Getting Things Done: Practical Web/e-Commerce Application Stress Testing

Robert Sabourin
President
AmiBug.Com, Inc.
Montreal, Canada
rsabourin@amibug.com
Practical Web/e-Commerce Application Stress Testing

Overview:

– Definitions
– Stress Testing & Development Process
– Stress Testing & Requirements
– Stress Testing & Analysis - Design
– Stress Testing & Development
– Stress Testing & Testing Process
– Stress Testing Tools & Services
Practical Web/e-Commerce Application Stress Testing

- Robert Sabourin, Software Evangelist
- President
- AmiBug.Com Inc.
- Montreal, Quebec, Canada
- rsabourin@amibug.com

What is Stress Testing?

• Testing operational characteristics of an application within a harshly constrained environment
  – Limit processor speed
  – Low memory, disk space
  – Diminished access to shared resources
  – Physical Environment, Static, Temperature, Humidity
Stress Testing Embedded Software

1. ENVIRONMENTAL CONDITION # 1
   → TEST

2. ENVIRONMENTAL CONDITION # 2
   → TEST

3. ENVIRONMENTAL CONDITION # 3
Stress Testing can take place as part of each phase of development.
Stress Testing can take place as part of each core workflow involved in development organization.
<table>
<thead>
<tr>
<th>Core Workflow</th>
<th>Requirements</th>
<th>Analysis</th>
<th>Design</th>
<th>Development</th>
<th>Testing</th>
<th>Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase</td>
<td>Inception</td>
<td>Elaboration</td>
<td>Construction</td>
<td>Transition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements: Performance</td>
<td>Requirements: Adapt as required</td>
<td>Architecture: Root cause failure analysis</td>
<td>Design: Identify weakest link</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load</td>
<td>New technologies</td>
<td>Re-factor architecture</td>
<td>Complexity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalability</td>
<td>New business needs</td>
<td>Adapt to new technologies</td>
<td>Reassess reliable new technology</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost, System Constraints</td>
<td>New budget</td>
<td>Hooks to facilitate stress testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formal inspections</td>
<td>Changing System Constraints</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture: Computer Simulation</td>
<td>Architecture: Computer Simulation</td>
<td>Design: Design alternatives</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototyping</td>
<td>Prototyping</td>
<td>Design alternatives</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design alternatives</td>
<td>Design alternatives</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fall backs</td>
<td>Fall backs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design: Reliability</td>
<td>Design: Reliability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weakest link</td>
<td>Weakest link</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testability hooks</td>
<td>Testability hooks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robust middleware</td>
<td>Robust middleware</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost tradeoffs</td>
<td>Cost tradeoffs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development: Develop unit test harness</td>
<td>Development: Develop unit test harness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop test case repository</td>
<td>Development: Develop test case repository</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Determine physical components to stress and define strategy</td>
<td>Development: Determine physical components to stress and define strategy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Team: Work with development to develop test case repository</td>
<td>Integration: Stress test as new modules come on line</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Support developing test harness</td>
<td>System: Stress test early in lab environment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set up test lab and build strategy</td>
<td>Live: Work with site monitoring team</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plan for all measuring methods</td>
<td>Preparing: Plan upgrade strategy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitoring: Site performance under load</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Study usage patterns</td>
<td>Server memory usage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security breaches</td>
<td>Server processor usage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customer service</td>
<td>Database usage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identify site monitor partners</td>
<td>System resources</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Requirements

- Response time, end user experience
  - Slow vs High Speed connections
- Number of concurrent users
  - Normal vs. Peak
  - Doing what?
- Performance Degradation
- Reliability
  - MTTR, MTTF
### Attributes

**Stress Impact**
- Performance degradation
  - Ensure delays never exceed about 8-10 seconds independent of load

**Usability**
- Additional resources should increase capacity
  - Vertical or horizontal scaling. Bandwidth, Memory, Processing Power, CPUs, Servers

**Scalability**
- Success will lead to increased load needing upgrades
  - If underlying components are upgraded does the system still have the same reliability?

**Serviceability**
- What is the time to failure, can we predict it? MTTF
  - How does the system act when a process or thread fails? Do we recover?

