Testing

That thing you don’t do enough of

Google told us so

MOVE
FAST AND
BREAK
THINGS

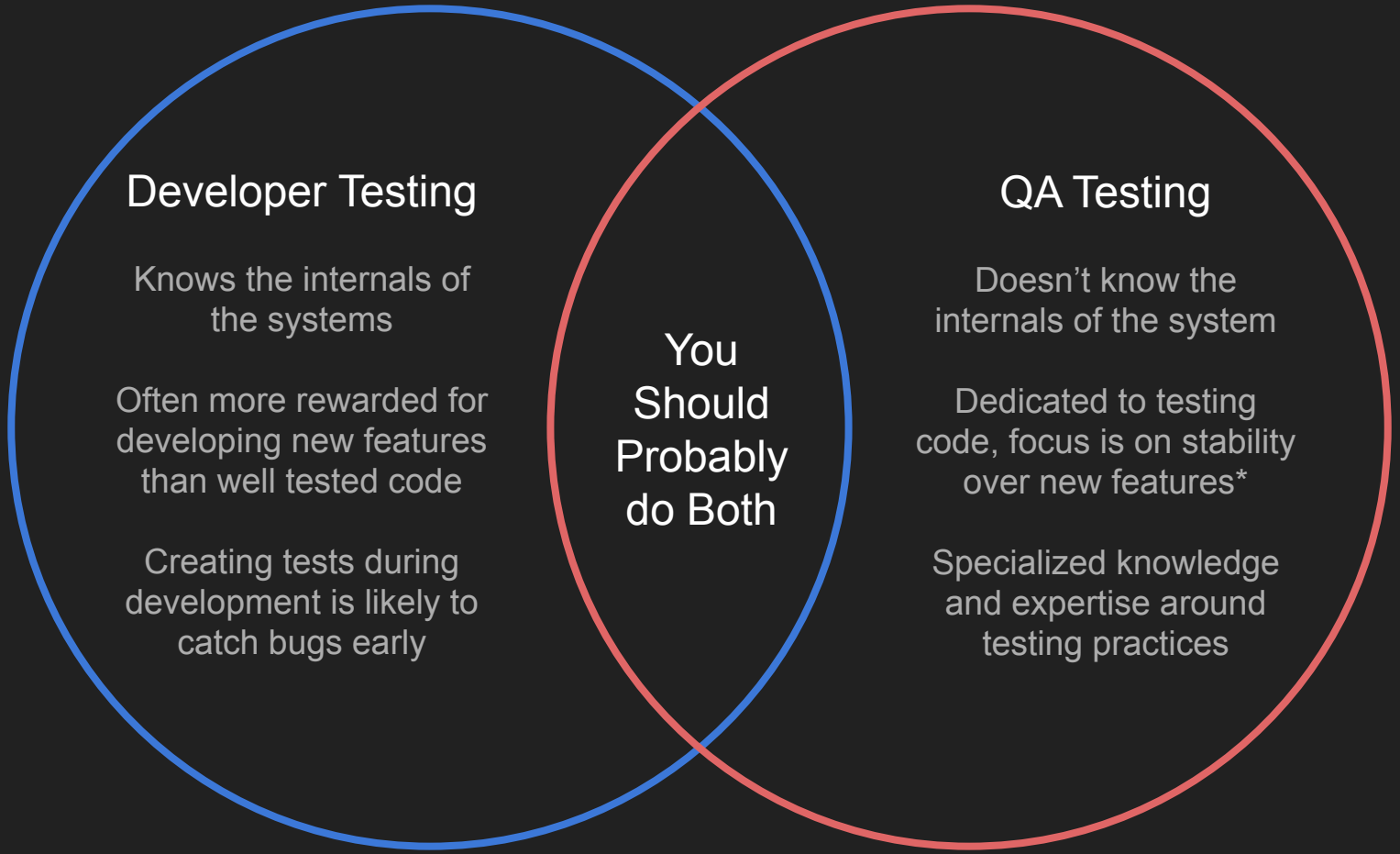
MOVE FAST WITH STABLE INFRA

Testing Methodologies

Typical: QA vs. Dev

Generally, at large companies there are two teams who sit in pseudo-opposition. Developers, whose primary purpose is to develop new products and features for end users, and Quality Assurance (QA), whose primary purpose is to make sure that new products and features are bug free.

Typically, developers create a new features which goes to QA for validation. If any bugs are found during testing, then the feature is sent back to the developer to be fixed. This cycle continues until the feature passes all tests and is approved to be released (often known as being “shipped”).


