GMAP: A Generic Methodology for Agile Product Line Engineering

Farima Farmahini Farahani, Raman Ramsin
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
e-mail: farimafarahani@ce.sharif.edu, ramsin@sharif.edu

***Abstract*—Agile Product Line Engineering (APLE) is a relatively novel approach that has emerged as the result of the combination of two successful software development approaches: Software Product Line Engineering and Agile Software Development. The main goal of this combined approach is to cover the weaknesses of each of these two approaches while maximizing the advantages of both. We propose the Generic Methodology for Agile Product Line Engineering (GMAP), which can be instantiated to produce a bespoke, concrete methodology for any specific project situation. GMAP is generic in that its process covers the main activities of existing APLE methodologies, while refraining from enforcing any specific and concrete method or technique for performing the activities. GMAP has been produced by studying existing APLE methodologies, identifying their strengths and weaknesses, abstracting them into a high-level framework, and finally instantiating this abstract framework so as to address the shortcomings of existing methodologies.**

***Keywords—Software Development Methodology; Software Product Line; Agile Method; Agile Product Line Engineering*.**

I. INTRODUCTION

Product Line Engineering (PLE) and *Agile Software Development* are two successful approaches in the software industry. Both approaches focus on developing high-quality software systems, reducing development costs, managing changes in requirements, and reducing time to market. These common goals have motivated researchers to investigate ways for merging them. This new combined approach, called *Agile Product Line Engineering (APLE)* [1], would help us use the positive features of the individual approaches while maximizing their benefits. Another potential advantage is synergy: each approach can cover the other's weaknesses.

Several methods have so far been introduced in the APLE context. From among these methods, those that have proposed a distinct *process* for this combined approach can be referred to as APLE *methodologies*. We have previously studied and analyzed existing APLE methodologies by using a criteria-based approach [2]. The results of this evaluation showed that despite the benefits that they provide, they are all afflicted with certain problems; for instance, none of them has prescribed a full-lifecycle approach that explains the details of the activities, work-products, and roles. Hence, developing a new APLE methodology that addresses the deficiencies of existing methodologies, while preserving their positive features, is of great potential value. We therefore propose the Generic Methodology for Agile PLE (GMAP) as a high-level full-lifecycle APLE methodology that provides these features and can be instantiated to yield concrete APLE methodologies for different project situations. To this aim, we first evaluated existing methodologies using a criteria-based approach to identify their positive and negative traits. The evaluation criteria and the results of evaluation have been extensively discussed in [2]. We have used the criteria defined in [2] as the requirements for constructing GMAP; this ensures that GMAP possesses the main characteristics of PLE and Agile Development, makes use of the positive features of previous APLE methodologies, and addresses their weaknesses.

GMAP was developed in two steps: We first produced a high-level APLE process framework through applying abstraction to existing methodologies; GMAP was then defined by instantiating this abstract framework and improving the resulting instance to satisfy the requirements of the target methodology. GMAP satisfies the high-level requirements, but is kept independent of specific techniques and practices so that it can be instantiated based on the finer-grained requirements of a specific project. GMAP was evaluated in two ways: 1) by instantiation to a concrete methodology in order to demonstrate that it has the potential to be instantiated into a concrete and applicable methodology, and 2) by applying a subset of the criteria presented in [2] in order to show that GMAP can indeed be considered an improvement to the status quo; the reason for using a subset of the criteria is that some of the criteria are not applicable due to the abstractness of GMAP. The evaluation results show that GMAP does indeed address the weaknesses of existing methodologies; for example, the need for a full-lifecycle APLE process, providing specifications for activities, work-products, and roles, has been adequately addressed in GMAP.

The rest of this paper is structured as follows: Section II discusses the related research; Section III presents the proposed abstract framework for APLE methods; Section IV describes GMAP; Section V presents the evaluation results; and Section VI discusses the conclusions and suggests ways for furthering this research.

II. RELATED RESEARCH

Prominent APLE methodologies and their important features are depicted in Table I; this table has been adapted from our previous research, reported in [2], which presents an extensive review of these methodologies.

