UiPath Reboot Work.™

UAT Guidelines

DISCLAIMER: Please keep in mind that this document should serve as a general guidance towards tailoring the best practices that fulfill your project and business specific needs.

Prerequisites and Preparation

The UAT is when our entire process codebase is put to test and where we see first-hand how our implementation measures up to our customer's expectations. It's easy to see why the UAT can be on of the most stressful parts of an RPA implementation and one of the most likely to be subject to delays and unforeseen complications.

However, with proper planning and preparation, a lot of the risks associated with it can be mitigated or even eliminated. Here are a few things that should be done before any UAT:

Preliminary Testing

UAT is the stage where we thoroughly test and validate our automation. However, starting this phase after some preliminary testing has been done can make the difference between properly covering all the testcases and finding time to write the necessary documentation and frantically jumping between fixing bugs and struggling to keep up with the customer's feedback.

Therefore, it is important that we do our due diligence and try to make sure that we eliminate as many bugs/issues as possible before the UAT commences. Ideally, we would want to focus on the following types of testing:

Testing Type	Description
Robustness Testing	As soon as we reach a point in the development where the automation can be ran end to end, we should start to do some **exhaustive testing** so that we **identify and fix any quirks and sporadic issues** that could occasionally appear in the UiAutomation Part of our process. • To performs robustness testing, all we must do is to periodically process a relatively large number of end-to-end transactions that make our automation go through all the screens.

Testing Type	Description
Robustness Testing	• This type of testing does not require our full attention, if **logging and error handling are properly implemented, logs and error screenshots should be enough to investigate and fix any potential bugs.** (so this can be done on a non-production robot while we're working on something else or simply when we're not using the machine) • Even if we do not have a very large dataset, we can simply have the robot run through the same transactions over and over again, **our objective here is to have the automation go through all of it's screens as many times as possible** and **iron out all of the quirks**, not fully testing all possible scenarios. • Once our robot is reliably processing dozens (or even hundreds) of transactions we know that our automation is approaching an optimal level of robustness. **Stress testing is closely tied to Test Automation and our automation's reliability can be drastically enhanced by leveraging the UiPath Test Suite (more on that below).**
Performance Testing	It's crucial we ensure that our automation is efficient and that it functions properly even under particularly large or complex workloads. To check this, we must perform the following tests:

Testing Type	Description
Robustness Testing	• **Understand the size/complexity limits** of the data that the automation is expected to process and test that the automation runs smoothly with input data that is close to the limits or exceeds them. (For example, if an automation is expected to process excel files that have up to several tens of thousands of rows, I need to test that it can successfully process.) • Test to see that the automation functions well in an environment similar to production. So it's necessary that we test **our automation on a machine with similar hardware/software** specs to the production machines and with the **same versions of the applications being automated**. • For **Unattended Automation**, we should **publish the automation** and check that it can **run end-to-end on a non-production machine** similar to production environment. • For **Attended Automation** especially we need to test that the automation: o is fast and efficient (this is a critical factor in the success of any attended automation) o works well even if the user is trying to interact with the application at the same time (if that's not possible, we need to test that block-user input works well)

Testing Type	Description
Robustness Testing	o is not affected by the screen resolution (simply running the automation under different resolutions like **1920×1080, 1366×768, 1440×900** should be enough) o that the user won't **wait for an excessive amount of time** in the case of an **unexpected exception** (For example, if an element is not found, having to wait a full 30 seconds for a timeout exception is too much in most attended automation scenarios). So we need to **cause unexpected behaviors while we're running the attended process** (for example closing the application halfway through the execution) and **check that the robot handles the error it in a timely manner** Any other configurations or particularities under which our automation is expected to run, should be tested too.
Stress Testing	It typically refers to measuring the robustness and performance of our automation under extremely heavy load conditions. Concretely, stress testing implies: • **Measuring the average transaction time** and estimating the amount of time the automation needs to clear the amount within the customer's SLA.

