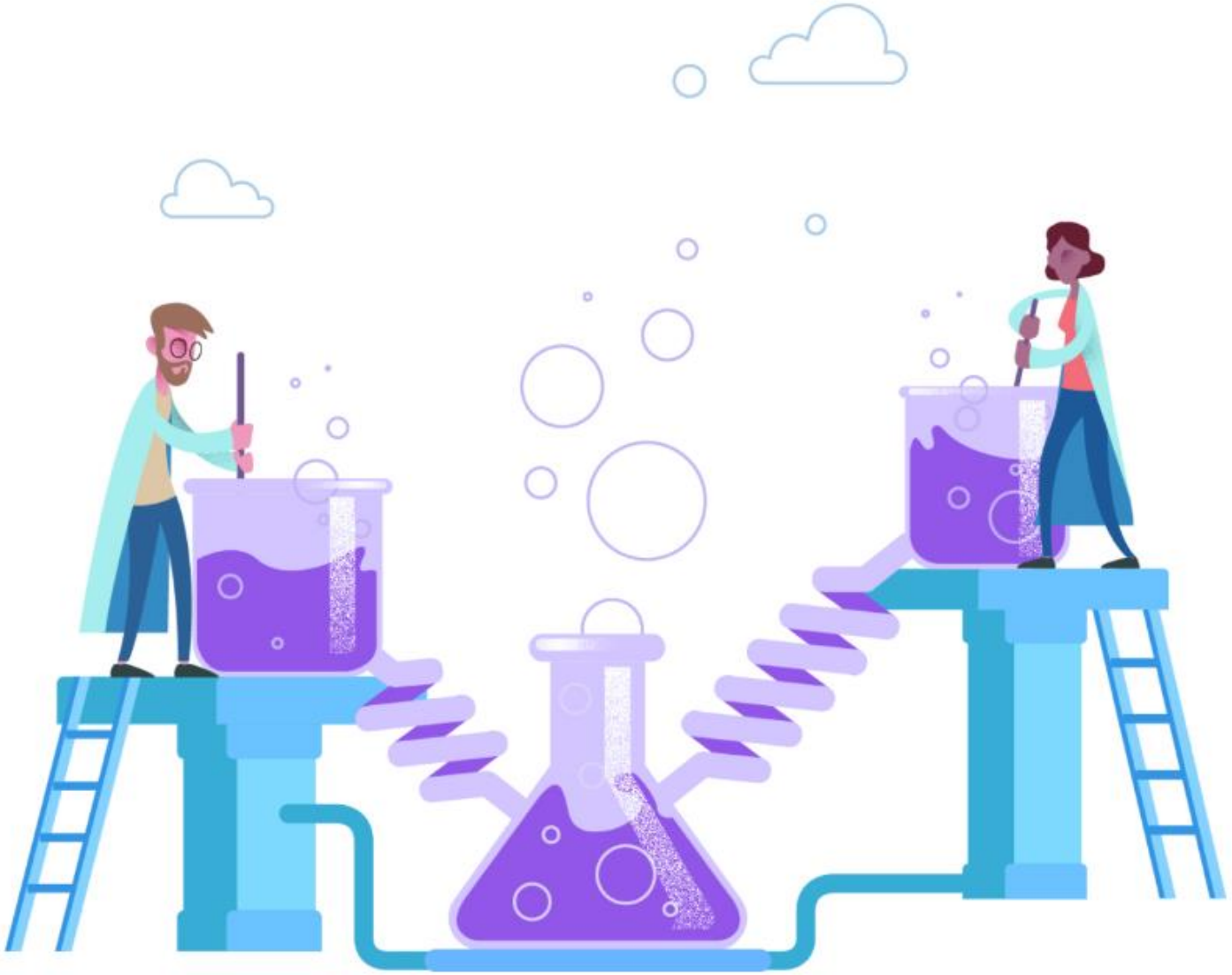
Azure Test Plans



About Testing

Testing is an investigation conducted to provide stakeholders with information about the **quality** of the **software** product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation.

DevOps and Test Automation

To achieve such speed and agility, it is important to automate all the testing processes and configure them to run automatically when the deployment is completed in the QA environment. Specialized automation testing tools and continuous integration tools are used to achieve this integration. This also necessitates the building of a mature automation testing framework through which one can quickly script new test cases.

Types of Tests

1. Unit tests

An automated, simple, repeatable, maintained test (piece of code) that invokes a unit of work in the system and then checks a single assumption about the behavior of that unit of work.

2. Integration tests

An integration test validates two or more dependent software modules (or components) as a group in multiple ways. In other words, an integration test validates that software modules work properly when they are integrated with other modules.

3. Automated UI tests

Automated tests conducted through the user interface to test components together in scenarios. These tests typically drive your application through its user interface (UI) and include functional testing of the UI controls.

Types of Tests

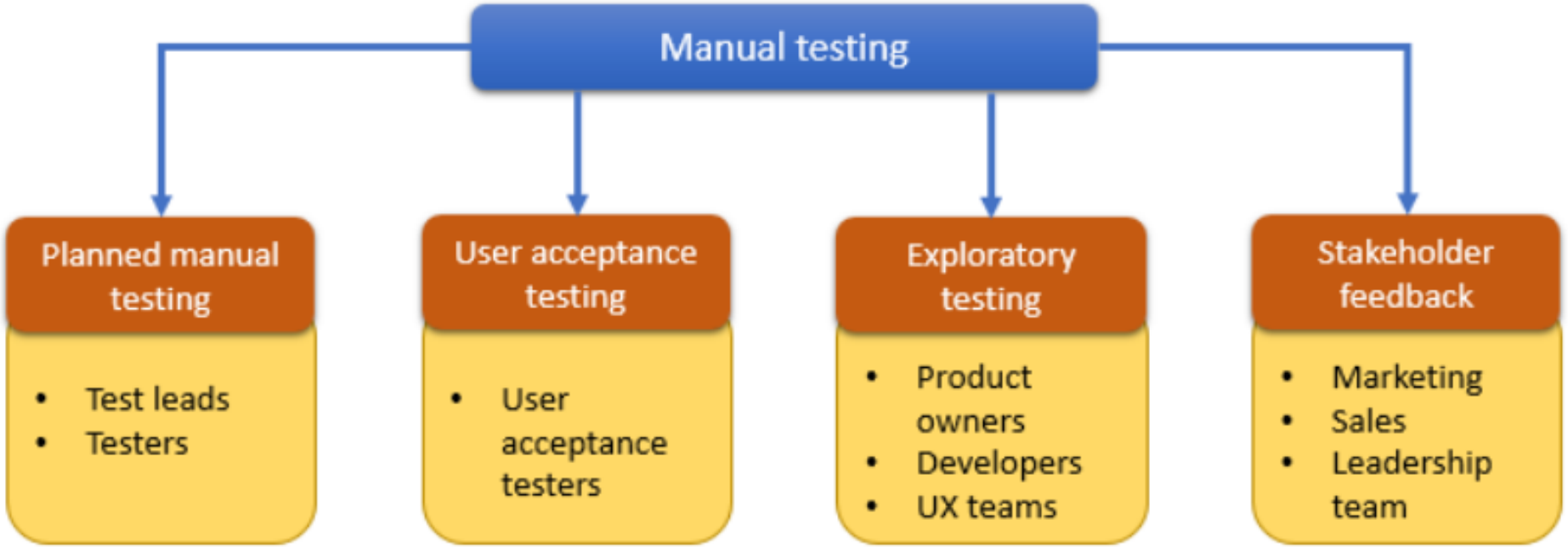
4. Performance tests and load tests

Performance tests validate the application's responsiveness, throughput, reliability, and stability. A load test puts demands on a system to determine behavior under varying levels of load, identify bottlenecks, and validate expected performance.

