Quality assurance with Blueriq

What is quality?

“Quality is not an act. It is a habit.” – Aristotle

Quality is about making organizations perform for their stakeholders from improving products, services, systems, and processes, to making sure that the whole organization is fit and effective. Managing quality means constantly pursuing excellence: making sure that what an organization does is fit for purpose, and not only stays that way but keeps improving. There is a lot more to quality than just manufacturing widgets without any defects or getting trains to run on time. Although those things are certainly part of the picture. What quality means for an organization is ultimately a question for the stakeholders. By stakeholders, we mean anyone who has an interest in the success of what their organization does.

Customers will be the most important group of stakeholders for the majority of businesses, also investors, employees, suppliers, and members of our wider society are stakeholders too. Delivering quality in an organization means knowing who the stakeholders are, understanding what their needs are and meeting those needs (or even better, exceeding expectations), both now and in the future.

This community page gives insights and tips and tricks to setup your testing strategy with Blueriq.

Testing Pyramid

“Quality is more important than quantity. One home run is much better than two doubles.” – Steve Jobs

The testing pyramid is often used in agile environments to set up a testing strategy. In this video the quality pyramid is explained on a basic level, but also the advantages of using the testing pyramid. At Blueriq we use this pyramid to setup up our automated test. By doing this, we maximize the benefit of creating automated tests.

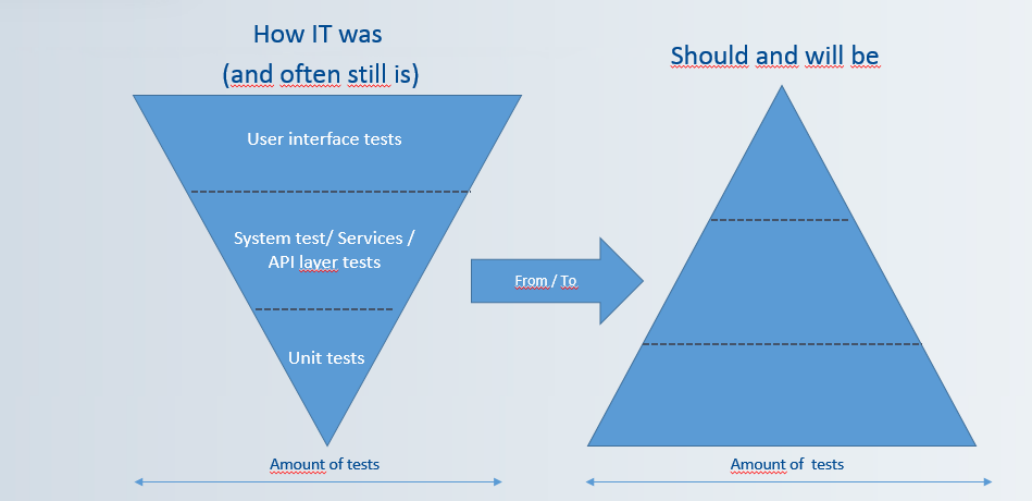
blocked URL

http://www.agilenutshell.com/episodes/41-testing-pyramid

The testing pyramid and Blueriq

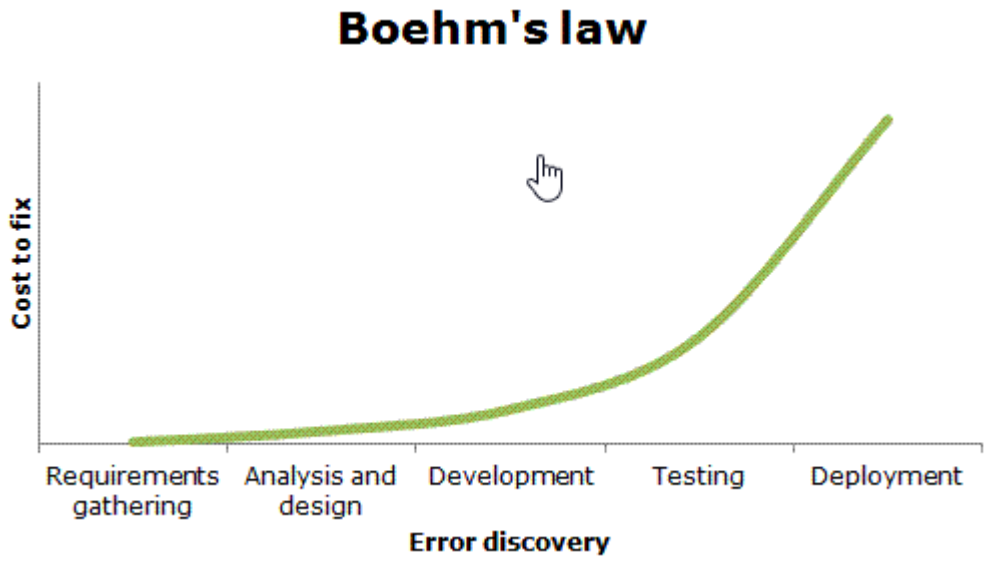
Most organizations without realizing use the testing pyramid in an inverted manner (see picture below). This is because most projects create a lot of tests on their user interface by using for example Selenium, or any other automated framework that runs tests against the user interface. To make the feedback loop as short as possible it is advisable to change this inverted way of working to match the testing pyramid. This results in:

