30 QA PRO TIPS

Evolve your development!

100% ORIGINAL CHAT GPT FREE

redsauce
SOFTWARE QUALITY

INDEX

INTRODUCTION

Would you like to get better results in your projects by optimising resources and improving Time-to-market?

In this ebook you will find **30 tips and practical strategies** that will help you face the most common obstacles in software development, to achieve higher levels of excellence.

We've created this ebook with the challenges you face every day in mind. Whether you are a developer, product manager or quality manager.

Get ready to discover the keys that will take you to the next level. Let's get started!

3 TIPS

VALIDATION AND CREATION OF USER STORIES

You will learn how to involve key users from the earliest stages of software development, enabling you to understand and address their needs more accurately.

With practical tips and examples, this chapter will guide you to improve the quality of your user stories, thereby optimising the development process and delivering software that truly meets the demands of your users.

1. Use of guidelines or acceptance criteria to validate user stories

Acceptance Criteria are a set of conditions that describe the expected behaviour once the user story has been developed. They are the requirements against which we assess whether the development matches our expectations from the end user's perspective.

Providing good acceptance criteria increases confidence that the product reflects our intentions, reduces the possibility of defects and speeds up development. Feedback rejections, once in production, should be less frequent.

TIP

We must have a tool to ensure that the User Story has been developed correctly. Acceptance Criteria are a great help for this purpose because they allow us to better define the User Story and provide a validation criterion for the User Story.

Providing good Acceptance Criteria increases confidence that the product reflects our intentions, reduces the possibility of defects and speeds up development, as rejections in Acceptance Testing and once in production are less frequent.

No job should be considered completed unless it meets all the Acceptance Criteria.

2. Creating user stories under the INVEST principles

INVEST is a mnemonic created to remember the characteristics of a quality backlog item (PBI). This item is generally in User Story format, fulfilling the following aspects:

I: Independent. Must have no or minimal dependency on other User Stories.

N: Negotiable. Its content can be modified, extended or deleted according to business needs.

V: Valuable. It must contribute clear value to the product, directly or indirectly.

E: Estimate-able. We must be able to size it, estimate the effort required to carry it out.

S: Small. It must be able to be done in hours, ideally within one working day. This helps continuous integration and quick feedback. If the size is too large, we should be able to break it down into smaller chunks.

T: Testable. It must be possible to test it to validate that its behaviour is as expected.

TIP

Following the INVEST (Independent, Negotiable, Valuable, Estimate-able, Small, Testable) principles when creating User Stories allows us to provide them with features that will help us in their development. In case of not following them, we can find ourselves with:

- Dependency between user stories, delaying deliveries, changing scopes, etc.
- Non-negotiable content, making development very rigid and difficult.
- Results of little or no value, making it difficult to answer the question: why is this HU necessary?
- Poorly defined HU, complex to estimate, to be defined a posteriori with the extra workload involved.
- Extremely large tasks, which block members of the team for days,

slow down code reviews, hamper continuous integration, etc.