5. Manual acceptance and exploratory tests

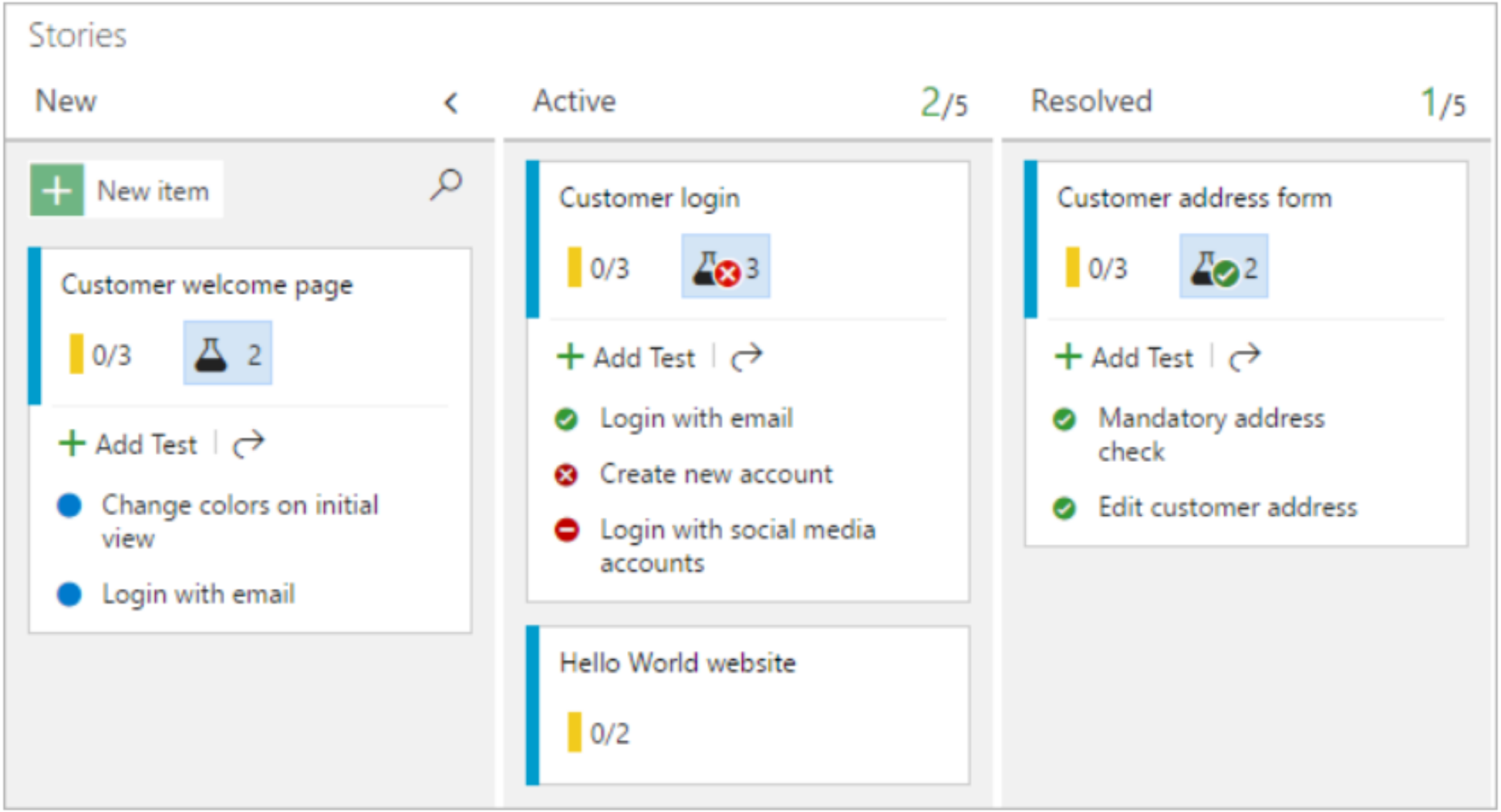
A manual acceptance test validates if a requirement or specification is met. An exploratory test uses a tester's intelligence and creativity to validate quality, usability, performance, and experience.

Testing capabilities



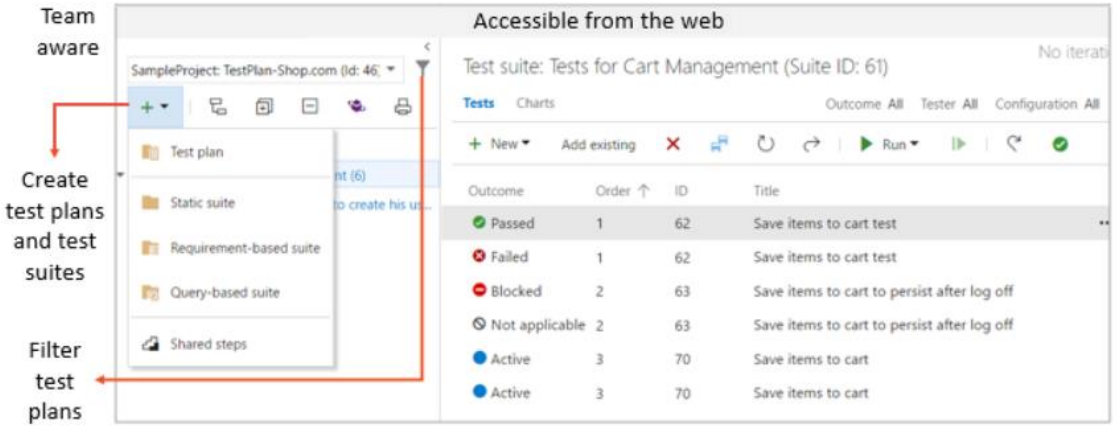
Holistic approach to manual testing, types of manual testing, and personas involved