- less scope change
- less related code
- fewer people involved and
- bugs which are found are cheaper to solve than at the end of the lifecycle

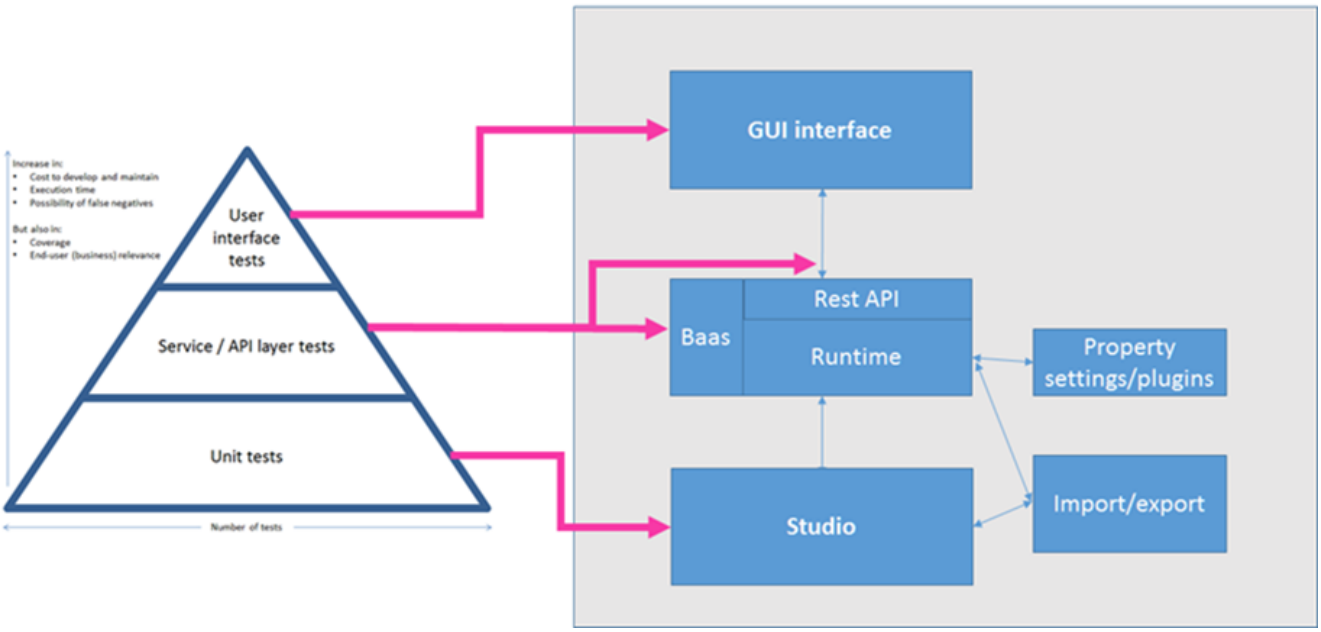


Boehm's law

The graph of Boehm visually describes that costs will exponential increase when bugs are found in the development process. This supports the integration of early testing in the agile sprint teams and creating most tests during development (in the scrum team).



Using the test pyramid, you can draft a test strategy against a standard setup of Blueriq. Which is made visible in the picture below.



