Developing an ***Effective*** *Performance Testing Strategy*

Ross Collard

Collard & Company

Overview

- This presentation addresses the tester's question: “What is the best way for me to test a system’s performance?”
- The answer is highly context-dependent.
- We will review a lightweight methodology for developing your performance testing approach, which applies to a broad range of contexts.

Overview

- Claim: performance testing is undegoing a revolution, a paradigm shift.
  – There is accumulating evidence that there are better ways to test performance and that these methods may be both repeatable and ready for use on a large cross-section of projects.

Overview

– Leading-edge projects report successes in areas as diverse as:
  - test project justification,
  - client and management awareness and support,
  - design reviews for performance, design for testability,
  - early testing: getting off to a quick start,
  - agile and exploratory testing,
  - automation frameworks and tools, test load design and development, continued….

Overview

– Leading-edge projects report successes in areas as diverse as:
  - move to open source tools,
  - focusing on what to observe and monitor,
  - predictions of live operational behavior,
  - test data interpretation,
  - methods for modeling, and
  - methods for scalability.

Overview

- Counter-claim: what revolution?
  - It is just *evolution*.
  - And – some think – it is *slow* evolution.
  - Some people were practicing all or most of these “advances” 20 years ago, especially in telecom.

Overview

- Claim: skill sets do not match needs.
  - A fool with a tool is *still a fool*!
  - Innumeracy (the equivalent of illiteracy with basic math) is widespread.
    - What is the probability that the person next to you has the same birthday?
    - Is the answer approximately 3%, or 0.3% or 0.03% or 0.003%, or none of these?

Overview

- ***Claim***: most performance testing projects are under-planned or poorly planned.
  - Reactive: “Fire, aim, ready.”
  - Tool-centric, not client- or problem-centric.
  - Harried by deadline pressures, attitudes, etc.
- ***Solution***: this methodology!
  - Necessarily generic, but that’s the beauty.
  - Pragmatic and agile, not doctrinaire.
  - First step is to tailor it to the situation at hand.
  - Steps and sequence may change with the context.

Generic Templates

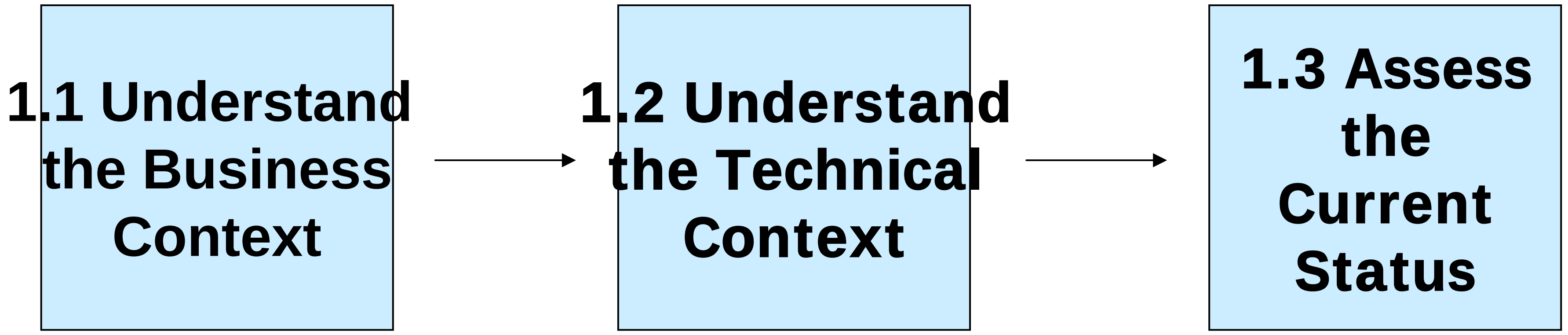
- The good:
- The bad:

Major Steps to Develop the Performance Test Strategy

1. Assess the Situation.
2. Plan the Performance Test.
   – Draft Strategy.
3. Develop the Test Ware.
4. Finalize the Strategy.
   – Ready-to-Go Strategy.
5. Start Testing.
   – Tuned Strategy.

1. Assess the Situation

A. Understand the Situation and Context


1.1 Understand the Business Context
1.2 Understand the Technical Context
1.3 Assess the Current Status

1.1: Understand the Business Context

- Where do we begin?
  - Who's who in the organization.
  - What business they are in.
  - *How they make their money (CSF).*
  - Business goals.
  - System performance goals.
    - Current sources of pain.
    - Tradeoffs (performance vs. security, usability, etc.

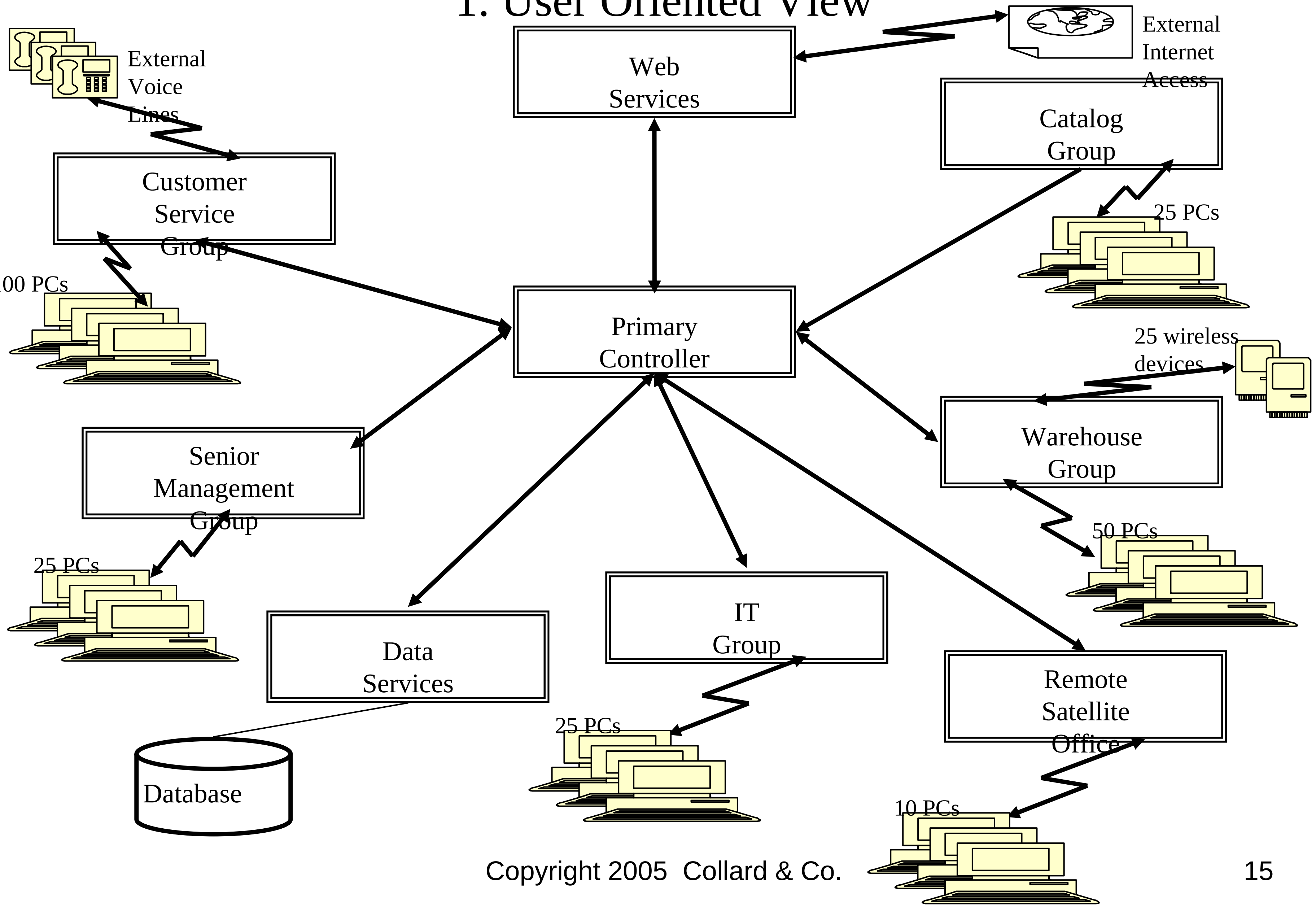
1.1: Understand the Business Context

- Type of project
  - New development vs. modification.
  - Agile vs. waterfall.
  - Existing personalities.
- Expectations
  - Schedule (yesterday?); budget (cheap?).
  - Fantasy vs. reality: levels of sophistication and maturity.
  - Prior experiences with 1) performance, 2) testing, 3) performance testing.