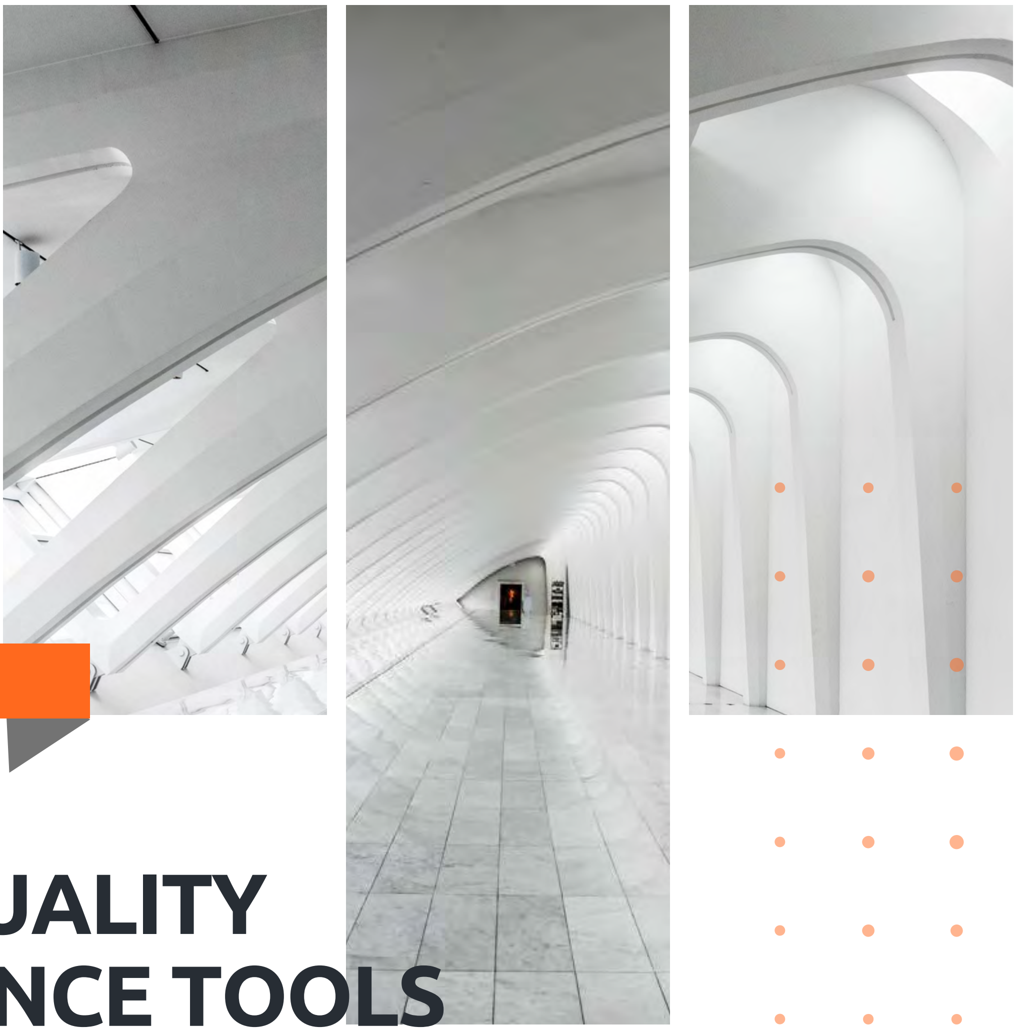
- Difficulty or impossibility to test the HU, which increases the likelihood of a bug appearing after deployment or in the future (detectable by automated testing).

3. Establishment of the DoR (Definition of Ready) to create the stories.

The Definition of Ready is an agreement between the team and the Product Owner on what it means for a backlog item (usually called a PBI) to be ready to be understood, assessed and executed.

TIP

Having a Definition of Ready speeds up the development of backlog items by helping to have them defined before programming begins.



4 TIPS

CODE QUALITY ASSURANCE TOOLS

We will explore various tools that will help you improve code quality in your software projects. We will talk about static analysis tools, unit testing and TDD to identify and correct errors in your code.

These tools will allow you to improve the readability, maintainability and efficiency of your code, ensuring a better quality end product.

4. Static code analysis tools

Static code analysis allows the evaluation of the developed code without the need to execute it. It is a test that should be performed automatically before code developed in one branch is accepted in another. Usually it is the continuous integration server that has a trigger that launches it. It is a quick test to run and brings great value to continuous integration. It provides great cost savings by anticipating problems and non-functional errors.

TIP

We recommend having a tool to validate the static code of a branch prior to an integration. The configuration is not complex and obtaining relevant information is very quick. With this information, it is also possible to propose different "quality gates" policies and provide the development pipeline with the criteria to stop them if necessary.

5. Using Linters in the Developer IDE

A linter is a tool that analyses our code in real time as it is being written. By means of a series of rules defined by the developer, it helps to follow the best practices or style guides of our language and detect errors or warnings that could cause compilation problems or bugs.

