


Flexiana

What is the role of a Product Owner



Product Owner/Product Manager is
a person who is responsible for
steering the product.

The main activities of a Product Owner



Product Analysis

- Understanding product users and their personas
- Understanding the goals people want to achieve with the product
- Learning from competitors and other relevant products
- Checking current trends in the market
- Checking how people use the current version of the product
- Prototyping (often together with a designer)



Product backlog maintenance

- Writing tasks for the team
- Prioritization of tasks
- Splitting large tasks/epics into smaller ones
- Aligning product development budget



Communication with the team

- Explaining tasks to the team
- Reviewing implementation and providing feedback
- Discussing how difficult and pricey is it to develop certain features

Topics covered in this guide

- Product Personas
- What you must know about your users?
- Backlogs
- The Kano model
- Prioritization
- User Stories
- User Story Splitting
- Product analytics tools
- Product management tools
- Acceptance criteria
- Product demos
- Product demos
- PO role for internal systems
- PO role for product startups
- Product-led growth & PO role here



Product Personas

Product personas serve as depictions of various user roles within your product. These representations can range from highly abstract to skillfully detailed, incorporating demographic information of the users. Additionally, I've encountered personas crafted from pseudonymized real individuals as well.

Your product is likely to have multiple personas ranging from individuals without or with the lowest system access level, as well as administrators, etc.

Personally, I find little value in including details such as hobbies, fictitious photos and fake names in personas. In my persona creation process, **I concentrate solely on relevant information for the product, such as the individual's role within the company, department, etc.**

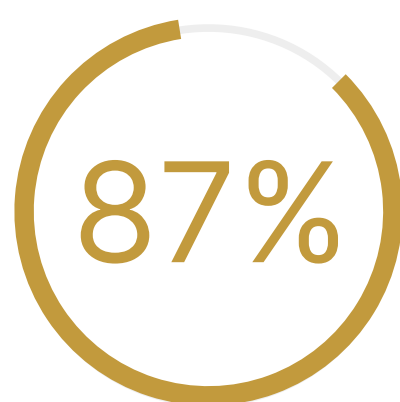
I often tend to name personas based on their most significant attribute; for instance, if a user holds an administrative role, the persona will be named Administrator, not Murray Jones.

What you must know about your users?

Each user has certain purposes to use your product. Some of these are more important than others (we will speak more about prioritization later on). Some of them might have overlaps with tools that are already available, while others might only hold significance during specific stages.

For example, based on their requirements, administrators may perform certain tasks only during the initial app installation and not revisit them, and accountants may require specific features only once a year.

Additionally, some features hold importance for the product provider but may not be directly relevant to the customer, such as the need for customers to make payments versus the provider's need to monitor the app's operational status.



Each of these individuals has tasks they need to accomplish.

As the Product Owner, you are responsible for knowing about ALL of your users' tasks. You must understand every user persona, their motivations, and situations.



Backlogs

All improvements in the product must be managed. Currently, the most widespread approach is done via backlogs.



Product Backlog

The backlog of **all things that the product should contain in the future**, although the future might be extended to years (which isn't always recommended).



Sprint Backlog

Backlog of **things we picked to work on right now**.

Teams usually pull top tasks from the product backlog to their sprint backlog to work on. The tasks from the product backlog are usually large. To manage them effectively, teams split these into smaller pieces and put them into their sprint backlogs.

It's also the best practice to have some estimates on tasks for the Product Owner to determine when the tasks will be worked on.

The accuracy of our predictions decreases as we look further into the backlog. At the start, we can pinpoint specific weeks for tasks. After a few weeks, we switch to estimating in months. Several months ahead, we may be able to estimate within a quarter.

Please keep in mind that backlog is often reprioritized and changed with tasks added and removed.

