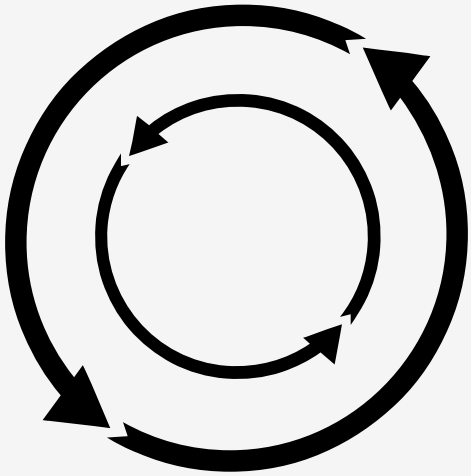


Leveraging Agile to realize a Fully Automated Enterprise™

Agile RPA

Does the A in RPA stand for Agility? Sometimes yes, generally no. Is this a problem? Probably.
In this paper, we explore why agile delivery frameworks (e.g., Scrum, Nexus) can be essential for scaling robotic process automation (RPA) to turn traditional enterprises into fully automated enterprises.

SCRUM.ORG • UIPATH.COM • INGO PHILIPP

Contents

1	**Vision**	**1**
2	**Mission**	**2**
2.1	Evangelism	3
2.2	Discovery	4
2.3	Delivery	5
2.4	Framework	5
2.5	Transition	6
3	**Approach**	**7**
3.1	Predictability	7
3.2	Transparency	8
3.3	Adaptability	9
3.4	Segmentation	11
3.5	Complexity	13
3.6	Controllability	14
3.7	Engineering	16
3.8	Scalability	17
3.9	Governance	20
3.10	Orchestration	21
3.11	Collaboration	24
4	**Benefits**	**25**
5	**Conclusion**	**27**

This is an experience report. It tells the story about Susan. Susan has been leading an RPA initiative in a large enterprise in the financial industry. In the first two chapters, we will provide some contextual information to enable you to understand why and how Susan got started with RPA. In the remaining chapters, we will explore the main lessons Susan has learned in transitioning from a classical to an agile way of RPA delivery.

1 Vision

It is the year 2018. The HR department in Susan's organization conducted a series of interviews and surveys to assess the level of repetitiveness in the everyday work of their employees. It was found that their world of work was filled with numerous dreary, mundane, and repetitive tasks.

The spectrum of these repetitive tasks ranged from performing business activities such as payroll processes, invoice processes, risk-assessment processes, credit-checking processes, and billing processes to on/off-boarding of employees, collecting data for competitive research, generating mass emails, processing sick leaves, submitting expenses, and issuing refunds. This list could go on and on. It's not complete. You get the point. 32% of the daily activities in 68% of all job functions across the major departments (e.g., legal, finance, marketing, sales, human resources) in Susan's organization were repetitive. This gave rise to taking the still emerging technology of robotic process automation (RPA) seriously.

Just imagine your most tedious and dreary task at work, and then imagine that some robot could do it for you. That's the main idea behind RPA. These robots aren't physical robots; they are software robots. So, a robot is just a piece of software that performs automated actions to free up human labor. Freeing humans from these repetitive tasks means enabling them to spend their time on the more exciting, fulfilling, and valuable parts of their jobs. This includes duties that require creative thinking, critical thinking, emotional thinking, problem-solving, strategic decision-making, and many other skills that machines don't have yet. In short, through RPA you outsource repetitive tasks to robots. You move these tasks from human workers to virtual workers.

The results gathered by HR made the leadership team (e.g., senior, upper, executive management) prick up their ears. External reports about the potential benefits of RPA then captured their interest even more. For example, a survey by Capgemini (2018) showed that a proactive approach to automation, in contrast to a reactive one, can lead to a variety of business benefits such as increased revenue and profitability, improved customer experience, faster product/service delivery, better compliance with regulations, higher quality of software development, among many others. This showed that RPA could reap significant ROI. Simply put, the dollars you put into automation create more dollars coming out.

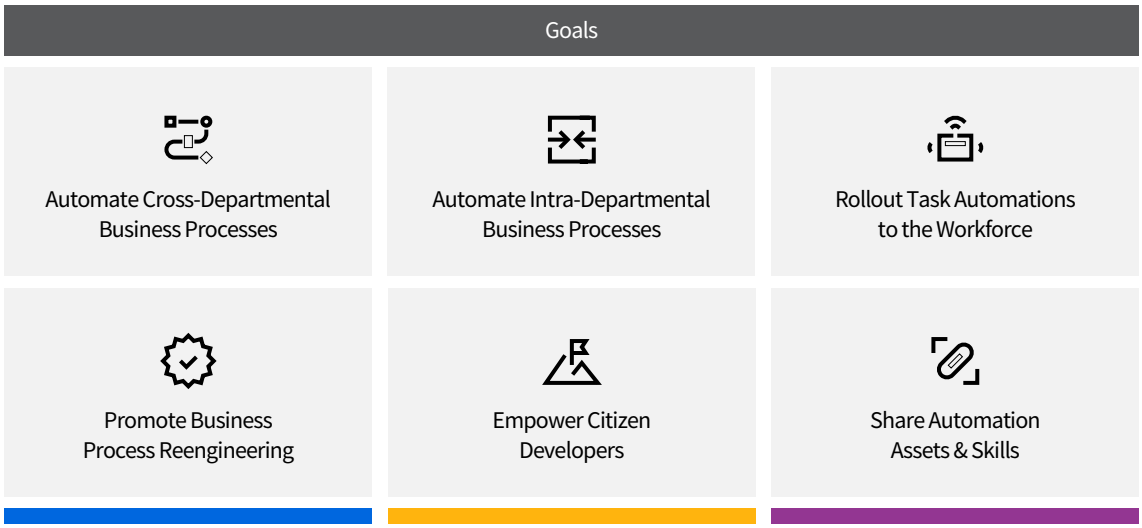
Long story short, leadership sponsorship was established. A company-wide initiative for RPA was launched. RPA was considered a strategic rather than a tactical discipline right from the beginning. It was part of a bigger digital transformation program driven by the leadership team. The objective of this digital transformation program was manifold. It ranged from increasing operational efficiency, reducing costs, mitigating risks to accelerating innovation, boosting corporate growth, improving employee happiness, and enhancing customer experience. The reason was simple. Susan's organization realized that its competition was eating their lunch, and every hour the competition got more and more hungry. It was getting harder and harder to outpace them.

RPA was considered an integral component to accelerate digital transformation. The main mantra of this RPA initiative was to let humans be humans. The slogan was 'make robots so that humans do not need to be robots' to liberate the boundless potential of people. The vision was not to do this just for some employees but for all employees in Susan's organization. In other words, the ultimate goal was to have a robot for every person to turn Susan's organization into a fully automated enterprise.

2 Mission

This is where Susan gets into the game. Susan was put in charge to translate this big hairy audacious goal into reality. She has been with the company for about five years and has led multiple successful initiatives for scaling test automation across the enterprise. So, she knew the internal workings of the company and she understood what it takes to cross the chasm between successful pilots and large-scale deployments of automation. Susan became the first RPA lead in her organization.

Susan started to put a team of technologists together to form the first robotic center of excellence (CoE). She started to look for candidates within her organization. She knew that the drive to automate dreary, mundane, and repetitive tasks was prevalent. It just wasn't called RPA and it just didn't arrive in the business domain yet. Automation was the daily business of software development and IT Ops teams. These teams were swamped with repetitive activities (e.g., provisioning, deploying, configuring, orchestrating, and testing infrastructure and internal/external software applications). Susan leveraged this in-house experience, passion, and expertise in automation. She managed to convince former test automation engineers, software developers, system architects, and infrastructure engineers to form the core of the CoE. These roles turned into RPA developers, RPA solution architects, and RPA infrastructure engineers. This implies that the knowledge of IT Ops as well as software development was baked into the CoE right from the start.



The goals of the RPA CoE were manifold: (1) Automate cross-departmental business processes (e.g., employee on/off boarding). These are processes that cut across multiple departments and engage multiple people. (2) Automate intra-departmental processes (e.g., end-of-month closing). These are processes that are contained in a single department. (3) Rollout task automations to the workforce. These are relatively simple processes that only involve one person. (4) Promote business process re-engineering (e.g., simplification, standardization, elimination) in close collaboration with business, IT Ops, and software development. (5) Empower technically savvy employees (citizen developers) to develop and maintain RPA for themselves and their departments by providing trainings and by putting guardrails around RPA. (6) Accelerate automation by sharing automation assets and skills across software development teams, IT Ops teams, and the RPA CoE on the business side.

Susan wanted to avoid that the CoE becomes an isolated, disconnected island in the company's landscape. She didn't want that RPA stops at the boundary of the CoE. Her goal was that the CoE becomes part of the company's DNA that weaves into the fabric of the company's culture. So, the goal was to make RPA become a

shared goal to realize the vision of a fully automated enterprise by collaboratively democratizing automation in the organization. Susan knew that collaboration and cooperation are key to scale RPA.

So, Susan decided to start with something that's apparently too big to then pull it back into reality. She did so because the main lesson she has learned in scaling test automation across the enterprise was: It's easy to pull things back, but it's hard to ramp things up. For that reason, Susan didn't set little goals but big goals for the CoE. She preferred shooting for 80 and being disappointed when hitting 70 rather than shooting for 20 and being ecstatic when hitting 21. Susan's motto was that little goals will lead to little achievements; big goals might lead to big achievements. So, Susan's goals were big. She made them smart by using SMART KPIs (i.e., simple, measurable, attainable, relevant, and time-constrained) to continuously measure the overall progress towards achieving these goals from an operational perspective (e.g., robot utilization, robot downtime, robot error rate) and business perspective (e.g., cost savings, time savings). In doing so, Susan distinguished between KPIs that measure the desired final result and KPIs that measure the path to getting to the final result.

Here's how Susan started to put these goals into practice. For the sake of simplicity, we will only briefly outline the big picture. Please refer to our automation operating model to learn more about how to quickly progress through the early stages of RPA and how to operate at scale.

2.1 Evangelism

First, Susan started to elevate the conversation around RPA. RPA was still in its infancy. The concept of RPA was poorly known and understood. For that reason, the CoE conducted a series of internal webinars, seminars, hands-on workshops, and trainings to evangelize RPA. Marketing collaterals such as factsheets, one-pagers, and brochures complemented these sessions to give the employees the information they need, every step of the way. For example, these marketing collaterals included information about how employees can recommend business processes for automation and how they can get feedback about the status of the implementation.

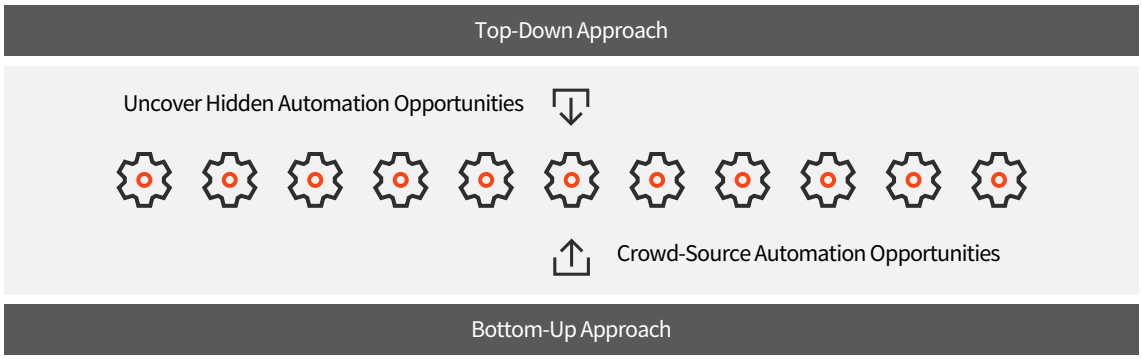
In parallel, a pilot project in one department (e.g., human resources) was carried out to demonstrate the value of RPA in concrete terms. The main mantra was to demonstrate maximum value with minimum effort. The pilot project included 'low hanging fruit' intra-departmental processes. These processes were characterized by their high potential (e.g., high transactional volume, high time/cost savings) and low complexity (e.g., rule-based, structured data). The goal was to let the people see and feel the real value of RPA based on a real-life project. So, this pilot project was both a motivational tool to get people excited about RPA and an educational tool that enabled the people to familiarize themselves with the still unknown technology of RPA.

Susan targeted these sessions to three stakeholder groups: senior managers, LOB managers, and employees. Even though Susan had the initial buy-in from the leadership team, it was crucial for the success of this RPA journey to keep them constantly engaged and informed about potential roadblocks. These people were critical to build and maintain the momentum of this initiative. In addition, they made sure that the goals of the LOB managers were aligned with their goals with respect to RPA.

Susan knew that internal resistance to innovation is always happening. It's inevitable. The goal is to minimize it. Therefore, she designed and implemented a comprehensive communication and advocacy plan to achieve the following: (1) Create awareness about the capabilities and potential business benefits of RPA. (2) Share the vision and mission of the RPA initiative. (3) Identify process owners, subject matter experts, and automation champions. (4) Get a first rough picture about suitable use cases for RPA. (5) Ease the employees' concerns that were mainly centered around the common misconception that RPA is coming to take their jobs. In summary, this communication and advocacy plan was key to jumpstart the demand generation for RPA.

