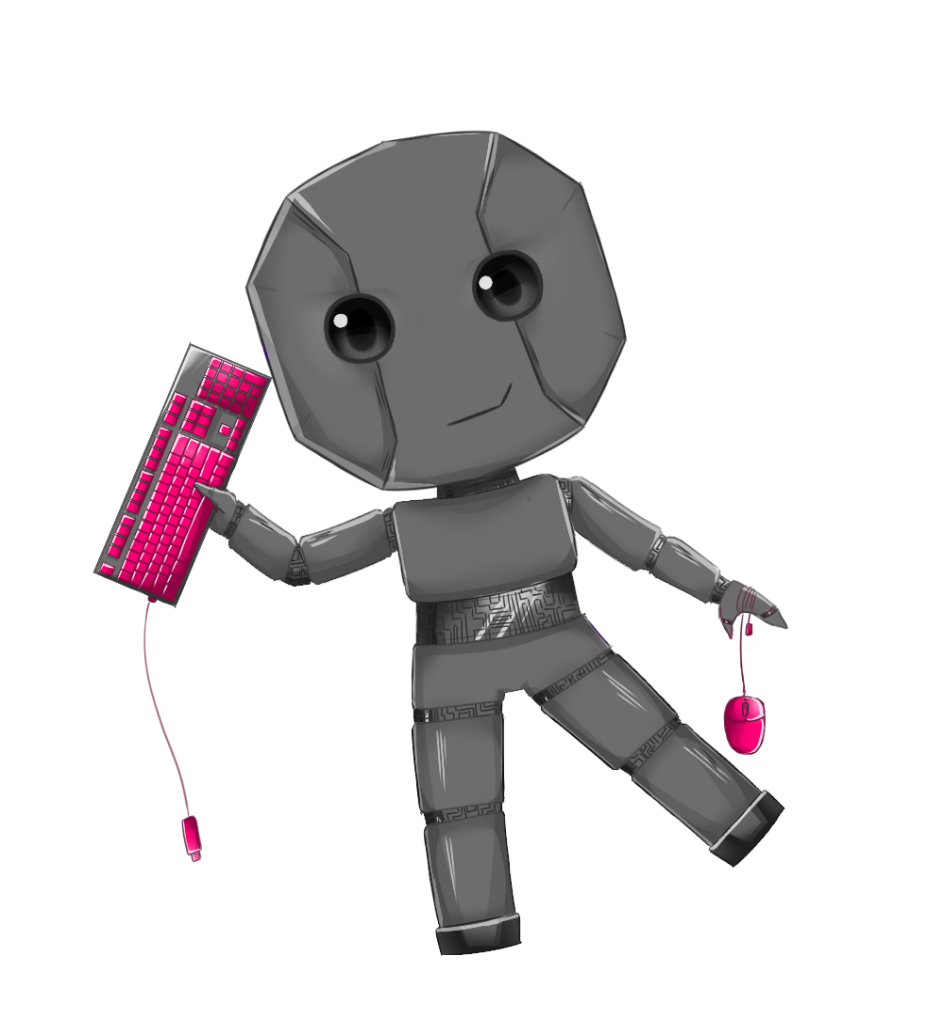
**Załącznik nr 14 do Umowy**

Wytyczne realizacyjne



**Podstawowe informacje o dokumencie:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Produkt** | Wytyczne realizacyjne | | |
| **Wersja** | 3.5 | **Klasa** | Dokument wewnętrzny zespołu RPA |
| **Data utworzenia** | 29.11.2018 | **Ostatnia modyfikacja** | 20.06.2025 |

**Dokument został wypracowany przez Zespół Projektowy:**

|  |  |
| --- | --- |
| **Opracował:** | **Jakub Gondek, Kamil Kuna, Kamil Bochenek** |
| **Zatwierdził:** | **Kamil Rekus** |

