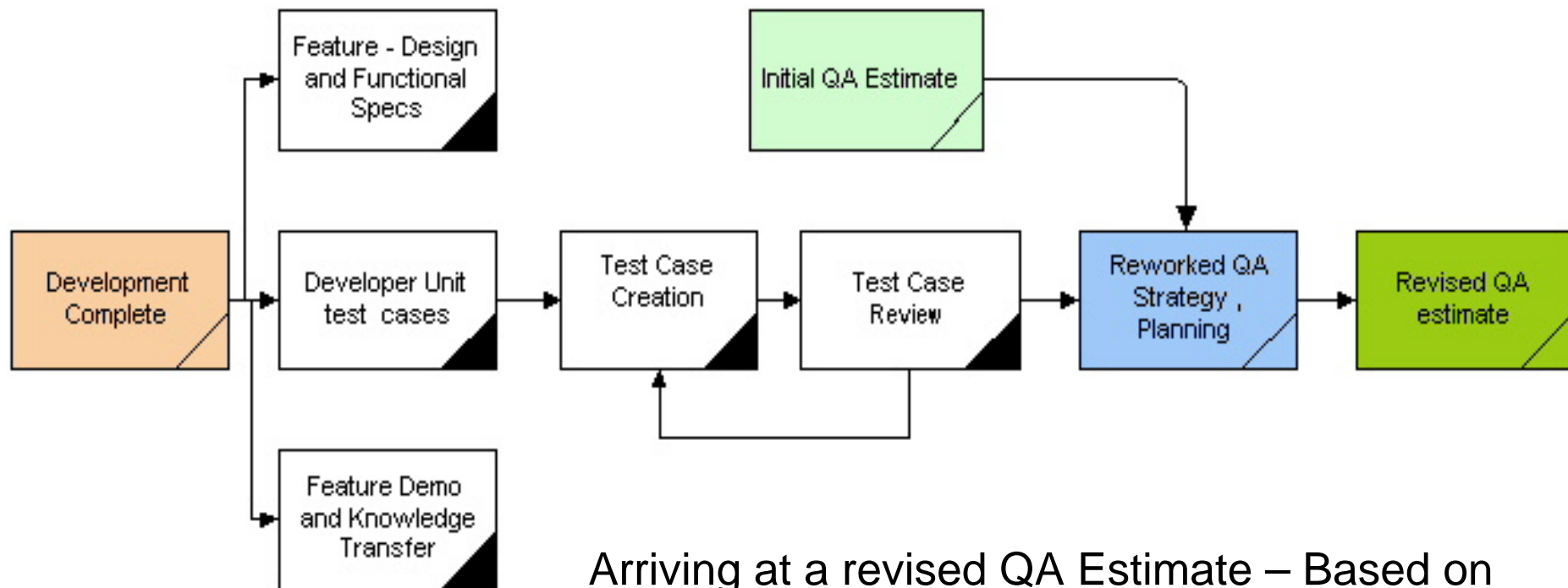
Brief overview of testing process



INITIAL ESTIMATE

- QA activities DO NOT start when development starts
- Dynamic environment – features/specs/design can change mid-way during development – this is normal
- Lots of R&D, PoCs
- Customer aligned delivery – open to expanding or contracting requirements – until the last moment

