TECHNOLOGY SOLUTION PROVIDER FOCUSED ON
VIDEO AND TEST ORCHESTRATION SOLUTION

Rebaca

Behavior Driven Development (BDD) and Continuous Integration & Delivery (CI-CD)

The goal of testing is to increase confidence of stakeholders through evidence

- Dan North (https://dannorth.neta)

ENSURING QUALITY ON SCHEDULE

Different levels of Testing

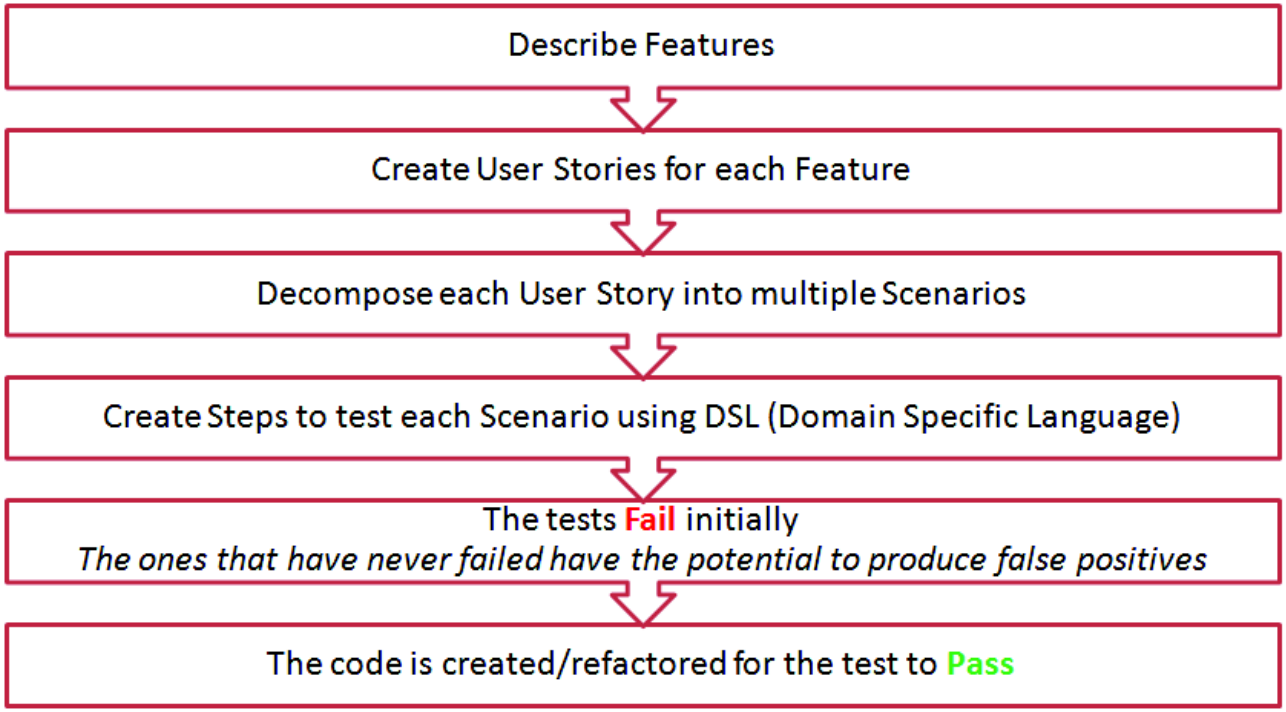
- **Unit Tests** focuses on testing a specific unit or component, such as a class method. It confirms that the function is built right.
- **Integration Tests** focuses on testing multiple “Units” together as a group.
- **System Tests** focuses on testing the complete software or application as a whole.
- **Acceptance Tests** are the final level of testing which verifies whether the system’s functionality matches the specification. It confirms whether we have built the right thing.

