RPA development approach - preparation

Contents

[1. Environment setup strategy 2](#_Toc483914049)

[a. Grouping by application groups 2](#_Toc483914050)

[b. Grouping by process 2](#_Toc483914051)

[c. Hybrid grouping 2](#_Toc483914052)

[2. Robot authentication method 2](#_Toc483914053)

[a) Using one generic technical account for all robots 2](#_Toc483914054)

[b) Using one technical account for each robot 3](#_Toc483914055)

[c) Using human user accounts – only for FOR 3](#_Toc483914056)

[3. Storing credentials 3](#_Toc483914057)

[a) Storing the credentials locally in Windows Credential store 3](#_Toc483914058)

[b) Storing the credentials as assets inside Orchestrator 3](#_Toc483914059)

[c) Using a third party for storing credentials (eg. CyberArk) 3](#_Toc483914060)

[4. Developer collaboration 3](#_Toc483914061)

[a) Choose the collaboration platform for the RPA developers. 3](#_Toc483914062)

[b) Decide on a common collaboration approach for developing workflows. 3](#_Toc483914063)

[5. Development methodology for prioritization 4](#_Toc483914064)

[6. Reusable components 5](#_Toc483914065)

[Recommended approach 5](#_Toc483914066)

[Alternative approach 5](#_Toc483914067)

7. Naming strategy…………………………………………………………………………………………………………………………..………6

8. Best automation practices document…………………………………………………………………………………………………7

# Environment setup strategy

After initial high-level process analysis, a decision should be made on grouping the robots in environments.

The three possible approaches are:

1. Grouping by application groups (example: environments with robots having Excel and SAP)

PRO: A better usage of the robots can be achieved, one environment can run multiple processes having the baseline applications deployed, staying busy for a longer time and benefiting from the dynamic allocation of the scheduler)

CONS: Not efficient when having multiple application combinations between processes

1. Grouping by process (example: one environment for SAP\_Invoice\_AR process)

PRO: Each process will have a dedicated environment, no mixing of the robots, always ensuring 100% availability of the robots for the dedicated process

CONS: Sometimes can lead to idle robots

1. Hybrid grouping – a combination of the above methods

PRO: the most flexible approach

CONS: must be very well documented to avoid confusion when troubleshooting

# Robot authentication method

Decide a per-process strategy on the accounts the robots are using.

The three methods are:

## Using one generic technical account for all robots

PRO: Less configuration

PRO: Less licenses

CONS: no granularity in access control and logging

CONS: the account could be locked out or compromised

CONS: parallel sessions from the same account might not be supported

## Using one technical account for each robot

PRO: better access control

CONS: might require additional application licenses

## Using human user accounts – *only for FOR*

CONS: although robot actions are logged, auditing could be more complicated

Cons: compliance restrictions

# **Storing credentials**

Decide a per-process strategy on storing the credentials that the robots are using.

The options are:

## Storing the credentials locally in Windows Credential store

PRO: Can be changed by the robot

CONS: Not centralized, have to be defined on each machine

## Storing the credentials as assets inside Orchestrator

PRO: Centralized, accessible to the administrator

PRO: independent on the computer the robot is running on

CONS: Might not be compatible with all corporate security policies

## Using a third party for storing credentials (eg. CyberArk)

CONS: Additional costs involved, could be used to fulfill strict corporate security policies

# Developer collaboration

## a) Choose the collaboration platform for the RPA developers.

You can opt between TFS and SVN, both having native integration with UiPath Studio. TFS is the more flexible solution. TFS is free to use for up to 5 users.

Suggestion is to use TFS Explorer for branching and to sync only one branch to Studio.

Group the projects by departments and have a separate project for reusable components.

## Decide on a common collaboration approach for developing workflows.

The suggestion is to have one person work on one process, if possible. If not, the solution architect must allocate the components to the developers, deciding the list of input and output parameters for each workflow.

Decide on the code peer review approach and the hierarchy for code approval, e.g. Workflow review will be done by one colleague (RPA Dev) and the Solution Architect before promoting to QA.

