Performance Testing Software Systems: A Heuristic Approach

Derived from:

*Microsoft patterns & practices*

*Performance Testing Guidance for Web Applications*

By: J.D. Meier, Carlos Farre, Prashant Bansode, Scott Barber, Dennis Rea

© 2007 Microsoft Corporation. All rights reserved.

http://www.codeplex.com/PerfTestingGuide

Scott Barber
Chief Technologist
PerfTestPlus, Inc.
Scott Barber, (A.K.A. “The Perf Guy”)  
sbarber@perftestplus.com  

Chief Technologist, PerfTestPlus, Inc.  
www.perftestplus.com  

Vice President & Executive Director,  
Association for Software Testing  
www.associationforsoftwaretesting.org  

Contributing Author & Expert Advisor  
*Performance Testing Guidance for Web Applications*  
www.codeplex.com/PerfTestingGuide
Credits

Some of this material was developed for, or inspired by, *Performance Testing Guidance for Web Applications*, a Microsoft patterns & practices book by J.D. Meier, Scott Barber, Carlos Farre, Prashant Bansode, and Dennis Rea.

Many ideas in this course were inspired or enhanced by colleagues including Alberto Savoia, Roland Stens, Richard Leeke, Mike Kelly, Nate White, Rob Sabourin, Chris Loosley, Ross Collard, Jon Bach, James Bach, Jerry Weinberg, Cem Kaner, Dawn Haynes, Karen Johnson, and the entire WOPR community.

Most of the concepts in this presentation are derived from publications, presentations, and research written and/or conducted by Scott Barber.

Many ideas were improved by students who took previous versions of this course, back to 2001.

This course has been heavily influenced by:


*Just-In-Time Testing* (Robert Sabourin, ©1998-2007 Amibug, Inc.)
I Assume That You:

Test software performance or manage someone(s) who does.
Have at least some control over the design of your tests and some time to create new tests.
Have at least some influence over your test environment.
Are worried that your test process is spending too much time and resources on things that aren’t important AND/OR
Are worried that your test process doesn’t leave enough time and resources to determine what IS important.
Believe that good testing requires thinking.

Test under uncertainty, resource limitations and time pressure.
Have a major goal to find important problems quickly.
Want to get very good at testing software performance.
“There is no such thing as a ‘junior performance tester’…
but there are people who are new to performance testing.”

--Scott Barber
Instructional Methods That I Use

The Class Presents My Editorial Opinions: I do not make appeals to authority; I speak only from my experiences, and I appeal to your experience and intelligence.

Not All Slides Will be Discussed: There is much more material here than I can cover in detail, so I may skip some of it. (If you want me to go back to something that I skipped, just ask.)

I Need to Hear from You: You control what you think and do, so I encourage your questions about and challenges to the lecture. (Talk to me during the break, too.)

If You Want Specifics, Bring Specifics: I invite you to bring real examples of testing problems and test documents to class. (I am happy to show you how I would work through them.)

The Exercises are the Most Important Part: I sometimes use immersive socratic exercises that are designed to fool you if you don’t ask questions. I usually do not provide all the information you need. Asking questions is a fundamental testing skill!

Instructional Methods That I Use

I am likely to push you

If I call on you, and you don’t want to be put on the spot, just say “Pass!”

But you can push back

What Not to Expect From Me

Untested theory.

Marketing fluff.

Pulled punches to protect the guilty.

The “One True Answer” to anything.

Every concept to apply, precisely as presented, to every context.

Over simplifications without acknowledgement.

A disimpassioned, boring instructor!
Primary Goal of this Course

To teach you how to *think about, organize,* and *manage* performance testing effectively, under time and resource constraints, by examining nine *core principles* common to successful performance testing projects and examining how you can rapidly apply those principles to your project *context.*
Secondary Goal of this Course

To introduce you to how to apply *heuristics* and *oracles* to increase your ability to more *efficiently* and *effectively* achieving the objectives of your performance testing projects.
“Let’s face the truth, performance testing *IS* rocket science.”

--Dawn Haynes
Mnemonic

Mnemonics make difficult to remember information, less difficult to remember.

adjective:
“assisting or intended to assist the memory.”

noun:
“thing intended to assist the memory, as a verse or formula.”

