Case Study: My SaaS Experience – Dynamic (Predefined) Packages Module on Juniper Booking Engine

Project Overview

In this document, I describe my experience acting as a **local Product Owner** for the **Dynamic Packaging (DP) module**, a component of the Juniper Booking Engine — a modular B2B **SaaS platform** for travel operators. This project aimed to deliver a unique and flexible travel packaging solution for a newly created Spanish tour operator, positioning us as a first mover in a competitive segment.

Unlike standard Hotel+Flight dynamic packages, our concept was to build **predefined, flexible templates** that could include multiple travel service units — both static and dynamic (via API). The system needed to support advanced inventory logic, fast performance, and a highly customizable UI.

My Role and Responsibilities

- Defined the initial vision and scope of the Dynamic Packaging module
- Translated business needs into functional specifications and user stories
- Created activity diagrams and DFDs using **Enterprise Architect** to clarify backend workflows
- Participated in prototyping sessions with Juniper's project manager and development team
- Acted as the key liaison between internal users, my manager (who was also the final investment decision-maker), and Juniper's CEO and product leads
- Validated MVP and coordinated stakeholder feedback post-launch

Key Features Defined

- **Package Template Management**: Create reusable product templates with custom inventory units
- **Hybrid Inventory**: Combine static content (manual) and dynamic sources (via API)
- **Custom Price Rules**: Flexible pricing logic based on package structure and supplier rates
- **Multi-step Booking Flow**: A clear and user-friendly front-end design for partner agencies

Project Challenges

- Misalignment between internal expectations and MVP technical maturity

- Limitations of customizing a B2B SaaS product already in production: the DP module, while technically launched, was still under heavy development and not designed for deep client-specific modification. As a result, our requirements for advanced logic, template control, and interface flexibility often fell outside the vendor's delivery model or roadmap.
- Post-launch issues with **search performance** and **system stability**, resulting in frequent crashes
- Tight development timeline with limited testing buffer

Outcome

- MVP was launched but failed to meet performance goals within the first 3–4 months
- Internal adoption was limited due to instability in real usage scenarios
- My manager, the shareholder, decided to halt further development and seek alternative platforms
- A different vendor was eventually selected and acquired for future use

Despite the outcome, this project provided significant experience in:

- Designing a modular SaaS product feature set
- Managing vendor collaboration under a time-and-materials delivery model
- Navigating stakeholder pressure vs. product quality realities

Stakeholder Collaboration

- I established the coordination between the Juniper Project Manager, our internal key users, and my manager, who was responsible for final decisions regarding investment, and whether to continue or terminate the project.
- I regularly aligned the product roadmap with my manager and updated him on delivery progress, technical issues, and goal achievement.
- I also communicated with Juniper's global Product Manager align my roadmap expectations with Juniper's internal priorities and delivery capabilities.
- JPM handled coordination with the Juniper engineering team and also gathered feedback directly from users during testing. My role ensured that feedback, expectations, and risks were fully communicated on both sides.

Development & Collaboration Model

- Juniper's development followed an **incremental, ticket-based model** under a **time-and-materials agreement** (hourly billing per request).
- There was **no formal Agile** structure, but we used **prototyping** and **user-driven feedback loops** to clarify requirements and adjust solutions.
- The lack of a structured release process or CI/CD pipeline introduced inefficiencies in **updates, integration, and delivery reliability**.

- I identified that **more formal Agile or DevOps practices** on the vendor side could have significantly improved delivery predictability and product evolution.

Methodology and Tools

- **Process Model**: Incremental, ticket-based development with prototyping support
- **Budgeting Model**: Time-and-materials (hourly billing per ticket)
- **Techniques and Tools Used**:
    - **Juniper's internal ticketing system** – for submitting and tracking development/configuration tasks
    - **Wireframes** – to visualize user interfaces and clarify workflows for both local teams and developers
    - **Activity diagrams** and **DFDs** – created in **Enterprise Architect** to document backend logic and package flow
    - **ProjectLibre** – for internal roadmap planning, aligned with go-live decision points
    - **Spreadsheets and test plans** – to define pricing rules, package logic, and test coverage
    - **Live demos and validation sessions** – to gather structured feedback from stakeholders and internal users

Lessons Learned

- Launching innovative SaaS products requires clear performance thresholds, not just functional delivery
- Realistic roadmap pacing and internal piloting are essential when MVP is business-critical
- Even client-side POs can have a deep impact on vendor roadmaps through structured feedback and testing involvement
- Working under a time-and-materials model without Agile or DevOps practices led to unstable deliveries, frequent rollbacks, and increased uncertainty in both scope and quality
- The absence of automated testing and version control hindered stable releases; a modular architecture allowing independent deployment of the DP module could have reduced system-wide risk and improved iteration speed.
- SaaS products delivered to the market typically have structural limitations when it comes to deep customization. In our case, the need for highly unique functionality exceeded what the vendor was ready to support within the standard product roadmap, which caused misalignment and friction throughout the development cycle.

