Universidade Federal da Bahia
Instituto de Matemática e Estatística

Programa de Pós-Graduação em Ciência da Computação

HIGHLY-CONFIGURABLE SYSTEMS IN SOFTWARE STARTUPS: UNVEILING THE WHITE LABEL MODEL

Franklin de Jesus Silva

DISSERTAÇÃO DE MESTRADO

Salvador
25 de Fevereiro de 2021

FRANKLIN DE JESUS SILVA

HIGHLY-CONFIGURABLE SYSTEMS IN SOFTWARE STARTUPS: UNVEILING THE WHITE LABEL MODEL

Esta Dissertação de Mestrado foi apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Ivan do Carmo Machado

Salvador
25 de Fevereiro de 2021

A minha família, em especial a minha avó Vanda (in memoriam) e meu avô Manga Rosa (in memoriam), que me ensinaram a importância do estudo e da perseverança.

ACKNOWLEDGEMENTS

Em tempos onde a educação mostra ser tão libertadora. Agradeço à toda a minha família, que de perto ou de longe estiveram presentes na minha jornada de aprendizado. Nas entrelinhas das minhas conquistas estão os nomes de vocês. Agradecimentos em especial a minha mãe Rose Ane Oliveira de Jesus e meu pai Franklin Lima da Silva, que foram fundamentais na escolha da minha formação e por me moldarem ao que sou hoje. Agradeço aos meus primos Savio Albuquerque e José Ricardo Almeida pela ajuda no dia-a-dia, só vocês sabem quantas garrafas de café e madrugadas sem dormir foram necessárias para chegar até aqui. Agradeço aos meus avôs paternos e maternos Fernando Pereira da Silva (in memoriam), Eliete Gomes Rabelo, Eduvanda Oliveira de Jesus (in memoriam), e Adilson Oliveira de Jesus por me permitirem compartilhar esse sonho com eles. Ao meu irmão Caio de Jesus Rios por me acompanhar nas noites de escrita jogando Freefire. Aos meus amigos de mestrado, verdadeiros companheiros de jornada, Railana Santana, Sara Lima, Tassio Virgínio, Nildo Silva, obrigado por me ajudarem nos momentos de dificuldade que aconteciam quando o trabalho atrapalhava o meu rendimento, obrigado também Renata Souza, minha parceira de artigos e com quem compartilho a área de estudo, por me ajudar a filtrar minhas ideias malucas. Por fim, gostaria de expressar minha gratidão ao meu orientador Ivan Machado, suas palavras de apoio e sabedoria enriqueceram a minha jornada, muito obrigado por não desistir de mim.

Faça a sua fé maior que seus medos e seus sonhos maiores que suas dúvidas

—ROBIN SHARMA

RESUMO

Startups são empresas que buscam explorar novos negócios incorporando novas tecnologias a diferentes mercados por meio da inovação. Os ecossistemas de startups existem para fornecer um ambiente de suporte para essas empresas, sendo uma fonte valiosa de networking e conhecimento. Projetos de software *white label* são desenvolvidos por startups e conhecidos nos ecossistemas por serem altamente adaptáveis, capazes de gerar novos produtos com mais rapidez quando comparados com formas clássicas de desenvolvimento de aplicações, garantindo o melhor custo x benefício. As startups são divididas em estágios (geramente dividios *Startup, Stabilization e Growth*) e enfrentam desafios diferentes dependendo do estágio atual. No estágio inicial, elas mal planejam suas atividades de desenvolvimento, apenas avaliam as necessidades do mercado e encontram usuários para seu produto inicial. Os projetos de software *White Label* sofrem mais com esses problemas, pois não utilizam técnicas avançadas de reutilização de código que são amplamente conhecidas pela academia e perfeitamente aplicáveis na indústria de software. Nos demais estágios, as startups necessitam implementar processos ainda mais escaláveis de forma a expandir seus negócios. Este trabalho busca introduzir o conceito de projetos de software White Label e sua viabilidade para startups de software; investigar como técnicas avançadas de reutilização de código, como Sistemas Altamente Configuráveis, podem ser utilizadas como alavanca de oportunidade para startups de software; construir um framework para software White Label móvel que seja adaptável a qualquer estágio em que a startup esteja, e que facilite que estas empresas encarem os desafios que surgem como característica de projetos white label. Para cumprir os objetivos mencionados, aplicamos três estudos empíricos sob diferentes perspectivas. Uma entrevista semi-estruturada com startups que desenvolvem projetos de software White Label, um questionário aplicado a stakeholders de ecossistemas de inovação a condução de uma Multivocal Literature Review sobre projetos de software White Label. Os resultados obtidos apresentaram um panorama de como as startups de software têm lidado com a reutilização de software em suas práticas diárias, com particular atenção em como o desenvolvimento de software White Label tem sido explorado em seus projetos. Além disso, este trabalho apresenta um estudo de caso conduzido em um projeto White Label em ambiente real, onde foi aplicado o framework criado a partir de um sistema híbrido de desenvolvimento, o react-native. Por fim, direcionamos um caminho de melhoria que pode ser aplicado a startups que trabalham com projetos de software White Label em diferentes estágios, além de oferecer aos pesquisadores um conjunto de desafios abertos a serem estudados em trabalhos futuros.

Palavras-chave: Sistemas Altamente Configuráveis, Startups, Projetos White Label, Engenharia de Software

ABSTRACT

Startups are small companies that seek to explore new businesses by embodying new technologies to different markets through innovation. Startups' Ecosystems exist to provide a supportive environment for these companies, being a valuable source of networking and knowledge to them. White label software projects are developed by Startups and known in Ecosystems as highly adaptable products, capable of generating new products faster than stand-alone applications, ensuring the best cost x benefit. Startups are divided into stages, and they face different challenges depending on the current stage. In the early stages, they barely plan their development activities but rather assess the market's needs and find users for their initial product, which leaves technical debt. White Label software projects suffer more from these issues since Startups do not address advanced code reuse techniques widely known by the academy. In the late stages, startups need to solve the technical debt managed in the early stages and need even more scalable processes to grow their business that involves White Label Software projects. This work seeks to unveil the concept of White Label software projects and their feasibility for software startups; investigate if and how advanced code reuse techniques, such as highly-configurable systems, could be used as opportunity-lever for software startups; build an adaptable framework for helping White Label software projects manage technical debt and face its challenges. In order to accomplish the aforementioned goals, we applied three empirical studies under different public. A semi-structured interview with startups using White Label software projects, an online questionnaire applied for stakeholders from ecosystems of innovation, and a Multivocal Literature Review with gray literature articles over the internet about White Label software projects. The yielded results presented a compelling portrait of how software startups have dealt with software reuse in their daily practices, with particular attention on how White Label software development has been explored in their projects. The data gathered allowed the creation of the framework above for mobile White Label software projects using a hybrid development approach, the react-native. In addition, this work gives a path of improvement that can be applied for startups working with White Label software projects under different stages, besides giving researchers a set of open challenges to be studied in future work.

Keywords: Highly-Configurable Systems, Startups, White Label Software Projects, Software Engineering

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

BML	build-measure-learn
CEO	Chief Executive Officer
CD	Continuous Delivery
CI	Continuous Integration
GSM	Greenfield Startup Model
HCS	Highly-configurable Systems
IT	Information Technology
MLR	Multivocal Literature Review
MVP	Minimum Viable Product
SaaS	Software-as-a-service
SPL	Software Product Lines
TD	Technical Debt

Chapter 1

INTRODUCTION

1.1 CONTEXT

Entrepreneurship has become an effective economy-saving tool (FELDMAN, 2001), where society's restrictions and needs are transformed into innovative projects. The Startup movement embodies innovation, especially for technologies that existing organizations do not easily absorb (LUNDGREN, 1991). In this sense, software Startups are companies focused on creating innovative and high-tech software products or services, with little or no prior experience in the market and an urge to increase its performance through new business models aggressively (GIARDINO et al., 2016).

Decker et al. (2014) discuss the role of entrepreneurship in the US job creation and economic dynamism. The study points out that startups have created an average of 18% of the US workforce annually, between the years 1980 and 2010. While this number can be considered high, startups are very volatile, some die faster than born, and the job creation cannot be simply understood from the immediate contribution upon Startup's ascend. Still, the study shows that startups have the potential to contribute to the local economy. When it comes to the Brazilian scenario, in 2020, there are over 13 thousand Startups and 78 communities, indicating excellent potential for this market.[1]

Communities are "clusters" of companies that can be called Entrepreneurial Systems, Business Ecosystems, or simply Ecosystems (MOTOYAMA; KNOWLTON, 2017). Moore (2016) defines a business ecosystem as *"an economic community supported by a foundation of interacting organizations and individuals organisms of the business world"*. Moreover, *"in the business ecosystems, companies occupy niches, just as species do in a biological ecosystem, and the various members co-evolve and tend to align themselves"*. Mack and Mayer (2016) exemplify *six domains* present in an entrepreneurial system: *human capital, markets, politics, finance, culture* and *support.* Motoyama and Knowlton (2017) indicated the nature of the connections between people in the same ecosystem. They realize the importance of not only having pioneering entrepreneurs within an ecosystem but rather

[1]Source of extracted data: ⟨https://startupbase.com.br/home⟩, accessed in September 2020.

a mix of entrepreneurs with and without previous experience to favor cooperation in a non-competitive way.

Motoyama and Knowlton (2017) applied a theoretical model of interaction that, through the perspective of social networks in entrepreneurial systems, aims to discover how the relationships between people and organizations were created and maintained. In their investigation, the researchers used a *bottom-up* approach (from individuals to organizations), starting from entrepreneurs, followed by entrepreneurship support organizations and the connections between them. The researchers state that the most influential elements and the degree of interaction between users in a Startup ecosystem are applicable in decreasing order of influence: *entrepreneurs, supportive individuals, and organizations, events on entrepreneurship* and *academy.* They also pointed out how support organizations interact with entrepreneurs, which has a high impact on *how* and *why* they connect.

In the scenario defined by Motoyama and Knowlton (2017), the academy has a marginal influence on the ecosystem's interactions. However, it is noteworthy that there are thriving ecosystems due to universities' presence in their surroundings, in which different regions facing innovation share one or more academic anchors. For example, the city of Austin is home to the University of Texas; the city of Bangalore, India, harbors the Indian Institute of Technology; Silicon Valley was created from initiatives originated at Stanford University (ADAMS, 2005).

Etzkowitz, Mello, and Almeida (2005) pointed out the Brazilian history of building hybrid organizations called "incubators". These organizations apply the so-called Triple Helix innovation model: university-industry-government, which aims to support the startup process (ETZKOWITZ; ZHOU, 2017). They stated that the movement of creating incubators started after the collapse of the Brazilian military government when the universities achieved more autonomy. For example, the Federal University of Santa Catarina (UFSC) and the Government of Santa Catarina State founded the Technological Business Incubator of the Florianópolis Science Park in 1986[2]. Similarly, a technological incubator was founded in the same year in São Paulo, more precisely at the Federal University of São Carlos (UFSCar). Since then, technological incubators have been established at universities and research centers around the Brazilian country, attracting professors and students to participate in the formation of new businesses (ETZKOWITZ et al., 2005).

For ecosystems that do not have universities nearby acting as academic anchors, events focused on entrepreneurship are essential sources of knowledge and connection between individuals, opening different views and implementations in startups. On the other hand, knowledge exchange between startups and universities can improve Startup's processes and provide exciting findings for researchers to disseminate in applied studies.

One of the many concepts discussed in Entrepreneurship Events and known in Startups Ecosystems is *White Label.* A *White Label* consists of a software product that can be easily adapted to different companies, that in turn, can deliver it to final customers as if they had built it. This approach benefits from code reuse to deliver customized solutions

[2]http://www.celta.org.br/

quickly and with a better cost-benefit when compared to stand-alone applications made from scratch.

White Label has been originated in the development of physical products and comes together with another widely-used term, *Private Label*. A *Private Label* is a product manufactured on demand, which meets the precise specifications from a partner company, which in turn sells it to the end customer. Examples for *Private Label* are products from supermarket chain brands. White Label physical products refer to goods manufactured and sold on a large scale to companies that, without due concern with customization, only apply their brand and sell the product to the end customer. An example of a*White Label* product is a computer keyboard production line, where all keyboards will be identical except for the company's brand stamped on the product, as seen in Figure 1.1 (GEYSKENS et al., 2018; KAISER, 2017).

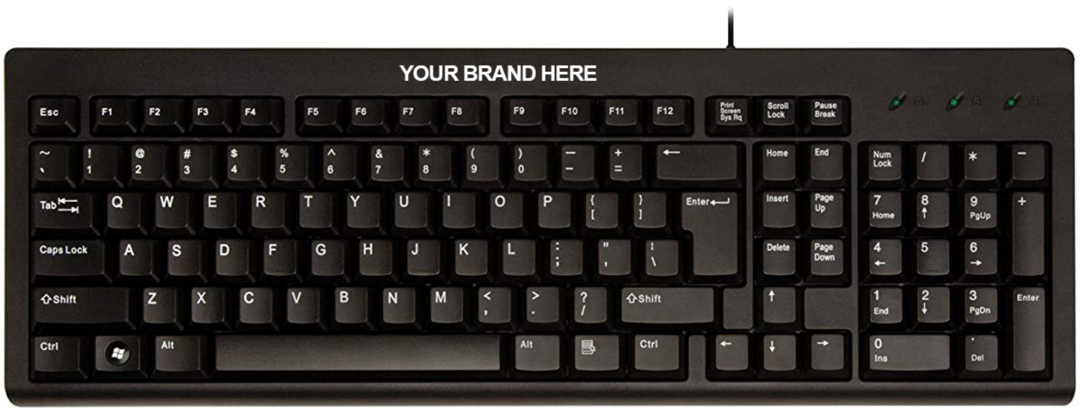


Figure 1.1 An example of White Label Keyboard

White Label software projects inherit characteristics from both Private and White Label physical products, i.e., although they enable large-scale customization from a base of common artifacts, each company partner could have a specific version of the product in the market.

The White Label approach applied to Software development has been discussed lately, with both pros and cons points of view (MACLEOD, 2012). The White Label approach can determine the business entirely, from marketing strategies to product delivery. This approach mainly defines how the software will be engineered and developed. However, few studies explore the White Label's potential as a software engineering technique, leaving a gap between what startups face in the industry and what the academy can assist.

1.2 PROBLEM STATEMENT

The theme Startups have been growing in interest by the Software Engineering research community in the last years. Recent empirical studies have overviewed the best practices, and current challenges presented (UNTERKALMSTEINER et al., 2016; BERG et al., 2018). Quite a few studies consider the differences that come according to the stages of maturity that startups should cross, which can be, according to Berg et al. (2018)

Startup, Stabilization, Growth, and Mature Organization, as illustrated in Figure 1.2. In addition, the methods used for software development in Startups are mostly *ad-hoc* and/or opportunistic (BERG et al., 2018). However, none of the following software engineering studies applied for startups take into account White Label software projects (OLSSON et al., 2012; NGUYEN-DUC et al., 2015; GIARDINO et al., 2016; NGUYEN-DUC et al., 2020).

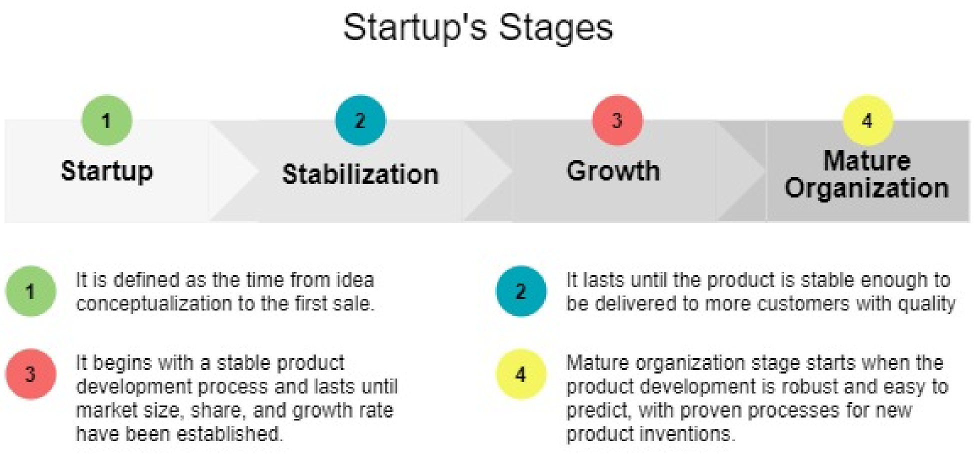

Startup's Stages
1
Startup
2
Stabilization
3
Growth
4
Mature Organization
1
It is defined as the time from idea conceptualization to the first sale.
2
It lasts until the product is stable enough to be delivered to more customers with quality
3
It begins with a stable product development process and lasts until market size, share, and growth rate have been established.
4
Mature organization stage starts when the product development is robust and easy to predict, with proven processes for new product inventions.

Figure 1.2 Startup's Stages by Berg et al. (2018)

In their earliest stage, startups barely plan their development activities but rather assess the market's needs and find users for their initial product. The way of developing a Minimum Viable Product (MVP) requires the team to take shortcuts and workarounds[3]. The absence of structure hinders essential activities, such as the following: a well-defined development process, tracking software requirements, and documenting information to share knowledge (BERG et al., 2018). When startups prepare for the growth stage, they should plan for scalable processes that require strategies they can benefit from, like reusable components and a shared architecture (NJIMA; DEMEYER, 2017).

The literature presents the results of several investigations on the causes of failures in Startups (CROWNE, 2002; MARMER et al., 2011; UNTERKALMSTEINER et al., 2016). Several symptoms are responsible for the failure in Startups, divided according to the Startup's maturity level. Many of them are directly related to the applied Software Engineering processes (CROWNE, 2002). Marmer et al. (2011) highlight the importance of avoiding *premature scaling*, which would be making the wrong decisions when scaling the product. We highlighted some of the symptoms as follows:

[3]MVP is a product version that collects the customer's needs with the least effort, based on a pre-defined thesis (RIES, 2011)

- **Startup:** There is no strategic plan for product development, meaning that the company's business plan does not establish objectives and goals for product development. The Startups in this stage make decisions on an ad-hoc basis.
- **Stabilization:** Requirements become unmanageable, meaning that the market requests more features than product development can deliver, and there is no satisfactory way of deciding between them.
- **Growth:** There is no process for product introduction, meaning that creating new products takes longer than expected. Activities take place serially rather than in parallel, and there is no coordination between teams.
- **Mature Organization:** Is where the aforementioned symptoms will not affect the product development. Software Engineering processes at this stage are robust and predictable.

Crowne (2002) claims that using ad-hoc Software Engineering practices commonly leads to project failures in startups. Equally, White Label software projects in startups suffer from it as well. They need mature Software Engineering practices to enable the creation of potentially cohesive, scalable, and consistent products (PATERNOSTER et al., 2014; BERG et al., 2018).

Startups face many common challenges, like market and financial factors, Software Engineering bad practices, resource and time scarcity. Additionally, White Label software projects have their specific struggles like maintaining many products without taking the team's overhead, making the generation of new products cost-efficient, and evolving the product without losing its identity.

In order to face the above problems for White Label software projects, startups can take benefit of advanced code reuse techniques adapted to them. As a result, Startup's experiences in adopting such techniques can be used to lever the White Label software approach in the academy.

When it comes to Software Engineering practices related to code reuse techniques, there is a paradigm that refers to the creation of software families through the systematic reuse of artifacts defined as Software Product Lines (SPL). A SPL is built through a common base of reusable artifacts and enables the generation of products with common and variable characteristics, in less time and cost, when compared to the individual construction of products at an acceptable quality level (APEL et al., 2016). This paradigm enables the delivery of a range of products, meeting different customers' expectations while maintaining a common base that can be reused and customized to meet one or another potential customer's particular needs. A SPL can be considered as a Highly-configurable Systems (HCS) (COHEN et al., 2008), which is a broader concept which encompasses software that adjusts to a determined context.

This chapter presents the White Label Software projects in Software Startups, which is one highly reuse-oriented approach in the entrepreneur environment. The Software Engineering area that discusses systematic reuse is called HCS, a concept associated with software systems that can adapt to a specific context. SPL comes as a one of the context reuse handling, which this dissertation investigates because of its similarity with White Label software projects.

For better understanding, we scoped boundaries between the aforementioned topics. This work assumes that HCS is a high-level concept that involves both White Label Software projects and SPL since they fit in the definition of HCS plus they are capable of generating many products with commonalities and differences. White label software projects may be able to absorb not only SPL practices but other context reuse-related techniques from HCS as well.

Given the similarities between White Label software projects and HCS, there is a need to understand how these concepts are related to each other. In this sense, we have established the following research question to address in this investigation:

Given the similarities between White Label software projects and HCS, there is a need to understand how these concepts are related to each other. In this sense, we have established the following research question to address in this investigation:

How could White Label software projects benefit from the Software Engineering techniques applicable to HCS in Startups?

Based on the main research question, we have established the following specific research questions:

RQ1 - What is a White Label Software Project in a startup context? This question aims to gather general concepts of White Label software projects. This question would evaluate the similarities between White Label software projects and HCS.

RQ2 - What are the professional's perspective, characteristics, and challenges involved in White Label Software Projects? This question aims to gather from professionals their experience-based knowledge in developing White Label software projects. This data will be essential to understand how the advanced code reuse techniques such as HCS can benefit this type of project.

RQ3 - How do startups develop White Label software projects? This question aims to leverage the techniques used to make viable the development of this kind of project in startups. In addition, we seek to understand whether the Startups use to adapt widely accepted approaches to meet their particularities.

RQ4 - Which Software Engineering techniques applicable to HCS can be adapted into White Label software projects ? This question seeks to identify Software Engineering Techniques that can be adjusted from the academic literature to the startup context, specifically for White Label software projects, taking into account the specificities of this type of project.

1.3 OBJECTIVE

This dissertation's overall objective is to investigate White Label software projects in startups and innovative companies, leveraging its main characteristics, main challenges, and practical evidence of its usage. Obtaining from the practitioners' voices how they implement this approach and investigate how HCS could enhance those projects.

The aforementioned objective has been refined in the following specific ones:

O1: Gathering from both practitioners and the gray literature, knowledge referring to White Label Software, characteristics, open challenges, whether and how startups commonly implement this approach.

O2: Understanding how White Label software in startups correlates with HCS.

O3: Building a framework for developing White Label software projects, applying the knowledge gathered from the empirical studies, that should be adaptable to meet Startups' particularities.

O4: Highlight key points that can enhance White Label software projects in startups based on the experience obtained after applying the framework to real-world projects.

1.4 METHODOLOGY

The research design we followed in this investigation encompassed three main parts: (1) Literature analysis and review, (2) Performing three research methods to gather information from White Label software projects under different perspectives, and (3) Creating a framework to be applied to startups and discuss a path of enhancements that startups can do to manage this type of project. Figure 1.3 shows the research design of this work.

We carried out each part by employing a set of research methods in order to reach particular goals. They are detailed next:

1. **Background, Related Work and Literature Review**
   - In the first phase, We carried out a secondary literature review by analyzing systematic literature reviews on Startups and Software Engineering concepts combined. We also gathered the state-of-the-art on HCS and White Label physical products definitions and usage.
   - With the data acquired in the review, it was checked whether the literature already discusses White Label software projects in startups.
2. **White Label Software Project Research**
   - In this phase, three research methods were performed. First, semi-structured interviews were executed (HOVE; ANDA, 2005). We conducted interviews

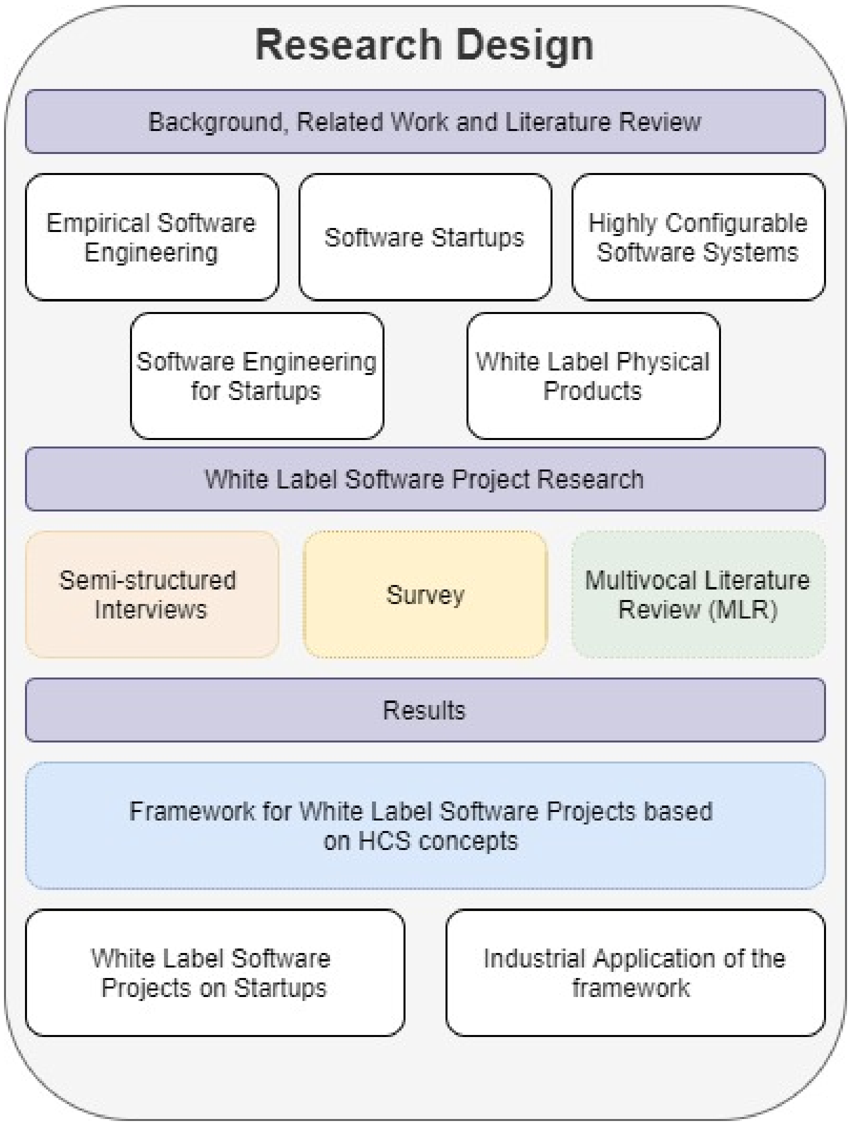

Research Design
Background, Related Work and Literature Review
Empirical Software Engineering
Software Startups
Highly Configurable Software Systems
Software Engineering for Startups
White Label Physical Products
White Label Software Project Research
Semi-structured Interviews
Survey
Multivocal Literature Review (MLR)
Results
Framework for White Label Software Projects based on HCS concepts
White Label Software Projects on Startups
Industrial Application of the framework

Figure 1.3 Overall Research Design.

with four software startups, with the intention to capture the initial perception from professionals working in White Label software projects about the practical day-to-day activities.

- Secondly, it was performed a descriptive survey to present more findings following the guidelines of Kitchenham and Pfleeger (KITCHENHAM; PFLEEGER, 2008). The first study's feedback served to guide the construction of the survey instrument, which targeted stakeholders connected to Innovation Ecosystems. The results from the two first studies were published in the SEAA 2020 conference (SILVA et al., 2020b).
- Lastly, it was performed a Multivocal Literature Review (MLR) following the guidelines of Garousi et al. (2019). We made an empirical search on Web

forums, articles across the internet about White Label software projects' origin, definition, use cases, challenges, and benefits.

