Test Strategy for High Quality EDA Software

Richard Léveillé
Synopsys Inc.
Mountain View, California, USA
Richard.Leveille@synopsys.com

Abstract

The complexity of EDA (Electronic Design Automation) software is rapidly increasing to keep pace with Moore’s law, technology advances, and time-to-market pressure imposed by the fast-pace consumer electronic marketplace. To be relevant in the value-chain of the electronic industry, not only EDA software needs to provide the right set of features, but it must also deliver them in a timely manner to integrate in the design flow of consumer electronics.

This paper describes a proven test strategy in use at Synopsys to achieve predictable quality and release by skillfully using code development tools and verification methods throughout the development cycle. The description offers insights as how turnaround time, cost-of-testing, and high standard throughout software development help shape the strategy for high quality releases.

1. Introduction

The EDA (Electronic Design Automation) software industry is an integral part of the value-chain enabling the electronic consumer market, providing the design engineers the tools they need for the development of Integrated Circuit (IC). The consumer market is cyclical and meeting these cycles often means success or failure for a product or a company. With an increasingly short product development cycle, there is little room for iteration to ensure success in a competitive environment.

In this context, the EDA software must meet customer expectations for functionality, performance and quality, as well as satisfy predictability of the release to meet needs of the product launch. In addition, the EDA software must: support latest technological breakthroughs, harness the increased complexity of the chip design, control the very high cost of production, and meet the time-to-market objectives. This requires executing quickly on the latest requirements while keeping stability high and predictable delivery dates.

This article describes an effective strategy, in use by Synopsys, for testing of EDA software. The document details the approach, the criteria used to define the strategy, the principles guiding its implementation and execution, and finally offers concrete actions taken and the results achieved.

2. Defect Prevention and Removal are the Goals

The Test Strategy is only one part of the overall Quality Program contributing in meeting the objectives of functionality, performance, quality, and timely availability. Synopsys’s Quality program is anchored by three vectors:

Quality by Design: Developing high-quality software. Focus on design-in quality method and continuous process improvement based on Best Practices for software development.

Quality by Validation: Detect and fix defects before release. Anchored by test methodologies and code analysis tools.

Quality by Defect Management: Timely resolution of customer reported defects.
This article focuses on Quality by Validation, more specifically for the portion of the development life cycle starting once the code is written. We, first, explain the business reasons supporting effort in optimizing testing, second, we describe the Test Strategy Matrix used to select the most efficient test methods, and finally, share Synopsys results achieved in the past few years based on this approach.

2.1 Managing Quality by Validation

Even today, despite the industry's best effort to design-in quality in early stage of development with better requirement definition, design reviews, etc.; the number of defects present in the code is still significant. Industry studies have shown that, even after completion of unit testing, 10 to 20 defects/kLOC (Lines of Code in thousands) are still present in the code as we enter the integration and system test phases [1]. Findings these defects is a constant challenge especially with applications reaching 1 million LOC; this means 10,000 defects.

There is a considerable amount of resources, capital and human resources, dedicated to the verification and validation of software products. Any gain in test productivity will directly benefit the whole organization by reducing cost of testing.

**Testing represents a significant portion of the development effort:** About 40-50% of the schedule is dedicated to one form or another of testing. Although, a sound test approach and automation will optimize the effort dedicated to the activity, the scope of testing is challenged by a constant increase in code complexity and the expanding number of configurations and platforms to support.

**Testing offers best measure of forward-looking Quality:** By taking a close look at the product development life cycle, we quickly discover that the testing activities provide us with most of the information on the quality of the product release. In fact, very little information prior to the first cycle of testing can help quantify the quality of the release. Although best practices prevent defect insertions, it does not quantify product quality. Predictive models can be used but they are estimation of quality, not an actual measurement of it.

**Testing contributes to cost reduction:** Well-established industry studies have shown that the cost of fixing defects found during post-release is 5 times higher than when found during design [2]. Another fact is that, typically, it takes on average 10 to 20 hours to find one defect and to fix it [3]. A quick multiplication by a factor of 5 for defect found post-release makes a convincing argument for early defect detection.