TIP

Generating quality software is difficult, not least because of the many different variables that affect it and the almost infinite number of ways in which the same thing can be done. We recommend that the development team use tools that allow code to be generated in a common style, good practices within the language and ultimately a product with the highest possible quality. Configuring a linter in the IDE of the work environment is one of the best alternatives, it is simple and has benefits from the first moment.

6. Use of unit tests during continuous integration

Unit tests are tests of small functionalities that are executed very quickly, always automatically. They are usually programmed by the developer in order to ensure a correct implementation and future stability of his work. This type of test is very efficient if they are well developed, as they can be executed in a few seconds (ideally in less than 1 second) and in a global way, offering quick feedback, something very desirable in agile developments.

TIP

Possibly one of the best bets for quality in a development team is the commitment to create, maintain and extend unit test coverage. It is the base of the classic Test Pyramid because they give us feedback very quickly and are cheap to create and maintain, especially compared to User Interface tests.

We strongly recommend integrating unit tests into the development process by including them in the quality strategy.

7. Use of TDD methodology

TDD is a software programming practice in which, before programming the functionality we want to implement, we write a test (in practice a unit test) that will validate it. As there is no code yet, the test will fail. Next, the minimum code is developed that makes the test pass successfully. Finally, that code is refactored and is ready for completion.

TIP

Implementing TDD has several advantages. On the one hand, the mere fact of thinking about the functionality to implement the test and that it is minimal puts focus on how it will be used, unveils edge cases and organises the code. It also reduces the development effort in mature stages of the application, producing higher quality and in less time.

If the decision is made to start applying TDD, we recommend that you contact someone with experience to guide you through the first steps.

2 TIPS

MANAGEMENT OF THE CODE INTEGRATION PROCESS IN THE REPOSITORY

The integration of code between different branches is a basic aspect of ensuring software stability and quality. We present two fundamental strategies to efficiently manage branches, resolve conflicts and guarantee a smooth integration.

8. Use of Merge request and code review

A Merge request (also called Pull request in some tools) is a way to review, test and approve a code change in the repository. It is possible to assign the review to one or more team members, require approval or not under certain rules, etc.

TIP

Having a Merge Request policy in place will allow code developed by one team member to be reviewed by others and reduce the likelihood of design errors, defects, etc.

Another great advantage of implementing Merge Request is that it is a great source of knowledge and training, by sharing different ways of tackling a problem and programming the solution.

9. Benefit from a unified approach on how to create a Merge Request

In addition to having the Merge Request review mechanism in place, it is possible to have an agreement in the development team on the characteristics of the Merge Request, such as for example:

- Format of the title, form of wording, maximum length, use of prefixes....
- Content of the description, format, size...
- Size of the code content in the merge request, number of changes, number of lines, number of files...

TIP

Not having criteria agreed by the whole technical team regarding how Merge Requests should look like can lead to:

- Difficulty in finding defects because there is too much code to review. That Merge Request would be susceptible to being split into smaller ones. This can lead to not finding some defects, as they are "hidden" among so many lines or to complicate validation if the change affects different functionalities.

- Delaying the identification of a problem at the time of the review or at the time of a subsequent postmortem analysis due to the existence of different title formats and not being self-explanatory.

- Increase review time by having to understand what the change does or having to ask questions if there is no good description.

2 TIPS

HOW TO VERIFY THAT QUALITY REQUIREMENTS ARE MET

We will provide you with 2 strategies to ensure that your product meets established quality standards. Learn how to define clear and measurable quality criteria and how to use effective tools and methods to assess compliance with these requirements.

10. Programmers check the "Acceptance Criteria".

Acceptance criteria (AC) are a set of conditions that describe the expected behaviour once the user story has been developed. They are the requirements against which we evaluate whether the development matches our expectations.

Validation that the acceptance criteria are being met can be carried out only in the acceptance test itself by the Product Owner, Stakeholder or the appropriate figure in your team ("classic" approach) or it can be carried out at other points in the product lifecycle, such as during the programming of the functionality.

