coverity®
A Synopsys Company

WHITE PAPER

Accelerate Time to Market with Change Impact Testing

Too many precious testing cycles are wasted in many organizations:

- 30% of tests performed are ineffective
- 30% of tests cover 65% of regression risks
- 30% of tests are redundant

The IT application lifecycle is constantly accelerating to meet the need of continuous and rapid evolution of information systems to correspond with business needs. This acceleration is enhanced by the adoption of agile methods that “continuously” produce new versions of applications.

Faced with this acceleration, it becomes imperative that teams focus their testing efforts based on the impact of change. It is impossible to fully retest each application release but quality must not be sacrificed, since the backlash from regression is immediate and expensive.

Analysis of testing activities on the applications in our study revealed that the number 30 is crucial to improving testing. This number is relevant when teams are required to:

- Shorten the release cycle to accelerate responsiveness of information systems
- Adopt agile methods
- Increase the overall effectiveness of tests

This white paper explores these results to help you identify areas for improvement on tests of your own projects and applications.

Development of this Study

This study was conducted with data collected from more than 24 applications using Coverity Test Advisor – QA Edition to improve their tests. The analyzed information is the result of aggregated and anonymous data without reference to specific applications. These applications were sourced from a variety of business sectors: insurance, banking, manufacturing, pharmaceuticals, software publishing, e-commerce, etc. Their nature was also very diverse: web applications, client-rich applications, SOA implementations, software customization, etc. A variety of application sizes is well represented, from small (80,000 lines of code) to very large (over 4.5 million lines of code).

30% of Functional Tests Performed are Ineffective

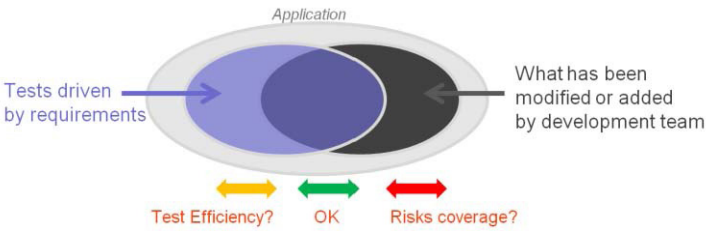
On any given release, 30% of tests provide no real value!

What is an ineffective test?

In our methodology, a test is evaluated as ineffective if its probability or its ability to detect a new anomaly is zero or very low. This probability is estimated from the test footprint which represents the test's execution path within the application. If, during its execution, a test does not touch any changed or added element (code, etc.) in the tested application version as opposed to the previous version, then it is categorized as ineffective for this version.

> *Test Footprint*
> *Also known as the test coverage, this is the set of executed instructions within the application code during a test run. In addition to the instructions, Coverity Test Advisor – QA Edition keeps track of the execution flow of each test (execution tree).*

The illustration below describes this concept, where the requirements and specifications will generate the execution of non-relevant tests. At the same time, the test will miss risk areas related to what has actually been modified by the development team within the application.



Why Avoid Ineffective Testing?

The majority of tests in our surveyed applications were still manual, so 30% represents many days of work and a significant impact on the release cycle. As time constraints and deadlines make it difficult to test everything, effort is devoted to these tests at the expense of other relevant tests which would ensure the quality expected by business stakeholders.

How to Avoid Ineffective Testing

Identifying ineffective tests before running them is not simple. It is necessary to know how to relate application changes to the relevant tests.

To do this, you must do three key things:

- Obtain a comprehensive view of the changes. This requires rigorous use of version or configuration control, but can usually only be taken advantage of during tests run by the developers (i.e. unit testing).
- Acquire a functional vision of changes to assess their impact. Changes are made in code by Development, but testing activities are made by Quality Assurance (QA) and are based on a functional vision. This requires strong cooperation between developers and testers. In addition, both teams must have the ability to share their different points of view about the application. Release notes, where they exist, rarely satisfy this requirement.
- Ensure good traceability between functional vision and test scenarios to identify effective tests. Beyond merely understanding the functional impacts of changes, it is important to identify the relevant tests which cover these risks. Rigorous use and excellent organization of the test repository are essential.