The Kano Model

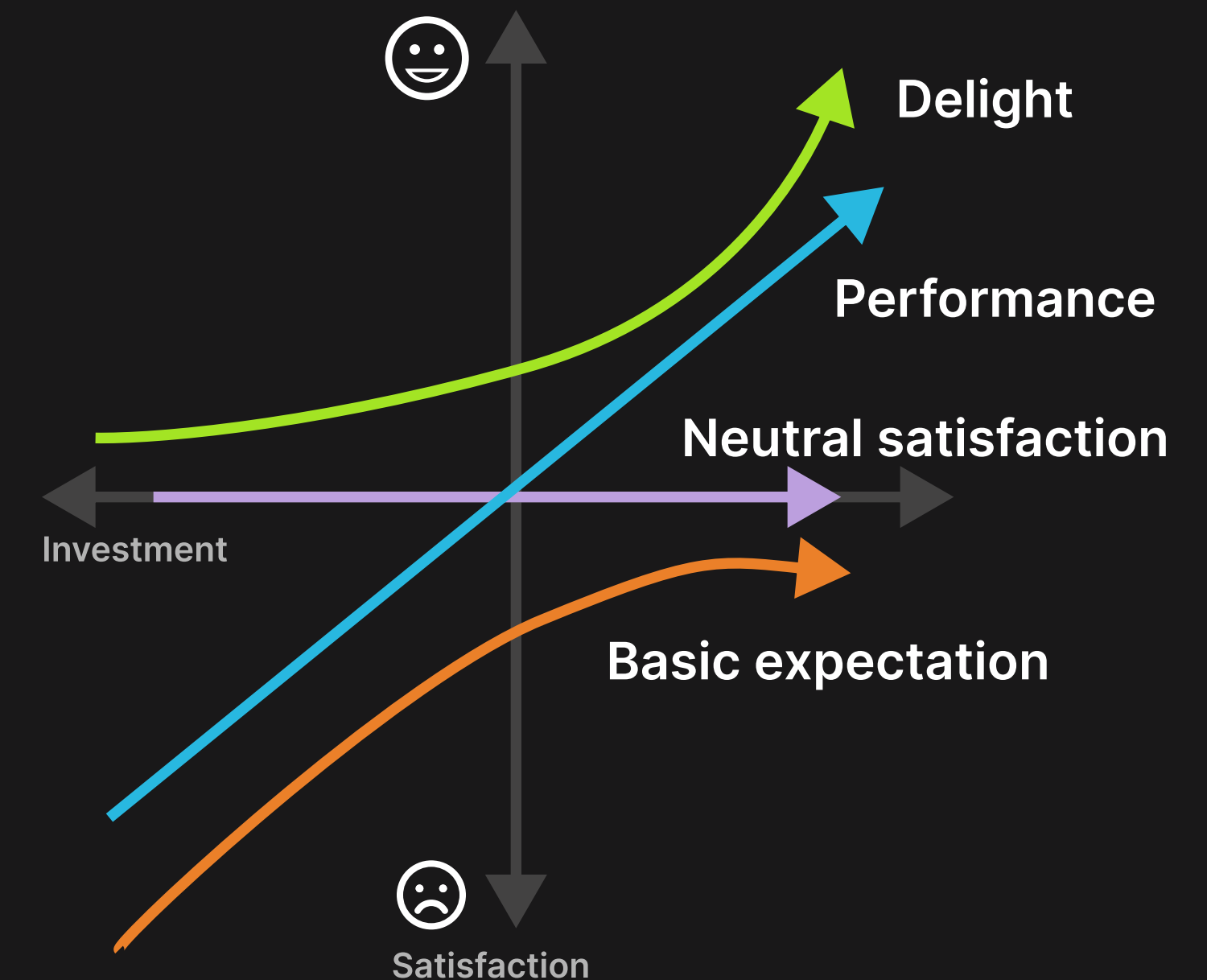
Every Product Owner needs to prioritize tasks, and there are different approaches to doing so. Some approaches are more systematic, while others rely on domain knowledge and situational awareness. I would say the Kano model falls in the middle, as it can be used based on tons of data, user interviews, and proper feedback. It is also effective in the hands of a knowledgeable Product Owner familiar with their market.

This is why we at Flexiana often use the Kano model to prioritize.

Basics of the Kano model

The goal is to categorize all tasks based on their importance.

1. **Various users would categorize different tasks differently**, but in general, one can say, there is a limited amount of personas in each product, and people within the same persona will share similarities.
2. So the point is to **recognize which tasks are 'Must have' for at least one persona and put those on the top** of the backlog.
3. Then find tasks that are 'Performance' for at least one persona and put them second.
4. Later find tasks, that are 'Delighters' for at least one persona and put them third.
5. **Delete all 'Indifferent' tasks.**
6. **Identify all 'Detractors' and update the app to minimize negative impact**, especially if the features are necessary for one persona but disliked by another. Sometimes a feature might be a detractor for one persona but valuable for others. When this happens, you should update the backlog in a way that the persona who needs it will get it while the persona who dislikes it has its impact minimized.