Testing Type	Description
Stress Testing	For automations that are supposed to run on multiple machines simultaneously, testing that the automation can run in parallel on multiple robots is important, as a lot of complications can appear whenever multiple robots are trying to access the same application or resource at the same time.

Stress, performance and robustness testing will typically be continued as the UAT progresses but starting it during the Development phase gives us the opportunity to eliminate the initial bulk of issues early and to focus on thoroughly covering all the testcases and fixing any bugs/issues found while doing so.

Code Review

Ideally, before the User Acceptance Testing can begin, we need make sure that:

- our code is consistent with our teams' best practices and design standards
- any obvious performance problems are picked up on and solved.

Ensuring all of the above is a lengthy process that begins with the Solution Design of the process and continues throughout the entire development phase, where ideally the solution architect or a senior developer is involved in the progress of the automation through status meetings or other similar means. However, at the end of the development, a final thorough check to catch any remaining problems is necessary and one of the most rigorous yet efficient ways of achieving it is a code review.

The applications and performance or security concerns will vary greatly from one process from the other, but below we can find some very general things that need to be checked for every process:

- efficient usage of the process framework (**REFramework** most likely);
- **Flowchart/Sequence/State Machine** are used appropriately;
- **No more than 2 nested IF activities** in sequence - if complex logic is used, refactor the code to either a Flowchart or use Invoke Workflow activities or use ternary IFs;

• **Activity names are descriptive**, annotations are added for non-obvious behaviors, activity names should reflect the action taken;
• **Detailed logging and exception handling** are utilized (BRE vs AE);
• **Recovery/retry** is planned for errors at different stages of process;
• Use **assets** for variables that are likely to change and used many times;
• Utilize **configuration file to share constants** that are shared across workflows;
• Use default arguments/conditional loading of configuration for the workflows.

The 2 major challenges of every code review are making sure that **no issues are overlooked** and that the **feedback** is given in an **easy-to-understand and constructive way**. Given the large number of things that need to be checked, keeping track of everything is very difficult and while experience can help us deliver feedback constructively, its best if we have a standardized support that we can stick to. To this end, we have prepared **a code review document template** that can filled in with the feedback from the code review.

Not only does this document give us a **complete list** of all the **items that need to be inspected** but it also helps us **deliver quality feedback consistently**.

Observations:

• A common practice is to conduct the UAT, but in the eventuality that significant flaws which require significant code refactoring and rework are uncovered, then all of the testing done until that point, needs to be redone, so the code review itself should happen before the UAT.

• A good way to optimize this process would be to use Workflow Analyzer to create your own set of validation rules that developers can use on before the code review to automatically spot inconsistencies with the standard best practices. Our official documentation contains an in-depth guide for Workflow Analyzer: https://docs.uipath.com/studio/docs/about-workflow-analyzer.

• If you want to learn more about how to use the Workflow Analyzer feature in Studio, you can access the Academy Course here.

• Please keep in mind that the UiPath Best Practices for code review are meant to help you in establishing the best practices that suite your project specific needs.

Test Cases and Data

A **Test case** is a specification of the input, execution conditions, testing procedure, and expected results that define a single purpose.

Test Data Requirements
One of the most important things needed whenever we conduct the UAT of a process is test data. Usually, our customer's team needs time to prepare the test data for our automation so not only do **they need to be told in advance** that they are responsible for providing it, but since UAT is a thorough and complex process it's likely that without our help, the dataset we will be provided will be either too small or incomplete. The test cases doc-ument provided to the business should contain all of the information necessary for them to provide a complete dataset, but since there isn't necessarily a one-to-one match between test scenarios and types of test data, we need to provide them with a complete list with all the types of test data required in order to perform the UAT and the minimum number of items required for each type as well as any other special requirements or mentions.

This list should be short, brief and easy to understand. A template can be found below:

This should be prepared at the same time with the test cases/ test scenarios and this list should be submitted to the business as soon as possible.

