

# **Riktlinjer vid RPA- utveckling**

**En handbok för UiPath-utvecklare**



## Dokumentinformation

<b>Titel</b>	Riktlinjer för RPA-utveckling
<b>Skapat av</b>	Claes Nyborg
<b>Datum</b>	2025-08-18
<b>Godkänt av</b>	Jari Koponen
<b>Version</b>	2.1

## Förändringshistorik

Version	Datum	Status och eventuell förändringsorsak	Utfärdare
2.0	2022-11-21	Omarbetat och förenklat stora delar av dokumentet.	Claes Nyborg
2.1	2025-08-18	Förtydligat rutiner rörande säkerhet i avsnitten; 2.2.1.1, 2.2.1.4, 2.2.1.5, 2.2.11, 2.2.12, 2.2.14, 2.2.15, 2.2.1.4	Claes Nyborg

## Relaterade dokument

Ref	Version	Datum	Benämning
1	1.1	2021-04-08	Modell för RPA-utveckling

## Innehållsförteckning

1	Inledning.....	5
1.1	REGLER OCH RIKTLINJER .....	5
1.2	KÄLLOR.....	5
1.2.1	Källor från dokumentationen för UiPath Studio .....	5
1.2.2	Källor från dokumentationen för UiPath Orchestrator.....	5
1.2.3	Kodprinciper.....	6
2	Kodningsstandarder.....	6
2.1	NAMNKONVENTION .....	6
	Variabler.....	6
	Argument.....	7
2.1.1	Aktiviteter.....	7
2.1.2	Flödesfiler (Workflow) .....	7
	Projekt och delprojekt (Projects i UiPath).....	7
2.1.3	Robotobjekt .....	8
2.1.4	Bibliotek (library).....	8
2.2	UTVECKLING OCH DESIGN .....	8
2.2.1	Utvecklingsstadier .....	8
2.2.2	Principer att följa vid utveckling .....	9
2.2.3	Variabler och värden .....	10
2.2.4	Rent och snyggt.....	10
2.2.5	Källkodshantering .....	11
2.2.6	Ramverk för utveckling av robotprocess .....	11
2.2.7	Projektuppdelning .....	11
2.2.8	Noteringar och kommentarer .....	12
2.2.9	Felhantering.....	12
2.2.10	Kö-ärenden .....	13
2.2.11	Bibliotek och återanvändning.....	13
2.2.12	Externa kodpaket.....	13
2.2.13	API:er och maskingränssnitt .....	13
2.2.14	Loggning .....	14

2.2.15	Peer review .....	14
2.3	BEHÖRIGHETER .....	14
3	Orchestrator .....	15
3.1	NAMNKONVENTION .....	15
3.1.1	Maskiner .....	15
3.1.2	Robotar .....	15
3.1.3	Tillgångar (Assets) .....	15
3.1.4	Köer (Queues) .....	15
3.2	STRUKTUR .....	16
3.3	BEHÖRIGHETER .....	16



# 1 Inledning

Detta är tänkt att vara ett stöd för framförallt utvecklare av automatiseringar i UiPath men också till viss del för administratörer av plattformen, mer om det nedan. När alla utvecklare använder samma praxis blir underhållet och förvaltningen av automatiseringarna lättare och det blir även lättare att samarbeta. Dokumentet sätter också en standard som ökar säkerheten rörande automatiseringar, gör automatiseringarna stabilare och återanvändning lättare.

Dokumentet är också, som nämnt ovan, i vissa fall tänkt som stöd för administratörer av UiPath-plattformen främst genom att sätta en tydlig och beskrivande namnstandard för entiteter i orchestratorn. Detta för att underlätta och strukturera arbetet som utförs i Orchestratorn.

Detta dokument bygger på grundriktlinjerna som finns uppsatta i UiPaths dokumentation för både Studio och Orchestrator. Till dessa grundriktlinjer har mer allmänt kända utvecklarriktlinjer adderats.

## 1.1 Regler och riktlinjer

Detta dokument innehåller både regler, som är specifika och obligatoriska, samt även riktlinjer, som inte är obligatoriska utan rekommenderade förhållningssätt.

