Software testing for continuous delivery

Learn the benefits of software testing, and its role in continuous delivery.

What is software testing?

Software testing is an organizational process within software development in which business-critical software is verified for correctness, quality, and performance. Software testing is used to ensure that expected business systems and product features behave correctly as expected.

Software testing may either be a manual or an automated process.

- **Manual software testing** is led by a team or individual who will manually operate a software product and ensure it behaves as expected.
- **Automated software testing** is composed of many different tools which have varying capabilities, ranging from isolated code correctness checks to simulating a full human-driven manual testing experience.

READ ON BELOW

Software testing articles

ARTICLE

The different types of testing in Software

Compare different types of software testing, such as unit testing, integration testing, functional testing, acceptance testing, and more!

ARTICLE

Exploratory Testing

Learn what is exploratory testing and its history. Discover the pros and cons of exploratory testing and when to best use it.

ARTICLE

Introduction to Code Coverage

Learn how to get started with code coverage, find the right tool, and how to calculate it.

TUTORIAL

Integration Testing Tutorial

Learn how to run integration tests in this tutorial with Bitbucket Pipelines.

Try this tutorial →

[CONTINUED]

Benefits of software testing

Software testing will save an organization time and money by reducing software development and maintenance costs. Software testing builds stability guarantees into the development of new features. Testing ensures that a feature is working as expected and users are not encountering bugs.

Development time on new features is reduced by specifying a set of test cases that the new feature must match to be considered complete and deliverable. This gives developers a fixed target to work towards enabling more accurate timeline estimates and lowering the introduction of new bugs. Once these test cases are in place the overall maintenance costs are lowered. The tests can be run against an already delivered feature to ensure that it still behaves as expected.

Levels of software testing

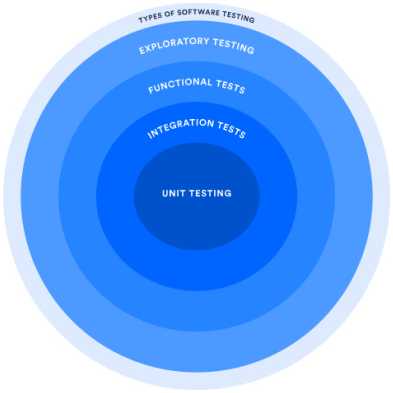
There are several fundamental levels within software testing, each examining the software functionality from a unique vantage point within the development process. Let's take a look at each type of testing in turn and examine its practical use.

Unit testing

The foundational level of software testing is unit testing. Unit testing is the practice of instrumenting input and output correctness checks for individual units of code. The measurement unit, in this case, is standalone code functions or methods.

During unit testing, production code functions are executed in a test environment with simulated input. The output of the function is then compared against expected output for that input. If the output matches the expected the test passes. If not it is a failure. Unit tests are a great way to validate derived data functions.

A hypothetical unit test user story example would be something like: “function 2VAL, Given 2 values x and y always returns x+y”. The unit test would then execute 2VAL with two values and confirm that the output was x+y. Unit tests are great for confirming the correctness of code that operates on monetary values.



Integration testing

When a software test case covers more than one unit, it is considered an integration test. When developing a software test case, the lines between unit tests can quickly evolve into integration tests. Often times a unit test may be developed that operates against a third party code dependency. The dependency itself will not need to be tested and the integration to it will be mocked or faked.

Functional or end-to-end testing

Test cases that simulate a full user-level experience are called functional tests or end-to-end tests. End-to-end tests use tools that simulate real human user behavior. Common steps in an end-to-end test:

- Click this button
- Read this text
- Submit this form

Because of the full experience execution context, end-to-end tests verify correctness across all the layers of a software stack.

Exploratory testing

Exploratory testing is a testing exercise in which testers are assigned a loosely defined task to achieve using the software being tested. This means you can learn a lot about the way people use your product in the wild. Exploratory test sessions can even motivate their users by offering rewards for the most number of issues, best defect, or doing something unexpected with the product.

One of the benefits of exploratory software testing is that anyone can join in to help test because all they need to do is wander about the product in a free form manner. Exploratory testing is not random, yet they aren't scripted like manual tests, either.

Software testing in continuous delivery

Continuous delivery leverages all the aforementioned testing strategies to create a seamless pipeline that automatically delivers completed code tasks. An optimal setup would allow a developer to push recently completed code into the continuous delivery pipeline for evaluation. The pipeline would then run the newly pushed code through the levels of testing. If the code passes the testing, It will be automatically merged and deployed to production. If however, the code fails the tests. The code will be rejected and the developer automatically notified of steps to correct.