Overview of BDD

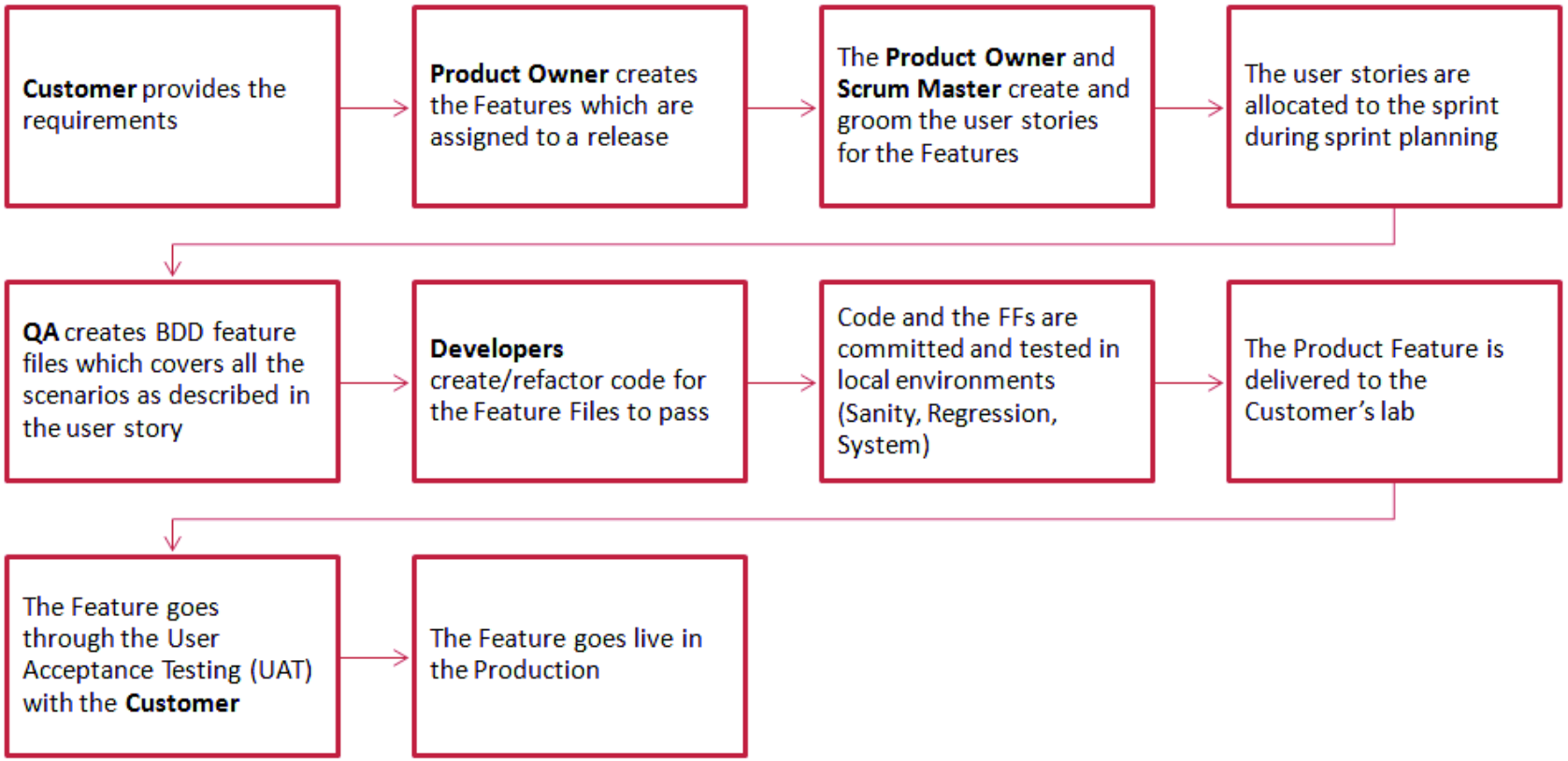
- The Agile methodology implements testing and validation as an ongoing process, via the “as a user” approach. BDD using the same approach, makes it the most suitable practice within Agile.
- BDD focuses on the acceptance criteria from the inception by defining how each feature of the application should behave from the end user’s perspective.
- BDD enables collaboration across all stake holders: Customer, Dev & QA
- In BDD, the Scenarios are created using DSL (Domain Specific Language) and the Code is written to support the target behavior
- BDD results into faster user acceptance cycles leading to faster deployments

More insights

BDD is an abstraction of TDD (Test Driven Development) .They both essentially follow the same practices, but, BDD focuses more on the behavior of the application, rather than implementation, using a language where all stakeholders can participate and collaborate.



