White Paper

QAMentor
Software Quality Assurance Expert

The IMPORTANCE of *Static Testing* and **How** to ACHIEVE High ROI

www.qamentor.com

Introduction

With only one quarter of US companies engaging in static testing, it's an option too often overlooked. Opponents to the process correctly state that static testing is time consuming and can be expensive. But what they fail to recognize is that the return on that investment is far higher than any money presumably saved by not utilizing it. Static testing is not always easy, but fixing defects is far more expensive and time consuming when they are found later in the SDLC, such as when this early testing is skipped.

The cost of a defect found during traditional testing is about 10-20x the cost of one found in static testing. The reason is simple. It takes longer to fix a defect once it's been coded into a functioning system. Finding the defect may take hours of a tester's time, additional hours to fix the defect that could involve more than one developer or DBA, and then additional time for the tester to verify the fix. All totaled, a defect found during traditional testing can take 5 – 20 hours (or more) to fix and verify. Contrast that with finding that same defect in static testing. Fixing it requires no code changes and only takes 30 – 60 minutes. Sometimes it takes even less time, so the difference in cost is dramatic. The inevitable conclusion is that relying solely on dynamic application testing to find defects isn't the best or most efficient means to testing.

What Static Testing Is

Most people know and understand that the sooner defects are found and fixed in the SDLC, the cheaper the process is to fix them. However, finding and fixing defects isn't only about testing and modifying code. Testing can and should start before a single line of code is written - immediately following the requirements gathering, or in some cases during that process. At this stage, defects cost nearly nothing to fix, as opposed to waiting until the issues hit the test or production environment.

Static testing involves checking documents such as requirements for thoroughness, standard deviations, or completeness. This testing can be done by developers, testers, and business analysts during requirements reviews, or by multiple team members in software walkthroughs.

Documents Tested

While any document can be used for static testing, the following (or variations of them) are the most common ones.

- Business Requirements Document (BRD)
- System Use Cases
- System/Functional Requirements
- Prototype or Mockups
- Prototype Specification Document
- Design Documents
- Traceability Matrix Document
- Test Plans
- Test Cases (Unit, System, Functional, etc.)
- Test Data
- Test Scripts (Automation, Performance, Manual, Security)
- User Manual/Training Guides/Documentation
- Functional Specifications
- Field Dictionaries

Benefits of Static Testing

As with any kind of testing, the ultimate goal is to reduce the number of defects in production. However, in the case of static testing, the process goes one step further to attempt to reduce the number of defects in the documentation that the application code is based off of. By looking at the documentation before developers start their job, we can identify, anticipate, and correct software issues at the earliest possible time. In this way, defect prevention is started immediately and the testing cycle is shortened, hardcoded defects are reduced, and a faster time to market is more easily achieved.

Lower Cost

Traditional dynamic testing requires doing in depth test case management, manual and/or automated testing, defect entry, fixes, and re-testing, and regression testing at a bare minimum. All of this work is done over long hours by multiple individuals and much of it after the application is built. It can end up being very expensive, time consuming, and cause many a manager to question the ROI.

While it may always be necessary to do this kind of testing, adding static testing into the mix at the beginning will reduce the time and effort needed during dynamic testing. It's much cheaper to have a meeting and identify issues than to let those issues surface months later while executing code that already has hours of labor put into it. That's assuming the issues are even found at all. Once they make it to production, the costs multiply even more. Using static testing from the get-go reduces testing costs to a bargain of potential expenditures.

Earlier Detection/Faster Fixes

Nearly half of the defects that traditional testing identifies could have been found much earlier in static testing. Before code is ever written, defects can be found and fixed – or eliminated, depending on how you look at it. Projects with aggressive deadlines will benefit even more from this type of testing by reducing the time-to-market dramatically. A defect found in dynamic testing can take several hours or even days to fix, while one found in static testing takes a mere hour or less.

Very Effective

Testers, designers, developers, and other subject matter expert's work together prior to coding to ensure that the requirements are clear, concise, and make sense for the goals of the project. Designers, developers, and DBAs work together to ensure that the design and/or architecture works well together to meet the requirements and fix any potential problems by running through mockups or prototypes. This continues through the static testing process until every piece of the application, every document, and eventually all of the code has been reviewed by multiple professionals. With nearly 100% coverage, static testing has proven to be many times more effective than dynamic testing alone.

How Static Testing is performed

QA Mentor has established a strong and proven methodology for static testing and recommends other companies do the same. Incorrect testing processes can and often do lead to insufficient outcomes. The following methodology is what QA Mentor recommends.

