Principles
Software Testing

Zsolt Tóth
University of Miskolc

2018
Table of Contents

1. Software Quality
   - Quality Gates
   - Dependability Properties
   - Code Metrics

2. Testing Principles
Software Quality

- Various project members
  - developers
  - managers
  - customers
- Vague definition
  - Quick
  - Well designed
  - Easy-to-use
  - Logical design
- Opposite requirements
  - Fast development
  - Administration
    - well-designed
  - Programming

User
- Nice GUI
- Logical layout
- Usability

Management
- Administration
- Documentation
- Workflow

Developer
- Well structures
- Extendable
- Comments
External Quality Gates

- Directly visible for customer or users
  - Reliability
  - Loading time
  - Usability
  - Performance

- User’s judgment and opinion

- Complex requirements

- Challenges
  - Clear definition
  - Various platforms and environments

1. Model files should be opened quickly.
2. A model has to be loaded within 5 sec.
3. This computer should load any model within 5 sec.
4. This computer has to open this model in 5 sec.
Internal Quality Gates

- Not visible directly for users and customers.
  - Maintainability
  - Documentation
  - Software design
  - Re usability
  - Transparent management

- Affect on External Quality Gates
- Increase Costs
  - Pressure from market

- Good design
  - Takes time
  - Experienced developers
  - Debate technology choices
  - Require documentation
  - Increase re usability
  - Facilitates further development

- Transparent Management
  - Administration
  - Workflow
  - Slower development
  - Leads to Bureaucracy
  - Managers loves it
  - Calm down the customers
Table of Contents

1 Software Quality
   - Quality Gates
   - Dependability Properties
   - Code Metrics

2 Testing Principles
Dependability

- Collective term
  - Correctness
  - Reliability
  - Availability
  - Mean Time Between Failures
  - Safety
  - Robustness

- Each property tested separately
- Precise definitions required
Correctness

- System meets with specification
- All functionalities are checked
- Possible outcome
  - Success
  - Failure
- Correct \(\equiv\) all functionalities are tested successfully.
  - Imprecise specification
  - Unachievable
Reliability

- Statistical approximation of Correctness
- 100% Reliability $\approx$ Correctness
- Probability of the function works properly.
- Functions behavior
  - Good
  - Wrong
Availability

- Amount of time when the system can be used
- Typical problems
  - Network Error
  - Server overloading (Black Friday)
  - Hardware Error
  - Physical distances
- Given as $[0, 1]$ or sec / year
- Availability $= \frac{T_{SystemWorking}}{T_{Total}}$

Planned Shut Downs
- Maintenance
- Server Upgrade
- Regular
- Improve users trust

Random Shut Downs
- Unplanned
- Error
  - Software
  - Hardware (our)
  - Network (provider, 3rd party)
  - Deny of Service
Mean Time Between Failures

- Availability is rough property
- Different occasions of error
- 1 hour $\neq 6\times 10$ min
- $MTBF = \sum \frac{|T_{erroroccured} - T_{systemstarted}|}{N_{error}}$
- Planned shut downs are good.
- Stochastic behavior decrease user’s satisfaction
### Example

<table>
<thead>
<tr>
<th>Time</th>
<th>Note</th>
<th>Time</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:05</td>
<td>Stack Overflow</td>
<td>10:13</td>
<td>Stack Overflow</td>
</tr>
<tr>
<td>0:15</td>
<td>OK</td>
<td>10:18</td>
<td>OK</td>
</tr>
<tr>
<td>3:27</td>
<td>Router down</td>
<td>11:40</td>
<td>Runtime Exception</td>
</tr>
<tr>
<td>3:52</td>
<td>Router Up</td>
<td>12:00</td>
<td>OK</td>
</tr>
</tbody>
</table>

**Availability**

\[
\text{Availability} = \frac{660}{720} = \frac{11}{12}
\]

**MTBF**

\[
\text{MTBF} = \frac{5 + 192 + 381 + 82}{4} = \frac{660}{4} = 165 \text{ min}
\]
Safety

- Random undesired behavior
- Hazard
  - Planned during design
  - Prepare the system for them!
- Ability to handle hazards
- Definition of hazards is required
- Without hazards, safety is not defined
- Critical for some kind of software systems
  - Healthcare
  - Finance
  - Transportation
- Less important
  - Word – auto save
  - Chainsaw - two hands required to turn on
Robustness

- Unexpected events could occur with correct systems.
  - Disk is full
  - Increased usage (DoS)
  - Blackout
  - etc.

- Robustness
  - Huge change in environment has small effect on systems behavior.
  - Undetected error for users
  - Not just hazards.

- More general than safety

Solutions
- Lode balancing
- Replication
- Geo location

Challenges
- Synchronization
- Costs are skyrocketing
Code Metrics

- Internal Quality Gate
- Analysis of Source Code
- Static
- Refactoring
- Risk detection
- State of the Project

- Aggregated data
- Statistical description
- Mathematical methods
- Automatic
- Shows
  - hidden errors
  - poor design
Code Metrics

Metrics
- Cyclomatic Complexity
- Block Depth
- Depth of Inheritance
- Class Coupling
- Lines of Code
  - per Method
  - per Class
- Number of Constructors per Type
- Number of Methods
- Abstractness
- Comments Ratio
- Efferent couplings
Cyclomatic Complexity

- Structural complexity
- Low value ✓
- Instruction graph
- Complex instruction flow
  - Difficult testing
  - Costly maintenance
  - Hard-to-understand

\[ M = E - N + 2P \]

- \( M \) complexity
- \( E \) edges
- \( N \) nodes
- \( P \) parts
Table of Contents

1 Software Quality
   • Quality Gates
   • Dependability Properties
   • Code Metrics

2 Testing Principles
Principles

- Theoretical background
- General solutions
- Bases of techniques
- Important during the design of tests

Cost of fixing bugs and errors

1. Syntactical Error
2. Unit Tests
3. Integration Test
4. System Test
5. Acceptance Test
Sensitivity

• Developers makes faults.
• Faulty system has wrong functions.
• "Constant fault is better than random one."
  
  constant Occur always when certain criteria is fulfilled.
  random Occurrence depends on unknown factors.

• Constant fault can be fixed easily.
• Unit Test
• Random faults are difficult to fix because
  ▶ Special environment
  ▶ unique configuration
  ▶ cause is hard to identify
I can hotkey alt to anything but the function keys. I can do \texttt{alt+f1, f2, f3} but when i get to \texttt{f4} it \texttt{crashes my game EVERY TIME!} Its annoying because I have to reset all my hotkeys cuz a game crash doesnt save them. Please fix soon!

http://www.dslreports.com/forum/r21448800-Epic-Fail-with-Hotkeys
Redundancy

- Software components contain an entire part of other components
- Allows fault detection or correction (Information Theory)
- Static type checking
- Java exception definition
Constraints

- Complex problems
- Complex property substituted by more simple properties.
- Transactions Serialization
  - Complex well-studied task
  - Own method is costly and useless
  - Use existing methods (see Database Management Systems)
- Decision during design
- Hard to modify later.
Partitioning

- Divide and Conquer
- Split complex problems into simple tasks

Testing Levels
- Unit
- Component
- Integration
- Subsystem
- System

Analysis
- Properties examined separately
- Software has the required functions?
Visibility

- Measuring progress
- Goals and current state
- Important for managers
- State of
  - development
  - components, subsystems
  - testing
  - entire project
- Observing
  - Data collection
  - HTTP, SMTP, ... protocols
  - Decrease performance
  - Facilitate testing and bug fixing
Feedback

Sources
- Users
- Test results

Improve
- product
- development process
- developers