**Historia zmian:**

|  |  |  |  |
| --- | --- | --- | --- |
| Wersja | Data | Kto | Opis zmiany |
| 0.1 | 29.11.2018 | Jakub Gondek | Utworzenie dokumentu |
| 1.0 | 04.12.2018 | Kamil Rekus | Zatwierdzenie |
| 1.1 | 21.12.2018 | Jakub Gondek | Rozwój dokumentu |
| 1.2 | 17.07.2019 | Kamil Kuna | Rozwój dokumentu |
| 1.3 | 29.07.2019 | Kamil Kuna | Rozwój dokumentu |
| 1.4 | 12.08.2019 | Kamil Kuna | Rozwój dokumentu |
| 1.5 | 29.08.2019 | Jakub Gondek | Rozwój dokumentu |
| 1.6 | 26.09.2019 | Jakub Gondek | Rozwój dokumentu |
| 1.7 | 30.09.2019 | Jakub Gondek | Rozwój dokumentu |
| 1.8 | 14.11.2019 | Kamil Kuna | Rozwój dokumentu |
| 1.9 | 15.11.2019 | Jakub Gondek | Rozwój dokumentu |
| 2.0 | 17.01.2020 | Kamil Kuna | Dodanie sekcji „1.7. Standardy nazewnictwa kolejek” |
| 2.1 | 27.03.2020 | Jakub Gondek | Dodanie wpisu o przeglądarkach w trybie prywatnym. Aktualizacja zapisów o assetach i nazewnictwie |
| 2.2 | 16.04.2020 | Jakub Gondek | Dodanie zapisu o Auto-retry, doszczegółowienie zapisu o słownikach |
| 2.3 | 09.11.2020 | Jakub Gondek | Dodanie zapisu o delayach. Doszczegółowienie procedury przekazywania haseł produkcyjnych. |
| 2.4 | 23.12.2020 | Jakub Gondek | Zmiana brzmienia punktu 2.3.  Dodano:  - informacje o wymaganych argumentach w raportowaniu  - informacje o wymaganiu dotyczącym prawidłowego funkcjonowania BOTa  - punkt 2.1 „Zasady ogólne” |
| 3.0 | 30.06.2022 | Kamil Bochenek | Aktualizacja treści, rozwój oraz edycja struktury dokumentu, podział na kategorie do rozwinięcia w przyszłości |
| 3.1 | 03.10.2022 | Kamil Bochenek | Korekta edytorska błędów, zaakceptowanie śledzonych zmian wersji. |
| 3.2 | 13.10.2023 | Kamil Kuna | Dodanie punktu 5 Obsługa Orchestratora, rozszerzenie punktu 1.1.5 oraz 1.1.6. |
| 3.3 | 22.01.2024 | Kamil Bochenek | Aktualizacja i uproszczenie treści zgodnie z komentarzami zespołu, rozwój oraz edycja. Głownie uwzględniono ostatnie wypracowane praktyki i realnie stosowane wytyczne w zakresie architektury i developmentu. |
| 3.4 | 15.11.2024 | Kamil Bochenek | 1. Zaktualizowano listę frameworków: podano SOD\_Framework\_DoKolejki 2. Poprawione literówki |
| 3.5 | 20.06.2025 | Kamil Bochenek | Zaktualizowano każdy z rozdziałów zgodnie z obecnie funkcjonującymi wytycznymi oraz uwagami Zespołu. Uzupełniono brakujące dotąd kwestie (1.3 Biblioteki, 4.4 RPA TV). Uproszczono treść i dokonano przeglądu wytycznych jakościowych. |

Spis treści

[1 Standardy przyjęte do stosowania w trakcie developmentu 6](#_Toc200963724)

[1.1 Ogólne standardy nazewnictwa 6](#_Toc200963725)

[1.1.1 Standardy nazewnictwa artefaktów 6](#_Toc200963726)

[1.1.2 Standardy nazewnictwa zmiennych 6](#_Toc200963727)

[1.1.3 Standardy nazewnictwa argumentów 6](#_Toc200963728)

[1.1.4 Standardy nazewnictwa czynności (ang. activities) 7](#_Toc200963729)

[1.1.5 Standardy nazewnictwa assetów 7](#_Toc200963730)

[1.1.6 Standardy nazewnictwa kolejek 8](#_Toc200963731)

[1.2 Standardy dotyczące struktury projektu (workflow) 9](#_Toc200963732)

[1.3 Biblioteki 10](#_Toc200963733)

[1.3.1 Common i biblioteki RPA 10](#_Toc200963734)

[1.3.2 Dependencies - biblioteki Uipath 10](#_Toc200963735)

[1.3.3 Wyjątek – system CSS. 10](#_Toc200963736)

[2 System kontroli wersji oraz repozytoria 12](#_Toc200963737)

[2.1 Standardy wersjonowania kodu 12](#_Toc200963738)

[2.2 Praca z repozytorium GIT 12](#_Toc200963739)

[2.2.1 Format commita 13](#_Toc200963740)

[2.3 Code Review 13](#_Toc200963741)

[3 Wytyczne jakościowe 14](#_Toc200963742)

[3.1 Ogólne wytyczne jakościowe do stosowania podczas tworzenia kodu 14](#_Toc200963743)

[3.1.1 Artefakt Do\_Kolejki 14](#_Toc200963744)

[3.1.2 Artefakt Przetwarzanie 15](#_Toc200963745)

[3.1.3 Artefakt Raport 15](#_Toc200963746)

[3.1.4 Config 16](#_Toc200963747)

[3.1.5 Przechowywanie plików wykorzystywanych w procesie 16](#_Toc200963748)

[3.1.6 ExceptionScreenshots 16](#_Toc200963749)

[3.1.7 Wysyłka maili 17](#_Toc200963750)

[3.1.8 Praca na aplikacjach web 17](#_Toc200963751)

[3.1.9 Czyszczenie plików tymczasowych 17](#_Toc200963752)

[4 Procedury 18](#_Toc200963753)

[4.1 Zasady ogólne 18](#_Toc200963754)

[4.2 Udostępnienie danych logowania z uprawnieniami produkcyjnymi 18](#_Toc200963755)

[4.3 Przejście DEV -> PRD 18](#_Toc200963756)

[4.4 RPA TV 18](#_Toc200963757)

[5 Obsługa Orchestratora 20](#_Toc200963758)

# Standardy przyjęte do stosowania w trakcie developmentu

W Sekcji RPA przyjęto szereg standardów oraz wytycznych mających na celu usprawnienie procesu tworzenia BOT-ów oraz późniejszego ich utrzymania.