Planned manual testing - Kanban board

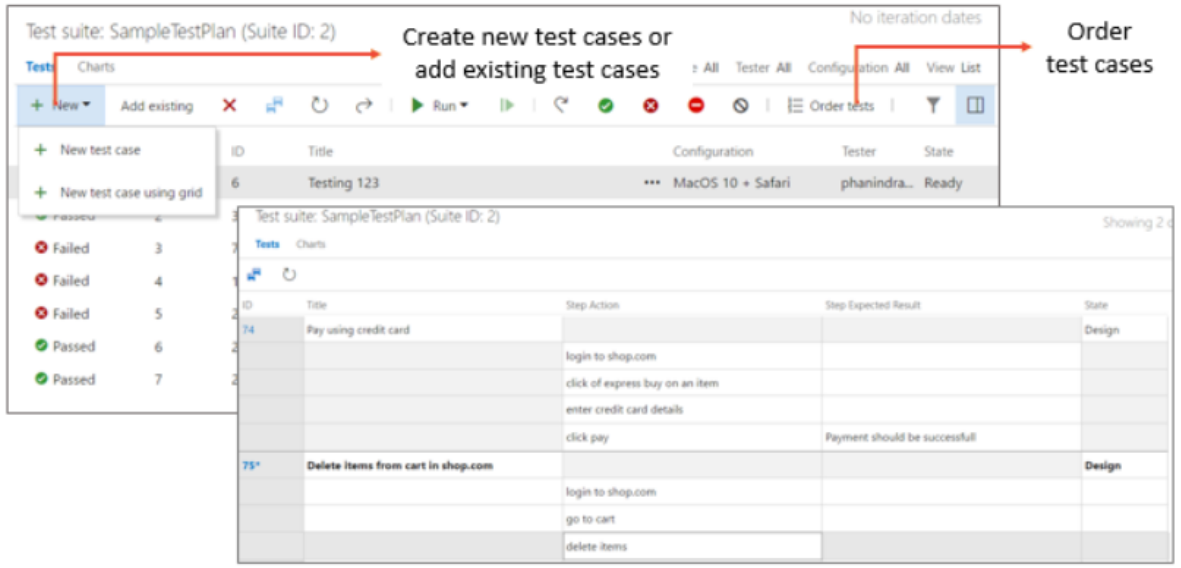


Manual testing - Test Manager

- **Test planning**

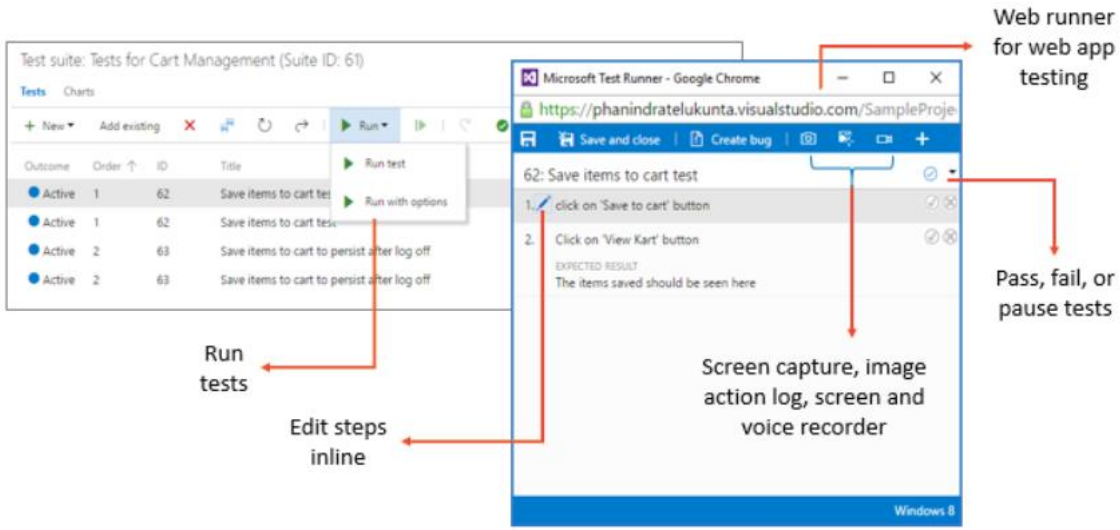