 ISBN: 978-1-61208-590-6

TABLE I. PROMINENT APLE METHODOLOGIES (ADAPTED FROM [2])

Feature / Methodology	Brief introduction	Year	Basis (Agile or PLE)	Reuse approach	PLE process coverage (DE, AE)
CDD [4]	Uses FDD [5] to combine PLE and agility.	2005	Agile	N/A	DE (Partially)
de Souza & Vilain [6]	Merging the generic PLE process with the Framework of Agile Practices.	2013	PLE	Proactive, Reactive	DE (Partially), AE
RiPLE-SC [7]	An agile process for PL Scoping in RiPLE methodology.	2011	PLE	N/A	DE (Partially)
Diaz et al. [3]	Utilizes Scrum [5] to define an APLE method.	2011	Agile	Reflexive	DE (Partially), AE
EPLSP [8]	A full-lifecycle methodology that utilizes AUP [9] for product development; thus, agility is limited to this particular activity.	2011	PLE	Reactive	DE, AE
A-Pro-PD [10]	A generic agile framework for product derivation.	2012	PLE	Proactive, Reactive	AE
Ghanam & Maurer 2008 [11]	Aims at agile organizations building several similar systems in a domain. Core assets are derived from the products in a bottom-up fashion.	2008	Agile	Reactive	DE (Partially), AE (Partially)
Ghanam et al. [12]	An agile approach for variability management in which variability analysis is only performed when a new requirement arises.	2010	Agile	Reactive	DE (Partially)
Ghanam & Maurer 2009 [13]	An acceptance-test-based approach for product derivation in PLE; core assets are retrieved (from a repository) based on acceptance tests.	2009	Agile	Reactive	AE
da Silva [14]	An agile process for PL scoping.	2012	PLE	N/A	DE (Partially)
Carbon et al. [15]	The result of incorporating agile practices into PULSE's product-instantiation.	2006	PLE	Reactive	AE
Noor et al. [16]	An agile method for PL scoping that utilizes Collaboration Engineering patterns to promote collaboration among stakeholders.	2008	PLE	N/A	DE (Partially)
SPLICE [17]	Incorporates Scrum [5] practices with core activities of PLE.	2014	PLE	Reactive	DE, AE (Partially)

The review reported in [2] shows that none of the methodologies possess all the features expected in an APLE methodology, including: full coverage of the PLE lifecycle, definition of work-units, roles, and work-products, attention to umbrella activities, management of expected/unexpected changes, configurability, support for learning, active user involvement, and team management. Three reuse approaches have been observed in these methodologies: Proactive (predicting and building the core assets at the beginning of the development process), Reactive (extracting the core assets from previously built products), and Reflexive (predicting the core assets at the beginning of each iteration) [3].

III. PROPOSED ABSTRACT FRAMEWORK FOR APLE METHODOLOGIES

The first step in building the target methodology is to produce an abstract framework for APLE methodologies, which is the result of applying abstraction to the activities prescribed in the methodologies reviewed (listed in Table I); this framework will be introduced in this section.

A. Description

Figure 1 shows the proposed framework. As it covers the activities of the reviewed methodologies in an abstract manner, these methodologies can be regarded as its instances. The white block arrows between DE and AE indicate that any transition between the internal stages of these sub-processes is possible; however, the transitions allowed in existing methodologies are shown with ordinary black arrows. The liberal attitude of the framework towards transitions promotes abstractness and allows the framework to be instantiated into any desired APLE methodology. The white arrow from DE to AE denotes the Proactive approach of reuse, the white arrow from AE to DE denotes the Reactive approach, and the combination of these arrows denotes the Reflexive approach. The activities that belong to some (but not all) of the reviewed methodologies will be referred to as "non-common".

1) Domain Engineering (DE) Sub-process

DE consists of *Scoping* and *Core Assets Development*.

a) Scoping

The PL's scope is determined in the following stages:

- Pre-Scoping: Business goals are identified; non-common activities are: understanding the operational and organizational context of the organization, analyzing stakeholders and target markets, and building a business case.
- Domains Selection: PL domains are selected from among the candidate domains.
- Products and Requirements Selection: The domains' requirements and products are identified.
- Prioritization: Requirements and products are prioritized and selected.

b) Core Assets Development (CAD)

Core assets are built through the following stages:

- Requirements Analysis: The requirements of the PL products are defined in a more fine-grained form, with the commonalities and variabilities specified.
- Core Assets Design: Components and PL architecture are designed. Non-common activities are: Detailed design, and documentation of design decisions.
- Planning: Implementation units are prioritized and assigned to iterations.
- Core Assets Implementation: Implementation units are built, and the required tests are developed.
- Core Assets Validation and Incorporation in the Repository: Implemented units are integrated with other parts, and are incorporated in the repository.