“Mnemonics Neatly Eliminate Man's Only Nemesis – Insufficient Cerebral Storage”

An acrostic mnemonic to remember how to spell “mnemonics”.

- http://www.mnemonic-device.eu
Mnemonics in this Course

Were created by Scott Barber to help Scott organize and remember stuff.

Have been, and will be, revised when revisions help Scott remember stuff better, or remember better stuff.

Have been used and liked by some people other than Scott.

Have proven to be “not so memorable” for some people.

If these mnemonics don’t work for you, create your own!!
Heuristics

Heuristics bring useful structure to problem-solving skill.

adjective:
“serving to discover.”
noun:
“a fallible method for solving a problem or making a decision.”

“Heuristic reasoning is not regarded as final and strict but as provisional and plausible only, whose purpose is to discover the solution to the present problem.”
- George Polya, *How to Solve It*
Types of Heuristics

**Guideword Heuristics:** Words or labels that help you access the full spectrum of your knowledge and experience as you analyze something.

**Trigger Heuristics:** Ideas associated with an event or condition that help you recognize when it may be time to take an action or think a particular way. Like an alarm clock for your mind.

**Subtitle Heuristics:** Help you reframe an idea so you can see alternatives and bring out assumptions during a conversation.

**Heuristic Model:** A representation of an idea, object, or system that helps you explore, understand, or control it.

**Heuristic Procedure or Rule:** A plan of action that may help solve a class of problems.

Some Everyday Heuristics

It’s dangerous to drink and drive.
A bird in hand is worth two in the bush.
Nothing ventured, nothing gained.
Sometimes people stash their passwords near their computers. Try looking there.
Stores are open later during the Holidays.

If your computer is behaving strangely, try rebooting.

If it’s very strange, reinstall Windows.

If it’s a genuinely important task, your boss will follow-up, otherwise, you can ignore it.

Heuristics ≠ Process

A heuristic is not an *edict*. Heuristics require guidance and control of skilled practitioner.

Heuristics are context-dependent.

Heuristics may be useful even when they contradict each other—especially when they do!

Heuristics can substitute for complete and rigorous analysis.

Heuristics in this Course

Were created by Scott to help Scott organize and remember stuff.

Are fallible (ask Scott about times when they have failed him).

Are incomplete (ask Scott about times when he used these and still missed “stuff”).

Are not all relevant to every project and every context.

If these heuristics don’t work for you, create your own (and update the mnemonic)!!
Principles ≠ Process

“One-size-fits-all” approaches fit performance testing poorly.

In my experience, successful performance testing projects involve at least active decisions related to each of the core principles in this course.

Core principles are neither exclusive, nor sequential. They go by many different names, have varying priorities, and may be implicit or explicit.

Core principles are not in themselves an approach or process.

Core principles represent a foundation upon which to build a process or approach based on the context of your project.
Performance Testing Principles:

CCD IS EAR!

(A mnemonic of guideword heuristics)
## Performance Testing Principles

<table>
<thead>
<tr>
<th>Context</th>
<th>Project context is central to successful performance testing.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Criteria</td>
<td>Business, project, system, &amp; user success criteria.</td>
</tr>
<tr>
<td>Design</td>
<td>Identify system usage, and key metrics; plan and design tests.</td>
</tr>
<tr>
<td>Install</td>
<td>Install and prepare environment, tools, &amp; resource monitors.</td>
</tr>
<tr>
<td>Script</td>
<td>Script the performance tests as designed.</td>
</tr>
<tr>
<td>Execute</td>
<td>Run and monitor tests. Validate tests, test data, and results.</td>
</tr>
<tr>
<td>Analyze</td>
<td>Analyze the data individually and as a cross-functional team.</td>
</tr>
<tr>
<td>Report</td>
<td>Consolidate and share results, customized by audience.</td>
</tr>
<tr>
<td>Iterate</td>
<td>&quot;Lather, rinse, repeat&quot; as necessary.</td>
</tr>
</tbody>
</table>
When assessing project context, I

COPE in PUBS

(Another mnemonic of guideword heuristics)
Why do we test software system performance?

To determine compliance with requirements?
To evaluate release readiness?
To assess user satisfaction?
To assist in performance tuning?
To estimate capacity?
To validate assumptions?
To generate marketing statements?
Do you know your performance testing mission?

Do you know the “Commander’s Intent”?

Can you find out?