A typical Agile and BDD based workflow



TDD vs BDD

TDD (Test Driven Development)	BDD (Behavior Driven Development)
Focuses on the **developer's** opinion on how functions of the software should **work**. It is basically a programmer's view.	Focuses on the **user's** opinion on how they want the application to **behave**. It is basically a customer's view.
A Low level approach	As a user approach
Verifies whether the implementation of the functionalities are correct	Verifies whether the application behaves the way user wants it to behave

Benefits of BDD for Program and Release management

- Better collaboration and communication within all stake holders
- Comprehensible and useable without programming expertise
- The Agile approach has lead to visibly better design and results
- Proven platform for Continuous Integration and Delivery (CI-CD)

The Gherkin language

- The Gherkin language is the Core of any BDD Framework
- It is a Domain Specific, Business Readable language
- Enables all stakeholders to understand the software without a deep technical understanding of the implementation
- In the BDD world, the Test Cases are broken down into the Gherkin Scenarios
- The Scenarios are then implemented using the Gherkin's **G-W-T** construct
    - ✓ **Given** – there is some context
    - ✓ **When** – some action occurs
    - ✓ **Then** – some result is expected

BDD Framework

Rebaca has extensive experience in the following BDD frameworks.

Cucumber (https://cucumber.io/)

- Available in Java, Ruby, JavaScript platforms
- Most popular BDD framework
- Much more vibrant community
- Better reporting support

Behave (http://pythonhosted.org/behave/)

- Available in Python platform

JBehave (http://jbehave.org/)

- Available in Java platform

Sample BDD feature file in action

```
Feature : SIP Single Call with RTP
@ sip-single-call-with-rtp
Scenario : End to end SIP call setup with RTP data transfer
```

Feature name, Scenario name and Tags

```
Given all configured endpoints for SIP are connected successfully
```

DSL grammar to set the context, ie. Setup connection

```
When | run SIP INVITE call-flows for SERVER using uas_invite.xml for 1 users configured
 in user.csv at ABOT
```

Example SIP DSL grammar

```
When | run SIP INVITE call-flows for CLIENT using uac_invite.xml for 1 users configured
in user.csv at ABOT
```

Example SIP DSL grammar

```
When | run the SSH command "cat output file | grep 'Successful call' | awk '{print $1,$2,$6}'" at ABOT
Then I verify the presence of the following values in the SIP response:
  | String                  | occurrence  |
  | Successful call 1       | PRESENT     |
```

