**1>**  **What is priority?**

**:-** Priority in software testing refers to the order in which defects should be addressed, fixed, and tested based on their importance and impact on the project's goals.

Priority levels are typically categorized into several levels, such as Immediate, High, Medium, and Low. The specific priority levels and their meanings can vary between organizations.

Severity refers to the technical impact of a defect on the software, while priority considers the broader context, including project goals.

**2>** **What is severity?**

:- One can define Severity as the extent to which any given defect can affect/ impact a particular software. Severity is basically a parameter that denotes the impact of any defect and its implication on a software’s functionality. In other words, Severity defines the overall impact that any defect can have on a system.

For instance, consider if a web page or an application crashes after clicking on a remote link. In such a case, a user would rarely click on the remote link. Yet, the overall impact of an app crashing is very severe. Hence, the severity gets high, and yet the priority gets low.

**3>** **Bug categories are…**

**:- Types of Bugs in Software Testing**

Software bugs can be the bane of any software project, causing delays, disruption, or even software failures. Any software development project must pay close attention to bug identification and resolution. This blog looks at some of the most common bugs in software testing processes. We’ll also explore how they can be identified and solved quickly and effectively so software projects can stay on track and meet their goals.

**Different Types of Bug in Software Testing**

**1>Functional Bugs**

**2>Logical Bugs**

**3>Workflow Bugs**

**4>Unit Level Bugs**

**5>System-Level Integration Bugs**

**6>Out of Bound Bugs**

**7>Security Bugs**

**1. Functional Bugs**

Functional bugs are associated with the functionality of a specific software component. For example, a **Login** button doesn’t allow users to login, an **Add to cart** button that doesn’t update the cart, a **search box** not responding to a user’s query, etc.

In simple terms, any component in an app or website that doesn’t function as intended is a functional bug.

Such bugs are often detected when testers conduct comprehensive functional Testing for their apps or websites in [real user conditions](https://www.browserstack.com/real-user-conditions-testing-on-browserstack). Teams need to ensure that all the functional bugs are resolved in the early stages so as to avoid delivering bad user experiences in the production environment.

#### ****2. Logical Bugs****

A logical bug disrupts the intended workflow of software and causes it to behave incorrectly. These bugs can result in unexpected software behavior and even sudden crashes. Logical bugs primarily take place due to poorly written code or misinterpretation of business logic. Example of logical bugs include:

* Assigning a value to the wrong variable
* Dividing two numbers instead of adding them together resulting in unexpected output

#### ****3. Workflow Bugs****

Workflow bugs are associated with the user journey (navigation) of a software application. Let’s consider an example of a website where a user needs to fill up a form regarding their medical history. After filling the form, the user has three options to choose from:

* Save
* Save and Exit
* Previous Page

From the available options, if the user clicks on **“Save and Exit,”** the user intends to save the entered information and then exit. However, if clicking on the **Save and Exit** button leads to an exit from the form without saving the information, it leads to a workflow bug.

#### ****4. Unit Level Bugs****

Unit level bugs are very common, and they are typically easier to fix. Once the initial modules of software components are developed, developers perform unit testing to ensure that the small batches of code are functioning as expected. Here’s where developers encounter various bugs that get overlooked in the coding stages.

Unit level bugs are easier to isolate as developers deal with a comparatively small amount of code. Moreover, replicating these bugs takes less time, so developers can track the exact bug and fix it in no time.

For example, if a developer creates a single page form, a unit test will verify whether all the input fields are accepting appropriate inputs and validate buttons for functionality. In case a field doesn’t accept the appropriate characters or numbers, developers encounter a unit-level bug.

#### ****5. System-Level Integration Bugs****

System-level integration bugs primarily pop up when two or more units of code written by different developers fail to interact with each other. These bugs primarily occur due to inconsistencies or incompatibility between two or more components.

Such bugs are difficult to track and fix as developers need to examine a larger chunk of code. They are also time-consuming to replicate.

Memory overflow issues and inappropriate interfacing between the application UI and the database are common examples of system-level integration bugs.

#### ****6. Out of Bound Bugs****

Out of Bound Bugs show up when the system user interacts with the UI in an unintended manner. These bugs occur when an end-user enters a value or a parameter outside the limits of unintended use — for example, entering a significantly larger or a smaller number or entering an input value of an undefined data type. These bugs often pop up in form validations during functional testing of web or mobile apps.

