Introduction of Developer Testing in an Embedded Environment

Lakshmankumar Mukkavilli
Cisco Systems, Inc.

Topics

- Acronyms
- What is Developer Testing
- Organization
- Background
- Action
- Tool for UT
- Pilots
- Criteria for the Pilots
- Results
- Key Factors for Success
- Status

Acronyms

- DT – Developer Testing
- UT – Unit Testing (used interchangeably with developer testing)
- MBT – Model Based Testing
- CFD – Customer Found Defect
- ROI – Return on Investment

What is Developer Testing

Creation of whitebox tests by the development engineers with a view to a reduction in the defects found post-development.

Organization

- 100’s of development and test engineers
- Very large embedded software
- Major revenue generator
- Development spread over multiple Business Units
- Testers perform blackbox testing and create test scripts. Typically testers start testing a feature after handoff from development.

Background

- Root cause analysis of the CFD 's indicated that a significant % of the defects were UT escapes.
- Significant % of defects found by the test teams should have been caught during developer testing.
- Since test teams spend good part of their effort on basic bugs, they did not have much time for other defects.
- Developer testing involved basic blackbox testing.

Background (Continued)

- No serious whitebox testing
- Big holes in the coverage that could not be filled by just blackbox testing.

Action

- Managers and senior engineers from development, tools and process groups got together and created a set of guidelines for the developers. Mandated Static Analysis (SA), Reviews and UT .
- UT guidelines included a set whitebox testing techniques applicable to our software.
- SA and reviews were adopted. UT adoption was close to zero.

Causes for Lack of Adoption

- Slow builds. Whitebox tests require several builds.
- No budgeted time for UT
- Notion of rigorous whitebox testing was novel. There were hardly any examples to emulate.
- No standard tool
- No evidence of value of UT
- Feeling that testing was the job of the testers

Tool for UT

- Several external tools were evaluated and were found to be inadequate for our needs. One of them was subjected to trials by various development engineers. Feedback was not positive.
- We developed a tool internally to meet the needs of our developers to cover various testing strategies indentified in the guidelines for UT. Some of the salient features are
  - Innovative technique to dramatically reduce build times
  - Support for lightweight MBT
  - Software Fault Injection
  - Support for automation

Tool for UT(Continued)

- Robustness test generation
  - Test generator is included in the executable. Contrast this with the tools where a test is generated on the host and shipped to the target.
- Test/subtests organization
- Low memory footprint
- Support for scalability testing
- Profiling/Tracing
- Code coverage
- Memory leak detection
- Library of functions
- Features to help test code modularity and reuse

Tool for UT(Continued)

- Quality of the tool is an important. Aim is zero CFD’s.
- Close liason with the development groups.
- Goal of the tool is to get minimal input from the users and provide maximum functionality.
- Created training materials
- Built a large collection of working examples to cover various test strategies.
- High quality of support. In many cases the initial tests were created by the tool team.

Pilots

- Two sets of candidates for pilots.
  - First set of candidates was interested in evaluation and possible adoption of UT .
  - The second set of candidates came from a major code refactoring effort. Worked jointly with this team to make whitebox testing by development engineers a standard practice.
- Provided training in
    - Using the tool effectively.
    - Various techniques for whitebox testing
- In almost all cases development engineers were writing whitebox tests for the first time.

Criteria for the Pilots

- Reducing development escapes.
- Precision/reproducibility of the problem reports created.
- Time to resolve the problem reports.
- Cost of finding the defects. Norm for the test groups is three weeks/defect.

Results(Continued)

Project	Weeks	Defects	Comments
Project 1	40	125	Software fault injection was a key contributor. The feature is released and there are no high/medium severity bugs against the feature.
Project 2	6	59	51 from Light-weight MBT and 8 from API Robustness testing
Project 3	8	18	6 from API Robustness, 4 from concurrency testing

Results(Continued)

Project	Weeks	Defects	Comments
Project 4	3	9	
Project 5	4	10	4 from CLI Robustness, 4 from light-weight MBT
Project 6	10	47	12 from software fault injection

Key Factors for Success

- The tool
  - Integrated into developers workflow
  - Feature richness
  - Quality and reliability
  - Support for rapid incremental builds
- Buy in from the management of the development engineering
- Hands-on workshops
- Very high ROI

Status

- Developer Testing is considered valuable
- Steady growth in adoption
- Effort for UT is included in the schedules
- Tool is being enhanced
  - Automation (whitebox test regression runs)
  - Newer test strategies
- Some test teams are trying to take advantage of the whitebox tests created by the developers. Early results indicate a positive synergy.
- With the loss of easy defects, test groups are trying to explore newer techniques for defect finding.

Status(Continued)

- Advanced the state of testing. Some techniques like lightweight MBT, Software fault injection, robustness testing have become widely used.