TIP

We consider it essential that developers verify that the Acceptance Criteria are met in the work they are doing. In this way, feedback in case something does not work as expected will come much sooner and the Acceptance Test is much more likely to be successful.

As a result, no development should be considered completed if, among other things, it does not meet all the Acceptance Criteria.

11. Establishment of the Definition of Done (DoD)

The Definition of Done, or DoD, is an agreement reached by a development team on what it means for a release to be ready to be deployed in Production.

Generally, this agreement encompasses aspects on three components: functional requirements (user stories, acceptance criteria...) or business requirements, quality (test coverage, defects...), and non-functional requirements (performance, security, usability...).

TIP

We recommend drafting and having the Definition of Done document available and iterating on it. It will help us to reduce the possibility of deploying an incomplete, unsafe, inefficient, defective, etc. product.

6 TIPS

TEST AUTOMATION

Find out how to reduce time and resources, performing exhaustive tests with frameworks and automation tools.

6 practical tips that will give you the keys you need. Identify and fix bugs quickly and efficiently, delivering a higher quality end product to your users.

12. Automatic integration tests

The integration test is the phase in which the communication interface between software modules that we have previously been able to test in isolation is tested. In this way, the modules are "integrated" and the whole is tested as a unit.

TIP

Integration tests are an essential part of a testing strategy, because they reduce the risk of errors appearing when connecting modules that, in isolation, we have validated that they work correctly. We recommend including them in the test plans because although they are slower to execute than unit or contract tests, they will allow us to get faster feedback than User Interface tests.

13. Automated API testing

API tests, or service tests, validate their functionality, structure, reliability, performance, security, etc., forming part of the integration tests. When it comes to automating them, they are quicker to execute than User Interface tests.

TIP

We include the automation of API tests as part of the application integration tests. We consider it very important to have coverage of these tests to add a layer of confidence, especially as the number of APIs grows. Thanks to its speed of execution compared to User Interface tests, we will have a quick feedback of the state of the application, the possibility of having a great functional coverage and a great ease of use.

The new technologies have a wide functional coverage and are easy to include in the continuous integration.

14. Automatic User Interface tests

There are applications that, by their nature, do not require User Interface testing simply because they do not have an interface. However, most commonly, there is an interface with which we interact. The functionality offered by this interface can be validated with manual tests and with automatic tests that simulate the action of a user. Due to their nature, these automatic tests are usually slow to run and costly to maintain, so it is essential to have a good selection criteria of which functionalities will be covered by them.

TIP

If the application has a user interface, we recommend performing a battery of tests to validate that the user interface meets the acceptance criteria for which it was designed. It is possible that the unit, integration and system tests "underneath" have been automated, but it is possible that these tests indicate that everything is fine and yet a problem in the frontend or in the communication between the frontend and the backend causes the application to fail.

15. Performance testing

Performance tests are a type of non-functional test that evaluate the response time, stability, reliability, scalability, etc. of a system or application under certain loads. These tests do not validate functional aspects, but they allow us to know under what circumstances the system is going to offer a better or worse user experience, allowing us to debug it by finding bottlenecks, inefficiencies in the code, etc... and thus be able to size it correctly. By their very nature, they need a tool to be able to be executed.

CONSEJO

We consider it vital for the user experience that systems that are going to be subjected to variable access loads are subjected to performance tests with the aim of detecting non-functional problems that are impossible to find with other types of tests.

For this reason, we recommend developing, within the company's testing strategy, test batteries oriented towards the performance of the application and that these are included in the development pipeline.

16. Regression tests

Regression testing is a set of generally automated tests that ensure that the existing functionality of the application has not been broken.

They cover the entire application to ensure that changes made in one functional area have not affected others that, on paper, have not changed.

They are usually a selection of existing tests that run through virtually the entire system. They also usually include tests that validate that certain bugs found in the past do not happen again.

TIP

It is necessary in each release to ensure that core functionalities for the business that have not been modified by the development of that release continue to work correctly and that the defects detected and the historical ones are not reproduced.