De regler som är obligatoriska har angivits som ”ska” medan rekommenderade handlingsätt men som är inte obligatoriska har angivits som ”bör”. Regler som inte har angivits som varken ”ska” eller ”bör” ska ses som rekommenderade men inte tvingande regler, dessa kan t.ex. ha angivits som ”kan”, ”gärna” eller annat.

## 1.2 Källor

Lägg till vad på varje länk som hänvisas till.

### 1.2.1 Källor från dokumentationen för UiPath Studio

- **Workflow Design** (<https://docs.uipath.com/studio/docs/workflow-design>): Naming Conventions, Comments and Annotation. Datum: 2020-04-23.
- **Projekt Organization** (<https://docs.uipath.com/studio/docs/project-organization>): Alla avsnitt. Datum: 2020-04-23.
- **Automation Lifecycle** (<https://docs.uipath.com/studio/docs/automation-lifecycle>): Test, Release, Monitoring. Datum: 2020-04-23.

### 1.2.2 Källor från dokumentationen för UiPath Orchestrator

- **Automation Best Practices** (<https://docs.uipath.com/orchestrator/docs/automation-best-practices>): Robots, Environments, Queues. Datum: 2020-04-23.



- **Deployment and Configuration Considerations**  
(<https://docs.uipath.com/orchestrator/docs/deployment-and-configuration-considerations>): User and Robot Permissions. Datum: 2020-04-23.

### 1.2.3 Kodprinciper

- 10 Basic Programming Principles Every Programmer Must Know  
(<https://www.makeuseof.com/tag/basic-programming-principles/>) Datum: 2022-02-04.

## 2 Kodningsstandarder

Denna del är inriktad på standarder att beakta vid utveckling av automationer.

### 2.1 Namnkonvention

Då UiPath är på engelska och automatiskt döper aktiviteter till engelska ska engelska användas för all namnsättning, loggning, och kommentering för att det ska vara så lätt som möjligt att läsa och skapa flöden. Endast i väldigt specifika fall kan svenska ord användas, där det t.ex. inte finns något motsvarighet i engelska språket.

Beskrivande och meningsfulla namn ska alltid användas för allt, t.ex. workflow-filer, aktiviteter, argument och variabler för att förklara deras syfte och användning i projektet.

Projekten i sig ska också ha meningsfulla beskrivningar, eftersom dessa visas i Orchestratorns användargränssnitt. Att ha en tydlig namngivningskonvention i Orchestratorn för t.ex. tillgångar (assets), köer (queues), mappar osv. underlättar användandet av Orchestratorn.

Alla arbetsflöden bortsett från Main-flödet (som finns i ramverken) ska innehålla ett verb som beskriver vad arbetsflödet gör, t.ex. *GetTransactionData*, *ProcessTransation*, *TakeScreenshot*.

#### Variabler

- En variabel bör endast användas för ett syfte.
- Minimera omfattningen för varje variabel, dvs. använd i ett så begränsat flöde som möjligt.
- Håll aktiviteter som använder samma variabel/er så nära varandra som möjligt i flödet.
- Variabler ska **alltid** ha meningsfulla namn som beskriver variabeln utan förkortningar (om de inte är väldigt vedertagna, t.ex. Url).
- Vid namngivning ska **kamelnotation med stor första bokstav** (eng. upper Camelcase eller Pascal Casing) användas för variabler. För kamelnotation med stor första bokstav används sammansatta ord, inga andra tecken mellan orden, där varje ord börjar med en stor bokstav. T.ex. *TransactionNumber*, *FilePath*, *ReportName* osv.
- Variabler bör inte namnsättas utifrån ungersk notering, dvs. variabelnamn ska **inte** ha prefix som anger variabeltyp (t.ex. str, int, bool, osv). Detta gör namnen svårlästa och

samt att variabeltypen är lätt att se när variabeln dyker upp i UiPath. Undantag för detta är om det finns variabler som skulle kunna ha samma namn men behöver skiljas mot varandra då kan t.ex. listor heta *UserList*, eller en tabell heta *UserDataTable*.