#### ****7. Security Bugs****

Security is a major concern for software development. Security Bugs are a major risk for users and should be taken very seriously and resolved. Due to their high severity and vulnerable nature, security bugs are considered among the most sensitive bugs of all types and should handled with criticality and urgency. These bugs might not hinder the operation but can compromise the whole system. These should be checked thoroughly at regular intervals.

**4>** **Advantage of Bugzila.**

**:- The various benefits of using Jira are as follows:**

# > **Standard tool for teams of all sizes**

# > **Simple licensing model**

# > **Lightweight tool**

# > **Low maintenance**

# > **Ease of use and intuitiveness**

# > **Increased productivity**

# > **Better visibility**

# > **Integration with other tools**

# > **Marketplace apps**

# > **A RESTful API for limitless possibilities**

# > **Atlassian Community**

# **1> Standard tool for teams of all sizes**

When you install Jira for the very first time, you'll find out that it's extremely easy to get started with the tool. You can use various out-of-the-box templates to create a project. Irrespective of your team size, you always have this option to use an industry-standard tool. Even for a small team, the configurations that come with the out-of-the-box templates are well defined. Jira doesn't differentiate its features based on the team size; whether you're a small team or a big enterprise, the features that you get in the tool are the same. The way a small team of 10 developers work on an Agile board will be the same as a big team of 50 developers working on multiple Agile boards.

# **2>Simple licensing model**

Jira is proprietary software developed by Atlassian and you need to purchase the license to use it. The licensing is based on the number of users you have in the team who will be using the system, and it's very straightforward.

Let's say you have 15 developers, five managers, and two administrators then you would have, in total, 22 users using the system and to understand the license that you need to purchase, just check on the Atlassian website. In the case of 22 users, there's a slab of 25 users that you need to purchase.

# **3>Lightweight tool**

Installing Jira on your own server doesn't require heavy investment in infrastructure. It can run reasonably well on a server with 2 GB RAM and a multicore CPU; of course, it also depends on the size of the instance. For small instances with a few thousand issues to big instances with millions of issues, the server specifications can vary and that is the great part of using Jira.

If you're just getting started with a small instance then choose the specifications that are recommended by Atlassian in the preceding link. You aren't forced to set up a big infrastructure in the beginning.

# **4>Low maintenance**

We discussed previously the challenges that you should be aware of when choosing the tool; apart from the features and cost, which are important in the beginning, you should also be aware of the running cost of the tool. Like any other tool, Jira also requires continuous maintenance where the administrator needs to ensure that the system is cleaned up regularly, backups are managed properly, and the instance itself is upgraded from time to time.

There are new minor versions of Jira released every other month that contains not only new features but also bug fixes.

Upgrading Jira and maintaining it are not at all a daunting task. With the right governance in place, it's quite simple to ensure good performance of the tool. Most Jira administrative actions are done from the UI and it's very convenient for the administrator to handle customizations and other recurring administrative tasks from the browser window.

# **5>Ease of use and intuitiveness**

Using Jira is extremely intuitive and easy. Each user—whether a developer, manager, or administrator—will log in to Jira using a web browser. The moment they log in, they're presented with the Dashboard and depending upon the rights and permission of the user the appropriate features are shown and enabled for them.

Reading the documentation or a book like this one will help you quickly get started on the tool, and there are plenty of resources available to learn the tool to get the most out of it. However, if you're trying to roll out Jira in your company, then you can be assured that users can very quickly adopt it; with very short training or coaching sessions, it's very easy to introduce a tool such as Jira in your team.

# **6>Increased productivity**

The ease of use and intuitiveness help a lot, as we just discussed in the preceding adoption of the tool section but when people start using Jira, you can then expect your productivity to increase. Not only is Jira a great tool for planning your activities or projects, but the day-to-day tracking is also very good in Jira. There are various ways through which your team is always up to date with the latest activities in the project. There are dashboards where everyone with the right access can see the information relevant to their team and themselves. Also, the tool has the capability to send notifications and reminders to users.