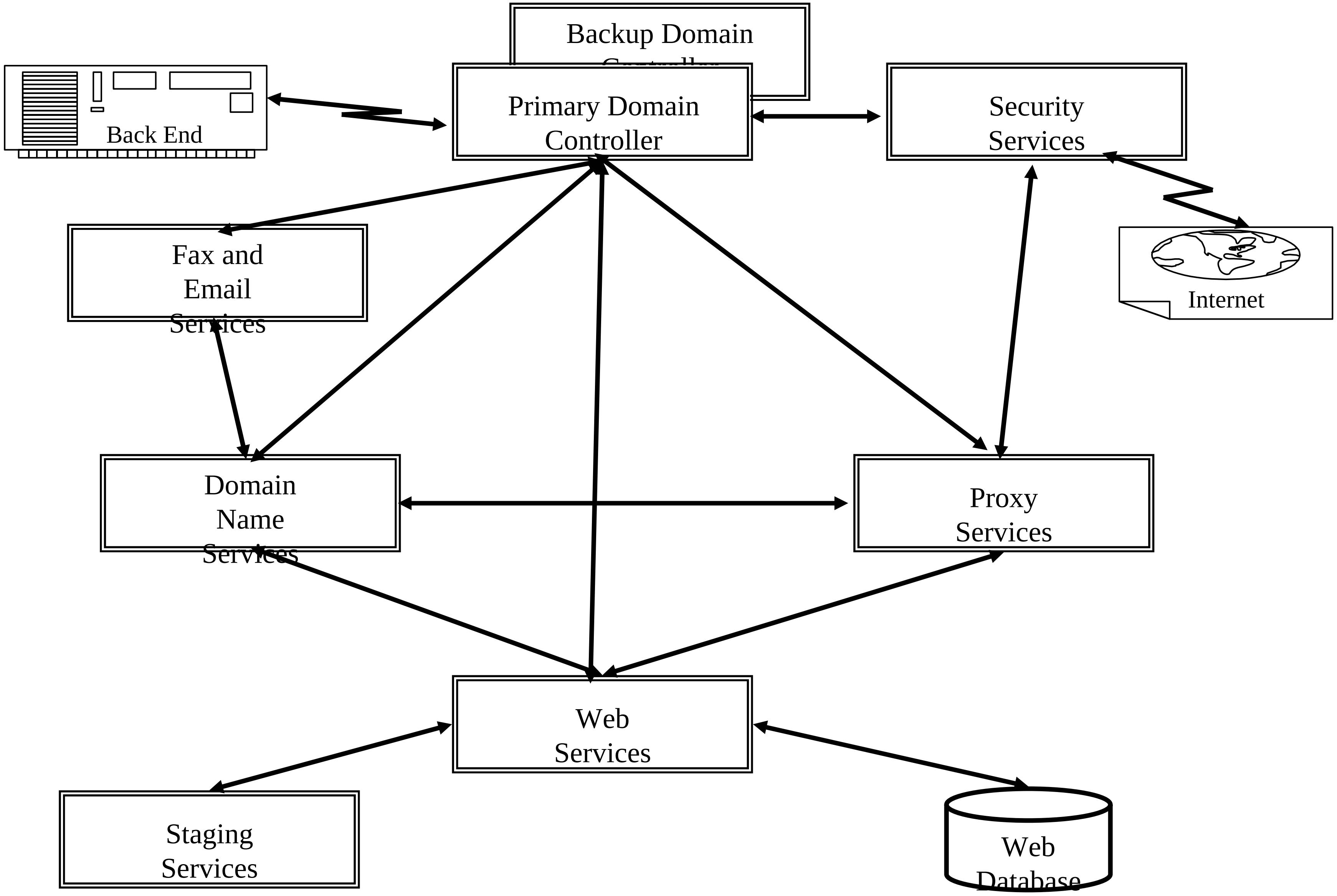
1.2: Understand the Technical Context

- System architecture and topology.
  - Servers / processors, networks, databases, web sites, applications, support software (e.g., OS), protocols & standards.
  - Accuracy, currency and detail of available information.
  - *Need to model:*
    - Logical vs. physical architectures.
    - Whole infrastructure vs. changes.

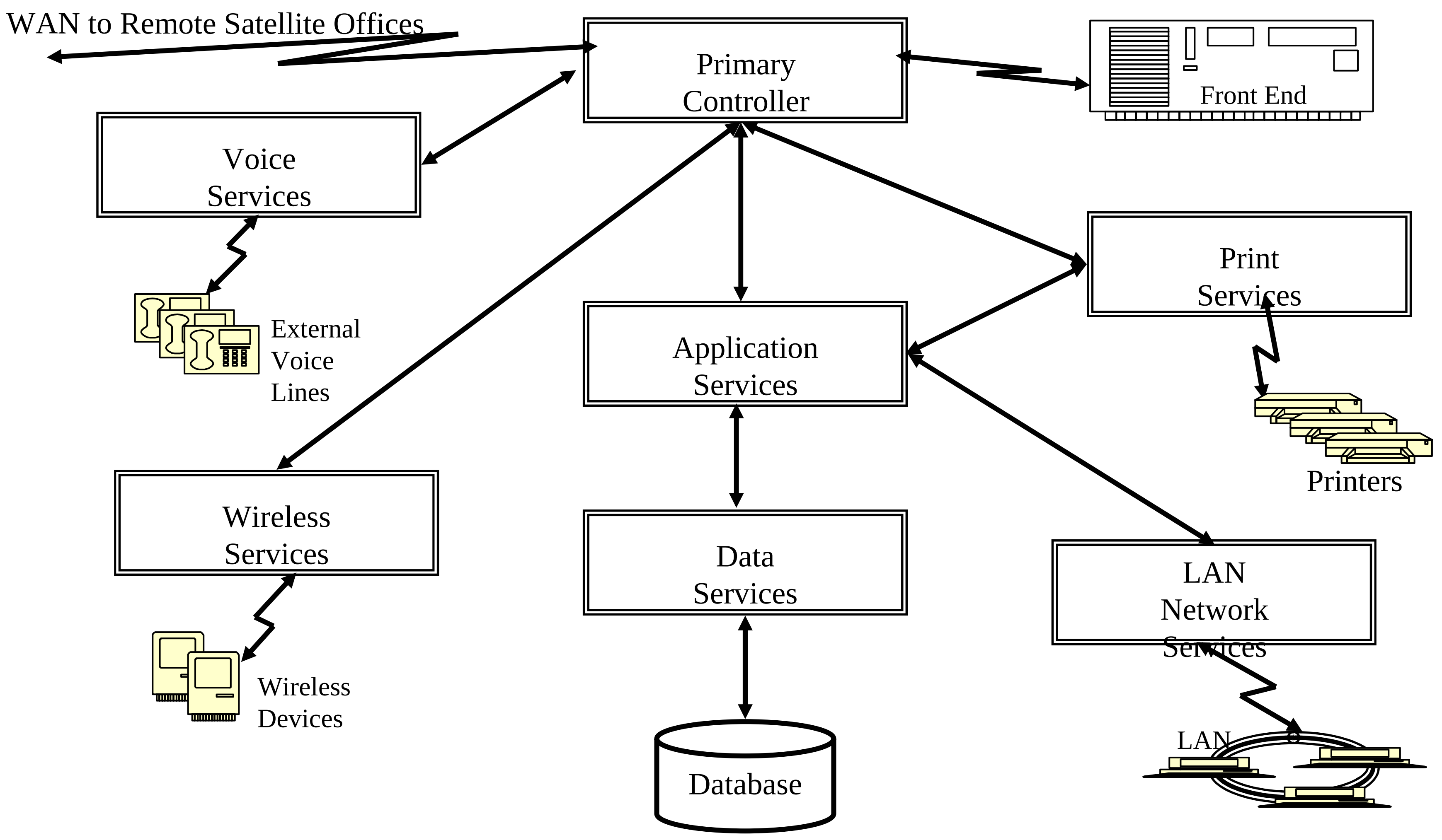
System Architecture
1. User Oriented View



System Architecture
2A. Front End Functional View



System Architecture
2B. Back End Functional View

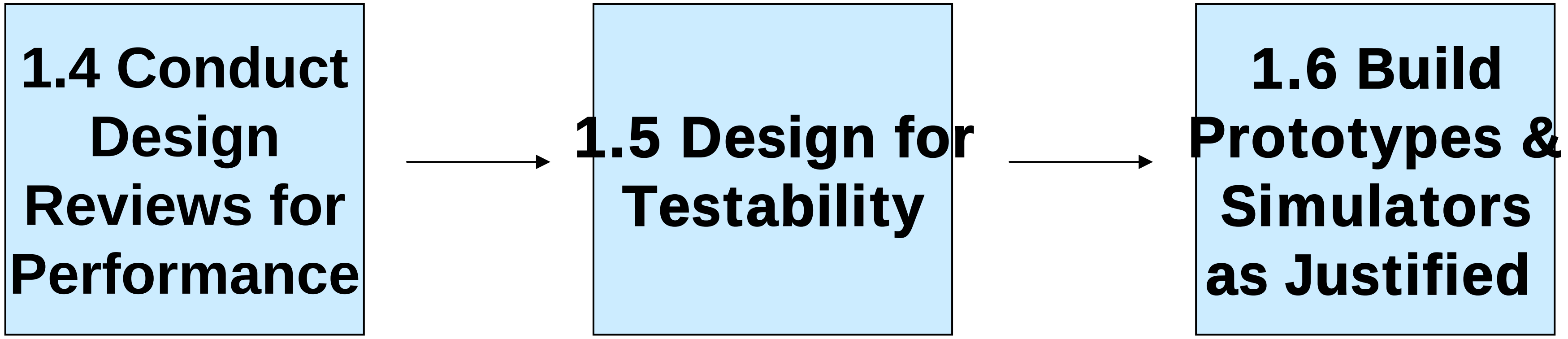


1.3: Assess the Current Status

- Operational track record, compliance with service level agreements (SLAs).
- Current test ware: framework, equipment, tools, test script libraries, staff, skills.
- Other projects and activities to coordinate.
- *The most important data is unwritten:*
  - Politics, personalities and relationships.
  - Culture and style.
  - *Effectiveness, sophistication, image.*

