<https://www.blueprism.com/automation-journey/how-to-implement-rpa/>

<https://www.blueprism.com/automation-journey/rpa-glossary/>

<https://www.blueprism.com/automation-journey/process-discovery-rpa-automation/>

<https://www.blueprism.com/automation-journey/rpa-challenges-overcoming-barriers-automation-adoption/>

<https://www.blueprism.com/resources/blog/how-to-select-an-rpa-vendor/>

<https://www.blueprism.com/automation-journey/automation-rpa-center-of-excellence-coe/>

**What you’ll do**

* **Design, develop, and operate high scale applications across the full engineering stack using SDLC best practices.**
* **Apply modern software development practices (UIPath Robots, GCP services, CI/CD, web development, etc.)**
* **Work across teams to integrate our systems with existing internal systems.**
* **Generate innovation roadmap and architecture discussions to turn business requirements and vision into reality.**
* **Lead a globally distributed engineering team.**
* **Manage project priorities, deadlines, and deliverables.**
* **Research, create, and develop software applications to extend and improve on Equifax Solutions**
* **Actively participate in Sprint planning, Sprint Retrospectives, and other team activity.**
* **Lead business needs with capacity to handle technical and non-technical conversations.**

**What experience you need**

* **Bachelor's degree or equivalent experience**
* **5+ years of software engineering experience**
* **3+ years experience in a l eadership role within a technical team**
* **5+ years experience with Cloud technology: GCP, UIPath, web development**
* **5+ years experience designing and developing RPA-AI solutions**
* **Fluent English Level (C1+)**

**What could set you apart**

* **Master degree in: technology, administration, or other similar areas.**
* **UI development (e.g. HTML, JavaScript, Angular and Bootstrap)**
* **Agile environments (e.g. Scrum, XP)**
* **Relational and non-relational databases (e.g. SQL Server, MySQL, BigQuery)**
* **Atlassian tooling (e.g. JIRA, Confluence, and Github)**

A screenshot of a computer program

Description automatically generated

A computer screen shot of a computer code

Description automatically generated

3.

**SDLC best practices.**

Design, develop, and operate high scale applications across the full engineering stack using SDLC best practices.

Implementing best practices in the Software Development Life Cycle (SDLC) can significantly enhance the quality, efficiency, and security of your software projects. Here are some essential SDLC best practices:

1. **Formalize Requirement Analysis**:
   * Clearly define and document the software requirements. This helps ensure all stakeholders have a shared understanding of the project's goals and reduces the risk of scope creep.
2. **Adopt a Suitable SDLC Model**:
   * Choose an SDLC model that fits your project needs. Agile is great for projects requiring flexibility and iterative development, while Waterfall is suitable for well-defined projects with clear stages.
3. **Emphasize Code Quality and Reviews**:
   * Implement coding standards and conduct regular code reviews. This practice helps maintain high code quality and allows for early detection of potential issues.
4. **Automate Testing**:
   * Use automated testing tools to perform unit, integration, and regression tests. Automated testing ensures consistent and thorough testing, which improves software reliability.
5. **Continuous Integration and Continuous Deployment (CI/CD)**:
   * Integrate CI/CD pipelines to automate the build, test, and deployment processes. This practice accelerates release cycles and ensures that new code changes are quickly and safely integrated into the main branch.
6. **Focus on Security**:
   * Incorporate security best practices throughout the SDLC. This includes conducting security assessments, using secure coding practices, and performing regular vulnerability scans.
7. **Maintain Comprehensive Documentation**:
   * Keep detailed documentation for every phase of the SDLC. This includes requirement specifications, design documents, test plans, and user manuals. Good documentation facilitates better communication and knowledge transfer.
8. **Monitor and Maintain**:
   * After deployment, continuously monitor the software for performance issues and bugs. Regular maintenance and updates are crucial to keep the software secure and efficient.
9. **Optimize Developer Workflow**:
   * Streamline workflows by using tools and practices that enhance productivity, such as version control systems, task management tools, and collaborative platforms.
10. **Regularly Review and Improve Processes**:

* Periodically review your SDLC processes and practices. Gather feedback from the team and stakeholders to identify areas for improvement and implement necessary changes.

