

Design for Testability

TestNet
Najaarsevenement



Bryan Bakker

October 11th 2010

Contents

- Sioux
- Intro
- What is Design for Testability (DfT)?
- Test Automation
- Design Rules
- Pre-requisites
- Watch out
- Conclusion

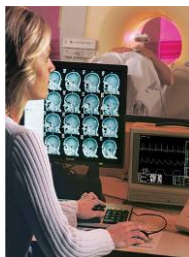
- Examples from embedded/technical domain but concepts also hold for office domain
- Scope: integration and system testing

About Bryan Bakker



- Test Architect
- Certifications: ISTQB, TMap, Prince2
- Member of ISTQB Expert Level on Test Automation
- Accredited tutor of ISTQB Foundation
- Domains: medical systems, professional security systems, semi-industry, electron microscopy
- Specialties: test automation, integration testing, design for testability, reliability testing

About Sioux



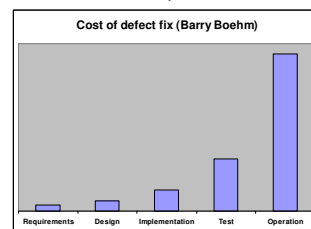
- Device including HDD
- During test phase no serious HDD issues
- After release: HDD failures in field
 - Customers return units (NFF)
 - False alarms!
- SW not robust against HDD imperfections
- Firmware upgrade needed to prevent more returns
- Could this have been prevented?

- Simulate HDD imperfections
 - find defects during development/test
 - more robust SW/System



- Definition:
 - Take testing into account during design/architecture definition*

- Main goals:
 - More efficient testing (find defects earlier, automation)
 - Increase coverage of testing (manual and automatic, make it possible to detect other problems)
 - Enable automatic testing



- Think of:
 - Testing without HW (not finished or expensive)
 - Simulate environment (for automatic testing or unfeasible environment)
 - Replace mechanical switches/buttons (test automation)
 - Support for test automation
 - Negative testing (failures from HW or environment)
 - Support for test/sw engineers (diagnosis)
 - Logging/Tracing
 - Test components in isolation (modular architecture)
 - Support for integration testing (test for messages)
 - Test without UI
 - Reliability/Profile testing: record user actions and replay
- By
 - Visibility
 - Control

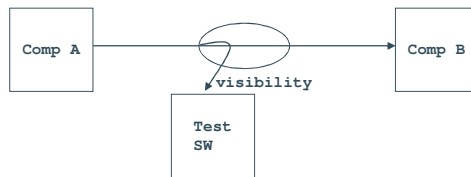
- Visibility
 - Usually: subset of system information is shown to end-user
 - DfT: interface defined to extract info from system
 - Also for “hidden” info



- Normal transfer of information

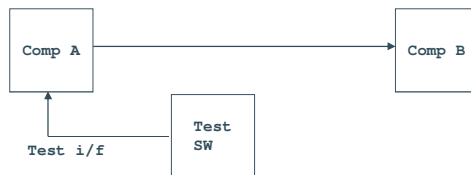


- Offer information to test software:



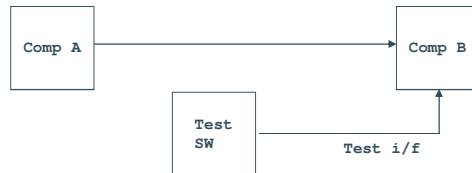
- Define test interface (test hook) to inspect info from Comp A
- On Comp A or Comp B or in between?

- Test interface on Comp A:



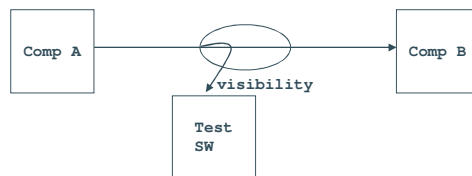
- Comp A is aware of interface

- Test interface on Comp B:



- Comp B is aware of interface

- Use wrapper or message queue inspector (e.g. VxWorks)



- Comp A and B are unaware of interface
- But not everything is sent to other components...
- Where to interface is design decision

- Extract all kinds of system information
 - Temperature
 - #Images passing through image chain
 - Recording speed of recorder
 - Mechanical movements verification
 - Inspect messages (for integration tests)
 - State information (of system or components)
 - Logging (better inspection/analysis, tool support)
 - Resource usage (cpu, memory, network)
 - ...