1. Assess the Situation

B. Avoid Surprises


1.4 Conduct Design Reviews for Performance
1.5 Design for Testability
1.6 Build Prototypes & Simulators as Justified

1.4 Conduct Design Reviews for Performance

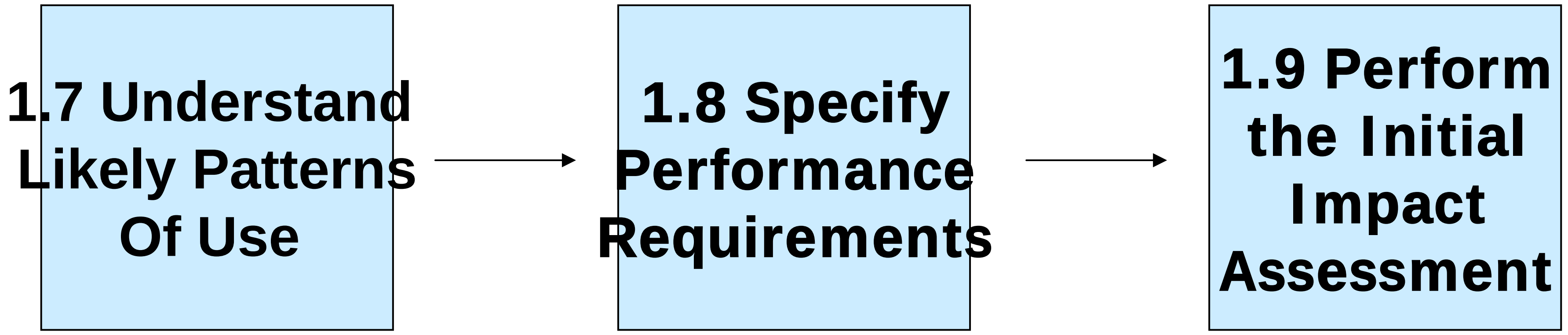
- Generally the testers are not involved in early design reviews. *A pity.*
- Many designers and architects are earnest and hard-working – but this does not mean they know what they are doing.
- Most technical advice comes from vendors -- not the most unbiased source.
- Checklists can frame and guide reviews.

1.5 Design for Testability

- *Much talked about, rarely done.*
- Access to hidden system internals, instrumentation.
  - Hooks and probes desired by testers.
- Fleeting data, transient states.
  - Data to be captured in logs.
  - Environment status as well as system status.
- Test ware compatibility with development and live environments.
- Maintainability of test scripts as system evolves.
- Overhead: Heisenberg's law.
  - Probes in versus probes out: difference in system behavior?

1. Assess the Situation

C. Justify the Need to Test


1.7 Understand Likely Patterns Of Use
1.8 Specify Performance Requirements
1.9 Perform the Initial Impact Assessment

1.7 Understand Likely Patterns Of Use

- Identify the main users.
- Understand their main types of work.
- *Ascertain their success factors and satisfaction levels.*
- Become acquainted with the mainstream work flows.
  - Use cases, user-oriented flow models, process engineering charts.
  - Interviews with users and observations.

1.8: Specify the Performance Requirements

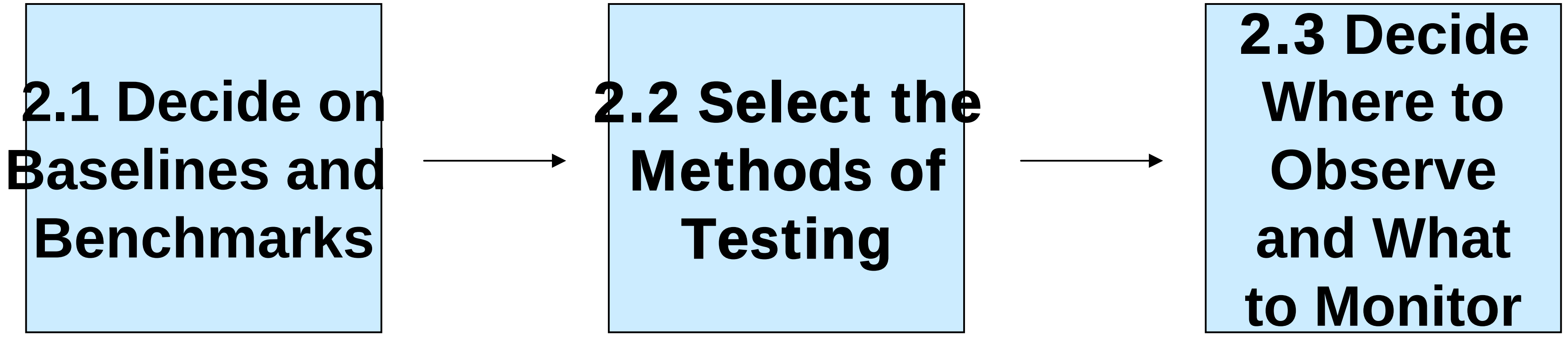
- Ideally, business goals drive performance goals which drive the test objectives.
- *Most test objectives need re-writing!*
- Desirable characteristics: objective, credible, observable or measurable, relevant, meaningful, actionable, agreed-on, focused, specific.
- A few carefully selected points on the envelope.
  - Scope?
  - Black box or white box (invasive)?
  - Bellwether or corner case (extreme point)?

1.9: Perform the Initial Impact Assessment

- *Early, quick and crude evaluation* to guide a “go / no” decision on performance testing.
- Utilizes a QRW (quick review worksheet).
  - Major impacts: resource intensive or shared resources, timing critical, heavy usage, SLA sensitive, suspected bottleneck, high visibility, high liability.

2. Plan the Test

A. Decide How to Test (Part 1)


2.1 Decide on Baselines and Benchmarks
2.2 Select the Methods of Testing
2.3 Decide Where to Observe and What to Monitor

2.1: Decide on Baselines & Benchmarks

- Baseline: before-and-after comparison of performance and robustness.
  - From existing system to replacement system.
  - Or from system version to version.
- Benchmark: industry-standard test work load.
  - For side-by-side comparisons of competing products, e.g., DBMS.

2.2: Select the Methods of Testing

- Rank the usefulness and ease of application for this project of these test methods:
  - *1.0 Testing driven by what we want to measure.*
    - 1.1 Response time measurement.
    - 1.2 Throughput measurement.
    - 1.3 Availability measurement.
    - 1.4 Measurement of resource utilization.
    - 1.5 Error rates.
  - *2.0 Testing based on the source or type of the load.*
    - 2.1 Usage-based testing.
    - 2.2 Standard benchmark testing.

Methods of Testing (Continued)

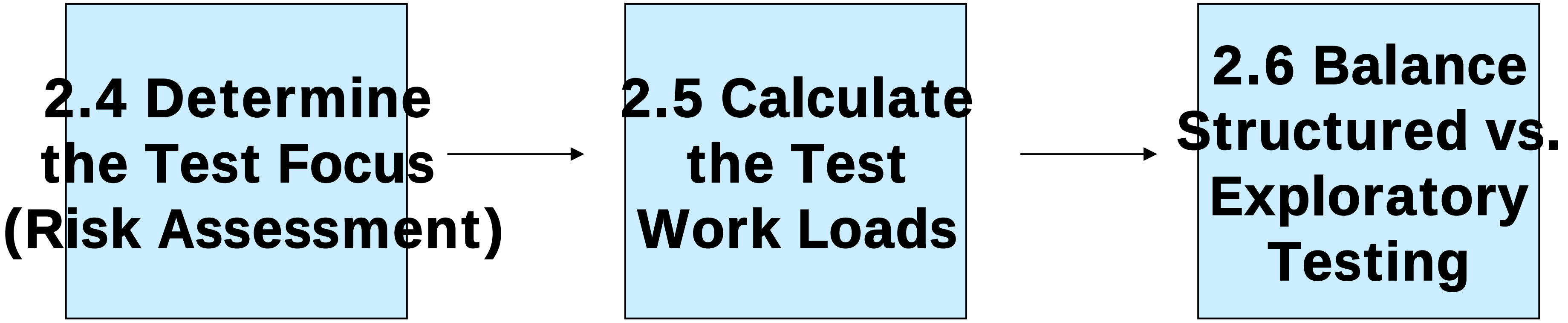
- *3.0 Testing to stress the system.*
  - 3.1 Hot spot testing.
  - 3.2 Spike and bounce testing.
  - 3.3 Breakpoint testing.
  - 3.4 Degraded mode of operation testing.
- *4.0 Concurrency testing.*
  - 4.1 Interaction / interference testing.
- *5.0 Risk-based testing.*
  - 5.1 Risk assessment.
  - 5.2 Bad day testing.