3. **White Label Software Projects in Startups**

    - The collected data from the interview, survey, and MLR studies were analyzed using descriptive statistics and correlation of variables. The first part of this study is to understand White Label software projects and identify how empirical software engineering would help them.
    - This study defined the White Label software project concept, its characteristics, main challenges and explained the apparent correlation with HCS in the startup context.
    - Once this work has understood the necessities of White Label software projects. The next step involved adapting HCS techniques to the startup context.

4. **HCS Industrial Application**

    - With the definition, characteristics, and correlation with empirical software engineering defined, we proposed a framework targeting White Label software projects using hybrid technologies for developing mobile apps. This framework has been developed and tested in a real industrial environment. As a result, it provides flexibility to the Startup when performing changes, being easy to understand and maintain.
    - Our proposal is based on whether HCS is related to White Label software projects. We obtained the approval after gathering data from the multiple empirical studies applied and whether the framework built helped or not real-world White Label software projects.

1.5 SUMMARY OF CONTRIBUTIONS

Through this work, we claim the following contributions:

- Introduced the concept of White Label software projects in Startups as application sources for HCS in an Entrepreneurial context.
- Built a White Label mobile framework that follows HCS concepts that should be easy to reproduce and evolvable by startups. We have tested and validated the framework in an industrial environment.
- Created a reference of enhancements for existing White Label Software projects.

1.6 DISSERTATION STRUCTURE

The following chapters of this work are structured as follows:

- **Chapter 2.** Literature Review: Presents the state-of-the-art about Startups, Software Engineering for Startups, HCS, and White Label products, which are fundamental concepts necessary for a complete understanding of this work.
- **Chapter 3.** Semi-structured Interview Study: It presents the study applied, with its specific research methods and particularities. It describes the startups interviewed and their White Label Projects and shows the results obtained.
- **Chapter 4.** Survey Study: It presents the guidelines that we have followed for conducting the survey, the strategies, and how we planned a three-level survey that would gather respondents' experience with White Label Software projects. This chapter has also presented the results obtained for the survey.
- **Chapter 5.** Multivocal Literature Review: It presents the guidelines for conducting a MLR in Software Engineering, the step-by-step for scavaging the technical articles over the internet, and performing the quality assessment to find the chosen ones. This chapter has also presented the results obtained for the MLR conducted.
- **Chapter 6.** Adaptable Framework for Mobile White Label Software application: It presents a case study performed in a real-world scenario, where it a framework was built to develop, maintain and evolve a White Label software project. The framework was created over the learnings from previous studies and target mobile hybrid projects with react-native.
- **Chapter 7.** White Label Software Projects: It presents the discussion combined with the results of each empirical study previously explained. This chapter shows the definition, characteristics, and challenges involved in White Label software projects. This chapter also seeks to elucidate HCS techniques that White Label projects can benefit from, setting up a path for future work, and answer the third question of this work.
- **Chapter 8.** Final Considerations: This Chapter contains the final considerations of this work, as well as limitations, next steps, related work, and final contributions.

Chapter 2

BACKGROUND

This Chapter introduces the underlying concepts necessary to the understanding of this dissertation. We discuss fundamentals about Software Startups, their stages, and the methodologies employed to develop innovative products, specifically Software Engineering for Startups. Next, this Chapter introduces the concept of White Label physical products in which entrepreneurs have adapted it to the Startup context. Finally, we discuss Highly-configurable Systems (HCS) concept and its ideas that this work uses as a basis for understanding White Label software projects.

sed.

2.1 STARTUPS

2.1.1 Software Startup

Carmel (1994) first mentioned software startups when he analyzed the completion time of these projects and their accelerating elements. The author refers to Startups as innovative companies that are new to the market, working under new business models. Startups can be classified into different stages, from an immature organization, which does not have well-defined processes, to mature stages. Based on these characteristics, Carmel (1994) established that the development team is a crucial element for startups in order to deliver high-quality products. The team should be small, having involved and motivated stakeholders that usually have different skills and experience levels. Notwithstanding, startups have improved in several respects in the last years, and they cannot be compared to those as mentioned by Carmel (1994). For example, current startups range from three up to thousands of employees in terms of involved personnel.

Startups are generally small companies that seek to explore new business areas and commonly work on problems whose solutions are not well known yet (GIARDINO et al., 2014). They are willing to propose innovative solutions to existing problems using cutting-edge technologies available on the market. According to Coleman and O'Connor (2008), usually, the founders of Software Startups have few resources available and often no business model established[1]. From the perspective of software development processes, startups are more concerned with the business's survival than with the application of Software Engineering practices. Instead of developing software for a specific customer, Software Startups develop systems that contain features defined by the market, symbolizing that they sometimes have no customers before their product is launched (BERG et al., 2018).

In the following sections, we detail the core concepts of startups. They are relevant to understand the goals of this work, explain the decision-making of the Chief Executive Officer (CEO) when developing software products in startups, and understand the origins of the challenges faced.

2.1.2 Software Startup Stages

There are many definitions of the stages present in a startup's life cycle. Crowne (2002) and Marmer et al. (2011) similarly explain the life cycle of Software Startups. The latter shares a startup's life cycle as discovery, validation, efficiency, scale, support, and conservation, while the former divides into startup, stabilization, and growth. Besides, technical articles over the internet usually present the following phases in the life cycle of a Startup: ideation, operation, traction, and scaleUp [2].

Berg et al. (2018) described four stages present in a startup's life cycle: Startup, stabilization, growth, and mature organization, as follows:

- **startup**: This stage is defined as the time from the idea conceptualization to the first sale. A small executive team with the necessary skills is required to develop and deliver the product;

- **Stabilization**: It begins when the team starts to sell the product developed for more customers and begins to face escalation problems. This stage lasts until the product is stable enough to be commissioned to a new customer without causing any overhead on the product development;

- **Growth**: It begins with a stable product development process and lasts until the startup has achieved market size, share, and a high growth rate.

- **Mature organization**: The last stage starts when the startup has evolved into a mature organization. The product development is robust and easy to predict, with proven processes for new product inventions;

We have chosen the stages explained by Berg et al. (2018) to follow in this work because it covers a broad range of concepts, is easy to understand, and more meaningful when compared with the other sources found.

[1]Business models are ways to monetize the services and products offered by organizations

[2]Source of data extracted: ⟨https://startupbase.com.br/home/stats⟩, visited in September 2020.

2.1.3 Effectuation

Effectuation is a theory centered on the understanding of opportunity-seeking entrepreneurship, commonly associated with early-stage startups, which are startups facing the Startup stage. Shane and Venkataraman (2000) defined entrepreneurial opportunities as *"situations in which new goods, services, raw materials, markets, and organizing methods can be introduced through the formation of new means, ends, or means-ends relationships"*.

Effectuation claims that an entrepreneur's journey starts without clear objectives but with the desire to do the best which the resources he has at hand. Sarasvathy (2001) described the five principles of effectuation as follows:

- **Bird in hand**: Entrepreneurs start with what they are and what they already know.
- **Affordable loss**: Entreperneus invest only in what they afford to lose.
- **Lemonade**: Entrepreneurs are open to surprises and leverage them; they turn setbacks into opportunities.
- **Crazy quilt**: Build strong stakeholder partnerships with players in the ecosystem.
- **Pilote in the plane**: Entrepreneurs should focus on actions where they directly influence the outcome.

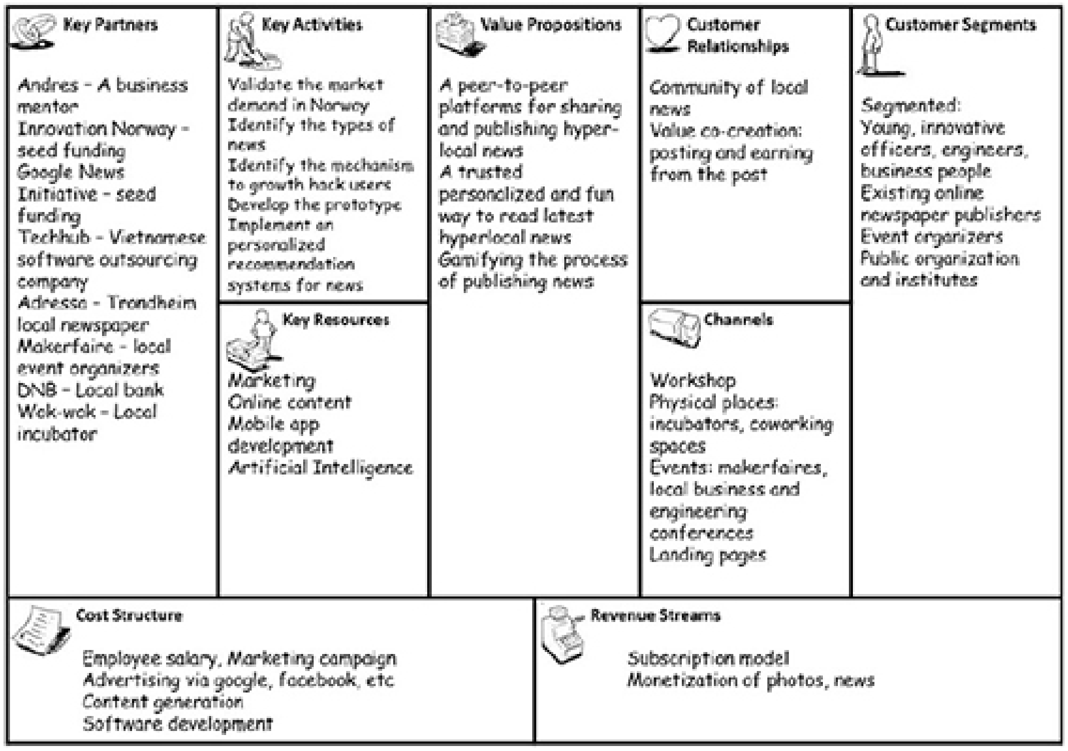
This theory is considered a useful framework to explain many engineering activities that occurred in early-stage startups. Decisions must not be influenced only by technologies and processes, but also from situations and business (DAHLE et al., 2020).

2.1.4 Business Models

Business models are the core communication between the entrepreneurs and their different advisors. It is a tool where divides an entire business into activities commonly defined (DAHLE et al., 2020). Osterwalder and Pigneur (2020) define a business model as a conceptual tool containing a set of objects, concepts, and relationships with the objective to express the business logic of a specific firm. Therefore, we must consider which concepts and relationships allow a simplified description and representation of what value the business provides to customers and the financial consequences.

There are some approaches to build business models with different structures, but the most widely used is the version proposed by Osterwalder and Pigneur (2020) called Business Model Canvas. With nine elements, the canvas highlights the *Target Customer* within the following aspects:

- The *Value* that the company creates;
- The *key channels* that describes how the company will contact the customer.
- *Customer Relations* describe the type of relationship obtained through the channels.



Key Partners

Andres - A business mentor
Innovation Norway - seed funding
Google News Initiative - seed funding
Techhub - Vietnamese software outsourcing company
Adresse - Trondheim local newspaper
Makerfaire - local event organizers
DNB - Local bank
Wok-wok - Local incubator

Key Activities

Validate the market demand in Norway
Identify the types of news
Identify the mechanism to growth hack users
Develop the prototype
Implement an personalized recommendation systems for news

Key Resources

Marketing
Online content
Mobile app development
Artificial Intelligence

Value Propositions

A peer-to-peer platforms for sharing and publishing hyper-local news
A trusted personalized and fun way to read latest hyperlocal news
Gamifying the process of publishing news

Customer Relationships

Community of local news
Value co-creation: posting and earning from the post

Channels

Workshop
Physical places: incubators, coworking spaces
Events: makerfaires, local business and engineering conferences
Landing pages

Customer Segments

Segmented:
Young, innovative officers, engineers, business people
Existing online newspaper publishers
Event organizers
Public organization and institutes

Cost Structure

Employee salary, Marketing campaign
Advertising via google, facebook, etc
Content generation
Software development

Revenue Streams

Subscription model
Monetization of photos, news

Figure 2.1 Example of Business Model Canvas extracted from Dahle et al. (2020)

- *Revenue Streams* that describe how the company will profit.
- *Key Resources* that addresses what type of resources the company needs to create in order to add value as proposed.
- *Key Partners*, which are the allies whose act helps make the value proposition possible.
- *Core Structures* which addresses the cost drivers to operate the business.

Figure 2.1 shows a canvas example. Dahle et al. (2020) explained that the business model was essential at the beginning of their project used as an example. Each element has evolved over time, as the team has gained experience with what works and what does not.

2.1.5 Lean Startup

Ries (2011) introduced the concept of Lean Startup in 2011, based on the principles discussed by Toyota (BERG et al., 2018). This method proposes a way to create and manage startups in order to deliver products or services to customers as quickly as possible. *build-measure-learn (BML)* is the process applied to achieve the Lean Startup objective, which aims to transform ideas into products in an interactive way, measure customer satisfaction and product change through customer feedback. *BML* is an interactive process where the construction of the project takes place, measurement of results, and learning for the next cycle.

The key to applying the *BML* process lies in Hypothesis testing, from building a Minimum Viable Product (MVP), which is the purest form of an idea, product, or service that can respond to the hypothesis. Any functionality, process, or effort that is not directly related to the belief is removed (BERG et al., 2018). The unit of progress for *Lean Startups* is called *Validated Learning.* Therefore, it must be understood that a Startup is an experiment to be validated through incremental growth plans, carried out only by efforts that are necessary (RIES, 2011).

In the *Lean Startup* everything that is done, each product launch, each *feature*, each *marketing* campaign, everything is considered as an experiment designed to search for *Validated Learning.* In this way, it is possible to systematically obtain what customers want and what they will crave to pay to obtain (RIES, 2011).

Ries (2011) emphasizes that a *Lean Startup* must also be prepared for failure, as this is where entrepreneurs learn the most. There are two general hypotheses for experimentation in *Lean Startups*:

- **The value hypothesis**: Seeks to test whether a product or service is delivering the value due to consumers who are using it.
- **The growth hypothesis**: Seeks to test new ways to get new users for a product or service.

The *Lean Startup* method proposes that the organization understands which product to develop through the above hypothesis tests, emphasizing the importance of delivering products to customers more quickly. Startups tend to focus on delivery speed rather than quality by eliminating formal Project Management processes, documentation, testing, and applying faster feedback forms such as customer satisfaction with the product. It is correct to say that Startups need specific development practices to solve the challenges proposed by the *Lean* methodology (BERG et al., 2018).

2.2 TECHNICAL DEBT IN SOFTWARE STARTUPS

Technical Debt (TD) refers to issues related to the quality of source code. It is associated with technical decisions in software development that can bring benefits in the short term, like savings of time and cost reduction. However, these decisions may bring some risks to internal software quality, hindering software products' maintenance and evolution. As far as the academy has studied this topic, it discusses whether, if not all, software projects face TD (TOM et al., 2013; AVGERIOU et al., 2016).

Nowadays, the Software Engineering community states TD as *"a collection of design or implementation constructs that are expedient in the short term but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability".*

Apa et al. (2020) reference an exploratory study done in startups in Brazil. The results show that after the initial stage, the management of TD is claimed as essential in the software development life cycle for startups. Furthermore, the perception of software product quality tends to change over time. At first, the study considered usability a significant characteristic since the main goals are related to the software product's acceptance and success in the market. When the product has a high probability of success or changes occur in the team or the number of clients/users increases, the quality concerns are associated with other characteristics such as maintainability and evolvability to meet the required changes and product scalability. Once the perception of software products evolves in startup organizations, their adopted software engineering practices also need to evolve (APA et al., 2020).

TD results can have a positive influence, if intentional and managed, or negatively, if unintentional and not managed, on the software projects and the quality of their software products. The perception and management of TD activities should be a continuous activity to mitigate internal software quality risks. Apa et al. (2020) point out fourteen types of TD, in table 2.1 shows ten types extracted to be used as reference is this work.

Startups intentionally gain some of the debts listed in Table 2.1 in the early stages. They tend to avoid engineering practices that consume time because their initial software solution is delivered as an MVP to experiment with new ideas and get fast feedback. Apa et al. (2020) made a replication to the Brazilian study above mentioned in Uruguayan software organizations.

First, the study has found no consensus about managing the TD types described in Table 2.1. Instead, the participants stated that TD identification and prevention are the essential activities to their projects, followed by TD communication and measurement (APA et al., 2020).

To support TD Management activities, the startups reported using tools such as JIRA and Trello. However, Apa et al. (2020) claim that the effectiveness of such technologies for managing TD activities needs further investigation.

Generally, TD awareness is more present in senior (experienced) software engineers and more mature startups. Since no consensus was found, more studies about handling specific TD activities in startups should be done to mitigate the problems in more advanced stages (APA et al., 2020).

2.3 SOFTWARE ENGINEERING FOR STARTUPS

Startups are flexible. The use of rigid processes does not serve this type of organization, which focuses on the rapid delivery of features to customers. Due to its limited resources, there is a greater focus on product delivery than following rigid processes (BERG et al., 2018).

Berg et al. (2018) carried out a systematic literature review about Software Startups. The authors found that early-stage Startups prefer not to follow systematic Software Engineering processes but instead use light and ad-hoc ones. The study noticed that startups need an engineering model based on innovation, which fits in complex and chaotic situations in which a startup goes through.

Table 2.1 Techinical Debt Types (APA et al., 2020)

TD Type	Description
Architecture Debt	Refers to suboptimal architecture solutions, impacting the internal quality (e.g., violations of the adequate and adopted architectural)
Build Debt	Refers to issues in the build process that may make this process harder (e.g., files of build containing code source that does not add value to this task and software products and manual build)
Code Debt	Refers to the issues in the source code that may hamper the modularity, reusability, analyzability, and modifiability of the software products (e.g., code source that does not meet the required coding standards)
Documentation Debt	Refers to the issues found in the documentation (e.g., lack, insufficient, outdated, or inadequate documentation of the artifacts' software)
Design Debt	Refers to technical shortcuts taken in the detailed design and may be found by analyzing the source code or design models (e.g., violations of the principles of good object-oriented design, code smells, and grimes)
Requirement Debt	Refers to the distance between the optimal requirements implemented and the actual software products implementation, under domain assumptions and constraints (e.g., an implemented requirement which, in a way, does not fully satisfy all the nonfunctional requirements)
Service Debt	Refers to the inadequate use of software services (e.g., poor selection and use of software services). Services refer to independent technologies that offer specific business functionality. These are described in a standardized way, having published interfaces and communicating with other services through remote calls
Test Debt	Refers to issues related to testing activities (e.g., lack, insufficient or inadequate tests; low tests coverage; and deferring testing)
Versioning Debt	Refers to issues related to source code versioning (e.g., unnecessary code forks)
Process Debt	Refers to issues related to the adopted software process that may contribute to the incurrence of TD (e.g., an inadequate process not providing proper support to the development activities)

One of the models presented is the Hunter-gatherer cycle, which Snowden and Boone (2007) proposed to assist startups in all organization phases, from the evolution of ideas to commercial products. The model is divided between the hunting cycle and the gathering cycle, focusing on growth activities in Software Startups over time (NGUYEN-DUC et al., 2015). The cycles happen at different times and can last from one day to a month, depending on the current order/disorder stage of the startup and the solution domain, from the unknown to the known. Hunger Gathering specifically assists decision-making in different scenarios from the chaos to complete organization (SNOWDEN; BOONE, 2007). As a result, correct choices are made based on each domain presented and defined in Figure 2.2.

Each domain requires different actions. The contexts *simple* and *complicated* are part of an ordered universe, where the transmission of cause and effect is noticeable, and the correct answers are facts determined. In the contexts *complex* and *chaotic*, they are part of a disorderly environment, where there is no apparent relationship between cause and effect, and the way of driving is through emerging standards of management. While the fact management is the base of the *ordered* world, the disordered world represents standards-based management (SNOWDEN; BOONE, 2007).

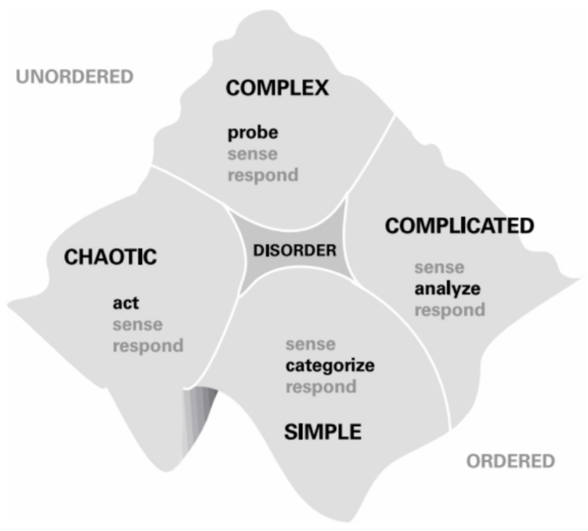


Figure 2.2 Cynefin Framework Diagram, adapted from Snowden and Boone (SNOWDEN; BOONE, 2007)

The context of *disorder* defines that it is particularly challenging to understand when the organization is in this domain. The way to break this barrier is to divide the situation into small parts, assigning each one to one of the four domains discussed above. From that, leaders can make decisions and intervene appropriately.

Another model of Software Engineering for Startups is called *Stairway to Heaven*, where it also focuses according to the current startup stage through the implementation of different Engineering concepts. At first, Olsson, Alahyari, and Bosch (2012) state that startups use traditional and agile engineering methods altogether, so this model aims to get the best of each method. Traditional engineering methods are waterfall-based, which means that one phase (requirements, design, implementation, verification, and maintenance) performs after the other in either a strict or flexible way. On the other hand, small deliveries under short development cycles with pre-established achievements are the base of Agile engineering methods (BECK et al., 2001).

The *Stairway to Heaven* model is based on five steps that a Startup should perform until it reaches the mature organization state. For example, one of the steps expected to achieve is the Continuous Integration (CI) through the establishment of practices that allow automated testing and delivery. Continuous integration is an automation process for developers. Successful CI means new code changes to an app are regularly built, tested, and merged to a shared repository. After implementing this practice, the startup should move on to the Continuous Delivery (CD) stage, which usually means a developer's changes to an application are automatically bug tested and uploaded to a repository, where they can then be deployed to a live production environment by the operations team. It is an answer to the problem of poor visibility and communication between dev and business teams. To that end, the purpose of CD is to ensure that it takes minimal effort to deploy new code. Then, the startup is ready to develop the Software as Experiment stage, where the engineers will add more software value according to the user input, similar to the aforementioned Lean Startup. Software delivery focus on improving the existing product rather than creating the new product (OLSSON et al., 2012).

A more recent Engineering model is called *Greenfield Startup*, which explains the priority that Startups have to launch their products on the market immediately. This strategy allows Startups to check if the product is suitable for the market, allowing a faster adjustment according to users' first feedback. Giardino et al. (2016) explain that it is necessary to reduce the time of *delivery* of the product to the market by accelerating its development by using low-precision engineering practices. Still, its adverse effects are offset by the fact that restructuring is mandatory before focusing on the *growth* stage of the business (GIARDINO et al., 2016).

The above models follow the same core concepts inherited from any startups and discussed previously. Such fundamental similarities do not make them generic. In fact, it is the opposite, where each model applies those fundamentals differently, to be used by Startups in specifics contexts. Similarly, a model specially designed for White Label software projects in startups would help mitigate the risks, reduce the technical debt, and face the challenges involved.

2.4 WHITE LABEL AND PRIVATE LABEL PRODUCTS

The term *White Label* for physical products is accompanied by another term, called *Private Label*, both refer to the model in which a company purchases products to apply its branding before selling to the final consumer. In this way, several brands buy the same product to resell as if they have made it (KAISER, 2017). There is a clear difference between the two terms mentioned above. However, *Private Labels* did not appear in the Software environment since no mention of it was found in the technical literature. In sum, *White Label* and *Private Label* are very complementary concepts treated simply as White Label Products by the authors from gray literature referenced in Appendix B.1.

A *Private label* is a product manufactured especially for a reseller, who adds his brand to the product and sells it to the final consumer. This product can follow specifications requested exclusively by the dealer, which can be concerning the type of packaging, recipe modifications, and others (GEYSKENS et al., 2018). The difference between *Private Label* and *White label* is that unlike the first that fulfills a specific order, White Label physical products are made on a large scale and moved to several reseller companies, which in turn add their brand to the product. There is no customization between brands in this case.

The *Private Labels* was created to be the cheapest products, but over time, they have been improving their quality, being today comparable to products of mainstream brands (GEYSKENS et al., 2018). Geyskens et al. (2018) brought an example of different *Private*

Label based on the type of final consumer, as Figure 2.3 illustrates. The *economy* jelly has both a more modest design and less fruit for every hundred grams of the product. In this way, the quality increases for *Standard* and *Premium* versions. This structure is an example of a *Private Label's* architecture.

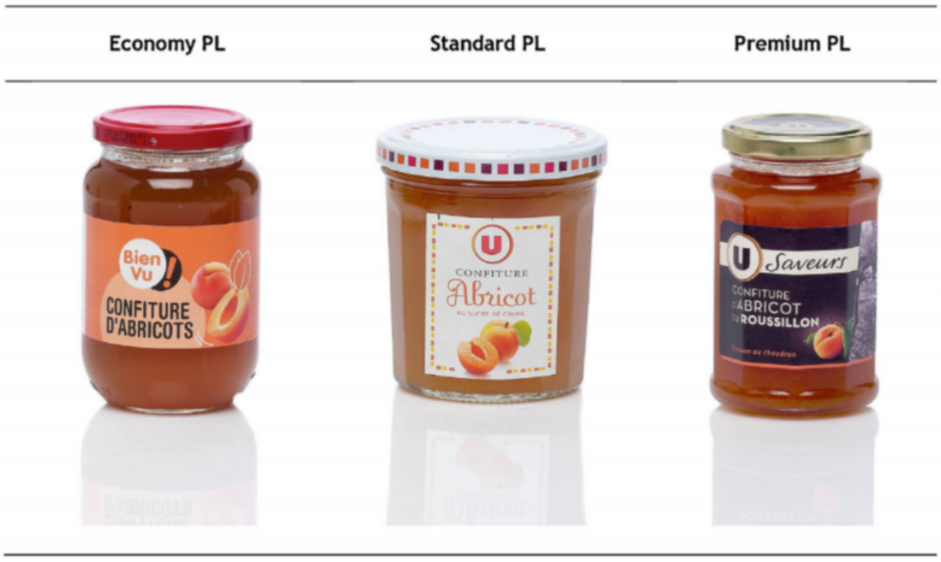


Figure 2.3 Example of a *Private Label*, adapted from Geyskens et al. (2018)

White Label or *Private Label* products tend to follow an application planning based on the market, based on a definition of variable points. The three types of gelly defined in Image 2.3 could be represented in White Label software projects as software for three types of users. A more expensive version would have the finest features implemented. In contrast, the cheapest one would have ads.

2.5 HIGHLY CONFIGURABLE SYSTEMS

Software development has been changing over time. What used to be based on the production of individual programs is created to conceive *families* of systems from a common set of artifacts, also known as *core assets*. A HCS proposes to adapt a software system to a particular context, which may vary in terms of the platform used, interfaces hosted, or the optional features it can provide to users. In a HCS, planning the variable and non-variable points from design to implementation is essential to apply systematic reuse to the code. With systematic reuse, the product will enable multiple configurations and serve as its purpose (COHEN et al., 2008).

Software variability is the ability of a software system or artifact to be changed, customized, or configured for use in a particular context (GURP et al., 2001). According to Bosch (2004), a high degree of variability allows the use of software in a broad range of contexts.

