System Testing & Performance

• System Testing
  – system testing vs. unit testing
  – planning for system testing
• Bug Finding
  – bug detection rates
  – release phases & ship criteria
• Performance Management
  – basic principles
  – performance design and repair
  – basic tools and practices

Unit vs. System Testing

• Goals
  – is this component ready to integrate?
  – is this system ready to ship?
• Context
  – testing components in relative isolation
  – testing the entire assemblage
• Focus
  – component functionality and specifications
  – system functionality and specifications
  – “whole system” behavior

Testing Responsibilities

• There are multiple players
  – developers
  – testers working with the developers
  – independent Test Group
• There are many types of testing
  – component functional validation
  – system functional validation
  – usability, security, performance, robustness
• Many theories about who should do which
  – organizational, philosophical, process, moral

Typical System Test Activities

• System functionality and error handling
  – does the system do the things it is supposed to do
  – including the correct handling of all specified error conditions
• Installation testing
  – do all of the parts install and configure correctly
  – different combinations of platforms, install, configuration options
• Usability testing
  – can customers use it
• Security testing
  – authentication, authorization, privacy, attacks
• Interoperability testing
  – platforms, devices, different clients & servers
• Performance testing
  – capacity, throughput, response time
• Stress testing
  – overload, resource exhaustion, error recovery

Planning System Testing

• Some system testing is fairly obvious
  – does the whole system meet specified requirements
  – fully exercise independent component interactions
  – derive test cases from architecture and specifications
• But what other tests do we need to include?
  – standard types of testing:
    – installation, usability, security, interoperability, performance
  – other types of product or domain specific testing?
• System Testing is about confidence
  – what will the product be expected to do?
  – what are we already confident that it can do?
  – what are we not yet sure of?

Testing and Bug Discovery
Basic Principles of Performance

• The Pareto Principle
  – 80% of cycles are spent in 20% of the code
  – in my experience it is more like 90% and 2%
• Performance requires real measurement
  – our intuition usually turns out to be wrong
• Performance demands eternal vigilance
  – if we aren’t getting faster, we’re getting slower
• Performance is mostly about design
  – code optimization is only occasionally useful

Design for Performance

• Establish performance requirements
• Anticipate bottlenecks
  – frequent operations (interrupts, copies, updates)
  – limiting resources (network/disk bandwidth)
  – traffic concentration points (resource locks)
• Design to minimize problems
  – eliminate, reduce use, add resources
  – and, sometimes, optimize implementation
• Include performance in design reviews

Fixing Performance Problems

• is a lot like finding and fixing a bug
  – formulate a hypothesis
  – gather data to verify your hypothesis
  – be sure you understand underlying problem
  – review proposed solutions
  • for effectiveness
  • for potential side effects
  – make simple changes, one at a time
  – re-measure to confirm effectiveness of each
• only harder

For Next Lecture

• McConnell chapter 3-3.2
  – the criticality of planning
  – adapting your model to your problem
• Kampe: S/W Process Models
  – overview of model classes and terms
• Wikipedia: Waterfall Model, Iterative Models
  – the rationale and arguments
• Boehm: Spiral Development
  – a realistic view of the waterfall process
• Laramee: The Game Design Process
  – what formal process looks like in games

In-System Unit Testing

• test new component in a whole system
  – other components with which to interact
  – we are able to exercise more functionality
  – we can identify component interface issues
• add new component to existing system
  – where other components are known good
• test builds of experimental components
  – tracking down problems can be difficult
• this is still unit, and not system testing

Supplementary Slides
Functionality Testing
- does system function as specified?
  - test all specified functional requirements
  - these are part of the acceptance criteria
- may involve a large number of test cases
  - many may be existing unit test cases
  - may include standards conformance suites
  - may include whole-system exercises
  - may include sample scenarios
- this testing should be automated

Robustness Testing
- introduce each specified error
  - invalid options, requests, and data
  - communications errors, resource failures, ...
- verify correct system response
  - check that system properly detects it
  - check that system properly reports it
  - check that system properly responds to it
  - check that system continues working
- this testing should be automated

Installation Testing
- installation is functionality too
  - component install may be checked w/unit test
- whole system install also needs testing
  - follow directions and use the defaults
  - try all of the options and combinations
  - is feedback correct
  - is correct software installed correctly
  - does upgrade work (preserving old data)
- some automated testing, some manual

Usability Testing
- most testing should be automated
  - so that it can be run frequently
  - so that it is run consistently
- this should be complemented with usage
  - users sit down and try to use the software
  - they try do do normal things
  - they try to make obvious mistakes
  - such testing turns up many problems
  - but at some point, the pay-off falls off

Security Testing
- much of this is specified functionality
  - authentication/authorization mechanisms
  - data protection mechanisms
  - response to unauthorized requests
  - covered by automated functionality test cases
- it is also necessary to look for holes
  - design review by domain security experts
  - penetration tests by experienced hackers

Interoperability Testing
- consider everything system depends on
  - hardware, operating system, other services
  - there are probably multiple versions of each
- consider everything system talks to
  - browsers, servers, routers, etc.
  - there are probably multiple versions of each
- we should test on all combinations
  - or one from each equivalence partition
  - defining these requires insight an experience
Stress Testing
• bugs are often found in special cases
  – resource exhaustion, error handling
  – unlikely combinations of events
• stress tests create these continuously
  – traffic generators running at full capacity
    • with changing random mixes of requests
  – continuous error generation
    • with random error selection
• stress tests can run for days or longer
  – they shake out many hard-to-cause problems

Load Generation
• system specifications include capacity
  – support up to 3000 transactions/second
• performance should be measured at load
  – response time a 1500 transactions/second
• many bugs involve concurrent operations
  – locking, allocate/free, protection, etc.
• these require automatic load generation
  – generate traffic of specified types
  – generate traffic at calibrated rates

Pre-FCS Exposure
• give key customers an early sniff
  – so they can better prepare for new release
  – so they can start work on related products
• get additional feedback prior to FCS
  – real customers will use it in different ways
    • they will find different bugs
    • they will find usability issues
    • they will find compatibility issues
• obscure the fact we are behind schedule
  – we shipped something!