The Kano Model

Categorization of each task can be easily done by asking two questions:

How would you feel if this feature was missing?

How would you feel if this feature was implemented?

Possible Answers

The goal is to categorize all tasks based on their importance.

1. I cannot use this product. I expect it. → MUST HAVE
2. It would hurt my ability to use this product. It would help me if this was present. → PERFORMANCE
3. I don't mind if it isn't implemented. But it can help me when it is. → DELIGHTER
4. I don't mind when this isn't implemented. I don't care if this is implemented. → INDIFFERENT
5. I don't mind when this isn't implemented. I dislike it if it is implemented. → DETRACTOR

Please note, you need to ask this question from each persona, and if it is a MUST HAVE for at least one persona, the story is in general a MUST HAVE for the whole product.

The second important thing is, that expectations of people change over time. In general, people expect more features and better functionalities. They expect the app not only to work on their desktop but also on their mobile phone and tablet. At the same time, users stop using certain features. Therefore, the Kano model evaluation changes over time.

Examples:

- An App always online, accessible from tablets and mobile phones is a MUST HAVE
- The ability to use applications in MS-DOS is an INDIFFERENT

In 1990, it would be the opposite.

Apps that can be used in virtual reality are INDIFFERENT in 2024 for most apps.

It might be a MUST-HAVE in 2050.

Prioritization

The Kano model is helpful in prioritization but there's more to that.

Examples:

1. You need to prioritize tasks within one group. How will you decide which **MUST HAVE** is the most important one? How will you do the same for the **PERFORMANCE** tasks?
2. You need to be able to fit tasks into one iteration/release. You might pick a couple of smaller tasks that fit into your sprint.
3. There's added value in doing tasks that make one whole flow because, after the release, people can immediately start using it.
4. There might be tasks that are highly uncertain and need research, whereas multiple variations and other tasks depend on the way you implement them.
5. The customer wants to see certain features earlier even when it is not that crucial for the users of the system.

There's no one way how to do prioritization, but let me share some of the best practices:

- If the customer pays for it, let him/her choose what they would prefer to prioritize.
- Tasks that are depended on should be on top.
- Tasks that have a lot of uncertainty should be on top.
- Tasks that form the whole flow/process/business value should be together.

If you use the Kano model and these four rules, you should be on the safe side.



User Stories

The primary method for describing work to developers is typically through user stories. The user story should describe the "who" (is using the product), the "what" (the user wants to achieve), and the "why" (this is important for the user).

User stories should be short, simple, and easy to understand. They are not meant to be detailed technical specifications, but rather a high-level description of what needs to be done.

Besides the classic who, what, and why, user stories often contain additional information, definitions of done (criteria without which the task will be automatically rejected), and acceptance criteria.

The definition of 'done' can be partly defined by the team and include best practices that the team believes in like acceptance tests, unit tests, etc

Acceptance criteria are usually in a so-called [Gherkin format](#), that focuses on three parts:

1. What is the initial situation (Starting with the given format)
2. What is the action (When section)
3. What is the outcome (Then section)

User stories often have 1-5 acceptance criteria.

Example of a user story:

As a user, I want the ability to change my password so that I can ensure the security of my account.

Given user is logged in

When he clicks on "Account settings"

And then "Change password"

Then he should be prompted for the current and new password

Given user is on the change password page

And has submitted the correct passwords

When he submits the form

Then his password is changed and accepted

Definition of 'done':

- Unit tests for password change functionality are written and passed.
- Integration tests for password change functionality are written and passed.
- The documentation page describing the change of password is updated.

The team might add their own notes like estimates, implementation details, etc. If there is an impact on the UI of the application (most of the time, there will be), changes in user stories in the given sprint should be visible and accessible to the team.

User Story Splitting

Each product backlog and sprint backlog should have upper thresholds on the granularity of tasks in it. If the task is too big, the Product Owner (sometimes with the help of the team, business analysts, designers, and customers) should split the backlog items into smaller segments.

Let's say a good granularity for software development is to target tasks to be doable in one day, and never allow tasks that are larger than three days.

There are many ways to split tasks

Let me present a couple of them:



Isolate core functionality and add extensions:

An e-shop checkout might support just one way of payment and just one mode of delivery service. Alternative payment methods and delivery services are considered as additional user stories.