Observation:

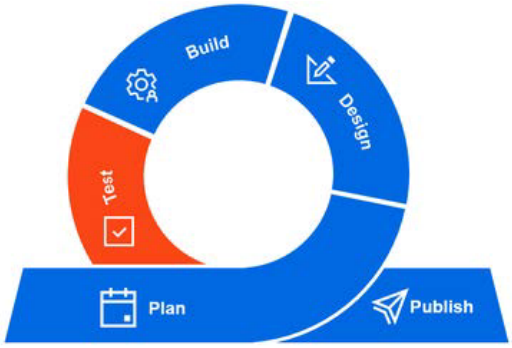
Please note that none of the items above are extremely time consuming, so starting to work on them during the second half of the development phase should not impact the amount of effort required to finish the automation (not that it would matter, since the Solution Architect should take these into account when giving an effort estimation for an automation). But, skipping these preparations or planning to do them during the UAT itself could push back the business signoff and thus seriously impacting the timeline.

For example, insufficient robustness testing could mean that some easily preventable bugs are discovered late in the UAT which would in turn rightfully cause the business to ask for additional testing/ UAT sessions (which would push back the timeline by several days) or, even worse, these bugs could go unnoticed until after the robot has been launched in production.

So, doing these preparations before the UAT itself starts is of utmost importance.

Leveraging Test Suite

The UiPath Test Suite is built to cover the end-to-end (and continuous) testing process. Using Test Suite, one can automate and centralize testing to ensure the quality of every automation before they go live.



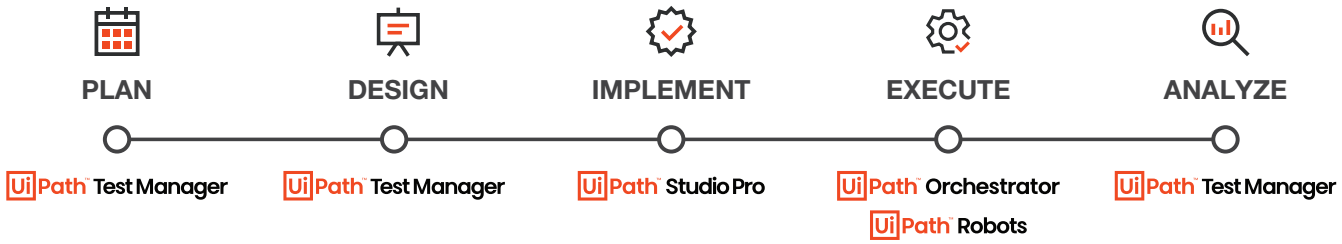
RPA Testing is a continuous process related to an RPA project where we check whether the developed process matches expected requirements, has the expected behavior and the expected output.

The quality of an RPA Project can be determined by testing all the possible scenarios with multiple data entries. Based on the results obtained, improvements can be made before deploying the process into production.

Test Suite benefits:
• Testing can be done 3-4 times faster during the initial deployment and increasingly faster for subsequent updates.
• Data driven tests combined with activity coverage metrics enable more thorough testing, reducing risk of production issues.
• Resilience minimizes maintenance required, freeing the team to focus on building more automations.
• Simulate the behavior of data in controlled ways by mocking parts of your automation.
• Create synthetic Test Data in a matter of seconds. Data is typed and stateful and can be created/used dynamically or prepared in central storage.
• Create continuous deployment cycle in RPA.

Testing flow in Test Suite:
1. Test Automation Architect defines Test Cases in Test Manager and optionally documents them with Task Capture
2. Based on the documentation, the developer automates defined Test Case in StudioPro
3. Developer links Test Case in StudioPro to Test Case in Test Manager

4. Developer publishes the Automated Test Case from Studio Pro in Orchestrator
5. Test Automation Architect creates Test Set in Orchestrator
6. Test Automation Architect links Test Set in Test Manager



When designing and implementing, one should be aware of the following guidelines:
• Test cases should be autonomous, one test case should not depend on another test case's run;
• Create small workflows, that tackle the smallest number of actions possible. In this way, it will be easier to understand it and unit test it;
• A test case should have one specific purpose - each test workflow should contain only one verification;
• Every feature should have a unit test; if exceptions can happen, also create a separate test case for each of them;
• To increase reusability between Test projects but also between Test and RPA projects, one should use libraries/Object Repository whenever possible;

Test Data Management
When procuring test data for our automations, one of the most common approach would be using production data meaning having a copy of the production database and use that for testing purposes. Although it's an old practice, there are still a lot of customers still doing it. That's no longer allowed in a lot of countries because of privacy regulations such as GDPR. With this approach there is the risk to have low coverage when testing as well as the risk of data becoming outdated.