## Ogólne standardy nazewnictwa

### Standardy nazewnictwa artefaktów

Dla ułatwienia komunikacji przyjęto, że artefakty będą otrzymywać aliasy w formie imion, aby ograniczyć potrzebę używania pełnej nazwy procesu wykonywanego przez artefakt, co pozwoli na zachowanie spójności nazewnictwa oraz ułatwi katalogowanie wszelakich danych.

* Do nazwy artefaktu dodajemy „\_DoKolejki”, „\_Raport” zależnie od funkcji danego artefaktu
* Nazwa artefaktu przetwarzania ma formę NazwaBOTa\_x wskazującą na wykonywane w procesie czynności (np. Edward\_PrzetwarzanieSOD, Edward\_Tworzenie, Edward\_Weryfikacja)

### Standardy nazewnictwa zmiennych

W celu poprawienia czytelności tworzonego kodu przyjęto standardy dotyczące nazewnictwa zmiennych, których należy przestrzegać.

* Zalecane nazewnictwo zmiennych w notacji „PascalCase” jako podstawowe, ale w razie potrzeby można używać „camelCase” lub „snake\_case”.
* Zalecane używanie Intellisense (ctrl+spacja) w celu zachowania porządku w zmiennych (kompilator przyjmuje case-insensitive).
* Dopuszcza się zmienne w języku polskim i angielskim.
* Wykluczenie spójnika „i”, z nazw zmiennych (w większości przypadków pogarsza czytelność i przyczynia się do błędów w korzystaniu ze zmiennej).
* Prefiks „dt” przed nazwami DataTable dla wyróżnienia tych zmiennych (np. dtTabelaDanych).
* **Niezrozumiałe skróty w nazwach zmiennych zabronione** – kładziemy nacisk na możliwość skojarzenia zawartości zmiennej przez inną osobę.
* W przypadku stosowania zmiennych typów innych niż typy proste (np. int, float, double, string) takich jak listy, kolekcje, typy czasowe, timery zalecane jest dodawanie wyróżnika (np. ListaUmow, DataPrzyjecia, KolekcjaJR, TimerStart) co ułatwi analizę kodu i utrzymanie.
* W celu ograniczenia liczby argumentów, zwłaszcza w przypadku zagnieżdżonych workflows zalecane jest korzystanie ze **zmiennych globalnych**
* Dobrą praktyką jest wyróżnienie zmiennej globalnej od zmiennych lokalnych za pomocą prefiksu np. „global”, „g”.

### Standardy nazewnictwa argumentów

Standardy nazewnictwa dla argumentów są analogiczne do standardów nazywania zmiennych co do rdzenia z dodatkowym uwzględnieniem poniższych reguł:

* Nazewnictwo tworzone analogicznie jak w przypadku zmiennych dodatkowo poprzedzone prefiksem (np. in\_NazwaArgumentu).
* Dopuszcza się używanie maksymalnie 20 argumentów w jednym workflow oraz powinno się **dążyć do przekazywania wartości za pośrednictwem kolekcji.**
* Wykluczenie spójnika „i” z nazw argumentów.

Przyjęto trzy prefiksy wykorzystywane dla argumentów:

* in\_NazwaArgumentu – dla argumentów o kierunku „In”
* out\_NazwaArgumentu – dla argumentów o kierunku „Out”
* io\_NazwaArgumentu – dla argumentów o kierunku „In/Out”

### Standardy nazewnictwa czynności (ang. activities)

Nazwy powinny być unikalne w zakresie pozwalającym na jednoznaczne zidentyfikowanie miejsca w całym kodzie (m. in. za pomocą wyszukiwarki UiPath Studio, na podstawie logów).

* Nazwa activity powinna zawierać: domyślnie generowaną nazwę wskazującą na typ i funkcję activity (np. Assign, Click itd.) oraz odniesienie do kroku z instrukcji jeśli występuje bezpośrednie nawiązanie (np. „1.25 Click Umowy”).
* Do numeracji należy podchodzić w sposób zbalansowany aby nie robić tego zbyt skrupulatnie jednocześnie umożliwiając szybką orientację na linii kod- instrukcja.
* W przypadku gdy **activity wykonuje akcję przeciwną do wskazanej w nazwie** należy to uwzględnić w opisie np. „Check ‘Checkbox’ xxx – UNCHECK” lub Uncheck ‘Checkbox’ xxx.
* Niektóre nazwy activities nie nawiązują bezpośrednio do instrukcji lub nie są generowane automatycznie, opis jest dowolny (ObslugaCMD, Inicjalizacja zmiennych itd.)
* **Opis funkcjonalny** – dołączenie nazwy czynności, nazwy elementu GUI/zmiennej, lub w razie potrzeby inny np. „1.27 Click Zapisz”, „2.60bType Into DataObowiazywaniaCennika”.
* W nazewnictwie również należy **zachować balans** między dobrym opisem umożliwiającym identyfikację kroku z instrukcją a czasem spędzonym na opisywanie pojedynczych activities, powinny być unikalne ale bez nadmiernej szczegółowości.

Można używać form skróconych np. T/C jednak należy mieć na uwadze łatwość zlokalizowania danego activity za pomocą wyszukiwarki w UiPath Studio oraz możliwość szybkiej diagnozy na podstawie logów.