**Testing**

**Functional Testing**

1. **Unit Testing**: Tests individual components or units of code to ensure they work as expected.
2. **Integration Testing**: Ensures that different modules or services work well together.
3. **System Testing**: Tests the complete system as a whole to verify it meets the specified requirements.
4. **Acceptance Testing**: Validates the software against user requirements and checks if it is ready for delivery.

**Non-Functional Testing**

1. **Performance Testing**: Assesses the speed, responsiveness, and stability of the software under various conditions.
2. **Security Testing**: Identifies vulnerabilities and ensures the software is protected against threats.
3. **Usability Testing**: Evaluates the user-friendliness and overall user experience of the software.
4. **Compatibility Testing**: Checks if the software works across different devices, browsers, and operating systems.

**Other Types of Testing**

1. **Smoke Testing**: A preliminary test to check the basic functionality of the software.
2. **Regression Testing**: Ensures that new code changes do not adversely affect existing functionalities.
3. **Exploratory Testing**: Involves exploring the software without predefined test cases to discover defects.
4. **Automated Testing**: Uses scripts and tools to perform tests automatically, which is useful for repetitive tasks.

**Redshift and Google BigQuery**

Amazon Redshift and Google BigQuery are both powerful data warehousing solutions, but they have some key differences that might make one more suitable for your needs than the other. Here's a comparison:

**Architecture**

* **Amazon Redshift**: Uses a cluster-based architecture where you provision clusters and nodes. This requires some management tasks like vacuuming tables and resizing clusters.
* **Google BigQuery**: Is serverless, meaning you don't need to manage any infrastructure. It automatically handles scaling and resource management.

**Performance**

* **Amazon Redshift**: Optimized for complex queries and can handle large-scale data analytics with high performance using columnar storage and parallel processing
* **Google BigQuery**: Also optimized for large-scale data analytics and can process terabytes of data in seconds and petabytes in minutes. It uses a distributed architecture to achieve this.

**Cost**

* **Amazon Redshift**: Offers a pay-as-you-go pricing model based on the amount of data stored and the compute resources used. It can be cost-effective for predictable workloads.
* **Google BigQuery**: Charges based on the amount of data processed by queries and the amount of data stored. It can be more cost-effective for variable workloads.

**Integration**

* **Amazon Redshift**: Integrates seamlessly with other AWS services and supports a wide range of third-party tools.
* **Google BigQuery**: Integrates well with other Google Cloud services and supports various third-party tools, especially those within the Google ecosystem.

**Management**

* **Amazon Redshift**: Requires some manual management tasks, such as resizing clusters and vacuuming tables.
* **Google BigQuery**: Automatically manages resources and optimizes performance without manual intervention.

**Use Cases**

* **Amazon Redshift**: Ideal for businesses with predictable workloads and those already using AWS services extensively.
* **Google BigQuery**: Suitable for businesses with variable workloads and those that rely on other Google Cloud services.

**CloudBees -Jenkins**

Both platforms are robust and capable, so the best choice depends on your specific needs and existing infrastructure. If you have more specific requirements or scenarios, feel free to share, and I can provide more tailored advice!

CloudBees and Jenkins are closely related but serve different purposes. Here are the key differences:

**Jenkins**

* **Open Source**: Jenkins is an open-source automation server used for continuous integration and continuous delivery (CI/CD) purposes.
* **Community Support**: It relies on community support for troubleshooting and updates.
* **Flexibility**: Jenkins is highly flexible and extensible through a vast ecosystem of plugins.
* **Deployment**: Can be deployed on-premises or in the cloud.