Mapping a standard Blueriq environment on the testing pyramid gives the following insight:

- Studio should be tested with unit tests
- Runtime/ BAAS / Rest API should be tested with service and API layer tests
- Front-end (Guided User Interface) should be tested with User interface tests

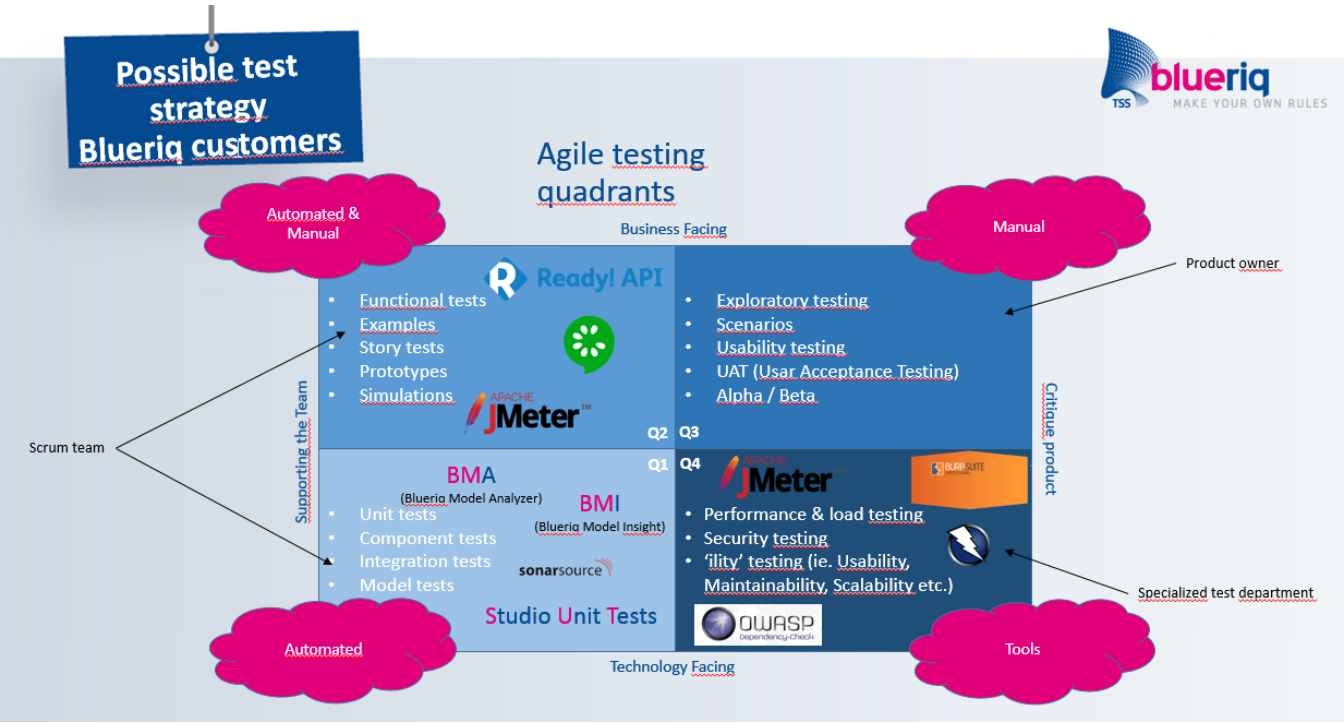
In this combined model of Blueriq and the testing pyramid, most of the tests are on the lowest level of the pyramid. This results in the following benefits:

- Cost to develop and maintain automated test is up-to-par
- Execution time on automated tests is optimal
- Possibility of false negatives is decreased
- Increase in coverage closest to the team

Agile test quadrants

For customers our advice is to use the agile testing quadrant to make a testing strategy of which type of tests you want to execute and which risks they mitigate. In this paragraph, we show how this can be implemented for Blueriq from a customer point of view. Keep in mind that every IT landscape has its own specific challenges.

"The definition of insanity is doing the same thing over and over again and expecting different outcomes." – Einstein



Agile Testing Quadrant in spreadsheet

In the previous section is a visual image of the testing strategy. In this section we give an example on how the Agile testing quadrant can be implemented (Who, Why and How ?).

For the Q3 quadrant (business facing) we do not have a proposal because this is always customer specific.