If these problems appear difficult to overcome, another option is possible: the one we used for this study.

How Coverity Test Advisor – QA Edition Identifies Ineffective Tests

Coverity relies on technology that was created to solve this problem. At each text execution, its footprint is automatically recorded. The footprint is used to link the test to each application instruction it calls. So, when a change is detected at the

instruction level in a new version, the tests that should be run based on change impact are immediately identified. As more tests are run, their footprints are consolidated in a knowledge base. Changes are also identified through a technique we developed which "x-rays" applications.

Identification of Changes

Coverity Test Advisor – QA Edition detects all changes in the application without the need for source code or access to the version control system. The analysis is done at the binary code level and on all application resources (web, SQL, etc.).

30% of Tests Cover 65% of Regression Risks

On any given version, 30% of tests can cover the majority of regression risks!

Regression Risk

A regression risk is related to changes made in the application code when adding new features or correcting an anomaly. These changes can impact the modified functional subset but very often generate side effects: malfunctions of other features that were not meant to change.

These edge effects of changes are difficult to determine, and therefore require, for safety, running a large number of tests to detect them. The cost of regression tests can comprise a substantial portion of the overall test time and test cost of an application. In addition, regression tests can cause a significant drag on the release cycle which hinders the agility of the information system.

Automated testing is one solution to try to reduce the test cost, but the cost and maintenance may not be suitable for every QA team. This is why it is so important to foster an effective approach for accurate identification of effective testing based on change impact.

Identifying tests that adequately cover risks requires collecting accurate identification of all changes and the footprints of previous tests. The tests which cover regression risks are those whose imprint is impacted by at least one of the changes made. Their results are not guaranteed because the code they run has been changed since their last execution.

Once all these tests have been identified, in order to maximize the effectiveness of campaigns, it is necessary to prioritize them.

Several possibilities are available:

- Use a functional view of the application to identify changes in the most critical business areas and prioritize tests to be executed.
- Select the tests which are most effective in risk coverage and provide the same level of security. This will be those tests which cover changes to the maximum extent, avoiding multiple tests where only one may be sufficient.

This analysis, performed on the applications used in this study, shows that it is possible to cover, on average, 65% of risks by executing only 30% of regression tests. An iterative approach is preferable: after performing a first series of tests, teams must analyze the risks of excluded regressions and choose any additional relevant tests.

Business Vision

Within the Coverity solution, this vision is provided through a configurable model which identifies the functional areas of the application to attribute a level of criticality to each functional area. Moreover, each area is linked to the application code that implements it. So, when modifying code, the functional area or areas affected are identified.

How Coverity Test Advisor – QA Edition Helps Define an Effective Strategy

The optimization of a test campaign is done via a wizard that allows the manipulation of all the data collected in the application's knowledge base: test footprints, changes in each version and the functional model of application. The selected tests are then exported to the testing tools such as HP ALM/ HP Quality Center, Selenium and many others to initialize campaigns.

Change impact testing is particularly applicable to regression testing.

30% of Tests are Redundant

Given the set of all tests on a development version at deployment, 30% of tests are not complementary but are actually redundant!

What are Redundant Tests?

Throughout the lifecycle of a release, the tests undertaken are very different. Whether unit testing, integration testing, system

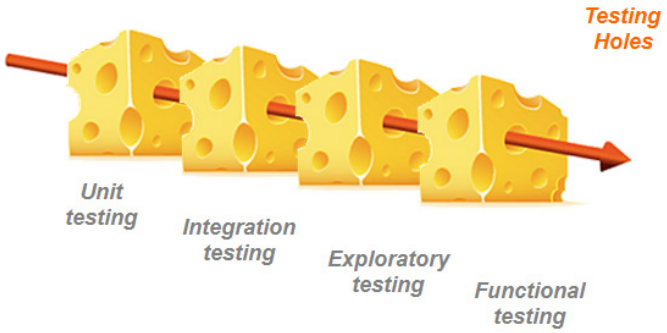
testing, acceptance testing or functional testing; these tests are run by different people: developers, integrators, functional testers and end-users utilizing both manual and automated systems from a GUI or running scripts.