Might COPE in PUBS help?

Example from my days as a U.S. Army LT:

**Mission:** Secure hilltop 42 NLT 0545 tomorrow.

**Commander’s Intent:** It is my intent that the supply convoy safely cross the bridge spanning the gorge between hilltop 42 and hilltop 57 between 0553 and 0558 tomorrow.
Instructions:

Assemble into groups of 3-5 people (4 is ideal) that you don’t normally work with.

Use COPE in PUBS to describe your current or most recent performance testing project to one another.

Pick the most interesting and jot down some notes about its context using COPE in PUBS.

Be prepared to brief the class on the context of the project you chose AND be prepared to answer questions about the contexts of your teammates.
Context

Project

User

Criteria

Objectives

Procedures

Environments

Criteria

Incidentals

Requirements

Desires

Targets

Mandates

Opportunities

Thresholds

Business

System

Performance Testing Software Systems
Criteria
Performance Criteria are *boundaries* dictated or presumed by someone or something that matters.

**Goals**: Soft Boundaries  
(User Satisfaction)

**Requirements**: Firm Boundaries  
(Business or Legal)

**Thresholds**: Hard Boundaries  
(Laws of Physics)

**Constraints**: Arbitrary Boundaries  
(Budget or Timeline)
Performance Goals are soft boundaries typically representing the opinion of someone that matters.

- Typically target whole-system performance characteristics
- Are often “unsubstantiated opinions”
- Are often unattainable
- Must be well qualified, but can be loosely quantified
Performance Requirements are *firm boundaries* frequently derived from contracts (that matter).

- Are actually required to pass to go live
- Are often externally dictated
- Are often continually monitored in production
- Are typically legally enforceable
- Must be both well qualified and quantified
Performance Thresholds are *hard boundaries* that represent physical properties of a system (that matters).

- Are the maximum acceptable values for component-level resources of interest.
- Based on published hardware or software performance recommendations or direct observation.
- Are practically non-negotiable without an environmental change.
Performance Constraints are *arbitrary boundaries* set or presumed by someone that matters (but likely doesn’t get “it”).

- Boundaries imposed by people, traditions, and/or assumptions
- Are sometimes exceedingly difficult to challenge
- Are almost always worth questioning if they jeopardize the project
Performance Testing Objectives
Performance Testing Objectives

What we actually hope to gain by testing performance

Are sometimes completely unrelated to stated requirements, goals, thresholds, or constraints

Should be the main drivers behind performance test design and planning

Usually indicate the performance-related priorities of project stakeholders

Will frequently override goals in “go-live” decisions

How do we know if we’re meeting our objectives?
How do you evaluate criteria? Know your oracles.

For a video lecture, see: http://www.satisfice.com/bbst/videos/BBSTORACLES.mp4
An oracle is the principle or mechanism by which you recognize a problem.

“..it works”
really means...

“...it appeared at least once to meet some requirement to some degree.”

One or more successes!

Without an oracle you **cannot** recognize a problem

and conversely...

If you think you see a problem, you **must** be using an oracle.
Consistency ("this agrees with that") is an important theme in oracles

History: The present version of the system is consistent with past versions of it.

Image: The system is consistent with an image that the organization wants to project.

Comparable Products: The system is consistent with comparable systems.

Claims: The system is consistent with what important people say it’s supposed to be.

Users’ Expectations: The system is consistent with what users want.

Product: System elements are consistent with comparable elements in the system.

Purpose: The system is consistent with its purposes, both explicit and implicit.

Statutes: The system is consistent with applicable laws and legal contracts.

Familiarity: The system is not consistent with the pattern of any familiar problem.

Consistency heuristics rely on the quality of your models of the product and its context.

All Oracles Are Heuristic

We often do not have oracles that establish a definite correct or incorrect result, in advance.

That’s why we use abductive inference.

No single oracle can tell us whether a program (or a feature) is working correctly at all times and in all circumstances.

That’s why we use a variety of oracles.

Any program that looks like it’s working, to you, may in fact be failing in some way that happens to fool all of your oracles.

That’s why we proceed with humility and critical thinking.

You (the tester) can’t know the deep truth about any result.

That’s why we report whatever seems likely to be a bug.

Coping With Difficult Oracle Problems

Ignore the Problem

Ask “so what?” Maybe the value of the information doesn’t justify the cost.