Developer Testing
Knows the internals of the systems
Often more rewarded for developing new features than well tested code
Creating tests during development is likely to catch bugs early
You Should Probably do Both
QA Testing
Doesn't know the internals of the system
Dedicated to testing code, focus is on stability over new features*
Specialized knowledge and expertise around testing practices

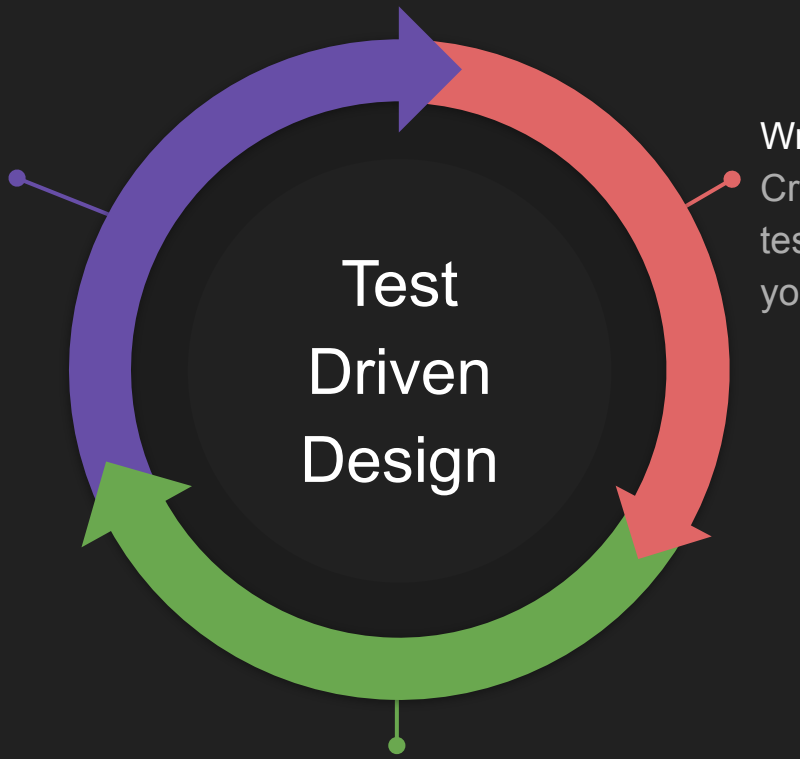
Test Driven Design

It is an evolutionary approach to development which combines test-first development where you write a test before writing just enough production code to fulfill that test and refactoring.

It is one way to think through your requirements or design before your write your functional code (implying that TDD is both an important agile requirements and agile design technique).

Another view is that TDD is a programming technique, the goal of TDD is to write clean code that works (or at least passes your tests).

This method is very effective for personal projects, and can be difficult to do if you have many developers working on the same sections of code.



Eliminate Redundancy
Remove any redundant code and refactor the system to make it more cohesive

Write a Test that Fails
Create one or a small number of tests for a piece of the feature you are creating

Make the Code Work
Write just enough code so that you pass the tests that you've just created

Function Testing

Unit Testing

Testing of an individual software component or module is termed as Unit Testing. It is typically done by the programmer and not by testers, as it requires a detailed knowledge of the internal program design and code. It may also require developing test driver modules or test harnesses.

Integration/System Testing

Testing of all integrated modules to verify the combined functionality after integration is termed as Integration Testing. Modules are typically code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to full software, client/server, and distributed systems.

Graphical User Interface (GUI) Testing

The objective of this GUI testing is to validate the GUI as per the business requirement. The expected GUI of the application is mentioned in the Detailed Design Document and GUI mockup screens developed by the User Interface (UI), User Experience (UX) and Product Management (PM) teams.

The GUI testing includes the size of the buttons and input field present on the screen, alignment of all text, tables and content in the tables. This can bleed into compatibility testing when these are performed on different browsers/devices.

This can also fall into accessibility testing, validating that colors are appropriate for those who are colorblind, elements register correctly with screen readers, etc.

Smoke Testing

Whenever a new build is provided by the development team then the software testing team validates the build and ensures that no major issue exists. The testing team ensures that build is stable and a detailed level of testing is carried out further. Smoke Testing checks that no showstopper defect exists in the build which will prevent the testing team to test the application in detail.

If testers find that the major critical functionality is broken down at the initial stage itself then testing team can reject the build and inform accordingly to the development team.

This is a fairly adversarial form of testing, which puts Quality Assurance (QA) at odds with the development team (author's opinion)

Alpha/Beta Testing

It is a formal type of software testing which is carried out by a subset of customers before a more open release. Alpha/Beta testing is carried out to ensure that there are no major failures in the software or product and it satisfies the business requirements from an end-user perspective. It opens the software to real-world scale, unknown user types, and unexpected user behaviors. Beta testing is successful when the customer accepts the software.

This is generally the last major form of testing for a piece of software that will be used by a general audience.

Acceptance Testing

An acceptance test is performed by the client and verifies whether the end to end flow of the system is as per the business requirements or not and if it is as per the needs of the end user. Client accepts the software only when all the features and functionalities work as expected. It is the last phase of the testing, after which the software goes into production. This is also called as User Acceptance Testing (UAT).

This is generally the last major form of testing for a piece of software that is being developed for a specific client or small set of clients.

