abstracta

ASSESSING YOUR CONTINUOUS TESTING CAPABILITIES:

TESTING IN A CONTINUOUS DELIVERY WORLD

Improve speed without rushing software out the door.

Testing needs to ¨shift left¨:

Testing is starting to be done by developers more frequently.

QA professionals are still doing manual work, but they're trying to automate the process as well.

Challenge for Testers: not just to be a good tester but also be able to engineer the process and take advantage of advanced automation practices.

OUR MATURITY MODEL

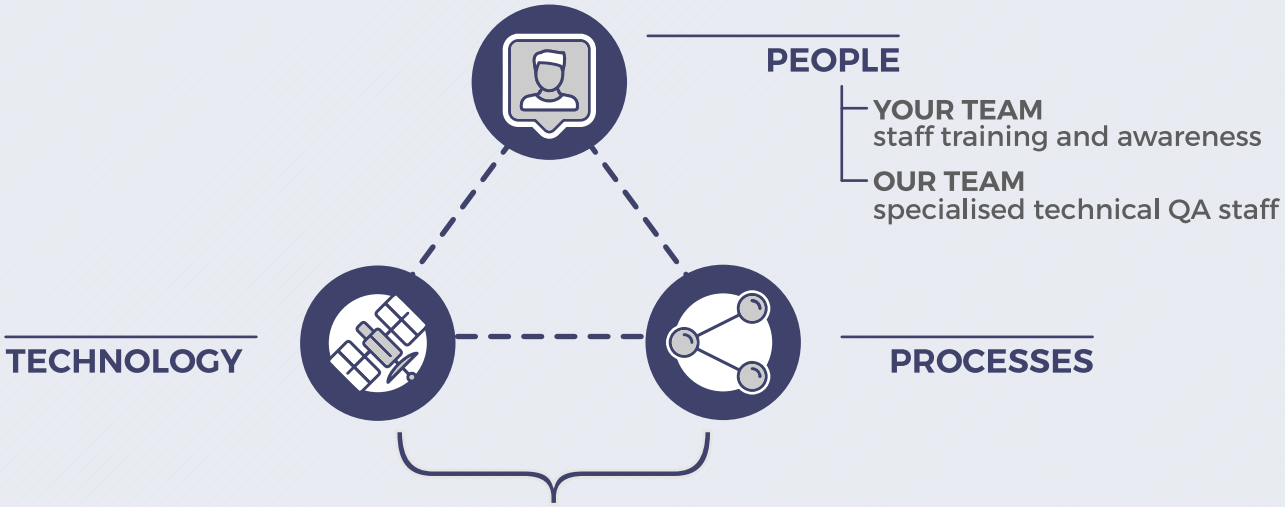
KEY AREAS IN CONTINUOUS TESTING

- Risk Assessment.
- Defect Casual Analysis.
- Code Quality Control.
- Traceability.
- Test Optimization.
- Service Virtualization.

KEY BENEFITS

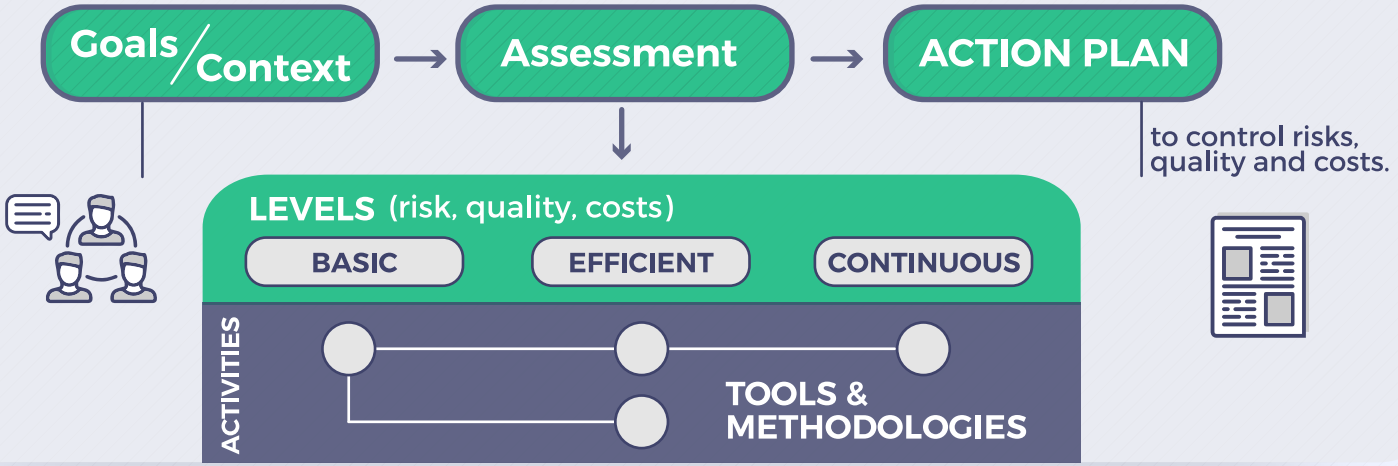
- Focus on the areas that matter
- Determine current gaps in maturity
- Control risks, quality and costs