2) Application Engineering (AE) Sub-process

PL products are built and deployed in two phases: *Product Development* and *Transition.*

a) Product Development

This phase is the pivotal part of AE. Its stages are:

 ISBN: 978-1-61208-590-6

- Requirements Definition: The product requirements document is produced, and the core assets are elicited.
- Planning: Implementation units are assigned to iterations.
- Design and Implementation: The PL architecture is instantiated and the product architecture is developed; detailed design is then performed for the product-specific parts (as a non-common activity). Next, product-specific parts are developed and integrated with instantiated core assets to form the final product.
- Increment Validation: The implemented increment is validated against the iteration requirements.

b) Transition

System Validation and Installation is performed. As a non-common activity, training material is produced and a short document of the system is developed for the user.

3) Maintenance Sub-process

In the *Support* phase, bug fixes and new requirements are supplied to the AE team (and to the DE team, if necessary).

B. Realization of the framework in APLE methodologies

Table II shows how the framework's constituent activities correspond to the activities of existing APLE methodologies, thus validating the proposed framework as to its coverage of existing APLE methodologies.

IV. Proposed Generic APLE Methodology (GMAP)

As discussed in [17], the criteria for evaluating methodologies in a given context can serve as the requirements for constructing a target methodology in that context; we have therefore used the criteria defined in [2] as the requirements for constructing GMAP. GMAP is an instance of the proposed APLE framework, and its activities and tasks are abstractions of the activities and tasks prescribed in existing APLE methodologies, composed so that the requirements of the target APLE methodology are satisfied; also, certain activities have been added from existing Agile and PLE methodologies. As GMAP is generic and abstract, it only provides general guidelines and does not enforce any concrete methods or techniques. Thus, it has a high degree of configurability and should be instantiated prior to application; typically, the instantiation process includes the following activities: Decision as to optional activities; addition of project-specific tasks; removing, merging, or decomposing the tasks; determination of concrete methods, practices, and guidelines for performing the tasks; selection from among the methods available for performing the tasks; and changing the roles involved in an activity or task. The process of GMAP and its roles are presented throughout the rest of this section.

A. Roles

GMAP roles have been determined through integrating the roles typically found in agile methodologies with the generic PLE roles defined in [18] and the roles observed in the APLE methodologies reviewed in [2]. The roles are explained below:

- *Senior Manager:* Responsibilities include managing the APLE project, and providing expertise on organizational/business goals and the market.
- *Product Manager:* Responsibilities are: managing PL products and planning for the development of current and future systems, providing expertise on business goals and the PL's target market, and maintaining the PL Scope Document [18].
- *Core Assets Manager:* Duties include performing maintenance and configuration management on the core assets, and helping with their extraction [18].
- *Senior Developer:* This role is performed by experienced developers familiar with the organization's products. Responsibilities include conducting project management, leading teams, and carrying out analysis and design activities during CAD and AE; scoping and code development will also be included if necessary. Responsibilities are equivalent to the collective responsibilities of Domain/Application (D/A) Requirements Engineer, D/A Architect, and D/A Developer roles of [18].
- *Developer:* Responsibilities are working in CAD and AE teams, and performing analysis, design, implementation, and test under the supervision of Senior Developers. Responsibilities are equivalent to the combined responsibilities of D/A Requirements Engineer, D/A Architect, D/A Developer, and D/A Tester of [18]. We recommend that the people in charge of this role be moved between CAD and AE so that their knowledge is shared (akin to the "Move People Around" practice of XP [5]). A number of Developers are also involved in Scoping.
- *Customer:* This role is performed by representatives of the customer organizations who know the system's requirements and act as domain experts.
- *Support Team Member:* This role performs support and maintenance activities on instances of the PL.

