Executing Tests

Bojan Cukic

Department of Computer Science and Electrical Engineering
West Virginia University
Some questions first

In our parking garage workshop, we chose an automated system that parks the cars without a human. The safety subsystem requires high reliability to prevent loss of life, say 0.99999. But, a glitch in the user interface is not severe. We don't care as much if, say, it truncates the attendant's name if it is too long. So, we assign a reliability of 0.98 to that class.

So, I'm talking to my customer whom I'm implementing this for and tell him there's a 2% chance the system will fail after parking 1000 cars. That sounds misleading. My customer is going to think that 20 out of every 1000 cars is going to bump into something, get lost, or injure somebody. I guess my point is that the overall system reliability seems to be less important as looking at the reliability of the classes.

You mentioned that we should start with a system reliability and then from there study the sub systems if we wish to have reliabilities for the sub systems, correct? If so, I don't understand how to get that total system reliability without first doing the subsystems or first doing the severity classes. I guess I need to read Chapter 3 again.
For systems which have drastically different failure severities, it is customary to express reliability for each severity class.
I think I'm confused about converting between Failure intensity (FI) and reliability (R). Suppose we have a goal of a system that processes on average 20,800 cars between failures. In terms of 1000 cars processed as our natural unit, this is 1000/20,800 = 0.048 failures/(1000_cars_processed). So, our failure intensity rate is 0.048, right?

Now I want to calculate the reliability which is the probability that the system will function without failure for the natural unit of 1000 cars. Is this right so far?

Well, the probability of failure for 1000 cars is 0.048, so the probability of success is simply 1 - 0.048 = 0.952. Correct?

If the above is correct, what, then, is the equation $R = \exp(-(\lambda)t)$ for (Equation 3.6 on page 171)? What is $\lambda$ and what is $t$? What is $R$ in this equation, for that matter?
Answer

- Failure intensity (FI): \(1/20,800\) parked cars
  \[= 0.048/1000\ \text{parked cars} = 0.000048/\text{parked car}\]
- Probability of failure: \(1-0.000048=0.9999519\)
- More generally, reliability \(R\) expressed as
  \[R = e^{-\lambda t}\]
  where \(\lambda\) is failure intensity, \(t\) is \# natural time units.
- \(\lambda t < 0.05 \rightarrow R=1-\lambda t\) (the same number as calculated above)
- Converting \(R\) into \(\lambda\) can be done as follows:
  \[\lambda = \frac{-\ln R}{t}, \text{ or } \lambda \approx \frac{(1-R)}{t}\]  
  (if \(R > 0.95\))
Executing Tests

Bojan Cukic

Department of Computer Science and Electrical Engineering
West Virginia University
So far

Preparing for tests included

- Concepts such as RUN, TEST CASE, INPUT VARIABLES (DIRECT and INDIRECT)…

Procedures

- Estimating the number of new tests
- Allocating tests throughout the system
- Allocating new tests to new operations (features)
- Specifying test cases
- Preparing test procedures (adjusting operational profiles, etc.)
- Test automation
The big picture

List Associated Systems

- Define Necessary Reliability
- Develop Operational Profiles
- Prepare for Test

Requirements and architecture
Design and Implementation

Test & Validation

Execute Tests

& Apply Failure Data to Guide Decisions
Concepts

- Feature test
  - Execute all new test cases of the release, independent of each other
  - Interactions and effects of the field environment minimized.
- Purpose is to identify functional features resulting from execution of test cases by themselves.
Load test.
- Execute all valid test cases from all releases together.
- Full interactions and all the effects of the field environment.

Purpose to identify failures from:
- Interactions among test cases.
- Overloading of and queuing for resources.
- Database or operating system degradation.
Regression test
- Execute a subset of all test cases of all releases at each system build with significant change independent of each other.
- Interactions and field effects minimized.

Test subsets contain:
- Tests for rare critical conditions.
- Certain number of test cases for non-critical (non-rare) conditions, selected randomly.

Purpose is to reveal functional failures caused by faults induced by program changes.
Executing Tests

- Use test cases and test procedures developed in the previous phase.