All these differences mean there is rarely an aggregate view of all tests performed, even in agile teams where the testers are close to the developers. It is impossible to really know what is tested or what is not tested.

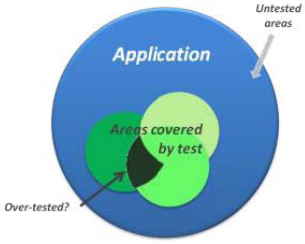
The Swiss Cheese Metaphor

The image below perfectly represents the idea of an aggregate view of all test activities on a given application version. Each type of test performed (unit, integration, system, acceptance, etc.) is represented as a slice of cheese. Untested application areas are represented by holes in the slices.

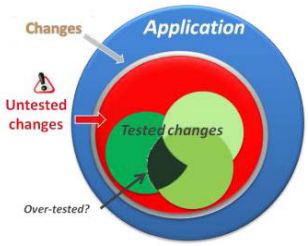
In the lifecycle of a version, slices (i.e. tests) are stacked, but overlapping holes aligned in all slices are to be avoided. These represent areas of the application never tested by any type of test. Any minor change in these areas poses a regression risk which is never covered.



By taking footprints of all tests during their execution and by aggregating these footprints, we identify what is tested – and is not tested – overall in the application.



When changes made in the current version of the application are added to this representation, "test holes" can quickly be identified.



The 30% of tests which are redundant are represented by the "over-tested" area. These tests minimize the risks in these areas, but at the expense of untested areas. To optimize a test strategy, it is worthwhile to rely again on a functional vision to identify what features are untested in the "test holes," and to prioritize actions based on the associated business risks.

Redundant testing is not a problem in and of itself, but exposing it is an important step to increase test effectiveness.

Conclusion

Most customers whose applications are referenced in this study are faced with a major challenge: to increase the agility of their information systems, or of their products in the case of software vendors. Following this agility is the ability to deliver more frequent releases to respond quickly to changing demands. All this without losing control of quality, because otherwise versions keep coming in a vicious cycle to resolve ongoing problems found in production.

On its own, the traditional approach taken from requirements and specifications is limiting in this situation, and does not provide the necessary effectiveness. Automation is a perfect solution in theory but in practice, it is not always so rosy.

Several options exist:

- Break the developer/tester silos to improve the capacity for dialogue and collaboration and to better target tests.
- Gain an overview of all tests performed on the release cycle. It is unthinkable not to rely on all tests performed for a real view of risk coverage.

- Mix these approaches with pragmatism to confront requirements-driven testing and white-box testing and make use of data to boost testing effectiveness.

Appendix: Overview of Techniques and Technology Used

This study is based on the analysis of anonymous data generated by companies utilizing Coverity Test Advisor – QA Edition.

Note: For the purpose of this study, we refer to the term "version" to mean any build that was created manually or by automated tools, whether an official release candidate or not.

Analysis of Each Version or Build

To analyze the relevance of tests, each version is automatically x-rayed to detect the changes introduced (see Figure 1) and evaluate its impact in functional terms (see Figure 2).

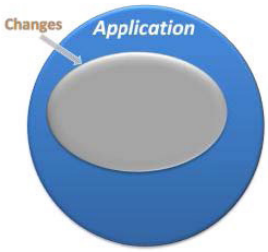


Fig 1: The proportion of modified elements of the application (code, etc.) in this version versus the overall application.