Ready for QA

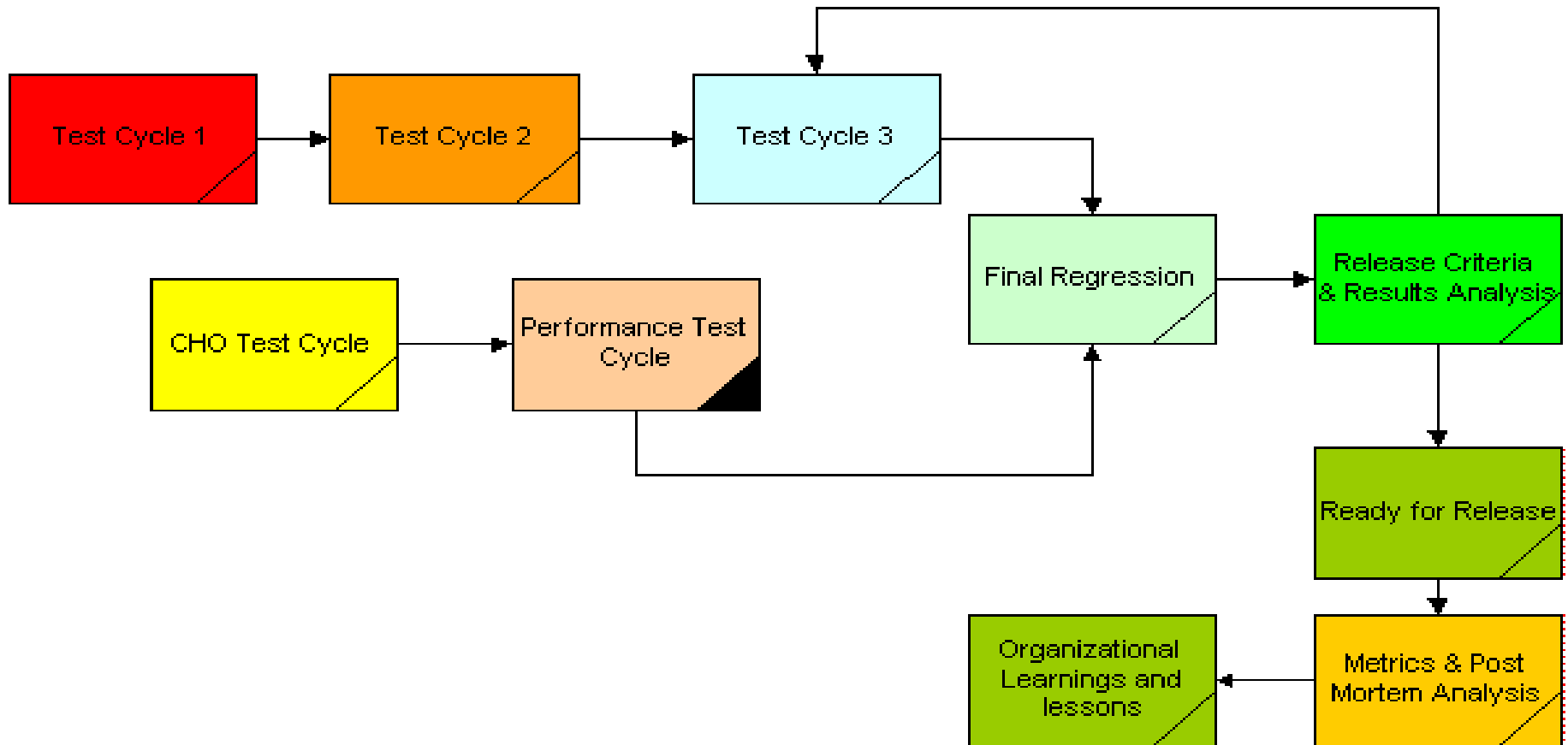


Arriving at a revised QA Estimate – Based on features, knowledge transfer, dev unit test cases, QA unit test cases and review, test strategy, customer/business deadlines, etc

Ready for QA - Reviews

- Peer and Dev Review of test cases
- Feature & E2E Champions in Dev & QA
- Test Cases are reviewed, qualified and re-written until coverage and depth found sufficient
- Intense and frequent knowledge transfer sessions later during the QA cycle as understanding of the feature and product increases