2.2 Discovery

In parallel, the CoE started to create a backlog for automation opportunities. If you don't make a clear choice, you're in fact trying to be everything to everyone, with the result of not being really good at anything (Michael Porter). Instead of focusing on cross-departmental processes, Susan decided to focus first on intra-departmental processes to fully automate one department (e.g., human resources, legal, finance). So, Susan followed a per-department approach rather than a per-process approach in the beginning (e.g., first 12 months) of this RPA initiative. The reasons were manifold. In a nutshell, the automation of cross-departmental processes usually leads to an efficiency gain of around 7% to 15% in terms of time reduction in a single department. This is much less than what people expect from RPA. On the other side, the automation of intra-departmental processes usually reduces about 20% to 30% of people's total time in one department. Those time reductions are big enough to reorganize people's work substantially. So, even though the per-department approach achieves slightly less time savings than the per-process approach from an organizational perspective, the per-department approach turns out to be more efficient and effective to prove to the entire organization, and most importantly, to the executive leaders and department leaders that RPA truly increases productivity. This approach also helps to turn department leaders into enthusiastic automation champions who then encourage other department leaders to follow suit. Susan calls this the virtuous cycle of automation.



The next step was to generate enough demand to sustain a continuous RPA throughput. So, process discovery and process prioritization were on top of the agenda. The automation opportunities were discovered in two ways. First, the top-down approach. Here, the CoE used discovery tools such as UiPath Process Mining to source automation opportunities from the event logs of business applications (e.g., SAP, Salesforce, ServiceNow). Secondly, the bottom-up approach. Here, the CoE used UiPath Automation Hub to crowd-source automation ideas from the employees, who used UiPath Task Capture to submit their ideas in the form of process design documents (PDD). The CoE then prioritized the automation opportunities based on their business value. The business value of an automated business process is given by the relation between the potential time and cost savings you expect by automating that business process and the total effort it takes to both automate and maintain that automated business process.

This hybrid approach for automation discovery enabled the CoE to sustain a continuous automation pipeline and to maintain a prioritized automation backlog that was accessible to everyone.

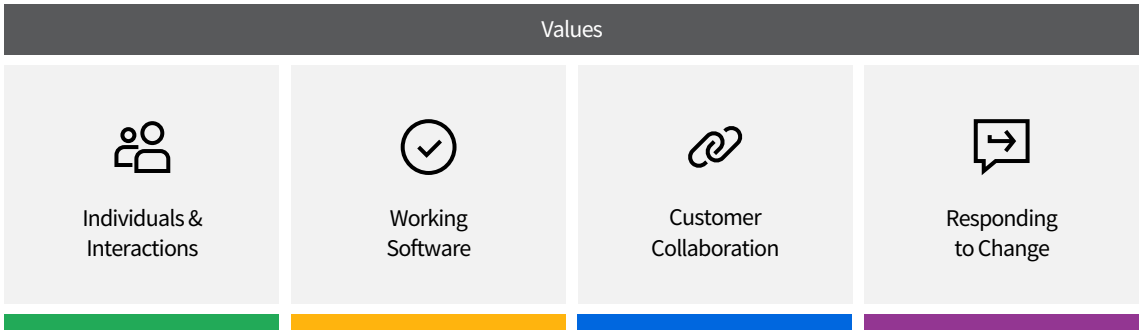
2.3 Delivery

This backlog was the actual source of work for the CoE. The automation delivery happened in two ways. First, it was centrally governed by the RPA CoE. Their automation mandate included simple task automations and complex intra-departmental processes in the beginning as well as complex cross-departmental processes in the further course of this RPA initiative. Secondly, automation delivery was democratized. The CoE built and distributed automations, which the employees then used. Once the employees got used to working with automation, they started to suggest ideas for more automation. Susan calls this the automation flywheel. For technically savvy citizen developers the CoE provided the tools, such as UiPath StudioX, and the required trainings to enable them to build their own automations. The goal of this approach was to connect employees. This implies that the value was not just in the CoE anymore, it was in the interconnectedness of the employees. So, Susan's goal was to develop a platform for RPA which was open and allowed regulated participation.

Note that we have used the term delivery instead of development. The CoE didn't just develop automation, the CoE delivered automation. For that reason, Susan preferred to refer to RPA deliverers instead of RPA developers since that is what they actually did. The CoE had then to decide which governance model to choose for their automation delivery process. Should RPA follow a classical or an agile delivery framework?

2.4 Framework

Classical RPA delivery means dividing the RPA delivery process into distinct phases based on activity. A typical example is the well-known Waterfall-Model. Each phase represents a certain activity (e.g., discover, analyze, design, develop, test, deploy, monitor) and depends on the deliverable of the previous phase. The delivery is non-iterative as well as non-incremental. The flow through these phases is unidirectional. This means that the delivery progress only flows in one direction ('downwards the waterfall') through the individual phases. For the automation of one single business process, this means automating all process components, to full fidelity, and then release the entire automated business process at once. It's a big bang release. For that reason, classical delivery is often called big bang delivery.



The term 'agile' was popularized by the Agile Manifesto in 2001. This manifesto is based on 12 principles and the following 4 core values: (1) Individuals and interactions over processes and tools. (2) Working software over comprehensive documentation. (3) Customer collaboration over contract negotiation. (4) Responding to change over following a plan. This means that agile is not a process but a set of values. So, the term 'Agile RPA' doesn't refer to a certain technique but to a philosophy of RPA delivery. In practice, this delivery philosophy is supported by a set of lightweight process frameworks (e.g., Scrum, Kanban, SAFe, Nexus, LeSS) and operational techniques (e.g., CI/CD) that help RPA teams to put these values into practice. So, there's a difference between doing agile and being agile. Doing agile focuses on processes and techniques. Being agile focuses on behaviors that are guided by the agile principles and values.

Agile delivery is a combination of an iterative and incremental style of delivery. Incremental delivery for the automation of one business process could mean automating some process components, one by one, to full fidelity, release them, and automate other process components in the next releases. Iterative delivery could mean automating all process components, at low fidelity, release them, and then increase the automation fidelity of the process components in the next releases. We have taken the terms incremental and iterative literally. Incremental means 'add onto'. In our example, this translates to 'adding process components' to your automation. Iterative means 'alter'. This translates to 'refining process components' in your automation. In this light, delivering an automated business process in an agile way could mean automating some business process components, one by one, at low fidelity, release them, and then both gradually increase their automation fidelity and automate other process components and in the next releases. Note that there's no magic threshold at which a change must be considered incremental or iterative. We will use these two terms synonymously.

In this thinking, classical delivery means doing one activity (e.g., design, develop, test, deploy) at a time for the entire automated business process while agile delivery means doing all activities for a subset of the entire automated business process at a time. So, the key to agile RPA delivery is to frequently produce so-called working robots (aka robot increments) to enable frequent feedback. A working robot can be understood as a subset of the final automated business process that creates value for the stakeholders. Note that nothing is ever really considered final in agile RPA delivery since you can always evolve automation in terms of functionality, performance, reliability, stability, security, usability (e.g., attended robots), and many other attributes. So, in a sense, the goal of agile RPA delivery is not to be perfect but to be progressively less stupid.

2.5 Transition

Susan favored agile over classical delivery, but the noes had it. The majority of the CoE was reluctant to work in an agile way. Their main reasoning was that the world of RPA delivery is almost perfectly predictable. Susan didn't agree but she knew that working with a reluctant team would be difficult. She knew that agile delivery is fundamentally people-oriented. She understood that imposing an agile way of working on reluctant people is at odds with the entire notion (e.g., values, principles) of agile collaboration. So, the CoE collectively subscribed to classical delivery. This decision was revoked after about eight months of intense automation delivery. During these months, the CoE moved through countless learning cycles and switched from classical to agile delivery. The transition didn't happen suddenly; it happened gradually. Susan didn't treat these two styles of RPA delivery as discrete, binary options. She considered them as fuzzy areas along a long and multi-dimensional spectrum ranging from more predictive, process-oriented approaches to more adaptive, people-oriented approaches. So, the CoE didn't jump but rather transitioned from one end of the spectrum below to another. We will discuss the dimensions of this spectrum in the subsequent chapters.

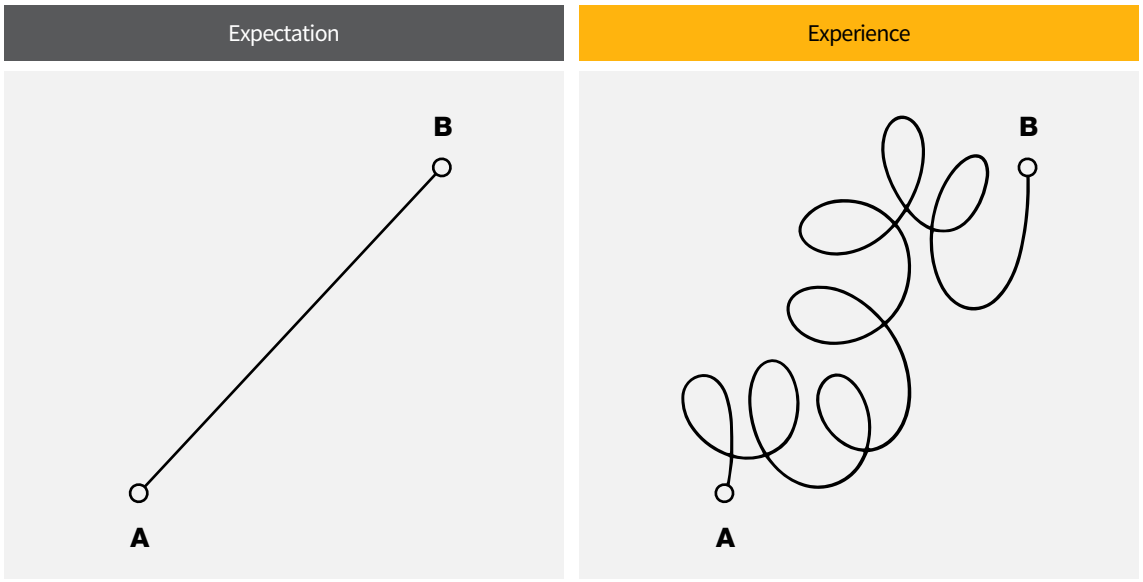
Classical Approach	Hybrid Approach	Agile Approach
Revolutionary		Evolutionary
Sequential		Iterative
Plan-Driven		Value-Driven
Predictive		Adaptive
Process-Oriented		People-Oriented
Resists Change		Welcomes Change

3 Approach

This chapter outlines the main reasons for this transition and discusses the core practices of agile RPA delivery.

3.1 Predictability

The CoE didn't experience what they were expecting. This gap between expectation and reality was mainly due to the high level of unpredictability involved in the delivery process.



In rough terms, the early phases (e.g., discover, analyze, design) of the classical delivery process were as follows. First, the discovery phase. For each crowd-sourced automation opportunity, the business stakeholders (e.g., process owners, subject matter experts) provided a high-level process design document (PDD). You can think of it as a rough, initial process analysis (IPA) performed by the business.

Secondly, the analysis phase. In this phase, the business stakeholders and the CoE (e.g., architects, developers) assessed whether the suggested business process is suitable for automation. You can think of it as a feasibility analysis. In case the business process passed this feasibility analysis, the business stakeholders refined the PDD. The PDD then turned into a detailed documentation that included step-by-step process instructions, process statistics (e.g., transaction volume, execution schedules), in/out automation scope, business and application exceptions, user roles and permissions (e.g., authentication, authorization), system security and privacy threat assessments, SLAs, system dependencies, and business benefits (e.g., time, cost savings). This list could go on and on. You get the point. The PDD described the 'as-is' process in great detail.

Thirdly, the design phase. Based on the PDD the CoE created a solution design document (SDD) that described the 'to-be' process by detailing the implementation approach (e.g., architectural design, design principles, exception handling, reusable components). This laid the foundation for the development phase.

The business wasn't really involved anymore after the analysis phase and until the automation was deployed to production after final acceptance testing. The CoE followed classical delivery in its purest form in terms of 'go away for a while and we will tell you when it's done'. Documentation was valued over communication. Note that in classical delivery you don't have close collaboration, you rather have handovers and sign-offs. The initial

assumption of the CoE was that the PDDs are sufficient to guide the entire delivery process. This proved false. During development, the PDDs were found to be incomplete, inaccurate, ambiguous, and sometimes even inconsistent. For example, cases that were documented in the PDDs never occurred, cases that occurred weren't documented, or cases that were documented differed from reality. This turned the delivery process from a predictable process into an unpredictable process.

Remember, classical methodologies compared to agile methodologies are rather plan-oriented than people-oriented. These methodologies impose a disciplined process upon RPA delivery to make it more predictable by having a strong emphasis on planning. The problem was that even comprehensive upfront planning couldn't make RPA delivery predictable. The plan of the CoE constantly collapsed like a house of cards. The CoE realized that they were using a predictive methodology in an unpredictable situation. This obviously doesn't add up. So, the CoE learned that they should always plan for the fact that no plan ever goes according to plan (Simon Sinek). This was the first gentle push into the direction of agile delivery.

