<Company> System, Inc.

Intelligent Automation

Bot Development - Best Practice Guidelines

Confidential

Versioning

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Description | Author |
| 28 Nov 2018 | 1.0 | Discussion version – David, Srinivas, Robert | Mayank Patel & Shubhen Sarangi, Datamatics |
|  |  |  |  |
|  |  |  |  |

Contents

[Introduction 4](#_Toc531204094)

[Purpose and Audience 4](#_Toc531204095)

[Planning and Documentation Review 4](#_Toc531204096)

[Understand technology limitations 5](#_Toc531204097)

[Modularity 5](#_Toc531204098)

[Avoid ‘hard coding’ 6](#_Toc531204099)

[Keep It Simple 6](#_Toc531204100)

[Readability 7](#_Toc531204101)

[Naming conventions 7](#_Toc531204102)

[Code review 7](#_Toc531204103)

[Testing 8](#_Toc531204104)

[Commenting 8](#_Toc531204105)

[Decoupling and Loose Coupling 9](#_Toc531204106)

[Error Handling 9](#_Toc531204107)

[Security 10](#_Toc531204108)

[Tips 10](#_Toc531204109)

[Don’t Underestimate Complexity 12](#_Toc531204110)

# Introduction

Coding standard and best practices are a set of informal rules that the software development community has learned over time which can help improve the quality of code.

Following a consistent coding standard helps improve the quality of the overall Bot design. The more readable source code is, the easier it is for someone to understand and maintain that code. Following a consistent style allows other developers to step in and help with maintenance or new development.

The goals of a good coding standard are: Improved code maintainability and Code quality; in particular: testability, supportability, portability, security, efficiency, robustness, loose coupling.

# Purpose and Audience

As <Company>’s Intelligent Automation COE Team will leverage a mix of Business Bot Developers and COE Bot Developers, this document is intended for reference by both audiences.

Business Bot Developers who may not have formal training and exposure to software development practices will benefit from guidance about basic software and bot development practices that will provide them a common language to communicate with COE Bot Developers and also ensure that certain basic principles are applied to achieve high quality Bot scripts or configurations as applicable.

COE Bot Developers are expected to be trained in formal software development practices. It is expected that COE Bot Developers shall leverage their formal software background and best practices applied in other enterprise software development projects to the Bot Development projects at <Company>. In addition, this document is meant to provide additional key areas for them to consider when developing Bots using RPA (Robotic Process Automation), Artificial Intelligence & Machine Learning, Chat Bot and other Intelligent Automation platforms or techniques.

# Planning and Documentation Review

 While developers work on the technical aspects of the solution, it is vital to always keep the end goal of the customer/automation in mind. All Bot Developers should review documented ‘As Is’ and ‘To Be’ processes prepared by COE and Business Users to understand the context and primary objectives. Few key documents to refer to include the Functional Design Document (FDD) or Process Design Document (PDD), the ‘As Is’ and ‘To Be’ processes, the business case with process performance expectations, etc.

To start the development process, the Solution Architect or a Senior COE Bot Developer shall prepare the Technical Design Document (TDD). This design shall form the basis of all the development work to be done by the Business Bot Developers &/or the COE Bot Developers. As a matter of sound technical practice, the Technical Design Document must be prepared, discussed, updated and approved prior to any development work commencing. This will help avoid technical challenges in the future and take into account key considerations such as solution expectations, technical feasibility, product capabilities and <Company> IT standards. As part of the Technical Design review, the topology on which the solution shall be deployed should be well understood by Bot Developers especially the COE Bot Developers as this may not always be intuitive to Business Bot Developers.

After Bot Developers are assigned components of the solution to be developed, they must ensure that they document on paper, or electronically if possible, for their own reference the flow of the components that are assigned to them. Doing so helps developers understand the end-to-end expectations from them, including identifying any gaps or lack of clarity in their understanding of the details to be implemented. As a matter of good practice, developers should discuss with other bot developers any specific development challenges they are facing during scrum meetings or with the Intelligent Automation Application Owner to continuously upgrade their and the Bot development team’s understanding of how to approach specific Bot development challenges.