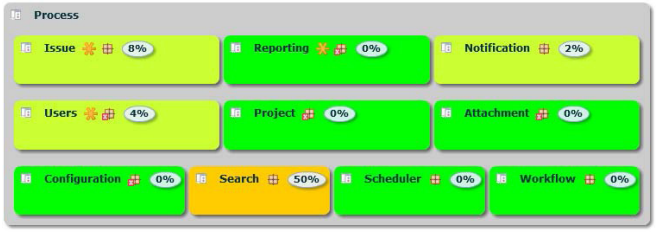


Fig 2: Identification of functional subsystems impacted by the changes made (% change).

Test Footprints

Tests captured by Coverity Test Advisor – QA Edition could be manual or automated, unit, integration or functional. The footprint of a test is the set of all instructions executed in the application from actions taken during the test. These include the call trees and executed instructions in the application which respond to the user's action. The footprint is collected automatically through an agent that records the operation of the application under test. Plug-ins for testing tools make this operation transparent to testers.

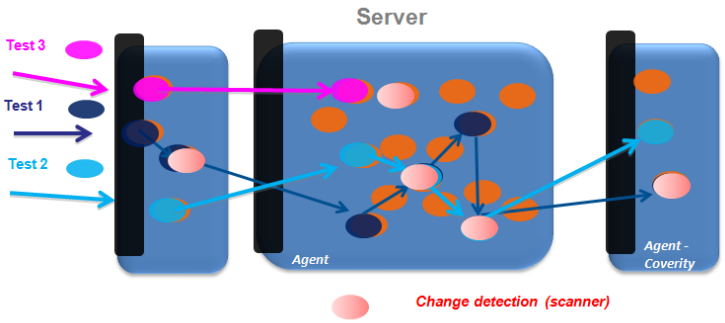


Fig 3: Test footprint collection

As tests are performed, their footprints are aggregated to identify tested and non-tested areas. This information can be viewed from several angles in order to best optimize tests: aggregate view (Fig 4), functionalist views (Figures 5 and 6).

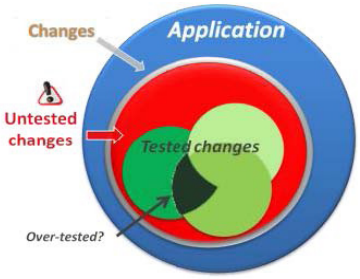


Fig 4: Risk identification through the correlation of changes and test footprints.

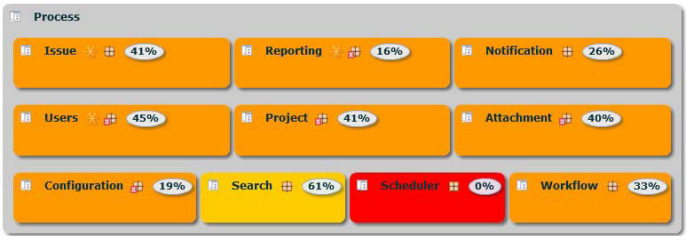


Fig 5: Identification of untested or poorly tested areas. Here the general coverage of functional subsets by testing. In this example the Scheduler subset has not been tested.

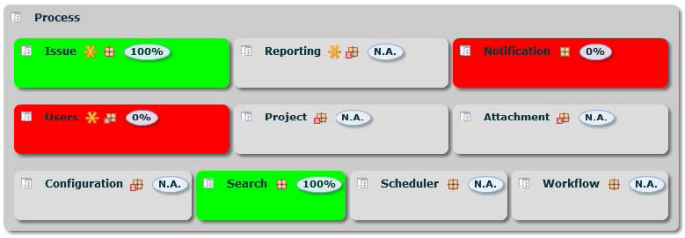


Fig 6: Identification of untested changes. Analysis of changes is crossed with test coverage. In this example, changes in the Notifications or Users subsets have not been tested.

coverity®
A Synopsys Company

For More Information
www.coverity.com
Email: info@coverity.com

Coverity Inc. Headquarters
185 Berry Street, Suite 6500
San Francisco, CA 94107 USA

U.S. Sales: (800) 873-8193
International Sales: +1 (415) 321-5237
Email: sales@coverity.com