We recommend having a battery of this type of tests in the development pipeline to increase confidence in the product to be deployed, being able to block the pipeline if a defect is found in the code.

17. Performance tests

By their very nature, performance tests need a tool to be run. This tool can launch them on demand, periodically and/or within a CI/CD server pipeline throughout the development cycle.

TIP

Experience has shown us that if automatic tests are not linked to a process that launches them periodically or under a specific trigger (a commit, a deployment...) they cease to serve the purpose for which they were created and may possibly cease to be used and become obsolete.

Our recommendation is that all tests, including performance tests, should be included at a point in the development cycle pipeline so that feedback can be obtained at at least one point in the development cycle.

3 TIPS

TEST ENVIRONMENTS

Discover the importance of test environments in software development and how they can help you ensure the quality of your applications.

We will discuss effective test environments that closely mimic the production environment, allowing you to test your software in realistic conditions before release.

18. Use of test environments other than the production environment.

A "test environment" simulates the production environment of an application. There we can deploy versions of the application in order to be able to carry out the tests that validate it.

Depending on the use to which it is put and the type of tests to be performed, the maturity of the code it hosts, etc., they can be classified into categories. For example: the development environment, the testing environment, the staging environment or the UAT environment (also called "pre-production").

With the use of techniques such as feature flagging, canary releasing, etc., development teams can carry out certain tests directly in Production.

TIP

Testing is essential to any software development methodology. A weak testing strategy can lead to buggy and flawed deployments. To mitigate these, testing is necessary and testing needs to be executed in a dedicated test environment.

We recommend having enough test environments to cover the types of tests required by the application we are developing.

19. The test environment and the application running in it have the same functionalities as in Production.

Test environments are designed to, once the application has been deployed, obtain behaviour and functionality as close as possible to that of production. Often, due to technical limitations, test environments do not have certain features that production environments do.

TIP

Deployments in these environments should follow the same steps as we need to do in production. In this way we will implicitly test the deployment steps as well.

It is possible that there may be environments where some functionality is missing that can be found in other environments. These undesired situations must be controlled and we must establish a mechanism that allows us to be sure that at all times we can launch in a non-production environment any functional cycle that may occur in production.

20. Automatic activation and deactivation of test environments with different versions of the application.

Test environments are used for various types of testing. It may be that we are interested in testing the latest development with a database similar to the productive one, that we want to carry out a UAT with the Stakeholder or that we need to carry out performance tests.

In all cases, it is recommended to have a tool that allows us to create and configure an environment from scratch, install a specific release of the application, have it operational and accessible for testing, generate a test report and finally, eliminate the environment, thus freeing up resources.

TIP

Current development methodologies, spurred on by Agile, encourage development in heterogeneous groups, squads, Agile teams... These teams generate versions of the application that must be quickly tested both manually and automatically to obtain agile feedback.

Having multiple permanent environments implies a high fixed cost for infrastructure, maintenance, management, etc. For this reason it is very important to have the creation, use and removal of these test environments automated, generally orchestrated by the CI/CD server.

1 TIP

INTEGRATION OF TESTING INTO THE DEVELOPMENT PIPELINE

We must incorporate automated testing at every stage of the development lifecycle, from build to deployment, to ensure software quality at all times.

It is the CI/CD system, the various scheduled development pipelines that orchestrate the execution of these automated tests at the appropriate time.

21. Integration into development pipelines

CI/CD stands for **Continuous Integration and Continuous Deployment** or Continuous Delivery. It is a software development practice in which code changes in the repository are carried out frequently and securely (CI) by automating builds and testing.

From this code, ready-to-install releases can be automatically generated in pre-production environments (C. Delivery). Many development teams stay at this point, running the deployment in production manually.

Finally, we can reach a state of maturity where the code is automatically deployed to production after a series of steps (C. Deployment).

Both CI and CD processes are carried out through a series of synchronised actions in one or several pipelines, where code downloading, testing, deployment, etc. are carried out.