- Bool-variabler: Dessa variabler bör ha namn som kan vara sant eller falskt. Använd gärna t.ex. prefixet *Is* följt av namnet, t.ex. *IsVisible*, eller *Exists* eller motsvarande efter namnet, t.ex. *ApplicationExists*.

## Argument

För argument till flödesfiler gäller samma regler och riktlinjer som för variabler men med nedanstående skillnad:

- Varje argument ska ha ett prefix beroende på riktning: **in**, **out** eller **io** följt av understreck ("\_"). Exempel: *in\_Config*, *out\_InvoiceNumber*, *io\_RetryNumber*, *in\_Employees*.

Ovanstående underlättar att se om en variabel har använts fel i koden eller om riktningen på argumentet är åt fel håll.

### 2.1.1 Aktiviteter

Aktivitetsnamn ska tydligt återspegla de åtgärder som vidtagits, t.ex. *Click "Save" Button*. Utgå gärna från den automatiskt skrivna titeln som beskriver åtgärden (t.ex. *Click*, *Type Into*, *Element Exists* osv) och försök justera för mer specificitet själv (ang. t.ex. elementet). I de fall en aktivitet kastar ett fel (exception) kommer felet innehålla aktivitetsnamnet vilket underlättar felsökning och förståelse av vad som orsakat felet. Även läsbarheten av koden förenklas.

### 2.1.2 Flödesfiler (Workflow)

- Namnen ska skrivas med kamelnotation med stor första bokstav.
- Flödesfiler bör grupperas utifrån funktion eller applikationens olika delar i undermappar för att lättare hitta bland filerna.
- Ändra inte namnen på mallramverkets fördefinierade flödesfiler.
- Flödesfiler som skapas för testning bör inledas med *Test\_*, t.ex. *Test\_ConvertToABC*.

### Projekt och delprojekt (Projects i UiPath)

- Namnen ska skrivas med kamelnotation med stor första bokstav.
- Namnen ska vara beskrivande för vad processen/projektet gör, eftersom processnamnet är verksamhetsnära och ofta används vid kommunikation kan dessa vara på svenska.
- Om processen utgörs av delprocesser (flera projekt/automationer för samma affärsprocess, t.ex. vid användande av dispatcher- och performerdelar), ska affärsprocessens namn anges först och därefter delprocessens namn som prefix: T.ex. *RegistreraTimanställda\_Performer*.

### 2.1.3 Robotobjekt

- Robotobjektets namn står i ett ett-till-ett förhållande för processen som utförs av robotobjektet.
- Med tanke på punkten ovan ska namnet ska vara beskrivande för processen som robotobjektet utför, t.ex. ”Skapa anställningsavtal för timanställningar”, ”Kontera leverantörsfakturor från leverantör A”.

### 2.1.4 Bibliotek (library)

Vid utveckling av biblioteksprojekt utgår man från samma namnkonventioner som är nämnda ovan men med nedanstående skillnader.

#### 2.1.4.1 Flödesfiler

- Namnsätts MED mellanslag för att passa in bland övriga aktiviteter i aktivitetsfönstret när biblioteket är importerat som ett beroende (dependency).

#### 2.1.4.2 Argument

- Kan namnsättas utan prefix t.ex. in, out, io för att likställa med hur övriga aktiviteter är utformade under egenskaper (properties) när biblioteket är importerat som ett beroende (dependency).

## 2.2 Utveckling och design

### 2.2.1 Utvecklingsstadier

Nedan beskrivs minsta möjliga stadier för utveckling av en färdig automation och vad som behöver beaktas inom varje stadie.

#### 2.2.1.1 Dokumentation och design

Dokumentation av processen utförs först tillsammans med processexpert från verksamheten. Efteråt (och till viss del under tiden) som dokumentationen sker skapas en lösningsdesign. Denna design ska godkännas av ansvariga från verksamheten (robotobjektägare) innan utveckling börjar.

