# SaaS Analysis Report

**Version 1.0**

Alejandro García-Fernández, José Antonio Parejo, Francisco Javier Cavero, Antonio Ruiz-Cortés

SCORE Lab, I3US Institute, Universidad de Sevilla, Spain

{agarcia29,japarejo,fcavero,aruiz}@us.es

**ICSOC 2024**

22nd International Conference on Service-Oriented Computing

July 22, 2024

# INDEX

# Chapter 1

# Introduction

The Software as a Service (SaaS) model is a distribution and licensing paradigm that has grown significantly in popularity over recent decades [7]. This model involves the delivery of software through the cloud, providing users with a set of features and support guarantees accessible by paying a periodic fee.

Information about the available fees is provided through a pricing, a structure consisting of various plans and optional add-ons that group and control access to features, imposing usage limits when needed. With this approach, SaaS providers can offer different sets of features and usage limits for different customer profiles, adjusting to varying budgets and requirements. In this way, pricings enhances flexibility for users, enabling them to suit their subscriptions to their needs and easily upgrade or downgrade their plans, and maximizes revenue for providers by catering to varying customer needs and promoting market expansion; as a variety of pricing plans attract a broader customer base.

As this paper demonstrates, changes in pricings are frequent (removal, modification, or addition of features, usage limits, plans, or add-ons). Therefore, to remain competitive, SaaS developers and operators must minimize the time needed to implement these pricing changes in the software and underlying infrastructure (typically a Platform as a Service), while maintaining quality and reliability. This process, which imposes significant challenges on development and operations teams and should be streamlined for efficiency, has been recently coined as *Pricing-driven Development and Operation of SaaS* [3][1].

Our goal is to pave the way for further research in the design and development of technologies that automate and optimize the Pricing-driven SaaS DevOps process. In order to obtain a deeper and detailed knowledge about the current support offered by the industry for Pricing-driven SaaS DevOps, in this paper we explore the dimensions of change in SaaS pricing by modeling more than 150 pricing plans from 30 different commercial SaaS, and tracking their evolution over six years (2019-2024). In addition, since feature toggling allows specific features to be enabled or disabled without deploying new code —merely by modifying configuration files— and it is one natural approach for implementing pricings in SaaS source code, this paper analyzes its current support for Pricing-driven SaaS DevOps in the industry, academy and open source community. In particular, we focus on feature toggling tools, examining up to 21 in detail.

This work includes the following original contributions in the context of the Pricing of SaaS:

1. The definition of the configuration and evaluation spaces of a pricing, two key concepts for analysing the evolution of pricings.

2. An unprecedented analysis of 162 pricings of 30 commercial SaaS, yielding significant

[1] For brevity, we may also refer to this concept as Pricing-driven SaaS DevOps.

insights into the structures and trends of SaaS pricing models.

3. A comprehensive dataset and repository of these 162 pricing models, derived from the 30 commercial SaaS analyzed, by using the only metamodel proposed in the literature—Pricing4SaaS [4]—, which will facilitate further analysis and research on SaaS pricing.

4. A comparative overview of the support offered by 21 feature-toggling-based solutions for implementing changes in pricings.

# Chapter 2

# Background

## 2.1 SaaS pricing

A pricing is a structure that organizes the *features* of a service —defined as the distinctive characteristics whose presence/absence may guide an user's decision towards a particular subscription [3]— into *plans* and *add-ons* to control users access to such features. While users can only subscribe to one of the available plans, they can subscribe to as many add-ons as they want since they are available for the contracted plan. Fig. 2.1 illustrates a pricing for PetClinic, a sample veterinary clinic management service that developers use to illustrate the features of a particular software framework or technology in a real-world scenario.[1] It includes ten features regulated by three plans and three add-ons (seven by plans, three by add-ons), with one add-on exclusive to the PLATINUM plan, and imposes *usage limits* on the "pets" and "visits" features. Given this running example, it is important to note that not all pricing features are necessarily translated into code within the service. Those that are will be referred to as *functional features*, while those that aren't will be referred to as *non-functional features*. The latter represent service-level guarantees, such as "Support Priority" and "SLA Coverage" in PetClinic.

According to our research, Pricing4SaaS (see Fig. 2.2) is the only metamodel proposed in the literature that formalizes these pricing elements [4], being capable of representing pricings regardless of their elements distribution. Additionally, its YAML-based serialization, Yaml4SaaS, has been pivotal in our analysis of industry SaaS (see Section 4.0.1), allowing us to model all the studied pricings while ensuring portability for our work, making it reusable for future research.

| | **BASIC** $4.95 per clinic/month | **GOLD** $9.95 per clinic/month | **PLATINUM** $19.95 per clinic/month |
|---|---|---|---|
| Max pets per owner | 2 | 4 | 7 |
| Max visits per pet | 1 visit/month | 3 visits/month | 6 visits/month |
| Appointments Calendar | | ✓ | ✓ |
| Vet Selecction | | ✓ | ✓ |
| Online Consultations | | | ✓ |
| Pets Dashboard | | | Add-On |
| Support Priority | LOW | MEDIUM | HIGH |
| SLA Coverage | 99% | 99.5% | 99.9% |

**ADD-ONS**

**Smart Clinic Reports**
**$3.95 / month**
only with Pets Dashboard

**Pet Adoption Centre**
**$15.95 / month**

**Pets Dashboard**
**$5.95 / month**

Figure 2.1: Sample pricing with ten features, three plans and three add-ons.

Given a SaaS pricing with a set of plans $P = \{p_1, p_2, \ldots, p_n\}$ and add-ons $A = \{a_1, a_2, \ldots, a_m\}$, a customer interacts with the service by establishing *a subscription*, i.e. a "bundle" that may

[1] The base version of PetClinic for Spring can be found here.