Test Process - Execution



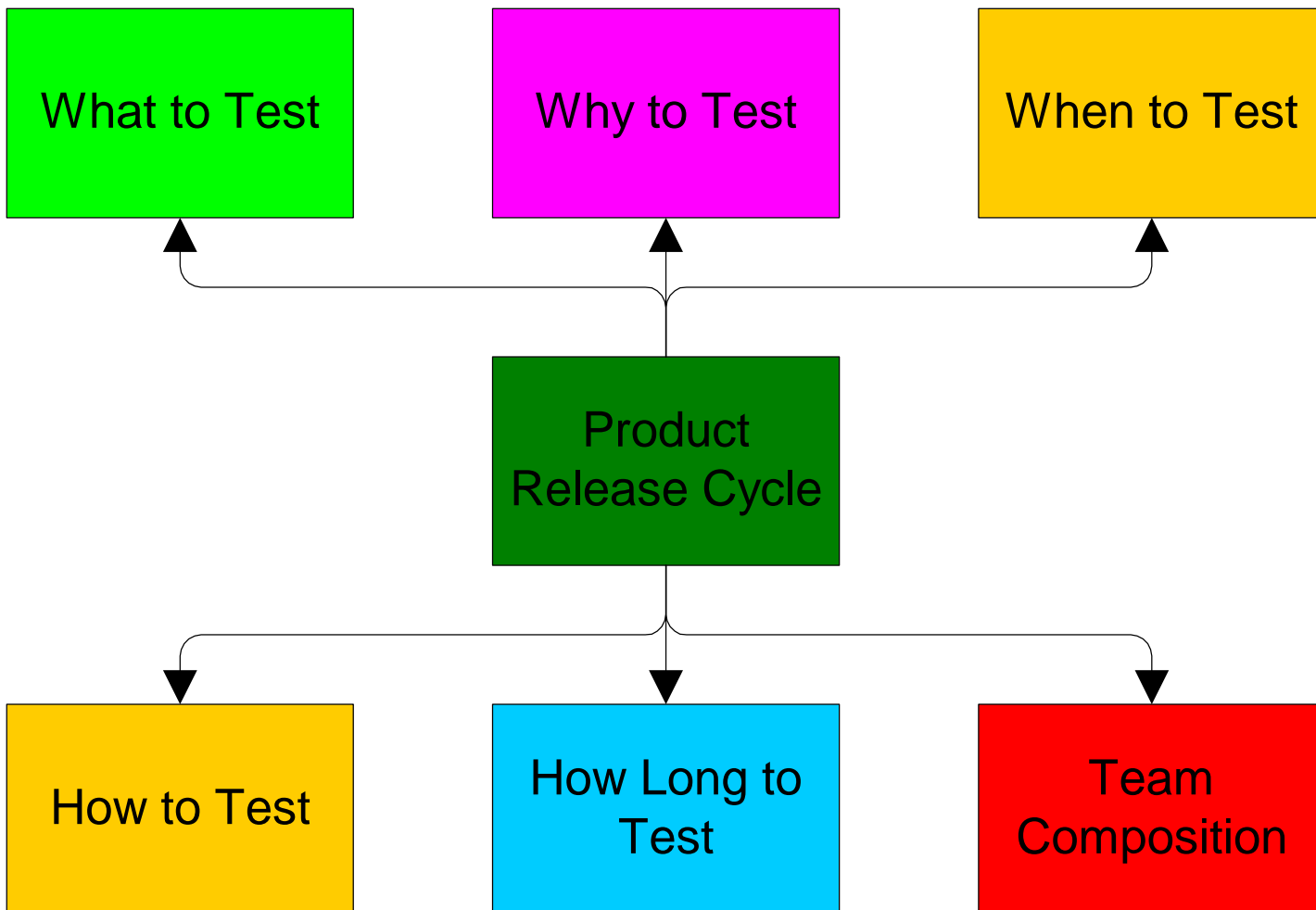
CHO - Continuous Hours of Operation

- Ensures Stability Testing
- Runs for un-attended many weeks, months
- Positive Load Stress – all features enabled – load generation is automated
- Used to prove end to end stability of product
- Runs on multiple platforms
- Serves as mandatory filter before Performance Testing can start

Performance Testing

- Standard & Specific Performance Requirements are known
- Runs for multiple weeks
- May result in bugs that can lead to new bug fix builds.
- Bugs may point to design flaws.
- Which bug to fix and which bug to defer is based on the seriousness of the bug, the probability of the bug arising in a specific scenario, business commitments, delivery timelines, etc.
- Followed by a final regression test cycle
- Followed by a Pre-Release QA and finally Product release.