I samband med skrivandet av dokumentationen ska infösäkerhetsklassning göras av det data som behandlas och även skapas av roboten vilket också noteras i dokumentationen. När nytt system ska hanteras av robot eller där det av annan anledning finns anledning ska även en riskanalys av processen utföras.

#### 2.2.1.2 Utveckling

Utvecklingen sker samtidigt som en dialog upprätthålls med processexpert från verksamheten eftersom det alltid uppkommer oklarheter under utvecklingens gång. Utvecklingen sker i applikationernas utvecklings- eller testmiljöer om det finns, annars behöver utvecklingen ske med stor försiktighet i applikationernas produktionsmiljö efter godkännande av ansvariga från

verksamheten. Utvecklaren behöver då ha full koll på vad som händer vid varje handgrepp som utförs i systemen.

### **2.2.1.3 Test**

Testningen sker i samråd med processexpert från verksamheten. Processexperten har störst förståelse för vilket data (både vanliga och avarter) som riktiga fall kan komma att innehålla i produktionsmiljön. Därför bör processexperten själv eller med hjälp av utvecklaren skapa testfall som tar motsvarar de scenarion som kan uppstå.

Testerna ska utföras i utvecklings- eller testmiljö för applikationerna men om applikationerna inte har denna typ av miljö kan tester utföras övervakat och med stor försiktighet i produktionsmiljön om ansvariga från verksamheten har godkänt det.

### **2.2.1.4 Produktionssättning**

Robotobjektet sätts i produktion efter utförda tester har blivit godkända av robotobjektets ägare (ofta genom förvaltaren av robotobjektet) och att både driften samt RPA-objektledaren har blivit informerad angående den planerade driftsättningen.

En driftsättning efterföljs (beroende på komplexitet i robotobjektet) av en stabilitetsperiod där robotobjektets körningar och jobb dubbelkollas för att eventuella buggar och fel snabbt kan åtgärdas.

### **2.2.1.5 Förändring**


Vid förändring av robotobjekt som redan är driftsatt ska ändringarna testas i sitt sammanhang i utvecklings- eller testmiljö (beroende på vad som finns att tillgå) med de olika data (och avarter) man förväntar sig kan komma i produktionsmiljön. I de fall där förändringarna är minimala eller av annan anledning inte anses behöver testas på ett omfattande sätt ska detta godkännas av de som är ansvariga för roboten inom verksamheten där roboten jobbar (dvs. ägare och/eller förvaltaren av robotobjektet).

Alltid vid förändring av robotprocesser ska beaktning göras över hur ändringarna påverkar säkerheten, både i stabilitet/driftsäkerhet samt ev. sårbarheter. Om möjligt ska säkerheten i användning av data, drift, eller annat förbättras i samband med förändring av process och om sårbarheter finns ska dessa minimeras.

Vid förändring av ett egenutvecklat bibliotek ska dess nyttjade paket samtidigt uppdateras om någon ny stabil version finns att använda. Detta för att få med ev. säkerhetsförbättringar som kan finnas i de nya versionerna.

## **2.2.2 Principer att följa vid utveckling**

Alla nedanstående principer är lätta att förstå som koncept men svårare att följa i praktiken. Hur långt man följer dessa är upp till utvecklaren och kan skilja sig från situation till situation men det är alltid bra att ha dessa i åtanke vid utveckling.



### 2.2.2.1 *Keep It Simple, Stupid (KISS)*

Denna princip är svårare att följa än man kan tro. Kort och gott handlar det om att inte utveckla något mer komplext än vad det behöver vara för att lösa uppgiften. Uppgiften ska lösas på det enklaste sättet för att underlätta förvaltning, felsökning och vidareutveckling.

### 2.2.2.2 *Don't Repeat Yourself (DRY)*

Kopiera inte kod från ett ställe till ett annat, gör istället flöden som går att återanvända. Var dock aktsam över YAGNI-principen nedan, gör inget som inte behövs men också KISS-principen och gör inte ett för avancerat flöde i onödan. Visar det sig att koden behövs på fler ställen, refaktorera och bryt då ut delen.