Split by the user rights:

At first, you describe how certain screens are used by one role, later on, you add the behavior of a second role. For example, in Internet banking, you will make an interface for users who can pay and then you simplify the interface for users who can only view transactions.



Split tasks by steps in a flow:

The CRM import of contacts might read all contacts from the CSV file and then show the validation page. Afterwards, it will show what records to import, and then the ability to undo this import.



Split by UI details:

Initially, you might just print some data as a simple table. Subsequently, you can add a fancy data grid later on.

Product Owner tools

The Analytics tools

The Analytics tools are essential for a Product Owner to effectively measure and understand the usage of their product. They provide valuable insights into user behavior and engagement, enabling data-driven decision-making and effective prioritization of product features or changes.

There are several powerful analytics tools a Product Owner might employ:

- 1. Google Analytics:** This is a widely used tool that offers insights into website traffic and user behavior. It can track metrics such as page views, bounce rate, and conversion rates. It can also provide demographic information about your users.
- 2. Mixpanel:** This tool provides a more in-depth analysis of user behavior on your site or app. With Mixpanel, you can track events, create funnels to understand the user journey and segment your audience based on their behavior.
- 3. Amplitude:** Amplitude provides product analytics that helps teams convert, engage, and retain customers. It provides insights into how users are interacting with your product and can help identify areas for improvement.
- 4. Hotjar:** Hotjar offers heat maps and session recordings to visually represent user behavior. This can be particularly useful for understanding how users are interacting with individual elements on a page.

- 5. AB Tasty:** This is a conversion rate optimization tool that lets you perform A/B tests on your website or app. This can be particularly useful for trialing new features and making evidence-based decisions about which version to implement.
- 6. UserTesting.com:** This tool allows you to gather user feedback and insights through remote usability testing. It can be used to get feedback on specific features or overall user experience.
- 7. Product board:** This is a product management platform that helps Product Owners prioritize features, gather feedback, and track progress. It can be particularly useful for coordinating cross-functional teams and managing multiple products.
- 8. Hotjar:** Hotjar offers heat maps and session recordings to visually represent user behavior. This can be particularly useful for understanding how users are interacting with individual elements on a page.

Each of these tools offers unique capabilities and insights and can be powerful when used in combination. As a Product Owner, choosing the right analytic tools will depend on the specific needs and goals of your product.

To learn more about product analysis, I recommend the following book that I frequently review: [Testing Business Ideas](#).

Product management tools

Once the Product Owner decides what needs to be done, s/he must communicate this to the team. Several tools can be used for this purpose, as outlined below.

- 1. Trello:** A versatile project management tool that allows you to organize and track tasks, collaborate with team members, and visualize project progress through boards and cards. It's a great default option if you need help figuring out where to start.
- 2. JIRA:** A comprehensive issue-tracking and project management software that helps teams plan, track, and release software with advanced features like agile boards, customizable workflows, and powerful reporting. In general, I wouldn't use JIRA, but it is very successful at some large companies.
- 3. Linear:** A modern and intuitive issue-tracking tool designed for software development teams, offering a streamlined workflow, prioritization features, and integrations with popular development tools. This tool is opinionated but also has very good usability. If their views on how development should be done are compatible with yours, it might be great.
- 4. Notion:** A flexible all-in-one workspace that combines note-taking, project management, and collaboration features. With its customizable and modular structure, Notion allows you to create and organize tasks, documents, and databases in one place. We use Notion a lot at Flexiana, as it is a great and very versatile tool.
- 5. GitHub Issues:** A built-in issue tracking feature within the popular development platform GitHub. It enables teams to manage and track software bugs, feature requests, and tasks directly within their code repositories, supporting collaboration between developers. This is also a very common Product Owner tool at Flexiana. The main requirement is that all people involved in development should be tech-savvy.

Common characteristics of these tools are:

Once the Product Owner decides what needs to be done, s/he must communicate this to the team. Several tools can be used for this purpose, as outlined below.



Collaborative:

These tools allow for team collaboration and communication in one central location.



Integrations:

They offer integrations with other tools, making it easier to manage all aspects of a project in one place.



Customizable:

Many of these tools have customizable features to fit the specific needs and preferences of different teams or products



Visualization:

Most of these tools provide visual representations of tasks and progress, making it easier to understand and communicate the project status.