Alpha Testing
• alpha products are usually incomplete
  – missing functionality and documentation
  – very buggy
  – non-standard installation and management
• alpha sites are carefully selected
  – they are prepared to deal with problems
  – they can be trusted to exercise the product
• goals of alpha testing
  – gather feedback on key features and content
  – early access for partners, key customers

Beta Testing
• a beta product is very near final form
  – all functionality and documentation complete
  – few significant known problems
• beta sites are real customers
  – they will put the product to real use
  – they have agreed to give us feedback
• goals of beta testing
  – confirm the product is ready to ship
  – gather last-minute feedback
  – early access for key customers

determining ship-ability
• many types of metrics are commonly used
  – list of tests that must be passed
  – hours of load/stress tests passed
  – performance criteria
  – beta report results
  – open bug counts
  – new bug arrival rates
• concurrence (engineering, Q/A, support)
• fiat (product executive says “ship it”)
• criteria must be set at start of project
Performance Testing

- identify key performance metrics
  - throughputs, response times, capacities
  - some may be external competitive numbers
  - some may be internal assessment numbers
- define ways to measure each
  - test transactions and measurement points
- write suites to exercise and measure
  - there are often performance benchmarks
- this testing should be automated

Performance: what to measure

- competitive performance metrics
  - used to compare competing products
    - nominal response time for simple query
    - standard transactions per second
- engineering performance metrics
  - used to spec components
  - used to analyze performance problems
    - time to perform a particular sub-operation
    - channel utilization, idle time, cycles per operation
- must be meaningful and well defined

Perf: meaningful measurements

- measure under controlled conditions
  - on a specified platform
  - under a controlled and calibrated load
  - without other conflicting activities
- ensure validity of results
  - measuring very brief operations
    - ultra-high resolution timers
  - perform a large number of operations
  - repeatability of results
    - collect many samples, explain variations

Execution Profiling

- automated measurement tools
  - compiler options for routine call counting
    - one counter per routine, incremented on entry
  - statistical execution sampling
    - timer interrupts execution at regular intervals
    - increment a counter in table based on PC value
    - may have configurable time/space granularity
  - tools to extract data and prepare reports
    - number of calls, time per call, percentage of time
- very useful in identifying the bottlenecks

Simple execution profiling

<table>
<thead>
<tr>
<th>time</th>
<th>seconds</th>
<th>cum %</th>
<th>procedure (file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.9</td>
<td>0.0029</td>
<td>42.9</td>
<td>print (profsample.c)</td>
</tr>
<tr>
<td>42.9</td>
<td>0.0029</td>
<td>85.7</td>
<td>add_vector (profsample.c)</td>
</tr>
<tr>
<td>14.3</td>
<td>0.0010</td>
<td>100.0</td>
<td>mull_by_scalar (profsample.c)</td>
</tr>
</tbody>
</table>

Profiling with call counting

<table>
<thead>
<tr>
<th>% cumulative</th>
<th>self</th>
<th>total</th>
<th>time</th>
<th>seconds</th>
<th>self</th>
<th>total</th>
<th>recursive</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.9</td>
<td>0.0029</td>
<td>0.0029</td>
<td>20</td>
<td>0.1450</td>
<td>0.013</td>
<td>0.013</td>
<td>print</td>
<td></td>
</tr>
<tr>
<td>42.9</td>
<td>0.0058</td>
<td>0.0058</td>
<td>20</td>
<td>0.1450</td>
<td>0.013</td>
<td>0.013</td>
<td>add_vector</td>
<td></td>
</tr>
<tr>
<td>14.3</td>
<td>0.0068</td>
<td>0.0068</td>
<td>2</td>
<td>0.5000</td>
<td>0.000</td>
<td>0.000</td>
<td>main</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mull_by_scalar</td>
<td></td>
</tr>
</tbody>
</table>

Time Stamping

- application instrumentation technique
- create a log buffer and routine
  - call log routine for all interesting events
  - routine stores time and event in a buffer
    - requires a cheap, very high resolution timer
- extract buffer, archive, mine the data
  - time required for particular operations
  - frequency of operations
  - combinations of operations
  - also useful for post-mortem analysis
## Time Stamping

### Dump of simple trace log

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Event</th>
<th>Sub-type</th>
</tr>
</thead>
<tbody>
<tr>
<td>05/11/06</td>
<td>09:02:31</td>
<td>packet_recv</td>
<td>0x20749329</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31</td>
<td>packet_route</td>
<td>0x20749329</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31</td>
<td>wakeup</td>
<td>0x4D8C2042</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31</td>
<td>read_packet</td>
<td>0x3C5CDAD0</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31</td>
<td>sleep</td>
<td>0x4D8C2042</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31</td>
<td>interrupt</td>
<td>0x00000003</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31</td>
<td>dispatch</td>
<td>0x16D324CO</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31</td>
<td>intr_return</td>
<td>0x00000003</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31</td>
<td>check_space</td>
<td>0x2D5F2D40</td>
</tr>
<tr>
<td>05/11/06</td>
<td>09:02:31</td>
<td>packet_recv</td>
<td>0x20749329</td>
</tr>
</tbody>
</table>