International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 9 Issue 4 ǁ 2021 ǁ PP. 59-62

Hash Back Testing in Agile Methodology

1.Binu.C.T Freelance software tester

Abstract:
Software engineering comprehends several disciplines devoted to prevent and remedy malfunctions and to warrant adequate behavior. The proposed system have Hash Data common to all phases of software lifecycle in agile methodology. Indeed, software testing is a broad term encompassing a variety of activities along the development cycle and beyond, aimed at different goals. Hence, software testing research faces a collection of challenges. A consistent roadmap of the most relevant challenges to be addressed is here proposed. In it, the starting point client creates Hash Data at the time of requirement gathering. Hash Data is common to both client and testing team.

Keywords: Software Testing, Agile methodology, Exploratory Testing

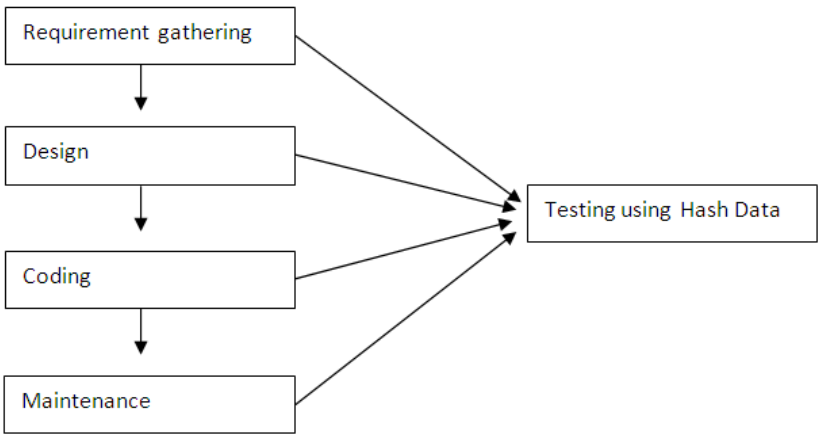
---

Date of Submission: 29-03-2021 Date of acceptance: 12-04-2021

I. INTRODUCTION

1) Data Processing speed can reduce testing effort
2) Adequate confidence that test data is perfect match with the system
3) Test on finite system and non finite system is same for agile system
4) Test on requirement stage reduce bugs
5) All the data available during all the phases may or may not be same

II. LIFE CYCLE OF HASH BACK TESTING



Life Cycle

III. HOW TO CREATE HASH DATA

- The Client and testing Team create Hash Data during in requirement gathering
- Both the client's and Team's Hash Data should be matching
- In the requirement gathering client's officials study the system and creates test data. This data is called Hash Data.
- During design phase the testing team test the design using the same Hash Data. That is the reasons why this is called Hash Back Testing.
- In the coding phase of agile methodology uses both Hash Data and other data for validation.
- But the different data is used for verification.

Hash back testing is a main part of agile software development. Unlike in previous software methodologies, where testing was a separate stage that occurred after development was complete, in an agile methodology testing begins at the very start of the project, even before design phase. Agile testing is continuous testing, which goes hand in hand with development work and provides an ongoing feedback loop into the development process. Another evolution in agile testing is that testers are no longer a separate organizational unit Testers are now part of client and testing team. In many cases, agile organizations don't have dedicated QA team"; Interaction is more in Agile methodology

IV. HOW AGILE METHODOLOGY IMPACTS TESTING

The agile manifesto, the cornerstone of agile methodology, has several points that are especially relevant for testers:

Agile Manifesto Directives	Testing
interactions over processes and tools	Testers should work closely with developers, product owners, and customers to understand what is being developed
Responding to change over following a plan	Agile testers must prioritize and re-prioritize their tests, focusing on what will help the team reach its goal, minimize risk and keep customers happy.

V. AGILE TESTING METHODS

1. Behavior Driven Development

This encourages communication between project stakeholders so all members understand each feature, prior to the development process. In BDD, testers, developers, and business analysts create "scenarios", which facilitate example-focused communication.

Scenarios are written in a specific format, the Gherkin Given/When/Then syntax. They contain information on how a feature behaves in different situations with varying input parameters. These are known as "executable specifications" as they are made up of both specifications and inputs to the automated tests.

The behavioral is that the team creates scenarios, builds tests around those scenarios which initially fail, and then builds the software functionality that makes the scenarios pass. It is different from traditional Test Driven Development (TDD) in that complete software functionality is tested, not just individual components.