- **Test authoring**

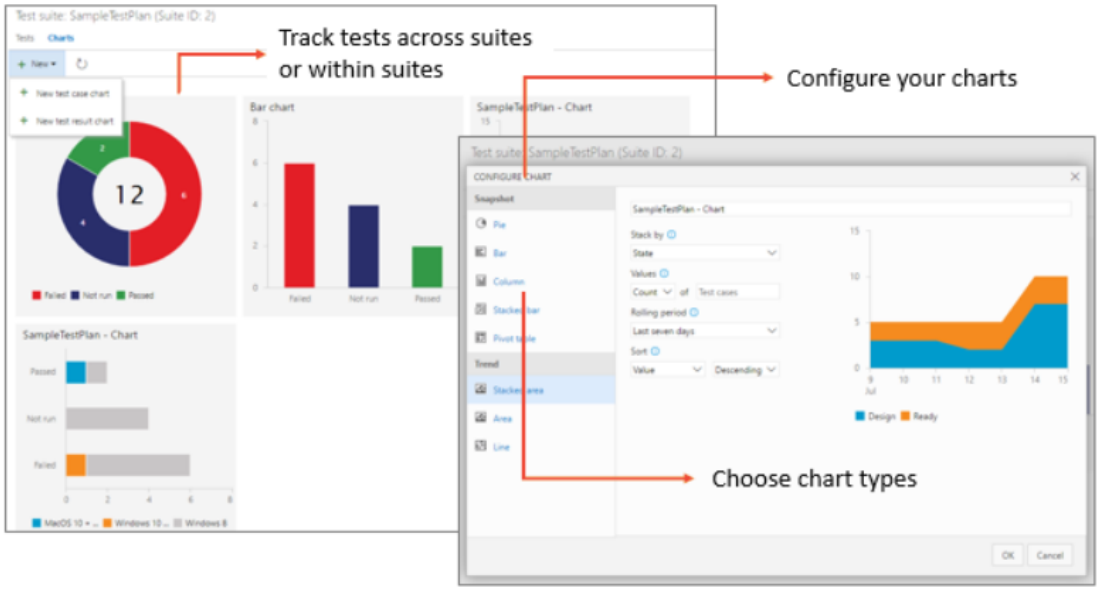


Manual testing - Test Manager

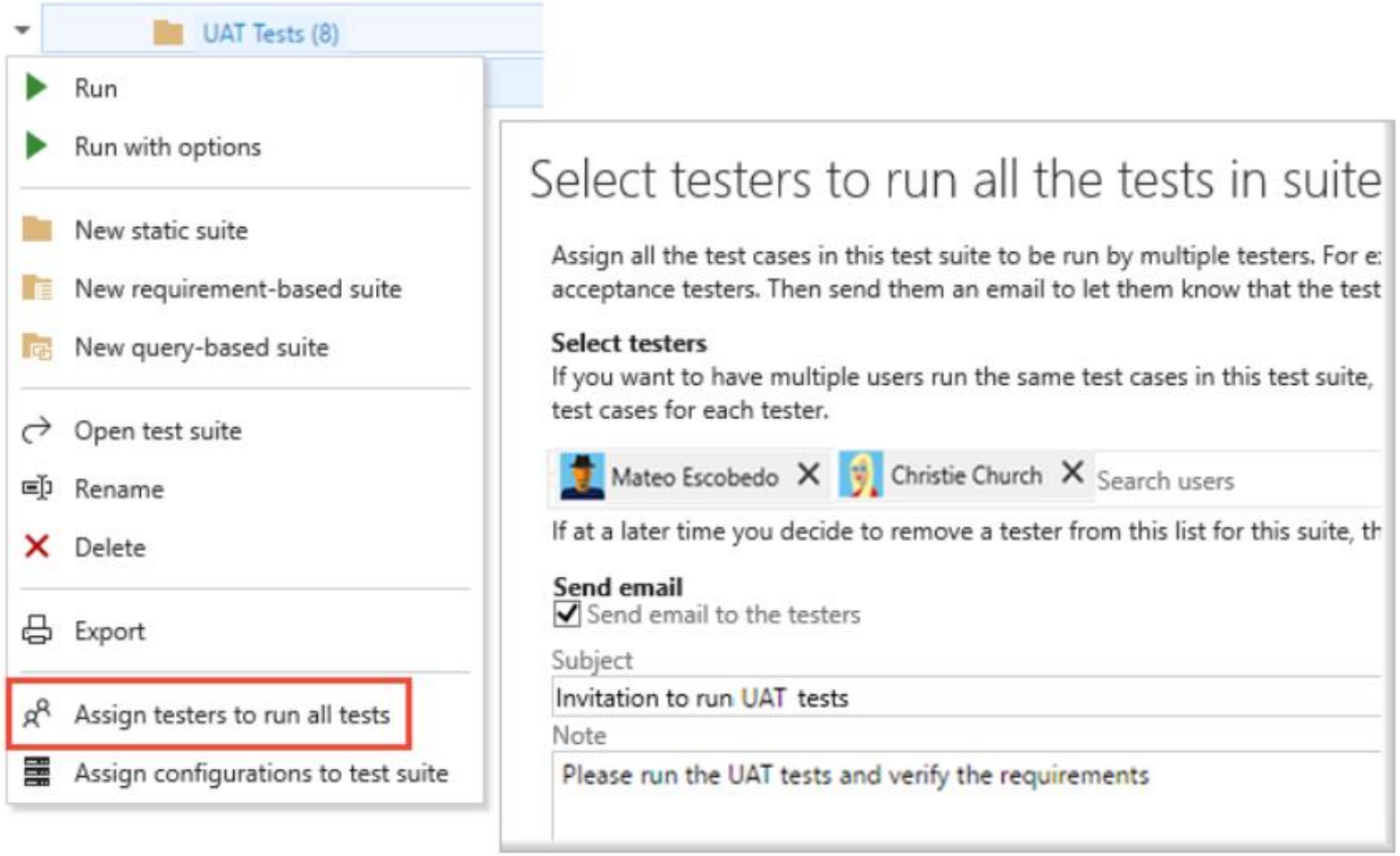