**CloudBees**

* **Commercial Product**: CloudBees offers a commercial version of Jenkins, known as CloudBees CI, which includes additional enterprise features.
* **Enterprise Support**: Provides enterprise-level support, including service-level agreements (SLAs) and 24/7 technical support.
* **Enhanced Features**: Includes features for scalability, high availability, security (e.g., role-based access control), and compliance.
* **Management Tools**: Offers advanced management tools for large-scale Jenkins environments, such as analytics, reporting, and integration with various enterprise tools.

In summary, while Jenkins is a powerful and flexible tool for CI/CD, CloudBees builds on Jenkins to provide a more robust, scalable, and manageable solution for enterprise environments.

**Spinnaker**

Spinnaker is an open-source, multi-cloud continuous delivery platform originally developed by Netflix and extended by Google2

. It helps you release software changes with high velocity and confidence by providing a powerful and flexible pipeline management system

. Here are some key features:

* **Multi-Cloud Support**: Spinnaker integrates with major cloud providers like AWS, Google Cloud Platform, Microsoft Azure, and Kubernetes.
* **Pipeline Management**: It allows you to create deployment pipelines that can run integration and system tests, spin up and down server groups, and monitor rollouts.
* **Deployment Strategies**: Supports various deployment strategies such as blue/green deployments, canary releases, and rolling updates.
* **Built-in Best Practices**: Encourages the use of immutable infrastructure and other best practices to ensure reliable and repeatable deployments.

Spinnaker is particularly useful for organizations looking to standardize their release processes and improve the quality of their software deployments

**Scrum**

Scrum is an Agile framework used for managing complex projects, particularly in software development. It involves several key steps or phases:

1. **Initiation**: This phase involves defining the project's vision, goals, and high-level requirements. The Product Owner creates the product backlog, which is a prioritized list of features and tasks.
2. **Planning and Estimates**: The team plans the work for the upcoming sprint (a time-boxed period, usually 2-4 weeks). They break down the product backlog into smaller, manageable user stories and estimate the effort required for each.
3. **Implementation (Sprint Execution)**: During the sprint, the development team works on the tasks defined in the sprint backlog. Daily Scrum meetings (stand-ups) are held to discuss progress, challenges, and next steps.
4. **Review**: At the end of the sprint, a Sprint Review meeting is held where the team demonstrates the completed work to stakeholders. Feedback is gathered to improve future sprints.
5. **Retrospective**: After the Sprint Review, the team holds a Sprint Retrospective meeting to reflect on the sprint. They discuss what went well, what didn't, and how processes can be improved for the next sprint.
6. **Release**: Once the product increment is ready and meets the Definition of Done, it is released to the users. This can happen at the end of each sprint or after several sprints, depending on the project's requirements.

These steps are iterative, meaning they repeat for each sprint, allowing for continuous improvement and adaptation throughout the project.

# Extreme Programming (XP) and Scrum

Extreme Programming (XP) and Scrum are both Agile methodologies, but they have different focuses and practices. Here's a comparison:

**Focus**

* **XP (Extreme Programming)**: Emphasizes technical practices and aims to improve software quality and responsiveness to changing customer requirements. It focuses heavily on engineering practices like Test-Driven Development (TDD), pair programming, and continuous integration.
* **Scrum**: Focuses on project management and team collaboration. It provides a framework for managing complex projects through iterative development and regular feedback.

**Iteration Length**

* **XP**: Typically has shorter iterations, usually one to two weeks.
* **Scrum**: Iterations, known as sprints, usually last from two weeks to one month.

**Flexibility in Requirements**

* **XP**: More flexible with changes during iterations. New features can be swapped in as long as the team hasn't started working on the replaced feature.
* **Scrum**: Less flexible once a sprint has started. Changes are generally not allowed until the next sprint.