**Testing is Time-Bound:** As highlighted earlier, the quality challenge is time-bound as features must be available on time to satisfy the time-to-market requirement from customers. Without the time constraint, the overall approach to the quality would be different and the effort would be shaped to optimize against a different set of parameters.

**Testing supports Developers:** Testing is also crucial as a mechanism to provide quick feedback to the software developers on his/her own progress during development.

2.2 Test Strategy Objectives

The previous observations and considerations for testing can be summarized in Figure 1 showing the typical Defect Finding Rate pattern observed in the software industry. It illustrates the fact that a) the majority of defects are found in the last phase of the development cycle during testing, b) late found defects increase the risk to the release, and c) testing represents a major portion of the development effort.
Based on these facts, the objectives of our testing strategy are to:

- Detect defects early
- Use efficient detection methods
- Minimize Cost
- Minimize Risk

By having a disciplined approach, selecting test methods with best return-on-investment, and optimizing the test cycle; quality releases are achievable. The following section describes the matrix for selecting the test methodologies.

3. Test Strategy Matrix

Based on the defined objectives for our test strategy, let’s examine the various testing methods used at Synopsys. Because more than one test method will be necessary to cover the test domain, and because each test method has its respective advantages and limitations, we characterize each testing methods by: Conditions for test initiation, Coverage, Turnaround time, Cost-of-Testing, Fix Complexity, and Risk of the Fix. The Test Matrix, in Table 1, provides classification of the test methods for the six characteristics.
<table>
<thead>
<tr>
<th>Testing Method</th>
<th>Test Matrix</th>
<th>Defect Reduction per Test Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Early Detection</td>
<td>Efficient Detection</td>
</tr>
<tr>
<td></td>
<td>Min. Conditions for testing</td>
<td>Coverage</td>
</tr>
<tr>
<td>Static Code Analysis</td>
<td>Code written</td>
<td>100%</td>
</tr>
<tr>
<td>Code Review</td>
<td>Code written</td>
<td>Partial</td>
</tr>
<tr>
<td>Regression Testing</td>
<td>Code integrated</td>
<td>Partial</td>
</tr>
<tr>
<td>Benchmark Testing</td>
<td>Code integrated, regressed and stable</td>
<td>Partial</td>
</tr>
<tr>
<td>User Scenario</td>
<td>High code stability</td>
<td>Limited</td>
</tr>
<tr>
<td>Flow testing</td>
<td>High code stability with multiple products</td>
<td>Limited</td>
</tr>
</tbody>
</table>

### 3.1 Early Detection

Testing should take place as soon as the conditions are sufficient to provide valuable data; not when one expects the results to be perfect. For example, for Static Code analysis, testing can start as soon as the code is written and compiled. In the case of the User Scenario testing, the code must be fully integrated,
regression tests passing, and feature testing complete before testing can start. By initiating testing early on, we obtain early feedback on the quality of the software and eliminate defects that might take hours to find later.

3.2 Efficient Detection

Defect detection is the ultimate objective but it must be achieved efficiently to support iterative development work and predictable results. We look at two aspects of efficient defect detection: High Coverage and Fast Turnaround Time.

**High Coverage**

All test methods serve a different purpose but they are not all equal in covering the test domain. The coverage and the probability of detection one test method offers compared to the other one drive the selection of the test method or tool.

The best example of high coverage is the Static Code Analysis where every line of code is covered. On the other end, Dynamic Code Analysis is a function of the regression testing coverage. Our test strategy favors the test method with high coverage which can be run early in development.

**Fast Turnaround Time (TAT)**

The objective that we pursue here is to create a quick resolution loop for the developer. The Turnaround Time is crucial to optimize the developer’s productivity in providing a short find/fix/validate cycle to resolve defects. Another factor favoring fast TAT test method is how frequent measurements can be obtained to track progress against set targets. Test automation is instrumental here as it provides efficiency, and timely metrics in support of decision-making and corrective actions.

3.3 Minimize Cost

As mentioned previously, the average time to find and to fix a defect is 10-20 hours [3]. Fixing defects is a considerable cost just by itself but we also have to account for the initial investment to create the tests and infrastructure to execute those tests.