2.3: Decide Where to Observe and What To Monitor

- Decisions are driven by the performance requirements and test objectives:
  - At what points in the architecture do we monitor the system’s performance during testing?
  - Have access points or probes have been built into the system, and if so where?
  - Is the access adequate? What observability and controllability are needed?
  - What data to collect at each monitoring point?
  - Is this data available (e.g., in server logs)?

2. Plan the Test

A. Decide How to Test (Part 2)


2.4 Determine the Test Focus (Risk Assessment)
2.5 Calculate the Test Work Loads
2.6 Balance Structured vs. Exploratory Testing

2.4: Determine the Test Focus (Risk Assessment)

- We cannot test and measure everything.
- *Where do we test intensely?*
- *Where do we choose to skim or investigate lightly?*
- Deeper analysis than the initial impact assessment.
- Based on identification and assessment of risks and vulnerabilities – *checklists.*

2.5: Calculate the Test Work Loads

- Identify the test scenarios and the mixes of demands (test work loads) needed.
- Visualize and map traffic flow patterns.
- Obtain operational profiles and user demographics.
- Consider scalability.
- Document volumetric assumptions: # users, # concurrent sessions, fluctuations.

Test Work Loads (Continued)

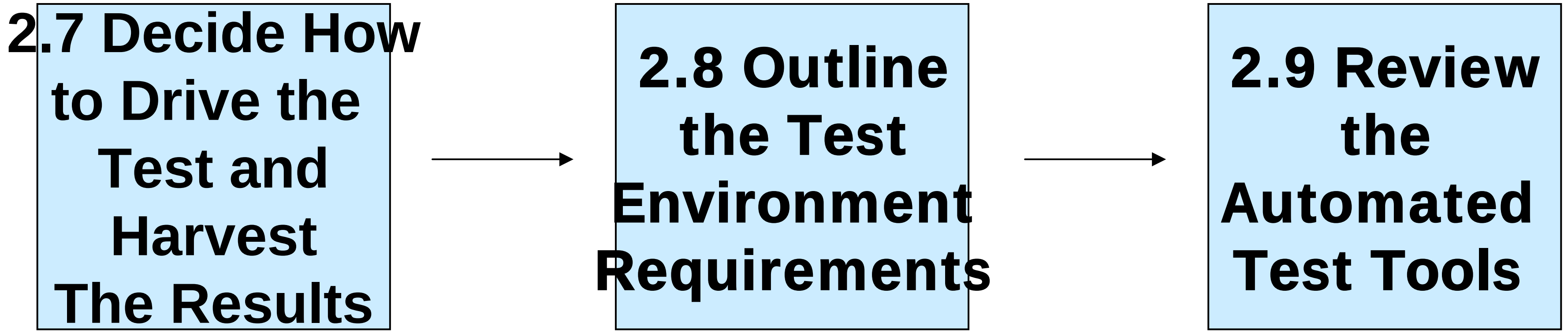
- Determine sample sizes, test durations.
- Factor in background noise.
- *Determine required accuracy & precision.* (Lorenz’s MIT data, chaos theory.)
- Adjust for differences between test lab and live environment.
- Use dimensional analysis, safety factors.
- Calibrate, use as internal benchmark?

2.6 Balance Structured vs. Exploratory Testing

- Purpose: find the right mix of exploratory and structured testing for this project.
  - Commonly, many unknowns complicate performance testing projects.
  - *The more the unknowns, uncertainties and volatility, the less structured and pre-planned the testing can be.*
  - Checklist approach gives ballpark ratio of exploratory and structured testing.

2. Plan the Test

A. Decide How to Test (Part 3)


2.7 Decide How to Drive the Test and Harvest The Results
2.8 Outline the Test Environment Requirements
2.9 Review the Automated Test Tools

2.8: Outline the Test Environment Requirements

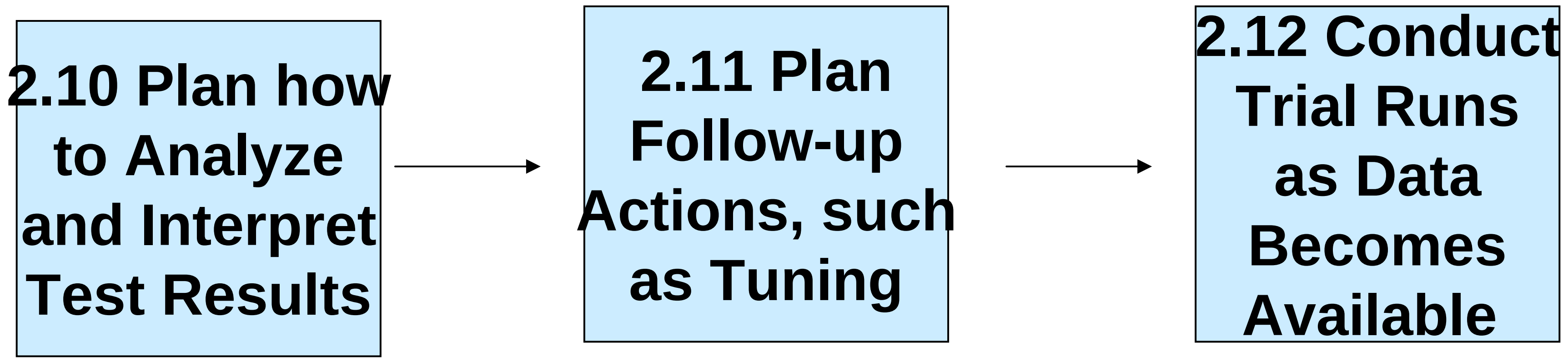
- Define the test lab requirements.
- Decide if it is a dedicated sustainable on-going resource, or temporary and shared.
- *Outline the test automation framework.*
- Decide whether – and how -- to re-use existing test ware.
- Determine budget, timeframe, feasibility and support demands.

2.9: Review the Test Tools

- Review track records with existing tools, if any.
- Decide if a new tool search is needed.
  - Use tool assessment and selection checklists.
  - Make-versus-buy decisions.
  - Fit with the test automation framework; compatibility with functional test tools.
  - Training, test script / data conversion and support needs.

2. Plan the Test

B. Plan the Test Data Analysis


2.10 Plan how to Analyze and Interpret Test Results
2.11 Plan Follow-up Actions, such as Tuning
2.12 Conduct Trial Runs as Data Becomes Available

2.10: Plan How to Analyze and Interpret the Test Results

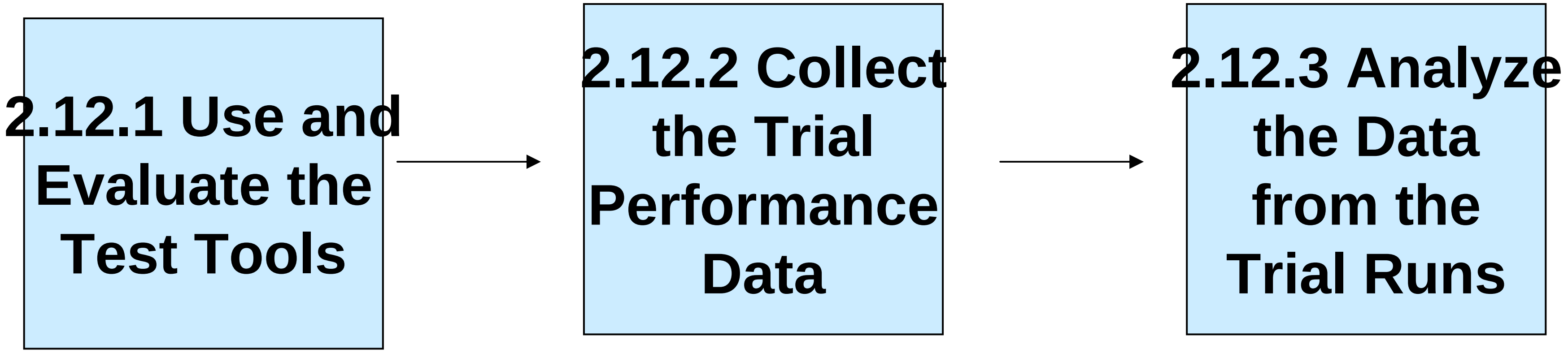
- Goal-driven: goals >>> metrics >>> data.
  - Build evidence to support or deny hypotheses.
- Possibilities of false test data, or misleading or false interpretations of data.
- How do we extract an ounce of wisdom from 1,000 tons of raw data?
  - Visualization.
  - Search for patterns, correlations and relationships.
  - Repositories of patterns and templates.