Before continuing the discussion of HCS concepts, we need to clarify what the literature says about *features*, which can be considered as the high-level artifacts needed to *design* a HCS. *Features* are intentions of *stakeholders* about the business, or design and implementation artifacts, used to make the variability in HCS happen.

Kang et al. (1990) define *feature* as *"an important or distinct aspect visible to the user, which represents a quality and/or characteristic of a software system"*. On the other hand, Batory (2005) defines feature as *"an addition to a program's functionality"*, representing *feature* within the project's implementation level and being an important part in the construction of *software*.

The definition used in this work is from Apel et al. (2016), which state features as *"a characteristic or behavior visible to a system user"*. They are used in HCS to specify and communicate differences and similarities between *stakeholders* to guide the structure, reuse, and variation in the software development and configuration phases of a HCS project. Features in HCS projects can be optional or mandatory, and they can have relationships where an optional feature is only available when another one is selected.

Kang et al. (1990) describes a widely-accepted representation of features in a Software Product Lines (SPL). The feature model encompasses a high-level representation of mandatory and optional features as well as their relationships. Within a feature model, the software engineer can visually analyze the current state of a product and what are the missing points where it can evolve. Figure shows an example of a feature model, displaying many relationships between mandatory and optional features.

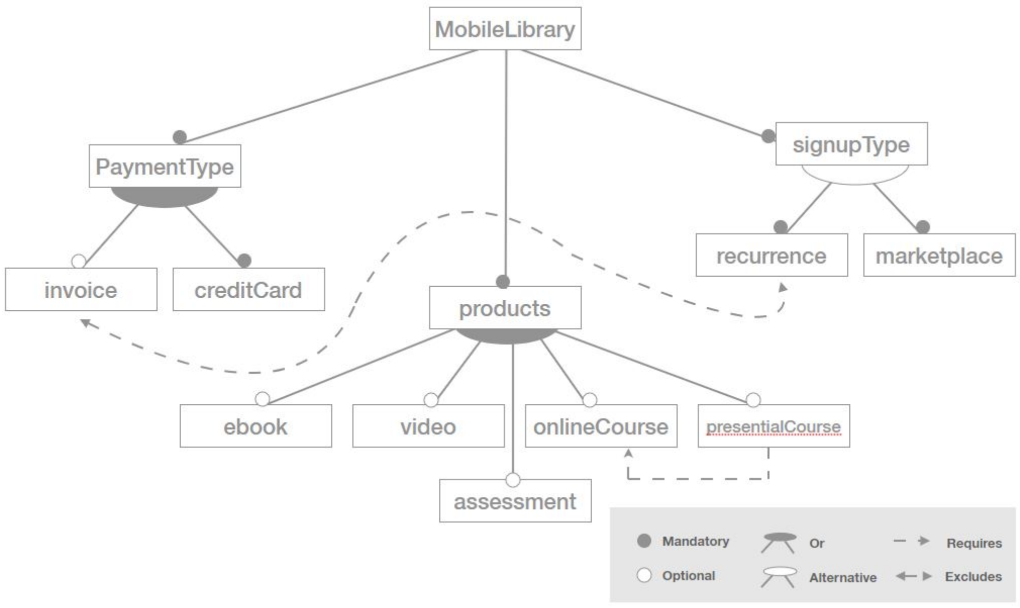


Figure 2.4 Feature model example developed by author

2.5.1 Feature Binding

Feature binding is a process inside HCS systems where the optional features defined in the product configuration are bound. A product configuration is where the characteristics of the product hold. Apel et al. (2016) describe three moments in which feature binding occurs during product derivation, also called variation pattern: compile-time, load-time, and runtime binding. In the former, developers should decide what feature will be included in a product or not during the product's build time. Next, the load-time binding is when a product configures itself before it loads to the user, usually after startup. In the latter, the runtime binding allows for changes in the product while it is running.

The SPL paradigm is an example of HCS where *feature biding* can occur in the product compilation or load stage. We can define a SPL as an architectural model that establishes a *family* of products from a code base and are customized through common points (variability) previously defined. SPL projects are usually stand-alone products that have their specificities, but when beside their family, displays many commonalities. The variability points help the developer allocate different variants in features while maintaining the system's architectural pattern.

Some systems dynamically reconfigure themselves. It is when *feature biding* happens at the moment of execution (*runtime*). Cohen, Dwyer, and Shi (2008) cites, as an example, a NASA system that uses online planning to either enable or disable modules based on the state of the mission and the aircraft. The configuration is done dynamically, on execution time, and without human interference.

A typical class of user-configurable systems could also fit in HCS project. These programs, which can be *desktop* applications, *web servers* or databases, allow users to modify them from a series of pre-defined options, through parameters configured via the command line, or configuration files (COHEN et al., 2008).

At the beginning of a project, to assist developers in choosing the most suitable variation pattern, Metzger and Pohl (2007) proposed questions to be asked:

- *What will vary?* The answer to this question leads to establish the variable features in the project. Ex: The artificial gravity module in the NASA system.

- *How will it vary?* To answer this question, the engineers must identify all possible instances of a given *feature*, which are called variants. Ex: Variations of the "artificial gravity module" variation point can be "enabled," "disabled," and "enabled in certain areas of the aircraft."

- *Why should this vary?* This question takes a more rational view of the defined variation point and variant. From the examples shown, keeping gravity in certain situations can consume unnecessary energy.

- *For whom is this variation point documented?* This question identifies the target group for the variation point where the engineering team will document the software variability and define who will have access to it.

Planning a HCS is challenging because it is necessary to use methods that make the variability and systematic reuse feasible in practice. In addition to the definition of binding time, Apel et al. (2016) present two more dimensions of the variability application in HCS to consider:

- **Technology**. The first dimension is related to the technology used. There are two mechanisms in programming languages that help variability management. The first is called *language-based approach*, which allows variability to happen within the code. Another application of technology-based variability is called *tool-based approach*, which are automation tools responsible for applying variability. While the first mechanism focuses on direct modifications to the code, making it easy for developers to understand, in the second, the management of variability is completely separate from the code to simplify its structure.

  In the *language-based approach*, the conditional code is responsible for activating or not optional features. A system where configuration variables decide more than just the database connection, but the entire system's behavior is a perfect example of HCS using the language-based approach.

  In the *tool-based approach*, specific programs are responsible for applying optional features to the product. Gradle is an example of an automation tool that inserts variability when deploying a new product[3].

- **Exhibition**. The second dimension delimits how the code used will be exposed after the variability is bound. There are two common methods for implementing variability, the *Annotation-based approaches* and *Composition-based approaches.* In *Annotation-based approaches*, a mark in the code is responsible for each variable requirement, which disables unused functions after applying the variability. In the *Composition-based approaches*, software tools do the product configuration by attaching code blocks during the binding time, creating a final product based on the requested features. In the Figure 2.5 there is an example of each approach.

  On the Annotation-based approach, Figure 2.5 shows that every feature presented in the feature model is present in many software component parts, and enabling or not a feature would disable it in every part.

  Figure 2.5 shows that features are more isolated for each software component, which is the main characteristic of the component-based approach. Large blocks of code are enabled or not in the product generation.

2.5.2 Quality criteria for SPL Engineering

SPL is a multidimensional engineering process that software engineers should consider from different perspectives. First, it is necessary to consider both the problem space and solution space. While the problem space takes the perspective of stakeholders and

[3]Available at https://docs.gradle.org/current/userguide/what$_{is_g}radle.html$

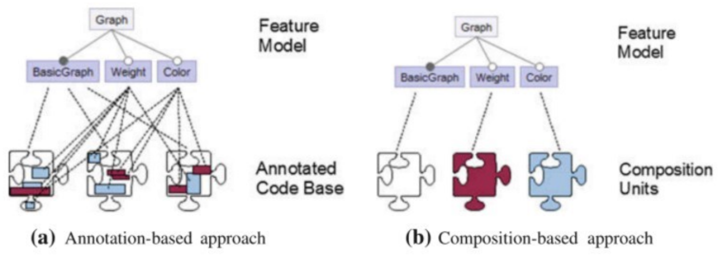


Figure 2.5 Annotation and Composition based approaches, adapted from Apel et al. (2016)

their expectations, the solution space represents the developer's perspectives capable of handling the former (APEL et al., 2016). Second, there are two engineering processes to consider, namely domain engineering and application engineering. While the former involves analyzing the target domain for the SPL and developing its artifacts, the latter is responsible for using the artifacts created in the domain analysis to create the target product.

An SPL implementation should meet the following criteria (APEL et al., 2016):

- **Low preplanning effort:** Which means ease the anticipation of changes and sources of reuse; this quality criteria indicates that an SPL should have a low effort on planning new products;
- **Feature traceability:** It is the ability to trace a feature from problem space to solution space;
- **Separation of concerns:** It is the capacity of code separation by features from other code parts, decomposing them into semantically cohesive code fractions;
- **Information hiding:** It means hiding the key aspects of a feature that is split into external and internal parts;
- **Granularity:** Disrespects the level of change that features would impact on others. They can be either fine-grained, like adding a new statement to a given method body, or coarse-grained, whenever a large number of modifications in the software;
- **Uniformity:** Gives the idea that all software artifacts should be encoded and synthesized in a similar manner.

The above criteria help ensure success in SPL projects. By learning them, we can adapt the White Label software projects and mitigate technical debt in startups that work with it.

2.6 CHAPTER SUMMARY

This Chapter introduced the fundamental concepts that are relevant to this work. First, we introduced the startup context and its core concepts. Software Engineering for startups was also covered in this Chapter, where we introduced three software engineering techniques.

With the startup core items covered, this Chapter introduced the origin of White Label software projects. They come from physical products and are present in everyone's life.

HCS was the next topic explained in this Chapter. It makes the software adaptable to a particular context. A type of HCS is called SPL, which are software families created from a common code base.

HCS and more particularly SPL, can bring more quality over White Label software projects. The correlation between them served as the foundation of this work.

Chapter
3

INTERVIEW STUDY

To understand White Label projects in startups and answer the first question in this work, we conducted semi-structured interviews with CEOs/CTOs from four software startups that develop White Label software. This chapter will introduce the semi-structured interview study definition, how we performed it, and an analysis of the yielded results. The audio transcription from the interviews is available in Table 3.2.

3.1 INTRODUCTION

The use of interviews in empirical research aims to collect data on phenomena that cannot be measured using quantitative metrics (HOVE; ANDA, 2005). Interviewing people directly related to the studied context ensures that their opinions, thoughts, and feelings are fully transcribed.

According to Hove and Anda (2005), there are two classic types of interviews: structured and unstructured. Structured interviews have a clear and specific objective for the type of information required in the interview, meaning that questions must be precise, allowing the answers to be quantified. In unstructured interviews is the opposite of the first. Semi-structured interviews combine the best of the two previously mentioned types of interviews. Therefore, Hove and Anda (2005) suggest combining two types of questions, specific (with a clear objective) and open questions (to obtain unexpected information). The following activities are necessary to conduct semi-structured interviews:

- **Schedule:** It is necessary to make appointments with the interviewees. In this stage, there is also a sub-selection to decide whether an analysis of the best candidates is necessary.

- **Collection of Specific Information:** It is necessary to know the study's object and its interviewees. It is essential to collect available information on the topic and the interviewees before conducting the interview (Curriculum's, about the company to be interviewed).

- **Preparation of the Interview Guides:** Interview guides are necessary to keep the conversation around the topic. The amount of time needed to prepare the guide varies depending on the study or whether adaptation would be required for each candidate (HOVE; ANDA, 2005).
- **Discussions/meetings:** It is necessary to have meetings between the researchers involved in the study before, after, or between interviews.
- **Audio Transcript and summary:** In this phase, we transcribed the audio of the interviews and summarized it for the study documentation. The literature states that it takes 3 hours to summarize 1 hour of audio material.

All these steps were applied in our study, which encompassed the following steps: *search the subject*, *leverage a set of qualified startups for the initial survey*, *schedule the interviews*, *organize the guide*, *conduct the pilot*, *reorganize guides for the next interviews*, *conduct interviews*, *transcribe audio*, *analyze the transcriptions and interview notes*, and *write results*.

Hove and Anda (2005) displays the skills required for conducting interviews by the researcher:

- Encourage the interviewee to speak freely, without judgment;
- Ability to ask relevant questions that do not deviate from the interview's focus but explore interesting topics;
- Good interpersonal skills;
- Excellent conversational skills.

Hove and Anda (2005) state that one of the pilot's functions is to train the interviewer, who must be prepared to obtain information from motivated and non-motivated respondents.

There are six types of questions to be used in semi-structured interviews:

- Those of *behavior/experience* that describe experiences, behavior in situations, and actions.
- Questions about *opinion* that investigate what the interviewee feels about specific questions.
- *Sentimental* questions that serve to understand emotional responses about experiences.
- *Knowledge* questions that seek to identify what information the interviewer has.
- *Sensory* questions that seek to obtain experiences of the senses.
- *Demographic* questions of the interviewee, which seek to understand their prior knowledge on the topic in general.

Hove and Anda (2005) make it clear that in all cases, questions that require only a *yes* or a *no* as an answer should be left out.

For correctly conducting the interviews, researchers should have specific tools and artifacts. The first highly recommended tool is the use of a *an audio recording instrument* that allows the complete apposition of the content of the interview, allowing a smoother execution of the interview without having to pause it for the complete annotation of the answer. Popular tools for analyzing qualitative research like Atlas.ti[1] or NVivo[2] can also help the interviewer after transcribing the interviews.

Especially in interviews about software development, it can sometimes be challenging to understand the interviewee's questions. For this, researchers use *Visual support artifacts* to help understand the question asked and clarify the answer. Usually, it serves as a mental trigger for the interviewee to answer the question more objectively.

We used the above concepts as the basis for creating the study defined below.

3.2 CONDUCTING SEMISTRUCTURED INTERVIEWS

In our study, We carried out the interviews in the All Saints Bay Ecosystem[3], located in Salvador - Bahia, Brazil, at the end of 2018. For each startup interviewed, it was mandatory that the CEOs and the CTOs needed to be involved in the software development and its planning process of the White Label projects.

The questions asked intended to understand how the interviewees defined White Label projects, how they managed these projects' product generation and the definition of variation points between products.

3.3 RESEARCH DESIGN

We used the Hove and Anda (2005) survey's design in this work, which involved the following steps for data collection and analysis: (1) define the protocol for the interview[4]; (2) define the questionnaires; (3) conduct the interview; (4) transcript the interviews; and, (5) perform qualitative analysis. Figure 3.1 shows an overview of the Interview Study's research design.

3.3.1 Research Questions

With the interviews, we sought to begin to unveil the White Label software concept and its feasibility, investigate how software startups used to apply it, and investigate if advanced code reuse techniques, such as Highly-configurable Systems (HCS), could be used as an opportunity lever for software startups.

Therefore, we planned to answer the following research questions (RQ):

[1]https://atlasti.com/

[2]https://www.qsrinternational.com/nvivo-qualitative-data-analysis-software/home

[3]http://allsaintsbay.com.br/

[4]Supplementary material are available at (SILVA et al., 2020a), the audio transcriptions summarized is available at Appendix A.1

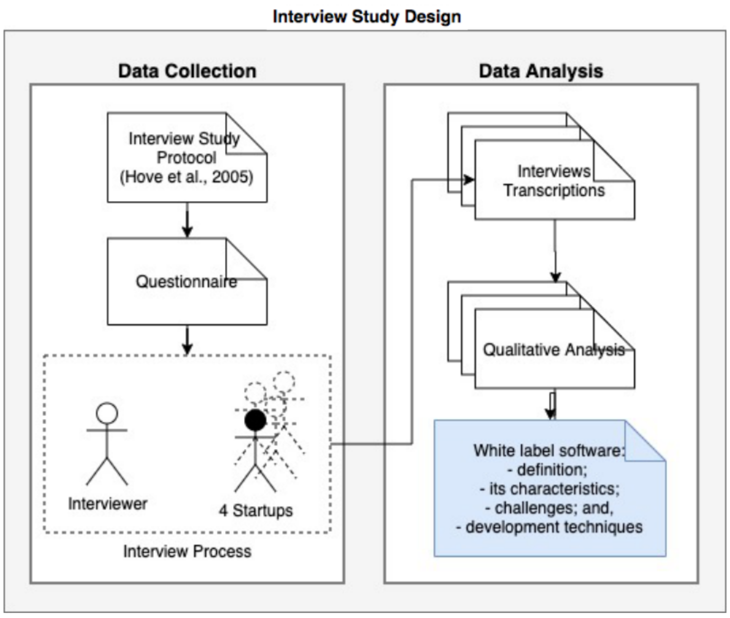


Figure 3.1 Interview Study Design developed by the author

RQ1 - To what extent is the industry professional's perception of White Label software products'? This RQ aims to unveil the concept of White Label software and its feasibility for software startups by practitioners.

RQ2 - What are the main White Label software characteristics? This RQ aims to identify the White Label software characteristics.

RQ3 - What are the software development strategies used to implement White Label software? This RQ aims to gather the techniques in developing White Label software.

RQ4 - What are the main challenges in the development of White Label software? This RQ aims to investigate the challenge in the process of developing a White Label software.

In addition, both RQ3 and RQ4 aim to shed light on if software startups could use HCS as an opportunity lever.

3.3.2 Characterization of subjects

We have defined some criteria for selecting startups: (i) they must belong to the All Saints Bay Ecosystem (SEBRAE/BA, 2016) because the interviews were carried out in the startups' workplace; and (ii) the startups must be involved in the development of White Label software.

The All Saints Bay is a Brazilian Ecosystem of startups from Salvador-BA. First, a search was done on the ecosystem's official website to find Startups with White Label characteristics. We validated the data obtained and selected ten companies that could be unduly linked to the study and possess stable products.

The selected startups were contacted by email, and four have responded positively to the interview invitation. Table 3.1 presents the list of startups interviewed for this study. We will refer to the startups studied with the SX label, where X is the letters from A to D for confidentiality reasons.

Table 3.1 Characteristics of the Startups Cases and Interviews

Case	Operating Market	Year	Size	NIP	IR
SA	AgroTech	7	Small	1	CTO
SB	EduTech	5	Small	1	CEO/CTO
SC	Fintech	3	Small	1	CEO/CTO
SD	Urban Mobility	2	Small	1	CEO/CTO

NIP: Number of interviewed people
IR: Interviewees roles

- **Startup A**: Startup focused on animal production. It creates personal stores for producers to sell high-performance animals (horses, swine, and cattle) all over Brazil. It offers online stores *White Label* so that partners can sell their products to end-users. It has been on the market for five years and has fifty stores online. Startup A is between the end of the stabilization stage and the beginning of the growth stage.

  Startup A has a very mature Software Engineering process but also owns some gaps in software variability management. Since their White Label software is simple, they were not facing scaling problems. Engineering is unaware of the many variations available and does not track its changes. However, they possess a successful product with partners from over Brazil.

- **Startup B**: White Label education startup. It licenses technology to book editors, so they can have a mobile app store for selling and consuming their e-books and courses. Startup B has over five years of existence, has more than 12 products generated, and it was the most complex project among the studied. We considered this startup in the stabilization stage.

  This startup has the most complex White Label software implementation, encompassing small optional features and design tweaks and a business model for the generated product. It was under the stabilization stage because the partner's demands were overloading the engineering team. The startup was not ready for growth because it lacked some structure, but its products were well-stabilized in the partners.

- **Startup C**: Financial startup in the stock exchange market. This startup type is commonly called *fintech* by individuals belonging to the Entrepreneurial Ecosystem. The White Label product in startup C works by allowing stock consultants to have its platform for client management and manipulate its stock portfolio. This startup is in the startup stage for the White Label software product developed. Startup C was finishing up its White Label product. It had one partner waiting for delivery, but the software was under the first MVP. For this reason, we evaluated Startup C in the startup phase besides its mature software engineering process.

- **Startup D**: Work by providing personalized apps for urban mobility, so groups of drivers can have their Uber-like app for transporting passengers. It has a total of thirty apps running and has the shortest lifetime on the market among all respondents. This startup is at the growth stage. We evaluated Startup D in the growth stage because they have many partners over Brazil, are well known in the market, and are aware of their software variability.

3.3.3 Data Collection

We built a questionnaire with 26 questions and conducted the interviews by following a protocol summarized in the list of questions (see Table 3.2). The protocol was created and tested in the pilot interview with Company C, and the result obtained was entirely satisfactory, enabling replication with other companies. During the pilot, it was necessary to induce questions not initially contained in the protocol, signaling a deviation from the original plan. The questions asked unexpectedly in the pilot were added to the protocol of the other interviews. Under this condition, we could add startup C to the results.

The interview questions were sent out to the participants 24 hours before the interview so that they could prepare themselves for the interview. We conducted the interviews at the startups' workplace and lasted an average of ten minutes. Since the interviewees knew the questions before, their answers were compact.

In the first part of the interview process, the interviewer gave a consent form to the interviewee, where he agreed to participate in the research. The first three questions were about the interviewee, where we asked about his role in the startup, his experience on it, and asked whether it was the first innovation project he participates in.

The following three questions referred to the startup's age and the dependency on external investors. This type of question would give the study insights about which stage the startup interviewed would be. The next question asked the respondent to explain, in their own words, the definition of White Label software. Then, we asked about their White Label software project developed and the idea behind it. Therefore, the following 18 questions in the interview were about the startup's White Label software project, its characteristics, development techniques, technologies used, supportive systems developed, and challenges the startups commonly face. The complete list of questions is available in Table 3.2.

The interviewer asked the questions in a way that looked like a conversation between business partners, meaning that we tried not to lock the interviewee at the end of the answer to each question in order to let him follow his line of reasoning. The interview was voice recorded, and the interviewer took notes. The rationale behind each question is available in Appendix A.1.

Table 3.2 Interview Questions

#	Questions
#	**Respondent Characterization**
Q1	What is your position in the company?
Q2	What is your experience time in years?
Q3	Have you worked on innovation projects? If so, how much experience do you have?
#	**Startup Current Stage**
Q4	How long has your startup been around?
Q5	Is your startup self-sustainable, that is, does it have enough financial resources to support itself?
Q6	How much dependence on investors?
#	**White Label Product**
Q7	What do you mean by white label?
Q8	Describe your white label product
Q9	How the generation/derivation process for a new product is done? how do you manage product versions?
Q10	Are mobile applications developed in a native or hybrid way?
Q11	If native, does it have an iOS and Android version?
Q12	If hybrid, which framework do you use?
Q13	How many days does it take to generate a product?
Q14	How much financial resources is spent during the process of generating a product?
Q15	How much does each resource cost?
Q16	How do you define the variability points?
Q17	What is the variation pattern?
Q18	How many optional features do you have?
Q19	How many products do you have launched and are available today?
Q20	How many products have you made?
Q21	Describe the system architecture.
Q22	How many databases do you use?
Q23	How many web services do you use?
Q24	What are the support management systems?
Q25	What is the cost of server today in the company monthly?
Q26	Do you perform software testing?

3.3.4 Data Analysis

After gathering the data, we transcribed the interviews' recordings, which can are available in Appendix A.2. The data obtained were analyzed qualitatively. Even though the startups were from very different fields, they presented the same issues regarding White Label software projects and were at the stabilization stage. We present the results and discussions in the next section.

3.4 RESULTS

(RQ1) - To what extent is the industry professional's perception of White Label software products'? The interviewed startups presented complementary definitions for White Label software projects. SA claimed that White Label software is like *"selling a product to multiple customers to sell their products to the end-user"*. SB defined White Label software as *"a business model that focuses on component-based architecture and product adaptation to create customized, low-cost applications"*. SC stated White Label as *"a product that you can give other faces and other features as needed"*. SD defined *"a White Label software product as a product that you license the technology, customizing some features, making it available for use by third parties - exploring the product to sell different versions"*.

(RQ2) - What are the main White Label software characteristics? We observed that the startups interviewed believe that developing a White Label is quicker when compared to stand-alone projects.

The main characteristic that we observed was that a White Label software is (i) one system that can hold many products, and (ii) scripts, configuration files, global variables, customizes the product by using *customers* personal information contained in folders. SA stated the benefits of this approach, *"we were able to quickly generate custom applications for customers already operating in the same niche market"*. SB stated that *"A folder is created with more customizable information from the clients. All products follow the same code version"*.

SC stated, *"the system has been parameterized to understand what the customer needs. A manager controls all optional features. The version is the same"*. SD presented the same pattern for software customization, *"flavors are made on android, as a settings folder for each client, with logos, colors, and specific information. For each client, we modify these assets"*.

(RQ3) - What are the software development strategies used to implement White Label software? The startups use a mix of configuration files, global variables, automated scripts, and control panel parameters. The process of generating a White Label software version, if not manual, uses low-level scripts to move files through project folders and build the release. SA claims that *"uploading versions are manual activity"* and complements *"the application and the server have the same code that is the basis of the products. There is already a deployment process on the server, and the product is automatically generated for each customer. However, for applications, everything is manual"*. SB states that they use scripts to automate the software generation, *"lastly, the generation scripts are modified to include a new app. All products follow the same version of the project code"*. For SC and SD, the build process is also manual, following platform specifications.

Reuse artifacts are built and never updated. There is no specific assessment for managing variability. Furthermore, the traceability of code changes can only be made visually by experts. They made use of some agile methodologies to develop these applications

(SOUZA et al., 2019). However, the interviewers were unable to answer how the software engineering process differs from a typical SCRUM implementation due to the different nature of White Label software.

The system architecture is based on components as much as possible in order to speed up code reuse. They use version control software tools like Git to support this activity. Each customer application is released individually. Most of the applications developed by the reported startups use hybrid frameworks (cordova[5], react-native[6], Xamarin[7]). The client drives the application's development, informing which behavior he expects. In addition, optional features are either created or adjusted as needed.

The partner validates the software development process. Startup SC defined the product's scope as follows: *"we involved three consultants to validate the platform and checked whether or not they agreed with the day-to-day activities, planned activities, and changes in the process"*. SB was not supposed to be White Label and used a reactive approach, as it states *"This project was made to be stand-alone, but other companies requested our solution. We created basic points of variability that were increasing as new partners entered and presented their needs"*. SA and SD defined their scope inside their team, as they had prior knowledge in the field as SD stated, *"because I already had a basic product from another business that I had failed, so by the time I started this business, I already had a certain range of customers to sell, and as for the technology I made adaptations"*.

(RQ4) - What are the main challenges in the development of White Label software? The interviewees exposed challenges when answering software-related questions, and our findings bases on the general understanding from them. The first challenge was to improve the process of generating new products taking into account the cost and benefit. While SA spends two days to generate a new product, SB spends half of it. The second challenge is related to trace code changes and how software engineers can add more optional features without affecting the old code. As an example, SD stated its strategy for controlling changes *"what change is the feature's visibility, we face problems maintaining old features and managing new ones"* The third challenge was to write down documentation and have it updated to support the product's development. The fourth challenge was to find the best scenario to launch the product and present it to new clients since the loss of this time-to-market might imply the loss of clients. SD states that if they are too slow on development, the possible partners can get ahead and build the product with their resources or get it from another White Label company.

3.5 DISCUSSION

When asked, the interviewed startups presented similar definitions for White Label software projects. Even though their products are from very different niches, they share common characteristics. For example, they follow the same pattern when creating a new

[5]⟨https://cordova.apache.org/⟩
[6]⟨https://github.com/facebook/react-native⟩
[7]⟨https://dotnet.microsoft.com/apps/xamarin⟩

product. First, they get the information needed to build the product, like its colors, logo, images, privacy policy, and other specific information. Then, they present an estimate for the product to be finished, which is very different from one startup to the other, and deliver the product to the partner having the startup's servers as host.