Simplify the Problem

Ask for testability. It usually doesn’t happen by accident.

Built-in oracle. Internal error detection and handling.

Lower the standards. You may be using an unreasonable standard of correctness or goodness.

Shift the Problem

Parallel testing. Compare with another instance of a comparable algorithm.

Live oracle. Find an expert who can tell if the output is correct.

Reverse the function. (e.g. 2 x 2 = 4, then 4/2 = 2)
Coping With Difficult Oracle Problems

Divide and Conquer the Problem

**Spot check.** Perform a detailed inspection on one instance out of a set of outputs.

**Blink test.** Compare or review overwhelming batches of data for patterns that stand out.

**Easy input.** Use input for which the output is easy to analyze.

**Easy output.** Some output may be obviously wrong, regardless of input.

**Unit test first.** Gain confidence in the pieces that make the whole.

**Test incrementally.** Gain confidence by testing over a period of time.
Instructions:

Reassemble into your group.

Identify at least 3 performance criteria for your project of each type. (Goals, Requirements, Thresholds, and Constraints)

Based on your criteria, identify the top 5 performance testing objectives for your project.

Identify your oracles for the top 5 performance testing objectives.

Be prepared to brief the class on your criteria, objectives, and oracles.
Design
“Enterprise grade load generation tools are designed to look easy in sales demos.

Don’t be fooled.”

--Scott Barber
To help me decide what tests to design, I use IVECTRA

(An acronym of guideword heuristics)
Do I need this test to:

Investigate or Validate/Verify

End-to-End or Component

response Times and/or Resources utilized

under Anticipated or Stressful conditions
When Building Usage Models, I

FIB LOTS

(Yet another mnemonic of guideword heuristics)
**Frequent**

Common activities (get from logs)

Some examples include:
- Resource hogs (get from developers/admins)

**Intensive**

Even if these activities are both rare and not risky

**Business Critical**

SLA's, Contracts and other stuff that will get you sued

**Obvious**

What the users will see and are mostly likely to complain about. What is likely to earn you bad press

**Technically Risky**

New technologies, old technologies, places where it's failed before, previously under-tested areas

**Stakeholder Mandate**

Don’t argue with the boss (too much)
Communicating System Usage
**Intent of Investigation:**
Collect configuration data for tuning. Collect data to assist in validating existing network.

**Prerequisites:**
Static prototype deployed on future production hardware.

**Tasks:**
Determine network bandwidth, validate firewalls & load balancer, evaluate web server settings.

**Tools & Scripts:**
Load generation tool, HTTP scripts to request objects of various sizes from a pool of IP addresses.

**External Resources Needed:**
Firewall, Load Balancer, Network Admins, network monitors, 20 IP addresses for spoofing.

**Risks:**
Schedule delay, availability of administrators, configuration of load generation tool for IP spoofing.

**Data of Special Interest:**
Network bandwidth & latency, load balancer effectiveness, resource consumption, response times.

**Areas of Concern:**
No internal expertise on load balancer configuration.

**Pass/Fail Criteria:**
Adequate available bandwidth, architectural assumptions validated.

**Completion Criteria:**
Critical data collected and assumptions validated.

**Planned Variants:**
1 to 20 IPs, volume of 1 to 500, size from 1Kb to 1mb, configuration settings.

**Execution Duration(s):**
6 days: 2 days ea. network & bandwidth, firewall and load balancer, web server configuration.
Instructions:

Reassemble into your group.

Referencing your previous work, use IVECTRAS and FIBLOTS to design the top priority performance tests for your team’s project.

Revisit your oracles. Will they work as planned? If not, choose new oracles.

Be prepared to brief the class on your tests and your Oracles.
Install

SUT, Load Generator, Monitors, Tools, Utilities, Processes, & Team

Configure

Installations, Data, Hooks, Stubs, Harnesses, & Schedule

Validate

Configurations, Assumptions, Design, Models, Integrations, Gaps, & Support

Adapt & Adjust

Everything so far based on Validations.

Coordinate

Prepare for execution
“Only performance testing at the conclusion of system or functional testing is like ordering a diagnostic blood test after the patient is dead.”