2.11 Plan Follow-up Actions

- Possible actions based on test findings:
  - More testing:
    - Provide extra evidence for confirmation / back-up.
    - Isolate and pinpoint cause of behavior.
    - Pursue new hypotheses created by earlier test findings.
  - Tuning: an “elaborate dance”: the tester as orchestra conductor.
  - Bottleneck resolution.
  - Re-design, infrastructure re-organization.
  - Resource capacity upgrades.

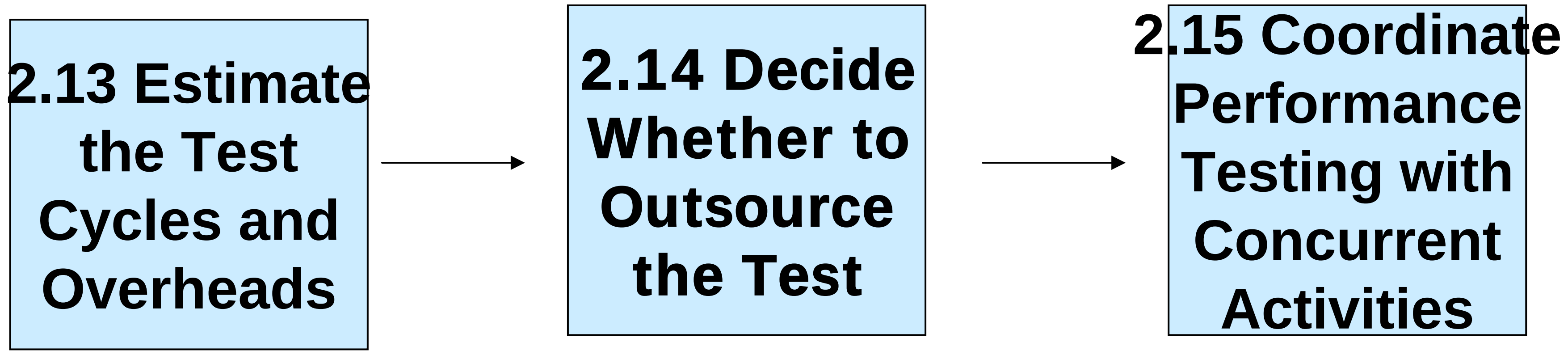
2.B Plan the Test Data Analysis

2.12. Conduct Trial Runs


2.12.1 Use and Evaluate the Test Tools
2.12.2 Collect the Trial Performance Data
2.12.3 Analyze the Data from the Trial Runs

2. Plan the Test

C. Plan the Use of Test Resources


2.13 Estimate the Test Cycles and Overheads
2.14 Decide Whether to Outsource the Test
2.15 Coordinate Performance Testing with Concurrent Activities

2.13.1: Estimate the Number of Test Cycles

- *Performance testing is usually highly iterative*, with rapid re-tests as bottlenecks are found and resolved, and as the system is debugged and tuned.
- An early if crude estimate of the number of test cycles is important – if we assume 3 cycles and the project actually requires 15, our deadline is imperiled.

2.13.2: Estimate the Test Overheads

- So-called overheads can easily consume 2/3 or more of the test effort. Visible part is the tip of the iceberg.
- Several estimating methods are available.
- Consider, for a 10-minute test run:
  - How much preparation effort is needed?
  - How much follow-up effort is needed?
  - Overhead ratios exceed 8-to-1 in 36% of companies surveyed.

Test Overheads (Continued)

Ratio of Total Test Effort to Hands-on Test Execution	Percentage of Testers
– 2-to-1, or less	0%
– 2-to-1 to 4-to-1	6%
– 4-to-1 to 6-to-1	21%
– 6-to-1 to 8-to-1	37%
– 8-to-1 to 12-to-1	16%
– 12-to-1 to 16-to-1	10%
– 16-to-1 to 20-to-1	5%
– 20-to-1 or more	5%

– Based on survey of 111 performance testers.

Test Estimation Techniques

Prior Experience Methods
  - Prior Test History
  - Dominant Factor in Test Duration
  - Dominant Factor in Test Resource Consumption
- Early Ball-Park Methods
  - Wishful Thinking and SWAG
  - Ratio of Testing to Development
  - Top-Down or Global Estimating
  - Rules of Thumb, Formulae and Models
  - Software Estimating Tools
- Detailed Analysis Methods
  - Bottom-Up or Micro-estimating
  - Templates and Worksheets

Test Estimation Techniques

- Iterative Feedback Methods
  - The Trial Run
  - Re-Estimating by Phase – Creeping Commitment
  - Estimating Testing within Iterative Development
- Constraint-Driven Methods
  - Mandated Deadlines
  - Estimating Based on Time and Resource Constraints
- Negotiation Methods
  - More Psychology than Computation

2.14: Decide Whether to Out-Source

Advantages

- Easier to out-source than feature testing, vendors need little domain expertise (maybe).
- Lower initial investment in tools and equipment.
- Lower project lead time (not guaranteed).
- Lower risk -- with the right vendor’s in-depth expertise.
- Another advocate of load testing (the vendor).
- Better scalability, because the vendor’s test lab has oodles of equipment and virtual user licenses (maybe).
- More objectivity and credibility of the test results (a bad sign, if outsiders have more credibility than employees).
- For bad news (“it won’t work”), vendor is the messenger.

Out-Sourcing (Continued)

Disadvantages

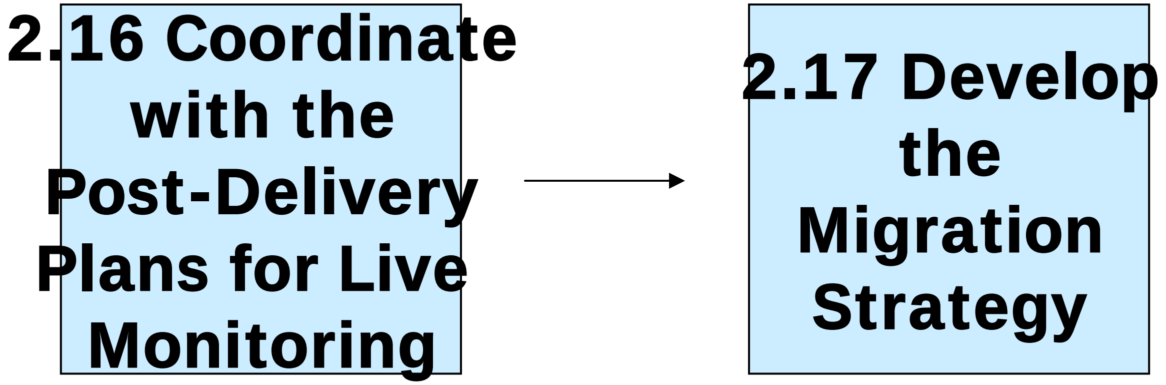
- Accumulated costs may be much higher.
- Do not get to build and establish expertise in-house.
- Lack of internal expertise means the client cannot competently question / manage the vendor’s work.
- Outsiders do not understand system and infrastructure as well as insiders.
- Possibility of us-versus-them conflicts.
- Potential re-use of internal test ware is low.
- Potential issues of security and confidentiality.
- If the system migrates and is re-tuned in the live environment after vendor testing, findings can be invalid.

2.15: Coordinate with other Concurrent Activities

- Performance testing is not done in a vacuum.
- Interdependencies occur with functional testing, system development and maintenance, software configuration management, system administration, and capacity planning.
- Performance testers tend to need more support than functional testers do from technical specialists (e.g., DBAs, network engineers).