For example, users involved in a project or any task will get emails from the system. Of course, these email notifications can be customized to reduce noise but there are mechanisms to ensure that the overall team productivity is enhanced when using a tool such as Jira.

# **7>Better visibility**

Planning and tracking activities are of no use when the team can't learn from their mistakes and improve upon them. There's a plethora of reports that we can create in Jira related to one or multiple projects.

In Jira, there's a concept of a dashboard where users, based on their permissions, can create one or multiple dashboards containing various gadgets to provide them with up-to-date reports.

These reports help the managers to not only stay on track but also to ensure that the overall progress is maintained and take appropriate action well before time. For instance, on the dashboard, we can see in graphical and tabular form the issues getting resolved versus closed issues in the past few months and a burndown chart, a pie chart to display the break-down of issues based on their workflow status.

The Agile boards that come with Jira Software always display the most up-to-date information to the whole team of developers and queues in Jira Service Desk also give agents an accurate picture of tickets in their backlog.

Bringing more visibility is one of the key points of Jira and it's really good at it.

# **8>Integration with other tools**

Jira itself is a great tool; however, there are other tools from Atlassian, such as Confluence, which is used for online collaboration; Bitbucket for code repository; and Bamboo for continuous integration. Jira and most of the other tools from Atlassian talk to each other natively.

Atlassian has this ecosystem and set of integrated tools that can be used standalone but, when integrated with each other, provide a complete solution.

For example, if your organization is looking to implement DevOps practices using Atlassian tools, then you can use tools such as Bamboo, Bitbucket in conjunction with Confluence, and Jira integrated with most of them.

Apart from integrating with tools from Atlassian, Jira can also be integrated with third-party tools and that's done using either an app or an add-on, which we'll discuss in a moment, but for any other integration where you cannot find an app, there's also the possibility of using the Jira RESTful API.

# **9>Marketplace apps**

Jira has lot of features out of the box and it comes with a few templates based on the type of application you use. For instance, when using Jira Software, you can create a project with either Scrum- or Kanban-based configurations. At the same time, you always have this option to modify and create your own set of configurations using various schemes in Jira. However, apart from the standard features in the tool and these customizations, if you need to extend the feature set of Jira, you can install various plugins better known as add-ons or more recently as apps.

When you go to the Marketplace, you have the option to look at various popular apps for each Jira application but let's say you want to do Test management in Jira and you're wondering whether an app can quickly give you this option; you can download various apps from the Marketplace and evaluate them.

Having these apps gives you endless possibilities in Jira—of course, it depends on whether the app is good enough for your use case or not, but at least you know that, with the help of an app, you can do what you can't do out of the box. These apps either provide more functionality or provide integration with other tools.

# **10>A RESTful API for limitless possibilities**

We just discussed the concept of an app that gives you more options in extending the features of the tool but, in case you are wondering how to programmatically push and pull data into Jira, then you'll be glad to know that Jira comes with a RESTful API, which is an amazing way to talk to Jira from other tools.

Whether you want to build your own interface, import data into Jira, or maybe interact with the tool from your existing legacy tool, then the RESTful API opens a lot of possibilities:

The preceding link will give you the details of this amazing set of APIs in Jira. Most of the functionality that you access from the UI and various features can also be accessed using RESTful APIs. This enables developers to write their own interface and integrate with the tool.

# **11>Atlassian Community**

We've been talking about the various benefits of Jira but this one requires special mention. Atlassian not only builds Jira, which is already quite popular in the industry, but it also provides a platform to have discussions about its tools:

Atlassian Community is one such platform where different types of users can come not only ask questions but also to share their knowledge and connect with other users. As shown in the following screenshot, the Atlassian Community home page will present you with an option to either click on a specific Atlassian product or search for the information you are looking for:

**5>** **Difference between Priority and Severity**

**:-**

|  |  |
| --- | --- |
| **Priority** | **Severity** |
| Defined by the impact of a specific problem on any application’s functionality. | Defined by the impact on business. |
| Category decided by testers. | Category decided by developers or product owners. |
| Deals with the technical aspects of the application. | Deals with the timeframe or order to fix the defects. |
| The value does not change with time, it’s fixed. | The priority value is subjective and may change after comparing with other defects. |
|  |  |
|  |  |