### Standardy nazewnictwa assetów

Standardy nazewnictwa dla poszczególnych kategorii assetów znajdujących się w Orchestratorze.

* Assety mogą być globalne lub przypisane do konkretnego procesu.
* Nazwy assetów powinny być bezpośrednio wskazane w kodzie, aby Orchestrator był w stanie je zidentyfikować podczas tworzenia procesu.

Kategorie assetów rozpoczynają się od konkretnego prefiksu:

* **PATH\_NazwaAssetu** - assety, które mogą być używane przez różne artefakty zawierające takie dane jak m.in. adresy URL, ścieżki do systemów, ścieżki do plików itd., np.:
  + CRED\_NazwaAssetu - assety typu „credential” zawierające dane logowania.
* **PATH\_NazwaBOTa, CRED\_NazwaBOTa** - w przypadku, gdy dana ścieżka jest linkiem do konkretnego elementu systemu np.:
  + PATH\_EWA lub BOT wymaga specjalnych uprawnień
  + np. CRED\_E-SWITCHING\_BAAL (uprawnienia administratora)
* **NazwaBota\_NazwaZmiennej** - assety pełniące funkcje zmiennych, które wykorzystywane są tylko przez konkretny proces. Ze względów porządkowych tego typu assety powinny być tworzone tylko w ostateczności np. kiedy wartość powinna zostać zachowana po zakończeniu procesu, przekazywana do innych instancji Bota lub kiedy pełni konkretną funkcję np. sterującą.
* W nazwach assetów zabronione są polskie znaki diakrytyczne.

### Standardy nazewnictwa kolejek

Nazewnictwo kolejek ma bezpośredni wpływ na prawidłowość **obliczeń oszczędności** BOT-ów. W związku z tym przyjęto następujące standardowe rozwiązania.

* **Rdzeniem** nazwy jest alias BOT-a.
* **Jedna kolejka** – jednowyrazowa nazwa, np. Anastazja.
* **Dwie kolejki** – jednym z kryteriów jest sposób rozliczania transakcji
  1. Liczone według **tego samego czasu transakcji** – jednowyrazowa nazwa\_wyróżnik, np.: Tymon\_UKTS, Tymon\_UKGZE (Tymon jest wyjątkiem, raczej stosujemy jedną kolejkę).
  2. Liczone według **różnego czasu transakcji** lub liczone dla innego beneficjenta lub ze względu na unbundling – jednowyrazowa nazwa, jednowyrazowa nazwa z wyróżnikiem, np. Fiona, FionaTD.
  3. **Liczona tylko jedna kolejka** (pomocnicza)- @jednowyrazowa nazwa, jednowyrazowa nazwa np. @Amelia, Amelia (w tym przypadku kolejka ze znakiem @ nie jest brana pod uwagę przy oszczędnościach).

**Poza standardowymi rozwiązaniami** istnieją również wyjątki np. kiedy liczba kolejek jest większa niż dwie, lub kolejki są wykorzystywane w inny sposób. Wówczas zasady nazewnictwa są ustalane indywidualnie ale z uwzględnieniem powyższych zasad.

**NAZWY KOLEJEK W ACTIVITY**

**Jeśli funkcjonalność danego activity na to pozwala nazwy kolejek powinny być bezpośrednio wskazane w kodzie, aby Orchestrator był w stanie je zidentyfikować podczas tworzenia procesu (unikamy zawierania nazw kolejek w pliku Config.xlsx BOT-a oraz przekazywania za pomocą zmiennej, wybieramy z listy).**

## 1.2 Standardy dotyczące struktury projektu (workflow)

Przyjęte zostały standardy dotyczące tworzenia workflow BOTa:

* **Nazewnictwo plików .xaml** – każdy z plików niebędący plikiem głównym (Main.xaml) powinien opisywać czynności wykonywane wewnątrz. Niedozwolone są znaki diakrytyczne.
* **Modułowość** - kod powinien mieć modułową budowę:
  + Stopień podziału na moduły powinien rosnąć wraz ze skalą oraz poziomem skomplikowania procesu**.**
  + Struktura modułowa powinna:
    - odzwierciedlać logikę procesu wraz etapami zawartymi w instrukcji
    - ułatwiać analizę kodu
    - zapewniać łatwość wprowadzania modyfikacji, sprawne prowadzenie testów
    - faworyzować stosowanie uniwersalnych fragmentów kodu.
* **Granulacja** - kod powinien być dzielony na mniejsze workflows by zapewnić wydajność środowiska na stałym i wysokim poziomie mając na uwadze ograniczenia technologiczne.
* **Poszczególny workflow poza nazwami activities powinien być dodatkowo opisany (np. w adnotacji)** jeśli istnieje przesłanka, że zidentyfikowanie jego głównego zadania i pełnionych funkcji będzie utrudniało wykonanie pracy innym osobom (**modyfikacja, refaktoryzacja, diagnoza, utrzymanie**).
* **Uniwersalne oraz powtarzalne fragmenty kodu** zawarte w odrębnych plikach .xaml powinny być zaprojektowane w sposób przewidujący ich ponowne wykorzystanie w innych rozwiązaniach lub pozwalający na zaadaptowanie rozwiązania na potrzeby biblioteki **Common** czy też umieszczenie w Repozytorium jako szablonu.
* **Bazę do rozpoczęcia prac deweloperskich** stanowią szablony („frameworki”) dostępne do pobrania z Repozytorium Git:
  + Artefakt „\_DoKolejki”:

- „Szablony/Framework\_DoKolejki”,

- „Szablony/SOD\_Framework\_DoKolejki”

* + Artefakt „\_Przetwarzanie”: „Szablony/ReFramework”
  + Artefakt „\_Raport”: „Szablony/Framework\_Raport”
  + Rozwiązania dla SOD 5.5: SOD\_API/FrameworkAPI\_SOD55
* **Przed rozpoczęciem prac musi zostać pobrana aktualna wersja danego frameworku** oraz dostosowana do potrzeb procesu:
  + gotowy szablon odnosi się do większości procesów o standardowej architekturze
  + dla szczególnych rozwiązań dopuszcza się zastosowanie odrębnej architektury co wymaga konsultacji z Architektem Rozwiazania RPA)
* Każdy artefakt przed publikacją do Orchestratora musi posiadać odpowiednią nazwę pliku oraz opis (dane te są zawarte w pliku .json):
  + project.json, pole „description” powinno zawierać nazwy wykorzystywanych systemów wypisane po średniku oraz **„Filenet” jeśli w procesie w sposób niebezpośredni bierze udział SOD** (np. „SOPP; SAP; Filenet” – tutaj SOD bierze udział w dodawaniu załączników przez SAP)
* W przypadku korzystania z wielu systemów **struktura plików oraz nazewnictwo powinno uwzględniać podział procesu** na części wykonywane w poszczególnych systemach (np. dodanie nazwy systemu w nadrzędnych .xaml, umieszczanie plików w odpowiednio opisanych katalogach)
* Katalog główny danego artefaktu powinien zawierać plik „.gitignore.txt” wykluczający pliki: „/.screenshots/”.
* Zmienne globalne – zaleca się świadome i uzasadnione stosowanie przy zachowaniu hermetyzacji (aby zmienne były dostępne tylko tam gdzie bezpośredni dostep jest wymagany).

*\*Problemy z używaniem Global Variables zostały rozwiązane w Uipath Studio 23.10.*

* Możliwe jest stosowanie outputów dla transakcji innych niż „Sukces”.

## 1.3 Biblioteki

### 1.3.1 Common i biblioteki RPA

W miarę możliwości zalecana jest aktualizacja wersji biblioteki Common:

* Dla modyfikacji przed testami aby mieć pewność, że skutek nie będzie negatywny
* Nowe Boty powinny powstawać z użyciem najnowszego Commona chyba, że sytuacja wymaga innego podejścia.

### 1.3.2 Dependencies - biblioteki Uipath

Powinno się dążyć do korzystania z najnowszych dependencies, co najmniej tych z aktualnego Commona ale decyzja powinna być oparta na zdrowym rozsądku gdyż nie zawsze oznacza to stabilność.

Tutaj decydują głównie względy stabilności pracy farmy Botów po ewentualnym wyjściu z użycia starszych bibliotek lub utraty kompatybilności.

### 1.3.3 Wyjątek – system CSS.

Jednym ze znanych wyjątków jest **system CSS**. Zastosowanie zbyt wysokiej wersji Uipath.Automation uniemożliwia tworzenie selektorów w aplikacji natomiast możliwe jest uruchomienie kodu i jego prawidłowe działanie. Możliwe rozwiązania:

* Zastosowanie Common\_CSS 1.0.2 i podnoszenie wszystkiego oprócz Uipath.Automation.
* Rozdzielenie artefaktów tak, aby wyłącznie CSS działał na niższych dependencies.
* Utworzenie selektorów w osobnych projekcie z Common\_CSS 1.0.2 i przeniesienie activities do właściwego projektu. Jednak tracimy możliwość modyfikacji bezpośrednio w projekcie.

# System kontroli wersji oraz repozytoria

## Standardy wersjonowania kodu

W Sekcji RPA zostały przyjęte standardy wersjonowania kodu mające na celu ujednolicenie sposobu przekazywania postępów prac nad kodem oraz zabezpieczenia wersji produkcyjnych.

Do wersjonowania przyjęto konwencję, w której skład wchodzą **Major.Minor.Patch (1.0.0)**:

* **Numer główny (Major)** – oznaczenie wersji kodu, która opiera się na tych samych założeniach opisanych w instrukcji i tej samej architekturze.
  + Zmieniana sporadycznie, w wyniku refaktoryzacji, zmiany koncepcji lub nowego podejścia do robotyzacji procesu.
  + Zakłada się, że **wersje przed 1.0.0 to wersje developerskie/testowe** niedopuszczone do pracy produkcyjnej.
* **Numer dodatkowy (Minor)** – oznaczenie kolejnych etapów rozwoju w ramach tej samej wersji major.
  + Zmiana numeru dodatkowego jest efektem zmiany istniejącej funkcji, dodaniem nowej funkcjonalności, modyfikacji.
* **Numer wydania (Patch)** – oznaczenie kolejnych wydań wersji minor.
  + Wersje patch zawierają w sobie głównie łaty istniejących funkcji oraz poprawki wydajnościowe.