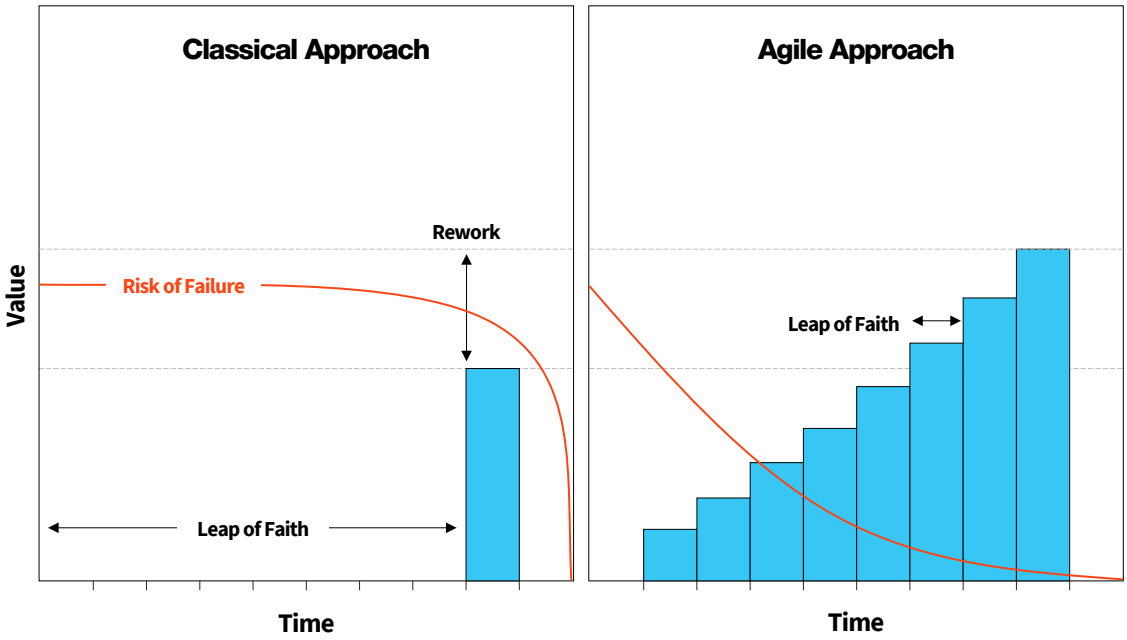
3.2 Transparency

So, RPA delivery turned out to be anything but predictable. For the CoE, this meant to let the whole notion of predictability go. This wasn't easy to digest. Predictability is a desired property. It gives a sense of security, safety, and control. It took the CoE some time to accept this bitter truth. For some time, the CoE bit the bullet. They kept pretending that they could follow a predictable delivery process when, in fact, they couldn't.

As the saying goes, rule number one of predicting is that predictions are always wrong. The perfect all-inclusive formula for prediction doesn’t seem to exist yet. Now, the more unpredictable the world is the more you rely on predictions. So, the CoE started to predict what the business is going to need. In other words, assumptions entered the game of RPA delivery. The problem is that assumptions are the termites of RPA delivery.

Here's an example. Strictly following the logic of classical delivery, the CoE automated the business process in its entirety. Several weeks of baking one assumption after another into the automation passed by. Finally, the automation was delivered to the business stakeholders for acceptance testing. The refrain Susan then heard after almost every delivery was two-sided. The business yelled 'where do all these bugs come from!?'. The CoE complained 'why are the requirements for the automation always changing!?'.

This doesn't add up. Susan learned that handing over documentation (e.g., process design documents) without fostering transparency through close collaboration is a brilliant way to misinterpret what other people mean. Remember, walking on water and developing automation from a specification (e.g., PDD) are easy if both are frozen (Edward Berard, 1983). So, mastering RPA means mastering continuous change. The core of the problem was that classical RPA delivery tends to try to plan out a large part of RPA delivery in great detail for a long period of time. This plan-driven approach doesn't really ask for transparency in the delivery process. Transparency is operating in such a way that it is easy for others to see what actions are performed. This works well until things change. The problem is that change is inevitable and that the nature of classical delivery methodologies is to resist change. This resistance to change is baked into the delivery process. This prevented the CoE to mitigate the risk of failure continuously which is depicted in the figure below.



The inability to mitigate the risk of failure continuously often caused a lot of rework at the time of delivery. This rework then plagued the PRA initiative by delays and cost overruns which, in turn, prevented the CoE to scale RPA appropriately to turn the enterprise into a fully automated enterprise. Susan realized that the big bang only worked once: at the beginning of time. This was another gentle push into the direction of agile delivery.

In contrast to classical delivery, agile delivery welcomes change. For that reason, Susan decided to switch from a plan-driven and process-oriented approach to an iterative and adaptive approach. This approach is depicted on the right-hand side of the figure above. Hence, Susan started to value adaptability over predictability. This implies that she decided to let go of predictability. However, letting go of predictability doesn't mean that you have to revert to uncontrollable chaos. You just need a framework (e.g., Scrum, Kanban) that gives you control over unpredictability. That is what adaptability in agile delivery is about (Martin Fowler, 2005).

3.3 Adaptability

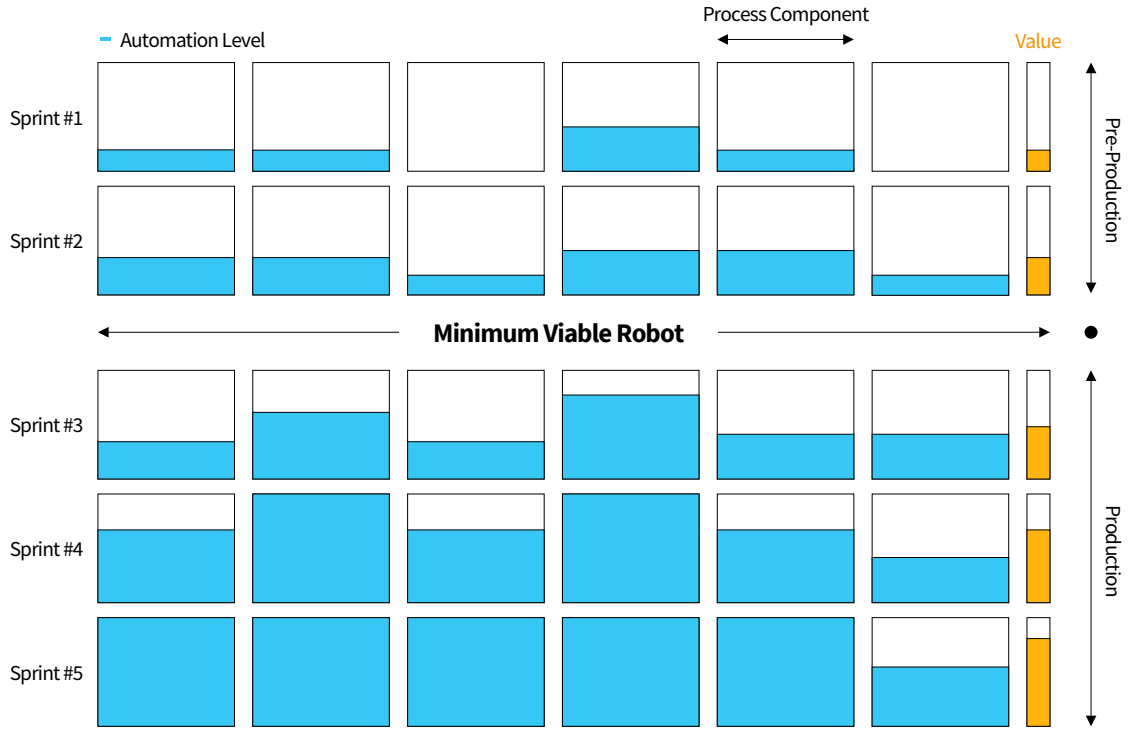
Susan fostered adaptability by first adopting a process framework that supports the iterative work of the CoE. The winner was Scrum with Kanban, a combination of Scrum and Kanban. It would be outside the scope of this paper to go into too much detail about each of those. We restrict ourselves to their main characteristics. The main reason for Susan's choice was that transparency, inspection, and adaptation are at the core of Scrum. Scrum is founded on empiricism and lean thinking. Empiricism asserts that knowledge comes from experience and making decisions based on what is observed. Lean thinking reduces waste and focuses on the essentials. Scrum employs an iterative, incremental approach to optimize predictability and control risk. Based on what we have learned in the previous chapters, that's exactly what Susan was looking for. Susan considered Scrum as a mechanism to control a mostly unpredictable delivery process.

The Scrum Guide states that the 'Scrum framework is purposefully incomplete, only defining the parts required to implement Scrum theory. Scrum is built upon by the collective intelligence of the people using it. Rather than provide people with detailed instructions, the rules of Scrum guide their relationships and interactions'. So, Susan realized that perhaps the most important thing she should do is finding someone more experienced in agile delivery to help the CoE to learn. She knew that doing anything new will inevitably lead to mistakes. Hence, she brought an experienced scrum master on board. This person has already made loads of mistakes and so

was more than helpful to avoid that the CoE is making these mistakes themselves. The scrum master became a central role who guided the CoE through the transition from classical to agile delivery and beyond.

Kanban is a strategy for optimizing the delivery flow. It's a set of practices scrum teams would need to achieve steadier, healthier, and more sustainable flow. Ergo, Kanban complements Scrum. According to Yuval Yeret (2018), the minimal simplest set of Kanban practices is visualizing the workflow, limiting the work-in-progress, actively managing work items in progress, inspecting and adapting their definition of 'workflow'. For more information, please refer to the Kanban Guide for Scrum Teams. In short, Scrum focuses on the management aspects of RPA delivery. Kanban focuses on optimizing the RPA delivery flow.

Susan took on the role of the product owner. Together with the scrum master the RPA developers, RPA solution architects, and RPA infrastructure engineers the scrum team was formed. The main stakeholders of the scrum team were people from the business (e.g., process owners, subject matter experts). The business stakeholders were called RPA business analysts during the time (e.g., sprint) of collaboration.



Susan divided the RPA delivery process into two-week iterations (aka sprints). After each sprint, a working robot was delivered and feedback from the business stakeholders was collected during the sprint review. So, the CoE iteratively and incrementally worked towards robots that provided more and more value to the business stakeholders. This approach is depicted in the figure above.

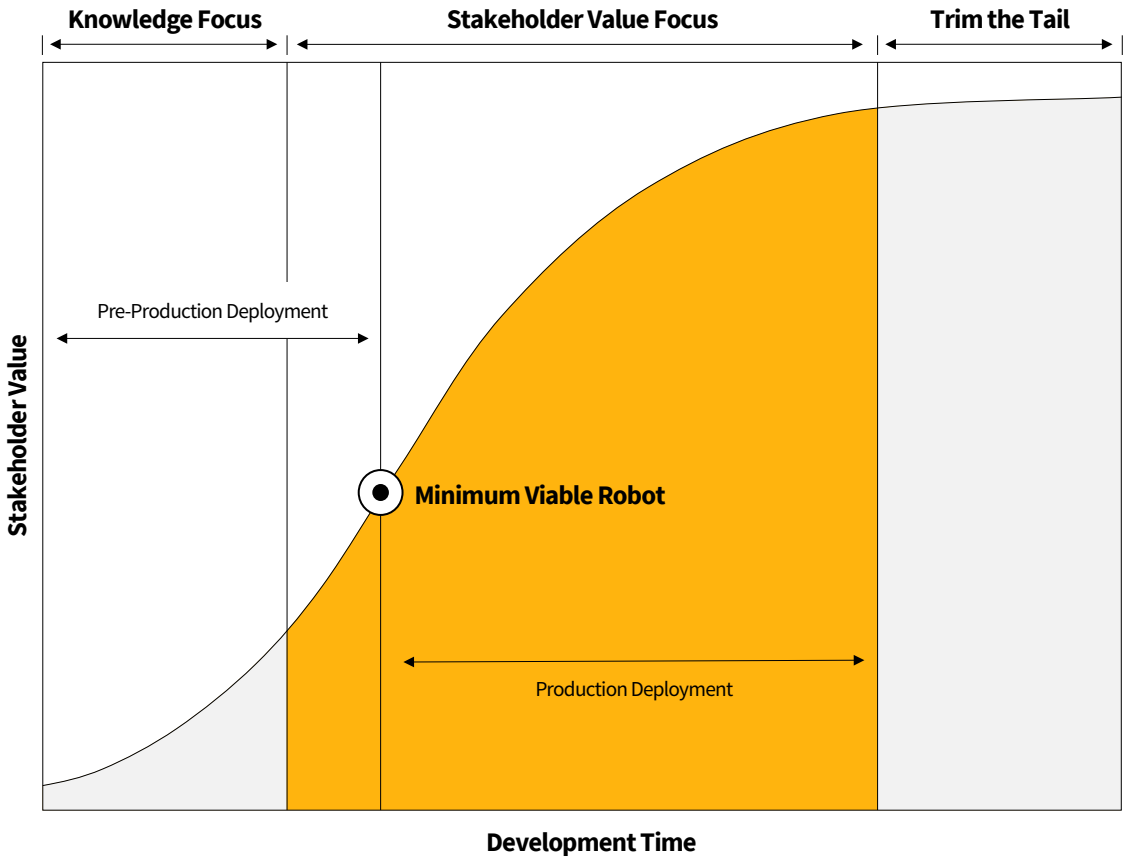
On the one hand, this agile (i.e., iterative, incremental) approach enabled frequent feedback in a structured way. On the other hand, frequent feedback was the key mechanism to get control over the unpredictable. In other words, the CoE opened up and invited the business stakeholders into their world. This enabled the CoE to continuously reduce the risk of failure and the amount of waste (e.g., rework).