--Scott Barber
Agile Performance Testing Activities

1. Understand Project Vision and Context
2. Identify Reasons for Testing Performance
3. Identify Value of Testing Performance
4. Configure or Update Tools and Load Generation Environment
5. Identify and Coordinate Tasks
6. Execute Task(s)
7. Analyze Results and Report
8. Revisit 1-3, Consider Performance Criteria
9. Reprioritize Tasks
CMMI Performance Testing Activity Flow

1. Understand Process and Compliance Criteria
2. Understand the System and Project Plan
3. Identify Performance Acceptance Criteria
4. Plan Performance Testing Activities
5. Design Tests
6. Configure or Update Tools and Load Generation Environment
7. Implement Test Design
8. Execute Work Items
9. Report Results and Archive Data
10. Modify Plan and Gain Approval
11. Return to Step 5
12. Prepare Final Report
Instructions:

Reassemble into your group.

Use the 9 core principles of successful performance testing projects to create an approach or process for your team’s project.

Consider how this approach or process will mesh with the overall project approach or process.

Be prepared to brief the class on your process.
Script
“MacGyver is a super-hero, *not* a career path.”

--Scott Barber
When creating scripts, I try to:

FIND HARM

(Yet another mnemonic of guideword heuristics)
<table>
<thead>
<tr>
<th>Functionality</th>
<th>Ensure obvious functional errors are detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Consider “pre-script” validation and/or transformation</td>
</tr>
<tr>
<td>Navigation</td>
<td>If you think there are 3 ways to do something, users will find 5… the other two could be performance killers</td>
</tr>
<tr>
<td>Data</td>
<td>Vary data to avoid unintentional caching and determine the difference between “speed” and “volume” effects</td>
</tr>
<tr>
<td>Human variability</td>
<td>“Super-users” yield “Stinky-scripts”</td>
</tr>
<tr>
<td>Abandon</td>
<td>Not logging out can leave resource consuming session artifacts</td>
</tr>
<tr>
<td>Ramping &amp; Marching</td>
<td>The step model, is not just unrealistic, it can invalidate results</td>
</tr>
<tr>
<td>Maintainability</td>
<td>If you don’t design your scripts to be maintainable, they won’t be maintained</td>
</tr>
</tbody>
</table>
Load generation tools:
• Do not interact with client side portions of the application.
• Do not natively evaluate correctness of returned pages.
• Often don’t handle conditional navigation.
• Do not handle abandonment well.

Scripting concepts:
• Record – EDIT – playback
• Add data variance
• Add delays
• Add conditional logic
• Add code to evaluate correctness of key pages
• Add abandonment functions
Real Users React

Ensure your tests represent the fact that real users react to the application.

Vary Data

Make sure that data being entered is unique for each simulated user.

Make sure that each simulated user is unique (this may mean more than just separate IDs and Passwords).

Vary Navigation Paths

If there is more than one way for a user to accomplish a task in the application, your test must represent that. Different paths through the system often stress different parts of the system.
Users Think… and Type

Guess what? They all do it at different speeds!

Guess what else? It's your job to figure out how to model and script those varying speeds.

Determine how long they think

Log files
Industry research
Observation
Educated guess/Intuition
Combinations are best
Abandonment

If a page takes too long to display, users will eventually abandon your site – thus lessening the load – changing the overall performance.

Not simulating abandonment makes your test unintentionally more stressful than real life.

<table>
<thead>
<tr>
<th>Page Name</th>
<th>Abandonment Distribution</th>
<th>Abandonment Min Time</th>
<th>Absolute Abandonment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home Page</td>
<td>Normal</td>
<td>5 sec</td>
<td>30 sec</td>
</tr>
<tr>
<td>Pay Bill</td>
<td>Uniform</td>
<td>10 sec</td>
<td>240 sec</td>
</tr>
<tr>
<td>Search Web</td>
<td>Negexp</td>
<td>8 sec</td>
<td>30 sec</td>
</tr>
<tr>
<td>Submit Taxes</td>
<td>Inverse Negexp</td>
<td>30 sec</td>
<td>900 sec</td>
</tr>
<tr>
<td>Validate Field</td>
<td>Normal</td>
<td>5.5 sec</td>
<td>20 sec</td>
</tr>
</tbody>
</table>
Delays

Every page has a think time – after you determine the think time for that page, document it.

These think times should cause your script to pace like real users.