Planning for Performance Testing



What to test - learnings

- Identify features and scenarios
- End to End scenarios mandatory
- Scenarios need to be valid – from customer/competition perspective
- Consensus from Customer facing team (support/implementation/pre-sales), & Dev is important
- Identify top N scenarios along with data flow
- Data flow – Input and Output – needed for generating work/test load at a later stage
- Test Scenarios for Performance needs to be realistic & real world - Prevent Wild Goose Chases

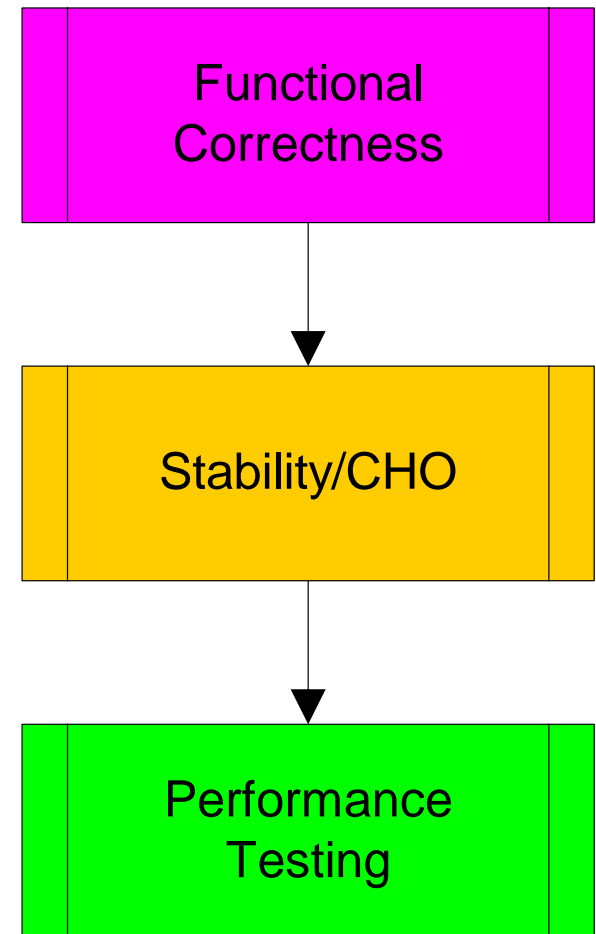
Why should be “X” tested for performance

- X could be a scenario or X could be a feature
- Answer should confirm validity of performance testing requirement
- Typical answers should be business or customer aligned,
- A “slow” product can still sell and make lot of money

- Pure Engineering centric testing can often cause loss of productive time as the gains from finding performance gaps may have no business relevance.

When to start testing for performance

- Ensure Correctness - ALL functionality (including end to end) is correct
- Ensure Stability – Application can run for many days with all cylinders firing in an unattended mode - without any exceptions, problems, resource leaks, etc
- After Correctness and Stability is proven, Root Cause Analysis for performance related bug/issue is simple and straightforward



How should “X” be tested for performance

- Real world inputs, workloads required
- Automation tools will enhance productivity
- User Interaction Recording tools – can help only to a minor extent
- Quick and tangible ROI on investments made in specialized commercial, third party tools for workload generation
- Encourage in-house automation tools – as these are easy to build and are much more relevant to testing
- Test is as good as the scenario being tested
- Test is as good as the amount and quality of workload being generated

How to ... learnings

- Adequate, appropriate and realistic hardware
- Avoid un-realistic expectation (either too low or too high) of what a customer is likely to spend on hardware for running the application.
- The test hardware must be validated by business facing (Support, Implementation, Pre-Sales) teams.
- The hardware mismatch can tend to skewed results and fruitless efforts.
- Bug fixes or patches must be applied very judiciously
- Consolidated performance specific patch – one that has been already tested for functionality and stability.