**Engineering Practices**

* **XP**: Prescribes specific engineering practices such as TDD, pair programming, simple design, and refactoring.
* **Scrum**: Does not prescribe specific engineering practices but focuses on the process and roles within the team.

**Roles**

* **XP**: Roles are less defined compared to Scrum. The focus is more on practices and collaboration.
* **Scrum**: Defines specific roles such as Product Owner, Scrum Master, and Development Team.

**Prioritization**

* **XP**: Features are developed in strict priority order as determined by the customer.
* **Scrum**: The Product Owner prioritizes the backlog, but the team decides the sequence of work within a sprint.

Both methodologies aim to improve software development processes and outcomes, but they do so in different ways. XP is more prescriptive about technical practices, while Scrum provides a broader framework for managing projects and teams.

# Good manager and leader

Being a good manager and leader involves a combination of skills, behaviors, and attitudes. Here are some key tips to help you excel in both roles:

**1. Communicate Effectively**

Clear and consistent communication is crucial. Make sure your team understands their tasks, goals, and the overall vision. Encourage open dialogue and be approachable..

**2. Lead by Example**

Demonstrate the behaviors and work ethic you expect from your team. Your actions set the standard for others to follow..

**3. Foster a Collaborative Environment**

Encourage teamwork and collaboration. Create an atmosphere where team members feel comfortable sharing ideas and feedback.

**4. Be Growth-Oriented**

Support your team's professional development. Provide opportunities for learning and growth, and help them achieve their career goals..

**5. Motivate and Inspire**

Recognize and celebrate achievements. Show appreciation for your team's hard work and motivate them to reach their full potential..

**6. Adapt and Learn**

Be open to feedback and willing to adapt. Continuous learning and improvement are essential for effective leadership..

**7. Build Trust**

Establish trust by being reliable, transparent, and fair. Trust is the foundation of a strong team.

**8. Make Informed Decisions**

Use data and insights to make informed decisions. Involve your team in the decision-making process when appropriate.

**9. Manage Conflicts**

Address conflicts promptly and fairly. Foster a respectful environment where issues can be resolved constructively.

**10. Prioritize Well-Being**

Support the well-being of your team. Encourage a healthy work-life balance and be mindful of their mental and physical health.

By focusing on these areas, you can develop into a strong manager and leader who not only drives results but also fosters a positive and productive work environment.

# Manifiesto por el Desarrollo Ágil

Individuos e interacciones sobre procesos y herramientas  
Software funcionando sobre documentación extensiva  
Colaboración con el cliente sobre negociación contractual  
Respuesta ante el cambio sobre seguir un plan

**Manifiesto por el Desarrollo Ágil de Software**

Nuestra mayor prioridad es satisfacer al cliente  
mediante la entrega temprana y continua de software  
con valor.

Aceptamos que los requisitos cambien, incluso en etapas  
tardías del desarrollo. Los procesos Ágiles aprovechan  
el cambio para proporcionar ventaja competitiva al  
cliente.

Entregamos software funcional frecuentemente, entre dos  
semanas y dos meses, con preferencia al periodo de  
tiempo más corto posible.

Los responsables de negocio y los desarrolladores  
trabajamos juntos de forma cotidiana durante todo  
el proyecto.

Los proyectos se desarrollan en torno a individuos  
motivados. Hay que darles el entorno y el apoyo que  
necesitan, y confiarles la ejecución del trabajo.

El método más eficiente y efectivo de comunicar  
información al equipo de desarrollo y entre sus  
miembros es la conversación cara a cara.

El software funcionando es la medida principal de  
progreso.

Los procesos Ágiles promueven el desarrollo  
sostenible. Los promotores, desarrolladores y usuarios  
debemos ser capaces de mantener un ritmo constante  
de forma indefinida.

***La atención continua a la excelencia técnica y al  
buen diseño mejora la Agilidad.***

La simplicidad, o el arte de maximizar la cantidad de  
trabajo no realizado, es esencial.