## Praca z repozytorium GIT

Kody źródłowe zrobotyzowanych procesów znajdują się w bezpiecznym repozytorium systemu kontroli wersji.

* Aktualny kod źródłowy powinien znajdować się w **zdalnym repozytorium** GIT.
* Commit’y do zdalnego repozytorium danego artefaktu powinny być wykonywane **przynajmniej raz dziennie** w celu zabezpieczenia postępów pracy.
* Modyfikacje oraz inne zadania zgłaszane w issue wykonujemy na nowym branchu utworzonym z **aktualnego mastera.**
* Po zakończeniu zadania kod może być zmergowany do mastera **tylko poprzez Pull Request** po przeprowadzeniu Code Review przez osobę do tego wyznaczoną.

### Format commita

Prawidłowo napisany commit powinien przyjmować określony porządek:

* główny opis,
* opcjonalnie opis poszczególnych zmian jeśli występują
* oraz odwołanie do issue w nawiasie (#numer\_issue) jeśli takowe zostało utworzone:

Ogólny opis zmiany (#numer\_issue).

[+] Dodane elementy + informacja o miejscu wystąpienia zmiany

[-] Usunięte elementy + informacja o wystąpieniu zmiany

[=] Aktualizacja + informacja o wystąpieniu zmiany

[^] Podniesienie wersji projektu, paczek itd. szeroko rozumiany upgrade

## Code Review

Kod po akceptacji testów przekazywany jest do Code Review a następnie na produkcję:

* **Każdy artefakt musi być w pełni funkcjonalny oraz odpowiednio skonfigurowany**.
* Przyjęto, że CR jest wykonywany po testach akceptacyjnych (z uwagi na to, że w praktyce zachodzą jeszcze zmiany wynikające ze sposobu oraz jakości przygotowywanych danych, a incydentalnie Opiekun BOT-a może wymagać zmiany pewnych funkcjonalności). **W razie rażących naruszeń funkcjonalnych testy powinny zostać powtórzone.**
* Za kod i efekty działania BOT-a po wprowadzeniu zmian odpowiedzialny jest Developer.
* Code Review nastawiony jest na weryfikację pod kątem architektury i przyjętych wytycznych realizacyjnych więc zakres odpowiedzialności nie obejmuje wychwytywania błędów i niedociągnięć popełnionych podczas prac developerskich (kod nie jest uruchamiany)
  + wizualnie wykryte nieprawidłowości powinny zostać wykazane,
  + dobrą praktyką jest zgłaszanie sugestii,
  + należy dążyć do określonego standardu i unifikacji rozwiązań.
* Code Review kończy się wykonaniem wystawionego przez Developera Pull Request’a jako formalne potwierdzenie zachowania wytycznych realizacyjnych przy założeniu, że BOT przeszedł testy i jest w pełni funkcjonalny. Code Review:
  + **nowego BOT-a (artefaktów)** wykonywane jest tylko raz po zakończeniu developmentu oraz testów akceptacyjnych.
  + **po modyfikacji** wykonywane jest do skutku z uwagi na konieczność bezpośredniej weryfikacji wprowadzonych zmian.

# Wytyczne jakościowe

## Ogólne wytyczne jakościowe do stosowania podczas tworzenia kodu

* **Wszelkie wątpliwości** związane z **zasadnością wykorzystania** rozwiązania zawartego w instrukcji powinny być konsultowane z Architektem Rozwiązania RPA oraz Analitykiem
* **Kod w repozytorium nie powinien zawierać danych wrażliwych w żadnej formie (testowe inputy, adnotacje, komentarze, pliki tymczasowe lub wejściowe itp.).**
* Rozwiązanie ma **odzwierciedlać zapisy ujęte w instrukcji**, tj. kod powinien być modyfikowany dopiero po wprowadzeniu zmian w instrukcji.
* Jeśli zostanie **zidentyfikowany błąd w instrukcji lub problem logiczny** możliwy do zdefiniowania dopiero w czasie pracy nad kodem powinien być niezwłocznie **zgłoszony do Analizy** w celu naniesienia poprawki w dokumentacji.
* Obowiązuje całkowity zakaz korzystania z activity „Delay” oraz właściwości „DelayBefore” oraz „DelayAfter”. Każdorazowa potrzeba użycia **rozwiązań sztucznie i w sposób stały wydłużających czas procesowania** musi być uzasadniona i skonsultowana z Architektem Rozwiązania RPA. Jeśli rozwiązanie zostanie warunkowo dopuszczone powinno być opisane w adnotacji do activity lub workflow.
* **Znaki specjalne** występujące w rozwiązaniu w pewnych przypadkach powinny być zdefiniowane **kodem ASCII** np. sytuacja czyszczenia dwóch rodzajów spacji, kiedy „gołym okiem” nie widać różnicy Replace(„ ”,””).Replace(„ ”,””) a gdzie widoczne znaki to w rzeczywistości chr(160) oraz chr(32).
* **Selektory** powinny spełniać zasadę minimalizmu, zwięzłe ale spełniające swoje zadanie oraz niezawodne.
* Nie należy pozostawiać **omitt**’ów w celu wykluczenia fragmentu selektora. Jeżeli jakieś dane w selektorze są zbędne należy je **usunąć**.
* Nazwa pola „Status” jest zarezerwowana dla artefaktu „Raport”:
  + Użycie tej nazwy w Specific Content i Output spowoduje błędy przy generowaniu raportów.
* **Assign** – wielokrotnie powtórzone po sobie zaleca się sprowadzać do Multiple Assign.
* Kaskadowe **Else If** – należy unikać tworzenia bardzo rozbudowanych warunków Else If, lepiej odseparować od siebie zbyt mocno związaną przez activity logikę (np. zrobić osobny If, zagnieździć, lub rozbić na więcej kroków)
* Nazwy kolejek nie powinny być przekazywane do activity za pomocą zmiennych, lecz wybieranie z listy (inaczej Orchestrator nie wykryje błędnie podanej wartości lub jej braku)