- **Test applications**



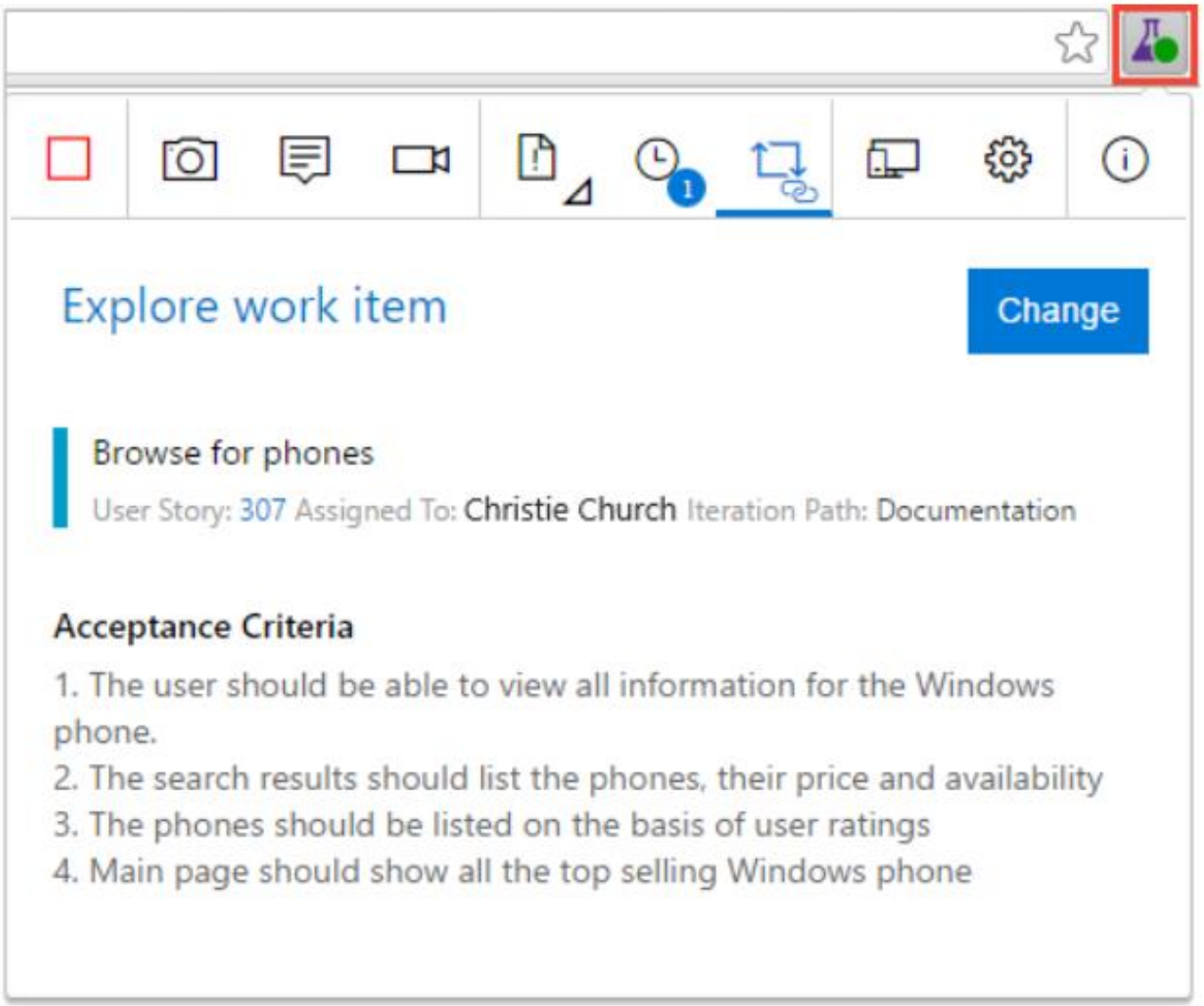
- **Test tracking**



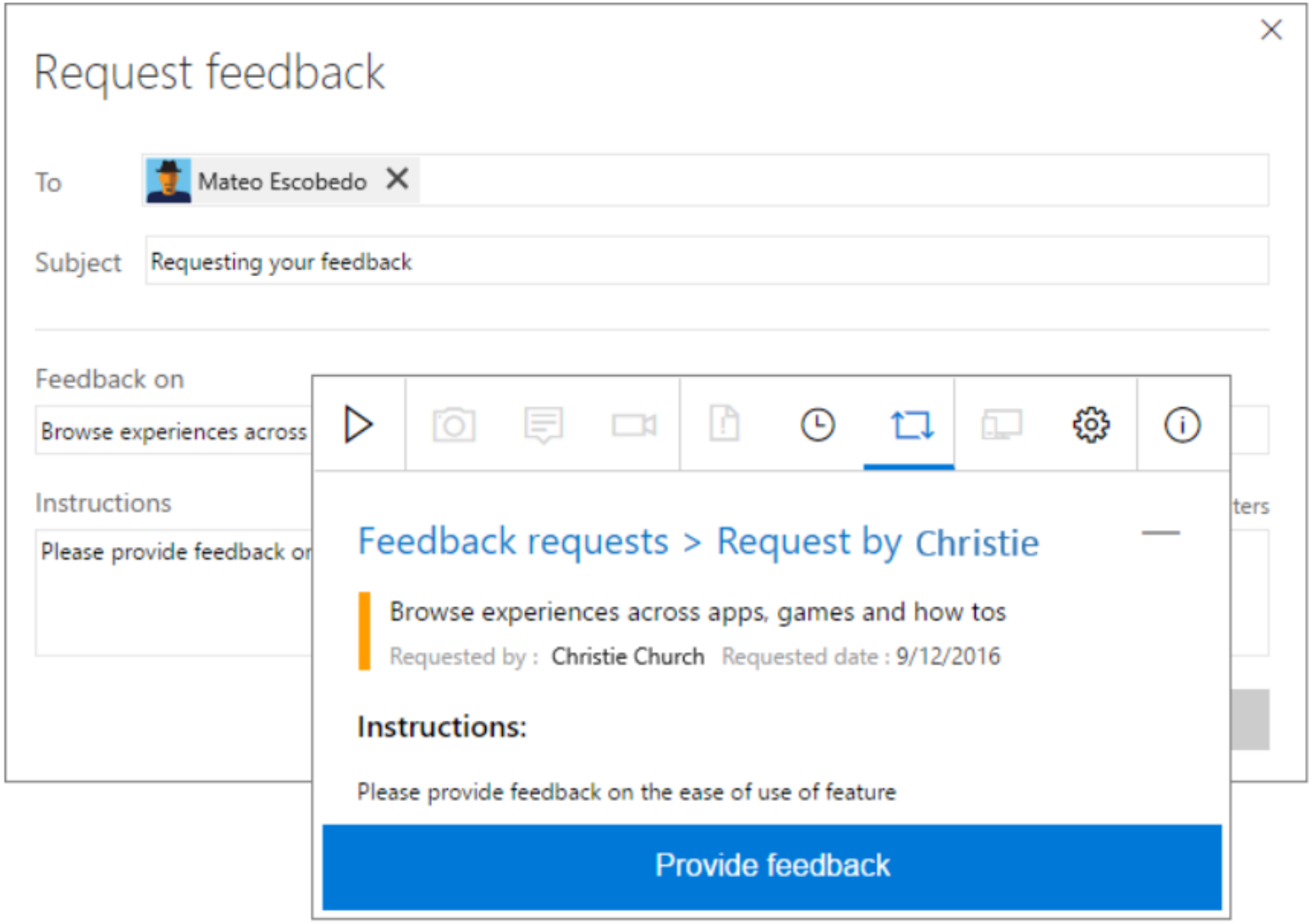
User Acceptance Testing (UAT)



