Agile Methods
and Software Testing

Paul Gerrard
Technical Director, Systeme Evolutif

Paul is the Technical Director and principal consultant for Systeme Evolutif. He has conducted consultancy and training assignments in all aspects of Software Testing and Quality Assurance. Previously, he has worked as a developer, designer, project manager and consultant for small and large developments using 3 and 4GLs. Paul has engineering degrees from the Universities of Oxford and London, is Co-Programme Chair for the BCS SIG in Software Testing, was a member of the BCS Software Component Test Standard Committee and Former Chair of the IS Examination Board (ISEB) Certification Board for a Tester Qualification whose aim is to establish a certification scheme for testing professionals and training organisations. He is a regular speaker at seminars and conferences in Europe and the US, and won the ‘Best Presentation’ award at EuroSTAR ’95.

Agenda

- Collaborative game of invention and communication
- Agile Manifesto: www.agilealliance.org/
- Some of the increasing number of agile test methods (for more info: www.testing.com/agile/)
  - eXtreme Programming (and testing)
  - Exploratory Testing
  - “Good-Enough” Testing
  - Bug Advocacy
  - Risk Based Testing (I’m working on this one)
- Watch this space…

The struggle for productivity

- We despair that software development productivity only advances a few percent each year
- We don’t worry about
  - Time it takes to make new laws
  - Time it takes to write a book
  - They take as long as they take
- Just like software development?
- Software development is a “resource-limited collaborative game of invention and communication”
- Maybe that’s why we struggle to improve.
“A Collaborative Game of Invention and Communication”

Types of game

- Zero-sum – I win, you lose e.g. tennis, draughts, darts, football
- Non-zero-sum – multiple winners and losers e.g. poker, marathon, hide and seek
- Positional games, where you plan and can trace every move e.g. chess, go, othello
- Non-positional games e.g. snooker, bridge, rock-climbing, football
- These games are all competitive.
Which game is software development?

- Positional?
  - Every move planned, executed, documented in triplicate
  - Documentation reflects both current state and history
- Zero-sum?
  - Developers deliver late, testers are squeezed?
- Competitive?
  - We deliver before the customer changes their mind (regardless of whether it ‘works’)
- Software development is a collaborative, non-zero-sum, non-positional game.

Software development as rock-climbing

- Collaborative
- Goal seeking
- No single route to success
- Team based
- Individuals with talent
- Skill-sensitive
- Training required
- Tools
- Resource-limited
- Planned
- Stressful
- Improvised
- Challenging
- Dangerous
- Fun (?)
Software development is invention and communication

- Requirements are a fuzzy, incomplete, ambiguous set of intents, ambitions and needs
- Deliverable is an executable language, difficult to express and unforgiving of error
- Development is a complex translation process
- Invent props and devices to understand and express the problem to be solved (at all levels)
- Need trails of markers for ourselves and others who come later.

Communication

- Poor communication is the main reason big projects need many more people and fail
  - It'll take 10 good developers six months
  - It'll take 200 people two years
- Excessive documentation in abstract models that are over complex, never finished and out of date anyway are barriers to progress
- Surely, it’s better just to talk face to face?
Cost of poor communication

- Pair programming (pair testing) demonstrably more effective
- Proximity has a huge influence on cost
  - Next desk – I see/hear everything
  - Next room – I see/hear nothing, it costs me five minutes to ask/answer a question
  - Next building – I see/hear nothing, I don’t bother to talk anymore, I write documents.

Diminishing returns

- Time, resource and money are sparse – don’t waste it perfecting products
- Products are ready when they allow you to make the next move
- When documentation becomes drudgery, rather than useful
- When test planning doesn’t generate useful tests anymore
- The “Good Enough” approach.
Levels of maturity

- Level 1 - following
  - people need written instructions to do everything – it’s novel, they are inexperienced
- Level 2 – detaching
  - know the process, knows where it breaks down, knows of alternatives, can adapt them
- Level 3 - fluent
  - process is irrelevant, knowledge is integrated with the task at hand and action required to perform it.

A touch of Zen?

- Kent Beck, when asked about XP and the Capability Maturity Model (CMM):
  1. Do everything as written
  2. Then, experiment with variations in the rules
  3. Eventually, you don’t care if you are doing XP or not
- A higher level of consciousness?
- Pretentious? Dangerous? Or what?
Agile Manifesto

www.agilealliance.com
www.agilealliance.org

Agile Alliance Manifesto

- Individuals and interactions
- Working software
- Customer collaboration
- Responding to change

XP, Adaptive Software Development, Crystal, RSDM, Feature Driven Development, Scrum, XP

We value these MORE

- Processes and tools
- Comprehensive documentation
- Contract negotiation
- Following a plan

We value these
Agile Alliance Principles

- Highest priority is to satisfy the customer through early and continuous delivery of valuable software
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
- Business people and developers must work together daily throughout the project.