Testing type	Possible Assignees	Targeting	Reasoning	Tool which can be used	Used at Blueriq
Unit testing Model testing (Q1)	Business engineers and testers	All the new models	Ensuring that the model is correctly developed. According to standards	• Blueriq Model Analyzer • Unit testing	• Blueriq Model Analyzer • Unit testing
API / Logic / Page models (Q2)	Business engineers and testers	Functionalities implemented in new stories, past issues or bugs with high recurrence. Testing on the page modeling and exposed services	Checking to see if the Runtime is working correctly on the developed models, both functional and remaining logic which is not tested in the unit/model layer.	• Ready API / Soap UI • Cucumber • Apache JMeter • Selenium • Ranorex • Etc	•Ready API / Soap UI
GUI testing (Q2)	Testers	The graphic interface and its logic, For example, the view controller	Making sure no GUI related bugs are introduced when committing new code	• Backstop JS • Testcafe • Nightwatch • Other capture and playback tool	•Backstop JS
User testing (Q3)	Customer specific	The actual future user is testing the software. This to check the interaction between the users and the software.	Ensuring that the user has the correct interaction with the software and that the user can interact with the software	Manual interaction of the customer is needed.	Manual interaction with the users
Performance (Q4)	Development team / External expertise (Testers)	All the Blueriq components (Studio, Runtime, Publisher)	Verifying how Blueriq behaves when it comes to processing time and reliability	• Apache JMeter • Smartbear / Load UI	•Apache JMeter
Security (Q4)	Development team / External expertise (Testers)	The Runtime and its relation to other third parties' components.	Keeping and improving security standards for our application	• OWASP ZAP • Burp Suite	• OWASP ZAP

Performance

“Just as athletes can’t win without a sophisticated mixture of strategy, form, attitude, tactics, and speed, performance engineering requires a good collection of metrics and tools to deliver the desired business results.” — Todd DeCapua

By using the complete performance test project that Blueriq has developed, it is possible to test the IT environment of the deployed Blueriq application. If there are significant differences in the results between Blueriq and the specific installation, then there could be hardware issues involved (e.g., server, network, client). After doing this initial check it is advised to create a performance test with for example JMeter in your own environment. As an advice to our customers, we strongly advise to include performance test from the beginning of the project In this way the performance can be monitored from the start and you lower the risks of not having an acceptable performance when introducing applications with Blueriq to production.

Performance information:

Performance tests are introduced from Blueriq 9.9: Performance reports

Modeling and performance tips: Performance Tuning

For more information on these performance services please contact our support desk.

Security

“Testing is an infinite process of comparing the invisible to the ambiguous in order to avoid the unthinkable happening to the anonymous.”— James Bach

Developing secure software is critical to a company’s reputation and revenue. Blueriq customers feel the legal and financial pressure to assure the security of their (mission-critical) applications and have strong regulatory and privacy requirements to protect private data. Therefore security is treated at highest priority in the development of Blueriq.

To introduce the security checks in an agile development process the following advice needs to be taken into consideration:

Penetration test

Keep in mind that it is advisable that a penetration test is advised on a production environment before your Blueriq application goes live.

OWASP Dependency Checker

A dependency checker is a utility that identifies project dependencies and checks if there are any known, publicly disclosed, vulnerabilities in 3rd party components. This dependency checker is integrated into our Jenkins CI server. The OWASP Dependency-check is used to scan our dependent libraries to identify any known vulnerable components. The core engine contains a series of analyzers that inspect the dependencies, collect pieces of information about the dependencies (referred to as evidence within the tool). The evidence is then used to identify the Common Platform Enumeration (CPE) for the given dependency. If a CPE is identified, a listing of associated Common Vulnerability and Exposure (CVE) entries are listed in a report.

Dependency-Check updates using the NVD Data Feeds. For more information about the dependency, checker see: OWASP dependency plugin (Jenkins). The standard libraries with Blueriq are checked this way. When using extra libraries, it is advisable to run the tool also over the additional libraries.

OWASP ZAP

The OWASP Zed Attack Proxy (ZAP) is one of the world’s most popular free security tools and is actively maintained by hundreds of international volunteers. It is used as a proxy to run selenium tests through and then ZAP can spider further throughout the complete application. ZAP attacks the application with the most popular (OWASP top10) attacks such as injections, clickjacking, XSS, csrf. ZAP should run every night to detect important vulnerabilities which are introduced by new code.

READY API

For more information, see: *https://smartbear.com/*

Why did we choose READY API as our test automation tool?

- Non technical interface
- JDBC connections
- Composite projects with GIT integration
- Dynamic environments

- Reusable test cases (Libraries)
- Data driven testing
- CI (Jenkins) integration

Ready API examples

As an example project we provide a Ready API example project on the 'Kinderbijslag' studio project. The 'kinderbijslag' project is delivered in the standard of Blueriq as an studio example project.

The explanation and instructions of the Ready API projects as downloadable below will follow soon. (Please notice: The projects only work with Ready API)

R10Kinderbijslag (JAVA).zip

R10NETKinderbijslag.zip

R9Kinderbijslag (JAVA).zip