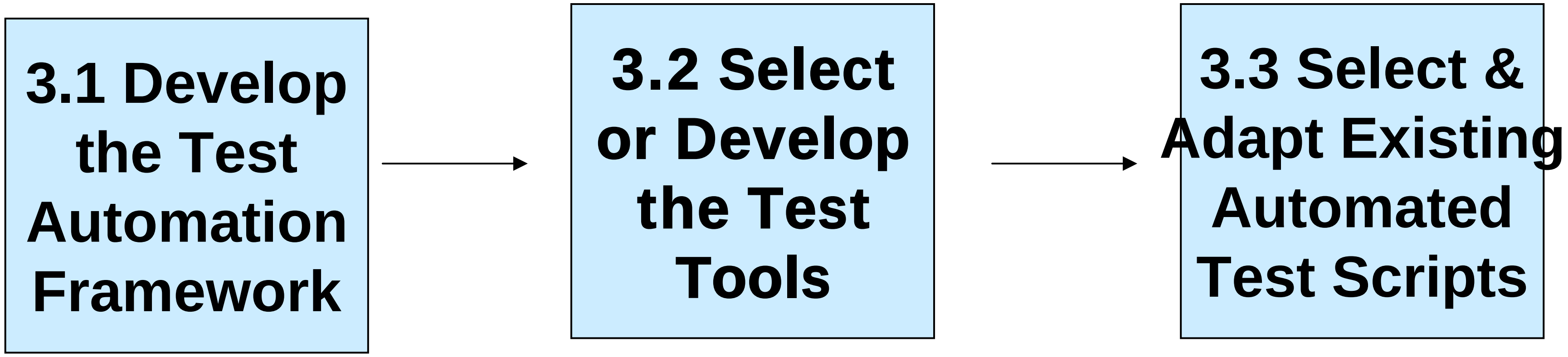
2. Plan the Test

D. Plan the Transition To Live Operation


2.16 Coordinate with the Post-Delivery Plans for Live Monitoring
2.17 Develop the Migration Strategy

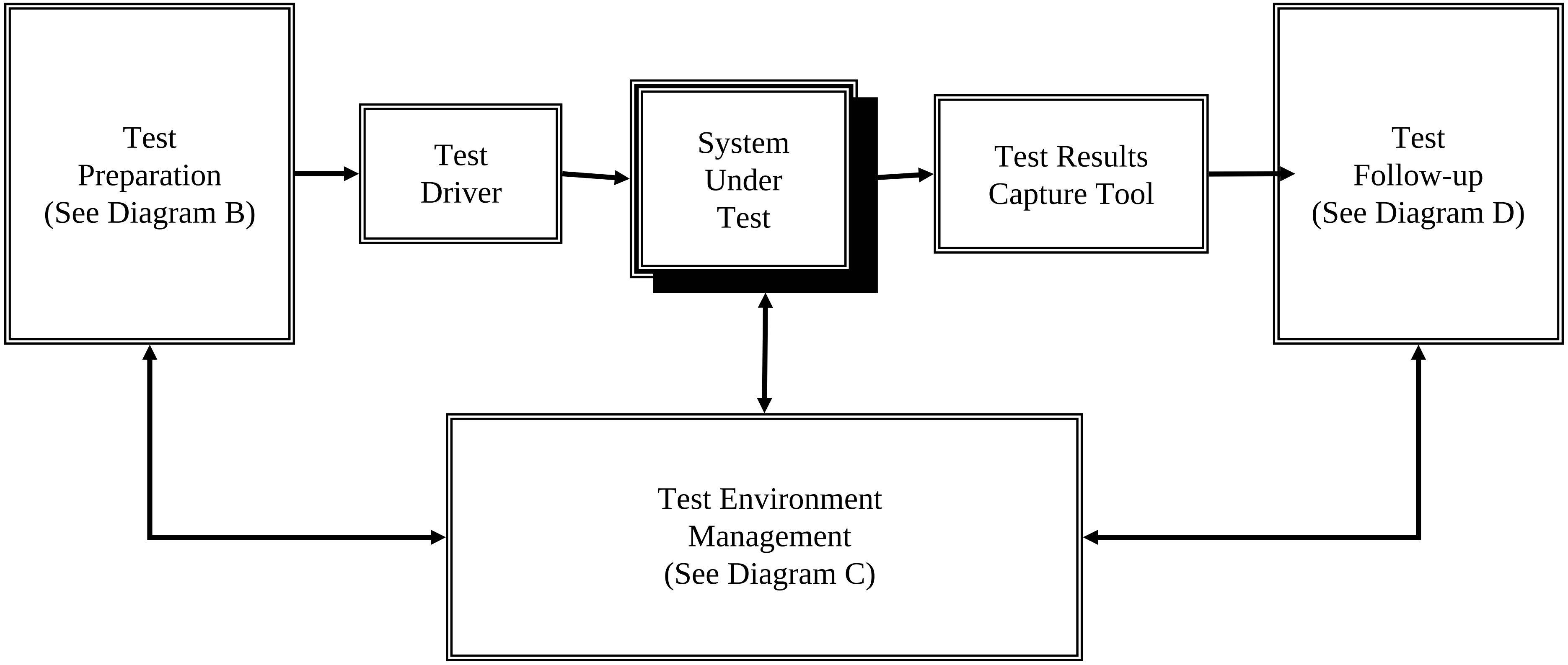
3. Develop the Test Ware

A. Automate the Performance Test


3.1 Develop the Test Automation Framework
3.2 Select or Develop the Test Tools
3.3 Select & Adapt Existing Automated Test Scripts