SaaS Buyer Insight: Customization vs. Extensibility

This project highlighted a common but critical challenge in SaaS implementation: balancing standard functionality with the need for business-specific innovation. The Juniper DP module, like many SaaS products, offered strong baseline capabilities but had limited flexibility for deep

customization. Our project required unique logic and user flows that exceeded the vendor's willingness or ability to support under their standard roadmap.

Key takeaway:
SaaS solutions should be evaluated not just on existing features, but on their **extensibility models** — such as plugin support, API layers, or custom workflows — especially when innovation or differentiation is part of the strategy.

Appendix: SaaS Customization vs. Standardization – Practical Insights

Overview

This appendix expands on the challenges faced during the Dynamic Packaging project, offering a broader industry perspective on the tension between standardized SaaS delivery and the need for deep client-specific customization. It reflects not only on what happened in the project, but also what I learned about SaaS platform design, vendor models, and extensibility strategies.

The Core Tension

SaaS platforms succeed by offering a standardized product to many clients. However, real-world business models often require customization that goes beyond what the vendor is ready or able to support.

In our project, we needed advanced packaging flexibility and performance guarantees that exceeded the core product scope. The vendor was not prepared to develop or support these features within the agreed budget, timeline, or architectural constraints.

How the Industry Manages This

1. Configuration Over Customization

Principle: Offer flexible configuration tools so that many use cases can be supported *without* touching the source code.

Examples:

- Rule engines (pricing, logic)
- Modular workflows
- Widget-based UI builders
- Dynamic APIs for data integration

Why it matters: It gives clients control without increasing vendor maintenance costs or breaking shared infrastructure.

2. Extension Frameworks (Plugins, SDKs)

Principle: Allow deeper custom logic or UI elements via defined extension points — **without modifying the core platform**.

Examples:

- Salesforce's Lightning Components
- Shopify apps
- SAP's extension framework
- Even Juniper allows some JS overrides or XML extensions

Why it matters: Custom functionality is sandboxed, so core updates don't break it.

3. Tiered Architecture

Vendors often structure their product in layers:

Layer	Role
Core	Shared across all clients, rarely changed
Verticals	Modules built for specific industries
Client Add-ons	Project-specific logic/plugins/scripts

You likely needed something at the third layer — which many SaaS vendors are hesitant to support unless it fits a broader roadmap.

4. Strategic Forking / Co-Development

Some vendors agree to **fork** or **co-develop** custom modules for strategic clients — with:

- Separate SLAs,
- Higher cost/time estimates,
- Limited support guarantees.

This approach is **expensive and risky**, but done for large deals or proof-of-concept innovation.

5. SaaS + PaaS Hybrid Models

Vendors now offer a **platform-as-a-service (PaaS)** layer for customers to build their own services *on top* of the SaaS.

Example:

- **Mendix, OutSystems, Microsoft Power Platform**

This trend is growing where SaaS alone is too rigid.

What Happens When the Vendor Refuses?

If the vendor won't support needed custom work:

- Clients resort to **workarounds and scripting**, often fragile,
- Or they abandon the platform and seek alternatives (as happened in your case),
- Or negotiate roadmap influence (usually via commercial leverage).

Why It Mattered in Our Case

- We relied on a standard SaaS product without a clear extension path.
- The needed feature set (advanced template control, performance safeguards) exceeded what Juniper could offer within its T&M model.
- Without DevOps and modular deployment practices, even small changes introduced instability.

Key Takeaway for Product Owners

When selecting or working with SaaS products:

- Evaluate not only the features, but also the **customization boundary**.
- Ask: *Can I extend this without breaking the product? Are there plugin APIs, custom workflows, or sandboxed logic?*
- SaaS is scalable, but innovation needs extensibility — not just configuration.

This reflection helped me form a more strategic approach to SaaS evaluation, and strengthened my ability to anticipate product fit risks in platform-based projects.