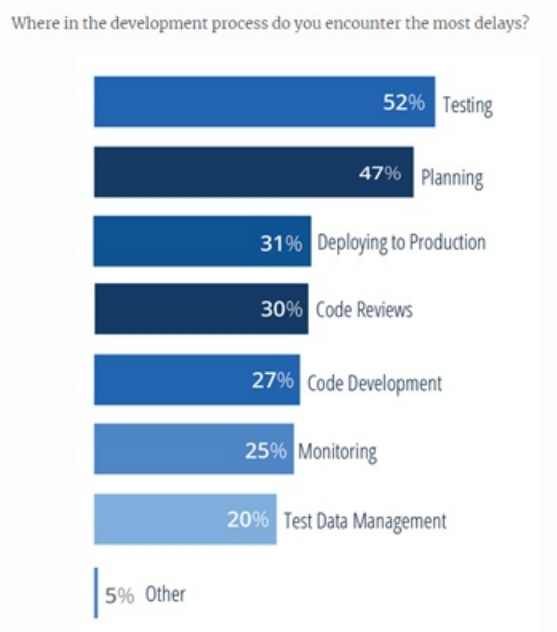
# Development - Cohesive Quality Assurance Process Purpose Statement

12/29/2023 4:25 pm EST

We need to create a comprehensive conformance strategy that encompasses the entire organization to bring unity and common quality standards. It will also bring clarity to the other departments on what the QA role provides. It outlines how we document, execute, report, and resolve discovered defects. Because most of our testing is manual and we have limited resources, we need a strategy that allows us to focus our testing in the most appropriate way driven ideally by documented statistical evidence. There is too much uncertainty on how QA is "testing" that leaves room for the statement of "QA should have caught this". We want to make it so that only if a test cases exists for that scenario and was part of the executed test plan, should we have caught it.

Additionally, we want to find a way to remove the "QA Bottleneck". In our industry, QA testing is still perceived to be the most common source of delay. Everyone wants us to code and release and having to "stop" and test in between feels like a delay to them. [5]



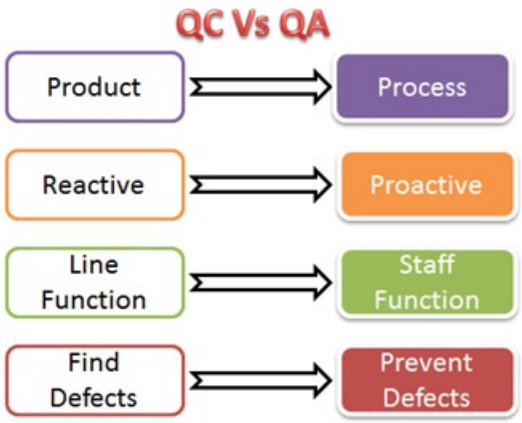

Some of the reasons for this given in the reference article align very closely with out process here at Bold Group. I would like to tackle several of these to try to reduce the bottleneck.

- Test for risk – driven by data
- Reduce Manual testing for regression tests
- Dedicated test environments (document what we have for awareness)

## Difference between Quality Control and Quality Assurance?[1]

Sometimes, QC is confused with the QA. Quality control is to examine the product or service and check for the result.

Quality assurance is to examine the processes and make changes to the processes which led to the end-product.




**I think we have focused on QC and not found a company-wide strategy to start doing QA.**

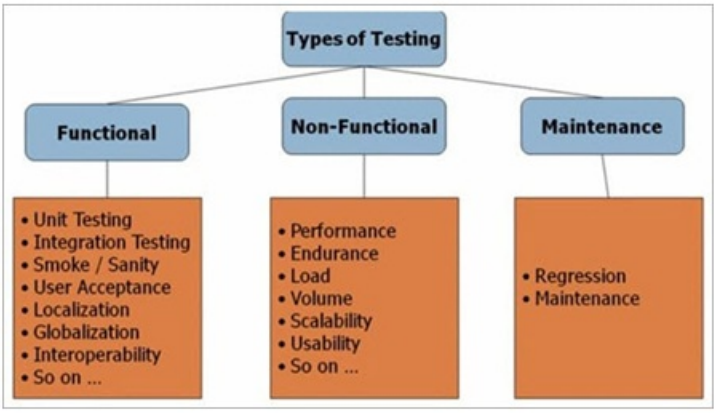
## Best practices for Quality Assurance: [1]