One solution is of course to still use production data and anonymize it. Anonymizing meaning masking it, but there is an issue with that for complex applications that have a lot of databases relations and having a consistent mask for all those tables for your data.

UiPath came up with **Synthetic Test Data**. Using the **UiPath Testing Activities**, one is able to create typed data, meaning the data, of course, follows a certain schema, otherwise, it would not be relevant for testing purposes. And all data is created synthetically so there's no relation to any real users or any real production data. The advantage with this approach is that one can **systematically create data** that **leads to high coverage** so one can make sure that the test data **addresses certain edge cases**, certain scenarios that might lead to an issue. Another benefit is that there are **no data regulations being violated**.

Guidelines for UAT Sessions

The UAT phase is **a joint effort** between the **development team** and the **customer's team** and given the relatively large number of people involved compared to the other phases of the project, proper planning is of the utmost importance. Therefore, setting **an expected end date for the UAT** and asking the business team to commit to provide test data and feedback so that this timeframe can be respected is essential. Usually, the complexity of a process is tightly correlated to the length of the UAT period, but the timeline might extend depending on SME availability issues or other process specific issues.

Still, for an average process, this is how much the UAT usually takes:

Process Complexity	Average UAT Duration
1-2 weeks	around 3 days (but since even a simple process requires multiple UAT sessions, the effort could easily spread over an entire week)
2-3 weeks	1 week of UAT
4-7 weeks	2 weeks of UAT
Over 7 weeks	3+ weeks of UAT

UAT sessions frequency

The UAT process is generally **centered around a series of meetings** in which the d**evelop-er and the SME** will be **testing the robot** and **checking** to see that the automation **passes all testcases**.

When deciding on the frequency of the UAT meetings, keep in mind that between those sessions the developers will need to have the time to fix bugs, implement feedback from the SME and to perform through testing to ensure that the latest changes are functioning properly. So, in the case of a complex/laborious process, **daily UAT Sessions** might be too much and you might want to opt for **holding these meetings every other day** instead.

At the start, the meetings for the entire UAT period should be scheduled, if that is not possible, the UAT sessions should be planned with **at least a week in advance**.

Organization
The first few UAT sessions should be the ones where the **most general testcases** are being checked and the **automation is run live** so that the entire team can see the robot at work and get more familiar with it.

Later UAT sessions should be focused around **covering all the testcases**, especially the niche ones and **running as many tests as possible**, so it's best if the SME will regularly provide test data (ideally, daily) and that the **developer runs these tests outside of the sessions** themselves. Those should be **focused around discussing** the SME's **feedback** and re-doing live some of the testcases that previously failed.

At the end of each meeting, a set of action items should be determined and what are the items that will be solved until the next session. If the developers decide that there's not enough time until the next meeting to make meaningful progress, then postponing it should be discussed.

Duration
Typically, UAT sessions last between 1-2.5 hours, but for most processes, we have found that 1.5 hours is an ideal duration.

If running a process takes longer than this, instead of prolonging the UAT sessions, the actual process execution should be done outside the meetings by the RPA developer and the UAT should focus solely on validating the output.

Cooperating with the SME(s)

SME(s) roles and responsibilities

As mentioned before, the UAT is very much a joint effort between the development and the business teams and since we, the dev team, are the more experienced party, it's our duty to ensure that the SMEs and Process Owners understand their responsibilities and that they fulfill them in a timely manner. Before the UAT begins (preferably as soon as the PDD is signed off), the business team needs to be clearly told what their responsibilities are. A short summary can be found below:

1. **Providing Test Data**: Since the SME is the most familiar with the process and has access to the production systems **sourcing test data is part of their responsibility**. Obviously, completely covering all of the testcases is going to be tough so the **development team should provide them with a complete list of item types** that need to be provided. (see Test Data Requirements section)

2. **Test Cases document review and signoff**: putting together a clear and well structured list of testcases is the responsibility of the development team, but since the SME/Process Owners are required to signoff process, it is important that they have a chance to weigh in on the tests we plan to perform in the UAT, so they need to be asked to closely review the testcases document and to weigh in on it.

