Raytheon

Customer Success Is Our Mission

AIR
LAND
SEA
SPACE
CYBER

Expanding Automated Test and Re-Engineering Old Processes

Eric Greene and Jim Knuff
Raytheon Missile Systems
September 19th, 2012

How can we speed up test?

Raytheon Missile Systems

- Expand automation from Preparation ➡ Results Communication
- Reengineer old processes
  - First consider both hardware & software changes for your worst bottlenecks
- Appreciate the synergies of software and hardware technologies
- Anticipate both hardware and software obsolescence
- Understand the savings outside of just direct costs
- Track hardware technologies that facilitate automation
  - Multi-core processors ➡ Rehost tasks from several PCs to one
    - Multi-threading with load leveling
  - Wide user of network connections into discrete test equipment
    - Greater network control and envelopment into automation
  - Wider/faster network pipes ➡ More data exchange and connectivity
  - Test Executive software technologies ➡ More versatile control

Look more broadly and focus maximum fire power

Innovative Applications of Automation

Raytheon Missile Systems

- Test Preparation
  - Control the out gassing of a vacuum chamber
  - Control an IR sensor cool down
- Pre-test (prior to test execution)
  - Perform a short pretest where a few data points are collected, analyzed and evaluated to validate system test readiness and test control
- Test
  - Greater test monitoring and statistics collection via a Test Executive
- Test Support
  - Telemetry control and data monitoring activities
- Post-test
  - Query networked workstations for availability and spread data processing out across all available workstations
  - Combined data tool with data trending, report creation, and email communication into one application

Automation opportunities are still available

Two Examples: Automated Data Processing

Raytheon Missile Systems

- Integrated tool for data processing/visualization/data basing
  - Integrated execution of specialized analytic MATLAB or IDL algorithms
  - Test Executive integration
    - Allow automatic startup immediately after test completion
    - Test results sent back for evaluation prior to next test event
    - Test Executive controlled task sharing across multiple workstations
  - Database
    - Pull in requirements metrics for test performance validation
    - Push out “this test” analytic results and metric performance
    - Pull in past results for integrated product/family trending on reports
  - Standardized report templates with graphs, numeric tables and programmed text plus Tailored user reports (failure, rollup, etc.)
  - Automated Report types based upon specific test or trending results
  - Email test results, alerts or reports to listed leads/engineers/managers

Integrate and automate across common activities

Two Examples: Automated Telemetry Tool

Raytheon Missile Systems

- Our automated testing utilizes Telemetry (TM) data
  - Manual button pushing significantly slows test cycle time
  - Telemetry skill barrier is a major hurdle
  - File maintenance and I/O checking spirals out of control
- A networked based TM services provider
  - Controls all networked based telemetry devices
  - Real-time network TM Data Exchange
  - Data archival to pre-defined network folder
  - Simple service commands
  - Checks storage space
  - Stand-alone GUI

Telemetry System

Search out bottlenecks in the test cycle

Process Re-engineering

Raytheon Missile Systems

- Hidden hardware/software synergies are the key
  - Consider one of our worst Test Cycle problems
    - A computationally intensive process
  - What to do? New hardware, software or both?

File Size	Baseline	Hardware Only	Software Only	Both
350 MB	27:17 min	20:44 min	9:12 min	2:12 min
Cost	--------	$ (~ $8k)	$$$	$$$$
Improvement	--------	24%	66%	92%

  - Test engineers amazed at the results – Test Cycle got a lot faster!
  - Five engineers no longer waiting for 25 extra minutes (~$100s)
  - But it is *Not* just direct cost, consider
    - Lab time, overhead, schedule, management fees, customer time, field conditions, customer satisfaction, market share capture, development time
- Attacking your largest bottlenecks with both hardware and software improvements can lead to great savings
  - Conversely, a hardware or software *only* approach can lead to less gains

Consider the synergies of hardware and software

Software Considerations

Raytheon Missile Systems

- Software re-engineering approaches:
  - Code refactoring: Internal optimization without external interface changes