The Cost-of-Testing encompasses the necessary effort and resources to define test cases, instrument the tests, create/maintain test harnesses, perform the tests, and validate the fixes. As applications become more complex, with a constantly growing size of code, the design of those tests requires a substantial engineering effort.

The following Figure 2 illustrates the Cost-of-Testing for the different test methods relative to the Turnaround Time required to perform these tests and close the find/fix/validate loop for a defect.
3.4 Minimize Risk

Fixing defects, even for a small change in the code, can be very risky. Considering the Fix Complexity is not sufficient; the risk to the release is also accounted in our development practice.

Fix complexity is linked to risk

Some of the tests will reveal flaw in the design, requiring fixes that can considerably increase the risk to the release. Benchmark testing, for instance, is more likely to require more fundamental changes to the code than the ones revealed with Static Code Analysis. The complexity of some of the changes becomes a major factor. Last minute code re-engineering may pose serious delay to a release and may compromise its quality.

Risk of applying the fix late in the release cycle

The other consideration is the risk in making changes to the code towards the tail-end of the release cycle. Good practice in code change management will gradually impose more restrictive conditions for code changes; as we get closer to the release date, the nature and number of changes being introduced are carefully controlled, and limited. In reality though, the changes might still be required in order to meet your release objective. This is why a disciplined approach and early testing is crucial.

4. Actions lead to Results

The strategy is only as good as the results it produces. Over the past years, Synopsys took specific actions to improve software quality and the development infrastructure to meet the set objectives: detect early, detect efficiently, minimize cost, and minimize risk.

The Test Matrix allowed the organization to identify any gaps in the test approach. Proactive actions were taken to complement and improve our test methodology, improve the development infrastructure, and improve test processes and execution.
• Complement testing methodology to increase coverage and to test earlier in development
  o Strengthen early detection and improve coverage by introducing new tool for Static Code Analysis
  o Promote best practices for Code Review and increase review productivity with collaborative tool
• Improve development infrastructure by providing better set of tools and increase test automation
  o Add testing compute resources to improve Turnaround Time for test results
  o Increase frequency of measurement for Static and Dynamic Code Analysis for better progress tracking
  o Automate benchmark testing
• Improve test processes for predictable release and quality
  o Release management: Setting more stringent goals, intermediary milestones for each metric
  o Plan and set measurement points to gain better visibility in the state of the release
  o Automated measurements/collection/reporting of key metrics in support of release tracking
  o Maintain high standard of code integration and regression right from the start of the release in support of early benchmark and flow testing

All these efforts have made a difference as shown both, internally, and externally, directly benefiting our customers. Over the course of the past few years the changes have contributed in many ways:

Internally:
• Reduced number of last minutes issues putting releases at risk
• Increased visibility of intermediate results throughout the development cycle to better manage and react to changes
• Increased productivity for developers: An internal survey revealed that more than 70% recognize that the effort had a positive impact on their productivity
• Improved Turnaround Time on daily build and regression testing by a factor of 3 in the last 3 years
• Improved Turnaround Time on local (developer sandbox) build and regression testing by a factor of 2 in the last 3 years

From a customer point-of-view:
• Reduced Incoming Defect Rate from release to release
• Reduced Defect Backlog by 35% over the past 4 years
• Reduced Median Age of customer defects by 34%
• Allowed Synopsys predictable releases to scale with increasing number of products

• Customer satisfaction surveys of the past 2 years show that 75% of Synopsys customers indicate quality is as good or better than previous year

5. Summary

Customer expectations dictate our approach to software development. The methods and the software development tools used are carefully chosen and tailored to meet those expectations. In understanding the advantages and limitations of each the Testing Methods, we have proven that predictable quality is attainable. The tools and methodologies are effective because they are framed by a structured approach to development. Setting goals, frequent measuring points, and planning for test activities are essential in achieving predictable quality. The reward comes from the fact that specific actions have led to significant results for quality improvement and on-time delivery.

6. References


7. Acknowledgments

Sincere thanks to Pramod Suratkar, Synopsys Quality Analyst, for his constructive feedback.