When speaking about software engineering techniques, they essentially use common startup practices (SOUZA et al., 2019) with almost no adaptation for the White Label Project scenario. What one startup does differently from the other is generating a new product, which encompasses both manual and automatic practices. The automation level depends on specific factors, like the number of people working on the project, the engineering team's seniority, the amount of feature development left, and the chief engineer's personal experience.

The challenges gathered from the interviews are also very similar. For example, controlling the many possible configurations is a challenge that is still somehow open by today's advanced techniques. Incorporating HCS to startups and adapting to their need could mitigate some risks and level down the challenges involved in developing White Label software projects.

Since no indication of usage of advanced code reuse techniques, such as HCS, has been identified in the interviews, it motivates further studies to identify possible use cases that handle the challenges gathered by applying the aforementioned techniques.

3.6 IMPLICATIONS FOR RESEARCH AND PRACTICE

The interview study has opened the possibilities for identifying gaps of knowledge, the common technical debt, on White Label Software Projects in Startups. Further studies should gather experience from different gray literature sources and personal experience in software startups, as shown in Chapter 4 and 5.

3.7 THREATS TO VALIDITY

Three threat types are present in this study. In the following paragraphs, we will explain what we made to mitigate the threats identified.

Construct validity threats were first identified. The coverage of the research questions could not have been well defined. We mitigated this threat by applying a pilot test. Another threat was related to the respondents' understanding of the questions. The interview questions were sent to the subjects 24 hours before the interview, and we answered any doubts that the subjects could have during the meeting.

An **internal validity** threat is related to data extraction bias. This thesis's author could add his bias to the research based on single data extraction and his experience. We have been able to mitigate this threat by discussing the data obtained during the data recording with a group of researchers.

The third threat group is related to **external validity**. The subjects interviewed may not know or provide the needed information for the interview; we interviewed CTOs and CEOs, which are high-importance positions in startups. Another external threat is related to the capacity of the generalizability of the achieved results. We believe that further replications, in other scenarios, could be useful in the sense that we could have the opportunity to compare the observations. This threat has motivated the other two studies related to White Label software projects.

3.8 CHAPTER SUMMARY

This chapter presented an interview study applied for startups that work with White Label software projects. The questions asked in the interviews aimed to understand this type of project, its characteristics, identify if and which are the software engineering techniques explicitly applied to White Label software projects, and the current challenges faced.

The result gives powerful insights about technical debts that software startups applying White Label projects face and indicates that HCS could be used to fill the current challenges if applied according to the specific needs present in the startup.

Chapter 4

SURVEY STUDY

The interviews with startups have shown that White Label projects are known not only by those who apply them but also by people inside innovation ecosystems. The applied survey aimed to understand the similarities between what was exposed by stakeholders working with White Label software projects in startups and what the ecosystem members know about the subject. This chapter will introduce the concepts of personal surveys, how we managed to conduct them, and this study's results.

By conducting the surveys, we confirmed the first findings from the interviews, validated the object of this work, and shared the White Label software projects concept in the academy by publishing a paper in SEAA 2020 (SILVA et al., 2020b).

4.1 INTRODUCTION

A survey is a comprehensive method of collecting information that seeks to describe, compare, or explain knowledge, attitudes, or behaviors. For the application of a survey, it is necessary to follow a set of steps (KITCHENHAM; PFLEEGER, 2008):

- **Establish an objective**: The first step in a survey concerns the process of defining research objectives. Each objective refers to an expected result or a question to be answered by analyzing the data obtained. With an established objective, it will be possible to define the hypothesis to be tested, what possible explanations to be investigated or excluded, and what resources are needed to reach the objectives found.

- **Survey's Design**: There are two types of design for a **survey**: the *cross sectional* and the *longitudinal*. While the former focuses on information over a specific time, the latter seeks to explore a population's information.

- **development of the study's instrument**: Survey instruments are generally applied questionnaires, which need to be constructed based on the population chosen. There are two types of questions: open and closed. Researchers should build open-ended questions to let the respondent answer the question by reviewing his experience; they should avoid questions that can be answered simply by a "yes" or "no." Respondents answer closed-ended questions rapidly, but the choices presented should be designed to cover what the respondent will expect. In sum, an excellent survey instrument should have a mix of both types of questions.

- **Evaluation of the instrument**: Once the researchers have created the study instrument, they should evaluate it. Also called a pre-test, it aims to check various topics, like validate that all questions are understandable, evaluate the instrument's reliability and validity, ensure that data analysis techniques match the expected responses, and check if there is research bias hidden in the instrument.

- **Obtaining valid data**: For data validation, it is necessary to define the total number of responses based on the study population, called *sample*. After validating the data obtained, the analysis begins.

- **Data analysis**: In this step, data analysis is performed and checked for integrity, consistency, and completeness. Kitchenham and Pfleeger (2008) say that it is rather important to have a policy for handling inconsistent and or incomplete questionnaires. In our example, the data gathered was divided into three parts. Each part contained specific questions that were revealed to the respondent based on his answers.

4.2 CONDUCTING THE SURVEY

In the following sections, we will describe the steps performed to design and conduct the survey and the results obtained after data analysis. We carried out the survey during the second semester of 2019.

4.3 RESEARCH DESIGN

We conducted the survey targeting stakeholders from the All Saints Bay Ecosystem. Like with the interviews, we sought to unveil the white label software concept and its feasibility, investigate how software startups used to apply the white label concept, investigate if advanced code reuse techniques, such as HCS, could be used as an opportunity lever for software startups.

Therefore, we planned to answer the following research questions (RQ):

RQ1 - **To what extent is the industry professional's perception of White Label software products'?** This RQ aims to unveil the concept of white label software and its feasibility for software startups by practitioners.

RQ2 - **What are the main white label software characteristics?** This RQ aims to identify the white label software characteristics.

RQ3 - What are the software development strategies used to implement white label software? This RQ aims to gather the techniques in developing White Label software.

RQ4 - What are the main challenges in the development of White Label software? This RQ aims to investigate the challenge in the process of developing a White Label software.

In addition, both RQ3 and RQ4 aim to shed light on if software startups could use HCS as an opportunity lever.

We have chosen the longitudinal survey type because we wanted the respondents to answer everything they knew about White Label software products (KITCHENHAM; PFLEEGER, 2008). First, in the data collection phase, the survey protocol was created[1], where the purpose of this study was defined. After defining the objective, we created the survey questions. Moreover, the survey instrument was built.

We carried out a pilot with five respondents, three from the target audience and two survey experts. With the pilot, we learned that a three-level depth survey would better serve this study since we would obtain richer data if the respondent that worked with White Label software products presented his experience. After these adjustments, we invited professionals via developer groups from the All Saints Bay Ecosystem, Instagram Story posts on pages related to startups, and individual emails. In the data analysis phase, we first analyzed the data collected in the survey. Then, we crossed with the information included in the semi-structured interviews, as Figure 4.1 shows.

The research design of the survey study involved the following steps: (1) define survey objective; (2) create the questionnaire; (3) construct the survey instrument; (4) recruit participants; (5) validate the instrument; and (6) collect data.

4.3.1 Survey objective

The survey study's primary objective is to understand White Label software projects from the practitioners' perspective in innovation ecosystems. Moreover, find whether their answers were similar to what has been exposed by stakeholders working with this project type. Comparing the results obtained previously with the interviews consolidates the understanding of White Label software products, characteristics, challenges, and software development approaches.

4.3.2 Survey Instrument

We prepared an online questionnaire encompassing 19 questions, a mix of six open and thirteen closed questions, which can are available in Table 4.1). We prepared the questionnaire in the Google Forms platform.

We divided the survey into three levels, i.e., a sequence of questions. Depending on the participant's characterization and their knowledge and involvement with White Label software development, the survey could either enable or disable a certain level/group of

[1]Supplementary material is available in Table 4.1

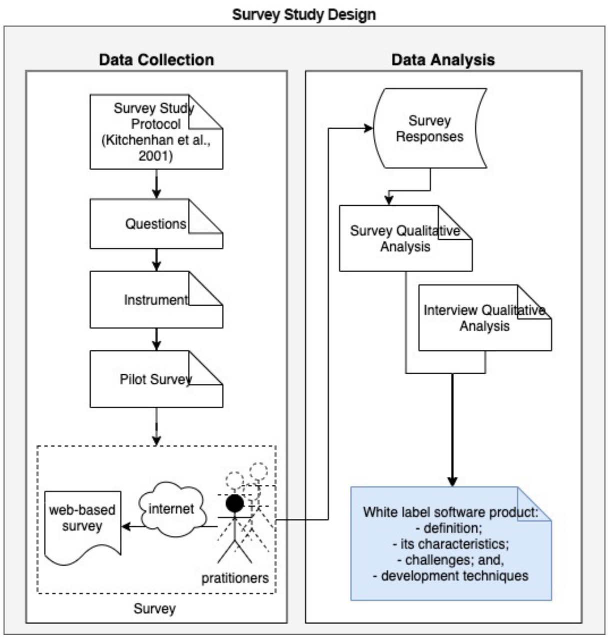


Figure 4.1 Survey Study Design

questions. Therefore, those who knew more about the subject of study would collaborate more. If the respondent did not know about the prerequisite, the survey would send it to the end, otherwise present a new level. With this approach, we could make the survey experience short and objective for the respondent. Figure 4.2 shows at a higher level the survey design.

We explain each survey's step below:

- **level 0 - Cover Page:** The survey's cover page brings an agreement where the respondent agrees to participate in the survey.
- **Level 1 - Respondent Characterization:** The first level's objective was to identify whether the respondent belongs to an innovation ecosystem and what role he fits into (e.g., developer, designer, entrepreneur), and if he has heard about White Label software, the survey will open a new level.
- **Level 2 - White Label Software Definition** At the second level, respondents were asked to choose an answer or fill a new one for designating the best meaning of White Label software for them. The default definitions were obtained in the interview study by the respondents. Next, the survey would ask where the respondent has heard about this term if he is aware of startups that work with White Label software projects. If a respondent had previously experienced White Label software products, the survey would open the third and last level.
- **Level 3 - Respondent Personal Experience:** The third level asked the respondent about his personal experience of working with White Label software projects. The survey asked about the benefits and challenges involved in the development

Table 4.1 Survey Questions

#	Questions
#	**Level 1**
Q1	In which state do you live?
Q2	What is your profession?
Q3	What is your experience in Startups?
Q4	What is the business domain of Startup in which you worked / work?
Q5	What ecosystem of Startups your company / are you closest to / linked to?
Q6	Have you heard of the term Startup White Label?
#	**Level 2**
Q7	Where did you first hear about White Label Startups?
Q8	Do you know any Startup that uses this concept?
Q9	Choose from the alternatives the phrase that best represents the meaning of white label software to you
Q10	Do you work or have you ever worked at a White Label Startup?
#	**Level 3**
Q11	How long has the white label startup you work in exist?
Q12	In your view, what are the main advantages that a White Label Startup has?
Q13	In your view, what are the main challenges that a White Label Startup has?
Q14	What are the Software Engineering methods used by your Startup?
Q15	How many employees?
Q16	Describe your White Label product
Q17	How many products have you generated?
Q18	Which of these strategies do you use in generating a product?
Q19	How many days does it take to generate a customized product?

of White Label software products. It also asked the development strategies used according to the respondent's experience and if any particular technique was applied to generate new products. Also, the survey asked more particular questions in order to identify the startups' size.

4.3.3 Respondents Recruitment

The survey instrument was sent to professionals working in software startups in the All Saints Bay startup ecosystem, earlier introduced in Chapter 3. Since we carried out the semi-structured interview study in this ecosystem, we kept the survey respondents from the same context, including respondents from other ecosystems.

4.3.4 Pilot Survey

We conducted a pilot trial using the same artifacts and procedures designed for the survey, including the survey questionnaire and the execution method. We discussed the survey instrument among experts in survey studies. We conducted a pilot with a small number of participants from the target population and survey experts to obtain feedback on the

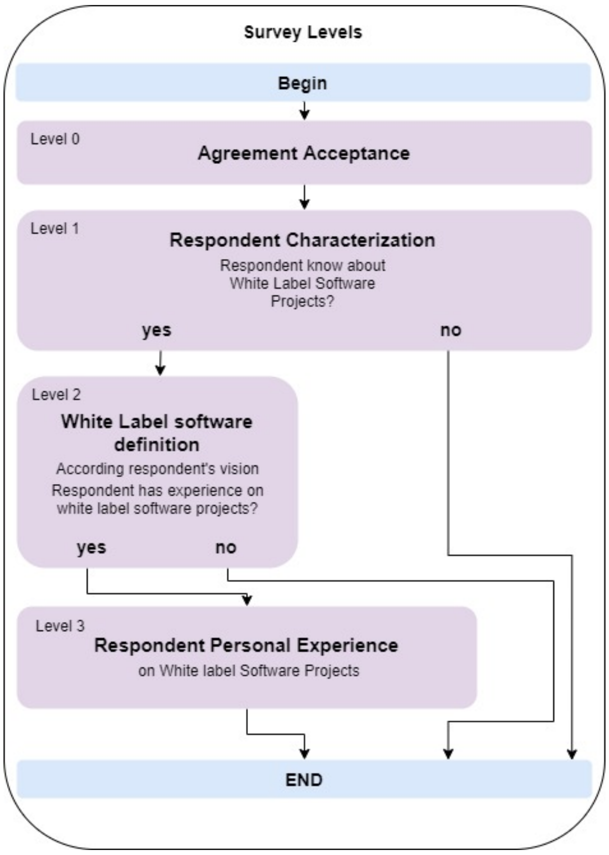

Survey Levels
Begin
Level 0
Agreement Acceptance
Level 1
Respondent Characterization
Respondent know about White Label Software Projects?
yes
no
Level 2
White Label software definition
According respondent's vision
Respondent has experience on white label software projects?
yes
no
Level 3
Respondent Personal Experience
on White label Software Projects
END

Figure 4.2 Survey Level

materials.

4.3.5 Data Collection

The invitation to the survey was sent to email lists and social media groups, where the invitation comprised the main instructions and the URL for accessing the online questionnaire. The survey received 22 responses and remained open for sixty days. After incomplete submissions removal, our sample result had 20 responses.

4.3.6 Data Analysis

From Google Forms, we exported the data to an electronic spreadsheet. The respondents' answers were all tabulated, which enabled us to identify similarities and differences in answers and perform qualitative data analysis.

4.4 RESULTS

This section describes how each research question was answered and adds evidence by respondents' quotations.

(RQ1) To what extent is the industry professional's perception of White Label software products'?

The data obtained in the survey showed that almost 55% of respondents knew or heard the definition of White Label and startup White Label software products. However, the **Q6** revealed that roughly 5% of them did not know how to explain its meaning, as Figure 4.3 shows.

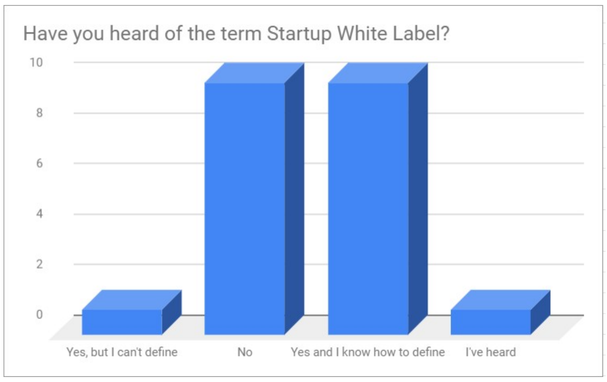


Figure 4.3 Answers from the question: Have you heard of the term Startup White Label?

From the interview, we obtained five definitions of White label Software Products, and those definitions placed as answers in the survey's **Q9**, so the respondents that were aware of the concept of White Label Software would select the best meaning of this concept. The most chosen answer was the one stated by SD: *"White Label Software is a product in which technology is licensed, customizing some features, making available for third parties to use and explore the product, selling it to an end-user."* This answer is very near to the proper concept of White Label Physical Products (KAISER, 2017).
We also observed that the answers from the interviews were complementary, and the answer chosen by more respondents in the survey was the full-reaching one. We believe that because the survey respondents had to choose only one option, their choice was toward the most meaningful answer.

RQ2) What are the main White Label software characteristics? Besides the inherited characteristics from White Label physical products, we observed four additional properties regarding White Label software products.

The first characteristic was the ease of scaling the product, meaning that it is only necessary to set up configuration files to either activate or deactivate features and load customization files for each client.
The customer is more likely to be involved with the product development process since several products will be generated using the same "core" set of features.
The White Label software products make it possible to expand the customer base without generating more effort or generating minimal effort for the development team.
Moreover, adopting a White Label approach facilitates and speed-up the software product customization process. It allows companies to have their software at a lower cost and development time, with the same benefits as they have been made from scratch.

RQ3) What are the software development strategies used to implement White Label software? When asked about software development strategies of respondents who work or worked at startups with White Label software, the survey results have pointed out that they use global configuration files, which aligns with the interviews' results. They also presented answers related to Agile Software Engineering, like Scrum and Sprints, as beneficial techniques when developing White Label software projects.

RQ4) What are the main challenges in the development of White Label software? Due to the innovative nature of White Label projects, survey's respondents presented challenges related to the business of commercializing this type of software:

- The need to demonstrate to clients that a White Label solution is the best for their scenario, convincing partners to have a White Label product instead of developing its product from scratch.

- Position the startup as a brand. Since the startup works producing White Label software for other companies, it is complicated for the startup to launch new products since existing partners can interpret it as competition.

- Develop a complete solution for the target audience.

Different products have different needs, how a startup can develop a White Label software that meets specifics user's requirements from partners.

4.5 DISCUSSION

In this section, on top of the discussion related to the four research questions above, we will describe observations not related to the main questions, which confirms initial thoughts and helps us understand the next steps for this work. The survey received answers from very heterogenic profiles, like developers, CEOs, investors, students, product owners, product managers, founders, mentors, and consultants. Despite that, the respondents have presented a very similar experience in years, as Figure 4.4 shows. These two characteristics give us the certainty to affirm that stakeholders inside ecosystems recognize White Label software projects.

For those who knew about White Label software projects, over 60 % of them did not work in this type of project, as Figure **??** shows. In fact, respondents first heard about this concept mainly from the internet, as Figure **??** shows. We can highlight that not only people with previous experience in White Label software projects know about this type of project.

For the respondents who have experience working in White Label software projects, the advantages highlighted are that these projects are easy to scale, very customizable, and enable market expansion in the startup through the client's commercial strength. The challenges are yet related to the benefits, signing the existing technical debt among the projects.

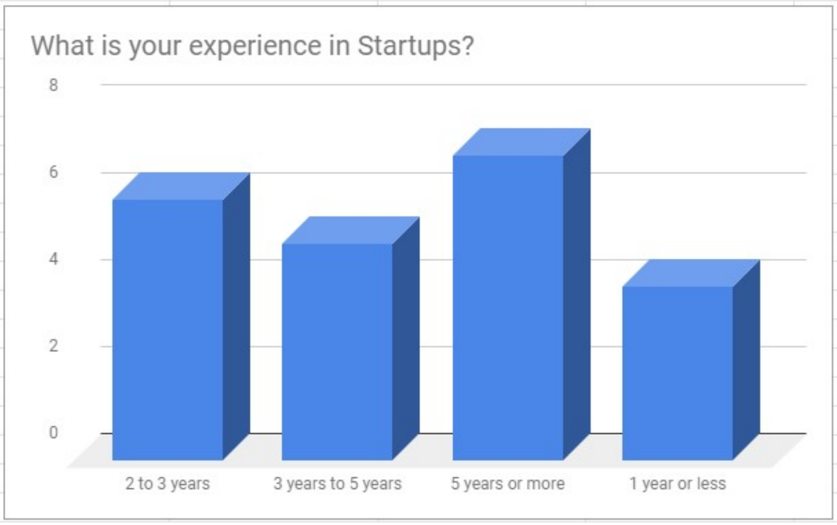


Figure 4.4 Answers for the question: What is your experience in Startups?

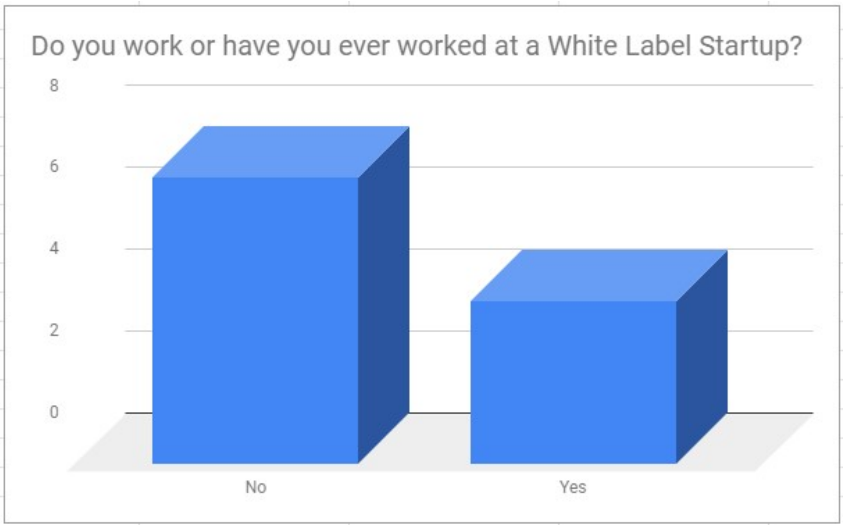


Figure 4.5 Answers for the question: Do you work or have you ever worked at a White Label Startup?

4.6 IMPLICATIONS FOR RESEARCH AND PRACTICE

The survey study has validated our initial thoughts about the similarities between White Label software projects and Highly-configurable Systems (HCS) and showed that the challenges present by respondents are connected to this approach's benefits. Respondents also have some knowledge from the internet about White Label software projects; this result has motivated this work to go through gray literature for the next empirical study.

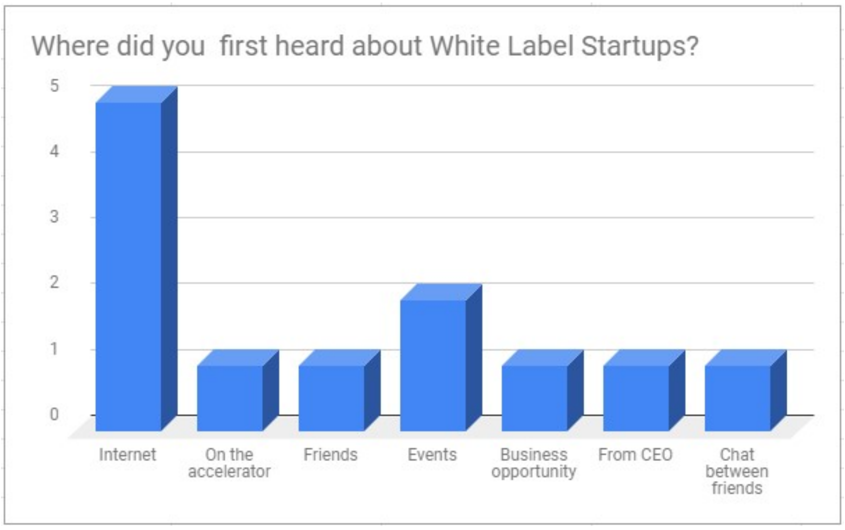


Figure 4.6 Answers for the question: Where did you first heard about White Label Startups?

4.7 THREATS TO VALIDITY

First, the coverage of the research questions could not have been well defined. The survey went through a pilot test with survey experts and the target audience. Another threat was the respondents' understanding of the questions. To help ensure the survey's understandability, we asked professionals and researchers to review it to ensure that the questions were clear and complete.

The respondents of the survey may not adequately represent all software practitioners. We believe that the 22 responses we analyzed provide a rich qualitative data source to reveal promising insights.

4.8 CHAPTER SUMMARY

This chapter presented a survey study applied to stakeholders inside innovation Ecosystems. The survey asked about their background, general understanding, and specific understanding of White Label software projects. The questions were unlocked or not by the previously given answer.

The results have given the direction for the next step in this work, agreed with the author's initial thoughts, and firm that startups applying White Label software projects need specific software engineering techniques to take the lead of their advantages.

Chapter

5

MULTIVOCAL LITERATURE REVIEW

By reviewing the surveys' results, where respondents indicated that the Internet was the primary source of knowledge about White Label software projects in startups, we decided to analyze the gray literature as an additional information source.

Articles that are not formally-published are called gray literature, and they encompass a large body of knowledge available. A Multivocal Literature Review (MLR) is used in emerging research topics to gather evidence from "multiples" voices, which are not related to academic bounds (GAROUSI et al., 2019).

Garousi, Felderer, and Mantyla (2019) presented guidelines for conducting MLR in Software Engineering, which consisted of three phases.: (1) planning the review, (2) conducting the review, and (3) reporting the review results. Figure 5.1 shows an overview of the Research Design.

This Chapter reports on the MLR on White Label Software project we carried out in the context of this investigation. Figure 5.1 shows the MLR design.

5.1 MOTIVATION

A MLR needs a solid motivation to make its conduction pertinent. As we already mentioned, we decided to carry out such a study after analyzing the survey study's results as a means to compare findings from both sources. We intend to gather from the Internet information about White Label software in startups and "listen" from the voices that could not be part of the first two empirical studies presented.

5.2 WHY TO CONDUCT A MLR?

MLR is an unconventional empirical study. We have chosen to perform a MLR because our previous research did not found an academic link between White Label startups[1] and Highly-configurable Systems (HCS) concepts. Considering that Web forums, blogs,

[1]White Label Startup is a short-term encountered for startups that work with White Label software projects, which is broader known in the technical literature

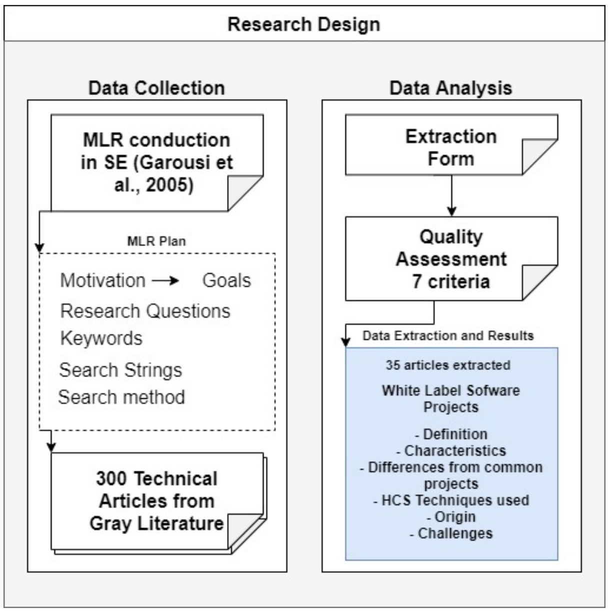


Figure 5.1 MLR Study Design.

events, and research companies are the primary sources of startup knowledge, performing an MLR could be considered a good strategy to leverage practical concepts about While Label and HCS combined.

5.3 MLR GOAL

The study's goal was to unveil the White Label startup concepts from what is said and discussed over the Internet, being an abundant resource of knowledge for searching the main topic. To achieve this goal, we proposed the following research questions derived from the main research questions of this study:

MLRQ1 - How could White Label Startups be defined?

MLRQ2 - What are the characteristics and benefits of this approach?

MLRQ3 - What are the differences between White Label Startups and regular Startups?

MLRQ4 - Which HCS techniques are used by White Label Startups?