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Event Name</th>
<th>Type</th>
<th>Min</th>
<th>Max</th>
<th>Std</th>
<th>Req't</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure name:</td>
<td>Initial Navigation()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timer name:</td>
<td>tmr_home_page</td>
<td>negexp</td>
<td>4</td>
<td>N/A</td>
<td>N/A</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Timer name:</td>
<td>tmr_login</td>
<td>normdist</td>
<td>2</td>
<td>18</td>
<td>4.5</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Timer name:</td>
<td>tmr_page1</td>
<td>linear</td>
<td>5</td>
<td>35</td>
<td>N/A</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Timer name:</td>
<td>tmr_data_entry</td>
<td>negexp</td>
<td>8</td>
<td>N/A</td>
<td>N/A</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Timer name:</td>
<td>tmr_page2</td>
<td>normdist</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Timer name:</td>
<td>tmr_submit_transaction</td>
<td>linear</td>
<td>2</td>
<td>4</td>
<td>N/A</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Timer name:</td>
<td>tmr_signout</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>
Frightening

Not Frightening
Actual Distribution of User Activities Over Time

Start of Model

End of Model

Users

Time
Server Perspective of User Activities

Start of Model

End of Model

Time
Instructions:

Reassemble into your group.

Discuss the most challenging scripting problems you anticipate based on your test design.

Consider alternatives to achieving the same degree of realism while minimizing the scripting challenge.

Be prepared to brief the class on your scripting challenges.
To remind me that execution doesn’t simply mean “break it”, I recall:

(A mnemonic of guideword heuristics)
(Not to mention an oddly placed Shakespeare reference.)
Determine what the script(s) actually do (Accuracy).

Check script(s), data, etc. for consistency (Precision).

If it’s worth checking more than once, it’s likely worth trending.

Vary between alternate extremes over a definable period.

Establishing an understood, reliable point of reference.

Increase the load systematically until learning stops.

Usage won’t be exactly what you think, ask “what if…?”

Work as a collaborative, cross-functional team.

When something looks odd “beat on it to see if it breaks”.
Execution Heuristics:

- “1, 3, 7, 11, More”
- “Best, Expected, Worst”
- “Marching & Resonance”
- “What if Greenspan sneezes?”
- “First user on Monday”
- “UAT under load”
- “If I can’t break it, I don’t understand it”
Instructions:

Reassemble into your group.

Spend a few minutes jotting down execution heuristics that may be valuable for your team’s project.

Be prepared to describe your heuristics to the class.
“With an order of magnitude fewer variables performance testing could be a science, but for now, performance testing is at best a scientific art.”

--Scott Barber
When I’m analyzing, I remind myself to:

G, STOP & CARE

(A mnemonic of guideword heuristics)
<table>
<thead>
<tr>
<th>Analyze</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C</strong>onfigurations</td>
</tr>
<tr>
<td><strong>S</strong>ignificance &amp; Repeatability</td>
</tr>
<tr>
<td><strong>T</strong>rends</td>
</tr>
<tr>
<td><strong>O</strong>utliers</td>
</tr>
<tr>
<td><strong>P</strong>atterns</td>
</tr>
<tr>
<td><strong>C</strong>ompliance</td>
</tr>
<tr>
<td><strong>A</strong>ccuracy</td>
</tr>
<tr>
<td><strong>R</strong>esources &amp; Times</td>
</tr>
<tr>
<td><strong>E</strong>rrors &amp; Functionality</td>
</tr>
</tbody>
</table>
Methods:
- Blink
- De-Focus & Re-Focus
- Overlay
- Plot
- Bucket
- Look for Odd
- Be Derivative
- Ditch the Digits
- Un-average Averages
- Manual
Facts:

- Analysis is a team sport.
- We cannot prove anything.
- Focus on patterns, trends, and feelings.
- Numbers are meaningless out of context.
- Qualitative feedback is at least as relevant as quantitative feedback.
All three have an average of 4.

Which has the “best” performance”?

How do you know?
Instructions:

Reassemble into your group.

Pay attention, I’m going to explain this group of exercises orally.

Be prepared to describe your findings with the class.
“Linear extrapolation of performance test results is, at best, black magic.

Don’t do it (unless your name is Connie Smith, PhD. or Daniel Menasce, PhD.)”