### Artefakt Do\_Kolejki

* Powinien być wykorzystywany aktualny **standardowy framework** zawierający dopracowaną obsługę błędów oraz podstawowe funkcjonalności, inne rozwiązanie należy skonsultować z Architektem Rozwiązania RPA.
* Artefakt powinien zawierać implementację **„Wymagalności danych”** w celu weryfikacji danych wejściowych i zmiennych przekazywanych do dalszego procesowania.
* Artefakt „DoKolejki” powinien zawierać Log Message z informacją o **ilości dodanych pozycji do kolejki** z wyszczególnieniem ich typów.
* Artefakt „DoKolejki” powinien zawierać **ExceptionScreenshots**, jeżeli jest wykorzystywane GUI.
* **Auto-retry** na kolejkach nie jest wykorzystywany.
* Obowiązkowo należy zaimplementować **obsługę możliwych duplikatów** podczas zapełniania kolejki jeśli rozwiązanie zakłada unikalny reference.
* **Selekcja danych wejściowych** - dobrą i zalecaną praktyką jest implementacja rozwiązania umożliwiającego selekcję danych testowych.

### Artefakt Przetwarzanie

Artefakt „Przetwarzanie” powinien bazować na oficjalnym ReFramework’u, który zawiera wszelkie wypracowane przez Zespół RPA rozwiązania gwarantujące stabilność jego pracy, a ponadto:

* dla **systemu AUMS**:
  + w razie wystąpienia Application Exception powinien dokonywać zamknięcia wszystkich okien w AUMS żeby zapobiec zablokowaniu któregoś z elementów systemu, np. umowy, zlecenia OT,
  + zapisywać szczegóły techniczne błędu aplikacji AUMS w lokalizacji ExceptionScreenshots odpowiedniej dla środowiska,
  + zalecane jest by korzystać z rozwiązań biblioteki Common dla AUMS,
  + praca na wielu instancjach AUMS musi być uzasadniona i skonsultowana z Architektem Rozwiązania RPA.
* powinien być przygotowany do pracy na wielu maszynach.
* powinien być skonstruowany w takim sposób, aby w przypadku błędu aplikacyjnego zrzut ekranu pozwalał na diagnozę problemu.

**NIESTANDARDOWY FRAMEWORK**

Dopuszcza się wykorzystanie niestandardowego framework’u jeśli wymaga tego proces biznesowy ale każdy taki przypadek powinien zostać zaopiniowany przez Architekta Rozwiązania RPA.

### Artefakt Raport

* Raportowanie musi zawierać argumenty wejściowe dla zakresu dat, z którego ma być generowany raport, argument ma być podany w formacie DateTime.
* Artefakt raportu powinien bazować na szablonie dostępnym w repozytorium, który spełnia szereg wypracowanych założeń, posiada odpowiednie cechy oraz jest zaprojektowany z myślą dopasowania do aktualnych potrzeb wynikających ze specyfiki procesu.
* Artefakt powinien pobierać dane z API Orchestratora w minimalnym potrzebnym zakresie jednokrotnym requestem aby uniknąć nadmiernego obciążenia bazy danych Orchestratora. Dopiero tak pobrany zakres danych powinien być poddany dalszej obróbce (np. podział na kilka raportów).
* Każdy artefakt oparty na niestandardowym rozwiązaniu (customowy) powinien być skonsultowany z Architektem Rozwiązania.
* Artefakty raportujące działające na dużej liczbie danych czy plików powinny być zaprojektowane w sposób przewidujący względy wydajnościowe i w razie wątpliwości skonsultowane z Architektem Rozwiązania, Analizą lub Utrzymaniem.
* Przyjęto, że tworzone raporty powinny być jak najprostsze i zgodne z oficjalnym szablonem a ich funkcjonalności analityczne ograniczone.

### Config

* Powinien być stosowany wzór Config.xlsx z osobnymi arkuszami dla DEV i PRD.
* Config.xlsx nie powinien zawierać zmiennych lub stałych, które mogą być umieszczone w kodzie i praktycznie nie będą podlegały konfiguracji.
* Nazwy kolejek należy umieszczać w kodzie.

