LOUVAIN
RESEARCH INSTITUTE
IN MANAGEMENT
AND ORGANIZATIONS

WORKING PAPER

2017/12

Agile Product Line Engineering: the AgiFPL method

Hassan Haidar,
Manuel Kolp,
Louvain Research Institute in Management and Organizations

Yves Wautelet,
KULeuven, Faculty of Economics and Business

LOUVAIN RESEARCH INSTITUTE IN MANAGEMENT AND ORGANIZATIONS

Louvain School of Management Research Institute
Working Paper Series
Editor: Prof. Frank Janssen
(president-lourim@uclouvain.be)

Agile Product Line Engineering: the AgiFPL method

Hassan Haidar, Louvain Research Institute in Management and Organizations
Manuel Kolp, Louvain Research Institute in Management and Organizations
Yves Wautelet, KULeuven, Faculty of Economics and Business

Summary

Integrating Agile Software Development (ASD) with Software Product Line Engineering (PLE) has resulted in proposing Agile Product Line Engineering (APLE). The goal of combining both approaches is to overcome the weaknesses of each other while maximizing their benefits. However, combining them represents a big challenge in software engineering. Several methods have been proposed to provide a practical process for applying APLE in organizations, but none covers all the required APLE features. This paper proposes an APLE methodology called AgiFPL: *Agile Framework for managing evolving PL with the intention to address the deficiencies identified in current methods, while making use of their advantages.*

Keywords : Software Product Line Engineering, Agile Software Development, Agile Product Line Engineering, AgiFPL, Scrum, Scrumban, Features.

JEL Classification: L86.

Corresponding author :
Hassan Haidar
CEMIS
Louvain Research Institute in Management and Organizations / Campus of Louvain-la-Neuve
Université catholique de Louvain
Place des Doyens 1
B-1348 Louvain-la-Neuve, BELGIUM
Email : hassan.haidar@uclouvain.be

1. INTRODUCTION

Over the past years, the software industry has been searching for methods, processes and techniques to increase the delivery of high quality software while reducing development costs. Several paradigms have been proposed in order to fulfil this target. "Agile Software Development" (ASD) and "Product Line Engineering" (PLE) are well-known approaches proposed by researchers and practitioners for dealing with the growing complexity of information systems and handling competitive needs of the IT industry [9]. The popularity of both approaches and their positive results have motivated researchers to find ways to integrate them [14] leading to "Agile Product Line Engineering" (APLE).

The main goal of APLE is to maximize the benfits of each individual approach and to fulfil their common goals [18, 19]. Moreover, combining ASD and PLE presents a significant advantage in terms of "synergy" in which, each approach addresses the weaknesses of the other. But, among the proposed APLE methodologies, only few have proposed a specific process for this combined approach, and can hence be actually referred to as true APLE methodologies [2].

After reviewing the literature on APLE and anlysing, comparing and evaluating the relevant APLE methodologies, each method can be considered as presenting strengths and weaknesses. Furthermore, the study of the existing APLE methods has revealed that no single method covers all the needed APLE features.

In this context, this paper proposes an APLE methodology called AgiFPL (Agile Framework for managing evolving SPL). The intention behind the definition of this APLE method is to address the deficiencies identified in current methods, while making use of their advantages.

1.1 Research method

In order to achieve our target we started by identifying, studying and evaluating the relevant APLE methodologies that exist in the literature. [12] and [9] have already surveyed several APLE methods in their systematic literature reviews. [14, 26] have presented a recent study in which they surveyed and analyzed APLE methodologies in a more precise and systematic manner through "criteria-based evaluation". On this basis, we performed an iterative evaluation of the methodologies according to a "criterion set". The results of the evaluation performed in each iteration have allowed us to obtain a deeper insight into the features of the methodologies. The results are therefore used to identify new criteria and thus enrich the criterion set. Indeed, the results of this criteria-based evaluation have highlighted the strengths and weaknesses of each methodology, have specified the features expected of APLE methodologies, and have identified the shortcomings of methodologies. Thus, these results could be used to improve current and future APLE methodologies.

Secondly, we have studied and evaluated some ASD methodologies such as Scrum [7], XP, Scrumban [24]. We aim with this evaluation to identify methods or reusable method chunks that could help us to produce our intended APLE. The results of this evaluation are used as a basis to select and assemble reusable method chunks, instantiate abstract process frameworks, and extend existing methodologies in order to define the bespoke APLE methodology.

Thirdly, after designing and defining the proposed APLE we aim to validate it by establishing an exhaustive case-study that shows the applicability of the bespoke APLE. Due to lack of space, we only present a short illustration in this paper.

1.2 Contributions

As said, this paper is an effort to propose an adequate APLE method that addresses the lacks and deficiencies (see Section 4) of classical APLE methods. For this purpose, we focus on the agile methods Scrum [7] and Scrumban [24] as a basis for AgiFPL as they adapt well to software evolution and are widely used by the agile community. In addition, we focus on the requirement engineering disciplines of I-Tropos [36] (i.e. Organizational modeling and Requirement Engineering) in order to define the RE process of AgiFPL.

According to [3] and (Hersari et al, 2010) software development methodologies consist of two integral parts: a "modeling language" and a "process". The modeling language part provides the syntax and semantics used to express the products, whereas the process prescribes the flow of activities that should be performed and explains how the products should be produced, enhanced and exchanged along this flow. The agility feature of APLE methods has deemphasized the role of modeling, and hence the modeling language. In the proposed APLE, we presented an adequate Requirement Engineering (RE) process for both Domain Engineering (DE) and Application Engineering (AE). The proposed RE process of AgiFPL represents a trade-off between the requested RE elicitation and the agility feature. The target of this RE process is to introduce a suitable reuse strategy and reduce the upfront design during the AgiFPL development process. Moreover, for the process part AgiFPL integrates Scrumban for the DE and Scrum for the AE.

The paper is structured as follows. Sections 2, 3 and 4 overview PLE, I-Tropos and ASD. Section 5 highlights the current context of APLE. Section 6 presents our AgiFPL framework. Section 7 introduces an illustration with part of case study. Finally we conclude the paper in Section 8.

2. PRODUCT LINE ENGINEERING

Software Product Line Engineering or Product Line Engineering (SPLE or PLE) provides an efficient way to build and maintain portfolios of systems that share common features and capabilities.

On the other hand, Software Product Lines (SPL/PL) aim at improving software vendors to tailor software products to the requirements of individual customers [1].

There are two essential phases to develop SPLs: Domain Engineering (DE) and Application Engineering (AE) [29].

DE is the phase of creating a set of reusable assets in order to build systems in a specific problem domain [12]. It determines the scope and handles the commonalities and the variability, i.e. defines the core-assets, points of variations, and expected variants for all the products of the PL [22]. The architecture can be designed once the domain has been specified. The Product-Line Architecture is considered flexible in the sense that it

is capable of accommodating potential members of the PL and, proactively, addressing the variability [34]. Reusable assets are implemented and tested, resulting in what is known as a common platform or core-asset base [21]. DE is summarized in the following set of activities or practices [28]:

1. Domain identification and scoping;
2. Requirements engineering;
3. Feature modelling;
4. Architecture design;
5. Core-assets development;
6. Traceability management;
7. Evolution and maintenance.