TIP

Automating the integration of code and deployments to pre- and production environments greatly streamlines the delivery of value to development teams.

Additionally, CI/CD pipelines serve as the backbone for launching both functional and non-functional tests on code and validating it as a release candidate for production. In this way, tests are launched at the right place and time depending on their nature. Static code tests before deployment, service or integration tests before User Interface tests... And in the event that one of them fails, the whole process can "harakiri" itself and send the relevant notifications.

The speed of having a test or demonstration environment to respond in the event of a rollback, or the use of Continuous Delivery pipelines to achieve Continuous Deployment are other reasons for recommending the development of CI/CD pipelines.

2 TIPS

SECURITY AUDIT PROCESSES AND ASSOCIATED POLICY

A security audit consists of a series of analyses and tests to assess the security status of an information system against a security attack.

The objective is to detect vulnerabilities that could allow a third party to cause damage to the company.

Because application code, company infrastructure, personnel employed, etc., change over time, security audits must be carried out periodically.

22. Regular security audits

A security audit consists of a series of analyses and tests to assess the security status of an information system against a security attack. The objective is to detect vulnerabilities that could allow a third party to cause damage to the company.

TIP

Since the code of the applications, the company's infrastructure, the personnel employed, etc., changes over time, we recommend performing periodic security audits to ensure that there is no breach in the system and that the proposed actions of preventive measures, reinforcements, corrections, etc., take effect.

The scope of these tests, their characterisation and periodicity will depend on the current state of the application and should be one of the sections of the quality strategy to be implemented.

23. Security policy documentation

Security policy" according to RFC 2196 is defined as: "formal statements of the rules to be followed by those who have access to an organisation's technology and information assets".

It thus affects employees as well as suppliers or customers. It takes the form of a set of documents that explain what is expected of all of them, in order to prevent security incidents and reduce vulnerability.

TIP

Today it is virtually impossible to have a totally secure computer system. However, we recommend having simple and clear security policies within the reach of the people involved in the company's activity. In this way, we will have the tools to protect ourselves from unauthorised access, prevent errors or security oversights and, in general, minimise vulnerability.

1 TIP

LIST OF BROWSERS / OS, SUPPORTED BY YOUR APPLICATION

Knowing which browsers, operating systems (and their respective versions) your application can support is crucial to ensure an optimal user experience and maximise the accessibility of your software.

24. List of supported browsers

The product may require a browser and/or an operating system in order for the user to run the product.

In that case, the product design and testing validates that the product meets the requirements under certain combinations of browsers and/or operating systems.

TIP

In case the developed product requires a browser and/or an operating system for the user to be able to run it, we consider it very interesting to indicate which combination of browsers and operating systems are supported.

This clearly demarcates the line between what is a defect that requires correction and what is an unsupported operation.

4 TIPS

USE OF CONTROLLED DEPLOYMENT PROCESSES IN PRODUCTION ENVIRONMENTS

Having a controlled deployment process coupled with effective monitoring will allow us to identify possible incidents early on.

In this way, depending on their severity, we will be able to take appropriate decisions: continue with the deployment even with the incident, stop it, solve it and deploy again, perform a hotfix after the deployment, etc.

25. Feature toggles

It allows code with unfinished features or features that you want to activate on demand to be deployed to production. This makes it easier for releases to go into production and features can be activated when desired for all users or subsets of users.

The activation of these functionalities can be done without requiring a code change in production, using a backoffice, a database query, etc.

26. Canary Releasing

In the case of Canary Releasing, the balancer is more complex and targets a certain group of users within the server with the new release, such as company employees, betatesters, users in a certain zone, etc.

27. Blue-Green deploy

Controlled deployment processes enable Zero Downtime Deployment (ZDD) and above all increase the resilience of the system.

Blue-Green deployment is the standard and simplest deployment, where (to simplify a lot) the "Blue" server corresponds to the current code in production and the "Green" server contains the new release. Once the release is ready, a balancer progressively disconnects the load from the Blue server and directs it to the Green server.

In this way the new code is transparently available to all users.