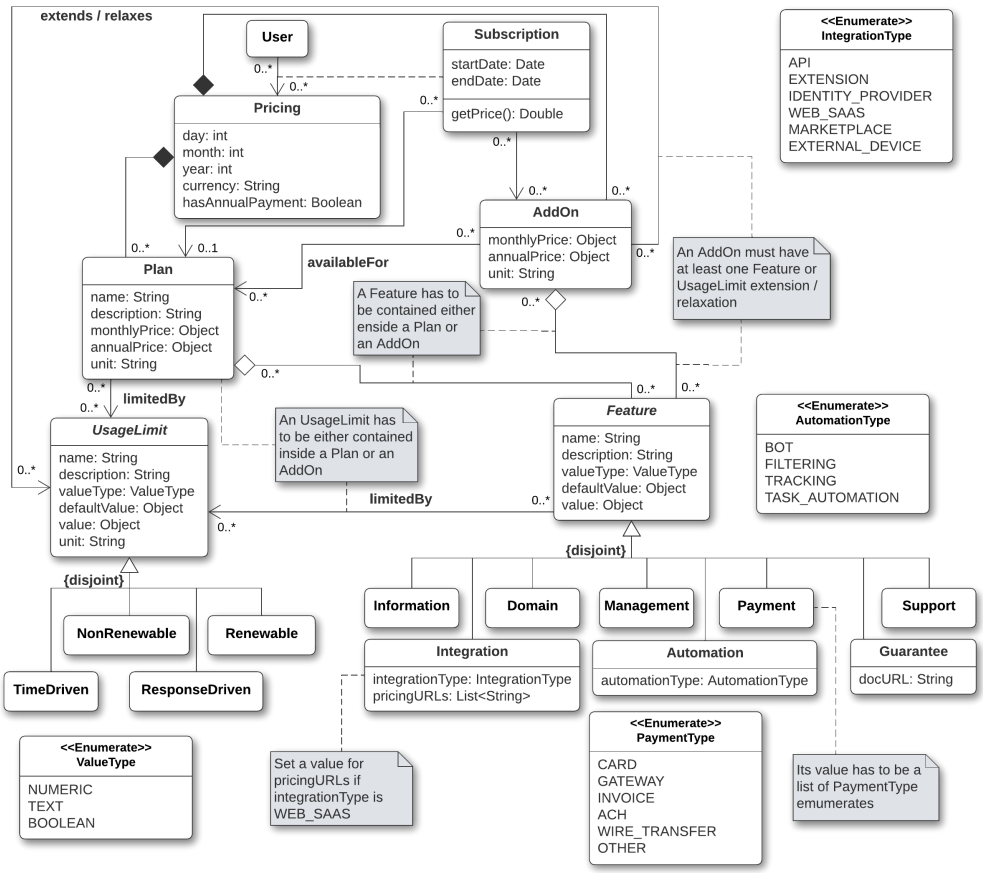


Figure 2.2: Pricing4SaaS UML model. Created from the original in [4].

include a plan and optionally a set of add-ons, ensuring that:

1. Subscription is not empty.

2. It contains exactly 0 plans if $P = \emptyset$, and 1 otherwise.

3. Any add-on included within the subscription is not excluded for the selected plan, e.g. the "Pets Dashboard" add-on of PetClinic is only available for the PLATINUM plan, meaning the add-on is excluded for BASIC and GOLD plans ($E(\text{“}PetsDashboard'') = \{BASIC, GOLD\}$).

4. All required add-ons for any add-on in the subscription also included in it, e.g. in PetClinic: $D(\text{“}PetsDashboard'') = \{\text{“}SmartClinicReport''\}$.

Once customers have made their selection, they commit to paying a periodic fee to gain the ability to access and leverage the features provided by the SaaS in the terms and usage limits set out by the chosen subscription. E.g. customers with the BASIC plan in PetClinic can register up to two pets in their account, cannot select a vet for their visits, etc.

Given this structure, determining the set of different subscriptions within a pricing may become very challenging. We have coined this concept as the *configuration space* of pricings, aiming to enhance our understanding of pricings. Formally, we describe a pricing's configuration space $C$ as:

$$C(P, A, E, D) = \{(p, a) \in P \cup \emptyset \times \mathbb{P}(A) \mid isValid(p, a, E, D)\}$$

where $isValid$ encodes the above defined rules and is defined as:

$$isValid(p, a, E, D) \iff \left\{ \begin{array}{ll} p \in P \cup \{\emptyset\} & (1) \\ \wedge\, a \in \mathbb{P}(A) & (1) \\ \wedge\, ((p \neq \emptyset) \vee (a \neq \emptyset)) & (1) \\ \wedge\, |P| \geq 1 \Rightarrow p \neq \emptyset & (2) \\ \wedge\, \forall a_i \in a,\ p \notin E(a_i) & (3) \\ \wedge\, \forall a_i \in a,\ D(a_i) \subseteq a & (4) \end{array} \right\}$$

In addition, the maximum cardinality of C is easy to determine, as it is equivalent to not considering the exclusion and dependency relashionships between plans and add-ons ($E = \emptyset$ and $D = \emptyset$). Therefore, the maximum cardinality of a pricing's configuration space with $n$ plans and $m$ subsets of $A$ is:

$$max\ |C| = \left\{ \begin{array}{ll} 2^m - 1 & \text{if } P = \emptyset \\ n \cdot 2^m & \text{if } P \neq \emptyset \end{array} \right.$$

As an illustration, the maximum cardinality of PetClinic's pricing configuration space would be:

$$max|C| = n \cdot 2^m = 3 \cdot 2^3 = 24$$

In this regard, add-ons play a crucial role in pricings, since they exponentially increase the size of the configuration space —enabling the accommodation of a wider range of user needs— while reducing customer decision fatigue. As the *Paradox of Choice* posits, "human beings tend to feel less satisfied with their decisions when faced with a greater number of alternatives to choose from" [9].

## 2.2 Feature toggles for Pricing-driven DevOps

*Feature toggles* are a software development technique that allows features to be dynamically enabled or disabled without modifying the code. In a nutshell, this behaviour is implemented by using boolean expressions on which some values can be are assigned, or modified, at runtime, thus providing dynamic evaluations [2]. Fig. 2.3 illustrates the simplest version of a feature toggle in the source code of a hypothetical implementation of PetClinic. This toggle evaluate the feature "Appointments Calendar" (see Fig. 2.1), enabling or disabling its web component based on the user's plan. As shown, the evaluation of the conditional block is hard-coded, but some values depend on dynamic data, e.g. "userPlan", which is retrieved, using the "fetchUserPlan" function, from the toggle context, an external source (e.g. a database) that contains the data needed for the evaluation.

In this scenario, interpreting a pricing as a software artifact that determines the behavior of the SaaS can be highly effective, as developers can translate it into pricing-driven feature toggles, i.e. permission feature toggles [2] that are used to provide each user with a different version of the service at runtime regarding their subscription. This approach leverages the full potential of SaaS, which is realized when multiple customers with diverse requirements can be accommodated within a single application instance [6].