Exploratory Testing – For everyone



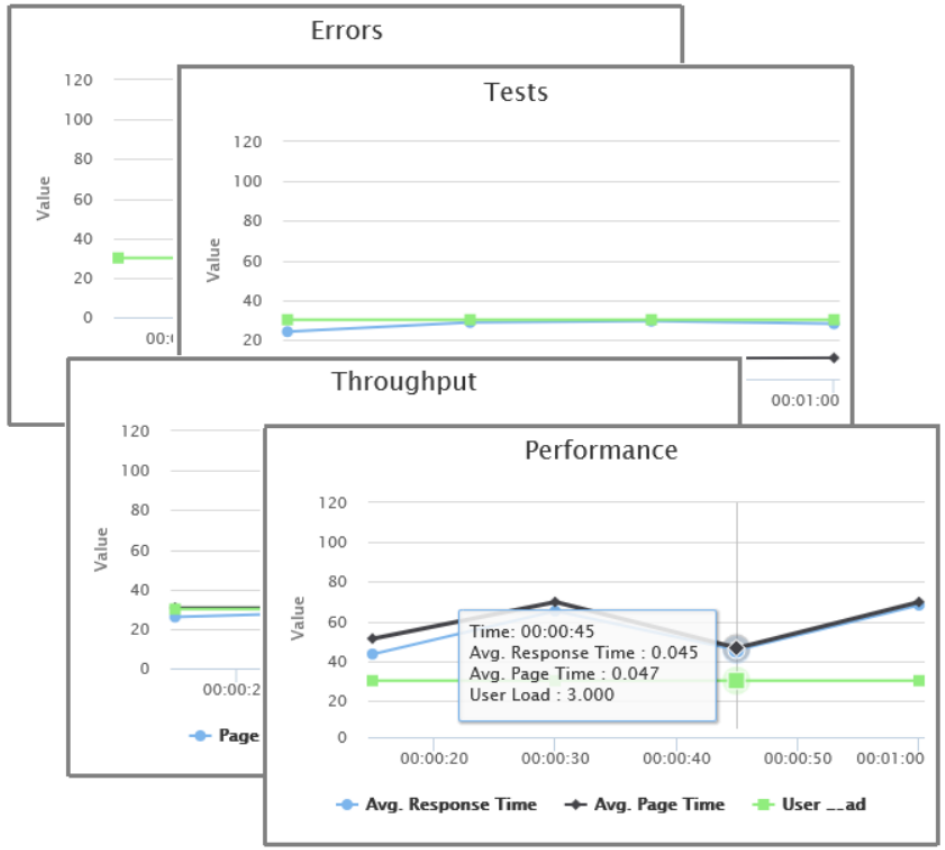
Stakeholder feedback

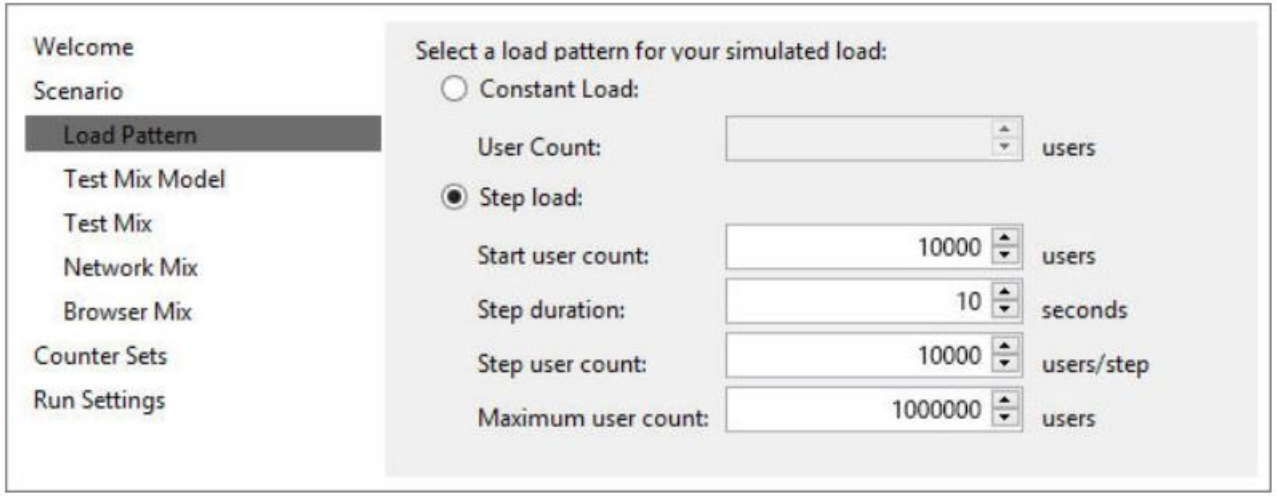


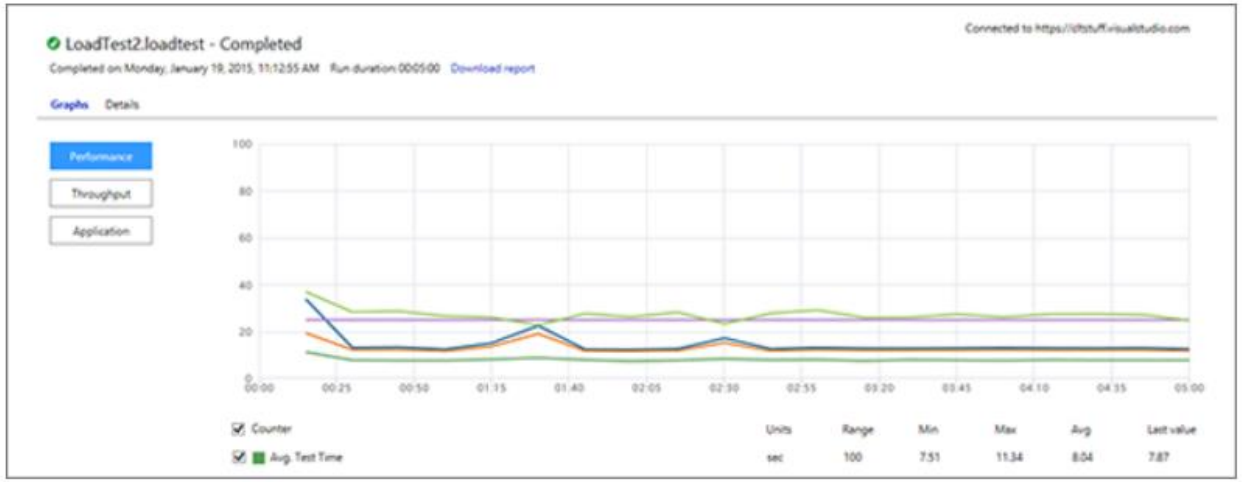
Load Testing

Testing Capabilities

- Load test web sites, apps, and APIs.
- Author tests using Visual Studio, Azure, and Azure DevOps.
- Quickly create load tests by specifying a website, referencing a JMeter test file, or recording and replaying your actions.
- Run tests or customize them using powerful tools in Visual Studio.
- You can even use existing unit or functional tests to generate load.


Errors
Tests
Throughput
Performance
Value
Time: 00:00:45
Avg. Response Time : 0.045
Avg. Page Time : 0.047
User Load : 3.000
Avg. Response Time
Avg. Page Time
Page


Welcome
Scenario
Load Pattern
Test Mix Model
Test Mix
Network Mix
Browser Mix
Counter Sets
Run Settings
Select a load pattern for your simulated load:
Constant Load:
User Count:
users
Step load:
Start user count:
10000
users
Step duration:
10
seconds
Step user count:
10000
users/step
Maximum user count:
1000000
users


LoadTest2.loadtest - Completed
Graphs
Details
Performance
Throughput
Application

Automated Testing Overview

- The world has moved to rapid product release cycles. Gone are the days where product managers planned for a big release every three to four years. Now developers are being asked to react swiftly to customer feedback and release new features rapidly, often against shorter release cycles. In this fast-moving world, developers face a challenging task of continuously updating their software while still ensuring that it meets a high quality bar.

- Automated testing helps developers run tests early and often to ensure that they are testing their software quality continuously and making informed decisions about whether to release their product. Test automation has various benefits, including but not limited to obtaining early feedback, shortening the release cycles, reducing cost, measuring quality continuously, avoiding regressions, shipping high-quality product, and finally, making their customers happy.

Why to DevOps Organizations

- **Early and timely feedback**
- **Speeds up software delivery**
- **Reduces cost**
- **Fundamental to continuous integration (CI) and continuous delivery (CD)**
- **Reliability**
- **Repeatability**
- **Robust testing architecture**

What Is Test-Driven Development?

- Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved to pass the new tests, only. This is opposed to software development that allows software to be added that is not proven to meet requirements.

Source: Wikipedia https://en.wikipedia.org/wiki/Test-driven_development

Benefits of TDD

Why does TDD work?