The AE phase consists of developing products through systematic reuse of core-assets by deriving the PL variability points, i.e. reusable assets are extended from variability points with the selected variants for a specific product [8, 29]. AE is summarized in the following set of activities and practices [12]:

1. Release planning (plan for product applications);
2. Product configuration (model application, architecture application, platform application);
3. Product development;
4. Test, integration and deployment;
5. Traceability management;
6. Evolution and maintenance.

Figure 1 shows a framework for PLE, which divides the product line into DE and AE.

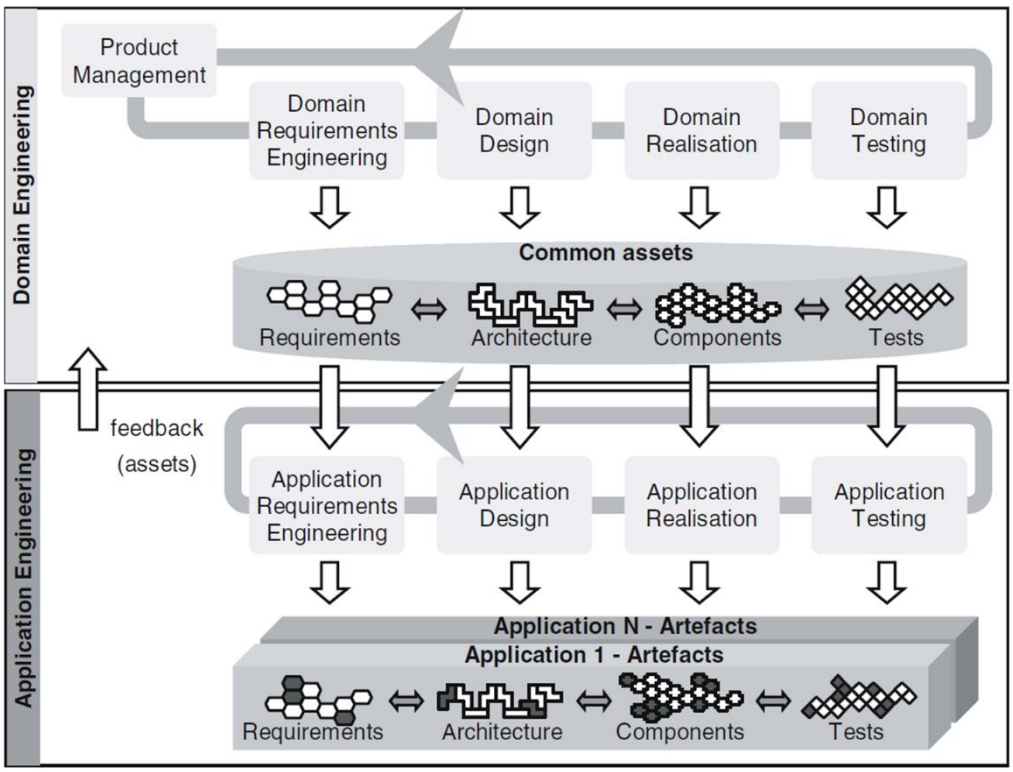

Figure 1 The Software Product Line Engineering framework [29]

3. I-TROPOS

I-Tropos [36] was built-up based on the combination of previously validated methodologies such as Tropos and RUP. Indeed, I-Tropos basically constitutes an enhanced version of the Tropos process where extensions are intended to furnish techniques and tools (driven by services) to deal with scalability and develop iteratively [37].

I-Tropos disciplines are "Organizational Modelling", "Requirements Engineering", "Architectural Design", "Detailed Design", "Implementation", "Test", "Deployment", and "Project Management".

Process phases to be planned within I-Tropos are "Setting", "Blueprinting", "Building" and "Setuping".

Each phase is made of several iterations. An "I-Tropos development" consists thus of disciplines repeated in a spiral and incremental manner while the effort spent on each discipline varies from one iteration to the other.

4. AGILE DEVELOPMENT

Agility is an umbrella term for a variety of agile methods that are based on the Agile Manifesto [33] principles and values. Agile Software Development (ASD) is an approach that is intended to enable rapid and flexible development of small-scale software solutions from scratch, usually addressing a single, well-defined customer [32].

Basically, ASD methods emphasizes the following principles [18]:

1. Self-organizing teams;
2. Close cooperation with the customer;
3. Informal and direct communication.

Documentation, including plans, is kept at a minimum, and work is organized in short iterations developing the software product in increments, continuously testing it in collaboration with customers and refactoring it if needed (See Figure 2).

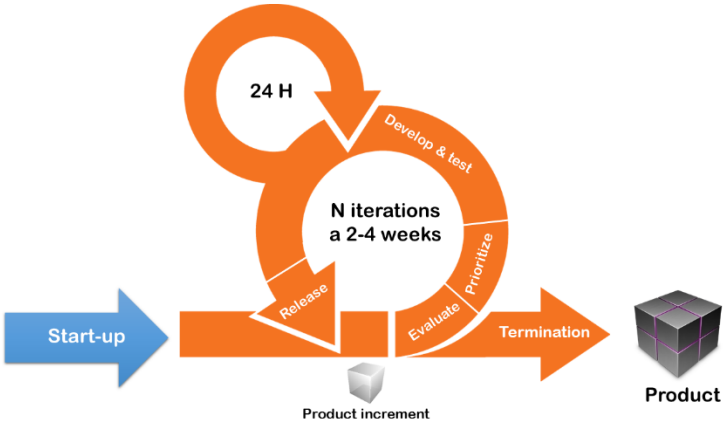

Figure 2 Basic ASD process [33]

External stakeholders such as clients and third-party actors participate in nearly all the steps of the project. Each iteration usually lasts only a few weeks and starts out by getting feedback from stakeholders on the results of the previous iteration. This feedback is based on a practical try-out or running demonstration of the product increment. Furthermore, stakeholders also participate in the prioritizing requirements for the iteration ahead [25].

All these methods, such as Scrum, XP, and Scrumban implement iterative incremental life cycles, share common values, and principles with each one of them defining their own practices. We only focus in this paper on Scrum and Scrumban since we will use them in the design of AgiFPL.

4.1 Scrum

In Scrum, projects move forward through aggressive deadlines via a series of iterations called sprints to implement dynamic requirements pulled from a backlog. Each sprint is typically two to four weeks long, at the end of which the performed work should create something of tangible value to the stakeholders (Chon, 2013 [23].

The common concepts in Scrum are:

- *Scrum roles:* "Product Owner", "Scrum Master", "Development Team", and "Stakeholders".
- *Scrum artefacts:* "Product backlog", "Sprint backlog", "User Story" (US), "Task", "Burn down charts", "Impediment backlog" and "Definition of done".
- *Scrum meetings:* "Sprint planning 1", "Sprint planning 2", "Daily Scrum", "Sprint review", "Sprint retrospective", and "Backlog refinement".