Please refer to COE Governance Document: “Bot Development Process” > ‘Documentation Requirements’ for list of relevant documents.

If you fail to plan, you plan to fail. The same goes for RPA development as well.

It is not advised for one to jump straight into RPA development without first considering the high level design. What this means is that you need to identify beforehand the various components that altogether make up your solution, and how these components interact with one another.

A key reason for planning is to identify critical challenges that team may face during development process and to plan beforehand and in future to address such challenges. This may include reaching out to support resources from vendor if required keeping in mind project timelines and ensuring any foreseen technical or feasibility risks are addressed prior to team getting deep into developing the solution.

# Understand technology limitations

 Just like with other enterprise software products and platforms, no matter which Intelligent Automation or RPA platforms you use, there is no one perfect tool.

There are always limitations to what commercially available RPA software can do, despite what your vendor may tell you. Vendor can provide capacity or limitation references which should be referenced for your Bot design.

As Intelligent Automation, RPA, AI & ML etc., are still emerging areas, it is vital for the Bot development team to stay abreast of the emerging functionality, new technical capabilities and bug fixes to their Intelligent Automation development platform products. Adequate attention should be given to ensuring that Bot Developers continuously upgrade their technical skills both in specific products and also in overall Intelligent Automation space.

# Modularity

 Modularity is a key construct in software development. Modularity in the software development context means simply to break down a complex script/program into bite-sized (more manageable) components. Modularity of the Intelligent Automation code should be a key agenda item as part of the planning process.

Following are few important benefits of modularizing your automation code:

* There are often common steps that occur across different processes, for example, logging into an SAP system. Rather than generating these steps from scratch every time you automate a new process, a smarter and more efficient way is to create a separate workflow that does so, and add this to your automation library as a distinct Bot or code-snippet.
* Modularity supports fine granular development and testing. This is especially useful when dealing with complex processes, as you can independently assign the development and testing of specific components to individual developers, mix of Business Bot Developers and COE Bot Developers or also integrate modules from external vendors.
* Over the course of time, the reusability of provided by modularity also makes it simpler to accelerate the development cycles thereby achieving greater ‘Bot Velocity’ i.e, velocity at which Bots get developed in future.
* Modularity is also very beneficial to Business Bot Developers as they can pick ‘black box’ reusable Bots or modular code snippets that have been prepared earlier by skilled COE Bot Developers rather than try to create complex automation code that they may not be trained for.

# Avoid ‘hard coding’

 Both COE and Business Bot Developers should avoid hard coding any external settings such as file paths, URLs, variables, values, etc., within Bot scripts. Hard coding such settings means that the only way to change such settings is to amend them from within the script itself which is both time consuming and expensive.

Hard coding should be avoided because these settings are prone to changes over time. For example, the file location where you download the daily report to might change due to business requirements. The most efficient way to store these parameters is to use a configuration file and to find ways to parametrize the Bot script.

The idea is to minimize unnecessary changes to the script, especially if it has already been deployed to production, to avoid introducing additional bugs or errors. Also, if these settings are used multiple times within the script, if you hard code, you need to know exactly where the changes are required.

Both COE and Business Bot Developers should pay attention to challenges created by hard coding during Code Reviews. The goal should be to identify ways in which the code can be made more generic and find opportunities for removing hard coded entries.

# Keep It Simple

 In Bot development as in programming, simple is beautiful although it is not always easy to achieve.

Where possible, try to reduce or eliminate unnecessary complexities in your script. This will help reduce the number of bugs or programming errors, which in turn means that you need to spend less time trying to debug or resolve them.

# Readability

 Ignoring the readability of your RPA script is a common mistake made by many new RPA developers.

Readability in this case means how easy is it for someone else (other than the developer) to take the script and understand what the software robot is being programmed to do. In fact, for complex processes involving hundreds or thousands of steps, the developer himself often at risk of losing track of what is going on if he does not take active steps to ensure readability. Ensuring good readability will also drastically reduced the amount of time you spend during troubleshooting and debugging, as well as during bots maintenance.