Example SIP and SSH DSL grammar to execute command and validate output

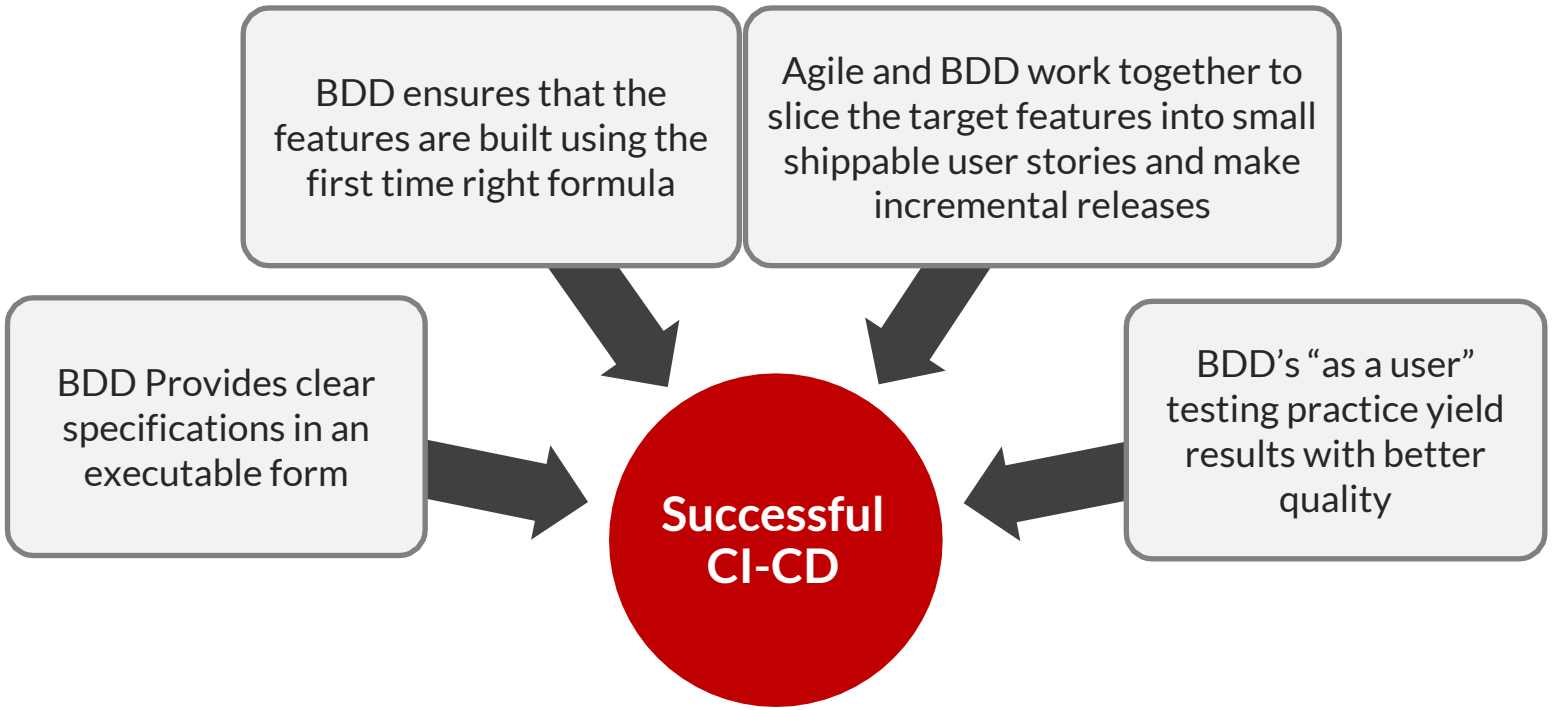
```
When | run the SSH command "cat output file | grep 'RTP pckts sent' | awk '{print $1,$3,$4,$5}'" at ABOT
Then I verify the presence of the following values in the SSH response:
  | String                  | occurrence  |
  | 236 RTP pckts sent      | PRESENT     |
```

Example SSH DSL grammar to execute command and perform validation

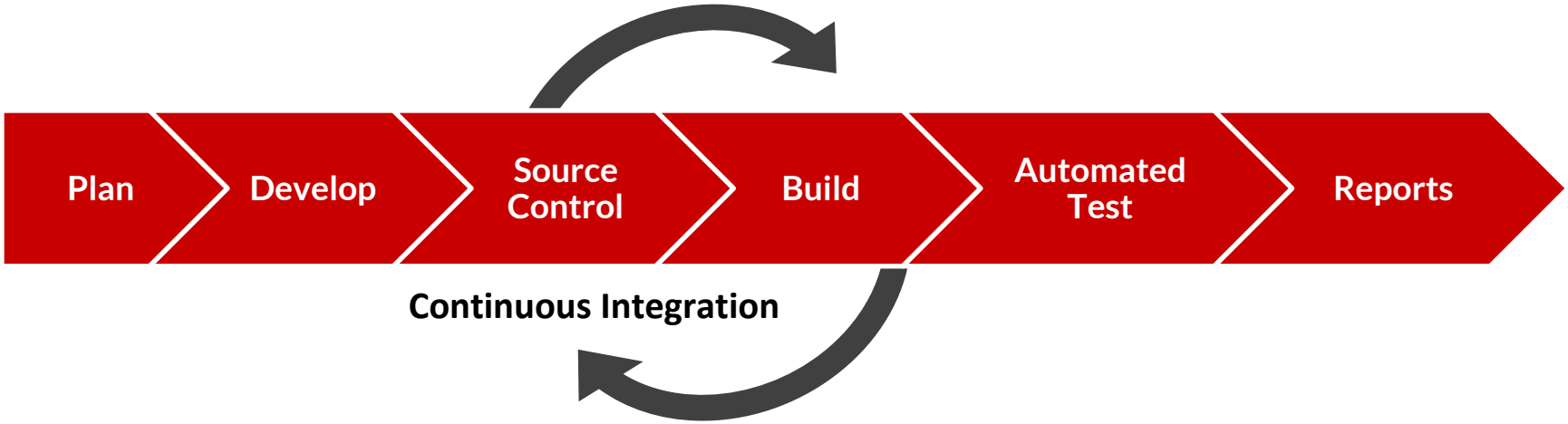
How BDD helps in Continuous Integration and Delivery