Each sprint has a sprint-planning meeting at the sprint beginning in which the Product Owner and Team plan together what is to be done for the sprint. The result is the sprint backlog, a list of US and tasks that must be performed to achieve the sprint goal. At the end of each sprint, the sprint review and retrospective meetings are held to put into practice continuous improvement.

4.2 Scrumban

Scrumban proposes a transition method for moving software development teams from Scrum to a more evolved development framework. Since the introduction of Scrumban, organizations have layered the Kanban Method alongside Scrum to help them achieve several different kinds of outcomes [24].

Experiences demonstrate that Scrumban has evolved to become a family of principles and practices that create complementary tools and capabilities. Briefly, over the years, Scrumban has been used to help teams and organizations accelerate their transitions to Scrum from other development methodologies [31].

The main Scrumban elements are summarized below:

- Scrumban *roles* are Product owner, Scrumban sensei, Development team, Stakeholders.
- Scrumban *artifacts* are Product backlog, Selected backlog, Story in Progress (SIP) Backlog, Task backlog, User Story (US), Task, Work in Progress (WIP) Limit, Parking lot, Cumulative flow diagram (CFD), Impediment backlog.

- *Meetings* in Scrumban are “Planning 1” (The “what”: Whenever the product owner pulls new user stories into the selected backlog.), “Planning 2” (The “how”: Whenever team members pull new user stories into the production flow.), “Daily”, “Review”, “Retrospective”, and “Andon” (Whenever a problem occurs).

Scrumban combines the framework of Scrum with the principles of Kanban. It is more prescriptive than Kanban, which has no roles or meetings, but in the same time also more responsive to change than Scrum, where change can only be accommodated at the sprint boundary [31]. Scrumban retains all the roles and meetings of Scrum but uses the Kanban continuous workflow board. The daily stand-up focusses on flow of tasks across the board and reviews what it would take to move each one forward (Thompson, 2015).

5. AGILE PRODUCT LINE ENGINEERING

Software product lines and agile methods have been an effective solution to deal with the growing complexity of software and handl competitive needs of software organizations. There has been growing interest in whether the integration of Agile and PL could provide further benefits and solve many of the outstanding issues surrounding software development [9].

The following methods are proposed as APLE methods and are considered as relevant:

- Component-Driven Development (CDD) [30, 35];
- Extend FAP [11];
- RiSE Process for Product Line Engineering (RiPLE-SC) [4];
- A-Pro-PD [28];
- Tailoring the Scrum Development Process to Address APLE [13];
- An Iterative Model for Agile Product Line Engineering [16];
- Extreme Product Line Engineering [15];
- Agile Approach for Software Product Lines Scoping [10];
- Agile PuLSE-I [5];
- Agile product line planning [27];
- Reactive Variability Management in Agile Software Development [17].

The main shortcomings (deficiencies) [14, 20] of these methodologies concern the following groups of criteria:

1. General criteria (e.g. modeling language, the process);
2. Criteria related to the characteristics of agile methods (teams, flexibility, leanness etc.);
3. Criteria related to the PL characteristics (DE activities, AE activities, core assets, scope, features, etc.);
4. Criteria related to the common goals of agile and PL (on-time software delivery, software quality, HR management, management of changes in requirements, etc.);
5. Criteria related to issues arising due to the combination of the two approaches (Reuse approach, basis of the methodology, etc.).

Reviewing the existent APLE methodologies shows the need of a new approach that would cover all the following APLE characteristics [12, 14]:

1. Full coverage of the generic software development lifecycle;
2. Comprehensive and precise definition of the methodology;
3. Sufficient attention to the non-SDLC activities (umbrella activities);
4. Prescription of a specific modeling language;
5. Provision of model examples;
6. Attention to learning at project and portfolio levels;
7. Attention to active user involvement;
8. Management of expected and unexpected changes.

6. OUR AgiFPL FRAMEWORK

The intention behind our AgiFLP framework is to develop a proper APLE methodology which effectively combines agility with PLE, and embodies the advantages of both approaches.

This section presents how our APLE is tackled by means of a tailored Scrum, Scrumban, and I-Tropos in which the DE and AE processes are performed in an iterative and incremental way. To overcome the PL-architecture challenge and reducing the upfront design, it is necessary to define the mechanisms and strategies to be applied during the APLE development process. This is why our contribution is based on two main aspects: *requirement engineering (RE)* and *development process* [38, 39]. In fact, integrating the disciplines of I-Tropos [40] that concern the requirement engineering will provide the mechanism to easily evolve PL-architectures in an agile context and will establish a suitable reuse strategy. Furthermore, adopting Scrumban and Scrum for the development processes in DE and AE will establish the agility of our method.

AgiFPL is thus an agile methodology designed to improve the agility within the PLE and effectively meet any new emerged business expectations. The main goal of AgiFPL is to move teams from the classical approach to a more evolved APLE framework.

In addition, the proposed approach adopts forecasting and metric models and some project management practices in order to address some management issues.

In other words, on the one hand, for the DE stage, AgiFPL implements an iterative process that combines some of I-Tropos disciplines with Scrumban. At this stage, the roles, activities, artefacts, board, and meetings of Scrumban are adapted to be used within the different phases of DE. On the other hand, for the AE stage, AgiFPL also implements an iterative process that combines the Scrum approach with I-Tropos disciplines. Moreover, the AE stage, roles, activities, artefacts, and meetings of Scrum are used by the different phases of AE.

AgiFPL is a two tier framework, where the first layer modeled in Figure 3, is dedicated to the Domain Engineering (DE) and the second one, modeled in Figure 4, is dedicated to the Application Engineering (AE). Moreover, the framework considers two spaces i.e. *Problem Space* and *Solution Space*.

Each tier has its start point. At the DE tier, the start point is the *"Software Vendor"* and at the AE tier, the start point is the *"App Owner"*.

6.1 DE tier

The DE tier is detailed in Figure 3. First, the *"Software Vendor"* has a business strategy for the domain. This strategy is built from market trends, potential investment opportunities and core-business improvements. The domain strategy constitutes the main base to define the scope of the domain and start the phase of *"Domain requirement engineering" (DRE)*.

The strategy pushes to start the *DRE phase*, first step in the problem space. DRE is a sub-process that aims to understand the problem space by studying the settings of the organizations related to the domain. Roles involved in this sub-process are "Domain Experts", "Domain Sensei" and "Development Team". During this sub-process, the people involved model the target domain in terms of social actors and their intentions. In addition, when the "Software Vendor" adopts a new or modified strategy, the "Domain Experts" pull this strategy and upgrade the "Domain Requirements" [41].

After receiving all the documentation and models delivered at the end of the DRE phase, the *"Domain Design" (DD)* phase starts. The *DD phase* takes the reference RE models as input and creates a reference architecture for the product line platform, which gathers the common assets. At this phase, the scope of the domain and the architectures have to be determined, i.e., decide which products should be covered by the product line and, consequently, which features are relevant and should be implemented as reusable artefacts. The DD phase contains two primary tasks: *domain scoping* and *domain modeling*.

