Quality Strategy Template

Version 1.0 June 1, 2022

Project Overview

Strategy Scope & Objectives

Quality Definition & Metrics
- Definition
- Metrics

Functional Quality Considerations

Development Level Quality Approaches
- Quality in the CI Pipeline

Organization & Resourcing

Environments, Tools & Frameworks
- Environments
- Tools
- Frameworks

Assumption & Constraints

Issues & Risks

Decisions

Project Overview

Define what the project intentions are as well as any information pertaining to third parties that it intends to interact with.

- For projects, likely a link to a project overview that already exists

Strategy Scope & Objectives

Define the purpose of this strategy document. Is it intended to cover part of the system? The whole system? Is it an in depth look or a general overview of approaches the team is considering?

> Example:
>
> The purpose of this testing strategy is to break down our approach to performance and scalability testing. We will be looking to add in tracking and insights into the performance of the system in our production environment, while also enabling the teams to test these areas early and often. In addition to this, we'll start to use predictive analysis in order to better plan for our customers future needs based on trends in the current usage. We are looking to make performance and scalability something that we are thinking about during every code change and make it easy to test against prior to release.

Quality Definition & Metrics

Definition

Your team should work together to define what "quality" means for your specific project. This will help to align your team on what approaches you will need to take in order to achieve that set quality.

> Example:
>
> Quality in our project will look like adoption of our product by a minimum of 3 customers, with a defect rate of less than 5 customer reported and less than 10 monitoring reported defects per sprint.

Metrics

This section should describe which metrics your team will be using to determine if you are meeting your definition of quality. It should also describe how you plan to measure those metrics.

Customer reported defect rate per sprint

- Running a script against JIRA each sprint to get the total number of defects reported by support as well as additional information on those defects and putting that into a CSV
- Aspects to track:
    - Priority
    - Severity
    - Report Type
    - Resolved?

Customer usage metrics by endpoint

- Using our monitoring tool we will track how many calls are made to each of our API endpoints
- Aspects to track:
    - # of successful calls made
    - # of unsuccessful calls made
    - Total # of calls made
    - Trend vs last month

Functional Quality Considerations

This section outlines requirements that need to be met by the end of the project in order to uphold quality for our users. These will generally become tickets within a project for things that need to be done, or processes that need to be adhered to.

Common Considerations:

- Resiliency & Disaster Recovery
- Security
- Monitoring & Alerting
- Performance, Load & Scalability
- Documentation
- Troubleshooting Steps

Example:

> **Disaster Recovery**
>
> We need to have a plan in place for how we will handle disaster recovery. This needs to have explicit steps we will take, or processes that will be done in order to allow for this to be done in production.

Development Level Quality Approaches

This section will outline the approaches you plan to take to achieve the quality your team has defined for the project on a per card/feature basis. This section should also define where in the process each approach fits into our pipeline.

It's great to frame this by asking questions such as: Do we need UI tests? Do we need performance tests? How much manual testing effort do we anticipate?

Common Approaches:

- Unit testing
- Integration testing
- Performance testing
- Scalability testing
- Monitoring
- Developer testing
- MR Reviews
- MR testing
- Regression testing

Example:

> **Unit & Integration Testing per MR**
>
> Due to the nature of our approach and processes we rely heavily on our automation to catch defects before production. We know that unit testing only covers our core paths and not the interactions we have with third parties. As such, unit testing and integration testing as required in our definition of done for any piece of code committed.

- Must include unit tests to cover the core functionality of your change
- Must include or already have integration tests for any third party reliant on your code change
- Use mocking for integration testing to reduce the complexity of running our tests

Quality in the CI Pipeline

Define your CI pipeline and map each of the approaches defined above to respective pieces.

Requirements definition	Grooming	Development	Merge Review	Ready to deploy	In Production
	Define key integration tests	Unit & Integration testing created	Run all automated tests	Manual testing	Monitoring in Production

Organization & Resourcing

While working on a project you may find it useful to clearly outline the organization of the team, the resources you have access to (i.e. how often can you consult with devops?). It will also be helpful to define how your product interacts with those around it for clarity on things like what sort of integration tests you may require and which other teams or companies you may need to consult or interact with.

Environments, Tools & Frameworks

This section allows the team to outline what types of environments, infrastructure and tools will be used to host, run, and test the code. You may want to highlight which environments you have access to and what each environment should be used for.

Environments

This section lets you highlight where the code is running. You can use it to highlight what environments you have access to and what each environment should be used for.

Each of these sections needs to explain when that environment should be used for general testing, how to get setup with general testing in that environment and the limitations of each environment.

Examples:

Local

1. What is it used for?
2. How to get setup in this environment?
3. What data is available in this environment? How do I access it?
4. What are the limitations of the environment

Stage

1. What is it used for?
2. How to get setup in this environment?
3. What data is available in this environment? How do I access it?
4. What are the limitations of the environment

Production

1. What is it used for?
2. How to get setup in this environment?

3. What data is available in this environment? How do I access it?

4. What are the limitations of the environment

Tools

This section highlights the different tools (internal and external) you may be leveraging as part of your work both in the development process. The tools used in the development process should outline how they will be tested.

Example:

GraphQL

Our team will be using graphQL as our query language to our API. We will manually check our queries for accuracy.

Kafka

Our team will be using Kafka in order to do publish and subscription messaging between components. We will test it through unit testing.

Frameworks

This section details what testing frameworks you will be leveraging as part of your testing process.

Example:

Cypress

We will be using cypress as our E2E testing framework. We will be using these tests to cover the core use cases in both Chrome and Firefox. See this page to get setup - JavaScript End to End Testing Framework

Assumption & Constraints

This section allows the team to highlight some of the assumptions and constraints that have been made or that influence the project at all. If you are integrating with a third party for example you may make the assumption that their API behaves in a particular way or that it will be published in time for your launch.

Example:

> **Assumptions**
>
> 1. The third party API that we are integrating with will be completed by January, 15 2022
>
> **Constraints**
>
> 1. We must use the existing design framework for creation of all components

Issues & Risks

All identified risks and issues that arise should be documented as part of the development process here.

Example:

> **Issues**
>
> 1. Log4J has a security vulnerability in it and rely on that as part of our repo - Resolved by updating libraries
>
> **Risks**
>
> 1. If our service breaks in production users cannot log into the system - Mitigated slightly by alerting in place for issues in production

Decisions

If at any time a crucial decision is made regarding the approaches to quality of your project you should list them here.

Example:

1. Moved away from Selenium - due to the length of setup time and decided to go with Cypress instead for our end to end testing.