Unfortunately, the use of feature toggles also increases the complexity of managing the service, since they generate "one of the worst kinds of technical debt" [10] and transform testing into a combinatorial problem [8]. The concept of pricing's *evaluation space* that we introduce becomes crucial in this context. Given that feature toggles have a inherent evaluation, and considering that any functional feature of the pricing has an associated feature toggle, the evaluation space can be defined as the minimum set of feature toggles that have to be evaluated in order to operate the functional features governed by a pricing, it enables DevOps teams to predict the impact of

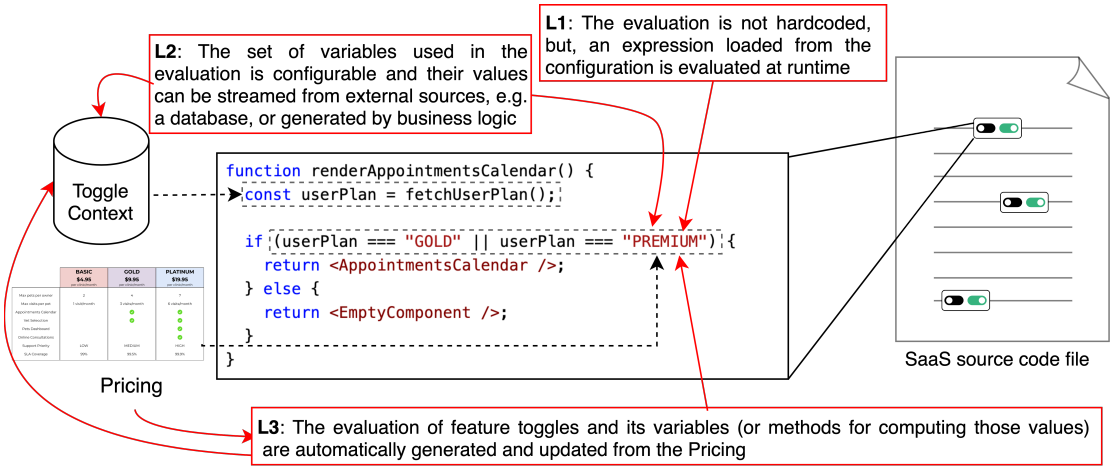


Figure 2.3: Feature toggles and how the defined capability levels relate to their elements. Based on the original from [3].

changes on pricing elements within the service. In order to approximate its size, the simplest architecture must be considered, i.e. monolithic client-server, with each feature evaluated only once on each side. For instance, in PetClinic (Fig. 2.1), since only eight features of its pricing are functional (all except "Support Priority" and "SLA Coverage"), the size of the evaluation space of its pricing is 16. Respectively: i) eight pricing-driven feature toggles on the frontend to activate/deactivate UI elements related to the specified pricing features, and ii) the eight corresponding toggles on the backend to manage access to such features.

As can be seen, the size of the evaluation space usually doubles the number of functional features governed by the pricing, highlighting the challenge of managing pricings with a large feature set. In addition, the number of features is not the only dimension of growth of the evaluation space; although the configuration space doesn't impact on the number of feature toggles required to operate a pricing-driven service, it affects the complexity of the evaluations these toggles perform. Recovering the example from Fig. 2.3, having 3 plans (FREE, GOLD, and PLATINUM) within PetClinic means that deciding whether "Appointments Calendar" is available for a user or not requires checking if they are subscribed to either the GOLD or PLATINUM plan. If there were only 2 plans (FREE and GOLD), the check would be simpler, as you would only need to verify if the user is subscribed to the GOLD plan.

However, the complexity associated with the increasing size of the evaluation space can be mitigated by using feature toggling libraries (see Tab. 5.1). The advantage of using such tools to manage pricing-driven feature toggles is that their evaluations don't need to be hard-coded, as they are abstracted into a configuration file. Unfortunately, not all industry feature toggling frameworks provide enough flexibility to efficiently manage pricing-driven feature toggles, so we propose a set of levels based on their capabilities:

- *L1: Configuration-based toggling.* Feature toggling tools that meets this level would allow to transfer the evaluation expression of feature toggles to a configuration file that can be modified at runtime to apply changes within the service. However, developers still need to maintain and update the configuration file for each change on the pricing.

- *L2: Dynamic and extensible contextual evaluation.* In addition to L1, some feature toggling tools allow to customize how the variables used in the toggling expressions, e.g. userPlan in Fig. 2.3, are evaluated at runtime, even loading their values from external sources as needed; so there is no need to declare them before the feature toggle is evaluated. In the running example of Fig. 2.3, this would allow to load the value of "userPlan" from an external source, such as the session of the current user.

- *L3: Pricing-aware evaluation.* This level involves the capability of feature toggling tools to automatically generate dynamic evaluation expressions for feature toggles from a se-

rialized pricing. For instance, the evaluation shown in Fig. 2.3 would be replaced by: $userSubscription[\text{“}appointmentsCalendar\text{”}]$ which dynamically checks the pricing to determine whether the “Appointments Calendar” feature is available within the user's subscription. This advanced control allows the tool to automatically adapt to changes in the serialized pricing, ensuring that any updates to the document are seamlessly integrated into the system's functionality and behavior.

# Chapter 3

# Research Questions

To meet our goal: pave the way for further research in the design and development of technologies that automate and optimize the pricing-driven DevOps process for SaaS —thus providing solutions to previously highlighted issues— the following research questions were defined:

$RQ_1$: **How are pricings' configuration and evaluation spaces evolving in modern SaaS?** Given that in recent years SaaS pricing models have incorporated new elements, such as add-ons, and modified their structure (in terms of plans and features); this research question addresses several key aspects of pricing dynamics to uncover the trends in design and complexity of these structures, thus identifying the challenges faced by SaaS providers to manage Pricing-driven Development and Operation of SaaS.

> $RQ_{1.1}$: ***At what rate does the number of features in a SaaS grow?*** Understanding such rate helps to identify whether SaaS products are becoming more feature-rich over time, thereby increasing the evaluation space of the pricing-driven feature toggling.
>
> $RQ_{1.2}$: ***How is changing the number of plans within SaaS pricings?*** I.e, is there a trend toward more diversified plans that cater to different customer segments, or is the trend to maintain a fixed number of plans despite adding new features?
>
> $RQ_{1.3}$: ***How is the number of add-ons evolving over time?*** Given that add-ons are the element that most affects the complexity of the pricing's configuration space, understanding their growth trend is crucial for assessing the complexity of pricing configuration spaces in the future.