**Reliability**
- Patches, New releases, MTTR
  - Database schemas updates

**Maintainability**
- Support stress tests!
  - Do any special test hooks (pages or APIs) work under stress?

**Testability**
- New technology - does weakest point change?

**Adaptability**
- Does addition of new services impact capacity?

**Expandability**
- Re-using stress test? Is reused component weakest link?

**Re-usability**
- Moving to different servers, services

**Portability**
- Weakness of independent servers interoperating!

**Interoperability**
Measurements

- Response time
  - Minimum
  - Maximum
  - Average
- CPU Usage
- Memory
  - available
  - page faults/second
- Disk
  - % Disk Full
Measurements

• Network
  – Bandwidth

• Web Servers
  – Files/Sec
  – Bytes/Sec
  – Maximum Connections
  – Errors
Measurements

• Database Servers
  – Transactions/Second
  – Cache hit ratios
• Functionality
  – Pass?
  – Fail?
  – Relation to load
Analysis & Design

Quebec City Bridge, 1916
- Construction collapse
- Stress due to scale

Tacoma Narrows, 1940
- Collapse during normal operation
- Stress due to instability
- Wind was only 42 mph
Computer Simulation

- Computer simulation is used to study how a design, or architecture, will react to stress!
  - Model typical transactions
  - Model atypical transactions
  - Model harsh transactions
  - What if analysis!
Network Simulators

• Many products on the market
• Used by
  – System architects
  – Software designers
  – Network designers
• Test early
  – *Before code is written*
## Network Simulator Features

<table>
<thead>
<tr>
<th>Type of simulation</th>
<th>Analytical and discrete</th>
<th>Analytical</th>
<th>Analytical, discrete for animation</th>
<th>Analytical and discrete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traffic loading</td>
<td>Messages/hour, message size, reply size, CPU ms, disk ms</td>
<td>Packet rate</td>
<td>Packet size, packet rate</td>
<td>Session rate, packet size, packet rate, response size</td>
</tr>
<tr>
<td>Server load</td>
<td></td>
<td></td>
<td></td>
<td>Via number of packets processed</td>
</tr>
<tr>
<td>Background traffic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load-balancing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Routed protocols supported</td>
<td>BGP, IGRP, OSPF, RIP, EIGRP, IS-IS, optimal</td>
<td>IGRP, IS-IS, OSPF, RIP</td>
<td>RIP, IGRP, OSPF</td>
<td>BGP, IGRP, OSPF, RIP</td>
</tr>
<tr>
<td>Cost accounting</td>
<td>Cost based on proportional use of links, servers and user wait time</td>
<td>Tariff costs</td>
<td>Equipment list</td>
<td>Equipment list</td>
</tr>
<tr>
<td>Scheduling simulation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change control</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web reports</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Topology import</td>
<td>ASCII text file</td>
<td>Compuware EcoScope, CA Unicenter, HP OpenView, Tivoli NetView</td>
<td>HP Network Node Manager, Visio</td>
<td>HP Network Node Manager, Tivoli NetView, Excel</td>
</tr>
<tr>
<td>Traffic Import</td>
<td>ASCII text file</td>
<td>Compuware EcoScope, Network Associates Sniffer Pro, NetScout Systems NetScout</td>
<td></td>
<td>HP NetMetricx, NetScout, Expert Sniffer, six ASCII formats</td>
</tr>
<tr>
<td>Tutorial</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single-unit retail price</td>
<td>$7,500</td>
<td>$24,500</td>
<td>$9,995</td>
<td>$19,000 plus $7,000 for MVI and $9,000 for ESP modules</td>
</tr>
<tr>
<td>Warranty</td>
<td>One year</td>
<td>90 days</td>
<td>30 days</td>
<td>90 days</td>
</tr>
<tr>
<td>Annual maintenance</td>
<td>12% of list</td>
<td>12% of list</td>
<td>24% of list</td>
<td>18% of list</td>
</tr>
<tr>
<td>Maintenance support</td>
<td>24x7 1-800 phone and e-mail technical support during business hours day upgrade</td>
<td>Software updates, documentation updates</td>
<td>Unlimited technical support 24x7, monthly database updates and tool updates, major release, model updates, access to our community through our Web site, access to Client Consultation and Computing Center</td>
<td>Unspecified</td>
</tr>
</tbody>
</table>

---

**Network Computing**

**THE TECHNOLOGY SOLUTION CENTER**
Performance Model

• SPE-ED
  – Software Performance Engineering
  Performance Modeling Tool
  – “…tool that produces performance data on
development alternatives without requiring
extensive knowledge of modeling theory…”
  – www.perfeng.com
Development

• Include stress testing during unit testing!
  – Before and after task assignment
    • Develop test cases as part of task
      – XP’ish is good
    • Does system behave the same?
    • Differences as expected?
  – Fewer surprises later in cycle.
  – Work with testing team to build harness
Test Harness

Module C - Under Unit Testing

Transaction Generated from Test Data
Simulated Transaction

Test Harness

Module A: Completed Unit
Module B: Stub
Module C: Unit Being Developed
Module D: Completed Unit
Module E: Completed Unit
Module F: Stub

Test outcome
Capture and process test results

Elapsed Time

Test Cases
Test Results

Testability Hooks

- It is wise to ensure that testability is considered in all code developed!
  - Simple static page to access each business logic function independent of GUI.
  - Runtime enable and disable of logging features. (*web, business logic, data tier*)
  - Access to key objects.
  - Hooks to facilitate measures!
Approaches