    1. Since this might be the first time the business team sees such a document, it makes sense for the **developers to walk them through the testcases document** and explain its contents before the doc is submitted for review
    2. Once a testcase is tested thoroughly, **the SMEs should mark it as passed** in the testcases document (if keeping this document updated throughout the UAT is not possible, at the very least, **this should be done in the last few UAT meetings**)

3. **Validating robot output**: This is something that the SME will be doing constantly during the UAT phase and that, depending on the process, could either happen live during the meetings or by email. Regardless, **setting an expected response time** for the SME feedback is very important in ensuring that the **UAT moves forward at a good pace**.

4. **Process Signoff**: This is the most important item, as **all of the points above ultimately lead to this one**. It's essential to establish right from the beginning that **this responsibility lies with the SME** and that the signoff should be **given by email** too.

The points above **should be discussed in a meeting** where the business team has ample opportunity to ask questions about their contribution in the UAT, but these responsibilities should also be **sent to them by email** so that they don't lose track of them.

Handling Scope Creep

As the UAT progresses and the SMEs get more comfortable with the way the automation work, some improvement ideas and requests for future changes are bound to arise. Some of these ideas are small and very easy to implement, but some could represent substantial changes that would require significant rework and additional testing, so right from the start we need to communicate that the purpose of the UAT phase is to **test that the automation behaves according to the requirements documented in the PDD**, and not to assess the degree of usefulness of the automation (this has already been done during the Analysis phase) or to come up with improvement ideas.

Obviously, we want the business team to benefit as much as possible, so we can mention that if time permits it, **we will accommodate some small changes** that **only require us to redo a small amount of tests**, but that u**ser acceptance testing and bug-fixing is our priority** and that we don't take a commitment to do any additional development in this phase.

If a change is too substantial to be done along the UAT effort, then this needs to be **communicated right away** to the business.

For starters, when dealing with changes that are substantial in scope, it's good to separate these changes in 2 categories:

• **Nice to have changes** - things without which the automation would still fulfill its primary purpose but that if implemented would improve its usefulness. **These need to be documented** and added to the future improvements section of the PDD and at the project a discussion about a change request that would include these additional requirements should be had with the business.

• **Must have changes** - things without which the automation would not bring any significant benefits - if this happens this is most likely due to some parts of the process being excluded from the PDD. **These need to be documented** and **the impact and effort** associated with the change **need to be clearly outlined**. Usually, **a discussion** needs to be had at **a higher level** with the **project management team and the stakeholders** where the change request should be discussed.

Communicating efficiently with the SMEs

To achieve this much needed level of cooperation between us and the SME(s), good interactions are paramount, however the difference between technical people and non-technical people can often lead to an inefficient communication stream, which in turn will result in delays and frustration. **For good interactions and communication with the business team**, here are a few notions that a technical person should be aware of when discussing with the SMEs or other non-technical roles:

• Keep in mind that a technical person (aka. a developer) sees the process differently, without thinking so much about the business side, so there needs to be **an adjustment from both parties during the communication**. Even when discussing technical limitations or matters strictly related to the performance of the automation, we need to **transpose everything down to what the impact is on a functional/business level** when we communicate it to the SMEs.

• At the start of the UAT, always do an end-to-end demo of the automation, this gives the SMEs the opportunity to understand and get more comfortable with the technology, which in turn makes them more cooperative (especially when giving the final sign-off)

• When discussing the process, make sure you always make the **distinction between the AS IS Process** walkthrough and the **TO BE** finished process result. Especially for complex processes, these 2 are can be very different things and when testing the TO BE process, it always helps to clearly **explain the distinction** and specify what is the **corresponding AS IS step for each part of the TO BE** process.