One of our internal products is AIGEN and it's a tool for Product Owners. It's a bit too early, but once we release it, it will become a valuable example of a great PO tool.

Acceptance criteria

Acceptance criteria are defined as a set of conditions that must be met for a feature or user story to be considered complete and functioning as intended. These criteria are typically written in collaboration between the customer, the Product Owner, and the development team. They serve as a guideline for all parties to ensure that expectations are clear.

Here are some of the best practices to keep in mind when writing acceptance criteria:

- **Clearly define the expected outcome:** Acceptance criteria should clearly state what the feature or user story is meant to accomplish.
- **Be specific:** Avoid vague terms and prioritize concrete, measurable language. This will help avoid confusion and ensure everyone is on the same page.
- **Keep it concise:** Acceptance criteria should be brief and to the point, focusing on the essential details rather than getting bogged down in unnecessary specifics.
- **Testability:** The criteria should be written in a way that makes it possible to test the feature or user story. This means avoiding ambiguous language and ensuring all requirements are clear and actionable.
- **Prioritize functionality over design:** While design elements may be important, they should not be included in Acceptance criteria unless necessary. The key focus should be on the core functionality of the product.

A good way to define Acceptance criteria is the Gherkin language.

The following example in the User Stories section gives more clarity on Acceptance criteria.

User Story:

The User can import his contacts to CRM in CSV format.

Acceptance criteria:

The given user has a valid CSV file

When the user selects the file

And clicks Upload

Then a screen with imported data is shown

The given user has an invalid CSV file

When the user selects the file

And clicks Upload

Then the user is redirected back

And the user can see validation errors

The given user is on the Imported data page

When the user clicks on Import

Then records are imported into the database

The given user is on the Imported data page

When the user clicks on Cancel

Then the user is redirected back to the import page

Acceptance test-driven development

Acceptance Test-Driven Development, or ATDD, is an agile technique for ensuring that the software meets its intended functionality by **defining acceptance criteria and writing tests before implementing the code**. This process consists of three stages: defining the requirements (User Story), creating acceptance criteria (likely in the Gherkin format) based on those requirements, and writing tests to validate the criteria.

The benefit of this approach is that it helps ensure that the software is built to meet the customer's needs and expectations. It additionally offers a clear understanding of what defines a successful outcome. It also allows for early detection of any issues or discrepancies in the requirements, reducing the risk of costly changes later on

These tests are similar to any other end-to-end tests that are focused on the whole product.

Some of the best practices for implementing ATDD include:

- Involving all team members: ATDD should be a collaborative effort involving the Product Owner, developers, and testers.
- Writing tests in a simple, understandable language: Tests should be written using Gherkin or similar tools to ensure that they are easily understood by all team members, regardless of technical expertise.
- Creating a shared understanding: Acceptance criteria and tests should serve as a common understanding for all team members of what needs to be built and how it should function.
- Continuously refining and updating acceptance criteria: Acceptance criteria should be continuously reviewed and updated as needed to ensure they accurately reflect the customer's needs. At the same time, the test catalog functions as a database containing the application's various functionalities.

By using ATDD, teams can ensure that their software meets the customer's expectations while also improving communication and collaboration within the team.

Sadly, ATDD is not a widespread technique yet. However, as a Product Owner, recognizing the value of ATDD and advocating for its incorporation into your team's development process is essential if you see its advantages.

Acceptance process

As a Product Owner, one crucial responsibility involves daily acceptance of completed tasks, often user stories, sometimes multiple times a day. This is one of my real-world findings, that quick acceptance has a very positive impact on productivity. However, acceptance isn't a mere "Yes, it is done"; it's a process that involves thorough review and testing of the completed work to ensure that it meets the defined acceptance criteria.

Here are some general steps to follow when performing acceptance:

1. Review the user story: Before beginning the acceptance, make sure you have a clear understanding of the user story and its acceptance criteria. If there are any questions or concerns, discuss them with the development team.
2. Check for completeness: Ensure that all aspects of the user story have been implemented, including any necessary design elements and functionality.
3. Test according to acceptance criteria: Use the defined acceptance criteria as a guide for testing the completed work. Make sure that all expected outcomes are met.
4. Document any issues or bugs: If there are any issues or bugs found during testing, document them and discuss them with the development team for resolution.
5. Provide feedback: Once acceptance is complete, provide feedback to the development team. This can include any necessary changes or improvements that need to be made before the final release.
6. Track progress: Keep track of the acceptance process and any issues or concerns that arise. This will help with future planning and prioritization.