• Peer review on check-in
• Formal inspections especially focused on identifying weak links in design or any problems with architecture or requirements
• Unit testing to include Stress over and above Functional testing
Issues

• Watch out for
  – Moving instabilities
  – Changes to database schemas for new features
  – Developers unfamiliar with code base
  – Reusing code “blindly” to save time
  – Increasing complexity of code
  – Differences between target and developer “Run-time” environments
Stress Testing Environment

- N-tier setup
- Generally
  - Client
  - Application
  - Data
Load Testing

• Load generator
  – Keep the system busy!
  – Simulate of user experiences
  – Several concurrent “Virtual users”
  – Generally you should model most common type of user experiences
  – Experiences can be:
    • TYPICAL
    • HARSH
Transaction Simulator

- Typical user transactions
- Measure how well the system performs these transactions as we vary the load
  - Correct?
  - Time?
Transaction Simulator

• For example:
  – Purchase scenario is run on the transaction simulator
  – We generate a load on the system with the load generators and study how this impacts the typical transactions
  – Load generator is focused on keeping the system busy!
Performance Measures

• User perspective
  – Time to complete a transaction from the user perspective
  – Time to get a response to an input event
    • Should be maximum 8-10 seconds

• Page load
  – Time to last byte

• “Response time” of the system
  – Time from a mouse click until the next page is finished loading
Performance Measures

- Components of “Response time”
  - Request Submission
    - Data to server
  - Processing Time
    - Time spent working on user request
  - Response Receipt
    - Time to send result back to user
Stressing the Data Servers.

- Force intermittent failures
  - In session disconnect

- Fill database
  - Just below capacity
  - Just above capacity
  - Ramp up size

- Resource “hogs”
  - Processor
  - Memory
  - Disk
  - Network bandwidth

- Battery/Blitz of operations
  - Insert
  - Delete
  - Search & Sort
Stress Testing the Business Logic Servers.

- Load generation
  - multiple sessions
  - multiple users
  - harsh operations
  - typical operations

- Continuous busy operation

- Force intermittent failures
  - In session disconnect
  - Third party component failures

- Resource "hogs"
  - Processor
  - Memory
  - Disk
  - Network bandwidth
Stressing the Web Servers.

- Lot’s of HTTP requests
- Attacks from hackers
  - Denial of service
  - Break in
- Forced error handling
  - Invalid operations
  - Generate error conditions
- Force intermittent failures
  - In session disconnect
- Resource “hogs”
  - Processor
  - Memory
  - Disk
  - Network bandwidth
- External Process
  - SSL
  - Firewall
  - External web resource unavailable

Load Test Configuration
- Lot’s of HTTP requests
- Forced error handling
- External Process
- Resource “hogs”
Scalability

• Vertical
  – Replace servers with more powerful systems
  – Add resources to existing servers
• Horizontal
  – Add more servers to the site
  – Needs load balancing technology
  – Increases availability of site
• Functional
  – Separate application functions onto different servers and scale them independently
Stress Test Planning

• Time
  • Generally there is not enough time to do exhaustive stress testing of all components of the system

• Risk
  • Spreading resources across different stress testing activities must be done carefully based on the technical risk and potential business impact of failures
Stress Test Planning

- Technical risk
  - New code
  - Old code used in a new way
  - New developer
  - New hardware
  - New third party stuff
  - High risk
  - Complexity
Stress Test Planning

- Business Impact
  - Impact of performance degradation?
  - Impact of missing functionality?
  - Load balancing?
  - What about diverting CPU power to more popular functions and disabling less popular operations?
Stress Test Planning

• Pattern Evolution:
  • List relevant stress tests
  • *Guestimate* risks working with peers in product management, development and other stakeholders
  • Spread testing across builds relative to risk from highest to lowest
  • Try to implement at least *one Stress Test experiment per build*.
## Stress Test Experiments

### Example Stress Experiment

<table>
<thead>
<tr>
<th>Concurrent Users</th>
<th>Load Generated</th>
<th>Database State</th>
<th>Technical Risk 1 to 10</th>
<th>Business Importance 1 to 10</th>
<th>Exposure</th>
<th>Rank</th>
<th>Build A</th>
<th>Build B</th>
<th>Build C</th>
<th>Build D</th>
<th>Build E</th>
<th>Build F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1000</td>
<td>Normal</td>
<td>Low</td>
<td>3</td>
<td>10</td>
<td>30</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-1000</td>
<td>Harsh</td>
<td>Low</td>
<td>5</td>
<td>5</td>
<td>25</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-1000</td>
<td>Normal</td>
<td>Medium</td>
<td>3</td>
<td>10</td>
<td>30</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-1000</td>
<td>Harsh</td>
<td>Medium</td>
<td>5</td>
<td>5</td>
<td>25</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-1000</td>
<td>Normal</td>
<td>High</td>
<td>8</td>
<td>8</td>
<td>64</td>
<td>1</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-1000</td>
<td>Harsh</td>
<td>High</td>
<td>10</td>
<td>5</td>
<td>50</td>
<td>2</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Critical**: Exposure > 50
- **Serious**: Exposure between 26 and 50
- **Important**: Exposure between 15 and 25
- **Nominal**: Exposure less than 15
Stress Test Preparation