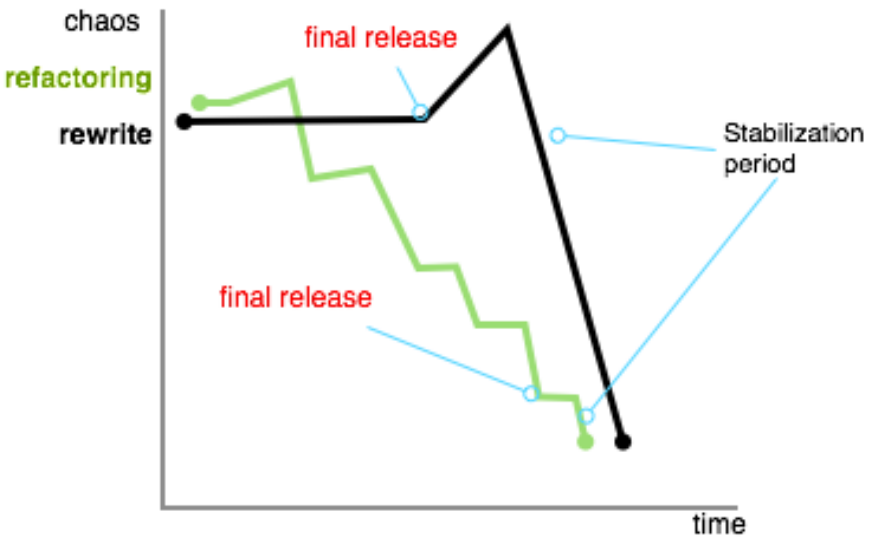
    "A disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior." – Martin Fowler
  - Rewrite – Start fresh, mimic behavior; little reuse
  - Program Transformation - Create a new program with significant reuse
- Software technology upgrades:
  - Added features
    - Expanded libraries: dll/jar/api updates and additions = larger support base
    - Components added: modularity and cohesion provide transportable code
    - More efficient/full featured Frameworks
  - Efficiency improvements in software development programs
    - IDE evolution leads to software development life cycle improvements
  - Increasingly user friendly end point applications

Software considerations are as *Substantial* as hardware

Refactoring

Raytheon Missile Systems

- In an ideal world, Refactoring:
  - Leads to full functionality and solid behavior quicker
  - Releases partial functionality more quickly in minor revisions
  - Rapidly creates stability from chaos
- Rewrites create a longer debugging period prior to release



Effect of refactoring and rewrite over time

- Software solutions of the *past* are often not ideal for the *now*
  - Hardware *and* Software technology shifts can obsolete old software
  - Lack of sufficient documentation/comments leads to indecipherable code
  - Legacy software is often highly dependant on outside sources or components which cannot be updated by in-house programmers
  - Quick “fixes” often lead to issues with long-lived processes

Refactor good code > Rewrite > Refactor “unique” code

Long-Lived Software

Raytheon Missile Systems

- Critical processes are often inherited and last longer than intended
  - Code written for a process will usually last the entire lifetime of the process
  - Development time constraints often impede “perfect” software creation
- Therefore program for code reuse and refactoring ease
  - EMP: Extensibility and Modularization and Portability
    - Complexity should be placed as far as possible from code sections requiring updates, patches, and modification
    - Avoid platform dependent code
    - Anticipate rehosting
  - Document and comment clearly, the designer won’t always be around
  - Use common programming languages with dedicated support and a wide knowledge base
    - A well developed program created in LISP isn’t worth much when your company software engineers do not know LISP
- Well designed software is often selected for reuse

Expecting obsolescence saves future re-development time

Our Computationally Intensive Example

Raytheon Missile Systems

- Process created in 2006 – Quick Development
  - Host Computer: Dual core: 32 bit OS: 2 GB memory
  - Software processes 2~3 GB binary data file
    - *myfile.seekg (offset, direction); myfile.read (memblock, size);* called many times in software, creating a large IO bottleneck
    - Performance limited by both hardware and OS
- Process refactored 2011
  - Upgraded Computer: Dual core: 64 bit OS: 8 GB memory
  - Software can now store the full file in memory
    - *myfile.seekg (0, ios::beg); myfile.read (memblock, size);* called once
    - memblock stored in memory until software no longer requires it
  - Drastically decreased software runtime (25%) by a fairly simple fix
- Final Solution: Refactoring plus high performance Hardware
  - Eight core Intel i7 with Solid State Drives in a RAID 0 Configuration
  - All disk file I/O at RAM speed

Maximum fire power applied to a critical bottleneck

Summary

Raytheon Missile Systems

- Understand the savings outside direct costs
  - Justifies “up front” reengineering expense
- Expand automation outside typical test activities
  - Integrate similar tasks into a common process
  - Leverage networking capabilities of test support hardware
  - Integrate all activities into a common Test Executive
- Reengineer old processes with both hardware & software updates
  - Focus resources on the worst bottlenecks
- Review: search out software/hardware redesign synergies
  - Potentially include extra unexpected savings
- Anticipate obsolescence
  - Design for both hardware and software obsolescence
  - Track technology changes in hardware & software that facilitate re-engineering

Opportunities to reduce test cycle time are plentiful

BACKUP

Historic Trade

Raytheon
Missile Systems

- Over time we have traded
  - Test time reductions
  - For increased test complexity and added test components