Of course, in the real world, the process is more informal and faster. You review the user story, open the development version of the app, and conduct testing swiftly.

You, as a Product Owner, have the right to refuse the User Story for any reason. For instance, you may have found an edge case that wasn't specified in the acceptance criteria, or the functionality is not working as expected. In this case, communicate with the development team and provide feedback on what needs to be improved before accepting the user story.

Product demos

Product demos are an important part of the development process, as they allow stakeholders and team members to see the progress being made on the product. As a Product Owner, it's important to plan and conduct these demos effectively to gather feedback and make necessary adjustments.

You may be part of the team presenting the demo to customers and stakeholders, which is the usual scenario. However, occasionally, the roles are reversed, and the team conducts a demo for the Product Owner.

Here are some tips for your product demos:

- **Prepare beforehand:** Make sure you have a clear agenda for the demo and that all the necessary materials are ready beforehand. Have data in the system, and test all flows to ensure the system is not failing (more common than you would expect).
- **Encourage feedback:** Ask for feedback and actively listen to suggestions or concerns from stakeholders and team members.
- **Highlight key features:** Take time to showcase the most important or impactful features of the product, as well as any changes or updates since the last demo.
- **Provide a live demonstration:** Whenever possible, do a live demo of the product to show its functionality and capabilities in action.
- **Address concerns immediately:** If there are any concerns or questions raised during the demo, address them immediately and provide reassurance or clarification if needed. Take note of any feedback for future improvements.
- **Ask for priorities, if relevant:** If multiple features or changes are being presented, ask for input on which should be prioritized for development.



Product demos are a valuable opportunity to gather feedback and ensure that the product is meeting the needs of stakeholders and customers. By following these practices, you can conduct successful and effective product demos that contribute to the overall success

Product Owner's role in internal systems

As a Product Owner, your role may extend beyond just the development of customer-facing products. You may also be responsible for managing the development of internal systems and tools that are necessary for the smooth operation of your organization.

This can include anything from ERP, custom internal systems, or customization of off-the-shelf systems.

This role frequently involves negotiating with stakeholders, and clarifying how changes affect various teams and internal systems. It bears resemblance to the responsibilities of a Business Analyst. Your primary focus is to comprehend the requirements of different departments and teams within the organization and prioritize development accordingly.

Some tips for managing internal systems as a Product Owner include:

- **Understand the needs of each department:** Learn how people in different departments work, what are their responsibilities, and general tech capabilities.
- **Prioritize based on impact:** Just as with customer-facing products, prioritize the development of internal systems based on their impact and significance to the organization. Frequently, you'll receive goals from higher-level individuals that guide your prioritization efforts.
- **Communicate changes effectively:** When changes or updates are implemented in internal systems, ensure clear communication to all departments. Offer support for any required training or adjustments. Additionally, you may be tasked with organizing training sessions for users.

- **Regularly gather feedback:** Software cannot be developed in the dark. When you and your team lack direct interaction with end users, it's crucial to actively seek connection with those using the systems. Gather feedback and suggestions from different departments on ways internal systems can be enhanced or optimized.
- **Collaborate with IT department/ops:** Work closely with the IT department to ensure that all internal systems are ready for deployment. Ensuring that all required integrations or connections are in place.
- **Stay updated on industry trends:** Keep an eye on the latest developments and trends in internal systems to ensure that your organization is using the most efficient and up-to-date technology. You might be asked to research appropriate solutions if budgets allow for it. This can also help with identifying areas for improvement or potential upgrades in the future. The [Nielsen Intranet Design](#) annuals is a valuable source to help in this matter.

Managing internal systems can be just as important and challenging as managing customer-facing products. To excel as a Product Owner for internal systems, understand departmental needs, prioritize development, and stay current on industry trends.

Product Owner's role for product startups

Product Owners in startups are often focused on figuring out what changes in the product are needed to deliver something people will pay for. Their role is crucial in the early stages of the product development process. They must balance limited resources and prioritize features that will have the most impact on potential customers accordingly.