# Development methodology for prioritization

Do two sets of prioritization:

1. First set is to choose the processes that will have a good return of FTE, are stable and don’t present technological challenges.
2. The second step is to choose the parts of the process that can be automated with the minimum effort returning an overall FTE benefit at process level. This can be done by analyzing the decision points inside the process map that lead to exception cases, calculate the effort for implementing the exception and decide on the FTE gain for the exception versus the actual effort needed for the implementation.

Never try to automate 100% of the process if the exceptions are very low volumes(number or frequence) and exceptionally high effort.

PRO: a faster FTE gain, so faster ROI

PRO: saving testing time.

CONS: knowledge is lost from the end point of the min implementation to resuming the development

CONS: full regression testing can be necessary

CONS: the impact for error reporting and manual handling can be high

***Example of analysis:***

Process1 – 50 FTE total

No exception – 80% - 10 MD

Exception 1 – 10% - 10MD

Exception 2 – 5% - 1MD

Exception 3 – 5% - manual work, no workaround

The first step of the implementation will include the No exception case and Exception 2, gaining 85% FTE (42.5 FTE) with 11 days of effort.

Exception 1 will be postponed to a later phase of the project.

# Reusable components

Define the strategy for reusable components.

a) Recommended approach is to have one project containing all the reusable components. An environment containing all the robots called ReusableEnvironment will be created and the package will be published on all the robots.

The reusable components will be invoked by the local path used by the robot to unzip packages from the Orchestrator: C:\ProgramData\UiPath\Projects\ReusableComponents.version\component.xaml

The version of the ReusableComponent package will be maintained as an Orchestrator asset and inserted into the filepath at runtime.

To publish a new version of the reusable components package, you must:

1. Upload the new package to Orchestrator
2. Run the workflow on the ReusableEnvironment environment (the main file can be empty, this step is needed to synchronize the project with the robots)
3. Update the Orchestrator asset containing the ReusableComponent version number

Be careful to apply the same strategy for Dev and QA environments as well!

**Best practice:** Each reusable component workflow, once in production, will not be modified. A new version of the xaml workflow will be created instead, or the code can be copied locally inside the main workflow.

Alternative approach bypassing Studio native publishing mechanism:

Use a tool that reads xaml dependencies (this can be a robot script) to identify the reusable components inside the workflow and use another solution (e.g. Jenkins) to create the package folder structure. After this step, use a command line script to invoke nuget to create the package and upload to Orchestrator.

# Naming convention and strategy

* Variables:
  + Use one variable for one and only one purpose.
  + Minimize the scope of each variable.
  + Keep statements that work with the same variable(s) as close together as possible.
  + Instead of using default values for variables, use specific Assign activities in the workflows, for a better readability
  + Variables will **always** **have** **meaningful** names. The variable name should fully and accurately describe the entity the variable represents. State in words what the variable represents.
  + We will use **Camel Case** for naming variables. This practice used compound words, no other characters between the words, where each word will start with a capital letter. Ex: TransactionNumber, FilePath, ReportName etc
  + The length of the variable name should be between 6 and 20 characters long. If you feel that 20 characters are not enough, consider abbreviating longer words. Shorter variables names can be used when using a local scope (like: index, file, row)
  + Datatable object: Start with **dt\_** prefix followed by the normal name. Ex: dt\_Employees, dt\_Reports
  + Boolean type: Give Boolean variables names that imply True or False. You can use the prefix **is** followed by the name. Ex. ApplicationExists, isRed, isFound etc. Always use positive names, negatives names (Ex: notFound) should not be used.
* Arguments:
  + Same guidelines as for variables, with the below differences:
  + Each argument will/can have a prefix In, Out, IO
  + Use default values for arguments either for testing individual workflow files, or, in case of reusable components, for using default configuration. Specify what is the default configuration in the description of the reusable workflow file.
* Workflow Files
  + **Camel Case** naming
  + Framework files come already created and are standard (including Main.xaml)
  + Test\_Framework files – Use the suffix **\_test.xaml** for a workflow file which run tests. Place these files in the Test\_Framework folder
  + Use number prefixes to emphasize the calling (invoking) hierarchy of the Project, where the root is always Process. Ex: First invoke is “1. Login.xaml”. In “1. Login.xaml “ there is another invoked file which is named: “1.1. OpenApplication”
* Projects
  + Group by department: Ex: AP, AR