Inspections

Establish a thorough inspection process to completely review the design of the application. Review standard button placements, menu usages, and optimal colors along with the technical aspects of the application design. Consider incorporating 'click tests' on prototypes to make sure users see what you expect them to see, and are encouraged to click where you'd prefer them to click.

Reviews and Checklists

Create comprehensive inspection checklists for each document that will be reviewed. Doing so will ensure all areas are completely covered. Consider checklists for the following documents especially:

- Use Case Requirements – The checklist should validate that all end-user actions are identified and any input/output associated with them. Check step sequences for easy test case creation, boundary conditions, pre/post conditions, and interface definitions along with any alternate paths or inheritance issues.

- Functional Requirements – This checklist should help to identify all necessary functional capabilities, user characteristics, constraints, calculations and methods, and reports. Database functionality, interface listings, hardware and software requirements, and network needs should also be identified.

- Prototype/Screen Mock-up – The checklist for this document should aid in matching use cases to requirements and verify usability, navigational flow, and simplicity of the interface. Primary and alternate paths should be valid.

- Field Dictionary – The field dictionary checklist verifies that each field in the UI is defined well enough to create field level validation test cases for min/max length, data type, list values, error message, required fields, and masking requirements. Form, field, and database level validations are expected, and it should include mapping to the prototype screens.
- Architecture Design – This checklist helps users review business level processes associated with development plans, database accessibility, server locations, network diagrams, protocol definitions, and load balancing for both production and test environments.

Walkthroughs

Full application walkthroughs should be performed with developers, testers, and business stakeholders. This process helps to identify anomalies in the design and allows consideration of alternative approaches with the input from business team members. Business and industry standards can also be evaluated during a walkthrough to ensure that the application adheres those legally or ethically required.

Traceability and Root Cause Analysis

The defect traceability should be fully documented for clarity and inclusiveness. This helps the transfer of information between developers and testers if they happen to move between projects. Performing a root cause analysis on defects also helps to minimize the impact of any future defects by providing documentation of issues already examined.

ROI Calculation

Performing an ROI Calculation upon completion helps to showcase how much money was saved by finding the defects early in static testing. This is especially useful for selling the process to management teams and to ensure that the development and testing teams understand the usefulness of Static Testing. Below is QA Mentor's recommended method for ROI Calculation.

Static Testing ROI Calculation

1. Calculate total # of hours spent static testing
2. Convert those hours into a dollar amount – this is the amount of money invested in static testing
3. Convert the # of defects found in static testing into the number of hours that would have been needed to fix them if they had been found in traditional testing (1 defect = 10 hours to fix/test traditionally)

4. Convert number of proposed hours fixing/testing defects into a dollar amount that would have been spent
5. Subtract the amount invested in static testing in step 2 from the proposed amount calculated in step 4 to see the amount saved by static testing

Final Notes

Static testing > Better Documentation > Better Code > Fewer Defects > Faster Delivery > Higher Quality > Happier Customers

Introduced right after the requirements gathering phase, static testing is crucial if you want to reduce the number of defects in the system or application under test. Regardless of whether you do the testing in-house or engage outside resources, QA Mentor believes that static testing is a necessary step for any organization wishing to increase the quality of their products while decreasing cost and time to market.

QA Mentor's Static Testing methodology has been proven time and again and can be implemented at any organization with our help. Our experienced QA professionals can lead the way for your QA organization into shorter testing phases, fewer production defects, and on-budget projects.

About the **Author**

The author, Ruslan Desyatnikov, is the CEO & President of QA Mentor. After working as the Global Head of Testing at Citi, Ruslan created QA Mentor to fill the gap he has witnessed in QA service providers during his near 20 years in QA. With Ruslan's guidance, unique services and methodologies were developed at QA Mentor to aid clients in their QA difficulties while still offering a high ROI. Ruslan offers monthly seminars aimed at imparting his extensive testing knowledge that can be applied to start-ups as well as large companies.

About **QA Mentor**

QA Mentor is independent software testing company and top provider of 37 different Quality Assurance testing types, with 24 different services – several of which are not found at any other company. With a global presence in 8 different countries, a staff of 100, and a pool of over 1500 crowdsourced testers, QA Mentor uses an effective and unique combination of onshore and offshore resources to keep costs low for our clients. QA Mentor aims to bring quality to the forefront of software development and become the central figure in Quality Assurance with an active QA community full of engaging discussions and resources.

QA Mentor, Inc. United States
1441 Broadway, 3rd Floor,
New York, NY 10018.
Toll Free: 1-800-622-2602
Manhattan Office: 1-212-960-3812
Brooklyn Office: 1-718-606-7086
New Jersey Office: 1-201-918-4616
Email: support@qamentor.com
Web: www.qamentor.com