Some tips for Product Owners in startups include:

- **Focus on customer needs:** In a startup, it's important to understand the needs and pain points of your target audience. Use this knowledge to guide product decisions and prioritize features that will solve real problems for potential customers.
- **Be agile:** Startups often must operate with limited resources and time constraints, so it's important to be adaptable and able to pivot quickly if needed. This may require being flexible with the product roadmap and making changes on the fly.
- **Collaborate closely with the development team:** As a Product Owner in a startup, you may have a smaller team or work closely with the development team. It's important to communicate effectively and collaborate closely to ensure that resources are being used efficiently and the product is being developed most effectively.
- **Stay on top of the market:** In a startup, it's important to stay updated on industry trends and competitor products. This can help inform product decisions and ensure that your product remains competitive in the market.



Usually, in the early stage of a startup's existence, its CEO is also a Product Owner. This role may be shifted to another person, as the startup grows and requires more focused attention on product development. However, the Product Owner must work closely with the CEO and align product decisions with the overall goals and vision of the company.

Product-Led Growth & Product Owner's Role

A product-led company prioritizes product value as the key driver for customer acquisition, retention, and expansion. In comparison with traditional sales-led or marketing-led approaches, in product-led growth (PLG) models, the product itself is the main vehicle for customer engagement and business growth. This approach relies heavily on delivering a superior user experience that solves real problems effectively. This fosters organic growth by ensuring high customer satisfaction and encouraging word-of-mouth referrals.

Essentially, in a product-led company, the product speaks for itself. The focus shifts from 'selling' to 'serving'. Emphasis is placed on crafting a product so valuable and user-friendly that customers naturally use it, advocate for it, and remain loyal.

In a product-led company, the role of the Product Owner becomes increasingly vital. Product Owners are at the forefront of understanding user needs and translating those into the product's features. They are responsible for creating and maintaining a product that not only meets customer expectations but exceeds them. This has an immediate impact on user satisfaction and, therefore, product-led growth.

Apart from comprehending client needs and prioritizing advocacy, in a PLG model, the Product Owner's role includes;

- Collaborative work with the product design and development teams to ensure that the product's usability and functionality align with the customers' needs.
- An ongoing analysis of product metrics to measure user engagement and satisfaction, which directly influences growth.

Let me stress here, that if product quality is crucial, so is the role of the Product Owner.

Another key area of focus is Time to Value optimization for users, especially new users. A user should be able to find value in your product as quickly and effortlessly as possible.

The continuous improvement of the product's ease of use is critical. It enables them to independently self-serve and onboard with minimal assistance from support or sales teams. This speeds up the customer journey and accelerates product growth.

In a product-led company, the Product Owner's role is pivotal in driving growth and success. They are responsible for creating a user-centric, intuitive product that solves real problems effectively and creates natural promoters out of customers.

When focusing on product-led growth, another relevant aspect is virality, word of mouth, and users sharing the product for the benefit of others. This organic growth is one of the key benefits of a product-led model and can significantly impact your product's success. As a Product Owner, it is essential to consider these factors in your product strategy and continuously drive towards creating a product that users genuinely love and want to share with others. Ultimately, this will lead to sustainable growth and continued success for the product.

Another common thing in PLG is the trial/freemium model or free trial. The Product Owner's role here should include closely monitoring the product usage and adoption during trials. They should focus on analyzing user feedback to make necessary improvements and ensure that the free version delivers enough value to drive conversion to paid subscriptions. A crucial aspect of this is continually measuring customer satisfaction and making data-driven decisions on pricing, packaging, and feature offerings for different tiers of subscription plans (if your product is SaaS).

The success of product-led growth is impossible without a profound understanding of how users engage with the product, including their experiences during registration and onboarding. This is where the Product Owner's role becomes essential in creating a seamless, user-friendly onboarding process that leads to immediate value for the users. By continuously analyzing and optimizing this process, Product Owners can drive increased adoption and retention rates, ultimately contributing to product-led growth.