Once commonalities and variabilities are identified and *feature models* are generated at the DD phase, "Domain Experts" define the *"Features Backlog" (FB)*. During this phase, the people involved document features in *"User Stories" (US)* format [6]. At this stage, stories contain rough estimates of both business value and development effort. At the end of this stage, the "Domain Experts" build a *"Selected Backlog" (SB)* with defined capacity from the FB. The SB represents a list of tasks the "Development Team" must address next. It has a defined capacity limit. As soon as capacity is available, it is filled up with user stories/features from the top of the FB.

After building the SB, the "Domain Experts" and "Development Team" hold the *"Planning 1"* meeting in order to shape stories and estimate "Cycle Time". "Planning 1" is considered as the "WHAT": Whenever the "Domain Experts" pull new user stories into the SB. The "Domain Experts" holds it to select the next user stories to work on, explaining the user stories of features backlog and answering open questions. After this analysis, the "Development Team" should understand the different features and be able to estimate the complexity of each user story.

Once "Planning 1" has taken place, the "Development Team" structures the *"Stories in Progress Backlog" (SIP Backlog)* with defined *capacity (K[n])*. The SIP Backlog is a list of user stories which the development team currently addresses. Team members pull user stories from the selected backlog when no tasks remains in the task backlog.

After structuring the SIP Backlog, the "Domain Sensei" and "Development Team" hold the *"Planning 2"* meeting. With "Planning 2" starts the *"Domain Implementation or*

Production of Common Assets" phase. The "Planning 2" is considered as the "HOW": Whenever team members pull new user stories into the production flow. Here, the team discusses solutions for new user stories in the SIP backlog and creates tasks for each user story accordingly. During the "Planning 2" meeting, the team establishes the *"Production Flow"* consisting of the following components:

1. Task Backlog;
2. Task in Progress with defined capacity (max k[6]). Once the "Task in Progress" is done, team members pull tasks from "Task Backlog".
3. Task Done with "Parking Lot". Once the story is completed (Story complete?) "Development Team" members pull user stories from the "SIP Backlog".
4. Story Testing with defined capacity (max k[2]);
5. Stories Done;

During the "Production Flow", the "Domain Sensei" and "Development Team" hold a *"Daily" meeting,* which is a short, time-boxed meeting, taking place every day at the same time. Every team member answers three questions:

1. What have I done since yesterday?
2. What am I planning to do today?
3. What are my impediments? Note that if there are impediments, these are maintained by the "Domain Sensei" in a list called *"Impediment Backlog"*.

Whenever the team ships an increment *(Increment Ready?)*, the "Domain Experts", "Domain Sensei" and "Development Team" hold a *"Review" meeting* in order to review the work accomplished. The team uses this meeting to present and review the work it has completed since the last delivery. Usually, it also includes a demonstration of the features created during the latest increment or iteration. Once the features have been implemented as reusable artefacts (Models, Source Codes,…), these are placed in the *"Common Assets Warehouse"*.

After any "Review", the "Scrumban Sensei" holds the *"Retrospective" meeting* to reflect on the past production cycle in order to ensure continuous process improvements. The "Domain Sensei" always asks two questions in the retrospective:

1. What went well during the last cycle?
2. What should be improved in the next cycle?

During the "Production Flow", whenever a problem occurs, the "Domain Sensei" organizes an *"Andon" meeting*. For example, a story is over the expected cycle time, or a task is frequently re-assigned and not yet solved. Both the "Development Team" and the "Domain Experts" participate in this meeting and work on a solution to solve the open issue.

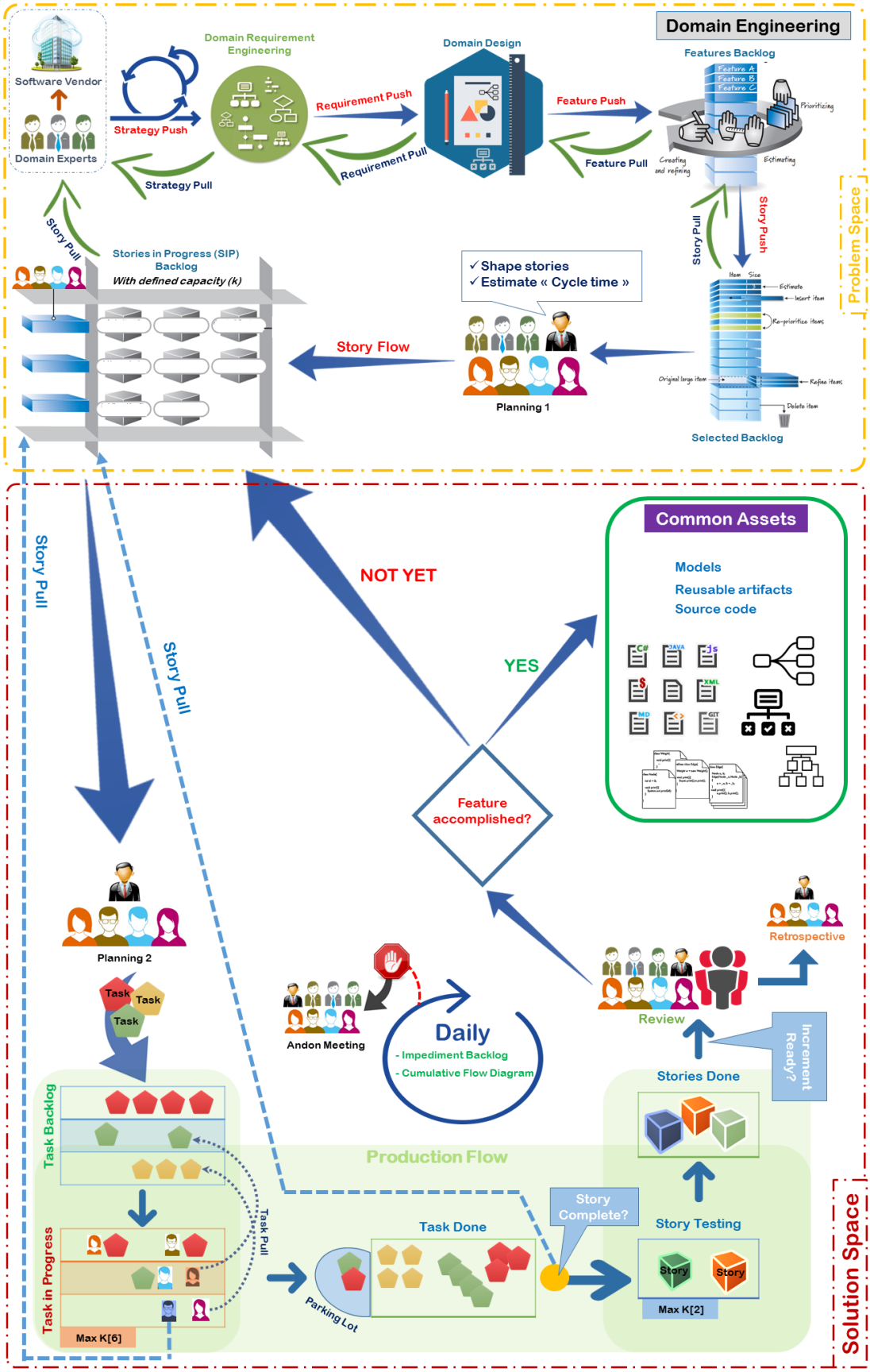

Figure 3 Domain Engineering (DE) process according to AgiFPL

6.2 AE tier

The AE tier, represented in Figure 4 includes several product lines where the outputs are client applications (products). At this stage, *"App i Owners"* and *"App i Stakeholders"* are more involved in the development process.