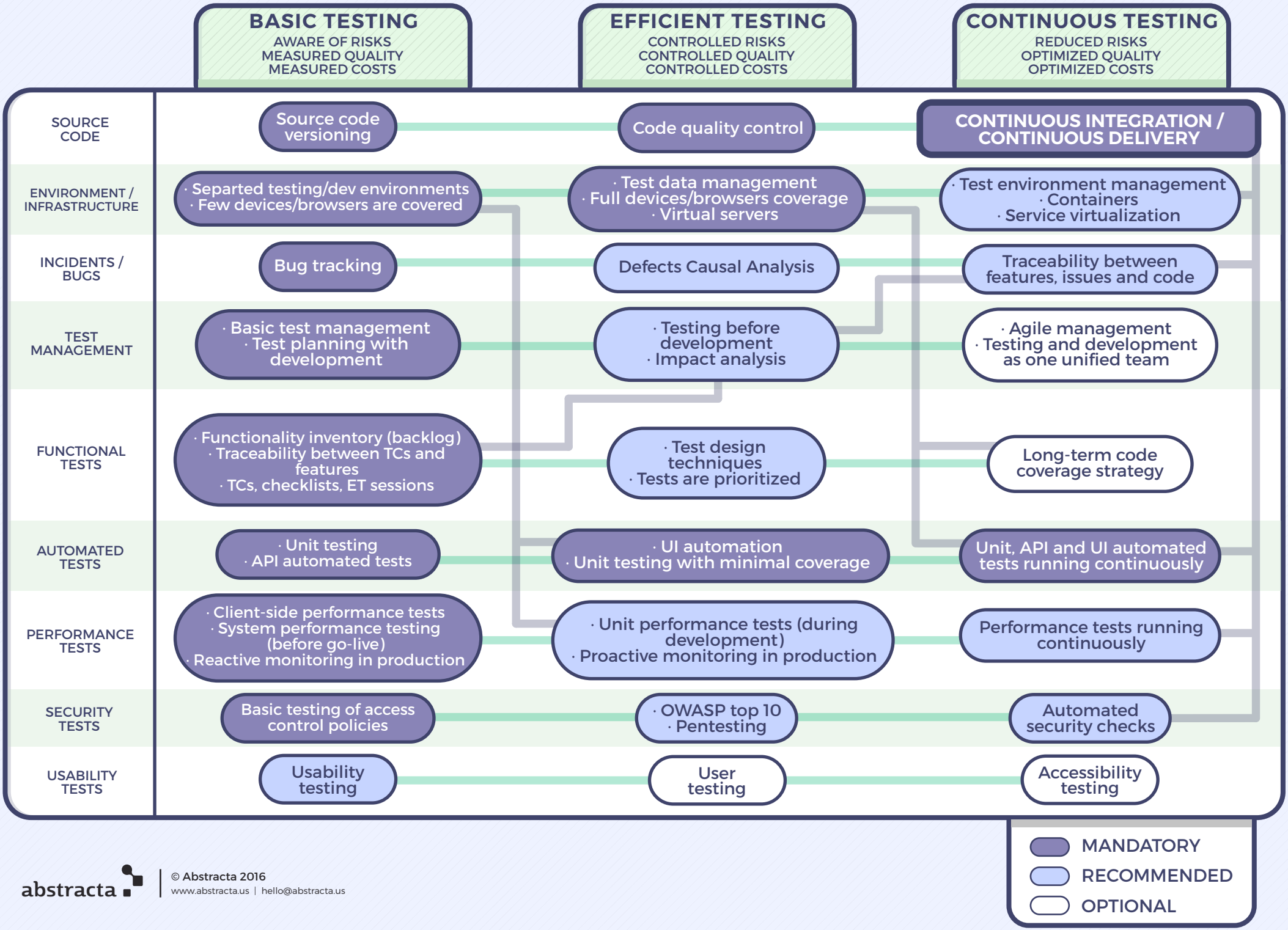
THE THREE PILLARS OF TESTING MATURITY



TESTING MATURITY ASSESSMENT

Abstracta's testing maturity assessment will help you quickly identify where your QA stands today and what activities you need to do to achieve the goal of testing maturity, enhancing your technology and processes.





PAINS TO SOLVE

	BASIC TESTING	EFFICIENT TESTING	CONTINUOUS TESTING
SOURCE CODE	· Pieces of source code get lost. · Not clear what version each client has, which makes it complicated to do fixes in the corresponding code.	· The code has a big technical debt, maintainability problems, poor internal quality, lack of documentation, dead or duplicated code, doesn't follow best practices in design or architecture, complex code (spaghetti), etc.	· Finding bugs and solving issues takes too long. · Integration is complex and costly.
ENVIRONMENT / INFRASTRUCTURE	· Not clear what is in each environment, everyone works in shared environments. · Not sure if we are testing with the latest version.	· Data is overwritten between developers, testers or automated tests. · There are devices that have problems. · Cannot test on all devices.	· Difficult to set up a new environment for a demo, test or whatever is necessary.
INCIDENTS / BUGS	· Bad communication between development and testing. · No knowledge of the state of each incident. · No knowledge of the version in which an incident was fixed.	· No knowledge of how to avoid incidents. · No knowledge of where the incidents come from.	· No knowledge of which feature is affected by a certain bug and to what line of code it relates to. · No traceability within code versions.
TEST MANAGEMENT	· No test cycles defined. · Testing is hard, not business focused, starts late, and takes too long to update a test case. · No knowledge of which incidents each test case corresponds to. · Not clear what needs to be tested or when.	· Testing starts after development, focused on detecting and reporting, not prevention. · When something changes, no knowledge of which test cases need to be executed.	· Gap between development and testing team, not sharing goals.