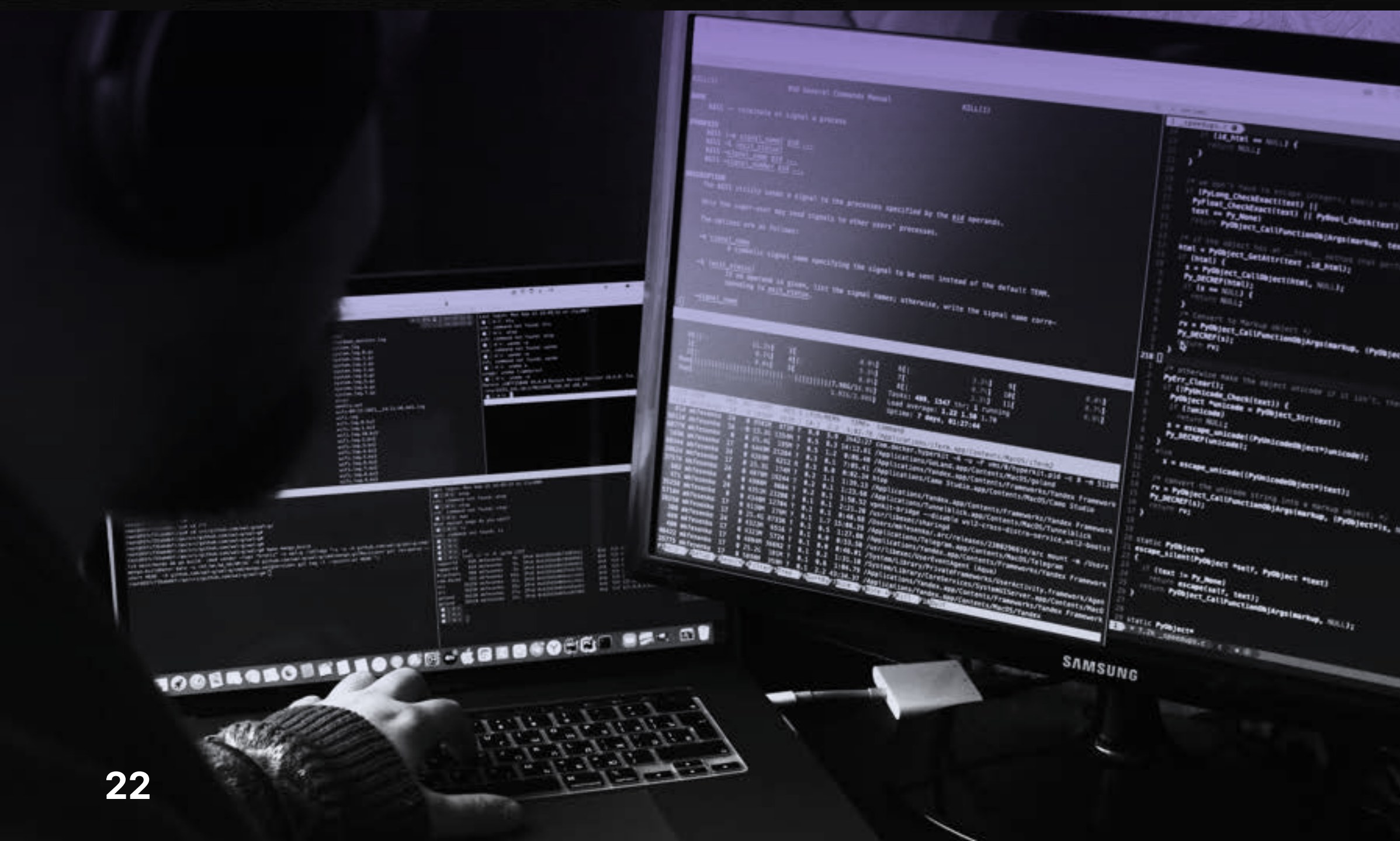
In conclusion, the role of a Product Owner in a product-led company encompasses being a customer advocate, a product strategist, and an analytical thinker. Product-led growth is not just a trend; it is a mindset shift that puts the customer at the center of all business decisions. By prioritizing user needs and focusing on customer feedback, Product Owners can foster organic growth for sustained product success.





About Flexiana

Flexiana is a software company specializing in digital services for businesses. Its portfolio includes full-cycle applications, web and mobile applications, and customized software solutions Guided by core principles including craftsmanship, transparency, autonomy, diversity, remote collaboration, and agility, Flexiana seamlessly integrates these values into its daily operations, ensuring a client-centric approach and unparalleled results.





Jiri Knesl, CEO

Jiri's started doing SW development 29 years ago. Before starting Flexiana, he worked in Business & IT management consulting for 9 years. Over his life, he published over 400 articles and presented on dozens of conferences and in more than 50 companies. His hobbies are learning, traveling & mainly time with his family.



Flexiana

Let's work together

Let's explore how we can assist in building or managing your product.

Website:

flexiana.com

Email:

hello@flexiana.com

Linkedin

[Linkedin](#)