The robot was enhanced step-by-step based on the feedback of the business stakeholders. The goal was to provide a minimum viable robot as soon as possible to the business stakeholders. There's no universal formula to evaluate when a robot turns into a minimum viable robot. This was agreed upon between the business

stakeholders and the CoE during the sprint reviews. A minimum viable robot was a robot that noticeably decreased the workload of the business. Once the robot turned into a minimum viable robot, the robot moved from its pre-production to its production era. From then on, the robot was continuously deployed to production after each sprint. The robot entered this stage after about 3 weeks (1.5 sprints) on average. The goal was that the robot creates value for the business earliest possible. The CoE then gradually evolved the robot to create more and more value for the business after each sprint. This turned the robot bit-by-bit from a good virtual worker to a great virtual worker. So, the CoE followed an evolutionary rather than a revolutionary approach.

3.4 Segmentation

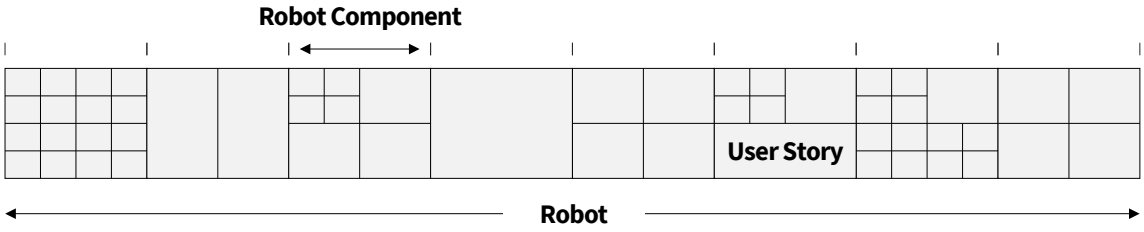
Note that this evolutionary approach wasn't just applied to the technical implementation of the robot. It was also applied to the development of the PDD and SDD. Instead of spending several weeks or even months working out every single bit of 'what could happen' and 'what might be relevant', these documents evolved too in an iterative and incremental way. This doesn't mean that there was no upfront planning at all. The planning just focused more on the bigger picture than on every tiny little detail. Remember, agile delivery emphasizes responding to change over following a plan. So, plans were written but updated regularly. The plans were continuously adapted based on what has been learned about the business process and its related automation. So, do not confuse adaptive planning with no planning. Susan understood that failing to plan is planning to fail.



This implies that the early phases of development focused on knowledge acquisition. The goal was to mitigate both business risk and technical risk. For example, business risk was mitigated by learning about the underlying business logic of the business process in close collaboration with the business stakeholders. The technical risk was mitigated by exploring architectural design decisions for different implementation approaches. That was

not too exciting for the business stakeholders but still valuable for the CoE since risk was reduced. So, this era was dominated by so-called spikes or chores to study business logic, research implementation concepts, create prototypes, or perform proof-of-concepts. These activities didn't create value for the business stakeholders in terms of time/cost savings, but they created value for the CoE in terms of acquiring knowledge. So, knowledge value was valued over stakeholder value. This is depicted in the figure above.

As uncertainty was reduced, the CoE then focused on creating stakeholder value as fast as possible by applying a concept Susan called business process slicing. Susan considered an automated business process as an epic that was described by its related PDD and SDD. The idea was to split an epic into smaller and more manageable chunks called user stories. So, an epic can be understood as a rubber band around a group of user stories that share a common goal. In contrast to a user story, an epic is an item that cannot be completed within a single sprint. So, simple task automations were usually considered user stories while more complex automations of cross-departmental and intra-departmental processes were considered epics.



From a business perspective, a business process (epic) was broken into a set of individual process components (user stories). From a technical perspective, this means that a robot (epic) was broken into a set of individual robot components (user stories). Ergo, one robot component reflected one process component (e.g., monitor folder, read file, classify file, extract data, interpret data, enter data). There's almost an infinite number of ways you can slice a robot. Susan's main mantra was to make the robot components not too big nor too small.

The robot components were treated and developed as mutually independent from each other. This turned the robot components into reusable automations that were maintained at one central location and used in many other robots. So, segmentation led to simplification. The robot development was like playing Lego.

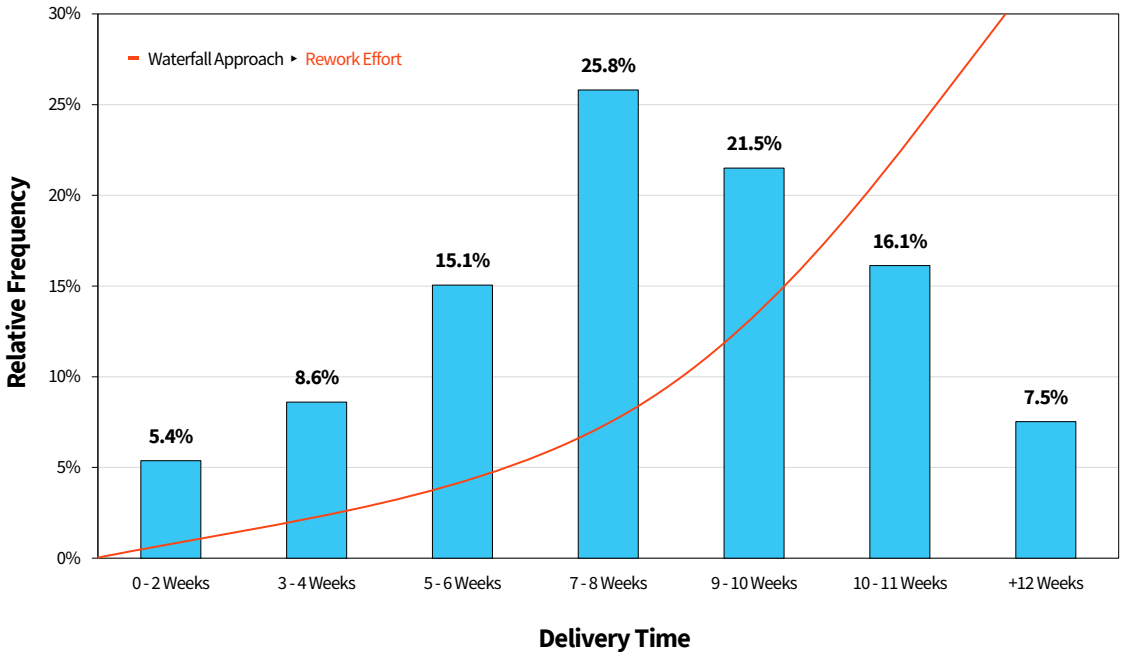
The robot components were triggered by certain events. The nature of these events ranged from user actions (e.g., keyboard taps, button clicks), service events (e.g., start, pause, resume) to database events, email events, and file system events (e.g., file added, modified, deleted). So, each robot component was a set of automated actions in response to a specific event. The output of one robot component was the input for another. So, the boundaries between the robot components were given by well-defined events.

The robot components (user stories) were further split into two or more user stories. This concept is called user story splitting. This was only done in case each resulting user story preserved a measurable stakeholder value. Think of a user story that is about extracting data from an invoice (e.g., order details, payment terms). An invoice can arrive in different formats (e.g., PDF, MS Word, image). In this case, you can split the user story into a set of user stories each dealing with a different type of format. You can go further and split these user stories according to whether the invoice contains structured data (e.g., machine-written invoice) or unstructured data (e.g., hand-written invoice). This is an oversimplified example that can already be addressed by the means of AI/ML (e.g., UiPath AI Center). The bottom line is that these scenarios are not equally important to the business. There are scenarios that matter more to the business than others. For example, in Susan's case, about 92% of the invoices the purchasing department received were machine-written PDFs. In this case, it wouldn't be a great idea to focus your initial automation efforts on all possible types of invoice formats and data structures.

This is the type of considerations Susan applied while slicing a robot into its components. She understood that the 80/20 principle doesn't stop for RPA. She knew that little automation effort can lead to high stakeholder value. This proved right in the vast majority of cases. Following this principle provided three main benefits. First, the CoE achieved maximum stakeholder value with minimum effort earliest possible. Secondly, it prevented the CoE from overengineering the robot right from the beginning. Thirdly, it empowered the CoE to know when to stop to avoid investing a lot of additional automation effort for little to no stakeholder value (trim-the-tail). In other words, knowing when to stop avoids that you sleepwalk from the honeymoon period (high value, low effort) into the period of diminishing returns (low value, high effort). Remember, there's nothing quite so useless as doing with great efficiency something that must not necessarily be done at all (Peter Drucker).

3.5 Complexity

Here's why the shift to agile delivery paid off in the long run. Susan tracked the total delivery time from discovery to the deployment of automations to production. After about 18 months the stats looked like this.



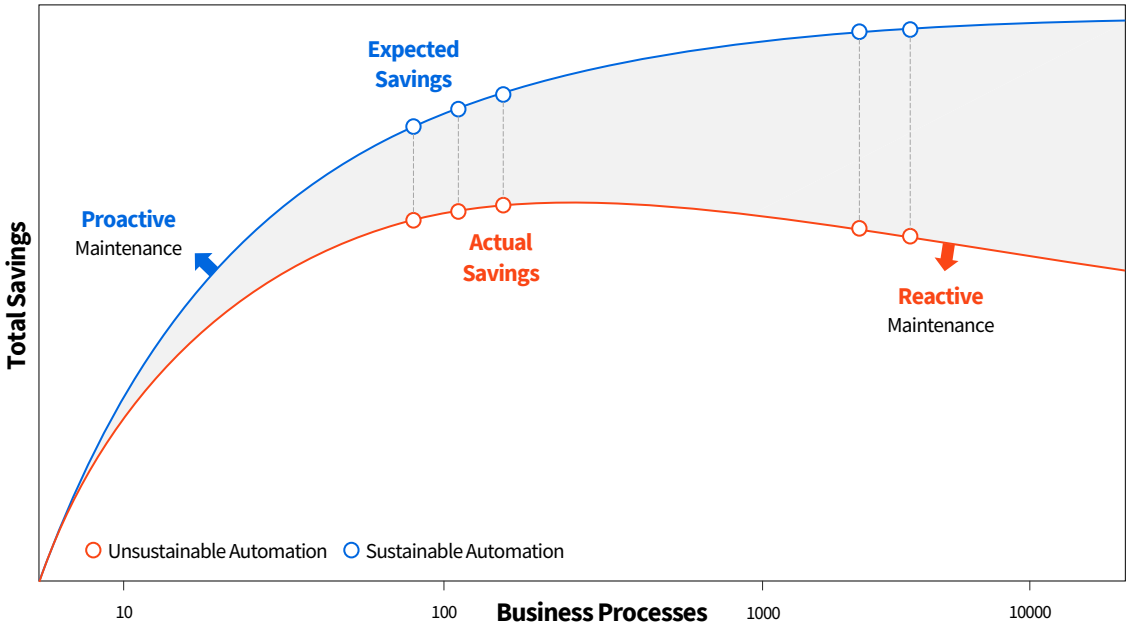
The distribution was slightly left-skewed. Most automations were clustered around its right tail. About 86% of the automation took more than 5 weeks of implementation effort, and about 71% of the automation took more than 7 weeks. This distribution evolved over time. In the beginning, Susan's CoE primarily focused on providing simple task automations to the employees to fuel the democratization of automation. The citizen developers then became more and more familiar with RPA and started to create their own automations. So, the CoE started to focus more and more on the more complex cross-departmental and intra-departmental processes.

On the one hand, classical delivery worked well for orchestrating the delivery of simple task automations. These task automations were delivered in a few days up to two weeks. Here, the outcome of automation delivery was relatively easy to predict due to the relatively low complexity of these automations. On the other hand, classical delivery was found to be unproductive (e.g., inefficient, ineffective) for the automation of more complex cross-departmental and intra-departmental processes. These automations were delivered in several weeks up to a few months. Here, the outcome of automation delivery was relatively hard to predict due to the relatively high complexity of these automations. The rule of thumb was: The longer you develop behind closed doors, the

more rework you can expect. The bottom line is that w/o shifting from classical to agile delivery, the CoE would have followed a delivery methodology that wouldn't have been productive for most of their future work. Well, afterward one is always wiser. Hindsight is easier than foresight.