--Scott Barber
I name good reports:

TRAVIS

(A mnemonic of guideword heuristics)
Stakeholders need data to make decisions. Many decisions can’t wait until tomorrow.

Reports are only interesting if they contain data that is useful.

A great report for developers is probably a lousy report for executives.

Try to use pictures over numbers and numbers over words. Save words for recommendations.

Strive to make reports compelling without explanation.

Unless you are hiding something, make the supporting data available to the team.
Facts:

- Most people will never read performance test results docs.
- Most people don’t really understand the underlying components to performance.
- It is our job to make it easy for them to understand, and understand quickly.
- Being skilled at graphical presentation of technical information is critical for us to help others understand the message we are delivering.
- Confusing charts and tables lead to wrong decisions causing lost $ and ruined reputations.
What consumers of reports want:

- Answers… NOW! (They might not even know the question)
- To understand information intuitively.
- Simple explanations of technical information.
- To be able to make decisions quickly and have the information to support those decisions.
- “Trigger phrases” to use with others.
- Concise summaries and conclusions.
- Recommendations and options.
What consumers of reports usually get:

<table>
<thead>
<tr>
<th>Timer Name</th>
<th>Baseline</th>
<th>250</th>
<th>500</th>
<th>750</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
</tr>
<tr>
<td>ec.Main.Page</td>
<td>10.46</td>
<td>18.05</td>
<td>6.41</td>
<td>8.22</td>
<td>6.33</td>
</tr>
<tr>
<td>ec.login help</td>
<td>0.98</td>
<td>0.98</td>
<td>0.56</td>
<td>0.59</td>
<td>0.55</td>
</tr>
<tr>
<td>ec.login</td>
<td>5.35</td>
<td>7.92</td>
<td>6.66</td>
<td>11.84</td>
<td>6.75</td>
</tr>
<tr>
<td>quick.learns</td>
<td>6.66</td>
<td>6.67</td>
<td>5.91</td>
<td>10.98</td>
<td>5.92</td>
</tr>
<tr>
<td>view.quick.learns</td>
<td>15.66</td>
<td>17.11</td>
<td>5.53</td>
<td>10.72</td>
<td>3.69</td>
</tr>
<tr>
<td>view.lag.window</td>
<td>2.45</td>
<td>2.45</td>
<td>1.47</td>
<td>1.52</td>
<td>1.53</td>
</tr>
<tr>
<td>view.lag</td>
<td>0.67</td>
<td>0.67</td>
<td>0.60</td>
<td>0.63</td>
<td>0.58</td>
</tr>
<tr>
<td>view.ec.status</td>
<td>8.08</td>
<td>12.55</td>
<td>1.73</td>
<td>6.66</td>
<td>1.80</td>
</tr>
</tbody>
</table>
Strive for something better:

- Concise verbal descriptions.
- Well formed, informative charts (pretty pictures).
- Focus on requirements and business issues.
- Don’t be afraid to make recommendations or draw conclusions!
- Make all supporting data available to everyone, all the time (Don’t sit on data ‘cause they won’t understand it).
- Report ≠ Document
- Report *AT LEAST* every 48 hours during execution.
Inspired by “ET”:

Edward Tufte, Ph.D., Professor Emeritus of political science, computer science and statistics, and graphic design at Yale.

According to ET:

Power Corrupts...
PowerPoint Corrupts Absolutely.
Relative Performance

Build Performance Summary (STB 2000)

Test Cases

Over/Under Percent

- TC1
- TC2
- TC3
- TC4
- TC5
- TC6
- TC7
- TC8
- TC9
- TC10

- vs. Baseline
- vs. Requirements
Graphs Make Some Things Obvious

TC1: Tuning: Analog Channel Up

- **Max**: Red diamonds
- **90th**: Blue crosses
- **Avg**: Yellow triangles
- **Min**: Green squares

Seconds vs. STB
Trends, Trends, Trends!!!
MotoSoc 5149, Standard Def Channel to Guide
Response vs. Time

Response Time

Relative Time into Test

Seconds
Avg: 0.00.64
Median: 0.00.66
90th: 0.00.66
STD: 0.00.07
Min: 0.00.55
Max: 0.00.98
MotoSoc 5129, Standard Def Channel to Guide
(STB on, but unused for 2 days prior to test)