– Warm up, caches and buffers to steady state
– System to desired starting state
– Ensure no one else is accessing system under test
– Ensure databases and system resources are in the correct initial state to start testing
Test Tool Evaluation

• Tools to help you perform stress testing
  – What to buy?
  – When to buy?
  – What to contract out?
  – What to do yourself?
Test Tool Evaluation

• Tools to help you perform stress testing
  – Event Simulation
  – Load Generation
  – Site Monitoring
  – Test Harnesses
  – Environment
    • Monitor
    • Fault injection
Test Tool Evaluation

• What to buy?
  – Things you will *reuse* on many projects.
  – Things which *save you time*
  – Things which will save you *money*.
  – Itemize your *specific requirements*.
  – Can you buy part of the solution now and part later to spread the cost?
  – Can you leverage internal expertise?
  – Will significant training be required?
Test Tool Evaluation

• What to buy?
  – Be careful about maintenance, especially for open source or otherwise free tools!
    • Technologies change fast in www
    • Support for third party gizmos and widgets is important.
    • How quickly does your supplier adapt to new technologies?
    • Will you be able to keep your own, in-house, tools up to date?
Test Tool Evaluation

• What to buy?
  – What if your requirements change!
    • If you swap between Linux, NT, Solaris will the tools still work?
    • If you move between SQL Server, Oracle, Interbase will monitors still work?
    • If you change http servers from Apache to IIS will tools still work?
Test Tool Evaluation

• When to buy?
  – Company issues
    • If your company is going to develop several projects using the same or tightly related technologies then it is wise to tool a test lab independent of a specific project schedule
Test Tool Evaluation

- When to buy?
  - Project issues
    - Identify vendors during early phases
    - Use first iterations to evaluate tools
    - Buy tools before full system testing
    - You should have some sort of load generator during Unit, Integration and System Testing Phases
Test Tool Evaluation

• When to buy?
  – After evaluation.
  – TRY BEFORE YOU BUY.
  – Be honest with vendors and tell them what type of evaluation process you have decided to use.
  – Let them know that other suppliers are also being considered.
  – Ensure they help you get stuff - really - working
Test Tool Evaluation

• What to contract out?
  – Things you are only doing once!
  – Things that you do not have the expertise for in-house! *(first time load scripts)*
  – Things that require investments you are not ready to take yet.
  – Things that will save you time.
  – Things that allow more parallelism in development.
Test Tool Evaluation

• Consultants?
  – Mapping out an effective strategy to achieve your business goals.
  – Offer guidance, coaching, mentoring and management consulting from someone who as been through the experience before.
  Each case is different!
Test Tool Evaluation

- What to do yourself?
  - Unit Test Harnesses
    - Your test and development team know your application and development environment best
    - These tools must change as the project is developed
  - Test Hooks
    - Put in special test access points
    - Simple HTML forms instead of animation
Test Tool Evaluation

• What to do yourself?
  – Monitors
    • Snap shots of system status
    • View system parameters of special interest
    • Special log file analyzers
  – What are you expert in?
    • If you have some expertise available then use it! Just make sure that you don’t just hack a tool together and forget about maintaining and evolving it as your needs change down stream!
Home Brew

- Monitors from Perl/Shell Scripts on Unix Box
  - CPU status
  - Virtual Memory Status
  - Available Disk Space
  - Available DBMS Space
  - Disk Usage
  - DBMS Usage
  - Number of concurrent users
  - Process status
  - Log file analysis
  - Spreadsheets and Excel Macros!
  - Database for tracking stress testing experiments
Testing Services

• Load testing
  – Perform load testing
  – Price generally related to volume

• Site monitoring
  – Remote monitoring and measurement of site performance

• Contract testing
  – Outsource testing to experts
Real World Frustration

- Often, during stress testing, applications fail while you are trying to develop test scripts, procedures or trying to debug old scripts on new builds.
Real World Frustration

- Bugs got you down!
  - Be patient!
  - Focus on stressing application not the testing team!
  - You will find bugs with stress testing because you are stress testing!
  - Sometimes the bug is in the test and sometimes the bug is in the program being test.
Real World Frustration

• Be organized!
  – When you are well organized there will be value in all bugs!
  – Rigor in the build process is the key!
  – Build progressively improve with every build
  – Prioritize and investigate all bugs found!
Thank You

• Questions?