3.6 Controllability

RPA is not a technology that you put in place and forget. RPA is a prime example of 'simple but not easy'. Scaling RPA means applying a proactive approach to maintenance. Change in an enterprise is a constant. So, delivering automation in an enterprise context isn't a one-time event; automation requires maintenance.

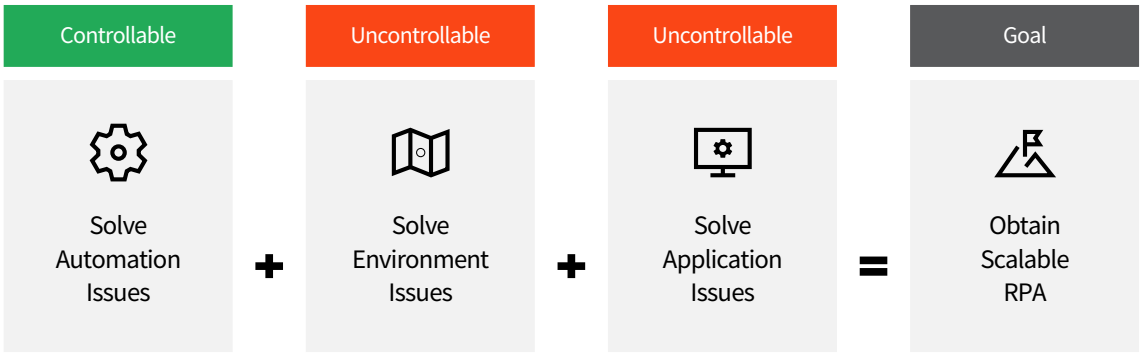


Susan learned this the hard way. Again, think about the value of an automated business process. Its value is given by the relation between the savings you create by automating that business process and the total effort it takes to both automate that business process and to maintain that automated business process. In case you automate your business processes in a prioritized way, that is, you start automating high-value business processes first, then the total savings (e.g., FTEs) you can expect through automation would evolve according to the blue curve. This is how it looks like in theory. However, in practice, there could be a significant gap between the expected and the actual savings. Susan learned that the actual savings are highly likely to evolve according to the red curve. This is mainly caused by the maintenance effort that is related to the automation of business processes. Note that this is not a special issue of RPA; it's a central issue of automation. Any kind of automation needs maintenance, and since RPA is primarily automation, RPA needs maintenance too. So, Susan realized that proactive maintenance is what they need to do. Reactive maintenance is what they need to avoid. She learned that she needs to pay attention to maintenance right from the beginning to avoid that the expected savings of RPA start to erode due to the constantly growing effort of maintaining automation.

According to Deloitte (2018), 78% of organizations that have implemented RPA are expecting to significantly increase their investment in the next 3 years. However, only 3% of these have been able to scale RPA beyond the initial pilot. The maintenance effort related to automation is a decisive factor that prevents companies to scale RPA. So, maintenance can be understood as the silent killer of RPA.

Maintenance is an umbrella term. It encapsulates a variety of problems related to RPA. High-maintenance RPA is mainly caused by fragile RPA. Fragile RPA means unstable RPA that causes bots to break in production. Susan identified three main reasons why robots break in production and so cause constant maintenance overhead.

The first group of problems is centered around automation issues. Robots are breaking in production due to synchronization issues between the robots and the software apps, due to poor object recognition, or due to no or missing recovery, exception, or error handling. The second group of problems is centered around application issues. These issues result from changes made to software applications that are consumed by robots. These changes are made by software development teams and include technical changes (e.g., changing UI controls, API definitions) or changes in the business logic of the software apps. The third group of problems is centered around environment issues. These changes are made by IT Ops teams and include performance issues, issues resulting from dependencies to third-party services, data-related issues, issues caused by system changes (e.g., antivirus updates, security patches, browser updates), or issues caused by customizations made to packaged CRM/ERP applications (e.g., SAP, ServiceNow, Salesforce) to account for regulatory changes.



So, Susan learned that unstable automation mainly resulted from the fact that the CoE all-too-often had a hard time incorporating application and environment changes into their RPA in a timely manner, that is before these changes were deployed to production. In other words, the applications and environments changed but the CoE didn't get the memo. This is what Susan calls the 'broken bot syndrome'. The bottom line is that the automation issues were under the control of the CoE but the application and environment issues weren't.

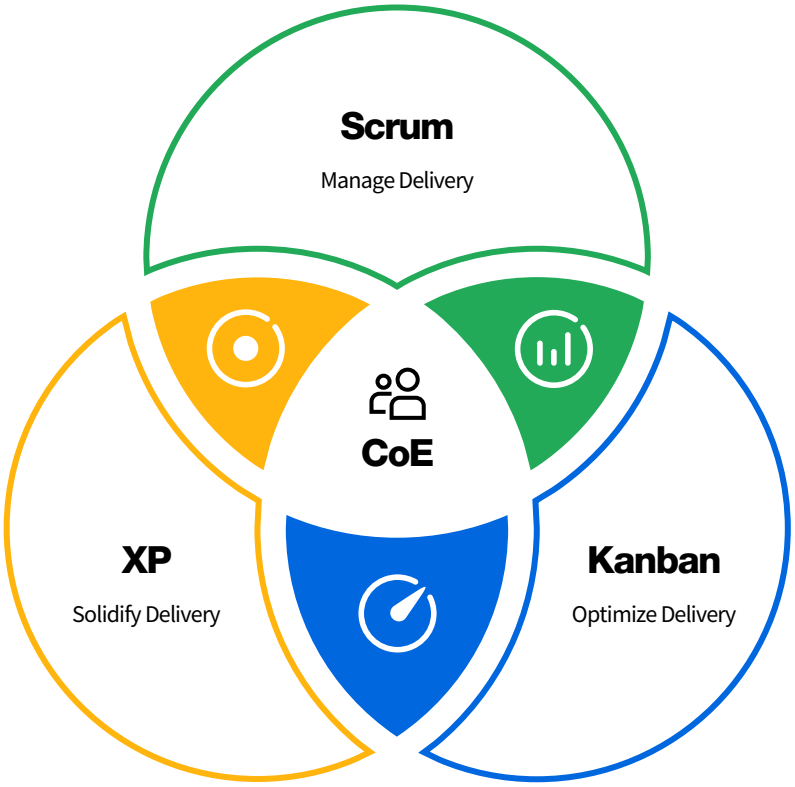
In order to control the uncontrollable, Susan build bridges between the CoE, software development teams, and IT Ops teams. These bridges were technical bridges that were realized in the form of automated test cases. For each robot, a set of automated test cases was created by using tools such as the UiPath Test Suite. Susan called these test cases contract tests. You can think of a contract as a binding agreement between some provider and some consumer. In Susan's case, the providers were software development and IT Ops teams, the consumer was the RPA CoE. In other words, the robots developed by the CoE consumed the services (e.g., environments, applications) provided by software development and IT Ops.

Long story short. These contract tests enabled these three parties to evaluate in an automated way whether a certain change to an environment or to a software application causes the robots to break. So, these contract tests weren't just part of the CI/CD pipeline of the RPA CoE, they were also part of the CI/CD pipeline of software development and IT Ops. This made the robots stakeholders for software development and IT Ops and avoided that application and environment changes pile up, hit the robots in production, and result in chaotic bug fixing. The consequent benefits were manifold. For example, the robot downtime due to the above-mentioned issues was reduced by a staggering 88%. The increase in the robot's uptime in turn increased stakeholder satisfaction. This boost in robot quality not only enabled the CoE to sustain but to increase the RPA throughput in the long run because the CoE didn't get drown in maintenance efforts anymore. Hence, the development of automated test cases for each robot was an integral part of the team's definition of done (DoD).

So, continuous testing significantly influenced the runtime performance of the robots. This directly translated to time and cost savings for both the CoE and its stakeholders. Remember, robots only create value in terms of saving time and costs if you keep them running. The cost of doing nothing in terms of no testing turned out to be orders of magnitudes higher than the costs that were constantly invested in what Susan calls Continuous RPA Testing. In other words, Susan learned that the best time to repair the roof is when the sun is shining.

3.7 Engineering

Scrum does not include engineering practices. According to Jeff Sutherland (2013) this was done on purpose. 'Scrum was designed to get a team started in two or three days, whereas engineering practices often take many months to implement. So, Scrum is easier to adopt when specific engineering practices aren't mandated. This leaves the question of when and whether to implement certain engineering practices up to each team. Scrum creators Jeff Sutherland and Ken Schwaber recommend that scrum teams get started immediately and create a list of impediments and a process improvement plan. As engineering practices are identified as impediments, teams should look to practices from extreme programming (XP) as a way to improve. The best teams run Scrum supplemented with XP practices. Scrum helps XP to scale, and XP helps Scrum to work well'. So, even though Scrum doesn't say anything about engineering practices, it suggests that you should be doing XP.



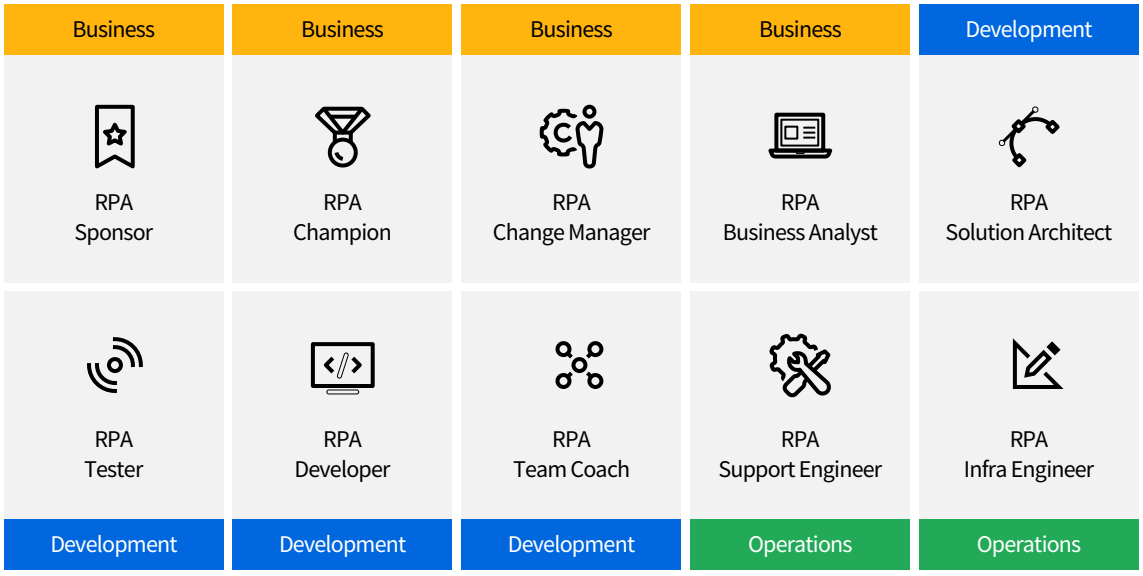
The CoE decided to supplement their management practice (Scrum with Kanban) with engineering practices from the world of extreme programming (XP). The objective was to improve the quality of automation with the XP practices. These practices ranged from pair programming, test-driven development, coding standards, design principles, code reviews to refactoring, collective code ownership, and continuous integration. It would be outside the scope of this paper to discuss all these practices. A brief but great overview of XP is given by Ron Jeffries (2011). For those who want to dive deeper into XP, Susan suggests the book by Kent Beck (2004). For example, in terms of coding standards and design principles, the CoE compiled a quality guideline based on the

UiPath Automation Best Practices and the UiPath Design Best Practices guides to continuously bake quality into their robotic process automation. This addressed the automation issues discussed in the previous chapter.

The bottom line is that Susan recognized that there are numerous parallels between RPA development and software development. Note that we are referring to development, not delivery. Susan even conjectured that RPA development is a special case of software development. Simply put, RPA development is inside software development. Susan argued that both an application (e.g., web app, mobile app) and an automated business process are software. It doesn't matter which tools (e.g., UiPath Studio, Visual Studio) you are using to develop software. Software remains software. Think of it this way: You can use a flipchart for your conference talk, or you can use a PowerPoint presentation for your conference talk. In both cases, you are delivering a conference talk. Hence, Susan concluded 'why not adopt engineering practices that proved valuable to improve software quickly and continuously and apply them to RPA'. If it works for software development, it will work for RPA development. Susan's success proved her right. The CoE ended up applying a framework for agile RPA delivery that was a good mix of Scrum, Kanban, and XP. Susan called it XrumBan, the 'holy trinity' of agile RPA.