To do so is fairly straightforward. Mostly, one needs to make a conscientious effort to provide meaningful names for workflow files, activities, arguments and variables. Moreover, as a best practice, all variables and arguments should be aligned to a naming convention:

* Variables should be in camelCase, e.g. firstName, lastName
* Arguments should be in TitleCase, e.g. FileName, DefaultTimeOut

Also, all your workflow names should contain the verb describing what the workflow does, e.g. GetTransactionData, LoginToSAP.

One should make judicious use of comments and annotations to describe sections of your code which are not as intuitive. It is important to be selective though as you do not want to end up clustering your program with redundant comments.

# Naming conventions

Use of proper and meaningful naming conventions is considered good practice. Sometimes programmers tend to use X1, Y1, etc. as variables and forget to replace them with meaningful ones, causing confusion.

In order to prevent this waste of time, it is usually considered good practice to use descriptive names in the code since Bots are about real business processes.

Example: A variable for taking in weight as a parameter for a truck can be named “TrkWeight” or “TruckWeightKilograms”, with “TruckWeightKilograms” being the more preferable one, since it is instantly recognizable.

# Code review

Code reviews are required to ensure that appropriate Bot development standards are applied and secure development practices as per <Company> IT compliance requirements are followed. Code reviews are also very helpful when it comes to refactoring your code. Code review is a phase in the Bot development process in which the authors of code, peer reviewers, and perhaps quality assurance (QA) testers get together to review Bot script. It is important that COE Bot Developers review Bots developed by Business Bot Developers.

Others might see better implementation to optimize your code or just make your code more elegant. It also ensures that developers adhere to standards and the work is double checked. In addition to all that, it is a wonderful way for developers to learn from each other.

The role of independent code reviewers should go beyond the compliance aspects such as detection of errors and malicious code. Code reviewers need to ensure that all Bot development is reusable to the maximum extent. Code review is the key to ensure that you have highly maintainable, performant and extensible bots.

# Testing

A common mistake made by new developers is not testing their code enough or at all. This is especially true in the case of RPA where developers normally start off working on the happy path.

Unfortunately, in reality, enterprise use-cases are usually complicated. There could be many variants of a single process, or the input data might not come in the format that you expect. Hence it is important to put in place a robust test plan with comprehensive test cases, so that we can ensure the robot is performing exactly the way we want it in all possible scenarios.

A failed test should not lead to finger pointing. Rather, it should be embraced in the sense that the error is discovered before the bots go into production (where far greater damage can be caused).

Also, Bot Developers should strive to test as frequently as possible. Do not wait until the entire development is completed before you start testing. If you have implemented modularity in your design, you should be able to independently test the various components and see if they are working as intended.

For best results, RPA development should not be an individual pursuit. To ensure highest quality Bots, the developer, the peer reviewer and the tester should all be different persons.

# Commenting

As in all software development projects, due to time restrictions or enthusiastic Bot Developers who want immediate results for their code, commenting of code often takes a back seat. Commenting is a critical piece of good software development practice and the development team should collectively ensure that no Bots are deployed to production without adequate comments.

A simple way to get new or Business Bot Developers to start using comments is to use comments a form of pseudocode. This means outlining intention of the code that follows prior to writing the actual code. Such a comment should explain the logic behind the code in plain English rather than the code itself.

Following are few common elements a Bot developer should incorporate in comments:

* Description of the Bot
* Bot Author & Date
* Bot Modified & Date;
* Reason for Modification
* Parameters
* What is the Bot or code going to do
* How is the Bot or code going to perform the steps that follow
* When call is made to another Bot, rationale for doing so with Bot ID, Version etc.

Most automations require changes after being placed into production. Changes could be frequent, depending on the type and scope of the automation. The difference between a change being a relatively straightforward task and a complete nightmare is determined by two things: how cleanly the automation was architected, and how well it is documented and commented. Good commenting can mean a major difference in time taken during maintenance and change cycles.

# Decoupling and Loose Coupling

When possible, Bots should not have a dependency on the calling Bot. In automation this is often unavoidable. However, developing with this in mind can improve the overall architecture and maintainability of automation.