MLRQ5 - Where does the White Label *name* comes from?

MLRQ6 - What are the challenges in White Label Software Projects?

5.4 SEARCH METHODS

To perform the MLR, we defined three search strings constructed from 2 keyword groups. Garousi, Felderer, and Mantyla (2019) indicate that the researcher should first identify the keywords related to the theme and then define the search strings. For the articles that presented references, we performed snowballing as well. We further present the keywords and research questions used in this study as follows:

Keywords:

- **Population:** “startup” OR “start-up” OR “early-stage firm” OR “early stage firm” OR “early-stage company” OR “early stage company”
- **Intervention:** “White label” OR “White-Label” OR “White Labelling” OR “White-Labelling”

Search Strings:

- What is “White Label” OR “white-label” AND “startup” OR “start-up” (What is White Label OR What is white-label OR What is White Label startup OR What is White Label start-up OR What is White-Label start-up OR What is White-Label startup)
- “Difference” AND “whitelabel startup” OR “white-label startup” OR “whitelabel start-up” OR “white-label start-up” AND “regular Startup” OR “regular start-up” OR “conventional startup” OR “conventional start-up”
- “startup” OR “start-up” OR “early-stage firm” OR “early stage firm” OR “early-stage company” OR “early stage company” AND “White label” OR “White-Label” OR “White Labelling” OR “White-Labelling”

5.5 DATA COLLECTION AND ANALYSIS

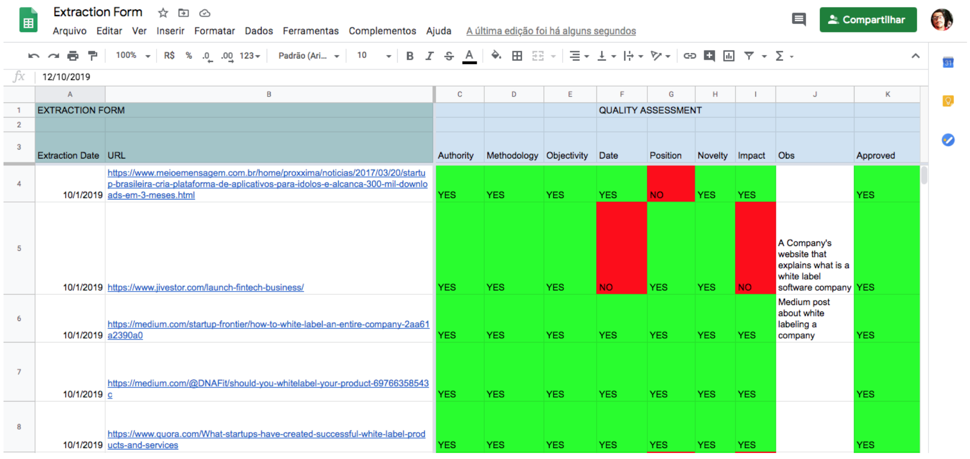
After defining the search strings, we used Bajwa et al. (2017) study as a basis for finding the best tool for data collection. We explain the performing steps as follows:

1. **Search method**: We used the Google search engine through the google chrome browser. We performed the search by accessing the www.google.com website, which guarantees not location-based results. Before starting the search, the researcher cleaned his browser data (websites historic, caches, and cookies) and logged out from his google account. The google predictions were disabled in the search configurations, and the option to show 100 results per page was enabled. We conducted the search in three days, October 01st, 12th, and 19th from 2019. That were 300 results, being 100 for each search string. If the results did not consistently point in one direction, we would gather the next 300 results.

2. **Exporting search results:** We reported the articles obtained to an excel spreadsheet with the help of a chrome extension, the SEOQuake[2]. The tool created three spreadsheets, one for each research question containing 100 links.

[2]www.seoquake.com

3. **Quality assessment, inclusion, and exclusion criteria**: Targeting the first 300 links, we made a new spreadsheet for applying the quality assessment, as Table 5.1 shows. If an article passed five out of seven criteria (we considered each criterion one point), we have elected it for later analysis. Figure 5.2 shows a sample of the Quality Assessment Form. The complete spreadsheet can is available at (SILVA; MACHADO, 2020).



Extraction Form
Arquivo Editar Ver Inserir Formatar Dados Ferramentas Complementos Ajuda
A última edição foi há alguns segundos
Compartilhar
100%
R$ % .0 .00 123
Padrão (Ari...
10

fx 12/10/2019

	A	B	C	D	E	F	G	H	I	J	K
1	EXTRACTION FORM					QUALITY ASSESSMENT					
2											
3	Extraction Date	URL	Authority	Methodology	Objectivity	Date	Position	Novelty	Impact	Obs	Approved
4	10/1/2019	https://www.meioemensagem.com.br/home/proxxima/noticias/2017/03/20/startup-brasileira-cria-plataforma-de-aplicativos-para-idolos-e-alcanca-300-mil-downloads-em-3-meses.html	YES	YES	YES	YES	NO	YES	YES		YES
5	10/1/2019	https://www.jivestor.com/launch-fintech-business/	YES	YES	YES	NO	YES	YES	NO	A Company's website that explains what is a white label software company	YES
6	10/1/2019	https://medium.com/startup-frontier/how-to-white-label-an-entire-company-2aa61a2390a0	YES	YES	YES	YES	YES	YES	YES	Medium post about white labeling a company	YES
7	10/1/2019	https://medium.com/@DNAFit/should-you-whitelabel-your-product-69766358543c	YES	YES	YES	YES	YES	YES	YES		YES
8	10/1/2019	https://www.quora.com/What-startups-have-created-successful-white-label-products-and-services	YES	YES	YES	YES	YES	YES	YES		YES

Figure 5.2 Quality Assessment Spreadsheet.

Before applying the quality assessment, we have defined a set of exclusion criteria for articles that were too personal, one-sided for the question, only presenting one view of the subject, extremely publicity, or more influential than informative.

4. **Quality Assessment Results:** From 300 items, our quality assessment found 275 unique links that had at least five out seven points approved. We marked two hundred forty resources as invalid, being 181 discarded after applying the exclusion criteria (news, pure marketing articles, articles related to other fields), and 59 rejected after applying the quality assessment seen in Table 5.1. Thirty-five articles were selected to be analyzed and have their data extracted.

5. **Extraction form:** In the same spreadsheet, we added a new group of columns, one for each research question. We read the 35 selected articles and extracted fragments of text that could answer the study questions, the extraction of each question for each article is also present in (SILVA; MACHADO, 2020).

5.6 DATA EXTRACTION AND RESULTS

In general, we did not need to analyze more articles due to evidence exhaustion, which means there is no more consolidated information to add. In Appendix B.1, is presented the list with the 35 articles where we extracted the results for this study. Below is a summary containing answers from the selected articles divided by the research question.

Table 5.1 Quality assessment criteria. Based on Garousi, Felderer and Mantyla (2019)

Criteria	Questions
Authority of the producer	Is the publishing organization reputable? Is an individual author associated with a reputable organization? Has the author published other work in the field? Does the author have expertise in the area? (e.g. Job title)
Methodology	Does the source have a clearly stated aim? Does the source have a stated methodology? Is the source supported by authoritative, contemporary references? Are any limits clearly stated? Does the work cover a specific question? Does the work refer to a particular population or case?
Objectivity	Does the work seem to be balanced in the presentation? Is the statement in the sources as objective as possible? Are the conclusions supported by the data?
Date	Does the item have a clearly stated date?
Position w.r.t. related sources	Have key-related gray literature or formal sources been linked to/discussed?
Novelty	Does it enrich or add something unique to the research? Does it strengthen or refute a current position?
Impact	Does the item has online metrics that proves quality? Number of citations, comments, likes, social media shares, etc.

At the end of each answer, we placed the article(s) reference, labeled as RSx, where X is the number of the URL available in Appendix B.1.

- **MLRQ1 -** How could White Label Startups be defined?
  - It develops customized apps for partner companies (**RS1**).
  - A company builds up a product, that can be used for their clients as their own, being fully customizable. This process should be invisible for the end clients (**RS2**)(**RS7**)(**RS8**)(**RS9**)**RS13**)(**RS14**)(**RS22**)(**RS28**)(**RS29**)(**RS34**).
  - Whenever a company uses an existing company's brand as a framework to develop its software (**RS7**).
  - A partner that develops technology personalized for new startups (**RS6**).
  - It is a business-to-business model involving manufacturers and resellers. Essentially, a software manufacturer sells an unbranded piece of software or service to a reseller who then places their branding on the offering and resells it to their business clients (**RS10**)(**RS11**).
- **MLRQ2 -** What are the characteristics and benefits of this approach?

- Building on-demand products could provide customers with particular (and needed) features. (**RS1**).
- Connecting with many companies, increasing networking, and opening more opportunities to the startup (**RS5**).
- From the outside, it looks like a third-party partnership, since the product comes from another company. The following list presents the benefits that a partner company seek to have when signing a contract with a White Label Software Startup: (**RS6**) (**RS16**) (**RS18**)(**RS23**)(**RS33**) (**RS34**)(**RS8**)(**RS14**)(**RS22**)
    1. Low development cost.
    2. No technical knowledge is needed since every development will be on the startup's side.
    3. No implementation risk.
    4. Quickly product build and market insertion.
- Characteristics to have in mind while creating a White Label startup (technical as strategy) (**RS29**).
- It allows the startup to obtain cash, recognition, and/or a strategic partner (**RS34**).

• **MLRQ3 -** What are the differences between White Label Startups and regular Startups?

- Other companies will operate the product (**RS33**).
- Agreement terms are very much different from those signed with a regular startup (**RS35**).

• **MLRQ4 -** Which HCS techniques are used by White Label Startups?

- The articles presented two types of building White Label software products: One of them being by creating a multi-tenant app, which means that the application instance will be the same, but each of the tenants will be getting an app with a bit different features set. Software-as-a-service (SaaS) is the most commonly used model in this context.

  The other way is by reusing the back-end code, but this requires changing the app's front-end to give a unique appearance to the app, so it will not be an exact copy of the same app. In comparison, the multi-tenant approach is quite complex to build and further maintain (**RS18**).

• **MLRQ5 -** Where does the White Label *name* comes from??

- The use of White Label software started in the music industry when record companies offered their musical bases for DJs. In the current digital industry, the idea is very similar, since the company that uses the finished platform is not the same as the one that developed it (**RS6**)(**RS7**)(**RS12**)(**RS32**).

- **MLRQ6 -** What are the challenges in White Label Software Projects?
  - Communication Problems between the client's feedback and advanced features, leaving for misunderstanding (**RS16**).
  - Customer needs in the second plan: The development team needs to know more about the business strategy, including its vision and marketing. In this way, startups can keep current clients' needs as a second plan (**RS16**);
  - Over-the-budget pricing and deadline prolongation. Miscommunication, unwanted features, and delays can lead the client to be unsatisfied and become a competitor (**RS16**).
  - White labeling may kill the future customer base: If the startup wants to serve as a White Label and create its labeled software, it may be killing its future customer base (**RS17**).
  - Issues with customers data: Since most projects are cloud-based, the White Label business clients let the White Label company knows everything about their clients (**RS18**)(**RS21**).
  - Code Quality: The White Label client does not have any control over code quality, which is necessary to sign any non-disclosure agreements between both parties, so the client will always have to trust in the White Label software product (**RS18**).
  - Possible rejection of White Label apps: Apple changed the app store guidelines, and now it prohibited apps that are pure clone (**RS18**).

5.7 MLR RESULTS COMPARED BETWEEN INTERVIEW AND SURVEY

The results provide a link between what was obtained in the previous empirical studies and add more key factors not presented before. We present the discussion for each research question below:

- **MLRQ1 -** Extractions for this research question agreed with the two studies presented in this work. The findings revealed that White Label Software could be either stand-alone products or services.

- **MLRQ2 -** Characteristics and benefits presented do not fall from what we are discussing in this work. The articles encourage the reader to trust White Label startups as a partner.

- **MLRQ3 -** The differences between a typical startup and a White Label startup are related to the type of contract made between partner and startup. The agreements must set data confidentiality between corporations, which opens a new question about how White Label startups' engineering team should handle user's data.

- **MLRQ4 -** Our search was not able to find HCS techniques used by White Label startups. However, the extraction from internet articles has bought an interesting addition to this work. When the article presents two types of White Label software products, it shows that SaaS can be considered a multi-tenant type of White Label software project. Linking SaaS with HCS has already been discussed in the academy (PATERNOSTER et al., 2014).

- **MLRQ5 -** For this answer, the results were different from past studies. While in the academy, a White Label product comes from physical products that follow the same philosophy used in software projects, technical literature shows a different origin for the term. All the articles pointed out the use in the music industry for helping DJs to produce music. Knowing that White Label also comes from reusable music bases shows how flexible this approach is.

- **MLRQ6 -** For challenges, this study has brought more business-related challenges than proper software ones. Besides, customer data is a topic that needs to be concerned when developing White Label software projects. One challenge presented in this study is related to the possibility of rejection when submitting to store apps. There is an app's category called "cloned," which are template-based apps that offer no difference besides small details in colors and logo. The app store usually bans them from the store. While this seems to be a block from the app store to the White Label apps, what happens is the opposite; apple is only concerned with the same app account handling identical apps[3].

With the amount of information gathered from the previous studies, we have been able to explain White Label software projects in deep and look forward to the next steps of this work.

5.8 THREATS TO VALIDITY

The **Construct validity** threat of this study refers to the search strings used. Since we restricted the search for "White Label", other articles that we did not gather may explain White Label without using this term. As a result, this work may have missed some key information about White Label Software in startups. However, "White Label" is a trendy and common term used in startup communities.

An **internal validity** threat is related to making inference on the data. A weak basis for data analysis could result from our search since we have no control over the data's quality and accuracy. We have mitigated this internal validity threat by reviewing a high number of articles reviewed and the proximity between what we found in this study with the relevant discoveries in the other studies applied for this work.

Another internal validity threat is related to the personal bias in the selected cases. To mitigate that, we defined quality assessment criteria with a review protocol applied to all articles. To ensure the study's replicability, we provided as many details as possible about how we conducted the search, and the review table is available at (SILVA; MACHADO, 2020).

[3]https://developer.apple.com/app-store/review/guidelines/

The last threat group covered is related to **external validity**. The 300 used articles may not represent all data available over the Internet. Besides, the extracted links were top-ranked by the Google search engine.

5.9 CHAPTER SUMMARY

This Chapter presented a MLR done to unveil White Label Software over the Internet. We extracted the results to a spreadsheet where the quality assessment was performed in order to select more trusting information.

From 275 out of 300 original Web links, 185 were discarded by applying the exclusion criteria, and 59 rejected after carrying out the quality assessment. Thirty-five articles were selected to be analyzed and had their data extracted.

The results confirm the other two research studies' findings and add new characteristics, challenges, particular care for customer data, and a new origin other than the previously mentioned in this work.

Chapter

6

ADAPTABLE FRAMEWORK FOR MOBILE WHITE LABEL SOFTWARE APPLICATION

Knowing that White Label software projects can benefit from Highly-configurable Systems (HCS) techniques, we started to ask ourselves how would be a real-world scenario would be, where Software Product Lines (SPL) concepts used to build White Label mobile projects. In this Chapter, we will be looking into a report for creating cross-platform highly-configurable mobile apps. This application was created by applying the knowledge acquired from the previous studies.

Software development practices have evolved to accommodate the ever-increasing needs posed by the market. Researchers have been proposing novel approaches as new features deployed if we look at some particular domains, such as mobile applications (mobile apps, for short) development. Nowadays, mobile phones are even more potent than ordinary personal computers and develop tons of new features so that the end-user could enjoy such devices' potential. Also, software development processes should be adapted to guarantee high quality at a low cost in a universe of various mobile devices.

We report an experience of developing highly configurable cross-platform mobile apps by employing the concepts of SPL engineering. SPL engineering enables building software products from reusable parts, and it provides a form of mass customization by constructing individual solutions based on a portfolio of reusable software components (APEL et al., 2016). Therefore, we seek to present all the challenges, benefits, and downsides a real-world company could face when attempting to implement the SPL engineering approach in practice for a poorly reported domain.

This work reports on a case study conducted at BRISA[1], a Brazilian innovation center that, since 1989, has been at the forefront of multiple innovative Information Technology (IT) projects, including software design, testing, and integration for major Telecom carriers in Brazil, a highly-competitive market, which poses several challenges to their

[1]Brisa's vice President authorized the presentation of this case study in this dissertation. However, due to business confidentiality, we will not provide any project's details. The list of features developed and project's specialties are at a high level of abstraction

partners. Since 2016, BRISA provides solutions for hotels and vacation homes. The Brisa USA branch is a startup-like company that has its fundings from the Brazilian headquarters. In 2017, the University of Central Florida Business accepted Brisa into the award-winning Incubation Program as a Soft Landing client[2]. Brisa's long-term objective is to grow in the USA by introducing themselves in new markets with innovative solutions.

In a particular project, a legacy message system needs to be available for mobile devices in the USA, providing a new way to communicate between the target user/company and its clients. A core system served three different websites. Each one of them had its specific characteristics. For example, their assets (logo, colors, and other images) differ from one version to another, and particular features were added, modified, or removed based on the running version. The distinct versions were maintained using three separate code repositories, and developers made improvements by employing a clone-and-own approach. The rationale behind choosing this project to perform the case study was because it has characteristics inherent to White Label software, and Brisa USA shares many commonalities with software startups.

The initial proposal was to find a way to translate the portals into three standalone mobile apps. After a brief analysis, we have discovered many downsides with this approach. In particular, every bug or implementation particularities would need to be fixed in six different locations (three mobile apps and three Web portals), making it hard to maintain and evolve the code. The project for the mobile app would be three as many times bigger for each new feature added. The Brisa USA has some characteristics present in both early and mid-stage Startups. They were unaware of advanced code reuse techniques, such as HCS. Even though their software engineering process was solid, the Brisa engineering team was not prepared for White Label software projects' specificities.

The team understood that they would spend more time if the same strategy employed to develop Web systems were also applied to create mobile apps. The team's engineering chief noted that this project was perceived to share key SPL characteristics. The products have a clear scope, which represents the software domain. Also, they share commonalities, which are the common software aspects that every product in a line should have, and vary in elements that make them unique (APEL et al., 2016).

Therefore, we attempted to create an SPL to handle mobile apps that should run on different platforms. The first challenge was implementing tool support, given that existing ones are not ready to handle mobile apps development yet. We then decided to implement a set of scripts that could enable the generation of products that could straightforwardly accommodate different product configurations. A primary document was created, being the base for explaining how to generate and configure a product and manage features.

Our solution enabled the configuration of both android and iOS projects, and engineering teams can apply to any project made with react-native for mobile apps. The scripts used are called shell scripts [3] that automate product derivation and generation.

[2]⟨https://incubator.ucf.edu/soft-landing/⟩
[3]https://www.shellscript.sh/

An advantage of these scripts is that they are compatible with any operational system.

After an initial configuration, the resulting scripts enable generating products in less than 10 minutes. The main output is an apk [4] file for android apps and a solution ready to be published for iOS apps. Apple puts some limitations on what can be automated using scripts or not. As a result, some processes are still manual after the script ran for the iOS version, a known limitation of our proposed approach.

In the following sections, we present the context in which we build our proposal, the problems faced, and the proposed solution. We will describe how our approach has been able to configure projects with files in five languages (javascript, XCodesproj, XML, Gradle, Java), presenting the technique in detail as well as its limitations and future work.

6.1 CONTEXT

There are three main approaches for introducing SPL engineering practices (APEL et al., 2016): *(i) proactive:* develops a product line from scratch, the concept and the code for the products are build in a planned way; *(ii) extractive:* starts by a collection of existing products that are refactored to be part of the SPL; *(iii) reactive:* starts with a small and easy to handle SPL, usually with only one product available, and it is extended incrementally with new products, features, and added variability.

We employed a *reactive approach*. Due to the demand was for a specific product, we would not have time to build the SPL using a *proactive approach*. Building a mobile app from scratch would give developers the necessary domain knowledge because the project's documentation was not available. Brisa's engineering team created the app, taking the legacy code from the Web-based version as a basis, and the product resulted was improved according to the users' feedback.

Three products were requested to be transformed into mobile apps, as shown in 6.1: PA, PB, and PC. Their main feature is to send messages for multiple contacts in a list, and each one does it in a particular manner by adding more specificity based on the target niche. PA was the first developed product. Brisa has requested this product to be the kickoff because of market needs. They developed PA from scratch without using reused components.

Next, PB was developed by Brisa. It targets a more general public and encompasses the default features, likewise in PA, and no particularities for a given product niche. PB was the first app made using the approach described in this study. The third product, PC, also targets a niche public. PC contains more specificities than his other brothers, as will be seen later in this Chapter.

The products respected a line of development in the company. Table 6.1 show that PC is queued for production but is estimated to be ready in less than a month, sooner than his predecessor.

The source code was developed by following the concepts of component-based and feature-oriented software development, as discussed by Prehofer (PREHOFER, 1997), which means to build reusable and standalone components across the project. Once the

[4].apk files are executable in any Android environment. It represents the running app

Table 6.1 Products available in the SPL

Product	Public	Creation Order	Time for Production
PA	niche	First	3 months
PB	general	Second	1 month
PC	niche	Third	¡1 month(queued)

Public: Is the target public in which the app was created for

first product (PA) was developed, tested, and considered stable, the development team started the variability implementation and management to build the next product, PB. Bosch (2004) explains that a high degree of variability allows the use of software in a broad range of contexts.

In our project, variability management was enabled by employing preprocessor directives in the source code, allowing it to configure itself while compiling (HUNSEN et al., 2016). This compile-time based method matches with the technology used for developing mobile apps in this scenario.

React-native [5] was chosen as the central technology to have the SPL constructed around. It is a hybrid mobile development framework that enables usage of the same code for building apps to android and iOS platforms. It saves implementation time by writing a single code type (javascript) that any platform can interpret. Although engineers write the code in javascript, native configuration and coding (Swift programming languages for iOS and Kotlin for Android) are necessary to build the mobile app.

Deciding to use this specific mobile framework has brought some issues when implementing the SPL. There was no tooling available in the market for variability management in a mobile platform that worked for iOS and Android apps. The existing tools are available for web and desktop apps only (BASTOS et al., 2017). Although reports for SPL use in industrial-scale is well studied (BENAVIDES; GALINDO, 2014; CLEMENTS et al., 2005; MACHADO et al., 2014; HANSSEN; FÆGRI, 2008; SOZEN; MERLO, 2012), few work has been done for applying techniques for mobile apps in real-world scenario.

The mobile apps had to achieve two main goals, create a better experience than the one obtained on the web and improve features that include media information through the use of native components available on mobile apps. These goals were the motivation for creating the first app from scratch, thus spending more time and not taking advantage of the old code written for the web.

6.2 THE APPROACH

This section introduces the proposed approach and discusses the decision-making process involved in each part of the solution.

[5]https://reactnative.dev/docs/getting-started

6.2.1 Brainstorming the solution

Having the PA made from scratch allowed the team to have a faster analysis of the differences from the other two products that would be part of this SPL. At first, we made an initial feature model for mapping the features available at PA, as 6.1 shows.

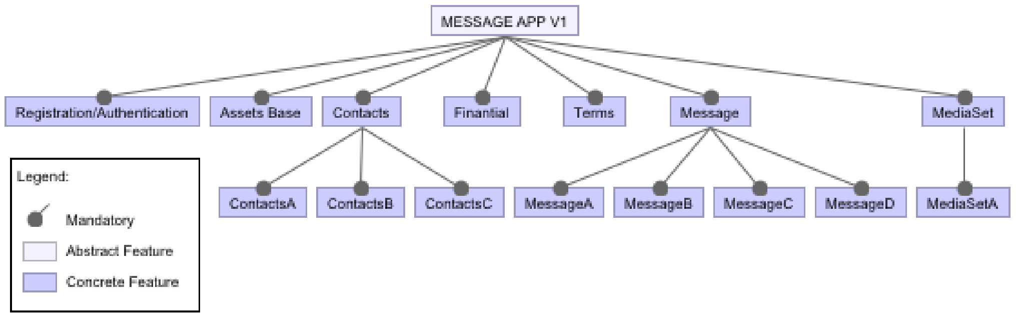


Figure 6.1 Project's feature model V1.

The feature model was built in the FeatureIDE tool (KASTNER et al., 2009). FeatureIDE allows the configuration of products according to a pre-defined set of features from a feature model. Because our SPL is both multiplatform and multilanguage, we used the featureIDE to construct the data tree view for high-level feature visualization, as Figure 6.2 shows. Due to the company's privacy policy, we omitted the actual feature names in this work.

With the first set of features visible, the team could analyze the other products and establish the differences. The team included the differences in a new version of the feature model. Figure 6.2 shows the updated model. In order to explain the differences, we divided into three variation groups called specific assets, platform-dependent changes, and available features, as listed next:

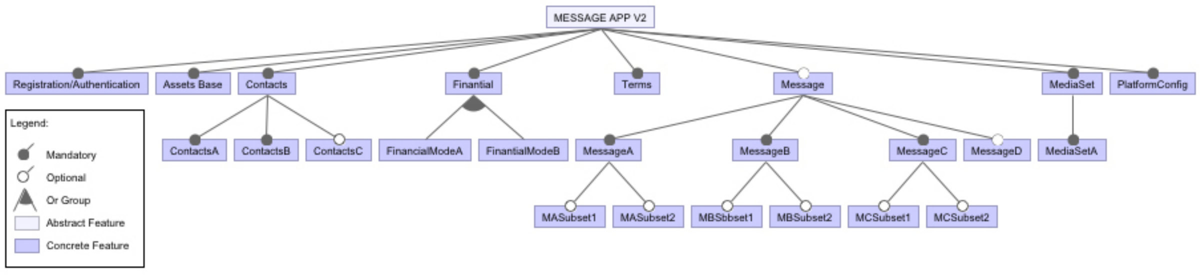


Figure 6.2 Project's feature model V2.

- **Specific assets:** Every product has a different set of colors, supported languages, logo, app images, Store Screenshots, Banners, and icons for iOS/Android. To generate a product, every needed asset from this group must be created following the same namespace and specifications. The lead designer has its patterns for fast creation of these assets;

- **Platform-dependent changes:** It is related to modifying Gradle properties, key security files, and scheme properties. Both Android and iOS apps have platform-specific configurations. This variation group is responsible for holding these configurations for every product available before building;

- **Available features:** These are the optional features derived in each product, as Figure 6.2 shows. They were either disabled (in a particular product) or enabled. For example, the feature *ContactsC* was mandatory in the first product. However, it becomes optional since it is not present in the other products. Table 6.2 shows a reduced view of possible product configurations, emphasizing the selection of optional features by each product (P).

Apel et al. (2016) presented a formula for obtaining the number of possible configurations for a SPL family. It is 2 to the power of N where N is the number of optional features.

$$2^n$$

Following the same formula, this White Label project can have 1024 possible configurations (2^{10}).

After being aware of what had to vary between products, we looked for tool support to implement the SPL project. As earlier mentioned, the available tools are platform-dependent and not built for developing cross-platform (or hybrid) apps. Support tools must be executable in any OS environment since programmers could develop in different platforms, such as Windows, Linux, or/and MacOS.

Table 6.2 Product configurations

#	**Optional Feature**	**Product PA**	**Product PB**	**Product PC**
1	ContactsC	✓	x	x
2	MASubsets	x	x	✓
3	MBSubsets	x	x	✓
4	MCSubsets	x	x	✓
5	Finantial Mode A	✓	✓	x
6	Finantial Mode B	x	x	✓
7	Specific Assets	✓	✓	✓
8	Platform changes	✓	✓	✓
9	Message (Create campaigns)	✓	✓	✓
10	MessageD	✓	✓	✓

Therefore, we implemented a set of modules of shell scripts files [6], to configure the products for both iOS and Android platforms, on both development and production sides. We further discuss the scripts in Section 6.2.2.2.

