NetApp

Agile Testing Quadrants

Ankita Sharma
Software Engineer

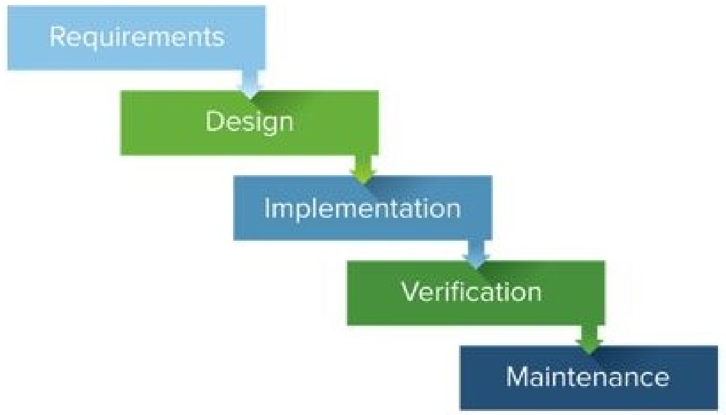
Contents

NetApp

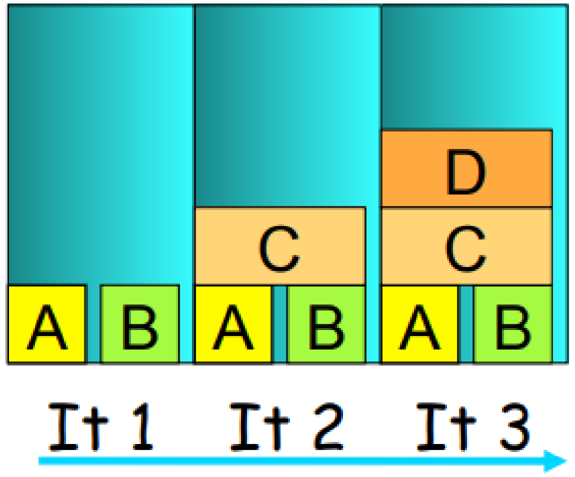
Why do we test?

- Find bugs
- Improve quality of product
- Validate that product meets requirements
  - functional, performance, reliability, security, usability and so on
- Learn about the application
- Guide coding
- Check for doneness
- Manage technical debt