$RQ_2$: **Are feature toggling tools appropriate for optimizing pricing-driven feature toggling?** Given that current pricing plans may include specific usage limitations of features per plan (e.g., a maximum of 2 pets for basic users), and that some of these constraints can be contextual (e.g., a maximum of 5 veterinary visits per month for basic users), this question translates into:

> $RQ_{2.1}$: ***Do current feature toggling tools support the use of evaluation expressions for dynamic feature toggling activation?*** i.e., do they evaluate at runtime a user-defined expression based on a set of predefined variables to decide enabling or disabling a feature, or do the use simple boolean values loaded from configuration files?
>
> $RQ_{2.2}$: ***Do current feature toggling tools with dynamic activation expressions support the extensibility of the variables on which such expression depend?*** Those extensibility mechanisms should include user-defined variables with values obtained from an external sources or user-defined business logic defined in the implementation language of the SaaS.

$RQ_{2.3}$: ***Are current feature toggling tools sensitive to pricing plans?*** Meaning that they can generate the dynamic expressions and configurations for feature toggles activation evaluation directly and automatically from a serialization of the SaaS pricing plan.

$RQ_3$: ***Does Yaml4SaaS provide enough expressiveness to represent real world pricings?*** I.e are we able to accurately model all the pricing versions analyzed in the study for $RQ_1$?.

# Chapter 4

# $RQ_1$: SaaS Pricings Evolution Analysis

### 4.0.1 Methodology for the analysis of SaaS pricings

**Sample selection**. As far as we know, there is no systematic methodology for selecting a sample of SaaS pricings. Therefore, to obtain our sample, we decided to start with the 13 SaaS included within the repository of the authors of Pricing4SaaS [4]. In our goal to expand their dataset to include a total of 30 SaaS, we extracted some additional services from [1] with this selection criterion: i) each selected SaaS must have a trackable pricing history with snapshots of their pricing webpage available in the Wayback Machine for at least four years between 2019 and 2024, inclusive; ii) their snapshots must contain a clear list of features for most of these years.

**Snapshot selection**. For each year from 2019 to 2024, priority was given to selecting pricing versions from October/November (when available), aiming to maintain a gap of six months to one year between each studied snapshot. For 2024, the pricing version from June/July was selected, creating a snapshot in the Wayback Machine when needed. On the one hand, Table 4.1 presents the resulting dataset of SaaS, detailing the number of snapshots and additional metrics for the latest version of their pricing (2024). On the other hand, Table 4.3 contains the whole list of collected snapshots (and Table 4.2 the specification of the selected product for SaaS with multiple products for certain years), using the following notations:

- ✓ indicates that the pricing page of the SaaS had an available snapshot in the Wayback Machine for that year and has been successfully modeled within the dataset.
- ✗ indicates that, despite having an available snapshot, either a clear list of features could not be found within the pricing page or the page could not be rendered.
- ✗ indicates that no snapshot was available for the pricing of the SaaS for that year in the Wayback Machine.

**Pricing modeling**. Given that, to the best of our knowledge, Pricing4SaaS (see Fig. 2.2) is the only metamodel available in the literature that represents SaaS pricings, we decided to model all the studied pricings using its YAML serialization: Yaml4SaaS. This approach not only validate the feasibility of this pricing model but also provides a dataset with enough portability for further research in Pricing-driven SaaS DevOps. In addition, to minimize human error and mismatches during the modeling phase, several rules were established:

1. Features containing "and" in their description are separated into distinct features, e.g. "Record and play audio notes" of Evernote 2022 must be splitted into "Record audio

| SaaS | S | F | P | A | C | SaaS | S | F | P | A | C |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Salesforce | 6 | 111 | 3 | 14 | 12544 | Buffer | 6 | 76 | 4 | 3 | 7 |
| GitHub | 6 | 81 | 3 | 14 | 8960 | Jira | 6 | 60 | 4 | 1 | 7 |
| Postman | 5 | 100 | 4 | 12 | 1412 | Notion | 4 | 58 | 4 | 1 | 7 |
| Databox | 6 | 62 | 5 | 8 | 786 | Figma | 6 | 90 | 6 | 0 | 6 |
| OpenPhone | 5 | 48 | 3 | 6 | 192 | Box | 6 | 50 | 5 | 0 | 5 |
| Wrike | 6 | 78 | 5 | 5 | 85 | Canva | 6 | 92 | 4 | 0 | 4 |
| Tableau | 6 | 41 | 3 | 7 | 48 | Dropbox | 4 | 82 | 4 | 0 | 4 |
| Zapier | 5 | 51 | 4 | 4 | 40 | Evernote | 6 | 32 | 4 | 0 | 4 |
| Slack | 4 | 44 | 4 | 4 | 21 | Hypercontext | 4 | 63 | 4 | 0 | 4 |
| MailChimp | 6 | 90 | 4 | 5 | 15 | Pumble | 4 | 34 | 4 | 0 | 4 |
| ClickUp | 6 | 135 | 4 | 2 | 13 | UserGuiding | 5 | 59 | 3 | 1 | 4 |
| Planable | 6 | 41 | 4 | 2 | 13 | Crowdcast | 5 | 16 | 3 | 0 | 3 |
| Clockify | 6 | 72 | 6 | 4 | 10 | Deskera | 4 | 100 | 3 | 0 | 3 |
| Microsoft 365 | 6 | 60 | 4 | 1 | 8 | Overleaf | 6 | 16 | 3 | 0 | 3 |
| Trustmary | 5 | 45 | 4 | 1 | 8 | Quip | 6 | 15 | 3 | 0 | 3 |

Table 4.1: SaaS dataset with the number of snapshots and some metrics related to their 2024 pricing. S indicates the number of available snapshots; F the number of features; P the number of plans; A the number of add-ons; and C the size of the configuration space.

| SaaS | Year | Product |
|---|---|---|
| Buffer | 2019 | Publish |
| Salesforce | 2019-2024 | Sales |
| Tableau | 2019-2024 | Tableau Online |
| Trustmary | 2021 | Full Suite |
| Trustmary | 2023 | Collect |
| Overleaf | 2023-2024 | Individual |
| MailChimp | 2020-2024 | Marketing |