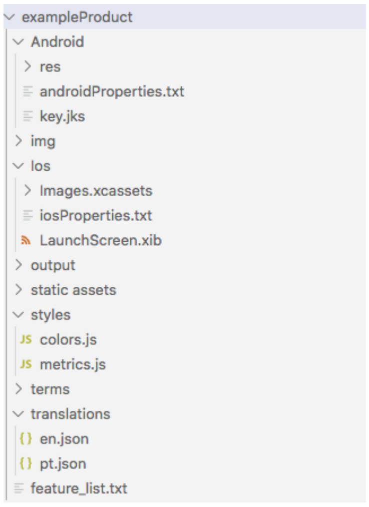
[6]a group of commands that serve to automate processes

6.2.2 Hierarchy Tree

The SPL project hierarchy tree is divided into two child trees, one that corresponds to every product's folder structure and another for the scripts module. We will explain this structure next.

6.2.2.1 Product structure

Figure 6.3 shows the hierarchy tree (structure of files and directories) for a sample product. The Android and iOS subfolders hold platform-specific configuration files. For example, the *androidProperties* is responsible for having android's app version, Gradle properties, and API connection addresses.



```
exampleProduct
  Android
    res
    androidProperties.txt
    key.jks
  img
  Ios
    Images.xcassets
    iosProperties.txt
    LaunchScreen.xib
  output
  static assets
  styles
    colors.js
    metrics.js
  terms
  translations
    en.json
    pt.json
  feature_list.txt
```

Figure 6.3 Product hierarchy tree

Inside the *img* folder, there are the general images used for the specific product. *Styles* hold the product's color palette and default metrics used in the application, which are base paddings, margins, font sizes, and border styles.

Terms and *translations* are both folders for holding specific text for the app. Inside *translations*, there are files available for each supported language. This project has English (EN) and Portuguese (PT-BR).

The *static assets* refers to files related to the product, not required for the building process. It is likely to include external documents, prototypes, and others.

The *output* folder is where the android file will be placed after the product's creation on the production side with the target version.

The *feature_list.txt* file shows what features are enabled and disabled for the target product. We use a flag to represent what is enabled or not and manually manage constraints.

6.2.2.2 Script Structure

We have created the scripts to have minimal changes through the project's development time. As Figure 6.4 shows, the scripts have been modularized according to each particular function. The main script is the *build_version.sh*, which is responsible for interpreting what the user prompted and run the initial configuration depending on the chosen platform, held on *configure_platformName.sh*. Another key variable when running the script is the build mode, set as either **development** or **production**. The development mode sets the app to run in a simulator or USB-connected device. The production mode sets the app ready to be published.

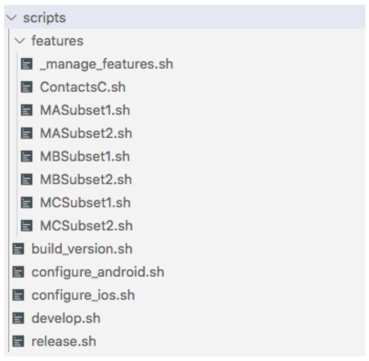

```
scripts
  features
    _manage_features.sh
    ContactsC.sh
    MASubset1.sh
    MASubset2.sh
    MBSubset1.sh
    MBSubset2.sh
    MCSubset1.sh
    MCSubset2.sh
  build_version.sh
  configure_android.sh
  configure_ios.sh
  develop.sh
  release.sh
```

Figure 6.4 Script Files hierarchy tree

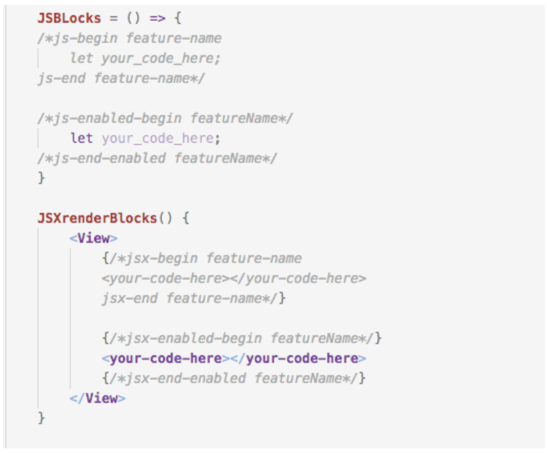
Releasing a product is made differently on each supported platform. For android, the script will place inside the output folder the *.apk* file. For the iOS platform, if the script runs on macOS, the final output opens the XCode app. It is where the iOS apps are submitted. Otherwise, the script would ask to be executed in an iOS-friendly environment.

The *features* folder holds the scripts for optional features. The *manage_features.sh* file attaches every feature available for either enabling or disabling a product. Besides, each file holds one optional feature, having controllers that comment and uncomment code bound to the feature.

The Javascript language does not handle preprocessor directives natively. As a result, we developed a pattern that worked like the C-native **#ifdef** preprocessor directive, but with comments in javascript. Santos et al. (SANTOS et al., 2016) created an approach for adding this annotation-based variability in javascript-based systems. React-native applications are more complex, a simple **#ifdef featureName** was not enough to generate different products.

In this approach, we developed two patterns of comments that would either enable or disable the code during compilation time. The first one is related to default javascript code, like the structure introduced in (SANTOS et al., 2016). The second one is called JSX and is an XML-like syntax used by react-native to enable tags in javascript files. The scripts would identify the type of code to be changed when configuring a product by looking for the signature, as seen in Figure 6.5 and toggling as requested. The comments are self-explainable, which might help developers to be guided by this annotation type when maintaining the code.

For code not written in Javascript, the scripts mapped the platform-specific information to be changed, in order to output a different product. We used the *sed* command,



```
JSBLocks = () => {
/*js-begin feature-name
    let your_code_here;
js-end feature-name*/

/*js-enabled-begin featureName*/
    let your_code_here;
/*js-end-enabled featureName*/
}

JSXrenderBlocks() {
    <View>
        {/*jsx-begin feature-name
        <your-code-here></your-code-here>
        jsx-end feature-name*/}

        {/*jsx-enabled-begin featureName*/}
        <your-code-here></your-code-here>
        {/*jsx-end-enabled featureName*/}
    </View>
}
```

Figure 6.5 Preprocessor Comments Examples

which seeks a text pattern and replaces the line found with the desired information. According to the content present in the platform-specific *.txt* files, the scripts search and change variables, shown in Figure 6.4. The changeable variables are mandatory on any mobile app. In this sense, the scripts for modifying the specific platform code are ready to be used in different projects, with no need for modification.

6.2.2.3 Documentation

This approach has generated a small number of documents. The main one explains the process of creating new products, building existing ones, and attaching features to the project. A 'common problems' section has been added and encouraged to be continuously updated by Brisa's engineering team with new information about issues and challenging times during the SPL development and evolution. The major problem found during the development was due to writing permissions. Since an automated script does the change inside the files, the person responsible for running the scripts needs to set permissions in some operating systems environments.

Another method of communicating with whom is building a product is through console when executing scripts. Figure 6.6 illustrates this communication for an iOS build for production mode.

The starter command has three mandatory variables, ***productName***, ***buildMode***, ***platform choice***. The final script command for our example would be ***./build_version.sh productA production ios'***.

The first information displayed on the screen is the confirmation of the product name, the folder location of the specific information, the build mode, and the target platform.

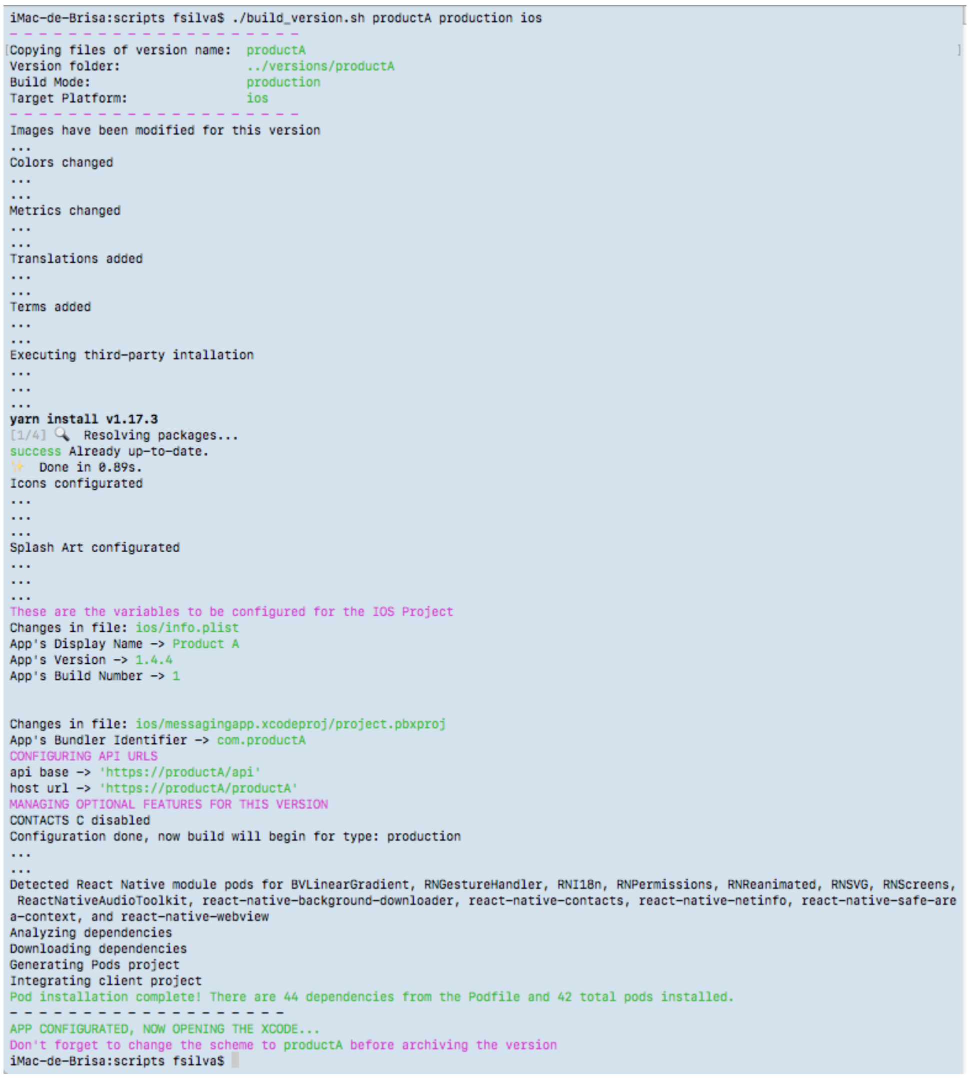
The code changes started with Images, Colors, General Metrics, Translations files, and specific Terms. The engineering team should create the static files under a pre-defined pattern for every customer.

Next, the script installs third-party libraries that can differ from one product to another. Then, it adds the static assets, icons, and splash arts according to the platform's choice.

Platform-specific changes are the following topic to be configured, as Figure 6.6 shows. The console logs the availability of optional features, and if any error occurs, the page where it happened is displayed. It is important to list everything that is changing because if any error occurs, the engineer will have a lead to begin his work through the execution logs.

The set of changes is related to the optional features, displayed in pink, where contactsC is disabled. Then, the script logs every optional feature that is either disabled or enabled in this step.

After the product's configuration finishes, the scripts will begin to build the product according to its platform and type. If the chosen platform was iOS and the user is on a macOS computer, the build system will open de XCode for finishing up the publishing process.



```
iMac-de-Brisa:scripts fsilva$ ./build_version.sh productA production ios
- - - - - - - - - - - - - - - - - - - - 
Copying files of version name:  productA
Version folder:                 ../versions/productA
Build Mode:                     production
Target Platform:                ios
- - - - - - - - - - - - - - - - - - - - 
Images have been modified for this version
...
Colors changed
...
...
Metrics changed
...
...
Translations added
...
...
Terms added
...
...
Executing third-party intallation
...
...
...
yarn install v1.17.3
[1/4] 🔍  Resolving packages...
success Already up-to-date.
✨  Done in 0.89s.
Icons configurated
...
...
...
Splash Art configurated
...
...
...
These are the variables to be configured for the IOS Project
Changes in file: ios/info.plist
App's Display Name -> Product A
App's Version -> 1.4.4
App's Build Number -> 1


Changes in file: ios/messagingapp.xcodeproj/project.pbxproj
App's Bundler Identifier -> com.productA
CONFIGURING API URLS
api base -> 'https://productA/api'
host url -> 'https://productA/productA'
MANAGING OPTIONAL FEATURES FOR THIS VERSION
CONTACTS C disabled
Configuration done, now build will begin for type: production
...
...
Detected React Native module pods for BVLinearGradient, RNGestureHandler, RNI18n, RNPermissions, RNReanimated, RNSVG, RNScreens,
 ReactNativeAudioToolkit, react-native-background-downloader, react-native-contacts, react-native-netinfo, react-native-safe-are
a-context, and react-native-webview
Analyzing dependencies
Downloading dependencies
Generating Pods project
Integrating client project
Pod installation complete! There are 44 dependencies from the Podfile and 42 total pods installed.
- - - - - - - - - - - - - - - - - - - - 
APP CONFIGURATED, NOW OPENING THE XCODE...
Don't forget to change the scheme to productA before archiving the version
iMac-de-Brisa:scripts fsilva$
```

Figure 6.6 An example output for a product build run

With the help of meaningful script logs, we can explain the product generation's process without maintaining extensive documents. This approach needs to be intelligent to help the developers quickly find errors and not make mistakes while building or developing.

6.3 DISCUSSION

We tested different methods to handle variability in this study, but they are all at the compile-time level. First, we attempted to allow code from entire pages to be changed while building a product, which was accomplished by storing the software code in its three. As a result, while maintaining the project, a developer would still have to change the code in three different places, one for each product, the code was going to be tripled.

Another attempt encompassed creating a new software folder from a default base. In this approach, the products would have only the desired features, and the app's size would be smaller. We did not follow this approach because it had development issues. A less skilled developer would face problems developing and maintaining the code, the scripts would have been a lot bigger, and every change would request the developer to recompile the product to see the update. From this point, we started developing the pattern shown in Figure 6.5.

It took about three months to develop and test PA. The team expected at least two months to develop PB as a separate product, with code duplication. Instead, the team finished converting the code to support variability and build PB before the initial deadline, as Table 6.1 shows.

As a result, we observed several gains in terms of time spent. It took about four weeks to release the SPL, add the new functionalities, and build PB. A reduction of three times developing PB compared to the PA development, and two times less confronted with PB's original time if built from scratch on its standalone version. Since a mobile app is constantly evolving, we noted that Brisa's development team saved much time by applying improvements that benefit both PA and PB.

Next, the quality team had saved time as well since they were oriented to use the same test cases of PA for PB, excluding the ones based on the disabled features. Letting the quality team know our approach allowed them to find issues not found in the previous testing. A code that is multiple times tested is more reliable.

A surprising benefit of this approach was that it enables a feature that is not natively available on Android projects. When the script stores the generated version in the product's *output* folder, it creates a version control outside the play store. The product's output includes its name, version, and build number, following a specific pattern defined in the scripts.

According to five of six quality criteria proposed by (APEL et al., 2016), this approach achieved the benefits of an SPL implementation on mobile:

- A reduced preplanning effort is achieved by reading the required documentation and understanding the graphical data, code implemented, and scripts used;
- For each optional feature, the created script should trace the code changes that disables/enables the target feature. Figure 6.5 shows the meaningful comments

pattern example that delimits the code for the optional feature

- We achieve the separation of concerns by the comments that are around the code for an optional feature;
- The granularity used is either medium-granted or fine-grained, not impacting other features;
- We also achieve uniformity in this framework. The modularized scripts were divided by code type that they change, meaning that variations in the product configuration scripts were separated from changes in the platform and build type, causing uniformity.

We could not guarantee the *information hiding* quality criteria by using this approach since more guided development is necessary. We would assume that using conventional code techniques for hiding essential secrets in apps would be enough.

This approach inherits the facility of creating new products in a SPL without impacting the code. The sales team can take advantage of an easy configuration process when selling new apps for specific customers. The development team can meet their expectations in a much faster way.

When speaking about challenges, the most difficult was involved in finding the right comments pattern that would be explicit to the developers that should maintain the code in the future but not too descriptive. Maturing the scripts was also a challenging task. They started as one file and were retouched by the engineering team until the scripts had six mandatory files plus X, being X the number of optional features available in the project.

The Product Manager noticed one downside of applying the proposed approach. The reported drawbacks refer to what he came across after building the products. First, the bundled apps' size was almost identical, no matter the product generated, even though PB has fewer features than PA. The ideal environment includes extracting the features not used by the target product, not disabling it by using comments.

A skilled engineer is required when modifying the generation scripts because it presents a hazardous activity. It represents another drawback of this approach, how fragile the scripts are for supporting modifications without impacting the process. The script's modularization comes helping to mitigate this drawback.

Besides these drawbacks, we can replicate this approach at any react-native project by following the specifications provided in section 5.1, allowing White Label software projects for mobile hybrid apps to be part of an SPL with the same benefits mentioned above.

6.4 A GUIDE FOR APPLYING THE CREATED FRAMEWORK TO WHITE LABEL SOFTWARE PROJECTS IN STARTUPS

This section will show how to apply the framework created and tested for either new or existing White Label software projects in startups. There are two main steps for applying it, mapping variation and enabling it systematically, adapting to the current startup's stage. We show these steps next.

6.4.1 Mapping Variation

First, the startup should map the variable points generating a Feature Model to show the current variation options. We have used FeatureIDE to generate the case study model presented, but any software diagram maker can be used to generate the document.

Startups should create the model showing all interactions between features if required, optional, whether one feature can block the other, and whether one must be selected to have the other. White Label software early products should not have all this kind of complex interactions, but instead, standalone components that can be isolated if not present in a product version. Figure 6.2 shows what the startup should expect after producing a simple Feature Model. Because the Feature Model is in a higher level of abstraction, software engineers can use it to make the project's stakeholders conscious of the software capabilities.

6.4.2 Enabling Variability

The framework created in this work uses a build system to generate new products. A startup may or may not have time to build a tool like this. Early-stage startups should keep the comments pattern in the code and manually swipe if a feature is active or not, since for them, what matters most is how to fast deliver software. Keep the comments under a pattern will mitigate the technical debt of not building a system for generating products.

Startups in the stabilization and growth stages should build a reliable tool to generate new products. In this framework, our build system targets mobile apps built with react-native. The core scripts explained in this Chapter are available at (SILVA, 2020).

Engineers should separate every product specificity from the whole software code. Colors used, special text, logo and images, the product's specificities should be can placed in distinct folders.

6.5 CONCLUSION

This Chapter reports on the real-world experience of implementing cross-platform mobile apps using SPL engineering concepts in White Label projects.

The study discussed the challenges, benefits, and drawbacks of employing a novel approach to support the development of SPL systems in the mobile apps domain.

SPL engineering can be suitable for developing mobile apps, particularly when tool support is available. In our experience, we employed an adaptation in an existing framework to support the development of both iOS and android platforms. This framework is available to be used by any project built with reactive-native. The strongness of our approach are listed as follows:

- This framework aims to work following the concept of hybrid mobile development, more precisely to react-native projects. Other frameworks can also apply the same strategy, like Flutter, Cordova Phonegap, and Ionic.

- Build a framework for SPL using native mobile programming languages is future work. This work's platform configuration represents the first step to build SPL under native apps.
- Running a single command that outputs the executable file encapsulates the complexity of building an app. Our approach makes it viable for engineers to make code changes and build up the project at any seniority level.
- For android, this approach enables a version history for the generated apps, which is not enabled by default locally in this platform. iOS apps already have this advantage.
- Building time is fast for new versions when compared to the build of standalone products in a non-automated environment.
- The code maturity provided by this approach. The built products derive from the same code base, meaning that they were tested multiple times in diverse circumstances.

We have found these strengths by solving the challenges presented when developing this framework. Below are some challenging points observed during the development:

- Establishing a culture of understanding inside the company. Since SPL is a new software engineering approach, many stakeholders would have to understand its key points and build trust on the platform before presented enough real evidence.
- Choosing the best supporting architecture for creating the scripts. The team has modified the scripts have uncountable times because the generation process would always need more retouches, and some changes were not possible initially through scripts.
- Choosing the best comments pattern that would be descriptive enough, but very modular

Some challenges are still opened, as well as room for future work, based on this study's materials. Down are the opened opportunities seen after this study:

- An opened challenge is related to improving the scripts' error handling. Some errors are not related to the code but due to the framework's specifies. The scripts need to be more intelligent in order to be ready for unexpected exists and provide a user-friendly message.
- The replication of this approach from researchers in industrial or academic environments. Enabling further development of this model by applying it in more advanced families from different domains.

- Future work should also encompass the creation of more conclusive documentation to unify the documents generated for each product in a solution, in one archive that can serve the whole family.
- Since there are other mobile-hybrid development frameworks like react-native, future work would make this approach adaptable.

6.6 CHAPTER SUMMARY

This Chapter presented a framework for developing White Label software projects in startups. We developed this framework following HCS methods and implements preprocessors in a flexible and adaptive build system.

We have validated this framework in a real-world mobile application. The SPL created has three products derived and over 18 variation points.

Startups can apply this framework to their mobile White Label projects, no matter the level of software maturity or startup's stage.

Chapter 7

WHITE LABEL SOFTWARE PROJECTS

This Chapter presents and discusses this study's main findings by analyzing data from the three empirical studies we carried out, synthesizing the results. Startups use this approach to create multiple products from the same code base, and these products differ in terms of features available and static assets, such as colors, images, and others. We also discuss how challenges can be solved and problems to be mitigated with the help of Highly-configurable Systems (HCS) projects.

In the interview, we obtained a definition for White Label software. Then the survey confirmed the first findings, and the ac MLR added more fundamental points to the research. Hence, we synthesize the definition for White Label software as follows:

"White Label Software is a product in which technology is licensed, customizing some features, making available for third parties to use and explore the product, selling it to an end-user"

Startups working with White Label software projects bring a concept analogous to the Software Product Lines (SPL), they can quickly generate a customized product for the partner, from a predetermined basis. The final product contains all the partner's inherent characteristics (visual identity, additional requirements) and is distributed to the end-user. Therefore, this practice is very similar to build *White Label* physical products (GEYSKENS et al., 2018).

White label projects could apply the processes that involve HCS in their daily activities to fade current ad-hoc processes used.

7.1 WHITE LABEL SOFTWARE CHARACTERISTICS

White Label software projects have inherited many characteristics from HCS. This section will highlight the additional characteristics observed in the empirical studies applied, which are related to the entrepreneurship context where startups are inserted.

- *A family of software from the same code base in the same context.* White label software projects can generate many different applications on different platforms (web, mobile, or desktop) within the same context.

- *Business Flexibility.* White label software projects enable flexibility also at the business level. Every potential partner has a different need, and startups can benefit from the software flexibility to enable different partnerships. For example, in the startup SB, they have a different contract of partnership for every product. While a product uses a custom API integrated with the partner for user management, other partners may need the product to be stand-alone.

- *Customer-driven development.* White label software is generated to meet customer needs, and it is vital for the project's success since startups make extensive and valuable contracts with partners. Startups developing White Label software projects need to focus their software evolution to be following the end customer's needs.

- *Treat customer data carefully.* Startups are very careful in their relationships with partner companies. Data gathered from one partner must not be accessible by other partners, although competitors may use the same White Label base. The information must be both transparent to the partner who owns the product and confidential to everyone else.

- *Opportunities for information exchange.* The primary source of knowledge for White Label software could be given in the following order: search for material and concepts over the Internet, attending related events as *meetups, hackathons*, and previous team experience. We observed that software engineers usually base their implementations on their personal experience without following any formal documentation. It implies that startups share common challenges.

7.2 WHITE LABEL SOFTWARE CREATION AND PROJECT SPECIFICITY

White Label software projects have differences in their life cycle. The studies first gathered how the projects have been born and how they plan to evolve. Next, the build of new products and how they plan to increase their performance, not so ever, testing this kind of project demands extra attention. Discussing how White Label software projects implement variability allows the engineering team to define the product's scope. Next, we discuss more topics about software engineering for White Label projects:

- *Software project.* There have been two patterns of creation for White Label Projects in startups. Ou analysis observed the first one in SC, SA, and SD. Their projects counted on either external consulting or previous experience to establish the features incorporated into the product. For the project SB, we noted the second pattern of White Label creation, where projects did not bear to be a White Label. They were transformed into a White Label by an emerging need, and its features were modified to accommodate variations, as requested by their consumers. As opposed to what we had initially thought, the number of variable features does not increase over time, only until the product has achieved a status of stabilization. In SB, they offer an unlimited number of new features for a partner, as long as it pays for the development.

- *Software build.* Companies range from 24 business hours in SA to 4 business hours on average in SD to generate a new product version. According to a survey respondent, it can go up to 5 working days, which is directly related to the automation level presented in the process, project complexity, and team expertise. For startups with more time in the market, they spend less time when generating new products.

- *Software testing.* During the interview, a question was asked about which software testing techniques startups used and if they have any particular approach for White Label software. They either say that no coordinated tests are done, only exploratory tests after creating the product. Result's analysis identified that testing White Label software is a new area of study which has not been fully explored yet.

- *Handling variability.* It was not possible to gather any feedback about the usage of HCS as a software reuse practice among the studied startups, besides the bond that this work has found between White Label projects and HCS. The observed lack of awareness about these concepts may explain why they struggle when getting into advanced software reuse methods, affecting the startup's capacity to hold many customers, evolve its products, and improve the build mechanism used. However, startups with more time in the market have improved their processes through creativity. One of the acceptable practices found is how they make use of the variation pattern. A variation pattern is how the product would either enable or not a feature. The interview has presented several ways of applying the variation pattern on different binding times, i.e., the moment in the project development (or execution), where the variable feature is bound. White Label Software uses methods in at least two of the binding times presented by Apel et al. (2016).

  For example, they could use configuration files to define global variables to make variability happen under compile-time and use control panels on the web to make changes at run-time. These are the most used variation patterns. Although the difference of generation time was considered significant between the projects known, we could infer that every startup is evolving their process in an ad-hoc way, without knowing about HCS or sharing knowledge between other White Label projects. It was interesting to perceive that even though startups present these characteristics, they are not far from what is suggested by the academy (APEL et al., 2016), but having the right consulting could improve their process of generating and evolving a White Label product.

White Label software projects correlate with HCS concepts in many ways. Because startups are unaware of systematic ways to work with variable systems, their ad-hoc practices will not fit in a long-term scenario.

7.3 WHITE LABEL SOFTWARE CHALLENGES AND FUTURE PATH

In the first part of this study, we have evidenced that White Label software projects are analogous to HCS projects by gathering information from technical articles and "listening" to practitioners' "voices". Then, we built a White Label software framework by applying the discoveries from the empirical studies done and concepts from SPL.

The project developed presents more evidence that the two terms are analogous. As a result, we cannot separate from the project what is White Label software is and what SPL is. This work assumes that White Label software projects are a particular type of HCS for entrepreneur context with SPL characteristics.