Traditional Testing vs Agile Testing

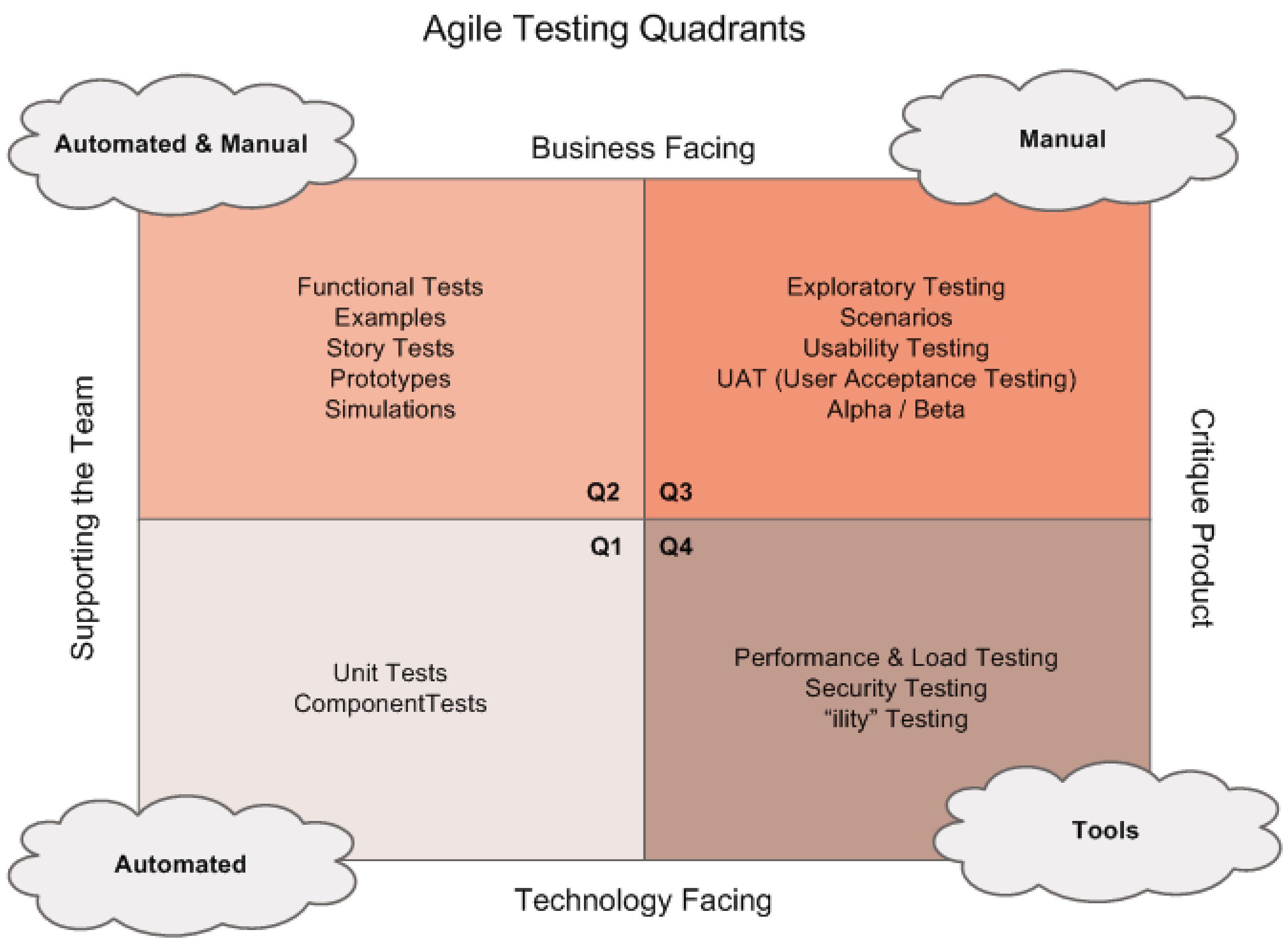


- Testing is done after feature development
- Development and testing teams work in silos
- Not enough time left for testing before release



- Iterative and incremental
- Testers test each increment of coding as soon as it's done
- Makes use of automated tests from the beginning to help speed up development

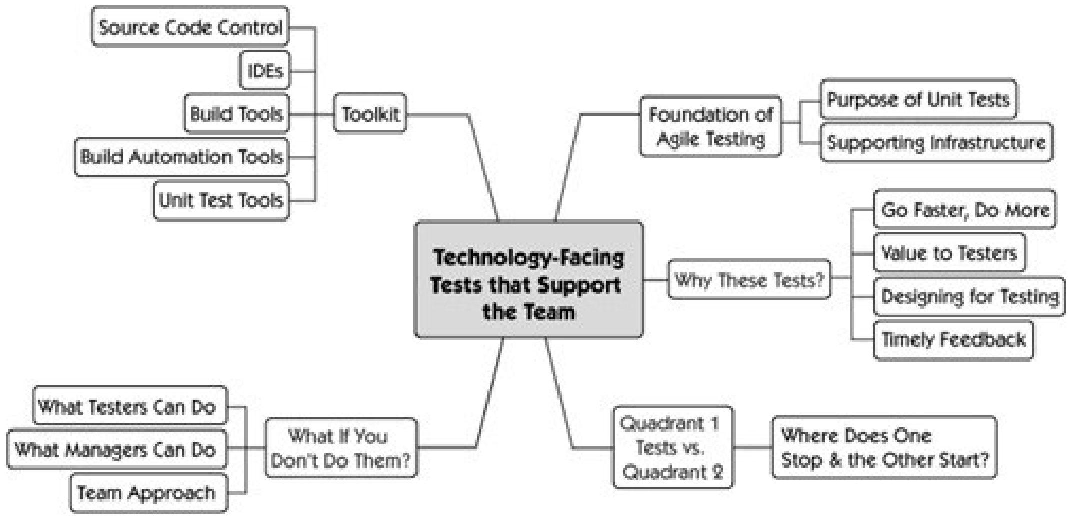
Agile Testing Quadrants



- Quadrants help teams cover and share all facets of product quality
- Programmers should write technology facing tests that support programming - with help from QA
- Testers take responsibility of business facing tests in tandem with customers
- 4th quadrant tests need specialists
- Each quadrant in matrix helps in keeping technical debt to a manageable level

- Some questions/checklist:
  - Are we using unit tests to find right design for application?
  - Do we have automated build process that runs automated unit tests?
  - Are we capturing right examples of desired system behavior?
  - Do business facing tests help deliver a product that matches customer expectations?
  - Do we budget time for exploratory testing?