Table 4.2: Selected products in snapshots with multiple products

| SaaS | 2019 | 2020 | 2021 | 2022 | 2023 | 2024 | SaaS | 2019 | 2020 | 2021 | 2022 | 2023 | 2024 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Box | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | Buffer | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Canva | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ClickUp | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Clockify | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | Crowdcast | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Databox | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | Deskera | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ |
| Dropbox | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | Evernote | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Figma | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | GitHub | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Hypercontext | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | Jira | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| MailChimp | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | Microsoft 365 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Notion | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | OpenPhone | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Overleaf | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | Planable | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Postman | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | Pumble | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ |
| Quip | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | Salesforce | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Slack | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ | Tableau | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Trustmary | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | UserGuiding | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Wrike | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | Zapier | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ |

Table 4.3: Dataset of SaaS pricings versions resulting from the study

notes" and "play audio notes" features.

2. Plans and add-ons without a clear feature list are not modeled, e.g. "Teams" plan of Evernote 2022.

3. Features offered as a limited trial or demo for a specific plan, thus not providing any permanent access, will not be included within the feature set of such plan. E.g. Slack's 2023 "Free" plan does not include "file history".

4. Recommended user limits must not be not modeled, as they do not restrict any other feature.

### 4.0.2 Saas pricings analysis results

Table 4.1 provides an overview of the metrics computed for the models of the 30 SaaS for 2024. Next, we report the results with regard to our research questions on SaaS pricings' configuration and evaluation spaces evolution. Fig. 4.1 depicts the trends observed in the aspects addressed by $RQ_1$ across the sampled SaaS pricings. The box plots are enriched with a dashed blue line that connects the mean values for the whole set of SaaS per year, which helps to visualize the evolution of the indicator more clearly. This is in addition to displaying the median (indicated by the green line within the box) and the overall distribution of values.

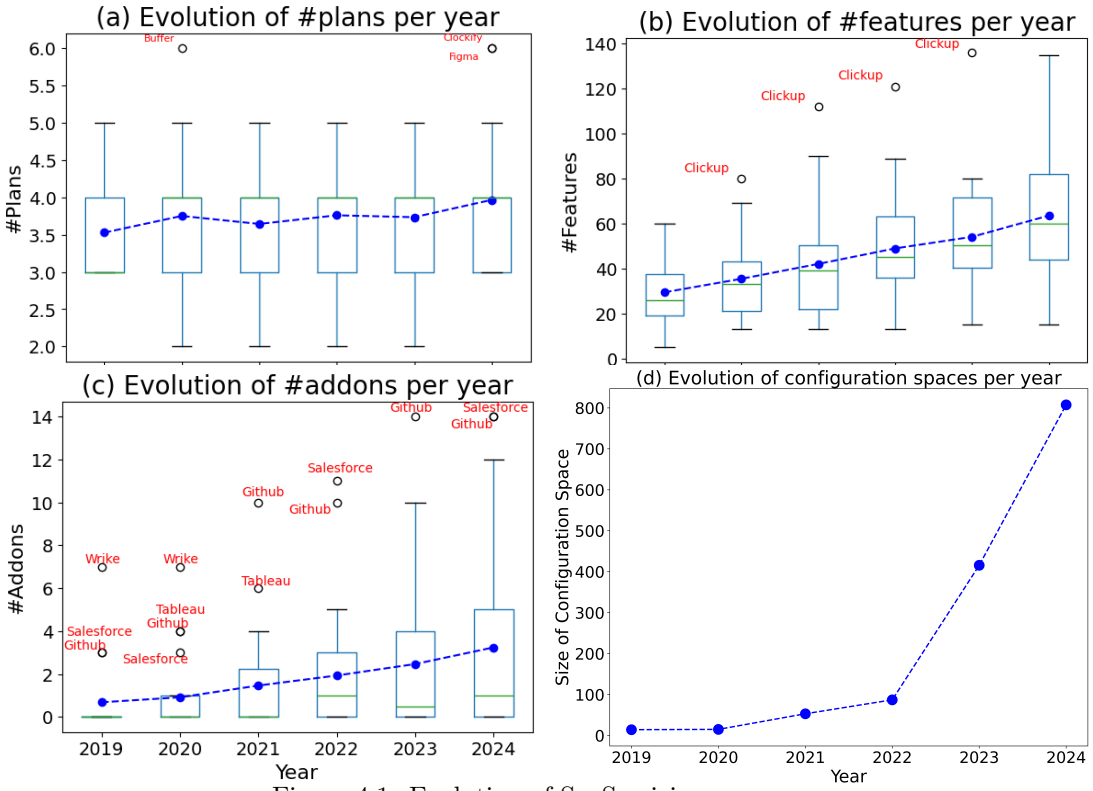


Figure 4.1: Evolution of SaaS pricings per year.

*Features growing rate.* The results (Fig. 4.1b) show a linear increase in the number of features over the years. From 2019 to 2024, the average number of features of the SaaS under study has increased by 115% (from 29.47 to 63.40). When normalizing the number of features (Fig. 4.2), considering zero as the baseline minimum, four different patterns of evolution are identified (see Fig. 4.3):

1. 11 SaaS exhibited a great feature addition rate at the beginning, but it gradually slowed down (e.g. Buffer). One possible explanation for this behavior is that managing pricing-driven feature toggling became increasingly challenging, thereby slowing the development of new versions and features.