3.8 Scalability

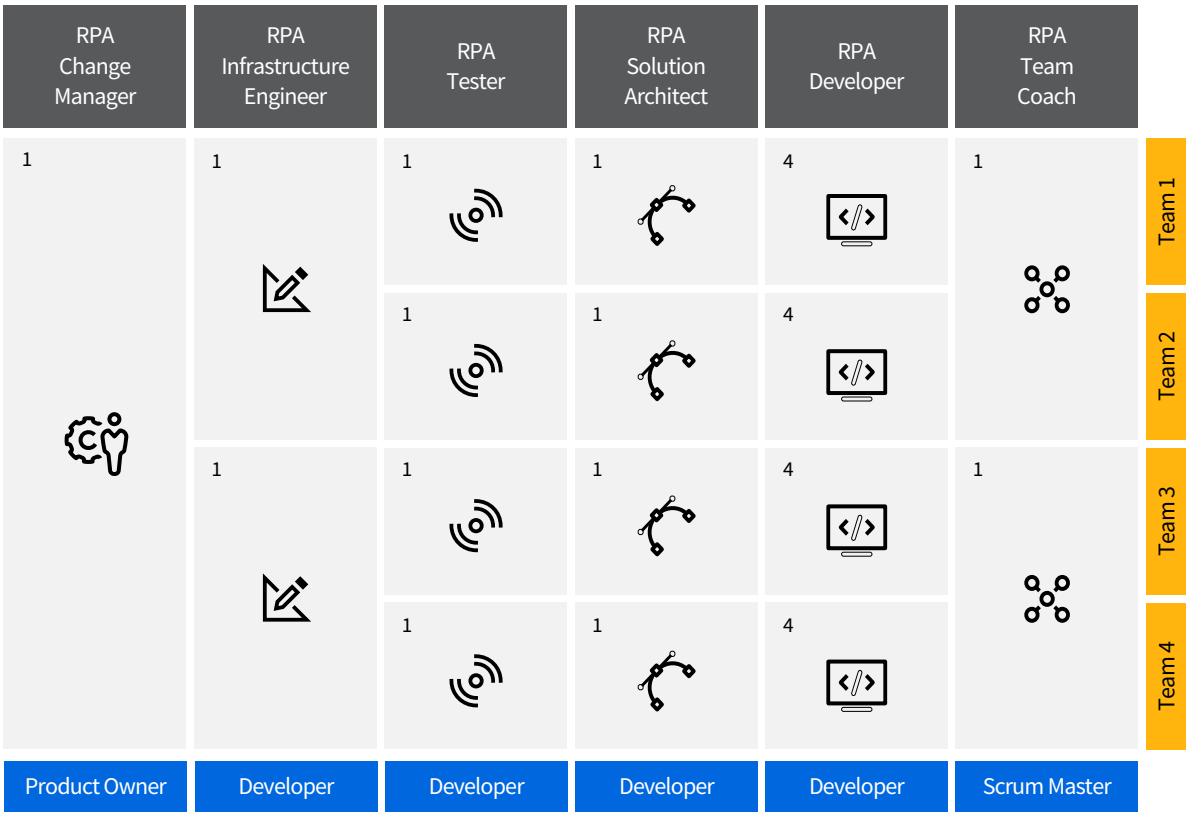
Fast forward. The RPA initiative was a great success. The employees became more and more excited about RPA. The demand for RPA started to skyrocket. So, the CoE not only had to grow in terms of people but also had to adjust its delivery processes to provide the necessary supply for this increasing demand. The CoE evolved from a single team in 2018 to a little department in 2020.



First, let's talk roles. Susan categorized the roles by business, development, and operations according to the portmanteau BizDevOps. Susan took on the role of the so-called RPA change manager. In this role, her duties were twofold. First, Susan was responsible for securing an easy adoption of RPA in the organization, overseeing the onboarding of stakeholders from business, IT Ops, and software development, and evangelizing RPA with a clear, open, and inspiring communication and advocacy plan. She was tasked with keeping the stakeholders well informed about the RPA adoption and comfortably tuned to the changes taking place. Internally, Susan was called the chief RPA evangelist. She had the rare gift to locate and turn people (e.g., department leaders) into enthusiastic PRA champions. These RPA champions helped to spread the word about RPA and encouraged other department leaders to follow suit. This ensured a healthy automation pipeline. Susan also identified RPA sponsors on the executive level and kept them actively engaged. The RPA sponsors were key to establish the

RPA initiative as an enterprise-wide strategic priority, underwrite corporate resources, gain the cooperation of peers, clear obstacles, and sustain the change due to RPA in the organization. We will elaborate more on this evangelism and collaboration aspect in chapter 3.11. Secondly, Susan was the owner of the RPA initiative. She owned the product backlog. Internally, the product backlog was sometimes called robot backlog. The product backlog represented the master list of all automation opportunities (potential robots). Susan was responsible and accountable for prioritizing the product backlog on the level of epics, not individual user stories. In Scrum terminology, Susan was the product owner. She set the strategic direction (e.g., vision, strategy) for the CoE.

The CoE contained 4 scrum teams. The overall team structure is depicted in the figure below. Each scrum team consisted of one RPA solution architect, 4 RPA developers, one RPA infrastructure engineer, one RPA tester, and one RPA team coach. Susan served as the product owner for those 4 scrum teams. The RPA infrastructure engineers and the RPA team coaches were shared resources. One resource was assigned to two scrum teams.



Scrum only talks about three roles: product owners, scrum masters, developers. So, the RPA developers, RPA infrastructure engineers, RPA testers, and RPA solution architects are developers in Scrum. The RPA team coaches are the scrum masters and the RPA change manager represents the product owner. The relations between these role names might be confusing at the first glance. Susan just kept established role names in the RPA industry (e.g., RPA change manager, RPA solution architect) and introduced roles (e.g., RPA team coach) that were independent of the agile delivery framework to enable people outside the CoE (e.g., executive management) to understand the team setup and dynamics without requiring any background knowledge of agile frameworks (e.g., Scrum, Kanban, XP). The mapping between the roles is shown in the figure above.

The RPA team coaches were accountable for the effectiveness of the scrum teams. They helped the scrum teams to focus on developing robot increments that meet the definition of done (DoD), removed impediments to the team's progress, ensured that the events (e.g., daily scrums, sprint planning) are productive and kept within the timebox, and coached the scrum team members in self-management and cross-functionality.

The duties of the RPA solution architects were twofold. First, the technical perspective. The RPA solution architects were seasoned professionals with profound infrastructure knowledge (e.g., servers, storage, network). They defined the architecture of the RPA solution (e.g., SDD), guided and assisted RPA developers in the implementation, selected appropriate tools, and aligned RPA developers around engineering practices and design principles that were shared across the scrum teams. Secondly, the business perspective. The RPA solution architects were also the proxies of the product owner (Susan) during the development phase. In each sprint, sprint backlogs were distilled from the product backlog. In doing so, the RPA solution architects bridged the gap between business and development. They split epics into user stories as outlined in section 3.4, continuously refined the process design documents in close collaboration with the RPA business analysts, prioritized and communicated sprint backlog items, and actively guided the scrum team in their automation journey by ensuring that the sprint backlog items are transparent, visible, and understood. In summary, Susan as the product owner provided strategic guidance to the scrum teams, the RPA solution architects provided operational guidance to their scrum teams.

The RPA business analysts (e.g., process owners, subject matter experts) created and maintained the process design documents (PDDs) in close collaboration with the RPA solution architects and participated in testing.

The RPA developers and RPA testers were primarily responsible for the development and testing of automation, respectively, and continuously challenged the overall design of the RPA solution architect. Even though testing was considered a team responsibility, Susan hired testing specialists to have someone who focuses on testing. She knew that if testing is no one's focus, the likelihood of it being someone's competence will drop to near zero (Michael Bolton). So, having a testing specialist on the team doesn't mean to restrict the responsibility of testing to that single person (Trish Khoo, 2013). Ergo, the RPA testers weren't solely responsible for testing. They rather took the responsibility to actively involve, guide, and advise their teammates in testing by stimulating them to continuously think critically about the automation they envision, design, build, and ship. Remember, diversity makes testing powerful. Great (testing) teams are teams of diverse talents cooperating.

The RPA infrastructure engineers were mainly in charge of the configuration, deployment, orchestration (e.g., monitoring) and maintenance (e.g., robot performance improvements, troubleshooting) of the infrastructure (e.g., servers, databases, applications, virtual machines) for production as well as preproduction environments (e.g., development, testing). These roles collaborated with the RPA solution architects on the RPA architecture. They were formally managed by IT Ops but dedicated to the RPA initiative to ensure that the RPA solution meets enterprise standards. This not only bridged the gap but also strengthened the link between the CoE and IT Ops.

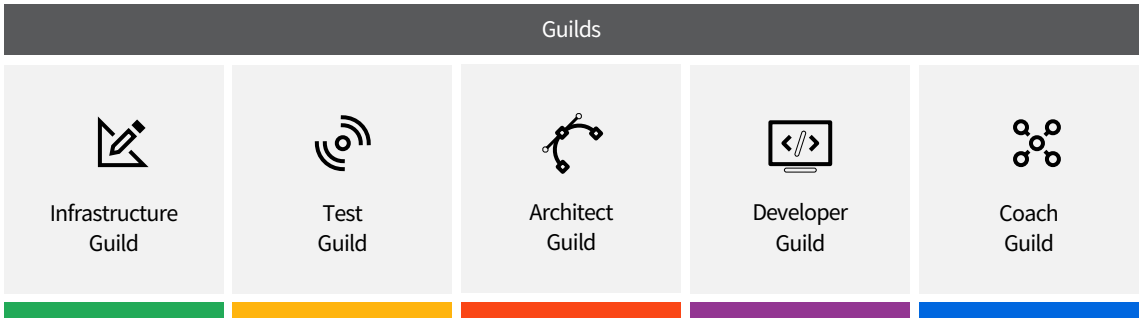
The RPA support engineers were the first line of assistance for the employees that were affected by the robots in production. Simply put, these roles were the first point of call where employees could reach out with a snag. These roles were primarily responsible to monitor, triage, troubleshoot, debug, and report defects related to the robots in the production environment. They weren't part of the scrum teams. According to Susan, the RPA support engineers and the scrum teams were loosely coupled but tightly aligned. The RPA support engineers were part of the support team in IT Ops but were dedicated to the RPA initiative. Ergo, the CoE leveraged the existing support architecture. Five support engineers in IT Ops were trained to become RPA support engineers.

In summary, apart from the RPA sponsors, RPA champions, and RPA business analysts, the RPA CoE scaled from a single team of 6 people to 34 people within about 2.5 years from 2018 to 2020. Remember, the ultimate goal was and still is to turn the enterprise into a fully automated enterprise. Big goals need big actions. Big actions need big energy. Big energy (usually) needs big people support. This asks for big investments. The good news is that these big investments were actually little investments since RPA reaped significant ROI. Simply put, the dollars that were put into scaling RPA created considerably more dollars coming out.

3.9 Governance

Each scrum team was assigned to one or more departments (e.g., marketing, sales, legal, purchasing, finance, human resources) and their respective subdivisions. This doesn't mean that a scrum team was exclusively responsible and accountable for automating business processes for their assigned departments. The list of departments that were assigned to a certain scrum team just reflected the core competencies of that scrum team. So, whenever task automations or intra-departmental processes had to be automated for a specific department, then the assigned scrum team was the preferred team but not the mandatory team to develop and deliver the automation.

So, the CoE followed a governance model that was a good mix between a centralized and federated model. The scrum teams collaborated closely on delivering automations of cross-departmental processes. It's important to note that all 4 scrum teams worked off a single product backlog. The product backlog was accessible for everyone in the organization. Transparency about the company-wide RPA priorities was on top of Susan's agenda. We will outline how these teams orchestrated their daily work in chapter 3.10.



From the team setup outlined above, so-called guilds emerged. A guild is a community of practice. It's a group of people with similar skills and interests who share knowledge, make joint decisions, solve problems together, and improve practices (Etienne Wenger, 2002). For example, the RPA developers came together regularly (e.g., bi-weekly) to share their passion for RPA development by exchanging implementation ideas, sharing project experiences, discussing tools, code, and engineering practices. These guilds weren't closed forums. Anyone in the organization was free to join any guild, to follow any or none of the guild activities, resign at any time or remain inactive for as long as they wanted. For example, the infrastructure guild was joined by people from IT Ops and software development too. Another example is the coach guild. Coaches in different forms but similar duties (e.g., scrum masters, agile coaches, project/program managers) were spread all over the organization. In these sessions, the coaches collaborated on high-level organizational improvement areas and tracked them on an improvement board. In this case, the guild was primarily an arena for joint problem-solving.