- Reduces defects by finding and fixing bugs earlier.
- Shortens feedback loop on design decisions.

Value proposition of TDD

- Ensures more predictable process of development.
- Allows for more intentional development.
- Improves long-term maintainability of both code and tests.
- Guarantees good unit test coverage of your code.

Defining Unit Testing

- Unit testing concepts have been around for many years and it is not a new concept. In 1970s, Kent Beck introduced the concept of unit testing in Smalltalk and went on to show that it is one of the best ways that a developer can improve code quality while gaining a better understanding of the functional requirements of a class or a method.

- Ref. from *The Art of Unit Testing book:*

A unit test is a piece of code that invokes a unit of work and checks one specific end result of that unit of work. If the assumptions on the end result turn out to be wrong, the unit test has failed. A unit test's scope can span as little as a method or as much as multiple classes.

Overview

- Unit testing has the greatest effect on the quality of your code when it's an integral part of your software development workflow. As soon as you write a function or other block of application code, create unit tests that verify the behavior of the code in response to standard, boundary, and incorrect cases of input data, and that check any explicit or implicit assumptions made by the code.

Elements of Unit Test

- the unit tests themselves are simple and repeatable, you still need to ensure that you are writing good unit tests if you want to leverage their benefits. You should write good unit tests, or not write them at all. If you write poor-quality unit tests, you will face problems later with the maintainability of your unit tests and time schedules. You should clearly define what a good unit test is and how it aligns with your objectives.

- Many times, developers who try to unit test their code give up after few trials and instead change their approach to perform tests later in the development lifecycle. At other times, they resort to manual testing of their code by using custom test applications. To avoid this pitfall, it is imperative that you understand how to create good unit tests and take full advantage of this simple, yet powerful method of testing your code.

Characteristics of Unit Test

- Automated and Repeatable
- Easy to implement
- Relevant tomorrow
- Anyone can run it at the push of a button
- Should run quickly
- Consistent in its results
- Full control of unit being tested
- Runs independently of other tests
- Easy to detect issues when it fails

Advanced Unit Testing - Tools

- **Test Explorer**—You can run unit tests and see their results in **Test Explorer**. You can use any unit test framework, including a third-party framework, that has an adapter for **Test Explorer**.

- **Microsoft unit test framework for managed code**—The Microsoft unit test framework for managed code is installed with Visual Studio and provides a framework for testing .NET code.

- **Microsoft unit test framework for C++**—The Microsoft unit test framework for C++ is installed as part of the **Desktop development with C++** workload. It provides a framework for testing native code. Google Test, Boost.Test, and CTest frameworks are also included, and third-party adapters are available for additional test frameworks. For more information, see Write unit tests for C/C++.

- **Code coverage tools**—You can determine the amount of product code that your unit tests exercise from one command in Test Explorer.

- **Microsoft Fakes isolation framework**—The Microsoft Fakes isolation framework can create substitute classes and methods for production and system code that create dependencies in the code under test. By implementing the fake delegates for a function, you control the behavior and output of the dependency object.

USER STORY 1

1 Test order page

Unassigned | 0 | Add tag

State: New | Area: Fabrikam | Updated by [redacted] 12/5/2016

Reason: New | Iteration: Fabrikam | Details | (4)

Description

Acceptance Criteria

Discussion

AH *Add a comment. Use # to link a wc*

Planning

Story Points

Priority

2

Risk

Classification

Value area

Business

Development

+ Add link

Development hasn't started on this item.

Create a new branch

Related Work

+ Add link

Tested By (2)

- 21 Test case step 2 TBD
  Updated 8 minutes ago, Design
- 20 Test case TBD
  Updated 9 minutes ago, Design

Child (2)

- 2 Check for error when placing an order
  Updated 12/5/2016, New
- 22 Didnt work
  Updated 3 minutes ago, New

All load tests / FabrikamHomeLoadTest.loadtest (Run ID: 5) / Completed

Rerun

Summary Charts Diagnostics Logs

AVG. RESPONSE TIME	USER LOAD	REQUESTS PER SEC	FAILED REQUESTS	ERRORS →	USAGE
5.4 sec	230 users	4.1 RPS	0 %	4 errors	460 VUMs
			0 failed requests 492 total requests	0 thresholds violated	