- Create a Robust Testing Environment
- Select release criteria carefully
- Apply automated testing to high-risk areas to save money. It helps to fasten the entire process.
- Allocate Time Appropriately for each process
- It is important to prioritize bugs fixes based on software usage and risk
- Form dedicated security and performance testing team
- Simulate customer accounts similar to a production environment

We need to define and document what each of these means to our specific product. Then we can look at how we will monitory and track our progress on each of these. Finally we can used this to drive decision making when we implement specific QA Testing Methodologies backed by a top level strategy that everyone can follow.

## QA Testing Methodologies/Areas: [2,3]

- Non-Functional Testing
- Functional Testing
- Maintenance




# QA Basic Building Blocks

- Test Strategy
- Test Plans
- Test Cases
- Test Tooling
- Test Setup and Data Integrity
- Test Reporting

https://www.softwaretestinghelp.com/difference-between-test-plan-test-strategy-test-case-test-script-test-scenario-and-test-condition/

# Test Strategy

Test strategy must be a separate document that describes the “why” we are testing in the manner we are. It is typical static but can change with business requirements. Some examples of what should be covered are:

- Scenario testing
- Interface/API testing
- Product area focus testing
- Regression testing
- Smoke testing [4]
- Load/Performance Testing
- Security/Penetration Testing
- Compatibility Testing
- UX/Usability Testing

# Test Plan

Test plans get down to the how, when, and who for testing specific things. This should be established at the project level.

- Project reference
- Project area
- Schedule
- Tester
- List of test cases

## Test Cases

We must move towards having documented, extract step test cases in a consistent format across the company. This is so that new QA hires can quickly start working AND so that we can clearly communicate with the rest of the company what we tested and how we tested it.

A test case must include:

- Test Case ID
- Description
- Steps to follow
- Expected results
- Status
- Test Environment
- Source Work Item link

## Test Tooling

Test tooling is driven largely by what you are testing. Web products, API, and native clients all may require something different. We would like to build a definitive document listing out our chosen test tooling and what should be used with each product type. This will also feed into how the tools tie into DevOps and our overall SDLC.

## Test Systems and Environments

Each product will need to have defined systems and environments. We would like to document what test systems are available, what environments they support, which integrations or other unique setup they have, and how they fit into the overall testing strategy. The end goal is to have a documented way to clearly explain to the rest of the company what we can test and what we cannot. Also, it drives which test environments are used in each test case. I would like to get away from each QA person testing everything on one or two “personal” systems and move to a more focused environment approach.

## Automated Testing

We want to have defined automated test strategy/standards for specific toolsets. This should fit what we are trying to achieve for reporting and integration back into DevOps and our SDLC. We want to pick and document the ones that we can integration into our build and release pipelines. Automated testing must cause the pipeline to fail on an error so that we can immediately alert someone to a problem. We can determine a lighter test that can be run on general builds and

then deeper test that run for releases as a way to balance speed vs coverage.

## Test Setup and Data Integrity

In many of our products, the issues we find are related to very specific data sets and configurations from various customers. These make it very difficult to build repeatable base level tests. We must get to a point where each test case for these specific scenarios can be tested repeatably. The test case should reference defined data sets and specific testing systems where the necessary setup has been done. This means we need to build and maintain a large master data set for testing. We should have a "stock" setup/configuration and then each test case that deviates should change the setting/config to the necessary one AND change it back to stock at the conclusion. There can be static setup such as building out a dedicated customer account to make running the test faster each time. But we do need to make sure that the data set returns to a known "base" at the conclusion of each test run. Otherwise, we will have unintended variation from one run to the next is a different test modifies a configuration that happens to impact our test. This rule of returning to a know data set applies to manual testing as well as automated testing.