When looking at the development challenges announced from the empirical studies done in this work, the academy could have answered already for HCS, meaning that White Label software Projects can take advantage of HCS techniques. Startups in this context can solve their challenges faced when they become aware of systematic software reuse techniques. For example, tracking all the changes is an open challenge obtained in both interviews and surveys. However, Svahnberg et al. (2005) shows techniques for tracking changes, in the same way, we found out that adding more variability to the code with new features was an open challenge but is covered in (APEL et al., 2016). Last, defining the product as *complete* regarding the general understanding of the project's domain limit is also explained in (APEL et al., 2016).

The studies also unveiled challenges directed to business issues in White Label software projects. Due to business complexity, an option to mitigate these challenges could be events, meetups, and special presentations hosted by Ecosystems of Innovation. This study does not cover startups' business aspects with White Label Projects, but as many other challenges are faced by startups, sharing information is always the best choice.

From the aforementioned analysis, White Label software projects should consider the HCS concepts and adjust them to the entrepreneur context, solve the challenges, and improve the processes seen in our first studies.

Mainly, an excellent White Label software implementation should be lightweight and built on top of harmonious concepts. Startups should not spend too much time establishing patterns for their White Label software, but instead, use a process that can let them take advantage of every strongness involved in building HCS apps. In Chapters 3, 4, and 5, we summarized the challenges faced by startups. As our work continues, we will be further discussing how HCS techniques can be adapted to White label software projects in the entrepreneur context by facing the challenges presented.

7.3.1 Facing White Label software challenges

7.3.1.1 Classic Variability Management for White Label Software Projects

Apel et al. (2016) cite two types of variability implementation and management, the classic and the advanced. The classic language variability implementation was the most used agent in the startups interviewed, in the survey's respondents, and in the technical literature gathered. Startups developing White Label software use parameters to apply variability, but the never-used product's features can negatively impact the code. What should be the best options for evolving variability implementation in White Label software projects?

Another classic approach for variability implementation is making use of design patterns. Design patterns are beneficial when a pre-planning time is available, and the engineers know its definitions and best application. Startups are not known for preplanning activities; they operate in an unorganized manner, facing a lack of guidance for adopting software engineering practices (BERG et al., 2018). This characteristic makes design patterns complicated for startups to follow when implementing variability.

Apel et al. (2016) considers frameworks and components as classic approaches for variability implementation. Frameworks are widely used in software startups of any kind. They are third-party software parts used in many projects under a creative commons license, and startups make use of them to speed up development. The data gathered could not find any frameworks targeting this type of project. On the other hand, components are the software parts created to be embedded in the software exclusively. Componentization is a common-sense approach for developing software in startups. The best advanced coding technique takes componentization as practice, from Object-Oriented Programming to Atomic Design. The advantages are related to the specific code parts that are easier to understand, better to maintain, and easy to track changes.

Startups use classic variability implementation available in the field of software engineering in White Label software projects. However, these projects do not have a specific framework to be used in this context because HCS for White Label projects are not in common sense by the Startup Community.

Advanced techniques and tools for White Label software projects can improve their variability management and code quality. The following sections will be overviewing the advanced methods used for variability adapted to this context and aim to solve the most relevant challenges that this study has brought.

7.3.1.2 Define project's domain limit The customer drives the product's evolution in White Label software projects. Still, a challenge presented from the interviews regards when the product manager has to interfere in how the product is evolving. How a manager has to delimit the scope of their solution?

A White Label product scope is the set of applications in a market segment, with the software artifacts reused. The project manager needs to keep the external boundaries clear so the project will evolve under the market segment's possibilities.

Surprisingly, keeping the SPL under a domain limit is an open challenge presented by Metzger and Pohl, which declares that if the scope of a SPL is defined too broadly, domain artifacts may become too generic, and the effort of realizing them may become too high (METZGER; POHL, 2014). Scope optimization is a challenge open for SPL and White Label projects.

7.3.1.3 Documentation Specific documentation has been a missing point for the startups interviewed and also forgotten by the survey respondents. The industrial application referred to in this work only documents the product's creation and derivation. How should variability in White Label software projects be handled?

Metzger and Pohl present two types of documentation from SPL Engineering, the integrated and the orthogonal, and they are presented as follows (METZGER; POHL, 2014):

- **Integrated Documentation:** It is a documentation for converging the information from a SPL in a dedicated or specialized model.
- **Orthogonal Documentation:** It separates the documentation from product line variability from the other software development artifacts, even though the variability of the product line is treated as a first-class product line artifact (METZGER; POHL, 2014).

The most used model for representing the integrated documentation is the feature model, introduced by Kang et al. (1990). It has today, over 40 different implementations are divided into three categories (METZGER; POHL, 2014). A feature model was essential for making the framework understandable to new practitioners in the real-world experiment. The number of varying feature models makes it exciting to understand and adapt to White Label software Startups. Still, at the same time, it is hard to analyze the effectiveness of the scenario.

Metzger and Pohl point out that feature models that only present a high-level document variability of a product line, like mandatory and common features, are considered orthogonal documentation as well (METZGER; POHL, 2014).

In this way, we can infer that the project designed and developed in Chapter six used orthogonal documentation. The feature model only represented the common and optional components, and more appropriate places held additional documentation. Metzger and Pohl agreed that orthogonal documentation is better for handling variability in SPL because it keeps the documentation simple to change and more comfortable to understand (METZGER; POHL, 2014).

Starting from the point that our implementation followed the orthogonal context, what would be the output if a startup previously interviewed experienced that amount of documentation for handling variability? Does teaching HCS concepts are enough to support this application?

Variability documentation in startups is unknown. A start point would be applying an orthogonal type and wait until it gets adapted to the startup's needs.

7.3.2 Tools for White Label software projects

The startups interviewed were affirmative about the importance of supporting tools in their work. Generally, they use Git for code versioning, Trello for software management, Slack for team communication, and many others.

7.3.2.1 Git tools White label software projects in startups can benefit from specific plugins inside tools already used by startups, like those we mentioned before. For example, the startup's developer responsible for Continuous Integration (CI) and Continuous Delivery (CD) can make Git aware of the software variability by mapping added features.

Apel et al. (2016) demonstrate how to promote variability using Git, the two approaches presented are called *a-branch-per-product* and *per-features-branch.* Before describing these concepts, an explanation for Git and its elements is necessary.

A Git source repository is divided into modules, and each module holds the software's developed state, called branches. Code is "pushed" or sent to branches through commits, which should describe its evolution. On top of branches and commits, developers can add tags for specific software evolution points, which they will use for declaring new software releases.

The first technique for managing variability using Git is called *A-branch-per-product*. It is based on having one branch as a base for the solution and N branches where N is the number of products available to generate. General improvements would be done in the base branch and merged with the others before producing new releases using tags. Specific product branches would hold specific features for the product and follow its development path. Figure 7.1 shows an example of this approach, where Product A, tag version 1.8, would be built from a specific branch after bug fixes.

Another technique similar to the former is called *Per-features-branch*. For making a new version, the engineering team would be by converging branches from each desired feature. Figure 7.1 shows two optional features that would be evolved on different branches and merged when creating a new release for product A.

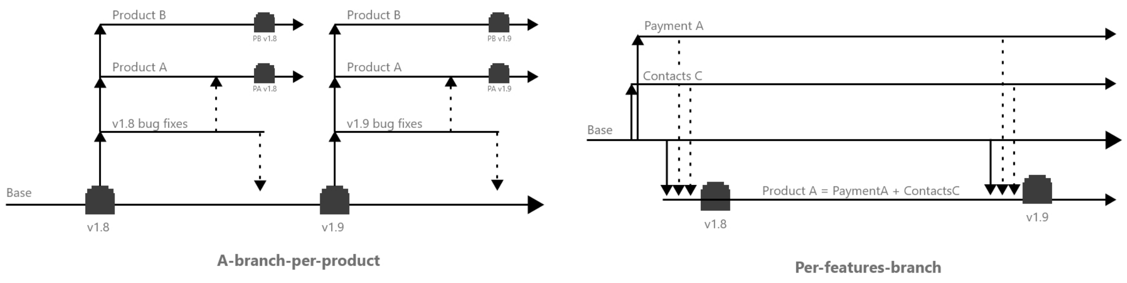


Figure 7.1 Git approaches adapted from (APEL et al., 2016)

While these strategies have been widely studied and their strongness and weaknesses presented in Apel et al. (2016), they can be improved for White Label software projects since Git usage is widespread on startups.

Additionally, T¨ernava, Mortara, and Collet presented a tool to identify and visualize variability in object-oriented variability-rich systems automatically. It helps developers to understand the variant points across the project (TËRNAVA et al., 2019). Visualization artifacts are essential to startups since they are easier to understand and quicker to review.

7.3.2.2 Build systems and preprocessors Build systems and preprocessors are not exclusive in SPL applications. A react framework called gatsby needs to use a build system to create static page websites. The java software uses the maven management tool to download dependencies, and the C preprocessor provides directives to C/C++

projects before compilation.

Apel et al. (2016) present build systems and preprocessors as classics variability mechanisms. While the former is responsible for scheduling and performing all build-related tasks, the latter is a tool that manipulates source code before compilation.

In the framework created for this work and specified in Chapter 6, we built a build system for mobile development with react-native for White label software projects. A preprocessor for a specific source code was used to enable high configuration capacity. As a result, White label software projects in startups can take benefit from a build system to enable and control variability in their products as they need.

7.3.2.3 Feature oriented programming and advanced techniques In the literature, FeatureIDE is a known tool for supporting SPL applications. It is a plugin for eclipse that helps modular programming, and it is oriented to the development of optional features attached to some software. More advanced techniques for variability implementation includes aspect-oriented programming and other tools.

While a perfect scenario would be made by managing White Label software projects in startups using a tool similar to FeatureIDE, startups usually use advanced and innovative technologies that do not have this type of supporting tool available. As a result, we affirm that it is best to have a lightweight, editable, and hybrid tool for configuring the many files across a White Label software for startups, like an evolution of the scripts made for the industrial application previously mentioned.

7.4 CHAPTER SUMMARY

This Chapter presented an explanation for White Label software projects based on the past three studies' results. It presents many characteristics similar to HCS but in the entrepreneurship context, which provides flexibility and new opportunities for the startup. This type of project has a customer-driven development approach, meaning that White Label software projects benefit from the aforementioned techniques for startups that enable continuous delivery.

White Label software projects can be born by emerging needs, pivoting an existing startup, or by planning since the early stage. The software can be build using many strategies like global variables or building systems with preprocessors, which leaves an enormous difference between product generation time and effort from one startup to the other, depending on the build strategy used. We did not observe any testing methods adapted to White Label software projects in any of the studies done.

Challenges opened for White Label software projects are present, basically due to the technical debt that startups have at this stage and because they are not aware of advanced code reuse techniques.

This Chapter has also presented directions for future study in White Label software projects and insights for startups to improve their ad-hoc processes. The knowledge from HCS can be used for early-stage startups that will face technical debt and mid-stage startups facing the challenges involved in these projects.

Chapter 8

CONCLUSION

This work investigated the nature of White Label software projects in startups. We conducted three empirical studies to understand the specifics of this type of software project in-depth. First, we conducted interviews with startups that work with White Label software projects. The results provided an in-depth knowledge of how startups define and handle this type of project, including its challenges. Next, we submitted a survey to stakeholders from the same ecosystem where the previous study was done. This study aimed to identify White Label software projects from the perspective of those who do not work directly with this type of project. Last, our third study focused on gray literature research, a Multivocal Literature Review (MLR), which helped us identify the most relevant articles and add new information to this work. Our results combined agreed that White Label is analogous to Software Product Lines (SPL), being a type of Highly-configurable Systems (HCS) implementation focused in the entrepreneur context.

The central points that supported the definition of White Label projects as HCS in an entrepreneur context are summarized below:

- The definition obtained in this work by the empirical studies applied to be very close to SPL definition: *"White Label Software is a product in which technology is licensed, customizing some features, making available for third parties to use and explore the product, selling it to an end-user"*
- The characteristics presented in White Label software projects are present in HCS projects as well, like enabling systematic variability over a determined context.
- Challenges faced by White Label startups are also faced by HCS projects, like managing documentation.

Using the expertise learned from the empirical studies and the practices from HCS, we made a framework for building White Label software projects. It takes advantage of preprocessors, serving as a build system for these projects built with react-native, a hybrid mobile development framework.

The created framework was tested in a real-world environment and used in a project with three products deriving from the same code base. Its structure is meant to be easily adaptable to the software based on its context, easily evolvable. Also, it benefits the startup to face technical debt by encapsulating qualities needed to any HCS. The project where we applied the framework is evolving without problems, meaning that the engineering team added new optional features, and the software variability is still well managed. The White Label software framework for mobile projects enabled variability in different forms while kept the code quality and delivered generated products on time.

React-native supports native applications from the iOS and Android operational systems. The framework created could be easily adapted to a hybrid or native development platform for mobile devices.

The framework's usage is through a single command from the terminal that executes a script file. This file encapsulates the complexity involved when building an app. As a result, it is viable for engineers to make code changes and build up the project no matter the seniority level.

Future work can be done by solving open improvements in the framework created. First, error outputs in the scripts should be improved and commented code removed after the product build. Moreover, the building system needs to be replicated in other White Label mobile software projects.

This work investigated the concept, characteristics, and challenges of White Label software projects, summarized a set of best practices for these projects based on HCS and built a framework for applying systematic reuse in White Label mobile projects. Considering HCS techniques and adapting to the entrepreneur context can potentially solve challenges in early-stage startups or mid-stage startups that develop White Label software projects.

8.1 RELATED WORK

We looked for studies in the literature addressing Software Engineering principles applied to innovation projects in startups. We believe such issues are directly related to the core of our investigation.

Much effort has been made to understand startups' processes and create answers to their struggles. Mapping studies serve to map the field of Software Engineering, and startups (PATERNOSTER et al., 2014; BERG et al., 2018). This dissertation's related work are projects built for startups that aim to impact possibly unknown areas or different perspectives.

The first work is the Hunter-gatherer cycle, which was proposed to assist startups in all organization phases, from ideas to commercial products. This framework assumes that startups have stages of order and disorder (NGUYEN-DUC et al., 2015). Nguyen-Duc et al. (2015) assist the decision-making in different startup scenarios. To know startups that develop White Label software projects, we had to dive into their processes and extract their challenges using three separate studies, targeting different personal.

Another Software Engineering model for Startups is called Stairway to Heaven, where it focuses according to the current startup stage by implementing different Engineering concepts. Olsson et al. (2012) had chosen four startups to perform a case study with semi-structured interviews. As a result, their 5 step stairway begins with startups using traditional development cycles to develop their first MVP. Next, they become more aware of agile methodologies until they reach the stage of continuous integration. By doing the semi-structured interviews, Olsson et al. (2012) found that the teams are often ahead of the organization as a whole. They adopt agile practices at an organizational level until they evolve into an institutionalized approach to software development. This study also shows how unorganized early-stage startups are and highlights that the development teams are often the nursery of innovation. HCS processes should start as well in the development team. Once the software is ready to perform under different configurations, the White Label Software project can evolve to be at an organizational level.

The Greenfield Startup Model (GSM) explains how development strategies and practices are engineered and used in startups. Giardino et al. (2016) created GSM after performing semi-structured interviews with CEOs and CTOs from 13 startups. The GSM synthesis agrees with what we found in the semi-structured interviews done in this work. As we obtained intentional and unintentional technical debt, startups may payback when achieving the stabilization stage. Early-stage startups do not implement standard development strategies and need to execute fast prototypes using lightweight methodologies adaptable to their specificities. Similarly, Souza, Malta, and Almeida (2017) build the Academic Startup Model by interviewing four software startups, and their findings correlate with GSM and the results of this work. Startups with White Label software projects were studied using a similar approach. This dissertation's findings are similar to the above studies and add more challenges to the target projects.

8.2 FUTURE WORK

This study has been able to go deep into White Label software startups. We have found enough evidence to enlighten the future path of working with HCS in Innovative Ecosystems by linking advanced code reuse techniques with White Label software startups.

Some business challenges for White Label software projects were found that are not covered in this work. We believe that startups could mitigate these challenges with events, meetups, and special presentations hosted by Ecosystems of Innovation. They are summarized below and are open for future study:

- How to convince partners to have a White Label product instead of developing its product from scratch.

- Position the startup as a brand. Since the startup works producing White Label software for other companies, it is complicated for the startup to launch new products since existing partners can interpret it as competition.

- How to find when the solution is complete to the target audience.

- Communication problems between partner companies and the Startup.

- How to find the best scenario to launch the product and present it to new clients since the loss of this time-to-market might imply clients' loss.

For software engineering, researchers can begin future work in White Label software projects from three different perspectives:

- Investigating other ecosystems for White Label software projects in startups, and understand their needs. Replicating the interview and survey study done in this work. The replication of these studies is needed to corroborate with the analysis that this work presents and contribute to the theme's universalization.
- Understand how White Label software projects elucidate their software requirements and how startups with this type of project can adapt the Minimum Viable Product (MVP) approach to their multiple configurations scenario, creating an MVPWL.
- Disseminate about how HCS could help startups with White Label software projects face its challenges.
- Apply the framework built in this work for enabling variability management in more White label Software projects and get data directly from the engineers on how startups' can improve their used processes.
- Considering how valuable the number of articles across the internet about White Label software and the information gathered by listening to the stakeholder's voices, we believe that this work should be shared across the same places where it helped to be made.

REFERENCES

ADAMS, S. B. Stanford and silicon valley: Lessons on becoming a high-tech region. *California management review*, SAGE Publications Sage CA: Los Angeles, CA, v. 48, n. 1, p. 29–51, 2005.

APA, C.; JERONIMO, H.; NASCIMENTO, L. M.; VALLESPIR, D.; TRAVASSOS, G. H. The perception and management of technical debt in software startups. In: *Fundamentals of Software Startups*. [S.l.]: Springer, 2020. p. 61–78.

APEL, S.; BATORY, D.; KÄSTNER, C.; SAAKE, G. *Feature-oriented software product lines*. https://link.springer.com/book/10.1007/978-3-642-37521-7: Springer, 2016.

AVGERIOU, P.; KRUCHTEN, P.; OZKAYA, I.; SEAMAN, C. Managing technical debt in software engineering (dagstuhl seminar 16162). In: SCHLOSS DAGSTUHL-LEIBNIZ-ZENTRUM FUER INFORMATIK. *Dagstuhl Reports*. https://drops.dagstuhl.de/opus/volltexte/2016/6693/, 2016. v. 6, n. 4.

BAJWA, S. S.; WANG, X.; DUC, A. N.; ABRAHAMSSON, P. “failures” to be celebrated: an analysis of major pivots of software startups. *Empirical Software Engineering*, Springer, v. 22, n. 5, p. 2373–2408, 2017.

BASTOS, J. F.; NETO, P. A. d. M. S.; O’LEARY, P.; ALMEIDA, E. S. de; MEIRA, S. R. de L. Software product lines adoption in small organizations. *Journal of Systems and Software*, Elsevier, v. 131, p. 112–128, 2017.

BATORY, D. Feature models, grammars, and propositional formulas. In: SPRINGER. *International Conference on Software Product Lines*. https://link.springer.com/chapter/10.1007/11554844$_3$, $2005.p.7--20$.

BECK, K.; BEEDLE, M.; BENNEKUM, A. V.; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R. et al. Manifesto for agile software development. 2001.

BENAVIDES, D.; GALINDO, J. A. Variability management in an unaware software product line company: an experience report. In: *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems*. https://dl.acm.org/doi/abs/10.1145/2556624.2556633: ACM, 2014. p. 1–6.

BERG, V.; BIRKELAND, J.; NGUYEN-DUC, A.; PAPPAS, I. O.; JACCHERI, L. Software startup engineering: A systematic mapping study. *Journal of Systems and Software*, Elsevier, v. 144, p. 255–274, 2018.

BOSCH, J. Software variability management. In: IEEE. *Proceedings. 26th International Conference on Software Engineering.* https://ieeexplore.ieee.org/abstract/document/1317504, 2004. p. 720–721.

CARMEL, E. Time-to-completion in software package startups. In: *1994 Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences.* https://www.infona.pl/resource/bwmeta1.element.ieee-art-000000323468: IEEE Comput. Soc. Press, 1994.

CLEMENTS, P. C.; JONES, L. G.; NORTHROP, L. M.; MCGREGOR, J. D. Project management in a software product line organization. *IEEE software*, IEEE, v. 22, n. 5, p. 54–62, 2005.

COHEN, M. B.; DWYER, M. B.; SHI, J. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Transactions on Software Engineering*, IEEE, v. 34, n. 5, p. 633–650, 2008.

CROWNE, M. Why software product startups fail and what to do about it. evolution of software product development in startup companies. In: IEEE. *IEEE International Engineering Management Conference.* https://ieeexplore.ieee.org/abstract/document/1038454, 2002. v. 1, p. 338–343.

DAHLE, Y.; NGUYEN-DUC, A.; STEINERT, M.; REUTHER, K. Six pillars of modern entrepreneurial theory and how to use them. In: *Fundamentals of Software Startups.* https://link.springer.com/chapter/10.1007/978-3-030-35983-6$_1$: *Springer*, 2020.*p*.3 − −25.

DECKER, R.; HALTIWANGER, J.; JARMIN, R.; MIRANDA, J. The role of entrepreneurship in us job creation and economic dynamism. *Journal of Economic Perspectives*, v. 28, n. 3, p. 3–24, 2014.

ETZKOWITZ, H.; MELLO, J. M. C. de; ALMEIDA, M. Towards "meta-innovation" in brazil: The evolution of the incubator and the emergence of a triple helix. *Research policy*, Elsevier, v. 34, n. 4, p. 411–424, 2005.

ETZKOWITZ, H.; ZHOU, C. Hélice tríplice: inovação e empreendedorismo universidade-indústria-governo. *Scielo Estudos Avançados*, scielo, v. 31, p. 23 – 48, 05 2017. ISSN 0103-4014. Disponível em: ⟨http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-40142017000200023&nrm=iso⟩.

FELDMAN, M. P. The entrepreneurial event revisited: firm formation in a regional context. *Industrial and corporate change*, Oxford University press, v. 10, n. 4, p. 861–891, 2001.

GAROUSI, V.; FELDERER, M.; MÄNTYLÄ, M. V. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology*, Elsevier, v. 106, p. 101–121, 2019.

GEYSKENS, I.; KELLER, K. O.; DEKIMPE, M. G.; JONG, K. de. How to brand your private labels. *Business Horizons*, Elsevier, v. 61, n. 3, p. 487–496, 2018.

GIARDINO, C.; PATERNOSTER, N.; UNTERKALMSTEINER, M.; GORSCHEK, T.; ABRAHAMSSON, P. Software development in startup companies: the greenfield startup model. *IEEE Transactions on Software Engineering*, IEEE, v. 42, n. 6, p. 585–604, 2016.

GIARDINO, C.; UNTERKALMSTEINER, M.; PATERNOSTER, N.; GORSCHEK, T.; ABRAHAMSSON, P. What do we know about software development in startups? *IEEE software*, IEEE, v. 31, n. 5, p. 28–32, 2014.

GURP, J. V.; BOSCH, J.; SVAHNBERG, M. On the notion of variability in software product lines. In: IEEE. *Proceedings Working IEEE/IFIP Conference on Software Architecture*. https://ieeexplore.ieee.org/abstract/document/948406, 2001. p. 45–54.

HANSSEN, G. K.; FÆGRI, T. E. Process fusion: An industrial case study on agile software product line engineering. *Journal of Systems and Software*, Elsevier, v. 81, n. 6, p. 843–854, 2008.

HOVE, S.; ANDA, B. Experiences from conducting semi-structured interviews in empirical software engineering research. In: . https://ieeexplore.ieee.org/abstract/document/1509301: IEEE, 2005. v. 2005, p. 10 pp.–. ISBN 0-7695-2371-4.

HUNSEN, C.; ZHANG, B.; SIEGMUND, J.; KÄSTNER, C.; LESSENICH, O.; BECKER, M.; APEL, S. Preprocessor-based variability in open-source and industrial software systems: An empirical study. *Empirical Software Engineering*, Springer, v. 21, n. 2, p. 449–482, 2016.

KAISER, S. *Difference between private label vs white label products*. 2017. Disponível em ⟨http://godirek.com/blog/2017/05/difference-private-label-vs-white-label-products/⟩. Acessado em Abril de 2019.

KANG, K. C.; COHEN, S. G.; HESS, J. A.; NOVAK, W. E.; PETERSON, A. S. *Feature-oriented domain analysis (FODA) feasibility study*. https://apps.dtic.mil/sti/citations/ADA235785, 1990.

KASTNER, C.; THUM, T.; SAAKE, G.; FEIGENSPAN, J.; LEICH, T.; WIELGORZ, F.; APEL, S. Featureide: A tool framework for feature-oriented software development. In: IEEE. *2009 IEEE 31st International Conference on Software Engineering*. https://ieeexplore.ieee.org/abstract/document/5070568, 2009. p. 611–614.

KITCHENHAM, B. A.; PFLEEGER, S. L. Personal opinion surveys. In: *Guide to advanced empirical software engineering*. https://link.springer.com/chapter/10.1007/978-1-84800-044-$5_3 : Springer, 2008. p. 63 - -92$.

LUNDGREN, A. Technological innovation and industrial evolution. *The Emergence of Industrial Networks, Stockholm School of Economics*, 1991.

MACHADO, I. do C.; SANTOS, A. R.; CAVALCANTI, Y.; TRZAN, E.; SOUZA, M.; ALMEIDA, E. Low-level variability support for web-based software product lines. In: *The Eighth International Workshop on Variability Modelling of Software-intensive System (VaMoS)*. https://dl.acm.org/doi/abs/10.1145/2556624.2556637: ACM, 2014.

MACLEOD, M. *Why I hate White Labeling for Startups*. 2012. Disponível em ⟨https://www.startupcfo.ca/2012/06/why-i-hate-white-labeling-for-startups/⟩. Acessado em Abril de 2019.

MARMER, M.; HERRMANN, B. L.; DOGRULTAN, E.; BERMAN, R.; EESLEY, C.; BLANK, S. Startup genome report extra: Premature scaling. *Startup Genome*, v. 10, p. 1–56, 2011.

METZGER, A.; POHL, K. Software product line engineering and variability management: achievements and challenges. In: *Future of Software Engineering Proceedings*. https://dl.acm.org/doi/abs/10.1145/2593882.2593888: ACM, 2014. p. 70–84.

MOORE, J. F. *The death of competition: leadership and strategy in the age of business ecosystems*. https://www.amazon.com.br/Death-Competition-Leadership-Strategy-Ecosystems/dp/0887308503: HarperCollins, 2016.

MOTOYAMA, Y.; KNOWLTON, K. Examining the connections within the startup ecosystem: A case study of st. louis. *Entrepreneurship Research Journal*, De Gruyter, v. 7, n. 1, 2017.

NGUYEN-DUC, A.; MÜNCH, J.; PRIKLADNICKI, R.; WANG, X.; ABRAHAMSSON, P. *Fundamentals of Software Startups*. https://link.springer.com/book/10.1007%2F978-3-030-35983-6: Springer, 2020.

NGUYEN-DUC, A.; SEPPÄNEN, P.; ABRAHAMSSON, P. Hunter-gatherer cycle: a conceptual model of the evolution of software startups. In: ACM. *Proceedings of the 2015 International Conference on Software and System Process*. https://dl.acm.org/doi/abs/10.1145/2785592.2795368, 2015. p. 199–203.

NJIMA, M.; DEMEYER, S. Evolution of software product development in startup companies. In: *CEUR workshop proceedings*. http://ceur-ws.org/Vol-2047/BENEVOL$_2017_paper_3.pdf : CEUR,$ 2017.