Learn more about metrics and criteria

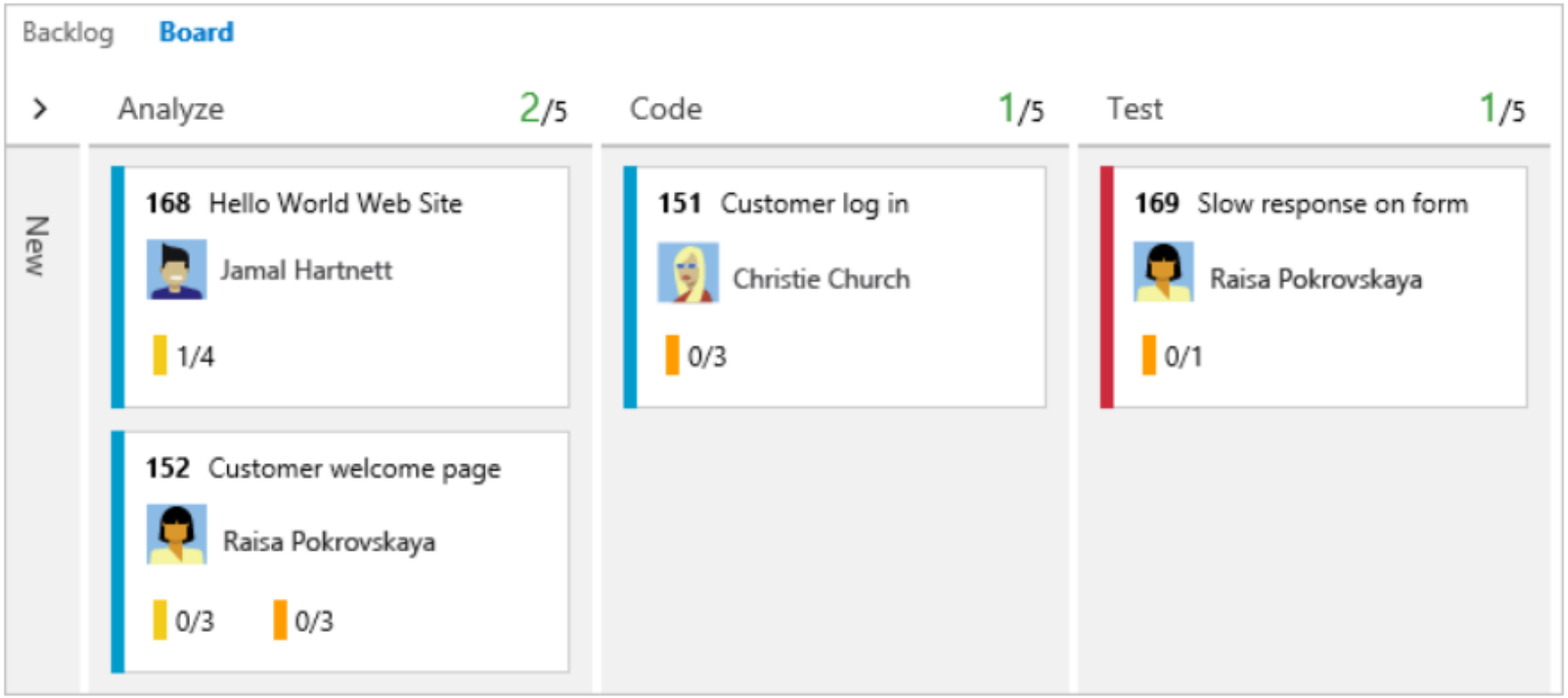
Test settings

Load duration:	2 min	**Requested by:**	C Reinhart	**Run source:**	-
Start time:	27/02/2016 01:22:24	**Test File:**	FabrikamHomeLoadTest	**Warmup duration:**	-
End time:	27/02/2016 01:24:26	**Location:**	West Europe	**Agent cores:**	1

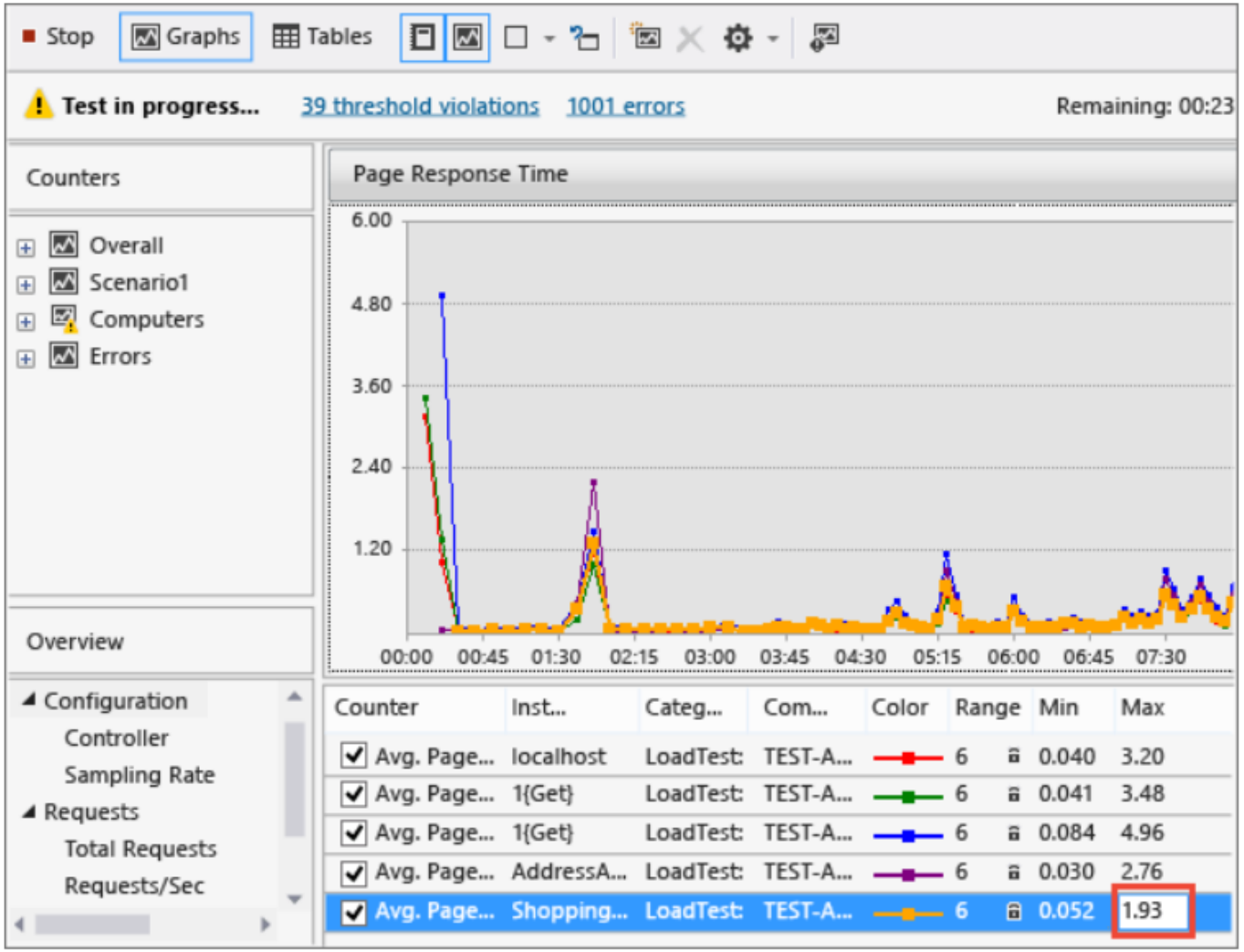
Top 5 slowest requests

Request URL	Scenario	Test	Avg. Response Time (sec)	Total Requests	Failed Requests
http://www.fabrikam.com	Scenario	webscenario1	0.062	168	0

Kanban Board



From Visual Studio



Live Unit Testing

Test Explorer

Search

Run All | Run... | Playlist : All Tests

Passed Tests (6)

- DirectCallWithNullOrEmpty 1 ms
- TestDoesNotStartWithLower 1 ms
- TestDoesNotStartWithUpper 3 ms
- TestHasEmbeddedSpaces 2 ms
- TestStartsWithLower < 1 ms
- TestStartsWithUpper 84 ms

TestStartsWithUpper Copy All

Source: UnitTest1.cs line 12

Test Passed - TestStartsWithUpper

Elapsed time: 0:00:00.0845947

Class1.cs | UnitTest1.cs

StringLibrary | UtilityLibraries.StringLibrary | HasEmbeddedSpaces(string s)

```
1   using System;
2
3   namespace UtilityLibraries
4   {
5       public static class StringLibrary
6       {
7           public static bool StartsWithUpper(this string s)
8           {
9               if (String.IsNullOrWhiteSpace(s))
10                  return false;
11
12              return Char.IsUpper(s[0]);
13          }
14
15          public static bool StartsWithLower(this string s)
16          {
17              if (String.IsNullOrWhiteSpace(s))
18                  return false;
19
20              return Char.IsLower(s[0]);
21          }
22
23          public static bool HasEmbeddedSpaces(this string s)
24          {
25              foreach (var ch in s.Trim())
26              {
27                  if (Char.IsWhiteSpace(ch))
28                      return true;
29              }
30              return false;
31          }
32      }
33  }
```

100 %

Microsoft Test Manager