Las mejores arquitecturas, requisitos y diseños  
emergen de equipos auto-organizados.

A intervalos regulares el equipo reflexiona sobre  
cómo ser más efectivo para a continuación ajustar y  
perfeccionar su comportamiento en consecuencia.

**5 rules of XP the extreme programming methodology**

The values of extreme programming are the more philosophical aspects. The rules, on the other hand, are the practical uses for how the work gets done. You’ll need both to run an effective XP team.

1. Planning

In the [planning stages](https://asana.com/resources/sprint-planning-meeting) of XP, you’re determining if the project is viable and the best fit for XP. To do this, you’ll look at:

* User stories to see if they match the simplicity value and check in to ensure that the customer is available for the process. If the user story is more complex, or it’s made by an anonymous customer, it likely won’t work for XP.
* The business value and priority of the project to make sure that this falls in line with “getting the most important work done first.”
* What stage of development you’re in. XP is best for early stage development, and won’t work as well for later iterations.

Once you’ve confirmed the project is viable for XP, ‌create a release schedule—but keep in mind that you should be releasing early and often to gain feedback. To do this:

* Break the project down into iterations and create a plan for each one.
* Set realistic deadlines and a sustainable pace.
* Share updates as they happen, which empowers your team to be honest and transparent.
* Share real-time updates that help the team ‌identify, adapt, and make changes more quickly.
* Use a [project management tool](https://asana.com/product) to create a [Kanban board](https://asana.com/resources/what-is-project-portfolio-management) or timeline to track your progress in real-time.

2. Managing

One of the key elements of XP is the physical space. XP purists recommend using an open workspace where all team members work in one open room. Because XP is so collaborative, you’ll benefit from having a space where you can physically come together. But that’s not always practical in this day and age. If you work on a remote team, consider using a [platform](https://asana.com/product) that encourages [asynchronous work](https://asana.com/resources/synchronous-vs-asynchronous-communication) for [remote collaboration](https://asana.com/resources/remote-collaboration). This way, all members can continue to work on the project together, even if they’re not physically together.

As in other Agile methods, use [daily standups](https://asana.com/templates/for/marketing/agile-daily-standup) meetings to check-in and encourage constant, open communication. You’ll want to use both a weekly cycle and quarterly cycle. During your quarterly cycle, you and your team will review stories that will guide your work. You’ll also study your XP process, looking for gaps or opportunities to make changes. Then, you’ll work in weekly cycles, which each start with a customer meeting. The customer chooses the user story they want programmers to work on that week.

As a manager or team lead, your focus will be on maintaining work progress, measuring the pace, shifting team members around to address bugs or issues as they arise, or changing the XP process to fit your current project and iteration. Remember, the goal of XP is to be flexible and take action, so your work will be highly focused on the team’s current work and reactive to any changes.

3. Designing

When you’re just starting out with extreme programming, begin with the simplest possible design, knowing that later iterations will make them more complex. Do not add in early functionality at this stage to keep it as bare bones as possible.

XP methodology teams will often use class-responsibility-collaboration (CRC) cards to show how each object in the design interacts. By filling out each field in the card, you’ll get a visual interaction of all the functions as they relate and interact. CRC cards include:

* Class (collection of similar objects)
* Responsibilities (related to the class)
* Collaborators (class that interacts with this one)

CRCs are useful for stimulating the process and spotting potential problems. Regardless of how you design, you’ll want to use a system that reduces potential bottlenecks. To do this, be sure you’re proactively looking for risks. As soon as a potential threat emerges, assign one to two team members to find a solution in the event that the threat takes place.