How long to test for performance?

- Until the objectives are met or until no more time is left
- In the beginning – the answer is ideal
- In the middle – needs to be answered periodically – end of each day
- If we test functionally correct and stable features, the performance testing cycle is better controlled.
- Most of the serious performance gaps can only be fixed by a design or architecture overhaul.
- When such gaps are found during the test cycle, the project plan needs to change to based on the decisions taken.

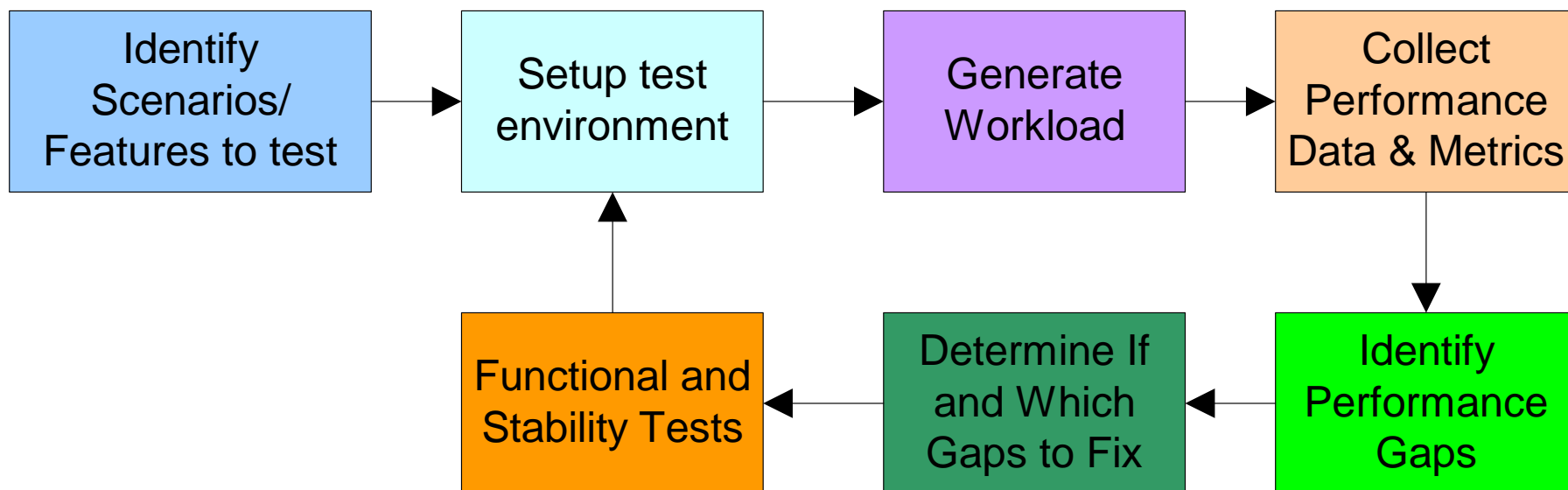
Integrating in a product release

- Must be a planned, pre-scoped activity
- Bug fixing for performance related issues is not trivial and takes time and design insight.
- Majority of Bug Fixes tend to change either the design or the database schema of parts of the application. Warrants risk assessment
- Need 1 or 2 more test cycles (not related to performance) where the effects of performance related bug fixes can be assessed independently.
- The test setup for performance testing should be separate and independent from the setup for functional testing.