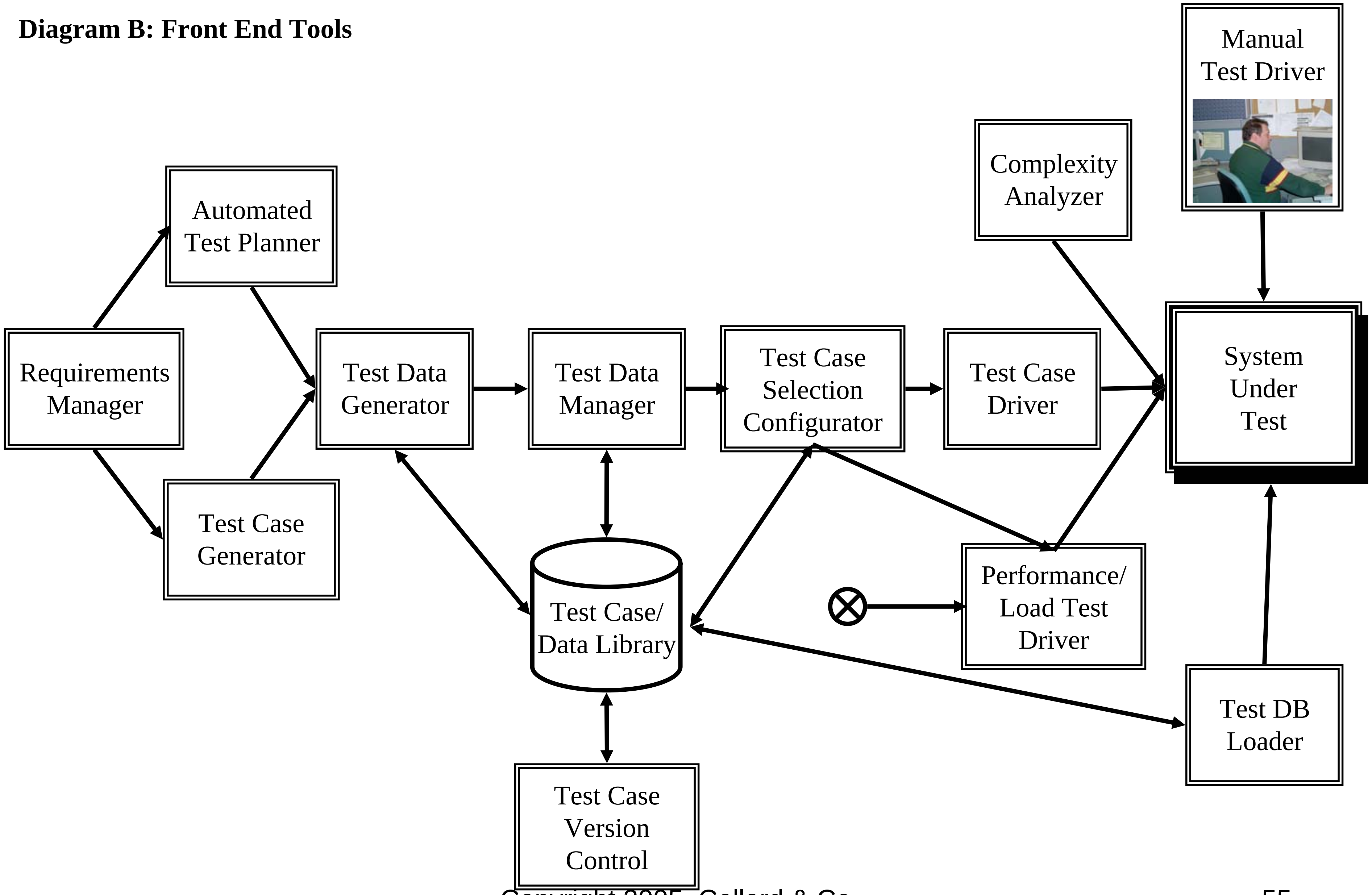
Test Automation Framework

Diagram A: Overview



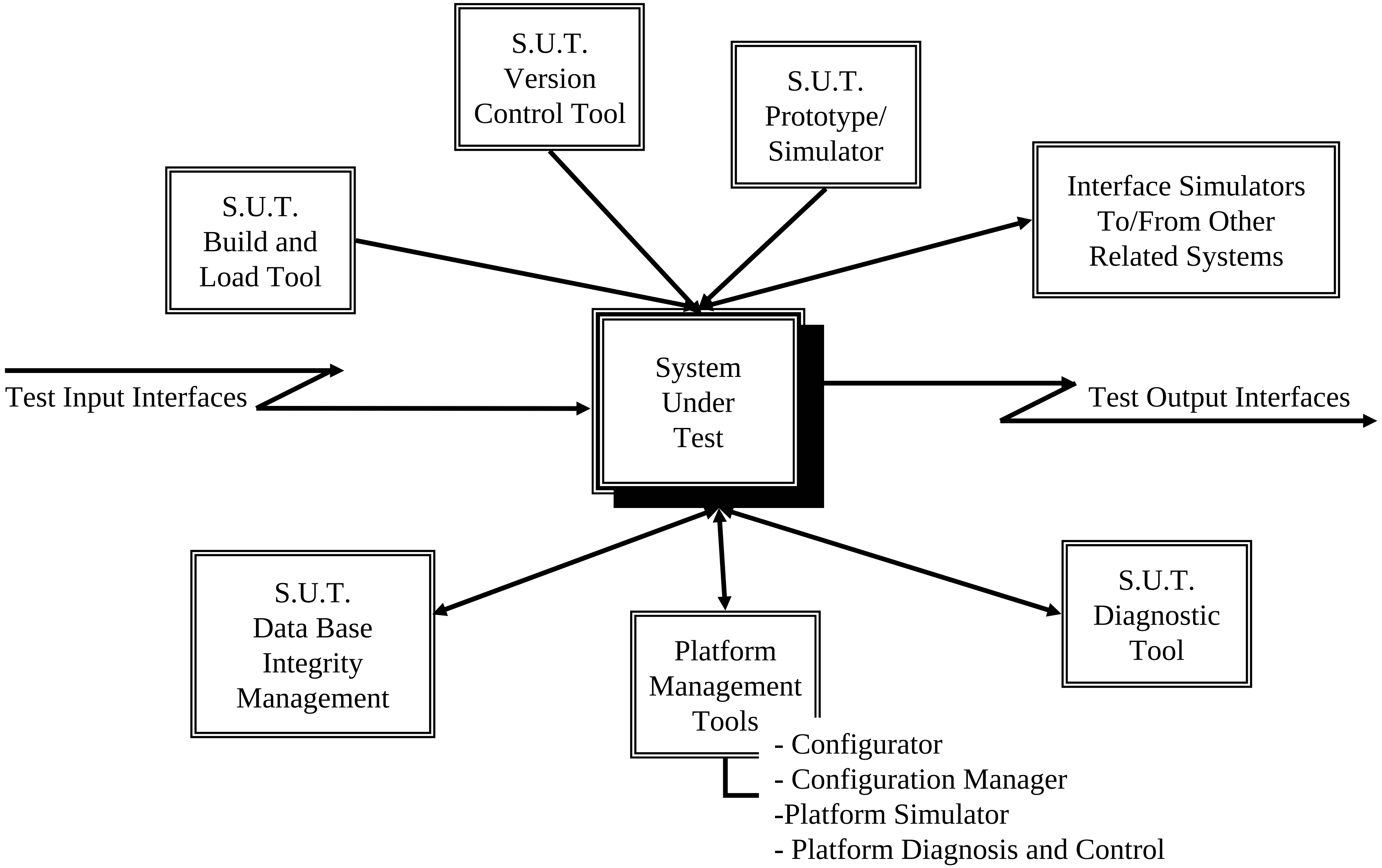
Test Automation Framework

Diagram B: Front End Tools



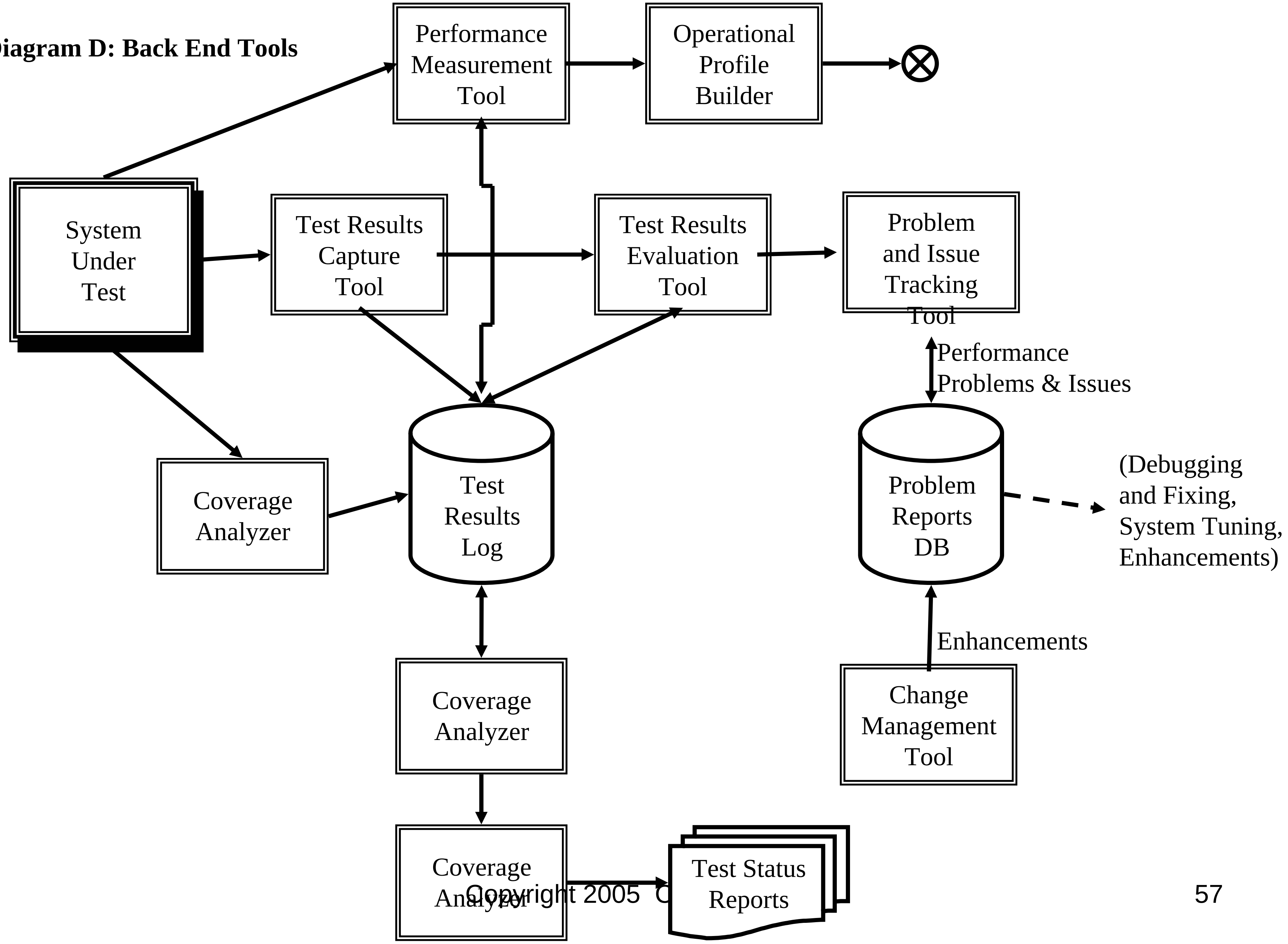
Test Automation Framework

Diagram C: Test Environment Management


S.U.T. Version Control Tool
S.U.T. Prototype/ Simulator
S.U.T. Build and Load Tool
Interface Simulators To/From Other Related Systems
Test Input Interfaces
System Under Test
Test Output Interfaces
S.U.T. Data Base Integrity Management
Platform Management Tools
- Configurator
- Configuration Manager
-Platform Simulator
- Platform Diagnosis and Control
S.U.T. Diagnostic Tool