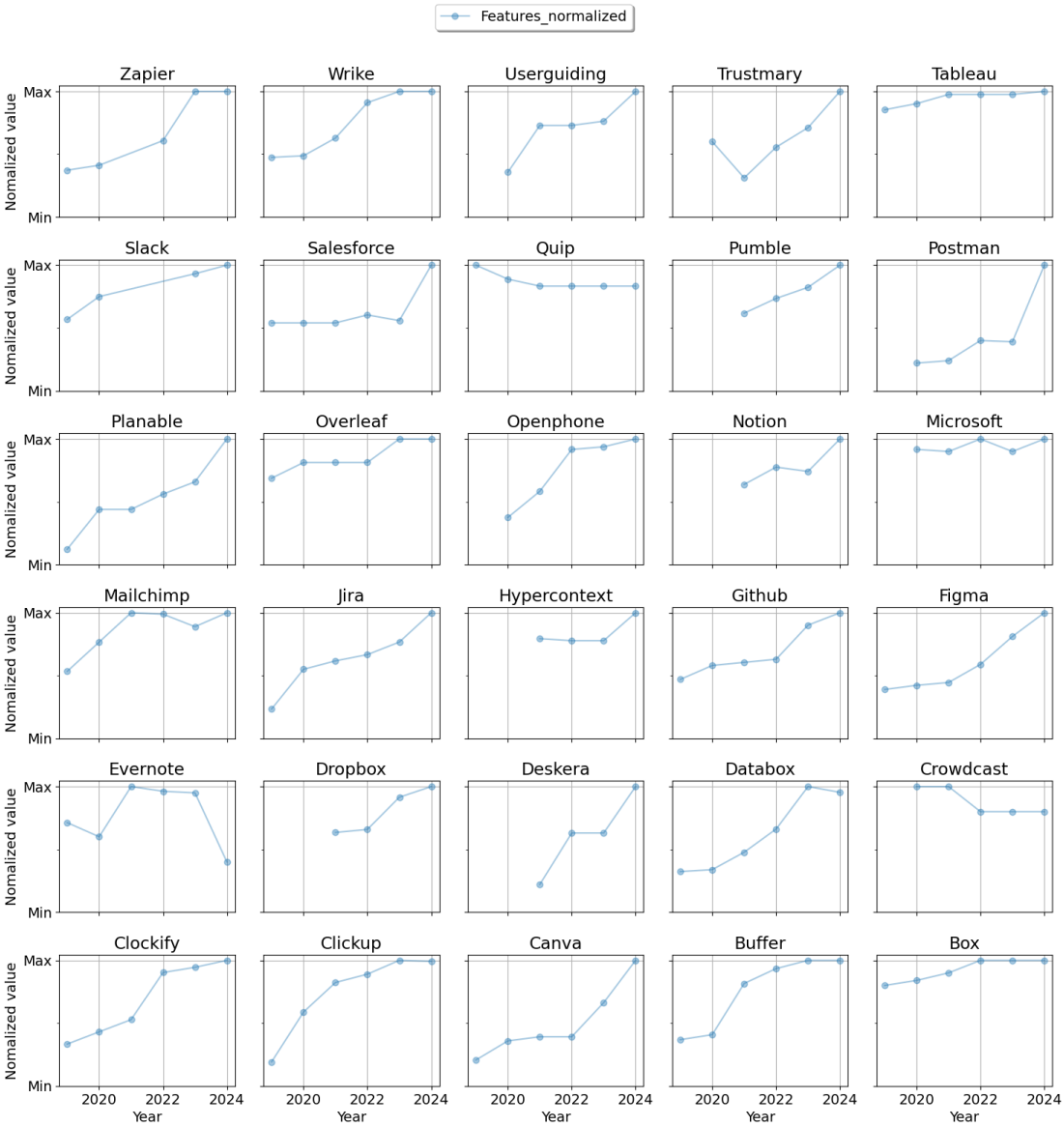


Figure 4.2: Evolution of number of features by SaaS

2. 12 SaaS overpassed such "slow down" period, recovering a high rate of addition (e.g., Canva). Curiously, these SaaS are managed by the companies with the largest infrastructure and most experience in the dataset, which may indicate that, based on their expertise, they have surmounted the challenges of managing pricing-driven feature toggles and successfully optimized pricing-driven development and operation.

3. 6 SaaS pricings didn't changed significantly, but still show variations (e.g. Tableau).

4. Evernote was the only SaaS of the dataset that reached a maximum number of features and then began to remove or group features into more general ones, thus reducing the total number;

*Evolution of the numbers of plans.* The number of plans has shown a relatively stable trend from 2019 to 2024 (Fig. 4.1a), with the number of plans hovering around three to four (the mean increase between 2019 and 2024 is an 11%). While there are occasional outliers, such as *Buffer* and *Clockify*, the overall distribution does not show significant increases or decreases in the number of plans offered. This suggests that while some SaaS providers may experiment with the number of plans, the general approach across the industry has remained consistent in terms of the number of pricing plans available to customers. The reason for this tendency might be

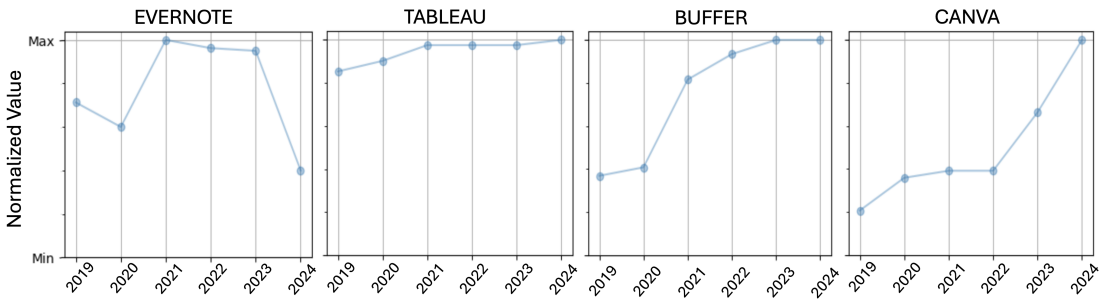


Figure 4.3: Identified types of number of features evolution patterns within the dataset

the Paradox of Choice (see Section 2.1), as keeping a reduced number of plans facilitates the decision of the customer.

*Evolution of the number of add-ons.* As shown in Fig. 4.1c, there is a distinct trend towards incorporating more add-ons within SaaS pricing models, with projections indicating a linear increase in the future. Between 2019 and 2024, the average number of add-ons has increased by a 363% (from 0.68 to 3.16). This trend may be attributed to the fact that add-ons expand the configuration space of a pricing model without falling into the Paradox of Choice, as discussed in Section 2.1. Fig. 4.1d shows an exponential increase of the configuration space due to the increase in the number of add-ons, validating empirically our formulation of the size of the configuration space (see Section 2.1).

*Relationship between plans and add-ons* By normalizing the number of plans and add-ons and comparing them in a single plot (Fig. 4.4, we observe many instances where the number of plans decreases/stands while the number of add-ons increases. This trend indicates a strategic substitution of plans with add-ons, aiming to maintain a streamlined number of plans (3-4) while offering a wider configuration space (as discuted in Chapter 2). This approach allows SaaS providers to enhance flexibility and customization for users without overwhelming them with numerous plan options.