OLSSON, H. H.; ALAHYARI, H.; BOSCH, J. Climbing the" stairway to heaven"–a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In: IEEE. *2012 38th euromicro conference on software engineering and advanced applications*. https://ieeexplore.ieee.org/abstract/document/6328180, 2012. p. 392–399.

PATERNOSTER, N.; GIARDINO, C.; UNTERKALMSTEINER, M.; GORSCHEK, T.; ABRAHAMSSON, P. Software development in startup companies: A systematic mapping study. *Information and Software Technology*, Elsevier, v. 56, n. 10, p. 1200–1218, 2014.

PREHOFER, C. Feature-oriented programming: A fresh look at objects. In: SPRINGER. *European Conference on Object-Oriented Programming*. [S.l.], 1997. p. 419–443.

RIES, E. *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. https://www.amazon.com.br/Lean-Startup-Entrepreneurs-Continuous-Innovation/dp/0307887898: Crown Books, 2011.

SANTOS, A. R.; MACHADO, I. do C.; ALMEIDA, E. S. de. Riple-hc: javascript systems meets spl composition. In: *Proceedings of the 20th International Systems and Software Product Line Conference*. https://dl.acm.org/doi/abs/10.1145/2934466.2934486: [s.n.], 2016. p. 154–163.

SARASVATHY, S. D. Causation and effectuation: Toward a theoretical shift from economic inevitability to entrepreneurial contingency. *Academy of management Review*, Academy of Management Briarcliff Manor, NY 10510, v. 26, n. 2, p. 243–263, 2001.

SEBRAE/BA, S. B. de Apoio às Micro e Pequenas Empresas da B. Estudo sobre o Ecossistema Baiano de Startups. 2016. Disponível em ⟨https://www.sebrae.com.br/Sebrae/PortalSebrae/UFs/BA/Anexos/EstudosobreoEcossistemaBaianodeStartups.pdf⟩. Acessado em Fevereiro de 2020.

SILVA, F. *Core Scripts for creating a build system to White Label Software Projects*. 2020. Disponível em: ⟨https://github.com/FranklinSilva/white_label_build_system⟩.

SILVA, F.; MACHADO, I. *Multivocal Literature Review for White Lavel Software Projects Supplementary Material*. 2020. Disponível em: ⟨https://zenodo.org/record/4514936\#.YB8em-hKhPY⟩.

SILVA, F.; SOUZA, R.; MACHADO, I. *Supplementary Material to "Taming and Unveiling Software Reuse opportunities through White Label Software in Startups"*. 2020. Disponível em: ⟨https://zenodo.org/record/3673788⟩.

SILVA, F.; SOUZA, R.; MACHADO, I. Taming and unveiling software reuse opportunities through white label software in startups. In: IEEE. *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. https://ieeexplore.ieee.org/abstract/document/9226291, 2020. p. 302–305.

SNOWDEN, D. J.; BOONE, M. E. A leader's framework for decision making. *Harvard business review*, v. 85, n. 11, p. 68, 2007.

SOUZA, R.; SOARES, L. R.; SILVA, F.; MACHADO, I. do C. Investigating agile practices in software startups. In: . https://dl.acm.org/doi/abs/10.1145/3350768.3350786: ACM, 2019. p. 317–321.

SOZEN, N.; MERLO, E. Adapting software product lines for complex certifiable avionics software. In: IEEE. *2012 Third International Workshop on Product LinE Approaches in Software Engineering (PLEASE)*. https://ieeexplore.ieee.org/abstract/document/6229764, 2012. p. 21–24.

SVAHNBERG, M.; GURP, J. V.; BOSCH, J. A taxonomy of variability realization techniques. *Software: Practice and experience*, Wiley Online Library, v. 35, n. 8, p. 705–754, 2005.

TËRNAVA, X.; MORTARA, J.; COLLET, P. Identifying and visualizing variability in object-oriented variability-rich systems. In: *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A*. https://dl.acm.org/doi/abs/10.1145/3336294.3336311: ACM, 2019. p. 231–243.

TOM, E.; AURUM, A.; VIDGEN, R. An exploration of technical debt. *Journal of Systems and Software*, Elsevier, v. 86, n. 6, p. 1498–1516, 2013.

UNTERKALMSTEINER, M.; ABRAHAMSSON, P.; WANG, X.; NGUYEN-DUC, A.; SHAH, S.; BAJWA, S. S.; BALTES, G. H.; CONBOY, K.; CULLINA, E.; DENNEHY, D. et al. Software startups–a research agenda. *e-Informatica Software Engineering Journal*, v. 10, n. 1, 2016.

Appendix

A

SEMI-STRUCTURED INTERVIEWS TRANSCRIPTION

The following is a list of questions asked to each interviewee, whose answers served as a basis to answer the questions asked to stabilize the concept of *white label software products*.

After the description of the questions asked, the audios of the four interviews will be transcribed.

A.1 QUESTIONS ASKED TO THE INTERVIEWEES

The questions to be asked to the interviewees, were elaborated following a logical order based on the research questions of the study.

- **What is your position in the company?** This question was necessary to level the information obtained according to the interviewee's position;
- **What is your experience time in years?** Also a question related to the validation of the information obtained.
- **Have you worked on innovation projects before? If so, how much experience do you have?** This question aimed to understand a little more about the interviewee's background, if his understanding of *White label Software* was related to his experience in innovation projects.
- **How long has your startup been around?** Question asked to establish a relation between the Startups interviewed.
- **Is your startup self-sustainable, that is, does it have enough financial resources to support itself?** This question sought to understand whether the *Startups* in this micro context have financial independence from investors.
- **How much dependence on investors?** If the previous question was answered as true, an approximate percentage of the dependency was asked, so we can understand the stage the startup interviewed was in.

- **What do you mean by white label?** Key question for establishing the relationship between Highly Configurable Software Systems and *White Label Software*.
- **Describe your white label product** Through this question, the interviewer can put in his own words which product is sold, its differentials, points of variability, etc.
- **How the generation/derivation process for a new product is done? how do you manage product versions?** The purpose of this question was to let the interviewee talk about the most interesting aspects of generating your product, without proposing any method of support.
- **Are mobile applications developed in a native or hybrid way?** This question was asked to understand if there is an implementation pattern for *Startups White Label*.
- **If native, does it have an iOS and Android version?** Information about the technology applied if the product is mobile.
- **If hybrid, which framework do you use?** Information about the technology applied if the product is mobile.
- **How many days does it take to generate a product?** As well as the years of experience of the company in the market, respondents were asked how long it took to generate a product, in order to serve as a base value for the generation of products based on the guidelines to be proposed.
- **How much financial resources is spent during the process of generating a product?** Like the question above, the goal is to offer an average of the number of people involved for later comparison.
- **How much more or less does each resource cost?** The final information to define an average cost for product generation per company.
- **How do you define the variability points?** It was explained, if necessary, what would be points of variability within this specific environment. After this explanation, the interviewee then the answer could be done.
- **What is the variation pattern?** A specific question was asked, related to the variation pattern of each startup.
- **How many optional features do you have?** It was asked how many optional features there are in their project, that is, how many requirements are directly configurable and adapted to the customer. This question served as a basis for finding the results regarding the organization's complexity.
- **How many products do you have launched and are available today?**

- **How many products have you made?**
- **Describe the system architecture** With this question, we tried to understand if there is a definition of standard architecture for the projects of *White Label Software*
- **How many databases do you use?**
- **How many web services do you use?**
- **What are the support management systems?**
- **What is the cost of server today in the company monthly?**
- **Do you perform software testing?**

A.2 INTERVIEW TRANSCRIPT

Below is the transcript of the interviews, the real names of the companies and respondents were omitted:

A.2.1 Company K

Interviewer - Today 6/11/2018, beginning the pilot of the series of interviews aimed at White Label Startups, with the representative Silvana Reis. - Let's start, what is your position in the company?

Interviewee - Software Analyst and Partner at the company

Interviewer - How much experience in years?

Interviewee - In the company or in the field?

Interviewer - in the field

Interviewee - 8 years of experience in the field

Interviewer - is this the first time you've dealt with innovation projects?

Interviewee - sim a primeira vez

Interviewer - yes the first time

Interviewee - 3 years

Interviewer - Startup is also 3 years old?

Interviewee - yes, I have been present in the project since the beginning

Interviewer - Has the company paid for itself, or does it still depend on investors?

Interviewee - No, we still depend on investors

Interviewer - what percentage?

Interviewee - 50% of our revenue comes from investments

Interviewer - What do you know as a White Label product?

Interviewee - It is a product that you can give other faces and other features to the same product

Interviewer - Do you have a White Label product? Tell me a bit more about it

Interviewee - We call it K B2B, the idea is that financial market consultants who are licensed by the market, which is the CVM, they can manage portfolios of people who are within their business. For example, as a licensed consultant with CVM, I manage the portfolio of João, Fernando, Marcelo and four other people, so I will manage their portfolio through K B2B, but they will see my brand as a consultant, my name, the name of my company and my ID, but the K that will do this management. So the face is mine in the case, but inside the product is K.

Interviewer - I see! So answer me one thing, how do you go about generating a new product? For example, I am a consultant and I look for you to create my B2B K, how is the derivation process for this product more or less?

Interviewee - An initial registration is made with the ID and after that it is all customized, you put the logo you want, the name you want, you put the people you want to fill that product. The first registration is done by us, but everything else the system was parameterized to understand that the consultant will do what he wants inside.

Interviewer - So, all the things that vary are registered on your portal. And how product versions are managed

Interviewee - The version of the product is the same, what changes is the visibility, management of access to new features.

Interviewer - Is your product a mobile app?

Interviewee - yes, is a mobile app

Interviewer - Is the software a hybrid or native?

Interviewee - It is hybrid with react-native

Interviewer - How many days does it take to generate a product in your scenario?

Interviewee - 1 to 2 days

Interviewer How many resources are allocated in these two days for product generation?

Interviewee - 3 resources

Interviewer How much on average does a resource capable of generating generation receive?

Interviewee - About 4 thousand months per resource

Interviewer How do you define what will vary?

Interviewee - It was when we wrote the project back there, we defined what would be variable, the business premises, what should vary and what should not vary. A study was done before, we took 3 consultants to validate the platform, where they checked if it is in agreement or not with their day-to-day, what we had foreseen and some things were changing during the process.

So there was a consultant who asked us to insert a feature that we didn't have at the time, which was to share the costs, what is that? So, it was not foreseen in our system, I as a consultant, for example, when I indicate you, that you are not part of my investment network, I do not manage your portfolio, but for some reason you want me to see your portfolio . We didn't have that, which was defined halfway.

We adjusted along the way until we reached a point where fixed things were fixed and variables were well defined. So it varies to some extent.

Interviewer - Your variation pattern, as previously said, is everything to the *run-time* of the application, configurable via the portal, nothing done in the compilation step, right?

Interviewee - yes, all modified via the portal

Interviewer - A base, how many optional *features*
Interviewee - I can't say for sure, but I believe that around 12
Interviewer - How many products have already been made?
Interviewee - 4
Interviewer - Anyone in the air?
Interviewee - Yes, one currently on the air
Interviewer - How is your architecture made, an application, a bank and a *web-server*?
Interviewee - Considering the production environment, we have a bank.
Interviewer - Do you do one *web-service* per product or one for all products?
Interviewee - It is a WS only for all products.
Interviewer - How many product support systems do you have?
Interviewee - Only 1 product support, alias, 2 if we consider the *Business Intelligence* module, where we do monitoring.
Interviewer - How much does the server cost to keep this architecture on the air today?
Interviewee - A lot, we already need to expand the database 3 times, we spend in the range of 3 thousand reais per month
Interviewer - Would you mind if I went back for another test interview?
Interviewee - No, I'd love to.
Interviewer - Silvana, thank you very much, we have reached the end of the interview

A.2.2 Company M

Interviewer - Today 11/14/2018, starting the interview with Ricardo Junior, from the company M. - Let's start, what is your position in the company?
Interviewee - I'm a director
Interviewer - How much experience in years in computing
Interviewee - 8 years
Interviewer - How long have you been working on innovation projects
Interviewee - I have always worked on innovation projects
Interviewer - How long has your Startups been?
Interviewee - 2 years
Interviewer - Has your Startup paid for itself?
Interviewee - Yes
Interviewer - How long did it take to pay
Interviewee - It paid off from the first moment, because I already had a basic product and another business that I had failed, so by the time I pivoted (technical term referring to the total change of the business model) I already had a certain range of customers to sell, and as for the technology I made adaptations, so I already started with a positive box.
Interviewer - It is great that you are at this level, which is not common within the interviewees for this study. Now tell me, what do you mean by *White Label*?

Interviewee - It is a product in which you license technology, customizing some features and made available for third parties to use, explore the product in order to sell.

Interviewer - Tell me about your White Label product

Interviewee - At M we offer a platform and customized applications for urban mobility, we provide technology so that entrepreneurs can work with car, mototaxi and taxi applications, similar to Uber. We provide the infrastructure, all the technology for the customer to become the uber in their city.

Interviewer - For cooperatives too?

Interviewee - as well, cooperatives, associations of taxi drivers, groups of individual drivers, who sometimes already run in an application and want to have their own. As well as visionary people, who see in their city an opportunity to explore this service that is not yet explored by other large companies.

Interviewer - How do you do to generate a product? **Interviewee** - It's all manual today, we generate the databases and servers for each client. After configuring the environment, we create the applications with the parameters to communicate with the servers and customize some things depending on the client.

Interviewer - About version control, how do you replicate an improvement for other published applications?

Interviewee - This version upload is also manual

Interviewer - is it the same code or different codes between each application?

Interviewee - Both the application and the server have only the same code that is the basis for the products. There is already a *deploy* process on the server, it automatically generates for each client. However in the applications it is all manual.

Interviewer - How do you make the application A, have the characteristics A

Interviewee - *flavors* are made on android, as a settings folder for each client, with logos, colors etc. For each client I modify these assets

Interviewer - Do you change anything through your management portal?

Interviewee - Some things do, we have a part of the code that the application communicates with the manager. For example, it is possible to add the passenger's cpf, not showing the destination to the driver, some things are personalized by the panel.

Interviewer - is your application native or hybrid?

Interviewee - Native, for android we have passenger and driver app, but for iOS only passenger.

Interviewer - When a customer of yours requires a product they are a website, a panel ..

Interviewee - That, website, dashboard and driver and passenger apps

Interviewer - How many days do you take to generate a product?

Interviewee - Half a day

Interviewer - How many resources do you put in generating a product?

Interviewee - 1 person

Interviewer - How much does this resource on average earn in the month?

Interviewee - On average 5,000

Interviewer - How was it defined which were the variant points between one application and another?

Interviewee - Based on our competitor, we realized that he did not change the client's colors or brands to differentiate himself from others. Now we don't have so many variable features, what really changes is the color part, the brand etc.

Interviewer - How many optional features do you have?

Interviewee - about 15

Interviewer - How many products do you have on the air today?

Interviewee - About 30

Interviewer - How many have been made?

Interviewee - About 45

Interviewer - Tell me a little about your architecture to make it all work

Interviewee - We have a server in node.js, it runs on a separate infrastructure for each client. It uses a separate mongoDb infra, we use the parse server as backend. Our management panel is made in PHP, running on its own server. Each application communicates with its own server. There are some other third-party services (email, push notifications).

Interviewer - In this case there are 30 banks and their general

Interviewee - Perfect

Interviewer - Do you only have this management system for support? Something related to Business Intelligence

Interviewee - Yes, we use third party tools for Business Intelligence, we are planning to start using our clients' data with more emphasis in 2019.

Interviewer - What is the cost of servers today?

Interviewee - The smallest 7 dollars, and the total about 1500 reais

Interviewer - Do you mind if I contact you in the near future about advances in research?

Interviewee - Yes, you can contact me

Interviewer - Thank you very much Ricardo, this is the end of our interview.

A.2.3 Company E

Interviewer - Today is the 12/11 and I am starting the interview with Vinicius Oliveira from Empresa E. What is your position in the company?

Interviewee - Software engineer

Interviewer - How much experience?

Interviewee - 2 years

Interviewer - Have you worked on any innovation project before?

Interviewee - This is the first

Interviewer defined - How long has this startup been around?

Interviewee - The startup turns 5

Interviewer - Has your startup paid for itself?

Interviewee - Unfortunately not, it depends on investors

Interviewer - How many percent dependency?

Interviewee - This end of the year has improved a lot, nowadays 70% dependency

Interviewer - What do you mean by White Label?

Interviewee - White Label would be selling products to different companies and / or people. We provide, for example, a store for each farm, a store for each representative, our White Label is basically this, ours is a store, we deliver to several people.

Interviewer - So basically it is a product where you can sell customized stores to rural producers who already do this without being on the web. Tell me how you do the process of generating a new store?

Interviewee - From the marketing side?

Interviewer - You can focus on the software engineering part, assuming you have all the necessary material

Interviewee - From that, we have a support system like an administrative panel, where a resource that works with us can upload all photos and videos, playing directly in the store.

Interviewer - Is he a computer person or not?

Interviewee - He's an ordinary person, he can be anyone.

Interviewer - Do you have the platform available for Mobile or just web?

Interviewee - Only responsive, but we are in the implementation phase of our hybrid solution

Interviewer - Nowadays, how many days does it take to generate a store?

Interviewee - Having video, photos, pdf, it takes about 3 days

Interviewer - How many resources are allocated for this task?

Interviewee - 2 to 3 people

Interviewer - How much more or less does each resource receive per month?

Interviewee - About 2 thousand reais

Interviewer - How do you define what varies

Interviewee - We already have some things defined, in the administrative panel all this control is done. What types of animals are available for example.

Interviewer - Do you have any person in the area responsible for checking if your system includes all features?

Interviewee - We have salespeople in the area, they know a lot, in addition to our CEO. They are always validating all things

Interviewer - What pattern of variation is everything even via the portal?

Interviewee - Much of it is from the portal, sometimes new things are included in the code to meet specific demands

Interviewer - How many optional features do you have?

Interviewee - About 4 to 5

Interviewer - How many products do you have on the air today?

Interviewee - About 50

Interviewer - How many products were made? that have been discontinued

Interviewee - About 60 in total

Interviewer - Tell me about the system architecture

Interviewee - Would you like to know how our service works? We use Amazon for servers and databases.

Interviewer - How many web-services do you have?

Interviewee - Production server only

Interviewer - Do you have other support systems?

Interviewee - Administration panel only. Customers are able to access their own store's dashboard, but only the company has access to the administrative panel.

Interviewer - What is your server cost?

Interviewee - About 600

Interviewer - Would you mind if I came back later with more questions with or without testing?

Interviewee - We don't have a well-defined testing process at the moment, just code review

Interviewer - Thank you very much Vinicius, this is the end of the interview

A.2.4 Company L

Interviewer - Today 11/15/2018, starting the interview with Pablo Lima, from the company L. - Let's start, what is your position in the company?

Interviewee - I'm a Developer Analyst

Interviewer - Time of experience in years?

Interviewee - 5 years

Interviewer - If you have already worked on innovation projects, if you have experience in it

Interviewee - It was my first time, I've been on the project for 3 years

Interviewer - How long has your startup been around?

Interviewee - 5 years

Interviewer - Is your startup paying for itself?

Interviewee - no

Interviewer - How much dependence on investors?

Interviewee - 50 % dependency

Interviewer - What do you mean by white label?

Interviewee - White label is a business model that focuses on componentization and adaptation of products to the customer's needs, in order to create customized low cost applications

Interviewer - Describe your white label product

Interviewee - It is a customizable education application that offers the possibility of consuming ebooks, videos, courses and assessments based on the customer's needs.

Interviewer - Tell me more about the generation / derivation process for a new product. How do you manage product versions

Interviewee - A customer registration is made on the management portal, where essential application parameters are added, such as means of payment, types of products supported, etc. After that, a folder is created with more customizable information (icons, splash arts, analytics settings, etc.) to be attached to other app version folders, lastly the generation scripts are modified to include a new app. All products follow the same version of the project code

Interviewer - Is your mobile part Native or hybrid?

Interviewee - Hybrid using the Ionic Framework

Interviewer - How many days does it take to generate a product?

Interviewee - 1 day having all the necessary files

Interviewer - How many resources do you put in during the process of generating a product?

Interviewee - 1 feature

Interviewer - How much more or less does each resource cost?

Interviewee - 2 thousand per month

Interviewer - How do you define the points of variability?

Interviewee - The project was built to serve only one project, but since there were more sectors of the education environment that needed this solution, we created basic points of variability that were increasing as new partners entered the project and presented their needs.

Interviewer - What is the variation pattern?

Interviewee - There is no standard to define the variation, some attributes are made in the runtime, such as the available product tabs. Other features are defined in compile time, such as the settings for push notifications.

Interviewer - How many optional features do you have? Say what points of variability are if people don't know

Interviewee - I believe that some 25

Interviewer - How many products do you have on the air today?

Interviewee - Currently 7 on the air

Interviewer - How many products have you made?

Interviewee - 20 already done

Interviewer - Tell me a little about your architecture:

Interviewee - We use Azure to assist us in deploying and managing servers and databases

Interviewer - How many databases and web services do you work with?

Interviewee - 1 database and 1 *web-service*

Interviewer - What are the support management systems?

Interviewee - We have two support systems for the project, the manager where some features are configured only by the administrator and our BI system.

Interviewer - What is the cost of server today in the company monthly?

Interviewee - Thousand reais per month

Interviewer - Do you agree to receive the results and be available for a new conversation, perhaps related to tests?

Interviewee - Yes you can

Interviewer - Thank you very much Pablo, this is the end of the interview

Appendix B

SELECTED RESOURCES FOR CONDUCTING THE MLR

B.1 RESOURCE LIST

The resource list of the Multivocal Literature Review (MLR) conducted are the following:

RS1 - **Startup brasileira cria primeira plataforma white label para influencers**: ⟨http://bit.ly/whitelabel-rs1⟩. *Last accessed: Feb 18th, 2021*

RS2 - **Can a White Label Bring You $20,000 per Month? Here is a Way to Launch a Lucrative FinTech Business in 2 Weeks**: ⟨http://bit.ly/whitelabel-rs2⟩. *Last accessed: Feb 18th, 2021*

RS3 - **How To White Label An Entire Company**: ⟨http://bit.ly/whitelabel-rs3⟩. *Last accessed: Feb 18th, 2021*

RS4 - **Should you white-label your product?**: ⟨http://bit.ly/whitelabel-rs4⟩. *Last accessed: Feb 18th, 2021*

RS5 **What startups have created successful white label products and services?**: ⟨http://bit.ly/whitelabel-rs3⟩. *Last accessed: Feb 18th, 2021*

RS6 **Plataformas White Label, a chave do sucesso no investimento digital**: ⟨http://bit.ly/whitelabel-rs6⟩. *Last accessed: Feb 19th, 2019*

RS7 **What is White Label**: ⟨http://bit.ly/whitelabel-rs7⟩. *Last accessed: Feb 18th, 2021*

RS8 **White Label VS Private Label: What is the Difference**: ⟨http://bit.ly/whitelabel-rs8⟩. *Last accessed: Feb 18th, 2021*

RS9 **What is White Labeling? How Does It Differ From Private Labeling?**: ⟨http://bit.ly/whitelabel-rs9⟩. *Last accessed: Feb 18th, 2021*

RS10 **7 White-Label Business Opportunities for Entrepreneurs and Agencies**: ⟨http://bit.ly/whitelabel-rs10⟩. *Last accessed: Feb 18th, 2021*

RS11 **Why A White Label Solution Is Easier Than Building Your Own**: ⟨http://bit.ly/whitelabel-rs11⟩. *Last accessed: Feb 18th, 2021*

RS12 **White-label product**: ⟨http://bit.ly/whitelabel-rs12⟩. *Last accessed: Feb 18th, 2021*

RS13 **What is White Label? Products, Software, and More**: ⟨http://bit.ly/whitelabel-rs13⟩. *Last accessed: Feb 18th, 2021*

RS14 **What are the 7 white labeling benefits**: ⟨http://bit.ly/whitelabel-rs14⟩. *Last accessed: Feb 18th, 2021*

RS15 **The Ultimate Guide To White Label Services in 2020**: ⟨http://bit.ly/whitelabel-rs15⟩. *Last accessed: Feb 18th, 2021*

RS16 **White Labeling, what are the pros and cons** ⟨http://bit.ly/whitelabel-rs16⟩. *Last accessed: Feb 19th, 2019*

RS17 **What to Consider Before White Labeling Your App**: ⟨http://bit.ly/whitelabel-rs17⟩. *Last accessed: Feb 18th, 2021*

RS18 **What are white label apps?**: ⟨http://bit.ly/whitelabel-rs18⟩. *Last accessed: Feb 18th, 2021*

RS19 **How White Label Services Help Your Agency Grow**: ⟨http://bit.ly/whitelabel-rs19⟩. *Last accessed: Feb 18th, 2021*

RS20 **White-labeling vs. co-branding, which one is right for your business?**: ⟨http://bit.ly/whitelabel-rs20⟩. *Last accessed: Feb 18th, 2021*

RS21 **Why I hate White Labeling for Startups**: ⟨http://bit.ly/whitelabel-rs21⟩. *Last accessed: Feb 18th, 2021*

RS22 **7 Experts Weigh In on How to Build a Successful White Label Agency**: ⟨http://bit.ly/whitelabel-rs22⟩. *Last accessed: Feb 18th, 2021*

RS23 **Your very own bank, made to order**: ⟨http://bit.ly/whitelabel-rs23⟩. *Last accessed: Feb 18th, 2021*

RS24 **AMAZON WHITE LABEL PRODUCTS 2020: HOW TO FIND PROFITABLE IDEAS NICHES**: ⟨http://bit.ly/whitelabel-rs24⟩. *Last accessed: Feb 18th, 2021*

RS25 **White Label Business Opportunities – Resell Products With Your Brand**: ⟨http://bit.ly/whitelabel-rs25⟩. *Last accessed: Feb 18th, 2021*

RS26 **Trouble with colour: art directing a white label product is hard**: ⟨http://bit.ly/whitelabel-rs26⟩. *Last accessed: Feb 18th, 2021*

RS27 **The white labelling business model**: ⟨http://bit.ly/whitelabel-rs27⟩. *Last accessed: Feb 18th, 2021*

RS28 **How to create a White Label website**: ⟨http://bit.ly/whitelabel-rs28⟩. *Last accessed: Feb 18th, 2021*

RS29 **The ultimate white label development checklist**: ⟨http://bit.ly/whitelabel-rs29⟩. *Last accessed: Feb 18th, 2021*

RS30 **White Labelling vs Co-branding – Which is Right for Your Business?**: ⟨http://bit.ly/whitelabel-rs30⟩. *Last accessed: Feb 18th, 2021*

RS31 **White Labeled Software Strategies for Startups**: ⟨http://bit.ly/whitelabel-rs31⟩. *Last accessed: Feb 18th, 2021*

RS32 **So, You Wanna White Label?**: ⟨http://bit.ly/whitelabel-rs32⟩. *Last accessed: Feb 18th, 2021*

RS33 **What startups have created successful white label products and services?**: ⟨http://bit.ly/whitelabel-rs33⟩. *Last accessed: Feb 18th, 2021*

RS34 **White-Label is the New Black for Startups**: ⟨http://bit.ly/whitelabel-rs34⟩. *Last accessed: Feb 18th, 2021*

RS35 **White Label Agreement in Simple Terms**: ⟨http://bit.ly/whitelabel-rs35⟩. *Last accessed: Feb 18th, 2021*