Agile Alliance Principles 2

- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation
- Working software is the primary measure of progress
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
Agile Alliance Principles 3

- Continuous attention to technical excellence and good design enhances agility
- Simplicity - the art of maximizing the amount of work not done - is essential
- The best architectures, requirements, and designs emerge from self-organizing teams
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Extreme Programming and Testing
Key attributes of XP

- Code reviews are good, so we’ll review the code all the time (pair programming)
- Testing is good, so we all test all the time (developers unit test, customers system test)
- Integration testing is good, so we’ll integrate and test several times a day (continuous integration)
- Design is good, so it’s part of everyone’s business
- Simplicity is good, so we’ll always strive to use the simplest design possible
- Etc. etc.

Developer testing

- Define and build tests from stated requirements - before coding
  - if the requirement (or the solution) is unclear write some tests - this will flush issues out
  - if there is a situation which the code must deal with, write a test to cover it
  - if you find a problem later, write another test to cover it
  - if you do any redesign (refactoring), write tests to cover the changed functionality.
Developer testing 2

- All code must have tests
- All tests are automated and maintained forever
- After integration, all tests must run 100\% successfully before release
- If a bug is found, more tests are written
- If test stops ‘working’, getting the test to work again is the top priority.

Exploratory Testing
What is Exploratory Testing (ET)?

- Concurrent
  - Product exploration
  - Test design and
  - Test execution
- But that's just error-guessing isn't it?

ETS process

- Briefing - one page manifesto to identify hotspots
- Get familiar with product & target hotspots
- Try extremes, anything you like to break product
- When you find a bug, explore around it
  - Look for patterns, clues for other bugs
  - Log an incident
  - Move onto next interesting area
- When to use?
  - Always? - probably not
  - When normal testing is suspended or you believe more focused tests in an area will be productive.
Good Enough Testing

“Good Enough”

- James Bach* is main advocate of the ‘Good Enough’ view (also Yourdon and others)
- A reaction to compulsive formalism
  - if you aim at perfection, you’ll never succeed
  - your users/customers/businesses live in the real world, why don’t you?
  - compromise is inevitable, don’t kid yourself
  - guilt and fear should not be part of the process.

* James Bach, “Good Enough Quality: Beyond the Buzzword”, Computer, August 1997
also STAR ’97 presentation, “Good Enough Testing”.

© 2002 Systeme Evolutif Limited
Version 1.0 © 2002 Systeme Evolutif Ltd
“Good Enough” means:

1. \textit{X} has sufficient benefits
2. \textit{X} has no critical problems
3. Benefits of \textit{X} sufficiently outweigh problems
4. \textit{In the present situation}, and \textit{all things considered}, improving \textit{X} would cause more harm than good.

All the above must apply

Contribution of testing to the release decision

- Have sufficient benefits been delivered?
  - tests must \textit{at least} demonstrate that features providing benefits are delivered completely
- Are there any critical problems?
  - test records must show that any critical problems have been corrected, re-tested, regression tested
- Is our testing Good Enough?
  - have we provided sufficient evidence to be confident in our assessment?
Bug Advocacy

- Bug (incident) reports are your main work product
- The best tester gets most bugs fixed (not just found)
- A bug report is a tool for getting bugs fixed
- Sales techniques
  - Motivate the buyer (get the developer to fix)
  - Overcome objections (argue against objections).
Bug advocacy 2

- Motivate the developer
  - It’s a dangerous, expensive, embarrassing, expensive bug that is easy to fix!
- Overcome objections
  - Make it easy to reproduce - research it further
  - It’s a bug, not a feature
  - Make the bug report perfectly clear
  - And so on.

Risk-Based Testing
(I’m working on this one…)
Proposed test objectives

- To provide evidence (and confidence) that
  - The benefits delivered are acceptably high
  - The risk of failure is acceptably low
- To report progress by identifying:
  - Risks addressed
  - Benefits delivered
- Stakeholders want to know about benefits delivered (and the benefits under threat)
- Project management want to know about the residual risks that block the benefits.

Key steps to RBT

- Risk Assessment
- Master Test Plan
- Test Objectives
- Techniques/Coverage
- Test Planning
- Test Specification
- Test Scripts
- Test Preparation
- Test Execution
- Test Incidents, Results, Logs.
- Technical involvement (developers, designers, business analysts)
- Stakeholder involvement (management, customers etc.)
Risk-based test reporting

Progress through the test plan

Residual Risks

all risks 'open' at the start

Planned end

today

Residual risks of releasing TODAY

Benefit & objectives based test reporting

Benefits available for release
Status of agile (testing) methods

- Multiple lightweight development methods
  - Trend from predictive to adaptive
  - No overarching theory just a set of principles
  - All promoted by influential individuals not corporations
- Same goes for agile test methods
  - Collection of disintegrated techniques
  - Few (if any) links to the development methods
  - Little guidance on when to use, how to combine/choose between them, how to fit into real projects
- Increasing attention being paid to them…

Watch this space.
Agile Methods
and Software Testing

Paul Gerrard
paulg@evolutif.co.uk

www.evolutif.co.uk
www.riskbasedtesting.com