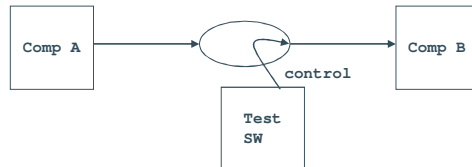
- Control
 - Usually: system controlled by system interfaces like user, environment, network, etc.
 - DfT: interface defined to control the system



- Normal transfer of information



- Information altered by test software:



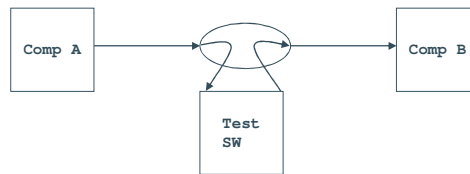
- Define test interface to control Comp B
 - set information
 - ignore control from Comp A (optionally)

- Trigger all kinds of system actions
 - Push buttons (UI, mechanical)
 - Set configurations
 - Simulate events (motion events, alarms, hot temps)
 - Mechanical movements
 - Simulate HW failures/imperfections
 - ...

- Normal transfer of information



- Information retrieved and altered by test software:

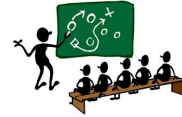


- Define get and set test interfaces

- Control used to trigger actions
 - Best practice: as “low” as possible in the architecture
 - close to hardware
 - as much coverage as possible
 - trade-off between costs and coverage
 - Possible to test below the UI
 - UI is volatile (except “mechanical UI”)
- Visibility used to verify expected result
 - Best practice: use logfile (also evidence) or internal system information
 - Avoid UI information (volatile)



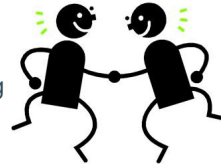
- **State visibility:**
 - Every component stores state information
 - In one dedicated component
 - Testcases can get this information
 - Possibility: with one key-press → dump the complete system information (for defect analysis)
 - Not to be used internally by system (no information hiding)
- **State machines trace/log state transitions**
 - “easily” test the state machines with state-transition testing
 - Determine coverage of testcases (n-switch coverage)



- **Communication between each set of components visible via interfaces (in tracing)**
 - Default functionality in VxWorks
 - Communication can also be altered
 - Used for integration testing
- **All user actions are logged, and can be “replayed”**
 - Input for profile tests (software reliability engineering)
 - Records error-guessing/exploratory tests for reproducibility
- **Failures in HW to be simulated via (test i/f in) drivers**
- **Most projects start with: logging conventions**

Pre-requisites

- Early involvement of test discipline
- Influence on architecture/design
 - By (test) architect
 - Architecture must support effective testing
- Test requirements
 - Functionality needed in the product to support testing
 - Real requirements, need priority
 - Implementation available on time
- Test interfaces
 - Are deliverables of project
 - Supported interfaces, thus maintained
 - Used for automatic tests
- Test req/interfaces become part of the product
 - Test functionality grows into supported functionality of the product (Excel, X Rays)
- Management commitment (DfT is an investment)



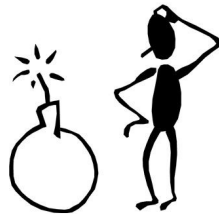
Watch out

1. Disable test functionality in release versions?
 - Like logging, tracing, test functions
 - Different version, will behave differently
 - Performance
 - Issues in release version not reproducible in development version
 - Test functionality may still be needed
 - Service/diagnostics/factory
 - Problem analysis in the field
2. Testing via test interfaces → not the real thing
 - Customer/environment uses different interfaces
 - Decide where to interface (coverage ↔ cost)

3. Beware: **Probe Effect**

- “unintended alteration in system behavior caused by measuring that system” (wikipedia).

Be ware of these effects!



- Design for Testability
 - More efficient testing
 - Increase coverage of testing
 - Enable automatic testing
- Visibility & Control
- Part of design/architecture
- Nothing new! But hardly practiced in a structured way
- Beware: different in real world!





Source of your development.



www.sioux.eu



bryan.bakker@sioux.eu



+31 (0)40 26 77 100