Best practices for testers using a BDD methodology:

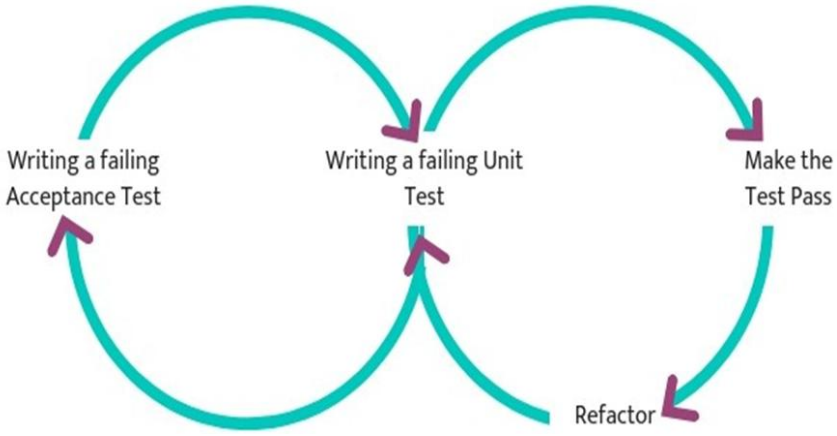
- Streamline documentation to ensure the process is accurate
- The team where the developer, product owner, and tester work together to define scenarios and tests
- Use a declarative test framework, such as Cucumber, to specify criteria
- Build automated tests and reuse them across scenarios
- Have business analysts write test cases and learn the Gherkin syntax

2. Acceptance Test Driven Development (ATDD)

This involves the customer, developer, and tester. "Three Amigos" meetings are held to gather input from these three roles, and use them to define acceptance tests. The customer focuses on the problem, the developer pays attention to how the problem will be solved, and the tester looks at what could go wrong.

The acceptance tests represent a user's perspective, and specify how the system will function. They also ensure that the system functions as intended. Acceptance tests can often be automated. Like in the BDD approach, acceptance tests are written first, they initially fail, and then software functionality is built around the tests until they pass.

Testing Methods Acceptance Test Driven Development (ATDD)



Best practices for testers using an ATDD methodology include:

- Interact directly with customers to align expectations, for example, through focus groups
- Involve customer-facing team members to understand customer needs, including customer service agents, sales representatives, and account managers
- Develop acceptance criteria according to customer expectations
- Prioritize two questions: How should we validate that the system performs a certain function? Will customers want to use the system when it has this function?

3. Exploratory Testing

In exploratory testing, the test execution and the test design phase go together. This type of testing focuses on interacting with working software rather than separately planning, building and running tests.

Exploratory testing lets testers "play with" the software in a chaotic way. Exploratory testing is not scripted – testers mimic possible user behaviors and get creative, trying to find actions or edge cases that will break the software. Testers do not document the exact process in which they tested the software, but when they find a defect, they document it as usual.

Best practices for exploratory testing:

- Organize functionality in the application, using a spreadsheet, mind map etc.
- Even though there is no detailed documentation of how tests were conducted, track which software areas were or were not covered with exploratory testing
- Focus on areas and scenarios in the software which are at high risk or have high value for users
- Ensure testers document their results so they can be accountable for areas of software they tested

4. Session-Based Testing

This method is similar to exploratory testing, but is more orderly, aiming to ensure the software is tested comprehensively. It adds test charters, which helps testers know what to test, and test reports which allow testers to document what they discover during a test. Tests are conducted during time-boxed sessions.

Each session ends with a face-to-face brief between tester(s) and either the developers responsible, scrum master or manager, covering the five PROOF points:

1. What was done in the test (Past)
2. What the tester discovered or achieved (Results)
3. Any problems that got in the way (Obstacles)
4. Remaining areas to be tested (Outlook)
5. How the tester feels about the areas of the product they tested (Feelings).

Best practices for session-based testing include:

- Define a goal so testers are clear about priorities of testing in the current sprint
- Develop a charter that states areas of the software to test, when the session will occur and for how long, which testers will conduct the session, etc.
- Run uninterrupted testing sessions with a fixed, predefined length
- Document activities, notes, and also takeaways from the face-to-face brief in a session report

VI. HOW TO FOCUS AGILE TESTING EFFORTS WITH QUALITY INTELLIGENCE

A central tenet of agile testing is that testing must be prioritized and focus on the user. You can't test everything – so you must focus on the things that users care about. However, beyond gut feelings, most teams do not have data to point them to the features or areas of the product that can have the biggest impact on their customers.

A new category of tools, called Quality Intelligence Platforms, can provide this data, allowing teams to sharply focus their work on customer-facing quality issues.

Test gaps are exactly where agile teams should be focusing testing efforts. Instead of over-testing, or reacting to previous production faults, they can target areas of the product which are at high risk of quality issues. These are the areas that customers will care about the most.

VII. EXAMPLE

Hospital management system
Hash data
Patient name, type, quality of data
Registation+Doctor comment
Tom string
10001 string
Dr.Sam string
General medicine
Fever string
Paracetmol string
Corex cough cyrup string
Salary details
John_Hok_1000 string
100000 Number
Epf_10001 Number
Tax_id_10000 string

VIII. CONCLUTION

The proposed system reduces human effort and it is much faster than existing testing methodology in Agile system. Also it gives more important to software testing in software development lifecycle.

REFERENCES

[1]. Guru99.com
[2]. The Fundamentals of Computational Intelligence: System Approach
[3]. Wikipedia

Mr. Binu.C.T is a post graduate in Computer Science and Engineering. He have 3 Years Experience in Academic field in different Engineering Colleges. He also have 2.3 Years experience in software Testing Domain.