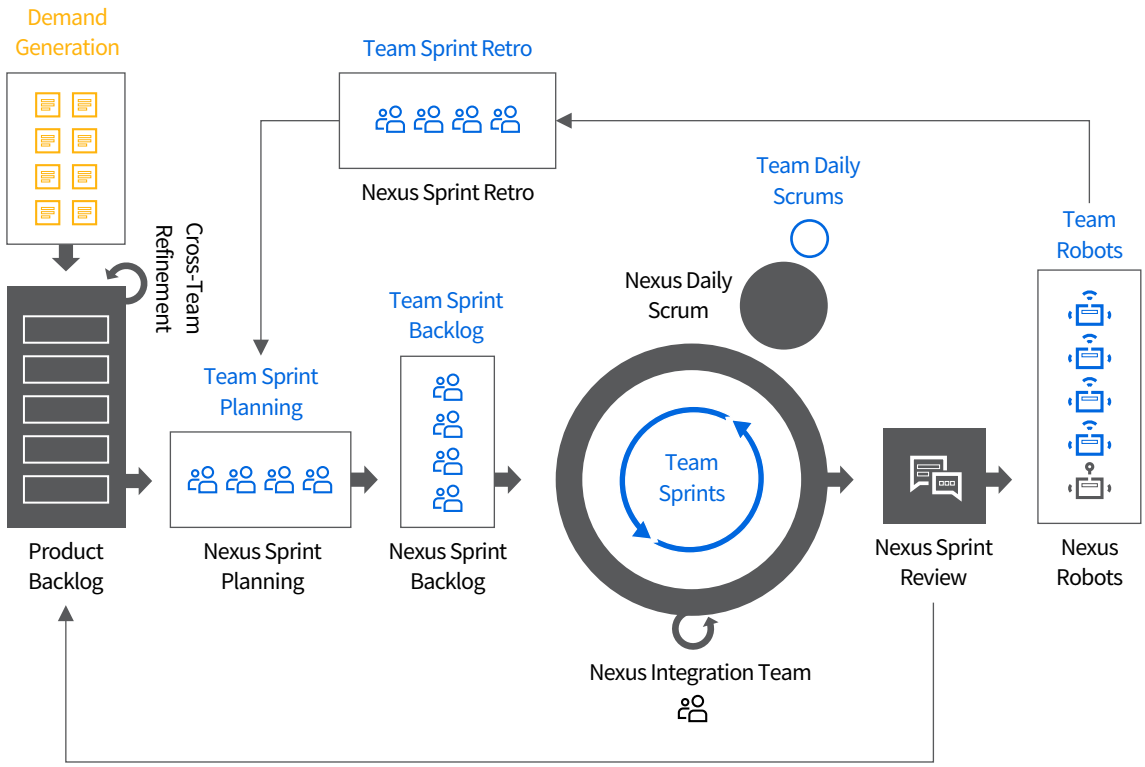
Everyone was welcome but no one was safe to just lean back and relax. A guild (usually) was a working group. The focus was doing, not talking. You know, an ounce of practice is worth more than tons of theory. The hands-on character of these sessions was key to keep the people engaged. The guilds had no fixed duration but usually lasted no more than two hours. Each session usually focused on one specific topic that was neither too big nor too small. Each guild was moderated. A guild coordinator introduced the topic, prepared exercises, moderated the discussion, collected action items, and summarized the lessons learned after each session. These sessions not only closed skill and knowledge gaps among similar roles in different teams and different departments but also reminded the participants that unity is strength and division is weakness. A cross-team, cross-department, and all too often also a cross-functional learning culture emerged. In short, guilds helped to scale knowledge. An overview of the individual and organizational benefits of the guilds at Spotify is given by Darja Smite (2020).

3.10 Orchestration

The CoE scaled agile RPA delivery with Nexus. There are numerous scaling frameworks (e.g., LeSS, SAFe, Spotify Model, Scrum@Scale) available. None is better than the other. Some work better for one company, some work better for another. Nexus just worked best in Susan's context. It would be outside the scope of this paper to go into too much detail about Nexus. We restrict ourselves to its main characteristics. First, Nexus builds upon Scrum. It extends Scrum only where absolutely necessary to minimize and manage dependencies between multiple scrum teams. Nexus neither changes the core design of Scrum, nor leaves out elements, nor negates the rules of Scrum. Ergo, scaled Scrum remains Scrum.

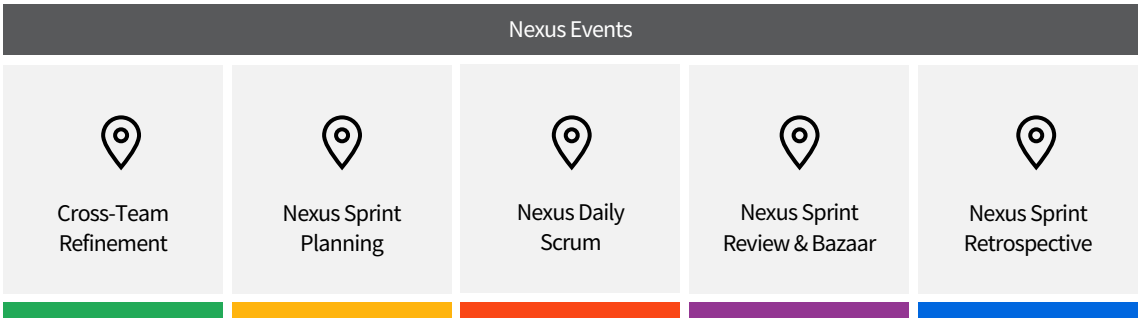
The term nexus refers to 'a connected group'. So, the goal of this framework is to help scrum teams to solve common scaling challenges like reducing cross-team dependencies, preserving team self-management and transparency, and ensuring accountability. Nexus helps to make dependencies transparent. This is exactly what Susan was looking for. The increasing number of people and teams in the CoE increased complexity in terms of team dependencies, the need for coordination and collaboration, and the number of communication pathways involved in making decisions. The scrum teams initially spent most of their time each sprint just staying out of each other's way. The quantity, quality, and speed of the RPA throughput per scrum team began to decline due to issues such as cross-team dependencies, duplication of work, and communication overhead. There is no such thing as free lunch when it comes to scaling RPA. A framework for coordinating this ecosystem of scrum teams efficiently and effectively was needed. The winner was Nexus. The goal of Nexus is to scale the RPA throughput as uniformly as possible. In other words, a certain percentage increase in the number of scrum teams should (at best) go hand in hand with a corresponding increase in the RPA throughput.

Nexus was designed to support multiple scrum teams that work on a single product. This was generally not true for the CoE. Each scrum team automated task automations and intra-departmental processes for different business units and only collaboratively automated cross-departmental business processes. Nevertheless, the task automations and the automations of intra-departmental business processes were highly interdependent.



The resulting robots shared the same infrastructure and often consumed identical applications, databases, and APIs. Ergo, the building blocks of the robots (e.g., process components) were shared across the scrum teams to avoid redundancy, speed up robot development, and reduce maintenance. So, Susan concluded that the set of robots developed by the scrum teams not only should be seen but must be seen as one integrated product.

Let's now briefly walk through the Nexus framework. For detailed information, please refer to the Nexus Guide. The Nexus framework as seen by the CoE is depicted above. The automation opportunities were discovered via the top-down and bottom-up approach explained in section 2.2. This ensured continuous demand for RPA. These automation opportunities were then prioritized in the product backlog by Susan and the RPA solution architects in so-called cross-team refinements. In these sessions, the product backlog items were decomposed from large and vague automation requests (e.g., epics) to actionable work items (e.g., user stories) that a scrum team could deliver during a sprint. The goal was to identify dependencies between scrum teams, roughly estimate the development effort and the business value (e.g., time/cost savings) for the product backlog items, and forecast which scrum teams will deliver which item. This refinement process was an ongoing activity.



The nexus sprint planning was a sprint planning event on top of each team's sprint planning. So, the teams were still doing their own sprint planning but before the individual teams started their sprint planning, they had a short coordination meeting where one representative from each team joined. The purpose was to discuss the bigger picture (e.g., align sprint goals, overall strategy), assign work items, and make cross-team dependencies transparent. From these sprint plannings the sprint backlogs were derived. The nexus sprint backlog can be understood as a combined view of the sprint backlogs of the individual teams. In the nexus sprint backlog the dependencies and the flow of work between the teams were highlighted. In short, the nexus sprint backlog was just another way to look at the individual team sprint backlogs to get the big picture.

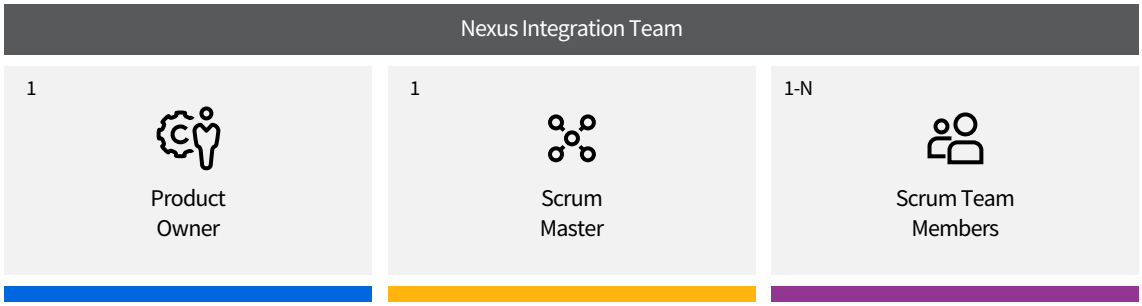
The teams then worked on their spring backlog items during two-week sprints. The resulting robot increments were simple task automations, and/or automations of complex intra-departmental and/or cross-departmental processes. Multiple teams usually collaborated on developing automations of cross-departmental processes. The resulting robot increments were called nexus robots, otherwise they were called team robots. Due to the numerous technical dependencies also between the team robots, it was key that the teams regularly integrated their work to avoid a big bang integration disaster at the end of each sprint.

This integration behavior was fostered by the daily scrum events. The nexus daily scrum was again an event on top of the daily scrums of the individual teams. The main purpose of the nexus daily scrum was to identify any integration issues between the teams and to inspect progress toward the joint sprint goal (aka nexus sprint goal). This information then fed into each team's daily scrum. One representative from each team usually joined the nexus daily scrum. The nexus daily scrum didn't happen daily but twice a week. This was sufficient to align the teams around integration challenges. The CoE continuously strived for minimizing standardization. One of their core principles was minimum viable bureaucracy which refers to a way of having just enough process (e.g., meetings) to get things done but not so much as to make it cumbersome.

On the last day of each sprint, the sprint review was held in two modes. First, it was held in the form of a status meeting. The goal was to collect feedback about the robot increments and to determine future adaptations. Everyone was welcome but usually the key business stakeholders (e.g., RPA business analysts) and Susan as the product owner joined this event. This event lasted no longer than 1.5 hours. Each team had maximum twenty minutes to showcase their individual or joint achievements. The primary goal was to demonstrate why something has been done and what has been achieved with only little attention to the how. The mantra was to keep these events simple by keeping them short. These sessions were recorded. The recordings were accessible to everyone in the organization. This event wasn't just valuable for the teams in terms of getting feedback; each team also got informed about the achievements of other teams. Hence, each team's in-sprint performance became visible to everyone. This fostered a healthy competition within the CoE.

Secondly, the sprint review was held in the form of an informal bazaar that happened immediately after the status meeting from above. The purpose was to focus exclusively on the how. It gave the participants (e.g., business stakeholders) the chance to get their (virtual) hands on the robot increments to provide more in-depth feedback. This was invaluable since the majority of robots developed by the CoE were attended robots, not unattended robots. This event turned into a cross-functional collaboration platform where stakeholders of one department (e.g., marketing) exchanged ideas and experiences centered around RPA with stakeholders from other departments (e.g., human resources).

The nexus sprint retrospective concluded the sprint. This event represented a container around the individual team's sprint retrospectives. One representative from each team joined. The goal was to highlight, discuss, and find resolutions to the challenges that the set of all teams (nexus) was facing during the last sprint with regards to individuals, teams, interactions, processes, tools, and its definition of done (DoD). This information was then fed into the team's sprint retrospectives. So, the nexus sprint retrospective was one opportunity for the nexus to continuously improve its operation at scale. These events usually happened on the first day of the next sprint.



Finally, the nexus integration team. This team consisted of the product owner (aka RPA change manager), one scrum master (aka RPA team coach), and one or more scrum team members (e.g., RPA developer, RPA tester). The composition of this team changed according to the changing challenges the scrum teams were facing. The nexus integration team was responsible and accountable that the scrum teams are well-integrated in terms of making sure that the processes and tools used for cross-team communication, collaboration, and coordination are effective and that (integrated) working robots are delivered every sprint. The nexus integration team also defined and constantly improved engineering practices, the definition of ready (DoR), and the definition of done (DoD) shared across all scrum teams. So, the members of the nexus integration team acted as servant leaders (e.g., coaches, guides) for the individual scrum teams. In other words, the nexus integration team acted as a scrum master acts to a scrum team, not as 'in charge' but through servant leadership (Rob Maher, 2016).

This is the short story of how the CoE scaled RPA delivery with Nexus. According to Ken Schwaber (2015), Nexus can facilitate between three and nine scrum teams. For nine scrum teams, Nexus is likely to start to fray due to increasing dependencies and integration issues. Susan likes to be prepared and so she is already taking a closer look at Nexus' bigger sister called Nexus+ to prepare the CoE for its future scaling adventures. Time will tell.

3.11 Collaboration

Organizations are made up of people. RPA initiatives are no exception. The organizational structure in Susan's company was centered around functional departments (e.g., sales, finance, marketing, software development, IT Ops). In its early days, the organization was characterized by hierarchical, siloed, and fragmented processes and cultures. These organizational silos reinforced an 'us versus them' feeling between departments. This was obviously a barrier standing in the way of cross-functional alignment which is imperative for an agile way of RPA delivery. The good news is that the digital transformation program driven by the leadership team also aimed at 'breaking down' these functional silos by changing the organizational structure. In the end, it was a good mix between a traditional, hierarchical and modern, holacratic structure that enabled Susan's company to act both reliably and fast. The evolution of the organizational structure is best described in this wonderful video by John Kotter (2013). The bottom line is that structure guides behavior (Jason Little). The goal of the leadership team was to allow a healthy corporate culture to emerge by changing the organizational structure. The result was that more and more cross-functional forums in Susan's organization evolved naturally.