B. Process

The process of GMAP is shown in Figure 2. It consists of three sub-processes: DE (consisting of Scoping and CAD), AE, and Maintenance. AE and the two internal phases of DE are run in an iterative-incremental manner. This methodology is applicable under two scenarios: Scenario-1 aims at organizations that have previously developed a number of similar systems. In this case, Scoping is first performed, followed by CAD and AE; after the first run of the CAD phase (so that the CAD team is formed and the reference architecture is built), CAD and AE can be run in tandem. Scenario-2 is aimed at organizations that have not built any similar systems, but are planning to build a PL while the products are being developed. In this case, the organization first develops a predefined number of systems (at least two [11]) using the AE sub-process of GMAP; Scoping is then performed for these systems, followed by CAD and AE. The two scenarios show that the methodology can cover the proactive approach of reuse as well as the reactive approach based on the target organization's needs. In both scenarios, if a new product is to be built in an available domain, core assets are retrieved from the product by requesting for PL extension, and the PL Scope is updated accordingly. If a new product in a new domain is to be built, scoping should precede CAD.

 ISBN: 978-1-61208-590-6

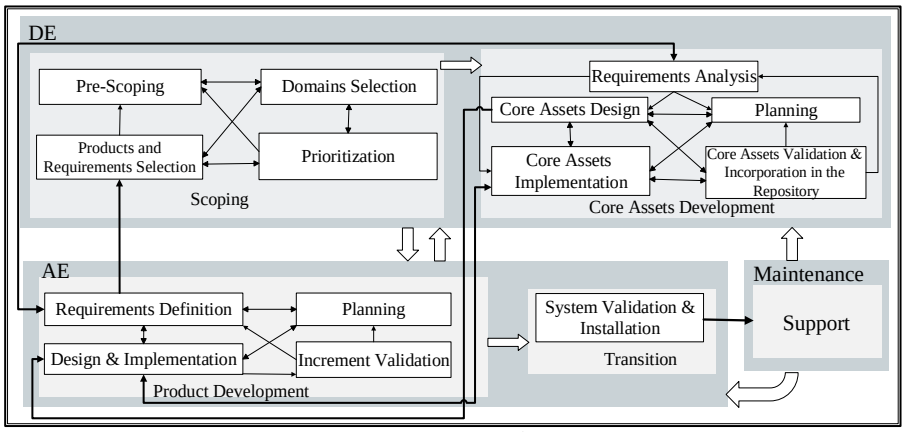

Figure 1. Proposed Abstract Framework for APLE Methodologies

TABLE II. REALIZATION OF PROPOSED FRAMEWORK IN REVIEWED APLE METHODOLOGIES

Stage in Framework			**Corresponding phases in APLE methodologies**
Domain Engineering	***Scoping***	Pre-Scoping	*Pre-scoping* in RiPLE-SC [7]; *Define pre-scoping* (Partially) in da Silva [14], *Select business goals and marketing strategies* in SPLICE [17].
		Domains Selection	*Identify and agree on relevant domains* in Noor et al. [16]; *Define pre-scoping* (Partially) in da Silva [14]; *Domain scoping* in RiPLE-SC [7].
		Products & Requirements Selection	*Define features for each domain*, *Discuss, analyze, and agree on products*, *Define products in terms of features* in Noor et al. [16]; *Product scoping* in RiPLE-SC [7]; *Define features*, *Define pre-scoping* (partially) in da Silva [14], *Identify products*, *Identify major features*, *Build initial product map* in SPLICE [17].
		Prioritization	*Release scope* in da Silva [14]; *Prioritize product map* in Noor et al. [16]; *Assets scoping* in RiPLE-SC [7], *Prioritize major features* in SPLICE [17].
	Core Assets Development	Requirements Analysis	*Develop an overall model* (Partially), *Build a features list* in CDD [4]; *Domain analysis* in de Souza & Vilain [6]; *Pregame* (Partially) in Diaz et al. [3]; *Evaluation and extraction* (Partially) in Ghanam & Maurer 2008 [11]; *Eliciting new requirements* (Partially), *Variability analysis*, *Updating the variability profile* in Ghanam et al [12]; *Analyze commonality and variability* in da Silva [14]; *Core assets development* (Partially) in EPLSP [8], *Sub-features definition*, *Commonality and variability analysis* in SPLICE [17].
		Core Assets Design	*Develop an overall model* (Partially), *Design SPL architecture*, *Build a components list*, *Design by components* in CDD [4]; *Domain design*, *Develop system increment* (DE) (Partially) in de Souza & Vilain [6]; *Architecture evolution* in Ghanam & Maurer 2008 [11]; *Sprint-domain engineering* (Partially) in Diaz et al. [3]; *Refactoring the architecture* in Ghanam et al [12]; *Core assets development* (Partially) in EPLSP [8].