### 2.2.2.3 *Single Responsibility Principle (SRP)*

Ett flöde ska bara ha ett ansvar, dvs. en uppgift. Detta underlättar felsökning, återanvändning, vidareutveckling.

### 2.2.2.4 *You Aren't Going to Need It (YAGNI)*

Utveckla inte för ett eventuellt framtida behov. Utveckla när behovet väl dyker upp för att slippa skapa onödigt komplex och svårförvaltd kod.

## 2.2.3 Variabler och värden

Vid automatisering används ofta variabler och värden av olika slag beroende på olika scenarion. Dessa kan kategoriseras och användas utifrån nedanstående tabell. Det är aldrig fel att använda en mer lättjusterad lokalisering framför en lokalisering som är svårare att justera. Dock bör inte alla värden och variabler ligga i orchestratorn på grund av bruset det skapar och den ytterligare tiden det tar för utveckling. Använd inte mappsökväg vid hämtning asset:ar (se även under avsnittet mallramverk).

LOKALISERING	ANLEDNING TILL LOKALISERING	EXEMPEL
HÅRDKODAS I FLÖDET	Ändras aldrig, används på enskilda ställen i koden	Statiska selektorer, aktivitetsnamn
VARIABLER I FLÖDET	Används på flera ställen i koden	Väntetider
KONFIGURATIONSFIL UTANFÖR FLÖDET	Ändras ibland och används på flera ställen i koden	Könamn, väntetider, filtreringar
TILLGÅNGAR/ASSETS I ORCHESTRATOR	Ändras ofta och/eller av verksamheten och/eller beroende på miljö (prod/test)	E-postadresser, webbadresser, meddelanden, användarnamn, servernamn

## 2.2.4 Rent och snyggt

Innan det slutförda projektet laddas upp till Orchestratorn bör projektet och alla inkluderade flöden gås igenom och rensas upp. Ta t.ex. bort oanvända variabler och avaktiverade (disabled) aktiviteter. Kolla även så namnen på aktiviteter är beskrivande och unika och att varje arbetsflödesfil (workflow) har en beskrivande notering.

Personuppgifter ska aldrig finnas i ett projekt, säkerställ detta med en genomgång och rensning av data i variabler, kommentarer och noteringar, aktivitetsnamn, skärmdumpar osv.

### 2.2.5 Källkodshantering

För säkrare lagring, ändringshistorik samt för att lätt kunna samarbeta med andra utvecklare ska källkodshantering användas i så stor utsträckning som möjligt.

En förvaringsplats (repository) ska skapas för varje projekt (och bibliotek) i UiPath.

Ett beskrivande ändringsmeddelande (commit message) ska anges vid varje ändring (eng. commit). Det är rekommenderat att hålla ändringarna så små som möjligt eftersom det då är det lättare att hålla reda på vilka ändringar som är nygjorda och därmed skriva ett bra ändringsmeddelande. Tänk också på att ta bort eventuella skärmdumpar som har personuppgifter innan uppladdning.

Skärmdumpar, eller andra filer som kan innehålla personuppgifter får aldrig pushas till publikt repository.

Ändringar bör commitas så ofta som möjligt men minst en gång per dag för att relevanta ändringsmeddelanden ska kunna skrivas.

### 2.2.6 Ramverk för utveckling av robotprocess

En automatisering bör alltid utvecklas med grund i ett mallramverk eftersom detta underlättar och förbättrar utvecklingen på flera sätt. I mallramverken finns färdig högnivåhantering av fel, en miniminivå av loggning, justerbara processinställningar i extern Excel-fil samt ärendehantering för Orchestrator-köer.

I projektets konfiguration ska inte sökvägar för mappar användas för hämtning av tillgångar/asset:ar, om det inte finns en mycket specifik anledning till detta. Först och främst ska tillgångarna läsas från mappen där processen körs och om asseten ska delas bör den istället länkas mellan de olika mapparna i orchestratorn.