Hereafter, we describe the AgiFPL proposition for the *"Line i"* which deliver at the end the *"App i"*. In fact, AgiFPL propose a simple agile. Each line of the AE tier will follow this process.

First, an "App i Owner" comes with an idea of what he/she wants to create. The App Line process starts and launches the *"App i Requirement Engineering" (ARE i) phase* [42]. This phase has two main missions:

1. The "App i Owner", "App i Stakeholders", "Line i Master", and "Development Team i" study the "Organization i" in order to understand the motivations and rationales that underline the "App i" requirements.
2. The "App i Owner" idea could be large, therefore, through an activity called "Grooming"; it is broken down into a set of features that are collected into a prioritized list called the *"App backlog"*.

The requirement assets produced in "Domain Engineering" constitute some basis, but they will not satisfy all "App Owners" and "App Stakeholders" requirements. The gap between what is available and what is required must be analysed, and a trade-off decision has to be taken for each unsatisfied requirement. At the end of the ARE phase two types of features coexist:

1. Features that exist and could be selected from the "Common Assets Warehouse". The set of these is called *"Selection of Features" (SoF)*.
2. Features that are required for the development of the application and do not exist in the "Common Assets Warehouse". Since these features are defined during the ARE phase, the set of these is called *"Definition of Features" (DoF)*.

Consequently, the building of the "App backlog" is finished and it contains an ordered list of documented "Use Stories" (US) that the team maintains for an "App". Note that all *new reusable artefacts* issued from the AE process have to be classified in the "Common Assets Warehouse".

Once the "App backlog" is finished, the work is performed in "Sprints" up to a calendar month. The work completed in each sprint should create something of tangible value to the customer or user. The "Sprint" starts with the *"Sprint Planning"*. The "App i Owner", "Line i Master" and "Team Development i" perform the "Sprint Planning" in order to determine the most important subset of "App backlog" items to build in the next sprint. During the sprint planning, the "App i Owner" and "Development team i" agree on a sprint goal that defines what the upcoming sprint is supposed to achieve. The "Sprint Planning" can be split into two meetings:

- *"Sprint Planning 1":* (60 min per sprint week) is held to select the work to be done for the next sprint (the "WHAT").
- *"Sprint Planning 2":* (60 min per sprint week) the designing phase for the selected backlog (the "HOW").

At the end of the "Sprint Planning", the "Sprint Backlog" is defined and the *"Definition of Done" (DoD)* list is established. In fact, DoD is a checklist of activities required to declare the implementation of a story to be completed.

Once the concerned people consider the “Sprint Planning” as finished and agree on the content of the next sprint, the “Development Team i”, guided by the “Line i Master” coaching, performs all the task-level work necessary to get the tasks done. In fact, “done” means there is a high degree of confidence that all of the work necessary for producing good-quality features has been completed. This step is called *“Sprint Execution”*. Tasks the team exactly performs depend on the nature of the work.

Every day during the sprint, ideally at the same time, the “Development Team i” members hold a time-boxed *“Daily” meeting*. This *“inspect-and-adapt” activity* is sometimes referred to as the daily stand-up. At the “Daily” meeting, each team member answers three questions for the benefit of the other team members:

1. What have I accomplished since the last “Daily”?
2. What do I plan to work on by the next “Daily”?
3. What are the obstacles or impediments that are preventing me from making progress?

The “Sprint” results are considered as a *“Potentially shippable App increment”*, meaning that whatever the “Development Team i” has agreed to do is really done according to its agreed-upon definition of done.

At the end of the “Sprint”, there are two additional “inspect-and-adapt” activities:

1. *“Sprint Review”:* The goal of this activity is to inspect and adapt the product that is being built. Critical to this activity is the conversation that takes place among its participants, which includes the “App i Owner”, “App i stakeholders”, “Sponsors”, “Line i Master”, “Development Team i”, and interested members of other teams. The conversation is focused on reviewing the just-completed features in the context of the overall development effort.
2. *“Sprint Retrospective”:* This activity frequently occurs after the sprint review and before the next sprint planning. Whereas the sprint review is a time to inspect and adapt the product, the sprint retrospective is an opportunity to inspect and adapt the process.

Once the “Sprint Retrospective” is completed, the whole cycle is repeated again — starting with the next sprint-planning session, held to determine the current highest value set of work for the team to focus on. At the end of the “Sprint” the “Line i Master”, “Development Team i”, and “App i Owner” perform a “Backlog refinement”: (max. 60 min) used to introduce and estimate new backlog items and to refine existing estimations as well as acceptance criteria. It is also used to break large stories into smaller ones.

After an appropriate number of sprints have been completed, the “App i Owner’s” vision will be realized and the solution can be released.