Technology facing tests



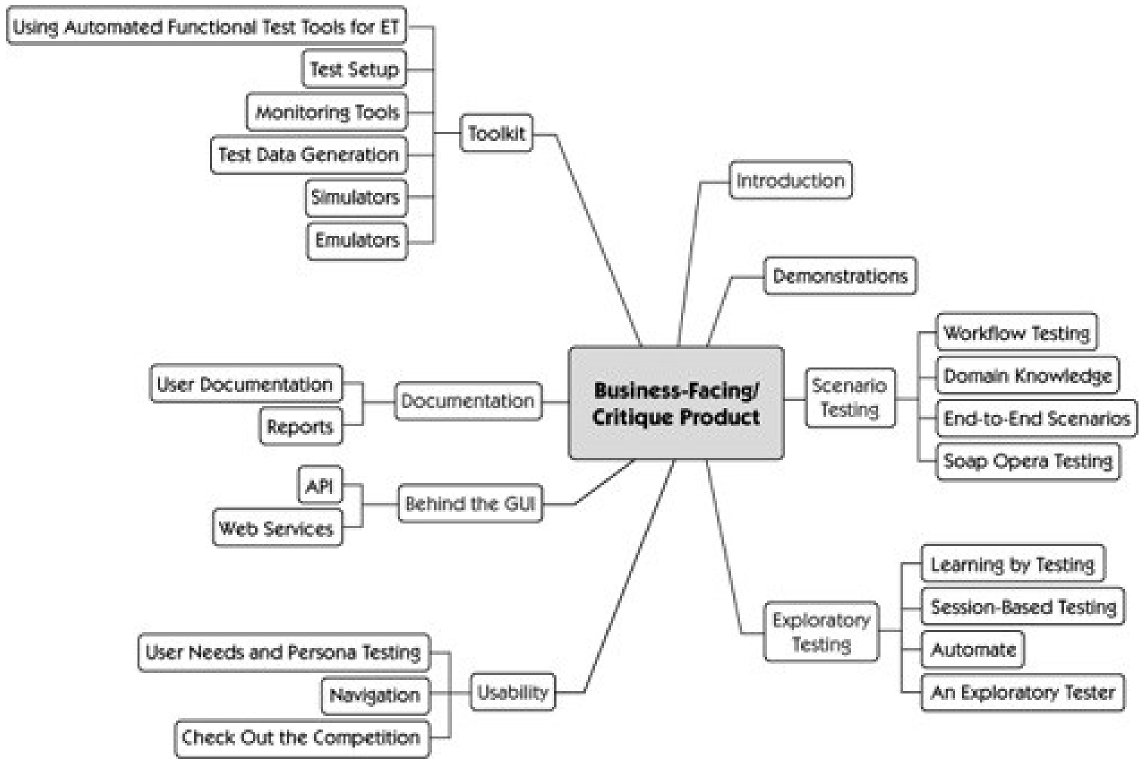
Quadrant 1 – Unit Tests

- They are the foundation of agile development and testing
- Helps the programmer understand what exactly the code needs to do
- Access each layer independently using fake objects . They verify behavior of single objects or methods.
- Source control, configuration management and continuous integration system are required
- A safety net of automated unit and code integration tests enables programmers to refactor frequently
- Testers waste lesser time on finding low level bugs
- Produces more testable code as features are designed with test in mind

Quadrant 2 – Functional automated tests

- These are ‘understandable’ tests
- They address business requirements
- Business facing tests express requirements based on language and format that both customer and development teams can understand
- They are part of automated regression suite so that future development doesn’t intentionally break system behavior
- It is important to start with happy path testing but also test improbable edge case tests

Business facing tests



Quadrant 3

- Exploratory Testing
  - Most of testing here is manual, but it cannot be done unless there are automated tests from quadrant 1 and 2 already present
  - Testing the system end to end while making spot checks to make sure data, status flags, calculations are behaving as expected
  - It is an investigative tool with a sophisticated and thoughtful approach to testing without a script

- Usability tests
  - Includes user needs and persona testing, navigation, behind the GUI and API testing.