GUI Testing

Unit Testing	Integration Testing	System Testing	Smoke Testing	Alpha/Beta & Acceptance Testing
Usually performed by the developer creating the code, requires knowledge of the internal workings of the system. Results only ever seen internally.	Usually performed by the development team or QA, requires knowledge of how different components should interact. Results only ever seen internally.	Usually performed by the development team or QA, requires knowledge of how the system is supposed to be used, at what scale, and with what performance. Results only ever seen internally.	Usually performed by QA team, requires knowledge of how the system is supposed to be used, at what scale, and with what performance. Results only ever seen internally.	Performed by the end-user, generates both intended and unintended uses from a “large” number of users. Results are “public”.

Non-function Testing

Performance Testing

Performance Testing is done to check whether the system meets the performance requirements. This can include things like function execution time, network latency, and webpage render times.

Load Testing

It is a type of non-functional testing and the objective of Load testing is to check how much of load or maximum workload a system can handle without any (or an acceptable amount of) performance degradation.

Stress Testing

This testing is done when a system is stressed beyond its specifications in order to check how and when it fails. This is performed under heavy load like putting large number beyond storage capacity, complex database queries, continuous input to the system or database load.

Security Testing

It is performed to check how the software, application, website, etc. is secure from **internal** and **external** threats. This testing includes how much software is secure from the malicious program, viruses and how secure and strong the authorization and authentication processes are. It can also include how strong internal controls, auditing, and logging are against internal threats.

Compatibility Testing

It is performed to see that equivalent (or appropriate) functionality for the software will be executed on different platforms. This can include everything from web pages rendering correctly across different browsers, applications running similarly across different operating systems, and software running correctly across multiple operating system versions.

Reliability/Recovery Testing

Recovery testing determines if the system is able to continue the operation after a disaster. This could include network outages, database failures, or receiving bad data.

Recovery testing is often part of reliability testing, which verifies that software is capable of performing a failure-free operation for a specified period of time in a specified environment.

These types of testing are important for services, especially those who have a Service Level Objective (SLO) as part of a Service Level Agreement (SLA) which dictates how available the service will be before a financial penalty is enforced.

Usability Testing

It is testing for application flow to know how well users can understand and navigate the software. This testing can be done with user focus groups as well as by gathering user metrics and user feedback.

Localization Testing

It is the software testing process for checking the the UI and content are appropriate for the different locations the software will be developed. This could include language, formatting, performance, and content.

Globalization (Internationalization) Testing

It is testing to ensure that the software can function in any culture or locale.

Compliance (Conformance) Testing

It is a kind of an audit which is done on the system to check if all the specified standards are met or not. This type of testing is often done by experts, as the regulations around specific compliance can be complex.

Examples include Family Educational Rights and Privacy Act (FERPA) for schools, Health Insurance Portability and Accountability Act (HIPPA) for health data, Defense Federal Acquisition Regulation Supplement (DFARS) for government data, etc.

Regression Testing

Regression Testing

Regression testing is re-running functional and non-functional tests to ensure that previously developed and tested software still functions correctly after a change. If not, that would be called a regression. Changes that may require regression testing include bug fixes, software enhancements, configuration changes, and even substitution of electronic components.

Testing Infrastructure

Common Functional Testing Frameworks

C++	Google Test
Java	JUnit
Go	go test*
JavaScript	Jest
Python	unittest*

en.wikipedia.org/wiki/List_of_unit_testing_frameworks

Testing Coverage

It is the percentage of code which is exercised by automated tests. Coverage helps to determine how well tests actually test your code, if you have enough tests, and to help maintain test quality over time. Getting to 100% coverage is not always practical or reasonable, as some sections of code (especially failure cases) may be very difficult to exercise. Coverage does not replace code reviews or good programming practices.

Coveralls is a common code coverage system, especially for open source projects.

Continuous Integration (CI)

In this system developers merge their changes back to the main branch as often as possible. These changes are validated by creating a build and running automated tests against it. By doing so, you avoid the integration hell that usually happens when people wait for release day to merge their changes. This system relies heavily on automated testing to validate changes.

Continuous Delivery/Deployment (CD)

Continuous delivery is an extension of continuous integration to make sure that you can release new changes to your customers quickly in a sustainable way. This means automating your delivery process as well as testing so that changes can be deployed through a button press. Continuous deployment goes one step further by removing the button press and automatically deploying new changes.

Puppet and Jenkins are build, test, and deployment automation systems which are typically run as a service internally to a company. These can be highly customized and have highly varied service offerings depending on the company.

Travis CI offers build, test, and (some) deployment automation. This is offered as an online service where Travis CI runs the infrastructure. It is used extensively in open source projects.

Questions?

References

softwaretestinghelp.com/

guru99.com/

agiledata.org/essays/tdd.html

confluence.atlassian.com/clover/about-code-coverage-71599496.html

atlassian.com/continuous-delivery/ci-vs-ci-vs-cd