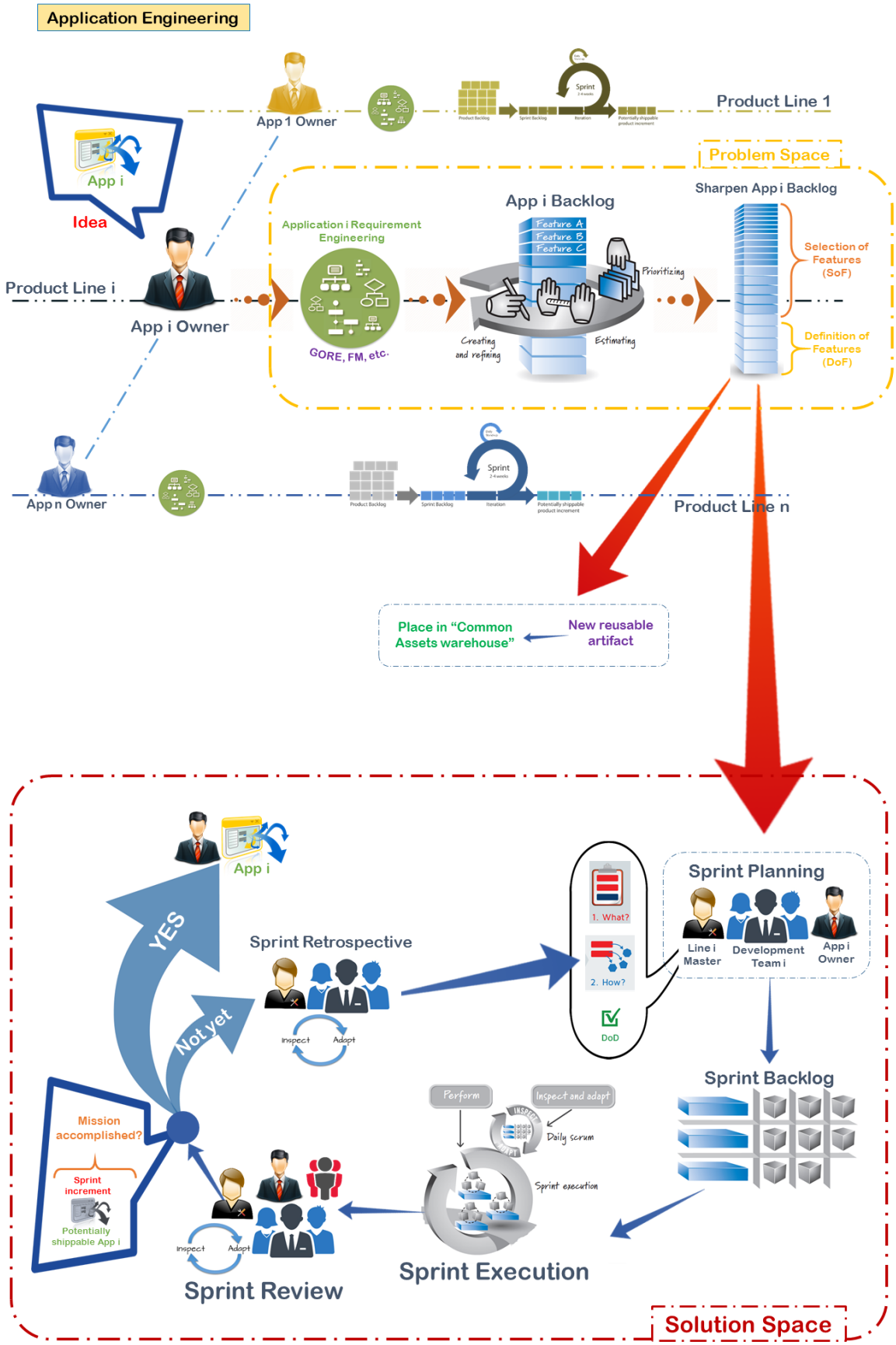

Figure 4 Application engineering (AE) process according to AgiFPL

6.3 AgiFPL processes

Each process of AgiFPL is based on an iterative and incremental development as shown in Figure 5.

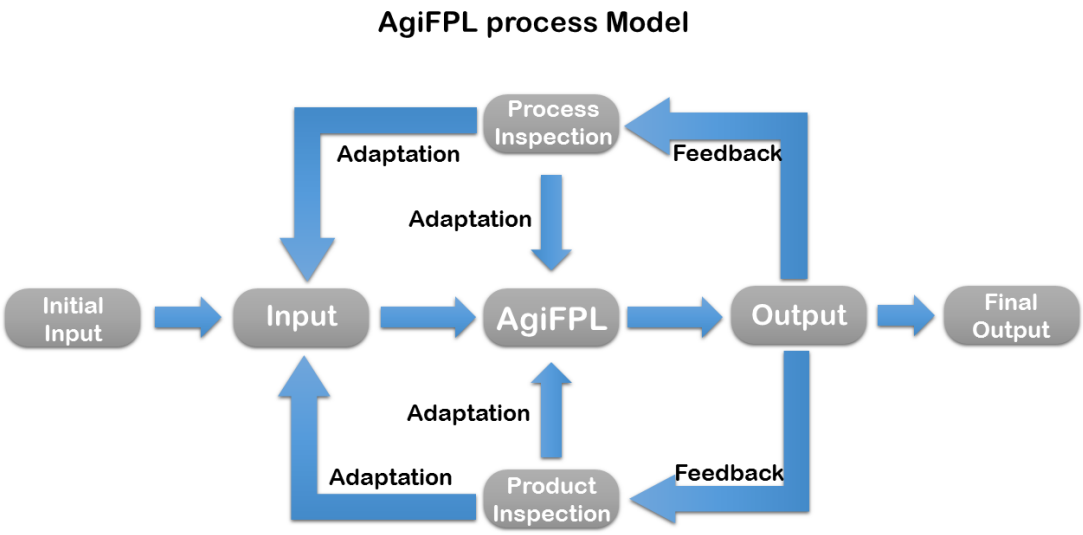

Figure 5 AgiFPL process model

7. ILLUSTRATION

In order to validate our proposed APLE method, we suggest introducing briefly a case study based on the Odoo open source ERP platform (Odoo Inc., 2017). In fact, Odoo offers an *All-in-one* ERP management software which fits small and medium companies and our APLE method (AgiFPL) could be applied within Odoo.

Due to lack of space, we only present a simple example concerning "NST Shipping" which is a maritime transporter working with Odoo to implement their cross-functional business processes. In the requested ERP, several modules and applications are needed to support NST activities, e.g. "Invoicing Application" and "eTracking service".

Odoo has an initial huge "Common Assets Warehouse". Among the reusable artefacts of Odoo we found the "Invoicing Module". Therefore, several modules of NST ERP would be found in Odoo's common assets warehouse.

Once NST signs the contract of the NST ERP implementation with Odoo, the Odoo "Line Team" will proceed following the AE phase of AgiFPL. Based on the results of the "ARE step" and "NST App Backlog" the "Line Team" will recognize that the requested "eTracking service" does not exist in the "Common assets Warehouse" and there are no similar artefacts among the reusable ones. Therefore, the "Line Team" will start the process of developing this service. In other words, the team will transform the related requirement model (See Figure 6) into a feature model and then into USs and organize these USs in the App Backlog. Then they will follow the process until delivering the final version of the NST ERP.

