***What is a User Story?***

They are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. They typically follow a simple template:

As a <type of user>, I want <some goal> so that <some reason>.

User stories are often written on index cards or sticky notes, stored in a shoe box, and arranged on walls or tables to facilitate planning and discussion. As such, they strongly shift the focus from writing about features to discussing them. In fact, these discussions are more important than whatever text is written.

One of the benefits of agile user stories is that they can be written at varying levels of detail. We can write a user story to cover large amounts of functionality. These large user stories are generally known as epics. Here is an epic agile user story example from a desktop backup product:

* As a user, I can backup my entire hard drive.

Because an epic is generally too large for an agile team to complete in one iteration, it is split into multiple smaller user stories before it is worked on. The epic above could be split into dozens (or possibly hundreds), including these two:

* As a power user, I can specify files or folders to backup based on file size, date created and date modified.
* As a user, I can indicate folders not to backup so that my backup drive isn't filled up with things I don't need saved.

***Adding Details to User Stories..***

Detail can be added to user stories in two ways:

* By splitting a user story into multiple, smaller user stories.
* By adding “conditions of satisfaction.”

When a relatively large story is split into multiple, smaller agile user stories, it is natural to assume that detail has been added. After all, more has been written. The conditions of satisfaction is simply a high-level acceptance test that will be true after the agile user story is complete. Consider the following as another agile user story example:

* As a vice president of marketing, I want to select a holiday season to be used when reviewing the performance of past advertising campaigns so that I can identify profitable ones.

Detail could be added to that user story example by adding the following conditions of satisfaction:

* Make sure it works with major retail holidays: Christmas, Easter, President’s Day, Mother’s Day, Father’s Day, Labor Day, New Year’s Day.
* Support holidays that span two calendar years (none span three).
* Holiday seasons can be set from one holiday to the next (such as Thanksgiving to Christmas).
* Holiday seasons can be set to be a number of days prior to the holiday.

***Who writes the User Stories and when are the written?***

Anyone can write user stories. It's the product owner's responsibility to make sure a product backlog of agile user stories exists, but that doesn’t mean that the product owner is the one who writes them. Over the course of a good agile project, you should expect to have user story examples written by each team member.

Also, note that who writes a user story is far less important than who is involved in the discussions of it.

User stories are written throughout the agile project. Usually a “story-writing” workshop is held near the start of the agile project. Everyone on the team participates with the goal of creating a product backlog that fully describes the functionality to be added over the course of the project or a three- to six-month release cycle within it.

Some of these agile user stories will undoubtedly be epics. Epics will later be decomposed into smaller stories that fit more readily into a single iteration. Additionally, new stories can be written and added to the product backlog at any time and by anyone.

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***What is PMD?***

PMD is an Open Source, rule based, static source code that analyzes java source code based on the evaluative rules that have been enabled during given execuetion. The tool comes with a default set of rules which can be used to unearth, common development mistakes such as:

* Possible bugs - Empty try/catch/finally/switch blocks.
* Dead code - Unused [local variables](https://en.wikipedia.org/wiki/Local_variable),[parameters](https://en.wikipedia.org/wiki/Parameter) and [private methods](https://en.wikipedia.org/wiki/Private_method).
* Empty if/while statements.
* Overcomplicated expressions - Unnecessary if statements, for loops that could be while loops.
* Suboptimal code - Wasteful String/StringBuffer usage.
* Classes with high [Cyclomatic Complexity](https://en.wikipedia.org/wiki/Cyclomatic_complexity) measurements.
* Duplicate code - Copied/pasted code can mean copied/pasted bugs, and decreases maintainability.

While PMD does not officially stand for anything, it has several unofficial names, the most appropriate probably being Programming Mistake Detector.

Typically, PMD errors are not true errors, but rather inefficient code, i.e. the application could still function properly even if they were not corrected.

PMD is released under a BSDish license while parts of it are under [Apache License](https://en.wikipedia.org/wiki/Apache_License) 2.0 and the [LGPL](https://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License)

***PMD Rulesets..***

PMD includes a set of built-in rules and supports the ability to write custom rules. The custom rules can be written in two ways:

1. Using [XPath](https://en.wikipedia.org/wiki/XPath)
2. Using Java classes

***PMD Plugins***

PMD has [plugins](https://en.wikipedia.org/wiki/Plugins) for [JDeveloper](https://en.wikipedia.org/wiki/JDeveloper), [Eclipse](https://en.wikipedia.org/wiki/Eclipse_(computing)), [jEdit](https://en.wikipedia.org/wiki/JEdit), [JBuilder](https://en.wikipedia.org/wiki/JBuilder), Omnicore's CodeGuide, [NetBeans](https://en.wikipedia.org/wiki/NetBeans)/[Sun Studio](https://en.wikipedia.org/wiki/Sun_Studio_(software)), [IntelliJ IDEA](https://en.wikipedia.org/wiki/IntelliJ_IDEA), [TextPad](https://en.wikipedia.org/wiki/TextPad), [Maven](https://en.wikipedia.org/wiki/Maven), [Ant](https://en.wikipedia.org/wiki/Ant_(software)),[Gradle](https://en.wikipedia.org/wiki/Gradle), Gel, [JCreator](https://en.wikipedia.org/wiki/JCreator), [Hudson](https://en.wikipedia.org/wiki/Hudson_(software)), [Jenkins](https://en.wikipedia.org/wiki/Jenkins_(software)), [SonarQube](https://en.wikipedia.org/wiki/SonarQube) and [Emacs](https://en.wikipedia.org/wiki/Emacs). For the purpse of our project we will be looking into the plugin used for Maven.

***CPD (Copy/Paste Detector)***

The Copy/Paste Detector (CPD) is an add-on to PMD that uses the[Rabin–Karp string search algorithm](https://en.wikipedia.org/wiki/Rabin%25E2%2580%2593Karp_string_search_algorithm) to find [duplicated code](https://en.wikipedia.org/wiki/Duplicate_code). CPD works with [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), [JSP](https://en.wikipedia.org/wiki/JavaServer_Pages), [C](https://en.wikipedia.org/wiki/C_(programming_language)), [C++](https://en.wikipedia.org/wiki/C%252B%252B), [Fortran](https://en.wikipedia.org/wiki/Fortran), [PHP](https://en.wikipedia.org/wiki/PHP), and [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) code.

**PMD checks source code against rules and produces a report. Like this:**

* Something passes a file name and a RuleSet into PMD
* PMD hands an InputStream to the file off to a JavaCC-generated parser
* PMD gets a reference to an Abstract Syntax Tree back from the parser
* PMD hands the AST off to the symbol table layer which builds scopes, finds declarations, and find usages.
* If any rules need data flow analysis, PMD hands the AST over to the DFA layer for building control flow graphs and data flow nodes.
* Each Rule in the RuleSet gets to traverse the AST and check for problems. The rules can also poke around the symbol table and DFA nodes.
* The Report is now filled with RuleViolations, and those get printed out in XML or HTML or whatever

***Resources:***

[www.javavillage.in](http://www.javavillage.in)

[www.wikipedia.com](http://www.wikipedia.com)

<https://www.mountaingoatsoftware.com/blog/>

//Remark:

A user story is a tool used in [Agile software development](http://searchsoftwarequality.techtarget.com/definition/agile-software-development) to capture a description of a software feature from an end-user perspective. The user story describes the type of user, what they want and why. A user story helps to create a simplified description of a requirement.