STAR WEST
celebrating 15 years
SOFTWARE TESTING ANALYSIS & REVIEW
The Greatest Software Testing Conference on Earth
October 22–26, 2007 • The Disneyland® Hotel

F3

Concurrent Session
Friday 10/26/2007 10:00 AM

JUMP TO:

50 Ways to . . . Improve Test Automation

Presented by:

Mark Fewster,
Grove Consultants

Presented at:
The International Conference on Software Testing Analysis and Review
October 22-26, 2007; Anaheim, CA, USA

SOFTWARE QUALITY ENGINEERING
330 Corporate Way, Suite 300 , Orange Park, FL 32043
888-268-8770 ▪ 904-278-0524 ▪ sqeinfo@sqe.com ▪ www.sqe.com

Mark Fewster

Mark has 20 something years of industrial experience in software testing. Since joining Grove Consultants in 1993, he has provided consultancy and training in software testing, particularly in the application of testing techniques and test automation. He has published papers in respected journals and is a popular speaker at national and international conferences and seminars. Mark is co-author of the book "Software Test Automation" with Dorothy Graham, published by Addison-Wesley. In 2006 he received the Mercury BTO Innovation in Quality Award.

STAR West 2007

Fifty Ways to ... Improve Test Automation

Prepared and presented by

Mark Fewster

Grove Consultants

Llwyncynhwyra, Cwmdu
Llandeilo, SA19 7EW, UK
Tel: +44 1558 685180
email: mark@grove.co.uk

www.grove.co.uk

Key Areas

- planning and management
- scripting techniques
- comparison methods
- pre- and post-processing
- testware architecture
- testware maintenance

Planning and management

- **weaknesses**
  - mixed (or no) views on objective(s)
  - unclear (or no) specific responsibilities
  - subjective measurement
  - vague (or no) development plans
  - autonomous resource
- **improvement areas**
  - define, quantify, agree objectives
  - assign specific responsibilities
  - objectively measure benefits & costs
  - improve automation capabilities
  - become service provider

Management improvements

- **define, quantify, agree objectives**
  - must be appropriate and specific to automation
    - e.g. reduce automated test cost, increase automated test benefit, contribute to testing
    - not e.g. reduce elapsed time, find more defects, improve software quality
- **assign specific responsibilities**
  - separate responsibilities of testing and automating
    - divide one person's time between testing and automating if necessary

Management improvements

- **objectively measure benefits and costs**
  - equivalent manual test effort (EMTE)
    - easy to measure and understand
  - build, failure analysis and maintenance costs
    - major cost factors for automation
- **improve automation capabilities**
  - more flexibility, better reporting, new capabilities
  - greater scope – seek opportunities beyond major tools (e.g. utilities to assist with tester chores)

Further improvement ideas

- **pilot (yes, another one)**
  - small scale (1 – 3 months for 2 or 3 people)
  - not on critical path but able to contribute
    - freedom to experiment important
  - weekly milestones
    - daily targets?
  - specific improvement goals
    - e.g. reduce build cost (explore scripting techs.)
      reduce maintenance cost (explore test design)

Scripting techniques

- **weaknesses**
  - chaotic / disorganised
  - lack of coding standards / guidelines
  - too much test specific scripting (lack of reuse)
  - little or no control over updates
- **improvement areas**
  - consistancy / standards / guidelines
  - reviews to check acceptability
  - appropriate techniques
  - configuration management

Scripting improvements

- **scripting (coding) standards / guidelines**
  - developed by consensus
    - identify selection of good examples, agree what makes them good
      - naming conventions (variable, functions, scripts)
    - measure and check (automated / tool supported)
      - size, complexity, header, comment/code ratio
    - justify standards, templates
- **formally review samples**
  - informally review (buddy check) others

Scripting improvements

- **appropriate techniques**
  - structured scripting, data-driven, keyword-driven
    - perhaps combine techniques
  - common language for defining tests
    - driven by testers, not automators
- **configuration management**
  - at least source code control
  - procedures / processes
  - tool support essential