• When discussing the test data requirements, provide all the scenarios that you can think of (ideally using the template in the **Test Data Requirements** section), when discussing them, maintain a structured top-down approach. Below, you can find the suggested order in which to present this information:

1. Input Data from these streams (*as it is currently done by SMEs*)
   - Stream A
   - Stream B
   - Stream C
2. This is how the process works (*as it is currently done by SMEs*)
   - If there are ramifications/corner-cases, cover them briefly
3. Expected Results (*as it is currently done by SMEs*)

Going through the process one more time when requesting the test data requirements is good practice as it will help you overview the process in your mind and identify any potential bugs or misbehavior of the solution as everything links together.

Developer guidelines

Bug Tracker

A **bug tracking system** or **defect tracking system** is a methodology for keeping an up to date report with the software bugs in an development project. It may be referred as well as a **issue tracker**. A major component of a **bug tracking system** is an online document that records facts about known bugs (**title**) or reported bugs. Facts may include the time a bug was reported (**date**), its **severity (Low, Medium, High)**, the erroneous program behavior, and **details on how to reproduce the bug**; as well as the identity of the person who reported it and any programmers who may be working on fixing it. Typical **bug tracking systems** support the concept of the life cycle for a bug which is tracked through the status assigned to the bug. This is part of the project management methodology and allows tracking and communicating the status of each reported bug.

#	Test Case #	Submitter name	Title	Description/ Behaviour	Owner	Solution	Impact	Status	Occurrence	Submission date	Resolution date	Production deployment date	Comments

- types of tests that need to be performed during the UAT
- definition of done
- how to handle scope creep/extra changes
- guidelines on how to do production testing

UAT Particularities

Companies activating in fields such as pharma, finance or accounting may require implementing a set of additional rules when automating processes that handle personal data, sensitive or confidential information.

Worldwide, there are common regulatory standards that companies implement, in order to ensure privacy and security of the usage and disclosure of sensitive information (for example, financial situation, healthcare-related information).

Here are some examples of standards found in practice:

• **GxP** - good practice guidelines and regulations to ensure that medical devices, drugs and food are safe for. Applicable in domains such as pharma, medical, life sciences and their suppliers, aggrotech, cosmetics etc.

• **SOX** (Sarbanes-Oxley Act) - US federal law for public companies to provide proof of accurate and data-secure financial reporting.

• **HIPAA** – healthcare regulations to ensure the electronic medical records are secure and not accessible by unauthorized persons.

What do these regulations mean for RPA?

These standards require an external person (i.e. not the developer or the business team) to validate the requirements and the robot execution.

• The Validation team will provide additional documentation that will need to be filled by the Development team.

• Prior to UAT, the Validation team may require the developer to perform also unit testing and functional testing in a controlled environment and to monitor the results. This session(s) can be similar to the ones we do for UAT, except for the fact that the business team are optional attendees. The unit tests and functional tests will be documented in the same way we do for UAT and recording can also be required. The Validation team can also ask for some additional compliance test to check if sensitive is safely handled (for example, passwords are securely stored, personal data is not added to a queue item, the robot account does not have more rights to applications than intended etc). The tests can be created using Test Suite, if it is available; if not, they can be regular workflows that invoke other workflows or modules and check the actual results against the expected ones.

• The UAT will be supervised by the Validation team, which will also provide the signoff, along with the business SMEs.

• The Validation team will also be monitoring the Hypercare to ensure the regulations are followed.

All the project documentation needs to be well-defined and developed according a clearly defined SDLC methodology, defined and approved by the Validation team and the client. In other words, the robot's requirements, system access, designs and actions should be thoroughly documented (at a keystroke level). The company will get external audits based on the standards they adhere to and the auditors will want to see how the automation steps match the actions of a person previously performing the manual process. Output data verification and thorough reporting can also be very important.

Keeping this in mind, at the beginning of the project make sure to add a contingency between 20 and 30% if the project is regulated.