* Using a login Bot as an example, if the login bot can only be called by one single master Bot, then it is tightly coupled to that master Bot. If the login bot is designed in such a way that the URL of the page it uses has to be set by the calling Bot, then it cannot run by itself. It cannot be unit-tested alone, and other bots cannot call it without knowing the URL of the login page before calling it. If the calling bot must provide the login page URL to the login bot, then all bots that use the login bot are more tightly coupled to that login bot. And if the URL changes, more than one bot must be changed.
* However, if the login bot contains all of the information it needs to login to the web application, including the URL, then it is a truly stand-alone bot. It can be unit-tested, and it can be called by any other bot without the need to be provided the URL. It is then “decoupled” from other bot and is much more modular and maintainable.

# Error Handling

One of the keys to a successful automation is predicting handling expected and unexpected events. Examples are, a save file dialog not appearing within a specific time frame or a file not found message that holds up further execution of a bot.

Never assume conditions in production environments will always be as you expect them to be. If your automation works with a web browser, do not expect that the site will always be up, or that the internet will always be available. Therefore it is critical to make use of the Error handling or Exception handling capabilities of your Intelligent Automation tools. Use exception handling to not only log errors, but to also provide means to recover from errors and exceptions based on your business logic.

For example, consider an automation that downloads a file from a web site. After clicking on the download link, the automation waits 15 seconds for the download prompt to appear at the bottom of IE. In this case the bot is designed to use a ‘wait for window to exist’ attribute to determine when the Save dialog appears.

In addition to functionality exceptions, the bot may also have to deal with system exceptions. The Code Review process and Testing should be used to identify such system specific error possibilities and systematically enable the bot to handle such exceptions.

# Security

Secure coding is the practice of developing computer [software](https://en.wikipedia.org/wiki/Software) in a way that guards against the accidental introduction of security vulnerabilities.

Identify and document security requirements early in the Bot development life cycle and make sure that subsequent development artifacts are evaluated for compliance with those requirements.

Develop and/or apply a secure coding standard for your target development language and platform for e.g. masking password during inputs, pass sensitive data as encryption, refer credential vault if it is exist. As Intelligent Automation requires access to multiple systems and applications, it is vital to pay attention to secure coding practices during planning. Under no circumstances should sensitive access credentials and sensitive user information or data be stored within the bots either in hard coded or other format. All information should be encrypted and the platforms’ secure development capabilities leveraged to eliminate any inadvertent unsecure practices.

# Tips

Following are few thoughts to be considered when beginning Bot development:

* UI Automation Technology: Most RPA technology is based on UI automation using detection and handling of UI objects and their properties. This should be the preferred mode of automation at all times.
* Bot Developers should endeavour to learn the base technology and how their chosen automation platform ensures automation. This will help them in developing reliable and robust Bots keeping in view the strengths and weaknesses of the Intelligent Automation tool.
* In situations where access to object properties is not possible or unreliable, other techniques such as image recognition, coordinates or proprietary technology that leverage such fundamental techniques are used for automation.
* Image recognition should be used only as a last resort when all else fails. This is because image matching only works when the image is fully visible on screen, i.e. it cannot be in the background. Also, image recognition is sensitive to the screen resolution, which can be sometimes tricky to handle.
* Version Control: Use of versioning should be considered standard practice for all bot development and intermediate Bot artifacts. Following such practices, in addition to ensuring you have previous versions for rollback, also ensures that collaboration among scrum team is most efficient, complete and timely.
* Attention should be paid to all standard software development practices couple of which are listed below:
  + Memory leaks: In software development Memory leaks occur when new memory is allocated dynamically and never deallocated. Typical situations where this happens is in loops which are used most frequently by Bots to run large volumes of repetitive transactions. The problem with memory leaks is that they accumulate over time and, if left unchecked, may cripple or even crash a program. COE Bot Developers and Testing team must pay mandatory attention to how memory allocation and deallocation is managed by each Bot/script. This should also be monitored as part of platform operational metrics.
  + Global & Local Variables: Software developers are very familiar with the appropriate use of global and local variables. However, for Business Bot Developers these may be new concepts. Global variables are declared at the start of the program, their global scope means they can be used in any part of the Bot. It is seldom advisable to use Global variables as they are liable to cause bugs, waste memory and can be hard to follow when tracing code. If you declare a global variable it will continue to use memory whilst a program is running even if you no longer need/use it. Local variables are usually preferred as they are declared within subroutines or programming blocks, their local scope means they can only be used within the subroutine or program block they were declared in. Once the function ends, the memory taken up by the variable is released. This contrasts with global variables which do not release memory.