Så ofta som möjligt ska köer i Orchestratorn användas för ärenden som processen ska hantera. Detta för att kunna se historik för varje enskilt ärende, se allmän statistik, automatisk omprövning av misslyckade ärenden samt att enkelt manuellt kunna starta om misslyckade ärenden. Kö-ärenden (queue items) som skapas i Orchestratorn ska **inte** i onödan innehålla direkta personuppgifter (data som på ett eller annat sätt kan härledas till en specifik person) för att minimera för spridning av personuppgifter utan anledning. I största möjliga mån ska inga personuppgifter användas, i andra hand kan indirekta personuppgifter användas (t.ex. ärendenummer till ärende där personuppgifter finns), i tredje hand kan maskerade eller pseudonymiserade personuppgifter användas och om inget av dessa fungerar kan direkta personuppgifter användas.

### 2.2.7 Projektuppdatering



Det är rekommenderat att bryta upp projekten i mindre arbetsflöden, gärna så små som möjligt (se SRP- och DRY-principen). Flödena kan då testas oberoende av varandra samt återanvändas på ett modulärt sätt. Projekten ska delas upp utifrån applikationsspecifika flöden, som bara hanterar applikationerna samt flöden som innehåller affärslogik.

### 2.2.7.1 Applikationsflöden och affärslogikflöden

Som nämnt ovan ska affärslogik separeras från flöden för applikationshantering, dvs. automatiseringens hantering av gränssnitt. Applikationshanteringen ska läggas i ett eget biblioteksprojekt (library) och vara generell och processoberoende för att kunna återanvändas i flera processer. Ett flöde för applikationshantering ska även enbart utföra **en** specifik handling/funktion i ett gränssnitt (t.ex. ”öppna kund”, ”läs kundinformation”, ”ändra kundinformation”) och **inte** flera handlingar/funktioner (t.ex. ”läs och ändra kundinformation”) enligt SRP. På detta sätt kan dessa flöden återanvändas av olika processer oberoende av vad som ska uträttas av processen i stort och vilken affärslogik som används i den specifika processen.

### 2.2.8 Noteringar och kommentarer

Noteringar och kommentarer används för att beskriva vad som händer och varför där det behövs. Åtminstone varje arbetsflödesfil ska ha en inledande beskrivande notering med text om:


- Vad flödet gör.
- Vad flödet behöver för ingående tillstånd (precondition).
- Vilket avslutande tillstånd (post condition) som flödet har skapat.

Ha alltid i åtanke att en annan utvecklare bör kunna öppna projektet eller flödet och lätt kunna förstå vad flödet gör och varför.

Kommentera gärna mer och för mycket än för lite eftersom det underlättar förståelse och förvaltning av koden. Dock är det viktigare med beskrivande namn än väl kommenterad kod eftersom kommentarer glöms ofta bort vid uppdatering/förändring av koden.

### 2.2.9 Felhantering

Två typer av fel/undantag kan uppkomma vid körning av en process, ett förutsägbart fel eller oväntat fel. Utifrån detta kan felet/undantaget hanteras antingen genom att explicita åtgärder utförs för att hantera felet inom det specifika arbetsflödet där det uppkommer, eller genom att eskalera felet till en högre nivå i processen.

- **Förutsägbara fel** hanteras t.ex. genom att placera känsliga aktivitetssteg inuti i en ’Try Catch’-aktivitet eller i ett Retry-block där felet kan hanteras/motverkas.
  - På den högsta nivån, ska huvudprocessdiagrammet ha breda korrigerande åtgärder för att ta itu med alla **generiska och oförutsägbara fel** och för ökad stabilitet.
- 

Ramverksmallar som nämns senare i guiden har redan denna felhanteringsmekanism inbyggd.

### 2.2.10 Kö-ärenden

Kö-ärenden i orchestratorn bör användas så ofta som möjligt eftersom utvecklare, verksamheten och eventuellt andra intressenter med behörighet då kan se och följa varje ärende i sin helhet direkt i Orchestratorns webbgränssnitt.

Som tidigare nämnt ska kö-ärenden som skapas i Orchestratorn **inte** i onödan innehålla direkta personuppgifter (data som på ett eller annat sätt kan härledas till en specifik person). Hänvisa istället till t.ex. ett ärendenummer som finns i annat system där personuppgifterna som roboten ska hantera finns. Eller också bör personuppgifterna maskeras eller pseudonymiseras.