Popular established software language development ecosystems have their own subset testing ecosystems. There are many tools available which provide utilities to help instrument and develop testing suites. These tools are usually installed through a package manager specific to the programming language used on the project.

In addition to testing instrumentation, tools for test execution and development are also available. Various test runners can be installed to provide output data from a test suite. A common practice is to measure the “test coverage” throughout a project. A code coverage tool can be used to indicate how much of a code base is adequately covered.

Once a testing suite has been developed and is working correctly on a local project it is generally straightforward to integrate into a CD pipeline. Most hosted CD/CI systems will have guides on how to integrate a testing suite into the pipeline.

How to make testing part of your CD pipeline

A true hands-off, value-add CD pipeline is built around a strong testing foundation. This testing foundation starts with manual test cases which evolve into automated solutions.

Emphasize quality at every step of the pipeline

Everyone–developers, testers, etc.–owns the quality relationship with the customer. Each line of code either makes the customer experience better or worse. The test suite of a CD pipeline is a multi faceted tool for developing high quality and correct code. During the product design phase, the test suite can be kept in mind for pre-emptive considerations on how to develop a feature. The test suite is primarily used to streamline the development process, but can also be executed staging and production environments to guarantee quality there as well.

Empower developers to prove the quality of features

Traditional test methodology holds that testing is a separate process out of step with the developer. Developer absence from quality assurance encourages a lack of customer empathy from the development team. Furthermore, the lack of developer involvement in quality allows issues to fester in the code base longer making them more expensive to fix. This methodology is also expensive in organizational employee cost as it encourages hiring a separate QA team to take responsibility.

Continuous delivery promotes developer awareness and empathy with the end user experience. Developers are tasked with delivering test coverage for the features they produce and overseeing them from development to production environments. This gives developers an opportunity to own and prove the quality of a feature.

Build in customer feedback

Continuous delivery enables rapid deployment and updates to a software project. This allows for immediate incorporation of customer feedback into an next release. In the event of a user reported issue, the CD pipeline test suite can be consulted to narrow down the scope of possible issue vectors. Development and test teams that quickly respond to customer feedback are more successful.

Want create your own continuous delivery environment? We'll help you get you started.

Build a solid software testing strategy

When devising a software testing strategy its best to keep the overall product, user, and business strategies in mind. Considerations will need to be made on what the most high value test coverage targets are.

In an ideal world, a software project would strive for 100% test coverage guaranteeing the code is bug-free and works as expected. Unfortunately in the real business world, with timelines and budget constraints, this is not so realistic.

Different testing strategies should be considered depending on the type of deliverable software as well. If the software is a GUI driven application, high level end-to-end tests will be highly valuable. Headless UI free software projects will forgo end-to-end testing and value highly from unit tests.

A general overall strategy for GUI-based user applications is as follows.

1. Instrument end-to-end tests on all the core user flows, login, signup, checkout, etc
2. Instrument unit tests on all data sensitive code functions like monetary transaction tools
3. Instrument integration tests for any points of 3rd party integration to ensure data is flowing to the 3rd and any errors are being propagated correctly

Improve your software testing with continuous delivery

Striving for a continuous delivery workflow has many business benefits. The organizational costs of hiring and managing separate teams for Quality Assurance, Release management, and Test engineering roles can be drastically cut with a commitment to a CD workflow.

Continuous delivery promotes an overall higher level of product quality than that of traditional QA testing workflows. CD testing encourages developers to take ownership and stake in the end user experience and the quality of the features they put out. CD lays a framework that makes it easier for company wide focus and discussion on release quality.

Implementing a robust software testing strategy is the foundation of continuous delivery, and automation is the key to a successful continuous delivery pipeline.

Are you ready to beef up your software testing? Learn more about testing in a CD environment.

CLAIRE MAYNARD

Claire is an Atlassian marketing veteran who's worked across growth, performance, and product marketing throughout her tenure. Currently she drives brand, content, and go-to-marketing strategy for Confluence Cloud. Outside work, you can find Claire surfing, running, or trying out new restaurants in San Francisco or new cities across the globe.

TUTORIAL

Integration Testing Tutorial

Learn how to run integration tests in this tutorial with Bitbucket Pipelines.

Try this tutorial →

UP NEXT

The different types of testing in Software

Compare different types of software testing, such as unit testing, integration testing, functional testing, acceptance testing, and more!

Read this article →

CI/CD Topics

- Continuous Delivery
- Continuous Integration
- Continuous Deployment
- Pipelines
- Software Testing
- Microservices
- Tutorials

Sign up for more CI/CD articles and tutorials.

Email

email@example.com

Subscribe

Up Next

Automated software testing for continuous delivery →