Email mike@testandverification.com
phone +447796307958

Dealing with testing debt in an agile world

Mike Bartley,
CEO Test and Verification Solutions Ltd.

Agenda

- What is testing debt?
- Spotting and measuring testing debt
- Fixing testing debt
  - Avoidance
  - Controlling scope
  - Reducing test times
  - Legacy tests and removing tests
- Avoiding technical debt
- Learning from best practice
- Some key takeaways

Understanding Debt

- Definition – Technical Debt
  - The implied cost of additional rework caused by choosing an easy solution now instead of using a better approach that would take longer
- Why is it important?
  - Debt decreases productivity
  - Unable to run in a sustainable fashion
  - Unpredictable schedules
  - Reduced software quality
- Testing Debt
  - Technical debt related specifically to testing

Evaluating testing debt

- Ways to “smell” testing debt
  - Is it possible to select tests for a particular feature?
  - Is it possible to measure what a test does?
  - Is it possible to assess whether a test is still effective or is now outdated?
  - What happens when a test fail is not understood?
  - What is the run history of the test?
  - Is it possible to identify which parts of the code are no long effectively tested?
- Poorly written tests contribute to testing debt

Measuring testing debt #1: Metrics

- Metrics Rules
  - Ensuring our metrics are effective
- Code metrics
- Test metrics

Measuring testing debt #2: Metrics

- Process Metrics
- Progress Metrics
- Effectiveness Metrics
- Efficiency Metrics

Fixing testing debt #1: Avoidance

- First – stop adding to technical debt
- Throw away tests that are no longer effective
- Reduce test suites that take too long
- Reduce testing debt with new tests

Fixing testing debt #2: Controlling test scope

- The single biggest reduction in testing comes from reducing combinations of:
  - releases
  - configurations
  - target platforms that need to be tested

- There are a number of ways of approaching this problem

Fixing testing debt #3: Reducing total test times

- Any testing, except release testing, should be time boxed
- Reducing the number of software versions, configurations and targets
- Improved test selection
- Split up big bucket test sets
- Replacing legacy tests that are ineffective or inefficient.
- Removing testing bottlenecks
- Minimise manual testing

Testing Stages

PHASE	FREQUENCY	SCOPE	TEST SCOPE	COMPLETE	AUTOMATION
Unit	Continuous	Unit	Optional testing of unit development code	Optional	Optional
Unit pre-release	Pre-release	Unit	Functional testing of unit release candidate, static analysis and code review	Time-boxed (< 10 mins.)	YES
Unit and Integration	Nightly	Code base	Code base functional testing of release candidate(s)	Time-boxed (< 12 hours)	YES
Stress	Weekends	Code base	Code base fuzz testing on trunk	Time-boxed (< 60 hours)	YES
Exploratory	Not repeated	Code base	Product non-functional testing on trunk eg: perf., useability, …	N/A	Optional
Product Release	Pre-release	Product	Product acceptance testing of release candidate	YES	YES

Fixing testing debt #4: Removing tests

- Effective test reduction requires
  - clear policies
  - good information that allow teams to make these hard decisions
- Test suite reduction requires the collection of metrics on individual tests
- This data can normally captured in the test management tool

Fixing testing debt #5: Legacy Test Issues

- Over time tests are added but rarely removed
- Understanding of the old tests is normally poor
- The software is typically required to support an increasing number of configurations and additional target platforms.
- More releases of the software are created
- Tests may require access to scarce resources that create a bottleneck.

Avoiding testing debt: A strategic approach

- Testing Strategy
- Test Development
- Workflow

Testing Debt: Best practise research

- Based on paper-based research on
  - Google
  - Spotify
  - Facebook
  - Microsoft
  - Ericsson
  - Unity
  - Panda Strike
- And consultancy at a number of companies
  - 3rd party independent consultancy to review and reduce technical debt.

The “Best Practise”: organisation

- Agile
  - process, organisation, culture
  - 'trust the developers + hold them accountable'
- Scalable
  - process, organisation
- Strong investment in infrastructure
  - including tools and continuous improvement

The “Best Practise”: process

- Clarity
- Avoid formal handoff and synchronised releases
- Automate all repetitive testing
- Include additional types of product testing
- Prevent an accumulation of 'testing debt'
- Metrics
- Release management
- Single code base with all changes subject to code review

What (I hope) we have covered

- What is testing debt?
- Spotting and measuring testing debt
- Fixing Testing debt
  - Avoidance
  - Controlling scope
  - Reducing test times
  - Legacy tests and removing tests
- Taking a strategic approach “Avoiding technical debt”
- Learning from Best practice

Some things to try tomorrow

- Time box your testing
- Reduce the scope of your testing
- Are your developers doing enough testing?
- Identify your worst test bottleneck
- Identify a legacy test to remove