### Przechowywanie plików wykorzystywanych w procesie

* Korzystamy z lokalizacji „Input-data-TST”, która powinna zawierać testowe odpowiedniki plików wykorzystywanych przez BOTa produkcyjnie..
  + Najpierw przygotowywany jest zasób TST a następnie przenoszony na PRD.
* Duże ilości plików lub pliki o dużym rozmiarze powinny być przekazywane na zasób biznesowy.
* Dozwolone jest zapisywanie plików tymczasowych lokalnie w folderze artefaktu ale należy pamiętać o obowiązkowym czyszczeniu lokalizacji podczas pracy BOT-a (najlepiej przed i po). Tutaj jest pole do interpretacji więc należy w razie potrzeby skonsultować z Architektem Rozwiązania.
* Pliki nie powinny trafiać do zdalnego repozytorium.
* Zalecane jest korzystanie z **magazynu** **Storage Buckets** udostępnianego przez Orchestrator za pomocą odpowiednich activities.
  + Użycie **magazynu** będzie uwzględniane w instrukcjach procesów RPA.
  + Dozwolone jest korzystanie z **magazynu** w innych celach niż tylko wskazanych w instrukcji ale po konsultacji z Architektem Rozwiązania RPA w celu zachowania porządku i ograniczenia stosowania nieoptymalnych rozwiązań.

### ExceptionScreenshots

* Exception Screenshots powinny być zapisywane do folderu o nazwie artefaktu *np. CyprianEDO, CyprianSOD* oraz jeśli trzeba, osobno dla kolejkowania.
* Powinny być wykonywane w momencie umożliwiającym diagnozę kodu (nie za późno, nie za wcześnie, nie na widoku którego nie dotyczą).

### Wysyłka maili

* **Unikamy wysyłania maili** przez BOT-a. Jeśli proces tego wymaga zastosowanie powinno być każdorazowo skonsultowane.
* Maile wysyłane ze skrzynki botów muszą zawierać informację, że jest to wiadomość wysłana automatycznie oraz **prośbę o nieodpowiadanie na nią.**
* Mail powinien korzystać ze standardowego szablonu.

**SKRZYNKA MAILOWA BOT**

[tok.robot.test@tauron.pl](mailto:tok.robot.test@tauron.pl)

### Praca na aplikacjach web

• Artefakty korzystające z przeglądarek mają korzystać z **trybu prywatnego** o ile nie ma to negatywnego impaktu na działanie systemu, wyjątkiem jest też system Maximo.

• Preferowaną przeglądarką jest **Chrome**.

• Gdy robotyzowany proces z jakiegoś powodu musi być realizowany w Internet Explorer lub nie działa poprawnie w Chrome należy korzystać z MS Edge w trybie Enterprise Mode.

### Czyszczenie plików tymczasowych

**BOT powinien usuwać wszelkie pliki tymczasowe utworzone podczas jego pracy.**

Zalecane jest również czyszczenie wykorzystywanych w procesie folderów roboczych również przed rozpoczęciem pracy artefaktu ani uniknąć nieprawidłowości w jego funkcjonowaniu.

# Procedury

## Zasady ogólne

Tworzenie kodu BOTa odbywa się w ramach procesu 3.1\_20\_6\_3 Projektowanie BOT’a, gdzie opisani są uczestnicy procesu oraz ich zadania.

* Każdy developer posiada dostęp do jednego konta AD dedykowanego dla prac developerskich co umożliwia dostęp do ujednoliconego środowiska wytwórczego, oraz rozliczanie wykonywanych akcji.

## Udostępnienie danych logowania z uprawnieniami produkcyjnymi

Developerzy nie posiadają uprawnień produkcyjnych. W przypadku zidentyfikowania na etapie Analizy bariery uniemożliwiającej napisanie BOTa z wykorzystaniem baz nieprodukcyjnych Analityk prowadzący proces powinien wysłać maila o zaistniałej sytuacji z argumentacją do Kierownika Sekcji RPA oraz Opiekuna Utrzymania. Będą oni podejmować decyzję o go/not-go i przekazaniu uprawnień do Developera.

Dostępy produkcyjne wysyłane są w następujący sposób:

* login przekazywany jest drogą mailową
* zabezpieczony hasłem plik zawierający hasło przesyłany jest osobną wiadomością
* hasło do pliku wysyłane jest SMS-em

## Przejście DEV -> PRD

* Po definitywnym ukończeniu pracy nad Botem należy posprzątać środowisko DEV – dotyczy to głównie kolejek ale też dodatkowo stworzonych assetów.
* Każdy Developer ma dostęp do tych samych assetów zlinkowanych z folderem Default.

Transakcje o statusie Delete są uważane jako zaopiekowane, przeznaczone do wyczyszczenia z poziomu bazy danych.

## RPA TV

Przed uruchomieniem BOT’a na RPA TV należy spełnić poniższe warunki.

1. Plik readme.md – sprawdzić czy są wskazane:
   * **nazwy kolejek** oraz ich konfiguracja (np. unique),
   * wykorzystywane **assety** wraz z wartościami default.
2. Dokumentacja – sprawdzić czy instrukcja nie zawiera komentarzy a DDP jest podpisane.
3. Szablony Issue – sprawdzenie czy są zawarte w kodzie
4. Przygotować **trzy typy transakcji** - upewnić się czy BOT prawidłowo zamyka okna w przypadku Sukcesu, BRE oraz Appex.