### 2.2.11 Bibliotek och återanvändning

Flöden som har skapats för möjlighet till återanvändning i andra projekt (t.ex. applikationsflöden som beskrivs ovan) ska skapas och användas som bibliotek (library). Detta gäller inte enbart applikationsflöden utan även andra flöden som kan tänkas vara intressanta att återanvända i andra projekt (t.ex. ”konvertera personnummer till ålder”). Biblioteken laddas upp och lagras i Orchestratorn. Från Orchestratorn kan biblioteken sedan återanvändas och uppdateras i flera projekt.

Vid varje ny automatisering ska alltid först officiella .Net-katalogen och UiPath-katalogen beaktas för färdiga funktioner och kodpaket och även i andra hand ej officiella funktioner och aktiviteter byggda av gemenskapen (community). Även den interna katalogen ska beaktas för färdiga kodpaket och om det redan finns paket/bibliotek, för t.ex. någon viss applikation eller funktion som ingår i den nya automatiseringen, ska detta bibliotek användas. Om ett bibliotek saknar en viss funktion eller del av applikationen ska biblioteket vidareutvecklas med dessa nya funktioner.

Vid förändring av ett egenutvecklat bibliotek ska dess nyttjade paket samtidigt uppdateras om någon ny stabil version finns att använda. Detta för att få med ev. säkerhetsförbättringar som kan finnas i de nya versionerna.

### 2.2.12 Externa kodpaket

Vid användning av externa kodpaket och funktioner ska dessa paket och dess utvecklare undersökas för att se att de håller en tillräcklig kvalitet. Bland annat kan antalet användare kollas, antalet och innehållet i eventuella community-kommentarer och uppdateringsfrekvens av paketet. Paketet ska godkännas av RPA-arkitekt, RPA-objektledare eller motsvarande vilket noteras i dokumentationen.

### 2.2.13 API:er och maskingränssnitt

När en applikation har ett maskingränssnitt bör detta användas framför användargränssnittet för att öka stabiliteten i automatiseringen och minska förvaltningskostnaden som kan uppkomma då användargränssnitt tenderar att ändras oftare än ett maskingränssnitt.

### 2.2.14 Loggning

Loggning ska utföras med 'Logga Meddelande'-aktiviteten (Log Message) eftersom man med den aktiviteten kan välja vilken nivå som loggningen ska ske på (Trace, Info, Warning, Error).

Meddelandet bör vara så beskrivande och specifikt som möjligt för att underlätta felsökning. Mer loggning ger en tydligare bild av vad som händer under processen och ger därför en tydligare bild av vad som har gått fel och varför. Åtminstone i början av varje flöde ska loggning ske (detta ska utföras) men gärna i slutet av flödet också (detta är utfört med detta resultat). Omfattande flöden kan även behöva loggning på många fler ställen än i början och slutet.

Fel som fångas ska också loggas samt om data hämtas från någon extern källa. Använd mest lämpad loggningsnivå utifrån situationen (Trace, Info, Warning, Error), där info är standardnivå och trace är detaljnivå för att logga specifika steg inom ett flöde. Warning används t.ex. vid avvikande situationer men som inte är rena fel eller kritiska för processens fortsatta hantering av ärendet.

Direkta personuppgifter (data som på ett eller annat sätt kan härledas till en specifik person) ska **inte** i onödan finnas i loggarna av samma anledning som det inte ska finnas direkta personuppgifter i kö-ärenden i onödan. Detta för att minimera spridning av personuppgifter i organisationen. Som vid övrig användning av personuppgifter bör även eventuella personuppgifter som används i loggar maskeras eller pseudonymiseras. Maskning ska noteras i dokumentationen.

### 2.2.15 Peer review

Innan en robot blir driftsatt ska denna ha genomgått en peer review eller motsvarande där en annan utvecklare än den som primärt har utvecklat lösningen får chans att se, kommentera, ställa frågor och på annat sätt diskutera lösningen. Detta för att lösningen ska hålla högsta möjliga kvalitet samt för löpande kompetensöverföring. Då är automatiskt fler än en utvecklare insatt i lösningen. Vem som utförde peer review:n noteras i dokumentationen.