Team Composition

- Multi-disciplinary with a mix of QA engineers and developers who understand the application end to end.
- Test Champion needs to be identified
- Domain experts are essential
- QA Engineers who know the “history of the product” must be included
- Developers do not participate in test execution – but provide advisory services in terms of test setup and execution
- Developer participation and alignment towards testing prevents conflicts and interpersonal while fostering better inter-disciplinary team work
- Senior business aligned, technical manager needed to oversee effort and determine/resolve priorities, conflicts, trade-offs, etc.

Executing Performance Testing



Performance test environment

- Realistic for the workload being tested
- Needs to be validated by Dev, Support, etc
- Phased approach in making the test environment available to team doing performance testing
- Start testing in a gradual manner and collect performance data when the system is running with 33% load, 66% load and 100% load.
- This approach provides invaluable trend analysis data that is very useful to find root causes of lack of performance.

Load Generating tools

- Buy and/or build automated tools
- Tools simplify the task of generating data that mimics a given scenario.
- Tools should have excellent throttling control that allows us to modulate the rate of inputs for the test bed.
- Enables our engineers to concentrate on profiling system dynamics including resource utilization, correctness of output data, etc.

Collecting performance data and metrics

- Collection of performance data and metrics is partly automated
- CPU Utilization, Memory Utilization, Thread count, etc is captured automatically and correlated to system load, application response times, etc
- Application exceptions and errors – that may occur as an imperfect application tries to process a large amount of data in a very short time.
- Suggested Metrics - Number of functional errors, Index for application usability, Index for application responsiveness, Number of data integrity errors, Index for Resource Utilization.
- Understand trend between amount of input data, amount of correct output data and resource utilization

Finding performance gaps

- Performance requirements must include upper and lower control levels for resource utilization for a given scenario and quantum of workload
- If actual resource utilization does not fall between UCL and LCL then a gap has been identified
- The question that needs to be answered is - If more resources have to be added for a given workload – will this add significant cost ?
- Sometimes adding more hardware does not solve the problem – because of structural design flaws

Quick gains

- Introduce parallelism/multi-threading
- Reduce Network I/O and Disk I/O where possible.
- Favor database connection pooling, object pooling, object caching, etc.
- Reduce expensive object creation
- Faster database operations – favor stored procedures, indexing, batch statements, server side cursors, etc

Fixing Gaps

- Identify Root Causes for each gap
- Analyze RCA and identify “low hanging fruits” – quick successes
- Design/Execute PoCs based on initial assessment
- Seek approval from Change Control Board or senior management
- Assess risk of change – will change result in more bugs ?
- What is the scope of change ?
- Does it mean that functional and stability tests have to be done again ?
- Pairing for design and coding – helps in reducing risk of bugs introduced because of change
- Performance related fixes need to be tested first for correctness and stability and then for performance.

Areas of conflict

- Between Dev and QA
- Dev is typically upset that issues are being identified so late in the release cycle
- Conflicts can be eliminated
- Development team must review and generally agree with the performance test strategy, the test setup and the test methodology at the beginning of the test.
- Daily reviews between Dev and QA teams
- Recognize “testing fatigue”. Appreciate Encourage & Praise often.