Comparison methods

- **weaknesses**
  - lack of reuse
  - too few solutions / lack of ingenuity
  - too many different solutions (for same problem)
  - too much work for testers
- **improvement areas**
  - identify definitive comparison req's.
  - be creative, divide complex problems
  - standardise
  - tool support tester tasks

Comparison improvements

- **identify definitive comparison requirements**
  - finite set of output types for each application
- **be creative, divide complex problems**
  - undertake complex comparisons in bits
    - divide output into separate pieces
      - e.g. report header and body
    - compare different aspects separately
      - e.g. key transactions and overall balance
  - use scripting languages and regular expressions
    - e.g. Perl, Python, Ruby, etc.

Comparison improvements

- **standardise**
  - define standard comparisons for each output type
  - implement a comparison process for each one
- **tool support tester tasks**
  - testers need only state which outputs are compared and with which comparison process
    - tools should determine location of files
      - requires consistent testware architecture

Pre- and post-processing

- **weaknesses**
  - not formalised (not recognised)
  - manual set up and clear up prone to human error
  - test specific implementations duplicate build and maintenance effort
  - lots of automated tests but little automated testing
- **improvement areas**
  - recognise and formalise setup and clear up tasks
  - automate setup and clear up tasks
  - encourage reuse
  - automate as many non-execution tasks as practical

Pre & post-processing improvements

- **recognise and formalise setup and clear up tasks**
  - many tasks similar
    - e.g. move, copy, create, delete, convert
- **automate setup and clear up tasks**
  - easy to do
  - easy to make generic

Pre & post-processing improvements

- encourage reuse
- automate as many non-execution tasks as practical
  - including checking

Testware architecture

- **weaknesses**
  - disorganised / scattered testware makes tool support difficult
  - hard to find scripts / data for reuse and maintenance
  - unclear how to name and place new artefacts
  - difficult to track changes and control updates
- **improvement areas**
  - structured and consistent testware organisation
  - organise testware around testers, not test tools
  - clear (documented) naming conventions
  - keep all testware under configuration management

Testware architecture improvements

- **structured and consistent testware organisation**
  - consistent across applications, projects, etc.
  - enables reuse and sharing of tools and methods
  - makes tool support much easier and more effective
- **organise testware around testers, not tools**
  - ergonomics of testware: reduce cost of test work

Testware architecture improvements

- **clear (documented) naming conventions**
  - removes all the guess work
- **keep all testware under configuration management**
  - at least version control but more sophisticated solutions add more value (reduce human error)
    - map versions of testware to versions of software
  - helps with estimation of maintenance effort

Testware maintenance

- **weaknesses**
  - more reactive than proactive
  - maintenance costs regarded as inevitable, and so unchallenged
  - true costs not measured
  - maintenance effort not well supported by tools
- **improvement areas**
  - manage and monitor testware maintenance
  - consider maintenance implications during test design
  - measure maintenance costs and learn
  - provide tool support for maintenance tasks

Testware maintenance improvements

- **manage and monitor testware maintenance**
  - identify most vulnerable tests / applications
    - by type
  - focus improvement effort on maintenance issues
    - spend some time investigating solutions
- **consider maintenance implications during test design**
  - identify software changes with greatest impact
  - implement automation to minimise their impact

Testware maintenance improvements

- **measure maintenance costs and learn**
  - measure costs
    - most frequent / largest maintenance tasks
    - as proportion of EMTE
- **provide tool support for maintenance tasks**
  - for most frequent / time consuming tasks
  - some may be one-off (single use)
    - needs programming skill (right place and time)

Summary

- **many aspects of test automation to consider**
  - management, scripting, processing, architecture, maint.
- **most fundamental pitfalls**
  - objectives and responsibilities
- **test automation is an ongoing process**
  - much to learn, new opportunities to harvest
- **invest in automation, don't stand still**
  - big rewards are possible