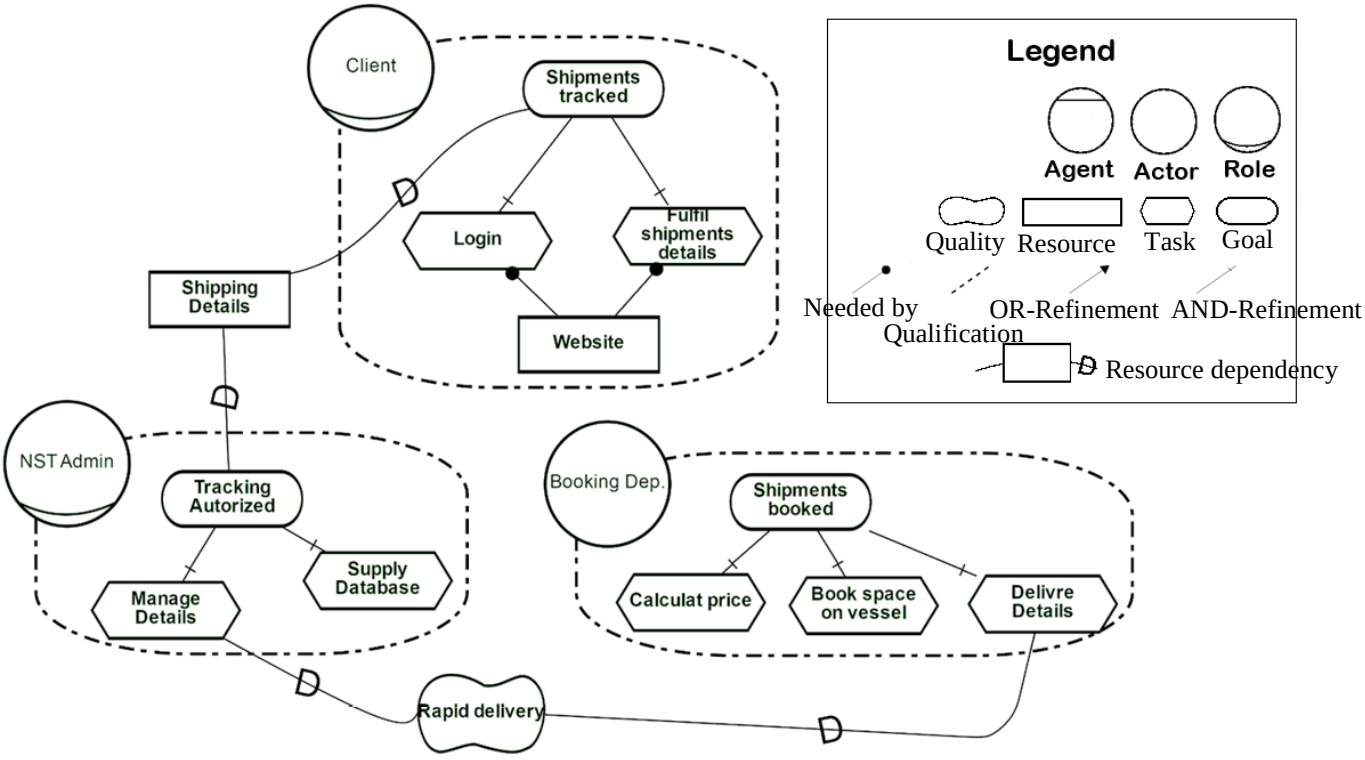

Figure 6 Goal Model for *eTracking* service

Following AgiFPL method, this "Goal Model" of the eTracking service will be transformed into a feature model (FM). Figure 7 shows the FM for the eTracking service" module.

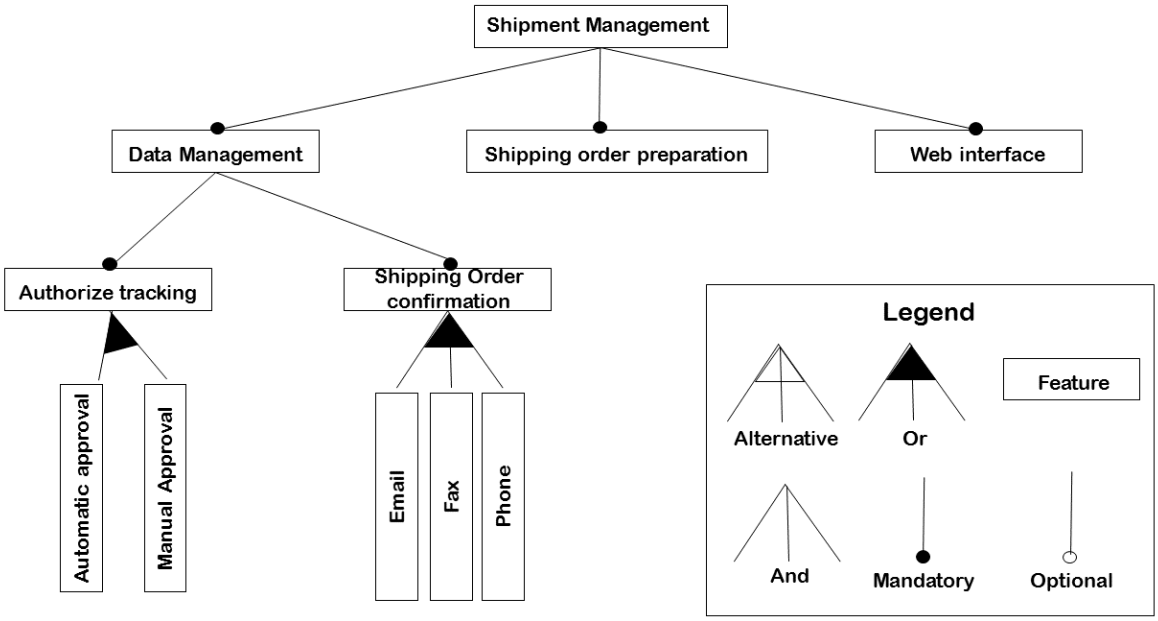

Figure 7 Feature Model for *eTracking Service*

As said above, this FM for “eTracking Service” will be transformed to USs in order to define the “App Backlog”. According to the “Planning” the development of this App will take a “Sprint” of 2 weeks. Table 1 shows the “Sprint” plan:

Table 1 Sprint plan

Week 1	One of the first days	Meeting with NST stakeholder Sprint post-mortem
	1 Monday	Technical meeting
		Development sprint kickoff
	2 Tuesday	Development
	3 Wednesday	
	4 Thursday	
	5 Friday	
Week 2	1 Monday	
	2 Tuesday	
	3 Wednesday	Internal demo
		Sprint “planning 2” and review meeting
	4 Thursday	Fix change requests
	5 Friday	Retrospective
		Project Status Meeting

At the end of the sprint the final version of the “eTracking service App” will be integrated to the NST ERP. In addition, all reusable artefacts generated from this step will be stored on the “Common assets warehouse”.

8. CONCLUSION

As mentioned in this paper, our contribution is to propose a new proper APLE approach that addresses the deficiencies identified in current APLE methods, while making use of their benefits. The proposed APLE approach is called AgiFPL (Agile Framework for managing evolving PL).

In order to design this APLE approach, we have reviewed the most relevant APLE approaches from the literature. In addition, we present as an illustration part of a case study in order to show how we intend to validate our approach.

Further work is undergoing to complete AgiFPL approach. For instance, we are working on adopting and developing metrics for performance. We are also developing tools that will facilitate the implementation of AgiFPL. Also, we try to simplify the RE processes by adopting adequate frameworks that fit the agility of our proposal.

9. REFERENCES

1. Appel, S., Batory, D., Kästner, C., and Saake, G. (2013). Feature-oriented Software Product Lines: Concepts and Implementation. Berlin: Springer-Verlag.
2. Asadi, M., and Ramsin, R. (2008). MDA-Based Methodologies: An Analytical Survey. In: Proceedings of the 4th European Conference ECMDA-FA. Berlin: Springer-Verlag, pp. 419–431.
3. Asadi, M., Bagheri, E., Mohabbati, B., and Gašević, D. (2012). Requirements Engineering In Feature Oriented Software Product Lines: An Initial Analytical Study. In: Proceedings of the 16th SPLC '12, 2, pp. 36-44.
4. Balbino, M., de Almeida, E. S., and de Lemos Meira, S. R. (2011). An Agile Scoping Process for Software Product Lines. In: Proceedings of International Conference SEKE2013, Miami (USA) pp. 717–722.
5. Carbon, R., Lindvall, M., Muthig, D., and Costa, P. (2006). Integrating Product Line Engineering and Agile Methods: Flexible Design Up-front vs. Incremental Design. In: Proceedings of International Workshop on Agile Product Line Engineering collocated with International Software Product Line Conference, pp. 1–8.
6. Cohn, M. (2004). User Stories Applied for Agile Software Development. 1st ed. Boston: Pearson Education In.
7. Cohn, M. (2013). Succeeding with Agile: Software Development Using Scrum. 1st ed. Upper Saddle River, NJ: Addison-Wesley Professional.
8. Clements, P., and Northrop, L. (2001). Software Product Lines: Practices and Patterns. Boston: Addison-Wesley Longman Publishing Co.
9. da Silva, I. F., Neto, P., O'Leary, P., de Almeida, E., and de Lemos Meira, S.R. (2011). Agile Software product lines: a systematic mapping study. Software: Practice and Experience, 41(8) 2011, pp. 899–920.
10. da Silva, I. F. (2012). An Agile Approach for Software Product Lines Scoping. In: Proceedings of International Software Product Line Conference, SPLC2012, Salvador, Brazil pp. 225–228.
11. de Souza, D. S., and Vilain, P. (2013). Selecting Agile Practices for Developing Software Product Lines. In: Proceedings of International Conference SEKE2013, Boston (USA) pp. 220–225.
12. Díaz, J., Pérez, J., Alarcón, P. P., and Garbajosa, J. (2011). Agile product line engineering—A systematic literature review. Software: Practice and Experience, 41(8), pp. 921–941.
13. Díaz Fernández, J., Pérez Benedí, J., Yagüe Panadero, A., and Garbajosa Sopeña, J. (2011). Tailoring the Scrum Development Process to Address Agile Product Line Engineering. In: Proceedings of JISBD2011, Spain: Coruña.

