

## **Towards Curbing Recidivism in Test Case Writing: The Social Model**

By Dave Gass

During the course of a software project at the functional test case-writing level, there are often reviews of the writing for accuracy, functional coverage, and adherence to standardization\style.

Unfortunately, even in more structured environments, the review process is frequently undertaken in an adhoc fashion without tracking and does not ferret out systemic or writer-based recidivism.

### Typical Review Areas of Focus

**Accuracy** - Addresses whether or not the tests are broken as a result of: the test steps not working, the requirement not being addressed by the test case written, or the test cases\requirements being truncated, mislabeled, or mismatched while being created.

**Rationale:** Reviewing for accuracy increases the likelihood that all requirements will have testable cases prior to the start of testing

**Functional Coverage**- Answers the question "Are the requirements being tested as designed or is test coverage being blown due to: the requirement not being implicitly covered by the test case(s) or the requirement not being explicitly covered by the test case(s)?"

**Rationale:** Checking cases against requirements for scope closes coverage gaps prior to the start of testing

**Standardization and Style** - These areas comprise the syntactical rules and testing nomenclature and are typically agreed upon as a group prior to the start of test case writing. The larger the group or the more formalized the testing environment, the more likely a style guide will be published and adhered to.

**Rationale:** Doing a team mind meld with regards to syntax and language enhances clarity and therefore testing productivity downstream

The SOCAL model avoids standardization and style, as they are largely emphasized in order to facilitate testing, and instead focuses on helping teams track accuracy and functional coverage as a means of evenly reviewing test case writing and pointing training efforts in the right direction.

### *The Model*

Easily implemented, the SOCAL model categorizes five different areas of potential accuracy or coverage deficiency, allowing reviewers to place errors into different buckets for later enumeration.

The categories are: Substance, Omission, Comprehension, Automation, and Logic.

### Categories Defined:

**Errors of Substance** - The requirement is not being implicitly covered by the test cases written. One or more test cases provide coverage for the requirement but do not address all possible scenarios.

Type of Error: Functional Coverage

Example:

Requirement: At application start, a user login dialog box shall appear.

Existing Test Case: 1. Start the application from the desktop icon.

Expected Result: The user login dialog box appears.

For our purposes, implicit in the example requirement is the notion that the application can also be started from the command prompt, the run box, and the start menu. As there is no definition of how the application is to be started, test cases for all applicable methods of starting the application should be generated. [Alternatively, one test that states "start the application" and refers to an established practice of starting the application in all possible ways

while testing would suffice though it would make the existing test case an error of Comprehension if the practice were established prior to the test case being written.]

**Errors of Omission** - The requirement is not being explicitly covered by the test cases written. One or more test cases provide partial coverage, but miss stated conditions or results in the requirement.

Type of Error: Functional Coverage

Example:

Requirement: Upon clicking the edit button, all fields except for the password field shall become editable.

Existing Test Case: 1. Start the application  
2. Login  
3. Click the edit button  
4. Attempt to type all keyboard characters into the password field

Expected Result: The password field is not editable.

Clearly, the writer should also have created test cases that test the editability of the other fields.

**Errors of Comprehension** - The requirement is not being addressed by the test case(s) written. The test case in question does not test the requirement.

Type of Error: Accuracy

Example:

Requirement: When closing application from the close icon [X], the application window shall close after the on-screen data is saved.

Existing Test Case: 1. Start the application  
2. Login  
3. Close the application window

Expected Result: The application window closes after the on-screen data is saved.

Merely closing the application window will not verify that the on-screen data was saved. The burden placed on the tester is to then reopen the application.

**Errors of Automation** - The test cases are being truncated, are outdated, or are mismatched while the test case document is being created or revised.

Type of Error: Accuracy

Example:

Requirement: Double-clicking on the installation icon after SQL client 1.3 is installed shall display an "Install from network" pop-up

Existing Test Case 1. Install SQL client version 1.1  
2. Double-click on the installation icon

Expected Result The "Install from network" pop-up is displayed.

At first glance it appears that someone mistyped the SQL version number. In actuality, the requirement was updated and the test case left alone.

**Errors of Logic** - The test case steps do not produce the intended result because the scenario does not exist and/or the steps do not produce any result.

Type of Error: Accuracy

Example:

Requirement: Double-clicking on a line item listed in an active or archived account file shall display the line item properties pop-up.

Existing test cases (1): 1. Open the application (active account file is loaded)  
3. Enter Edit mode.  
4. Double-click on a listed line item.

Expected Result (1): The line item properties pop-up is displayed.

Existing test cases (2): 1. Open the application (active account file is loaded)

3. Enter Read-only mode.
4. Double-click on a listed line item.

Expected Result (2): The line item properties pop-up is displayed

- Existing test cases (3):
1. Open the application
  2. Load an archived account file
  3. Enter Read-only mode
  4. Double-click on a listed line item.

Expected Result (3): The line item properties pop-up is displayed.

- Existing test cases (4):
1. Open the application
  2. Load an archived account file
  3. Enter Edit mode
  4. Double-click on a listed line item.

Expected Result (4): The line item properties pop-up is displayed.

The problem here is test case #4. For our purposes, there is no Edit mode for an archived account file. The scenario does not exist and therefore the steps do not work.

### *Implementation*

**How-** One suggested method for applying the model would be to:

1. Create an extra “review” field in the test case database and populate it with the 5 error types
2. Train reviewers on recognizing the error types.
3. Review test cases and select the error type for each test case corrected that can be categorized using the model.
4. Create team and individualized reports to track trends and focus team member and individual attention on deficiencies.

**When-** The timing of when to apply the SOCAL model is dependent upon your project’s test writing methodology. If you write often and early, trying to match at least one test case to each requirement as a means of meeting a preliminary submission deadline, and then expand the number of test cases as time goes on, it would be best to wait until you have declared that all sections should now be written with complete coverage before implementing. If you approach test writing by asking writers to provide complete coverage for all written cases from the onset, the model is applicable for review from the start of the review process and can provide a richer set of data.

**Where-** Though written at the test case level, the model has been successfully applied to the review of testable requirements, test cases, and procedures.

### *Benefits*

- 0Standardization of review nomenclature criteria
- 1Allows for metrics, reporting, cost justification
- 2Reinforces training on categories of errors to avoid for reviewers
- 3Can be used as training or tool for new hires (extract several examples of each category of error and put them in a training database for use as a guide or an exam tool)

### *Lessons Learned*

Anytime a tool puts a heretofore non-standardized piece of team member performance on display, there is concern. In short, proponents of the team dynamic will find much smoother waters.

Upon adoption, with a seasoned group of test writers, the percentages of overall errors were predictably low with the largest clusters of errors occurring:

- Right after lunch (Comprehension, Omission)
- When misinterpreting the first of a series of similar, complex requirements and continuing on in the same vein (Substance, Omission)
- When newer project members didn’t understand the technology well enough to write cases that worked (Logic)

In review sessions, when viewed through the lens of a test writer, some of the errors of substance, omission, and comprehension boil down to a lack of clarity on the part of the requirements writer. As such, it is important to have

an efficient communication mechanism between the test writer team and the requirements team before writing starts in order that uncertainties can be cleared up. For organizations that have this in place, it usually is a mistaken assumption on the part of the writer that causes the error.