## Test Reports and Metrics

Running tests without any way to report the results provides only a "feeling" of quality inside the development organization. We are not able to show the other internal departments or external customers how our efforts are truly resulting in a higher quality product. We also are not able to make decisions on issues from a data driven perspective. It leaves it to a perceived quality rather than an assured one.

## Report Data Analysis

We must create a way to report on several different targets. One is reporting on QA testing done in a Sprint Iteration or from our KanBan board work. This is more focused on the direct test cases and their results in line with active development.

The second is reporting done for Release Cycles. This one is focused more on test plans, product coverage, and historical metrics for decision making goals.

We need to define a way of doing these reports that allow the underlying tests to be reused easily. This means all the data from testing must be present in some form.

- Product Name
- Product Area (this needs to be a defined maintained list)
- Sprint Iteration
- Release build number
- Test Result
- Test Date
- Test Identifier

## Test Failure Process

Testing successes are a great sign and we want to show these through good reporting. But we also need to consider

what should happen when a test fails. This too should be governed by a process so that they can be accounted for and handled consistently through the organization. This should also be reportable for many reasons. We need to give the QA person credit for work done, we need to be able to determine if developers are doing an adequate job testing before sending to QA, we need to identify previously unknown dependencies and/or data issues, etc.

Again, we have to address two different scenarios: planned work and release cycle testing. With Azure DevOps, we will have to treat these slightly differently.

Planned work will be managed by adding Tasks (may create a new work item type) to the work item being tested. This should have the following information:

- Tag of "QA Test Failed"
- QA tester
- Date failed tested
- Date resolved
- Reason it was missed in the beginning (dev responsible)

Release cycle work will be managed by adding a new Bug. This bug will have the following:

- Tag of "Release failure"
- Found in Build populated with release candidate
- Severity set properly
- Priority set properly
- Quality Escapes reason set appropriately
- Blame assigned if known.

We need to start analyzing and understanding where our highest problems areas are coming from and what to do about them in an intentional fashion.

# How are we doing?

## CMMI level

The Capability Maturity Model Integrated (CMMI) is a process improvement approach developed specially for software process improvement. It is based on the process maturity framework and used as a general aid in business processes in the Software Industry. This model is highly regarded and widely used in Software Development Organizations.

CMMI has 5 levels. An organization is certified at CMMI level 1 to 5 based on the maturity of their Quality Assurance Mechanisms.

- Level 1 - Initial:In this stage the quality environment is unstable. Simply, no processes have been followed or documented
- Level 2 - Repeatable: Some processes are followed which are repeatable. This level ensures processes are followed at the project level.

- Level 3 - Defined: Set of processes are defined and documented at the organizational level. Those defined processes are subject to some degree of improvement.
- Level 4 - Managed:This level uses process metrics and effectively controls the processes that are followed.
- Level 5 - Optimizing:This level focuses on the continuous improvements of the processes through learning & innovation.

## Test Maturity Model (TMM):

This model assesses the maturity of processes in a Testing Environment. Even this model has 5 levels, defined below-

- Level 1 - Initial: There is no quality standard followed for testing processes and only ad-hoc methods are used at this level
- Level 2 - Definition:Defined process. Preparation of test strategy, plans, test cases are done.
- Level 3 - Integration:Testing is carried out throughout the software development lifecycle (SDLC) - which is nothing but integration with the development activities, E.g., V- Model.
- Level 4 -Management and Measurement: Review of requirements and designs takes place at this level and criteria has been set for each level of testing
- Level 5 - Optimization:Many preventive techniques are used for testing processes, and tool support(Automation) is used to improve the testing standards and processes.

TO DO: (Define why we are each level, concrete steps to get to the next level)

# References:


[1] https://www.guru99.com/all-about-quality-assurance.html

[2] https://www.softwaretestinghelp.com/what-is-actual-testing-process-in-practical-or-company-environment/

[3] https://mycrowd.com/blog/what-qa-testing-methodologies-are-there/

[ 4] https://www.guru99.com/smoke-testing.html

[5] https://thenewstack.io/why-software-testing-remains-a-bottleneck/