14. Farahani, F. F., and Ramsin, R. (2014). Methodologies for Agile Product Line Engineering: A Survey and Evaluation. In: Proceedings of the 13th International Conference SoMeT_14, Amsterdam: IOS Press BV, pp. 545-564.
15. Ghanam, Y. and Maurer, F. (2008). An Iterative Model for Agile Product Line Engineering. In: Proceedings of the 12th SPLC2008, Ireland: Limerick, pp. 377–384.
16. Ghanam, Y., and Maurer, F. (2009). Extreme Product Line Engineering: Managing Variability and Traceability via Executable Specifications. In: Proceedings of Agile Conference, AGILE '09, IEEE Computer Society, USA: Washington DC, pp. 41–48.
17. Ghanam, Y., Andreychuk, D., and Maurer, F. (2010). Reactive Variability Management in Agile Software Development. In: Proceedings of Agile Conference, AGILE '10, Portugal: Guimarães, pp. 27–34.
18. Hanssen, G. K. (2011). Agile software product line engineering: enabling factors. Software: Practice and Experience, 41(8), pp. 883–897.
19. Hanssen, G. K., and Fægri, T. E. (2008). Process fusion: An industrial case study on agile software product line engineering. Journal of Systems and Software, 81(6), pp. 843–854.
20. Hesari, S. Mashayekhi, H. and Ramsin, R. (2010). Towards a General Framework for Evaluating Software Development Methodologies, in Proceedings of Computer Software and Applications Conference, pp. 208–217.
21. KäKölä, T., and Dueñas, J. C. (2006). Software Product Lines: Research Issues in Engineering and Management. Berlin: Springer-Verlag.
22. Kang, K. C., Sugumaran, V., and Park, S. (2010). Applied Software Product Line Engineering. Boston: Auerbach Publications.
23. Kenneth, S. R. (2013). Essential Scrum: A Practical Guide to the Most Popular Agile Process. Upper Saddle River, NJ: Addison-Wesley Professional.
24. Ladas, C. (2009). Scrumban - Essays on Kanban Systems for Lean Software Development. 1st ed. Seattle, WA: Modus Cooperandi Press.
25. Larman, C. (2003). Agile and Iterative Development—A Manager's Guide (Agile Software Development Series), Cockburn A, Highsmith JJ (eds). New Jersey: Addison-Wesley.
26. Mohan, K., Ramesh, B., and Sugumaran, V. (2010). Integrating Software Product Line Engineering and Agile Development. IEEE Computer Society, 27, pp. 48-55.
27. Noor, M. A., Rabiser, R., and Grünbacher, P. (2008). Agile product line planning: A collaborative approach and a case study. Journal of Systems and Software, 81(6), pp. 868–882.
28. O'Leary, P., McCaffery, F., Thiel, S., and Richardson, I. (2010). An Agile process model for product derivation in software product line engineering. Journal of Software: Evolution and Process, 24(1) 2012, pp. 561-571.
29. Pohl, K., Böckle, G., and van der Linden, F.J. (2005). Software Product Line Engineering: Foundations, Principles and Techniques. Berlin: Springer-Verlag, pp. 3-88.
30. Ramsin, R., and Paige, R. F. (2008). Process-centered Review of Object Oriented Software Development Methodologies. ACM Computing Surveys, 40(1), pp. 3:1–3:89.
31. Reddy, A. (2016). The Scrumban [r]evolution: getting the most out of Agile, Scrum, and Lean Kanban. 1st ed. New York: Addison-Wesley Professional.

32. Sommerville, I. (2011). Software Engineering. 9th ed. Boston: Addison-Wesley.
33. Shore, J., Warden, S. (2007). The Art of Agile Development. [CA] Sebastopol: O'Reilly Media, Inc.
34. van der Linden, F., Schmid, K., and Rommes, E. (2007). Software Product lines in Action: The Best Industrial Practice in Product Line Engineering. Berlin: Springer-Verlag.
35. Wang, X. (2005). Towards an Agile Method for Building Software Product Lines, M.Sc. Thesis, University of York, UK.
36. Wautelet, Y., and Kolp, M. (2016). Business and model-driven development of BDI multi-agent systems. In: Neurocomputing Journal, 182(1), pp. 304-321.
37. Do, T.T., Kolp, M., Faulkner, S., Pirotte, A. (2004): Agent-oriented design patterns. In: ICEIS2004, Proceedings of the 6th International Conference on Enterprise Information Systems, Porto, Portugal, April14-17, pp.48–53.
38. Giorgini, P., Kolp, M., Mylopoulos, J. (2002): Multi-agent and software architectures: A comparative case study. In: *Agent-Oriented Software Engineering III, Third International Workshop*, AOSE2002, Bologna, Italy, July15, Revised Papers and Invited Contributions. pp. 101–112.
39. Kolp, M., Giorgini, P., and Mylopoulos, J. (2002). Organizational multi-agent architectures: a mobile robot example. In First I. *Joint Conf. on Autonomous Agents & Multiagent Systems, AAMAS 2002*, July 15-19, 2002, Bologna, Italy, pages 94–95.
40. Kolp, M. and Mylopoulos, J. (2001). Software architectures as organizational structures. *In Proc. ASERC Workshop on The Role of Software Architectures in the Construction, Evolution, and Reuse of Software Systems*, Edmonton, Canada.
41. Mouratidis, H., Kolp, M., Faulkner, S., Giorgini, P. (2005): A secure architectural description language for agent systems. In:4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, Utrecht, The Netherlands.pp.578–585.
42. Mylopoulos, J., Kolp, M., and Giorgini, P. (2002). Agent oriented software development. In Methods and Applications of Artificial Intelligence, *Second Hellenic Conference on Artificial Intelligence, SETN'02*, Tessaloniki, Greece, April 11-12, pages 3–17. Springer Berlin Heidelberg.