## 2.3 Behörigheter

Behörigheter för robotar ska sättas utifrån principen 'Principle of Least Privilege' (POLP). Robotar ska alltså ha så lite behörighet/tillgång som möjligt för att kunna utföra uppgiften. Detta betyder också att varje robot bör ha egna konton, för att inte behörigheterna på ett visst konto blir för omfattande.

### 2.3.1.1 Förvaring och användning av behörigheter



Behörigheter ska **aldrig** lagras i koden eller i inställningsfiler utan ska så långt om möjligt förvaras i orchestratorn och i andra hand i Windows Autentiseringshanteraren (Credential Store).

Behörigheterna ska användas i en så liten omfattning (scope) som möjligt i koden. De ska alltså bara finnas där de behövs för att autentisering ska kunna utföras. Var också noga med att markera aktiviteter med behörigheter som "private" för att inte logga behörigheterna.

I så stor grad som möjligt ska "secure string"-klassen användas för lösenord för ett extra lager av säkerhet.

## 3 Orchestrator

Denna del sätter standarder för arbete och struktur i orchestratorn.

### 3.1 Namnkonvention

En konsekvent namngivningsstrategi med beskrivande namn ska användas även i Orchestratorn.

#### 3.1.1 Maskiner

Maskiner bör ha en beskrivning som anger om den är för produktion och/eller test eller har specifika applikationer installerade.

#### 3.1.2 Robotar

Robotar bör ha en beskrivning som anger vilka processer som robot utför.

#### 3.1.3 Tillgångar (Assets)

Tillgångar ska namnges enligt följande:

- Namnen ska skrivas med kamelnotation med stor första bokstav.
- Namnen ska vara beskrivande för vad tillgång används för. Eftersom tillgångar (assets) är verksamhetsnära och kan komma att behöva ändras av verksamheten kan dessa vara på svenska, t.ex. SökvägTillSkärmdumpar.
- Tillgångar bör även ha kort förklarande beskrivning.

Behörighetstillgångar (credential assets) ska namnges som övriga tillgångar men med följande tillägg:

- Namnet ska ange i vilket system som behörigheten gäller.
- Om krav på att byta lösenordet efter antal dagar finns ska detta anges i beskrivningen

#### 3.1.4 Köer (Queues)

Köer ska namnes enligt följande:



- Namnen ska skrivas med kamelnotation med stor första bokstav.
- Namnet ska innehålla processens namn som första del
- Namnet ska vara beskrivande för vilken typ av ärenden den innehåller (vad utgör/ingår ett ärende).
- Exempel: HandläggNyaAnställningar\_Timanställda

## 3.2 Struktur

Mappstrukturen i orchestratorn ska först uppdelas till Produktion och Test och därefter byggas upp utifrån organisationsträdet (för vardera miljön). Inom varje funktion/affärsområde ska sedan en undermapp skapas för vardera robotobjekt.

Denna struktur möjliggör att kunna avgränsa tillgång/behörighet utefter användarens organisatoriska lokalisering men också att minimera brus för användare och utvecklare som ofta jobbar med enbart en automatisering åt gången.

För varje robotobjekt förvaras tillgångar (assets), köer, processer, storage buckets osv. inom det specifika robotobjektets mapp. Bara entiteter som delas mellan flera robotobjekt lagras på en högre nivå i strukturen och då i så nära nivå till robotobjekten som möjligt.

## 3.3 Behörigheter

Behörigheter för användare i orchestratorn sätts på samma sätt som robotarnas behörighet utifrån 'Principle of Least Privileges' (POLP), dvs. med så liten behörighet som möjligt utifrån vad situationen kräver. Det kan t.ex. finnas användare i verksamheten som behöver kunna justera robotobjektets tillgångar och starta om ärenden medan andra bara behöver kunna följa robotobjektets körningar utan att kunna ändra något.