*Statistical tests*, specifically the Mann-Whitney U tests, were conducted based on the results of normality and homoscedasticity tests, to compare the number of plans, features, and add-ons between 2019 and 2024. The differences were statistically significant across all compared magnitudes: plans, features, and add-ons. Effect sizes, calculated using the r estimator, indicate medium to large effects for the differences in the number of features and large effects for the differences in the number of plans and add-ons. This means the statistically significant differences identified also have a large practical impact. These results support our general research question $RQ_1$, showing a vibrant pace of change and rapid evolution towards increased complexity in both the SaaS pricings and the configuration and evaluation spaces.

Answers to **$RQ_1$: How are pricings' configuration and evaluation spaces evolving in modern SaaS?**

**(1)** Configuration spaces are experimenting an exponential rise due to the increase in the number of add-ons.

**(2)** The increasing number of features is expanding the size of evaluation spaces, making the management of SaaS pricings more challenging.

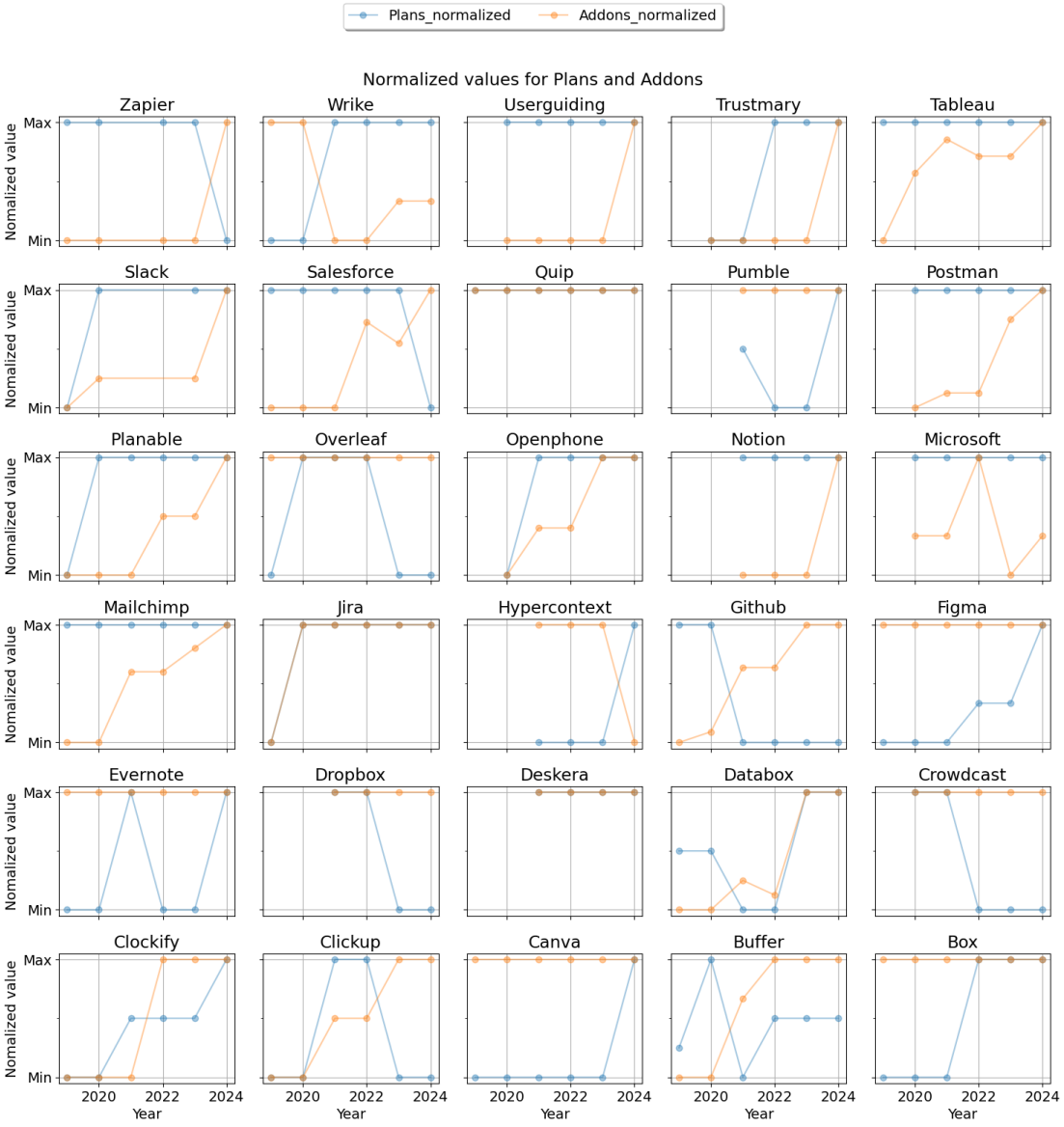


Figure 4.4: Evolution of number of plans and add-ons by SaaS

# Chapter 5

# $RQ_2$: Feature Toggling Tools for Pricing-Driven Optimization

## 5.1 Methodology for the analysis of feature toggling support

**Comparison levels**. Since the purpose of this study is not to perform a general comparative of the features provided by the feature toggling solutions, but to evaluate their suitability and support to streamline pricing-driven feature toggling, the capability levels presented in Section 2.2 are used.

**Sample selection**. In order to choose the specific set of tools to be compared in this study, we applied a methodology that combines keyword search with snowballing. First, we performed a search on Google using the keywords "feature toggling tool" and "feature toggling library" as search terms, resulting in the identification of five tools. Next, we read the description and documentation of each identified tool, looking for links to other feature toggling solutions. Finally, as a second form of snowballing, we performed a search on `alternativeto.net` using the name of each tool already identified as the search term, leading to the identification of the remaining tools in the sample.