[Read: The project risk management process in 6 clear steps](https://asana.com/resources/project-risk-management-process)

4. Coding

One of the more unique aspects of XP is that you’ll stay in constant contact with the customer throughout the coding process. This partnership ‌allows you to test and incorporate feedback within each iteration, instead of waiting until the end of a sprint. But coding rules are fairly strict in XP. Some of these rules include:

* All code must meet coding standards.
* Using a unit test to nail down requirements and develop all aspects of the project.
* Programming as a pair—two developers work together simultaneously on the same computer. This doesn’t add any time, but rather uses double the focus to produce the highest quality results.
* Use continuous integrations to add new code and immediately test it.
* Only one pair can update code at any given time to reduce errors.
* Collective code ownership—any member of the team can change your code at any time.

5. Testing

You should be testing throughout the extreme programming process. All code will need to pass unit tests before it’s released. If you discover bugs during these tests, you’ll create new, additional tests to fix them. Later on‌, you’ll configure the same [user story](https://asana.com/resources/user-stories) you’ve been working on into an acceptance test. During this test, the customer reviews the results to see how well you translate the user story into the product.

What are the 12 XP extreme programming practices**?**

To further hone the process, XP also uses a set of 12 practices throughout the process. They are based on the[Agile manifesto](https://agilemanifesto.org/), but adapted to fit XP needs:

1. The planning game: XP planning is used to guide the work. The results of planning should be what you’re hoping to accomplish and by when, and what you’ll do next.
2. Customer tests: When you finish a new feature, the customer will develop an acceptance test to determine how close it is to their original user story.
3. Small releases: XP uses small, routine releases to gain insights throughout the process. Often, releases go straight to the customers, though they can happen in-house.
4. Simple design: The XP system is designed for simplicity—you’ll produce only what is necessary and nothing more.
5. Pair programming: All programming comes from a pair of developers who sit side by side. There is no solo work in extreme programming.
6. Test-driven development (TDD): XP’s reliance on feedback requires heavy testing. Through short cycles, programmers release automated tests and then immediately react.
7. Refactoring: This is where you’ll pay attention to the finer details of the codebase, removing duplicates and making sure that the code is cohesive. This results in good, simple designs.
8. Collective ownership: Any coding pair can change the code at any time, whether or not they developed it. XP produces code as a team, and everyone’s work is held to the higher collective standards.
9. Continuous integration: XP teams don’t wait for completed iterations, they integrate constantly. Often, an XP team will integrate multiple times a day.
10. Sustainable pace: The intensity of XP works requires you to set a sustainable pace. Teams should decide how much work they can produce in this way per day and per week, and use that to set work deadlines.
11. Metaphor: The metaphor is, quite literally, a metaphor. It’s decided as a team, and uses language to describe how the team should function. For example, we’re ants working as a collective to build up the anthill.
12. Coding standard: XP teams follow one standard. In the same way that writers need to take on a brand’s voice to sound like the same person is always writing, XP developers code in the same, unified way so that it reads like one developer.

**2 ROUND**

Here is the translation:

I am a **SAFe® Product Owner/Product Manager** and **Scrum Master Certified (SMC™)** with a strong background in **Automation and Machine Learning**. Currently, I work as an **Artificial Intelligence and Machine Learning Engineer** at Western Union LAROC, where I build and automate high-quality data science pipelines and manage the entire ML lifecycle.

Previously, I was the **Director of the Technology Academy at CIT-ULACIT**, overseeing all activities related to the Technology Academy. I also managed **Robotic and Intelligent Automation at LAROC** for Western Union, supporting global RPA operations and developing data analytics solutions.

My technical expertise includes **Java, C#, Python, Data Analytics with Power BI, Kibana, Grafana, Snowflake, and SQL-Oracle databases**. I have a proven track record in design and integration problem-solving, and I am proficient in various technologies such as **AWS, Cloudbees, Splunk, Dynatrace, Blue Prism, UiPath, and more**.

Throughout my career, I have demonstrated my ability to analyze technical requirements, design workflows, and collaborate with business analysts to develop effective solutions. I also have experience in training and coaching teams, ensuring the successful implementation of automation and data science projects.

Feel free to adjust or add any details to better match your personal style and preferences!