Response vs. Time

Response Time

Relative Time into Test

<table>
<thead>
<tr>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg: 0.0198</td>
</tr>
<tr>
<td>Median: 0.0197</td>
</tr>
<tr>
<td>90th: 0.0219</td>
</tr>
<tr>
<td>STD: 0.0022</td>
</tr>
<tr>
<td>Min: 0.0098</td>
</tr>
<tr>
<td>Max: 0.0241</td>
</tr>
</tbody>
</table>
MotoSoc 5129, Standard Def Channel to Guide
(After Power Cycle, until Box became unresponsive)

Response vs. Time

<table>
<thead>
<tr>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg: 0:01.16</td>
</tr>
<tr>
<td>Median: 0:00.98</td>
</tr>
<tr>
<td>90th: 0:01.86</td>
</tr>
<tr>
<td>STD: 0:00.45</td>
</tr>
<tr>
<td>Min: 0:00.55</td>
</tr>
<tr>
<td>Max: 0:03.50</td>
</tr>
</tbody>
</table>
**MotoSoc 5129, Standard Def Channel to Guide**
*(After Power Cycle, 2 min delay between actions)*

**Response vs. Time**

<table>
<thead>
<tr>
<th>Seconds</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg:</td>
<td>0:02.75</td>
</tr>
<tr>
<td>Median:</td>
<td>0:01.20</td>
</tr>
<tr>
<td>90th:</td>
<td>0:07.63</td>
</tr>
<tr>
<td>STD:</td>
<td>0:02.97</td>
</tr>
<tr>
<td>Min:</td>
<td>0:00.66</td>
</tr>
<tr>
<td>Max:</td>
<td>0:10.00</td>
</tr>
</tbody>
</table>
MotoSoc, Menu to Guide

Build 5129 vs. Build 5149

Response vs. Time
(Moving Average, Prev 10 samples)

<table>
<thead>
<tr>
<th></th>
<th>Bld. 5129</th>
<th>Bld. 5149</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg</td>
<td>0:01.60</td>
<td>0:00.70</td>
</tr>
<tr>
<td>Median</td>
<td>0:00.98</td>
<td>0:00.66</td>
</tr>
<tr>
<td>90th</td>
<td>0:01.97</td>
<td>0:00.77</td>
</tr>
<tr>
<td>STD</td>
<td>0:01.90</td>
<td>0:00.08</td>
</tr>
<tr>
<td>Min</td>
<td>0:00.66</td>
<td>0:00.07</td>
</tr>
<tr>
<td>Max</td>
<td>0:10.00</td>
<td>0:00.88</td>
</tr>
</tbody>
</table>
Instructions:

Reassemble into your group.

Spend a few minutes discussing and sketching 1 graphic that would convey the key information to stakeholders that you don’t believe they are getting now.

Be prepared to let the class assess your graphic.
Iterate
Don’t confuse “Delivery” with “Done”

You will never have enough data (statistically), even if you already have too much (to parse effectively).

Ask “Rut or Groove”.

Don’t let complacency be your guide.

If you run out of new ideas, take old ideas to new extremes.

Above all else ask:

“What test that I can do right now, will add the most informational value to the project?”
# Performance Testing Principles

<table>
<thead>
<tr>
<th>Context</th>
<th>Project context is central to successful performance testing.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Criteria</td>
<td>Business, project, system, &amp; user success criteria.</td>
</tr>
<tr>
<td>Design</td>
<td>Identify system usage, and key metrics; plan and design tests.</td>
</tr>
<tr>
<td>Install</td>
<td>Install and prepare environment, tools, &amp; resource monitors.</td>
</tr>
<tr>
<td>Script</td>
<td>Script the performance tests as designed.</td>
</tr>
<tr>
<td>Execute</td>
<td>Run and monitor tests. Validate tests, test data, and results.</td>
</tr>
<tr>
<td>Analyze</td>
<td>Analyze the data individually and as a cross-functional team.</td>
</tr>
<tr>
<td>Report</td>
<td>Consolidate and share results, customized by audience.</td>
</tr>
<tr>
<td>Iterate</td>
<td>&quot;Lather, rinse, repeat&quot; as necessary.</td>
</tr>
</tbody>
</table>
Questions
Contact Info

Scott Barber
Chief Technologist
PerfTestPlus, Inc

E-mail: sbarber@perftestplus.com  
Web Site: www.PerfTestPlus.com