The RPA initiative was one of the first cross-functional forums. It brought people together across different levels and functions. So, the RPA initiative was part of a bigger company-wide change program. Susan's organization realized that an integrated culture change program is a critical component to successful digital transformation. In other words, the 'joint us' started to become stronger and stronger than the 'individual us' of the functional departments. Collaboration was key for the RPA initiative. Susan learned that not only the technical but also the cultural environment that surrounded the RPA CoE was critical to the success of the RPA initiative. Close collaboration had a multiplying effect on its success.

Therefore, the initial reluctance of the stakeholders from different departments (e.g., business, IT Ops, software development) to actively participate in this RPA initiative was surprisingly low. In hindsight, this low reluctance of the stakeholders can also be attributed to Susan's continuous evangelism efforts. It's important to note that Susan's effort was actively supported by the leadership team. The CIO once said that if you don't support RPA from a leadership perspective, it's like telling a pilot that you are going to double his salary for being on time, and then you load her plane an hour late.

Susan started 'bringing the good news' about RPA right from the beginning and she never stopped doing it. The slogan was 'make our story their story'. This means that Susan acted like a chief meaning officer who not only explained to her stakeholders where she wants to go but also showed what's in it for them if they join her on the journey (Jack Welch). It took some time for the employees to understand and to accept that automation is augmentation, not replacement. So, Susan's continuous evangelism helped the stakeholders to understand that RPA could have a significant positive impact on their overall quality of work. This then triggered their intrinsic motivation for collaboration. They developed the drive to support the CoE in getting the job done. They were generally open to learning new things and dealing with new ways of collaboration, which included an agile way of working. This was important since the technical complexity of RPA can already be quite challenging, but

then getting people from different departments onboard poses an additional challenge. So, doing RPA is one thing, selling it to your stakeholders is another.

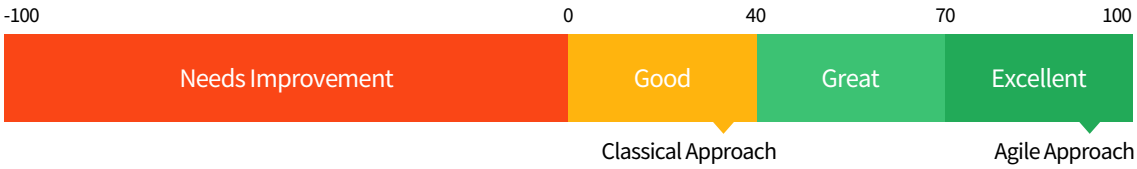
Apart from fast and continuous RPA delivery at high quality that was enabled by intensive cross-department collaboration (e.g., software development, IT Ops, business), three factors significantly contributed to the success of the RPA initiative. First, leadership support. We don't want to stretch this topic. You get the point. Secondly, continuous evangelism. Constantly creating awareness about RPA and sharing tangible benefits will turn awareness into excitement about RPA. So, do not focus on technology only; focus on sales too. Thirdly, corporate culture. A healthy culture supports learning, collaboration, innovation, and experimentation across departmental boundaries. You won't be able to emerge this culture by yourself. Engage your leadership (e.g., senior, upper, executive management) to break down silos and to build bridges between business, IT Ops, and software development. These bridges are your seeds that allow your RPA to grow. You won't go far without them. Remember, agile RPA delivery is fundamentally people-oriented. So, the set of your stakeholders is your social capital that enables your RPA to scale. In other words, great RPA comes from great collaboration.

4 Benefits

How did Susan recognize that the CoE is doing agile RPA well? Well, she looked at the robots after every sprint and assessed whether they are improving (e.g., higher reliability, availability, functionality, performance). Doing agile RPA means having working robots that improve visibly after each sprint. So, Susan's primary measure of success was a continuous stream of increasingly improving robots. This metric mattered most to the business stakeholders. Susan's goal was not to quantify how much the robots improved but to assess that the robots improved from one sprint to another. This continuous stream of increasingly improving robots led to constantly increasing time/cost savings and eventually to increased stakeholder satisfaction. This, in turn, increased the team morale of the CoE. The morale of the CoE was constantly monitored and assessed by the RPA team coaches. The stakeholder satisfaction was assessed periodically through anonymous surveys and discussions during the sprint review sessions.

Susan understood that if she doesn't pay attention to the feedback then there's no point in collecting feedback. So, she constantly tried to pinpoint where the gaps between what the stakeholders perceived and what they desired lied. Then she took action to close those gaps. Susan knew that one of the worst things you can do is to ask for feedback and then fail to act on the results of that feedback. This just allows distrust to emerge.

Susan considered the CoE as a service provider. Therefore, she introduced a stakeholder net promoter score (sNPS) to measure the stakeholder's satisfaction. The sNPS asks one simple question: On a scale of 0-10, how likely are you to recommend the services offered by the RPA CoE to others? If the respondents answer 9-10, they are promoters. If they answer 0-6, they are detractors. You then calculate the sNPS by deducting the percentage of detractors from the percentage of promoters. You ignore those who score 7-8. These are your passives. So, expressed in absolute terms, the sNPS ranges from -100 to +100. Susan started to measure the sNPS after about two months after the RPA initiative was launched. In the remaining six months of classical RPA delivery, the CoE reached an average sNPS of +38. The shift to agile RPA delivery increased the sNPS to +88.

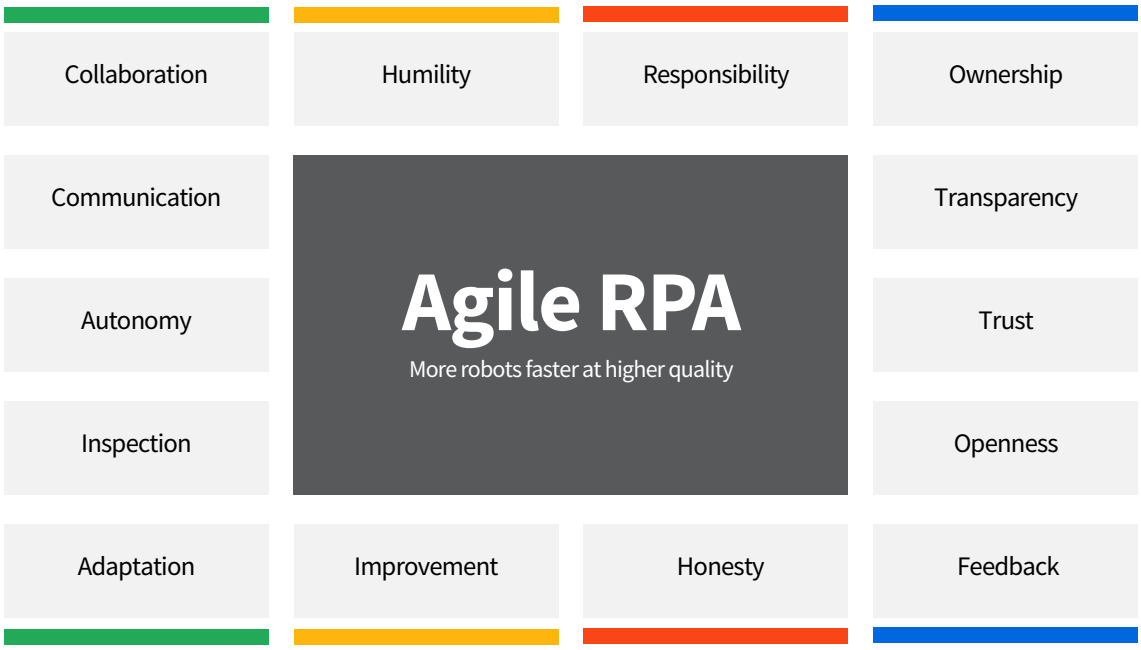


The CoE only rarely had distractors, mostly promoters and passives. Imagine you have 50 respondents with 19 promoters and 31 passives. This results in an sNPS of +38. If you have 44 promoters and 6 passives, the sNPS is

+88. So, the shift to agile RPA delivery mainly converted the passives to promoters. This implies that the CoE wasn't just seen as a good service anymore but as an excellent service by the business stakeholders.

For Susan, stakeholder satisfaction was the single most important indicator for success. She didn't hide behind vanity metrics. She regarded a robot as a solution to the stakeholder's problem. The better the robot, the better the solution to the problem. The better the problem solution, the more value for the stakeholder is created. The higher the stakeholder value, the higher the stakeholder satisfaction. The higher the stakeholder satisfaction, the more demand for future robots. The more demand for future robots, the bigger the opportunity to create value (e.g., time/cost savings) for the entire organization. The more value the CoE creates for the organization, the more valuable the CoE is for the organization. This was Susan's simple recipe for success.

The excellent sNPS rating can be seen as a numerical manifestation of the improvements that came along with the shift to agile RPA delivery: More robots were delivered faster at higher quality. This wasn't based on Susan's quantitative measurements; it was the general qualitative perception of the stakeholders. Susan once said that 'no matter how great the CoE believes itself to be, the real verdict lies in the hands of its stakeholders'. A true stakeholder-centric approach. This takes courage.



The success had many faces. It's hard to pinpoint individual factors. For example, the increase in the quality of the robots (e.g., increased uptime) was a direct consequence of continuous testing and advanced engineering practices. This laid the technical basis for developing more robots faster. So, just faster robot development without paying attention to quality doesn't lead to more robots in the long run. Tim Ottinger (2006) once wisely stated that if you are going faster than you are able to manage, then what you are doing is not agile, it's just hurried. Susan understood this very well. She was aware that the CoE needs to move at a sustainable pace instead of a reckless pace to avoid turning agile into fragile.

In addition, the reduced amount of rework (waste) involved in the delivery process was another key enabler to deliver more robots faster at higher quality. As soon as waste was detected, its cause was eliminated. The CoE adopted a posture of inspection and adaptation that resulted in a virtuous cycle of continuous improvement. This was enabled by a transparent delivery process embedded in an enabling as well as a supportive working environment that welcomed change, not resisted change. This facilitated fast and honest feedback through

close collaboration and open communication between software development, IT Ops, and the business. This allowed trust to emerge between these parties. This was vitally important since without trust there is no agile.

Trust also emerged within the CoE. Susan created a sense of collective problem ownership and responsibility by involving the CoE in making decisions rather than handing over decisions. The autonomy of the CoE was on top of Susan's agenda. This triggered their intrinsic motivation to continuously improve. The CoE accepted to never be satisfied as there is always room for improvement. This led to an increasing level of humility, where the CoE would be 'ashamed of delivering bad robots'. The CoE became self-regulated and self-organized. This was crucial especially in situations where uncertainty and chaos were the norm. Susan has learned that agile can only work if managers act as enablers rather than controllers.

So, she strived for providing a team-centered nurturing environment where everyone was equal. This created a psychologically safe environment where one wins and loses together as a team (Stephen Denning, 2018). This story could go on and on. You get the point. The success of this RPA initiative was significantly determined by something that is hard to describe and almost impossible to quantify: the agile mindset. In a word, it's a way of thinking, a frame of mind. Based on the findings by Carol Dweck (2007), Susan sees an agile mindset as a growth mindset. We adopt a growth mindset when we are able to embrace challenges instead of avoiding them, persist in the face of obstacles instead of giving up, see effort as the path to mastery instead of seeing it as fruitless, learn from criticism instead of ignoring it, and find lessons and inspiration in the success of other people instead of feeling threatened by the success of others.

This suggests that you only find out what agile really is when you see it. For Susan, the agile mindset developed by the CoE during transitioning from classical to agile RPA delivery was primarily characterized by the drive for achieving a state of being agile instead of merely doing agile. Susan has learned that doing agile can be achieved overnight, being agile takes time. She has learned that agile begins with attitude. It begins with you.

5 Conclusion

The intent of this paper is not to promote agile methodologies as universal tools to master RPA. There's no sure formula for success. While agile RPA delivery has numerous benefits, it also carries certain risks that are inherent to the agile approach. Carlijn Hattink (2016) explores the risks teams employing an agile approach are facing and how these can be addressed, including some suggestions on the auditability of this type of process. The bottom line is that working in an agile way worked in Susan's context. It might not work in your context. It's context-specific. It's not for everyone. It comes down to the people. Gerald Weinberg (1986) still seems to be right: 'No matter how it looks at first, it's always a people problem'. If the people involved in RPA delivery aren't willing to collaborate closely, then working in an agile way is most likely going to be a big struggle. You won't see its advantages. So, do not impose agile practices and principles on people that are reluctant to adopting it. Remember, you cannot really convince anyone of anything. You can only give people the right information so that they convince themselves. So, do not seek to convince, seek to explain what agile brings to the table.

Your partners for Agile RPA

AGILE

Learn more about Agile.

VISIT SCRUM.ORG

Scrum.org™
The Home of Scrum

RPA

Learn more about RPA.

VISIT UIPATH

UiPath Reboot Work.™