Quadrant 4 – ‘ility’ tests

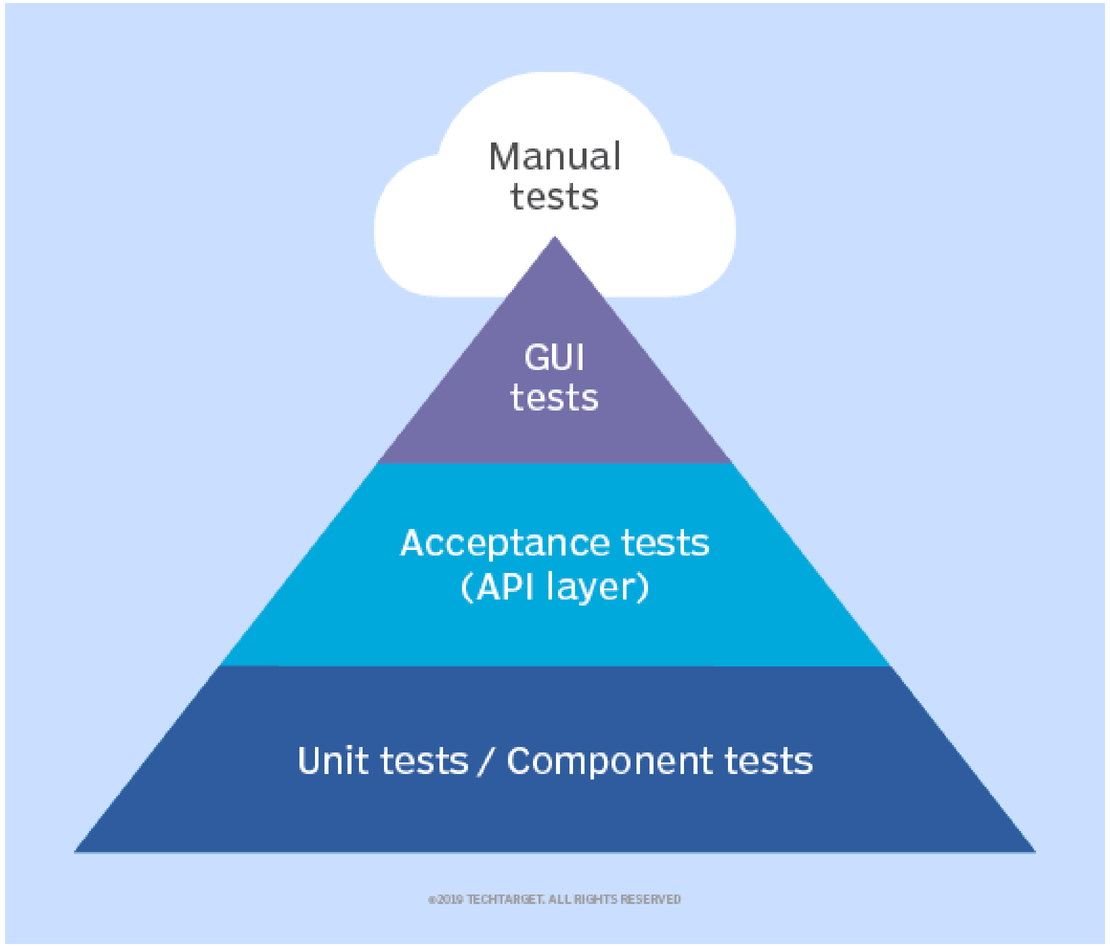
- Security: Security risk-based testing is performed by analyzing architectural risk, attack patterns and misuse cases
- Inter-operabilty: End to end functionality testing between diverse systems
- Compatibility: Testing with different operating systems/browsers/third-party applications that need to work with the product
- Maintainabilty: Development teams develop standards and guidelines they follow for application code, test frameworks and tests. It encourages shared code ownership and is an important factor for automated tests.
- Reliability:
  - Mean time to failure
  - Mean time between failures

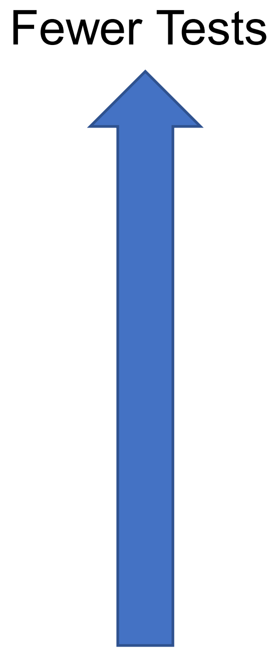
Quadrant 4 – 'ility' tests continued

- Installability: Build should be ready for testing anytime
- Scalability: Verify system remains stable when adding more users/capacity. It's important to test the whole system and not just the application.

- Performance and Load Testing
  - Performance testing is done to help identify bottlenecks in a system or to establish a baseline for the future
  - Load testing evaluates system behavior as more users access the system
  - Stress testing evaluates the robustness of the application under higher-than-expected loads

Test automation pyramid


Manual tests
GUI tests
Acceptance tests (API layer)
Unit tests / Component tests
©2019 TECHTARGET. ALL RIGHTS RESERVED



References

- [1] Crispen L & Gregory J (2009) “Agile Testing: A practical guide for testers and agile teams”. Pearson education, Inc.
- https://smartbear.com/learn/automated-testing/what-is-automated-testing/
- https://lisacrispin.com/downloads/AgileDenverQuadrants3.pdf

NetApp

Questions?