- Test execution phase involves:
  - Allocating test time.
  - Invoking tests.
  - Identifying failures that occur.
    - This information will be used in guiding tests and making decisions.
Allocating test time

- Measure test time in hours. For multiple configurations, Test_Time > Real_Time
- Proceed with allocation in 3 steps
  1. Among the systems to be tested.
  2. Among feature, regression, load test for each system (in reliability growth phase).
  3. Among operational modes for each system in load test.
- Load testing usually equals certification testing.
System allocation

1. Allocate test according to the estimated risk.
2. For the other systems (and remaining time), divide test time in same proportions as the numbers of new tests were assigned.
   - Reuse of previous allocations, based on operational distributions and ‘newness’.
   - Allow enough time in feature test for all the new test cases for the release + regression tests.
   - The rest of the time goes to load testing.
Example

Planned testing period 320 [h]
- 40[h] allocated to the testing of the “supersystem”, based on its criticality.
- System has 2 components, with test distributions of 0.71 and 0.29.
- Then, component 1 (Product) receives 200 [h], component 2 (OS) receives 80 [h] for testing.

<table>
<thead>
<tr>
<th>Operational Mode</th>
<th>Proportion of Calls</th>
<th>Supersystem</th>
<th>Product</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak hours</td>
<td>0.1</td>
<td>4</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>Prime hours</td>
<td>0.7</td>
<td>28</td>
<td>140</td>
<td>56</td>
</tr>
<tr>
<td>Off Hours</td>
<td>0.2</td>
<td>8</td>
<td>40</td>
<td>16</td>
</tr>
</tbody>
</table>
Invoking tests

- SRE starts after the components have been assembled into the *system*, so features can be tested.
  - Assumes unit testing completed.

- **Recommended sequence of system level tests**
  - 1. Acquired components
  - 2. Product and variations
    - 2a) Feature tests
    - 2b) Load tests
  - 3. Supersystem
Invoking Tests

Feature tests

- Select tests in random order from the set of new test cases for the given release.
- Invoke each test case after previously invoked test case is complete (avoid interaction).
- Provide setup and cleanup.
- Do not replace test case after execution
  - Execute each test case once.
Invoking Tests (2)

Load tests

- Operational profile based.
  - The number of tests run based on the time available.
- Fault density likely to be similar to the field operation.
- Select load tests with replacement (repetition OK)
  - Detect multiple faults possibly associated with an operation.
  - Yet, each RUN will be different.
- Repeat a run if you need to:
  - Better investigate fault behavior.
  - Verify fault removal.
- Random test case and operation selection usual.
Invoking Tests (2)

- Regression tests
  - Invoke each test case after previously invoked test case complete.
  - Choose a subset of test cases for each build.
    - All valid critical cases.
    - Randomly selected non-critical test cases.
  - Select test cases randomly.
  - Provide setup and cleanup.
  - No replacement of test cases.
Occurrence probabilities for selecting operations are constant. BUT, constant probability does not mean constant order for testing.

\[ P(A) = 0.7, \ P(B) = 0.3 \]

2 sequences: ABAABAAABA and AAAAAAABBB

<table>
<thead>
<tr>
<th>Operation</th>
<th>Ex. 1, P(A)</th>
<th>Ex. 2, P(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0.67</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0.6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0.67</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0.71</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0.75</td>
<td>0.88</td>
</tr>
<tr>
<td>9</td>
<td>0.67</td>
<td>0.78</td>
</tr>
<tr>
<td>10</td>
<td>0.7</td>
<td>0.7</td>
</tr>
</tbody>
</table>
Identification of system failures

- Involves
  - Analyze the test output for deviations.
  - Determine deviations which are failures.
  - Establish WHEN failures occurred.
  - Assign failure severity classes, to be used for priority assignment in fault resolution.
Analyzing Test Outputs

- Deviation: departure from the expected behavior.
- Tools exist for the analysis of:
  - Inter process communication failures.
  - Illegal memory references.
  - Deviant return codes.
  - Deadlocks.
  - Crashes, hangs.
Analyzing outputs

- Assertions can help in the analysis.
- Time deviations can be counted, depending on the nature of the application.
- **DO NOT** count cascaded deviations as multiple problems.
  - Count a cascade as a single deviation.
Are deviations failures?

- Higher severity failures usually easier to observe.
- Deviations in fault tolerant systems may not be failures.
- Violation of written or nonwritten requirements!
  - A complaint from the user community means it IS a fault.
- Record hardware and personnel failures too.
  - Help in system level problem resolution.
When did a failure occur?

- Use common reference unit chosen for failure intensities.
- If average load over processing time varies:
  - Take execution time measurements (better characterize failure inducing stress).
  - Convert execution time to operating time by dividing by average (over system life) ratio of execution time to operating time.

<table>
<thead>
<tr>
<th>Failure</th>
<th>Exec. Time</th>
<th>Utilization</th>
<th>Adjusted Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
<td>0.4</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>0.6</td>
<td>0.4</td>
<td>1.5</td>
</tr>
<tr>
<td>3</td>
<td>1.2</td>
<td>0.4</td>
<td>3</td>
</tr>
</tbody>
</table>