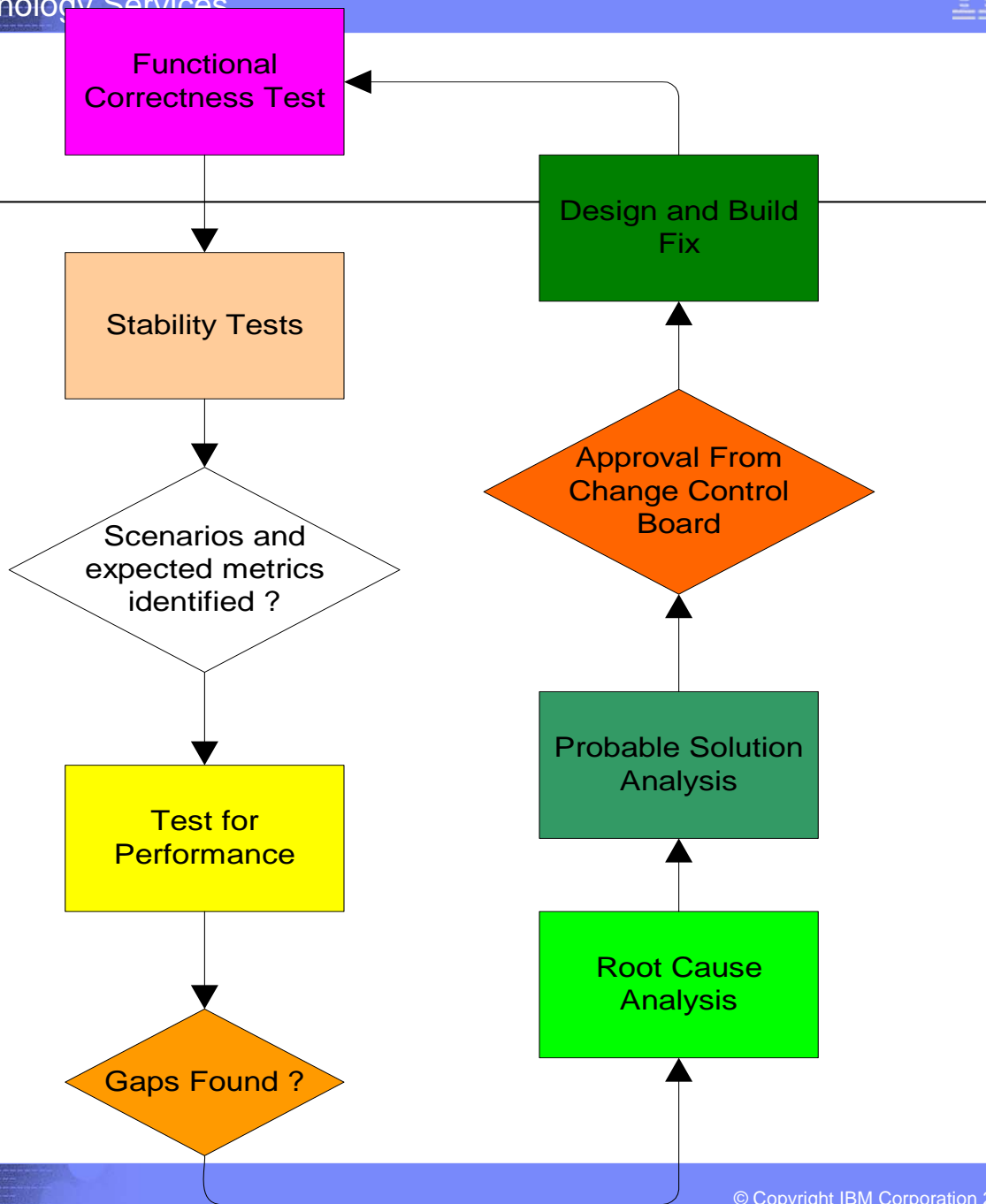
Typical Tradeoffs

- Depth of performance testing and delivery timelines
- Correctness and stability tests should be re-run again as a pre-requisite for bug fixes from performance testing
- To regress or not to regress ?? → Final Regression is needed – and is unavoidable

In Summary

- Start after Correctness and Stability has been verified
- Automated workload generation tools must be available
- Scenarios, features must be identified from a business perspective
- Expected Performance metrics (UCL and LCL) must be known and agreed
- Buy in from Dev, Support and Implementation is mandatory
- Be careful in deciding which gaps to fix/address
- Ensure that performance bugs are first tested for correctness and stability
- Staff for success – Mixed team, Senior Manager for decision making

Typical Flow



Elements of a Good Performance Test Cycle

Validated
Scenarios

Automated Work
Load Generation

Performance
Metrics Collection

Business &
Customer
Alignment

Change Control
Board

Realistic
Hardware

Capable Team

Risk Analysis

Conflict
Management