28. Dark Launch

Finally, Dark Launch is even more complex, with two types of productive environments:

"Stable Production" and "Production Testing".

In this way, in "Production Testing" we can activate functionalities only for some users or even that are there but not visible, such as a hidden button that is activated when the page loads and that allows us to evaluate the performance of your transaction in a completely transparent way to the user.

1 TIP

DEFINE QUALITY CONTROL METRICS IN APPLICATION

For more than 100 years Lord Kelvin has been credited with the phrase:

"What is not defined cannot be measured. What is not measured cannot be improved. What does not improve, always degrades".

Let's establish a solid foundation for quality control in your application by defining clear and relevant metrics.

29. Quality control metrics

In the field of software quality it is necessary to make decisions to improve something that is degrading or at risk of degrading.

These decisions must be made based on information systems based on indicators that allow us to plan, establish objectives, control results, etc.

Some examples could be the **DLR (Defect Leaking Ratio)**, **MTTR (Mean Time To Recovery), MTTA (Mean Time to Acknowledge)**, etc.

TIP

Quality improvement is a living process, which feeds back on data to maintain continuous improvement. If we do not have the right indicators, we will not have the information to articulate improvement levers.

For this reason, we recommend carrying out a study of the weakest points in the development cycle and determining SMART metrics with which to establish KPIs on the basis of which we can work and evaluate results.

TIP

ENABLE PROCESSES FOR MONITORING, EVALUATION AND PRIORITISATION OF INCIDENTS IN PRODUCTION.

Evaluating and categorising incidents according to their impact and urgency allows you to prioritise and allocate resources appropriately for their resolution.

This guarantees a quick and efficient response to problems that arise in production and minimises their impact on users.

30. Real-time monitoring with a common priority assessment criteria

A monitoring system keeps track of the activities carried out by users, applications and other services of our application. In this way, we can supervise all the processes that are carried out and show the results in different dashboards, reports, etc.
In the event that the user experience is degraded due to defects in the code, hardware or network failures, resource exhaustion, configuration errors, data inconsistency, etc., monitoring has a system of alerts that activate protocols to solve the incident.

Incidents in the Productive environment can come from different sources: a monitoring system, detection by the internal team, feedback from the customer support team, etc.
In all cases, it will be necessary to reflect the incident as a task to be solved and to characterise it. This characterisation generally involves assigning a "Priority" and, optionally, a "Criticality".

The technical team generally has the vision to assign the criticality of an issue but it is the Product team who usually has the criteria to assign the priority (extreme cases are an exception), supported by the technical team.

This criterion may or may not be defined, shared and agreed.

TIP

We recommend that, within the monitoring system, different alerts be configured depending on the nature of the incidents.
Among other things, this will allow you to organise them by priority or notify different teams based on their type.

If you start from scratch, the suggestion is to begin with those incidents that may affect security, brand image or user experience, those that may involve economic damage or those that facilitate the activity of the development team.

These alerts, once the issue has been confirmed, should be prioritised, converted into tasks and incorporated into the development process.

One of the reasons for possible tension between the technical team and the product team is the criteria for prioritising incidents.

To reduce this tension, our recommendation is to define, share and agree on a procedure that reflects as objectively as possible the different criteria and assessments that are made to assign a priority to an issue.

There will be times when this will not be necessary, but others when such a document will help to eliminate ambiguities in criteria.

GET TO KNOW US

ABOUT US

We are **specialists in functional test automation** and its integration within the software development cycle since 2004.

We help our clients, reducing their time to market and increasing confidence in their deployments thanks to SQA processes.

- We reduce the time spent on product testing, reducing costs.
- We identify your bottlenecks before going into production.
- We act throughout the SDLC to improve quality globally.
- We analyse your product to detect inefficiencies or vulnerabilities.

LET'S WORK TOGETHER

Tell us about your case and we will get back to you as soon as possible. We will create an action plan and take your development to the next level. Send us an email to:

info@redsauce.net

www.redsauce.net

Pablo Gómez

Co-founder of Redsauce