FUNCTIONAL TESTS	· No record of what has to be tested or with which level of priority. · No evidence of test executions. · No information on the quality status of each version.	· Uncertain about how well the tests are designed. · Not clear what to test first.	· No knowledge of what coverage we should have. · Not enough time to meet the expected coverage.
AUTOMATED TESTS	· Incidents already solved reappear. · Getting feedback after introducing a new change takes too long. · Automated tests take a long time to run. · Automated tests are expensive in terms of maintenance.	· Testers are bored and demotivated, always executing the same tests. · Regression tests are executed manually and take a long time. · Testers make mistakes when doing checkups.	· Fear and uncertainty when releasing a new feature to production.
PERFORMANCE TESTS	· Uncertainty when going live, lack of knowledge about how the system will perform. · No control over production systems or other environments. · No clear methodology to carry out tests that simulate the expected load.	· Performance problems are difficult to solve and are detected very late. · Unable to anticipate problems that occur in production.	· No knowledge of how a new change affects performance.
SECURITY TESTS	· Security breaches, uncontrolled risks or uncertainty concerning how unprotected the users are.	· Security standards are not met.	· No knowledge of how a new change affects security. · Need to release frequent security patches.
USABILITY TESTS	· Users find the system difficult to use. · No evidence that the application is usable.	· Users are resistant to change due their lack of involvement in acceptance testing. · No evidence that the application is user-friendly.	· No evidence that the application is accessible to all.

WHY CONTINUOUS DELIVERY?
CASE STUDY

By using continuous delivery practices, HP LaserJet Firmware team could:

- Reduce overall development costs by ~40%
- Increase programs under development by ~140%
- Reduce development costs per program by 78%
- Increase resources driving innovation by 5x

Source:
Thoughtworks - The Case for Continuous Delivery.

READY TO TAKE THE NEXT STEP?

Contact us at hello@abstracta.us or call us +1 408 757 0005.

abstracta

www.abstracta.us | hello@abstracta.us