Continuous Integration and Delivery (CI-CD) enables the features to be integrated, tested and deployed into Production on an ongoing basis.

It is possible to do CI-CD without BDD, but with the BDD practices, the benefits are evidently better.

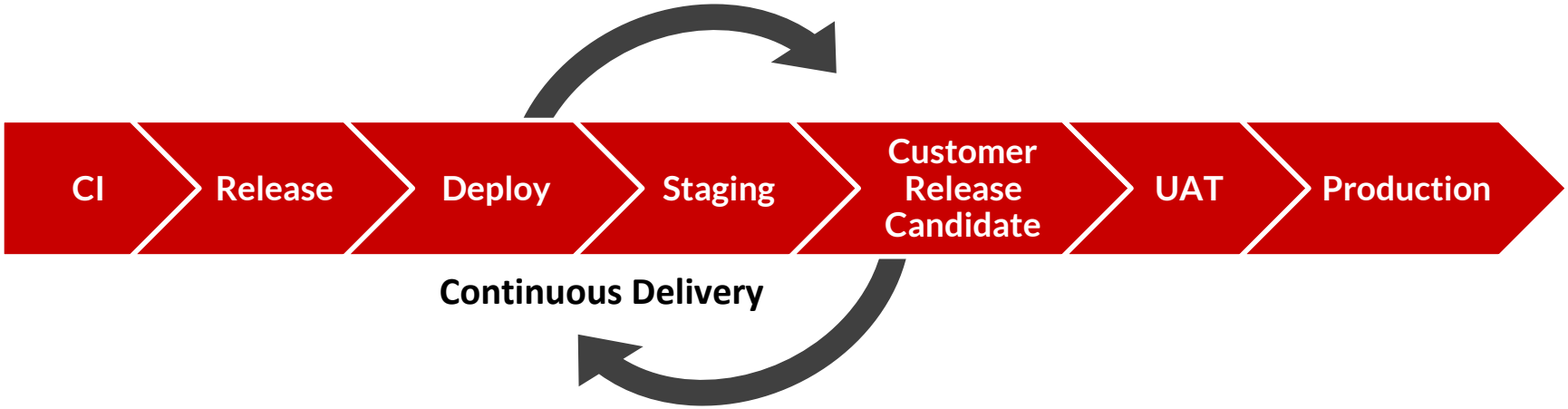


Continuous Integration (CI) workflow



- The developer creates/refactors the code and unit tests it
- The code is committed into the source control (SVN)
- The build is triggered in the build server (Jenkins) on demand/periodically
- The test beds are upgraded with the new build (CI Server – Jenkins)
- Automated Tests (Sanity, Regression, System) are triggered using a BDD based framework
- Comprehensive drill down reports are published and all stakeholders are notified

Continuous Delivery (CD) workflow



- CI-CD provides **Customer Release Candidate** in an on-going basis
- CI-CD enables instant notification for software, environmental, deployment & operational issues that would have otherwise been identified much late in the lifecycle causing delays
- BDD + CI-CD **enables faster User Acceptance** and **Production release**

Rebaca has extensive experience in building a BDD powered framework to implement CI-CD in the software development life cycle providing Automation at each step to create the environment (handle orchestration), deploy the software,perform all configurations, run test suites, generate reports, collect artifacts and notify all stakeholders.