As a result of applying this methodology, the set of tools identified and the results of the evaluation of the capabilities are shown in Table 5.1.

| Product / Company | L1 | L2 | L3 | Product / Company | L1 | L2 | L3 |
|---|---|---|---|---|---|---|---|
| Abtasty | ✓ | ✓ | | Apptimize | ✓ | | |
| ConfigCat | ✓ | ✓ | | DevCycle | ✓ | ✓ | ∼ |
| Facebook's Gatekeeper | ✓ | | | FlagSmith | ✓ | ✓ | |
| FeatureHub | ✓ | ✓ | | Harness | ✓ | | |
| javascript-feature-flags | | | | LaunchDarkly | ✓ | ✓ | ∼ |
| Molasses | ✓ | | | OpenFeature | ✓ | ✓ | |
| Optimizely | ✓ | ✓ | | Pricing4Saas | ✓ | ✓ | ✓ |
| react-feature-toggles | | | | StatSig | ✓ | ✓ | |
| Tggl | ✓ | ✓ | | Togglz | ✓ | ✓ | ∼ |
| Tweek | ✓ | ✓ | | Unleash | ✓ | ✓ | ∼ |
| XLNT | ✓ | ✓ | | | | | |

Table 5.1: Evaluation of various products/companies against levels L1, L2, and L3. A checkmark (✓) indicates that the product/company achieves the capability level, and a ∼ indicates that the level is achieved partially.

## 5.2 Feature toggling solutions analysis results

Next, we report the results with regard to our research questions on the capabilities of feature toggling solutions to streamline and automate pricing-driven feature toggling. Regarding, *configuration-based toggling* (level *L1*), the majority of feature toggling tools support this capability (90.4%), as it is also used for purposes such as user segmentation. However, the *dynamic and extensible contextual evaluation (L2)* is not as widely supported, with a 71.4% of the tools (15 out of 21) providing this feature. Finally, the *pricing-aware evaluation (L3)*, is only fully supported by Pricing4SaaS [5], but 4 out the 21 industrial and open-source solutions provide partial support (19%). This means that those tools include the extension mechanisms required to implement pricing awareness.

Answers to **$RQ_2$: Are current feature toggling tools appropriate for optimizing pricing-driven feature toggling?**

**(1)** Only one of the feature toggling solutions currently supports the implementation of pricing-driven feature toggling.

**(2)** A small set of industrial solutions has the potential to support pricing-driven feature toggling, as these included solutions possess the necessary elements.

# Chapter 6

# $RQ_3$: Limitations of Yaml4SaaS

Although we successfully modeled all the pricings in our sample using Yaml4SaaS, we identified certain limitations and details that the tool was unable to capture.

## 6.1 Limitations of Yaml4SaaS

- The model does not support defining formulas for computing numeric values, such as the price of plans/add-ons or an usage limit. E.g. if a plan's price is calculated using a mathematical expression, such as "tasks" in Zapier 2024.

- Yaml4SaaS is limited in that it cannot model inter-add-on dependencies or other complex restrictions. E.g. Microsoft Defender 2022 highlights the need for add-ons that depend on other add-ons.

- Yaml4SaaS cannot model custom subscription periods, such as semesters, it only supports monthly and annual periods. A new approach to support the definition of such periods is needed.

## 6.2 Suggested extensions for Yaml4SaaS

- Allow to define custom feature and usage limit templates to streamline the declaration of feature structures. These templates will allow users to define a feature schema once and generate multiple features based on that schema by passing a list of names. The same approach can be applied to usage limits. These templates should significantly reduce the size of Yaml4SaaS files and can be extended to plans and add-ons.

- Allow referencing one pricing from another by just indicating the path. This will permit to support sub-pricings for add-ons or other products within the same SaaS while assuring low coupling.

- Add a new attribute to the plan's structure indicating whether a plan is public or not. This could be very useful to define plans that have been designed for a single customer after a negotiation with sales.

Answers to **RQ$_3$: Does Yaml4SaaS provide enough expressiveness to represent real world pricings?**

**(1)** We were able to model most of the complexities of the pricings in our sample.

**(2)** The model does not support defining formulas for computing numeric values, such as the price of plans/add-ons or an usage limit. E.g. if a plan's price is calculated using a mathematical expression, such as "tasks" in Zapier 2024.

**(3)** Yaml4SaaS is limited in that it cannot model inter-add-on dependencies or other complex restrictions. E.g. Microsoft Defender 2022 highlights the need for add-ons that depend on other add-ons.

**(4)** Yaml4SaaS cannot model custom subscription periods, such as semesters, it only supports monthly and annual periods. A new approach to support the definition of such periods is needed.

# Bibliography


[1] Cruz, L.: 37 saas examples you need to know about in 2024 (2024), https://clickup.com/blog/saas-examples/, accessed: June 2024

[2] Fowler, M.: Feature toggles (aka feature flags) (nd), https://martinfowler.com/articles/feature-toggles.html, accessed: December 2023

[3] García-Fernández, A., Parejo, J.A., Ruiz-Cortés, A.: Pricing-driven Development and Operation of SaaS: Challenges and Opportunities. In: Actas de las XIX Jornadas de Ciencia e Ingeniería de Servicios (JCIS). SISTEDES (2024)

[4] García-Fernández, A., Parejo, J.A., Ruiz-Cortés, A.: Pricing4SaaS: Towards a pricing model to drive the operation of SaaS. In: International Conference on Advanced Information Systems Engineering. pp. 47–54. Springer (2024)

[5] García-Fernández, A., Parejo, J.A., Trinidad, P., Ruiz-Cortés, A.: Towards Pricing4SaaS: A Framework for Pricing-Driven Feature Toggling in SaaS. In: Web Engineering. ICWE. pp. 389–392. Springer (2024)

[6] Ghaddar, A., Tamzalit, D., Assaf, A., Bitar, A.: Variability as a service: Outsourcing variability management in multi-tenant saas applications. In: Proceeding of the International Conference of Advanced Information Systems Engineering, (CAISE). pp. 175–189 (2012)

[7] Jiang, Z., Sun, W., Tang, K., Snowdon, J., Zhang, X.: A pattern-based design approach for subscription management of software as a service. 2009 Congress on Services - I pp. 678–685 (2009)

[8] Rahman, M.T., Querel, L.P., Rigby, P.C., Adams, B.: Feature toggles: Practitioner practices and a case study. Proceedings - 13th Working Conference on Mining Software Repositories, MSR 2016 pp. 201–211 (5 2016)

[9] Schwartz, B.: The paradox of choice. Positive psychology in practice: Promoting human flourishing in work, health, education, and everyday life pp. 121–138 (2015)

[10] Tërnava, X., Lesoil, L., Randrianaina, G.A., Khelladi, D.E., Acher, M.: On the interaction of feature toggles. In: Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems (2022)