Test Automation Framework

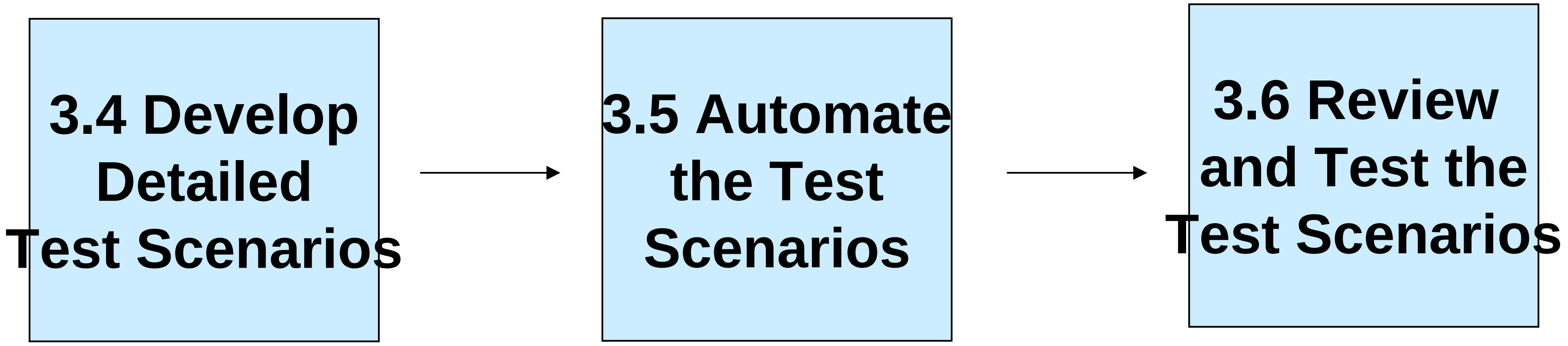
Diagram D: Back End Tools



Performance Measurement Tool
Operational Profile Builder
System Under Test
Test Results Capture Tool
Test Results Evaluation Tool
Problem and Issue Tracking Tool
Performance Problems & Issues
Coverage Analyzer
Test Results Log
Problem Reports DB
(Debugging and Fixing, System Tuning, Enhancements)
Enhancements
Coverage Analyzer
Change Management Tool
Coverage Analyzer
Test Status Reports

3. Develop the Test Ware

B. Develop the Test Scenarios


3.4 Develop Detailed Test Scenarios
3.5 Automate the Test Scenarios
3.6 Review and Test the Test Scenarios

3.4: Develop the Detailed Test Scenarios

- *Write* a detailed test scenario *before* starting to automate it.
  - Tends to force us to think things through.
  - Otherwise the tail (the test tool) wags the dog.
  - A fool with a tool is still a fool – but more dangerous.
  - Consider this task to be like writing much of the dialog for a screenplay (that's why we call it a script).

Test Scenarios (Continued)

- A good performance test scenario:
  - Is linked to performance goals.
  - Provides adequate direction to the tester/automater.
  - Defines test ware and support needs.
  - Shows how to set up and run the test.
  - Identifies what to monitor.
  - Identifies what to do with the collected data.
  - Is justifiable, realistic and feasible.
  - Is likely to be fruitful.
  - Is adequately described.

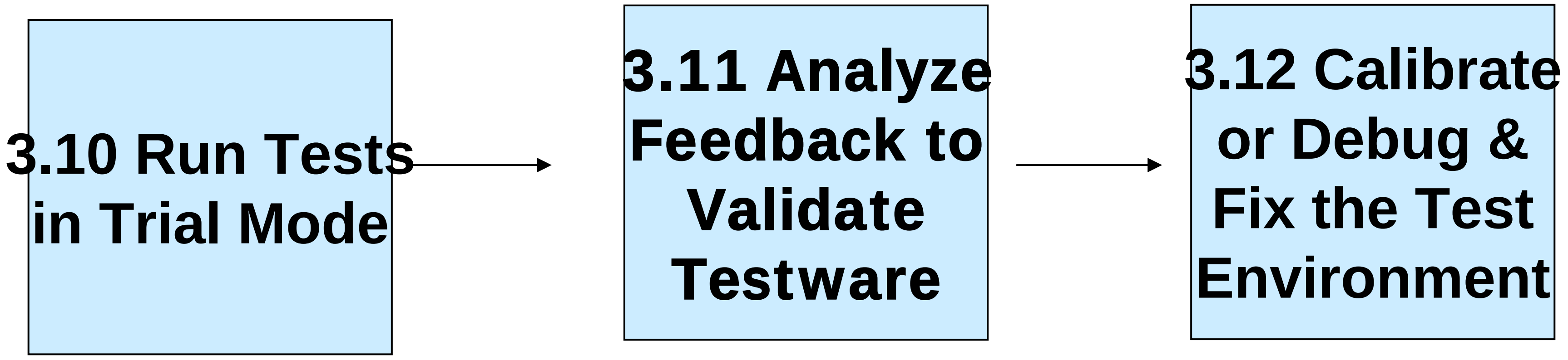
3. Develop the Test Ware

C. Set Up the Performance Test Lab

$$\begin{tikzcd}
\boxed{\text{3.7 Acquire the Test Equipment}} \arrow[r] & \boxed{\text{3.8 Load the Test Databases and Control Files}} \arrow[r] & \boxed{\text{3.9 Configure and Calibrate the Test Equipment}}
\end{tikzcd}$$

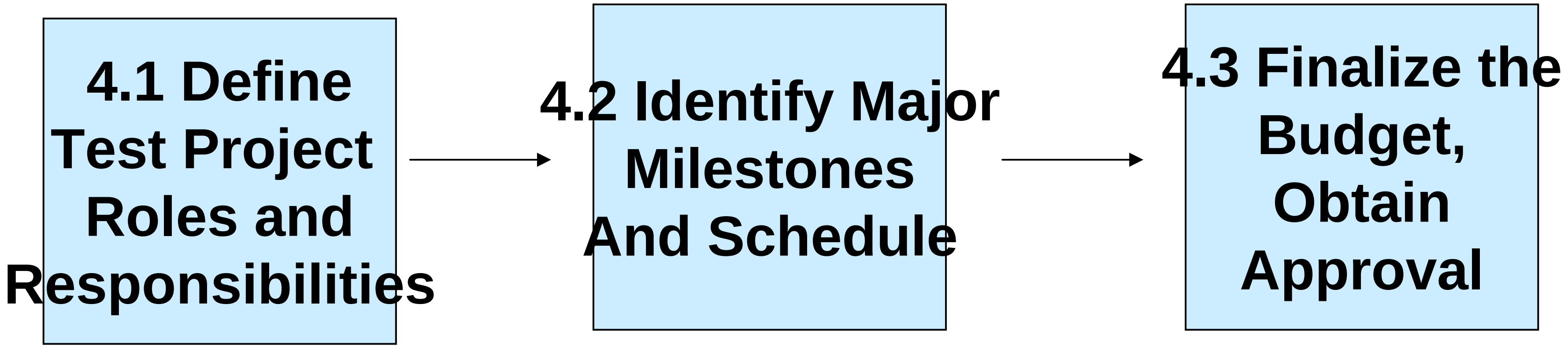
3. Develop the Test Ware

D. Exercise the Test Facilities Early in Trial Runs


3.10 Run Tests in Trial Mode
3.11 Analyze Feedback to Validate Testware
3.12 Calibrate or Debug & Fix the Test Environment

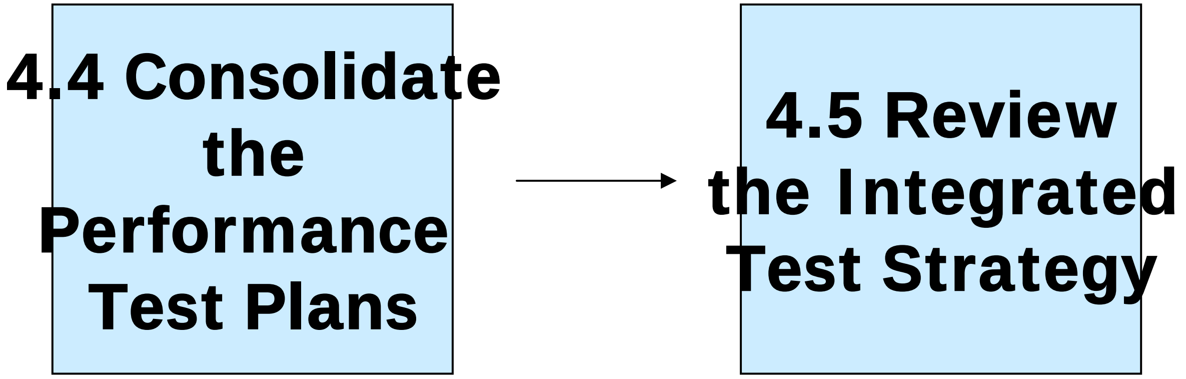
4. Finalize the Strategy

A. Add Roles, Schedules, Milestones & Budgets to the Plans


4.1 Define Test Project Roles and Responsibilities
4.2 Identify Major Milestones And Schedule
4.3 Finalize the Budget, Obtain Approval

4. Finalize the Strategy

B. Consolidate and Review the Proposed Strategy


4.4 Consolidate the Performance Test Plans
4.5 Review the Integrated Test Strategy

Useful Artifacts

- System architecture diagrams.
- Operational profiles of usage patterns.
- Math, statistics and probability tutorials.
- Templates and examples of performance requirements, load calculations, design reviews for testability, test scenarios.
- Checklists of test methods, test overheads, etc.
- Guidelines for initial impact assessments, performance test plan reviews, estimation, etc.

Next Steps

How can you use the ideas from this presentation? Please take 5 minutes to jot down your answers to these questions:

- What ideas are of value to me?
- How can I apply them in my work?
- What support do I need from the boss and others, in order to succeed?