* If your RPA tool provides a Recording Option, do not be overly reliant on the recorder. The recorder can be a very useful tool for you to understand how to construct your RPA script for a certain process, but rarely is the generated script resilient enough to be used reliably in a production environment.
* Bots no matter how carefully designed and developed, are unlikely to work all the time. Hence, it is extremely important for bots to be able to handle errors and exceptions. Logging should be done so that further investigation, if warranted, can be carried out.
* Most RPA tools allow you to extract text from desktop or web applications, either natively or using an OCR engine. If you are using the later, do be mindful that OCR is not hundred per cent accurate. In cases where accuracy is important, you might not want to rely solely on OCR.
* Avoid Too Many Sub-tasks: While modularization is an excellent way to design a Bot, it is possible however to “overdo it”. Unusually high number of sub-tasks, say more than 10, with thousands of lines of code usually indicates a business process that is too large for a single Bot. Such a process should be broken down into pieces, and each of those separate pieces encapsulated in their own automation.
* Avoid Bi-Directional Dependencies: Bots that are reused within other Bots should be as standalone and reusable as possible. If a calling/master Bot cannot be changed without all sub Bots being changed, they are not truly decoupled, and have bi-directional dependencies. This not only creates a maintenance nightmare, but it makes unit-testing nearly impossible
* The Single Responsibility Principle: In object-oriented programming, the single responsibility principle states that every class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class. All its services should be narrowly aligned with that responsibility. This principle directly applies to automation development as well. Now imagine a single automation task that is 2000 lines long. This task is far too long and should be split into several sub-tasks. The automation developer decides to put several of the repeated tasks into another sub-task. Consider for example he or she picks three repeated sections and puts them in a sub-task. This new helper task now handles printing a PDF, but also handles saving a file to a specific folder, and in addition to that it also handles moving a file from one folder to another. The developer manages this which part is called by passing an action variable. The above example would break the Single Responsibility Principle. The developer has reduced the number of lines in the master task, which is good, but has now a sub-task that is far too big. Additionally, if any one of those responsibilities (printing a PDF, moving a file or saving a file) need to be modified, the entire helper task must be modified. This creates the possibility for introducing a bug in task that would not have otherwise been effected. The proper approach would be to create three sub-tasks, each having their own responsibility – One for print to a PDF, another for moving the files, and another for saving files.
* When automating an application for the time, sufficient feasibility analysis and POC/Testing if required should be done on new application to determine best approaches.
* Web automation: Automating applications, especially browser-based applications, can be a moving target at times. Web pages are known for frequent changes, which in turn will often lead to exceptions in deployed automations. This can be managed to some extent by creating automated testing bots to verify for breakage and pro-actively work on updating routines. Attention should be paid to all applications that are similarly prone to frequent changes and consequent Bot breakage.
* Advanced Automation Technologies: RPA is increasingly combined with specialized and advanced automation technologies such as OCR (Optical Character Recognition), AI & ML (Artificial Intelligence & Machine Learning), ChatBots, etc. It is vital that the Bot Developers understand these techniques, product used with capabilities and limitations to ensure that any Bots that leverage multiple such technologies are well thought through. Modularity is a great way to leverage the capabilities provided by such tools.

# Don’t Underestimate Complexity

While Bots and intelligent automation platforms are designed for intuitive development and non-intrusive and deployment, they are not necessarily any less complex than most enterprise technologies. Bot developers should be careful not to underestimate the task at hand when developing Bots for today’s complex and global enterprises with high performance expectations. Attention is required to the complexities created by areas such complex enterprise business processes with multiple exception scenarios, capabilities and limitations of automation platforms and techniques, skill level of Bot developers, enterprise application deployment architecture, management and compliance standards, etc. Ensuring that Bot development is smooth, scalable and fulfils its business promise requires adequate attention to all such details during planning and Bot development as with other enterprise technologies.