		Planning	*Plan by components* in CDD [4]; *Iteration definition* in de Souza & Vilain [6]; *SPL release definition* (Partially); *Sprint planning* (Partially) in Diaz et al. [3]; *Select features for implementation* (Partially) in da Silva [14]; *Eliciting new requirements* (Partially) in Ghanam et al [12], *Release planning*, *Sprint planning* in SPLICE [17].
		Core Assets Implementation	*Build by components* (Partially) in CDD [4]; *Develop system increment* (DE) (Partially) in de Souza & Vilain [6]; *Refactoring* in Ghanam & Maurer 2008 [11]; *Select features for implementation* (Partially) in da Silva [14]; *Sprint-domain engineering* (Partially) in Diaz et al. [3]; *Realizing the new requirements* in Ghanam et al [12]; *Core assets development* (Partially) in EPLSP [8], *Sub-features implementation*, *Sub-features testing* (Partially) in SPLICE [17].
		Core Assets Validation & Incorporation in the Repository	*Running the tests* in Ghanam et al [12]; *Build by components* (Partially) in CDD [4]; *Develop system increment* (DE) (Partially), *Validate increment* in de Souza & Vilain [6]; *Managing core assets* in Ghanam & Maurer 2008 [11]; *Review and retrospective* (Partially) in Diaz et al. [3]; *Core assets development* (Partially) in EPLSP [8], *Sub-features testing* (Partially), *Sprint review and retrospective* in SPLICE [17].
Application Engineering	***Product Development***	Requirements Definition	*Definition of requirements* in de Souza & Vilain [6]; *Preparing for derivation* (Partially) in A-Pro-PD [10]; *Select acceptance tests*, *Execute acceptance tests*, *Extract code* (Partially) in Ghanam & Maurer 2009 [13]; *Plan for a product line instance*, *Instantiate and validate product line model* in Carbon et al. [15]; *Pregame* (Partially), *SPL release definition* (Partially) in Diaz et al. [3]; *Evaluation and extraction* (Partially) in Ghanam & Maurer 2008 [11]; *Product Development* (Partially) in EPLSP [8].
		Planning	*Assign requirements to iterations* in de Souza & Vilain [6]; *Preparing for derivation* (Partially) in A-Pro-PD [10]; *SPL release definition* (Partially), *Sprint planning* (Partially) in Diaz et al. [3].
		Design and Implementation	*Instantiate & validate reference architecture*, *Construct product* in Carbon et al. [15]; *Develop system increment* (AE) in de Souza & Vilain [6]; *Product configuration*, *Product development & testing* (Partially) in A-Pro-PD [10]; *Core asset incorporation* in Ghanam & Maurer 2008 [11]; *Extract code* (Partially), *Verify & build* (Partially) in Ghanam & Maurer 2009 [13]; *Sprint-application engineering* in Diaz et al. [3]; *Product Development* (Partially) in EPLSP [8], *Products derivation* (Partially) in SPLICE [17].
		Increment Validation	*Validate increment*, *Integrate increment* in de Souza & Vilain [6]; *Review and retrospective* (Partially) in Diaz et al. [3]; *Verify and build* (Partially) in Ghanam & Maurer 2009 [13]; *Product development and testing* (Partially) in A-Pro-PD [10].
	Transition	System Validation & Installation	*Validate system* in de Souza & Vilain [6]; *Product development and testing* (Partially) in A-Pro-PD [10]; *Deliver system* in Carbon et al. [15].
Maintenance	***Support***		*Product release* in EPLSP [8].

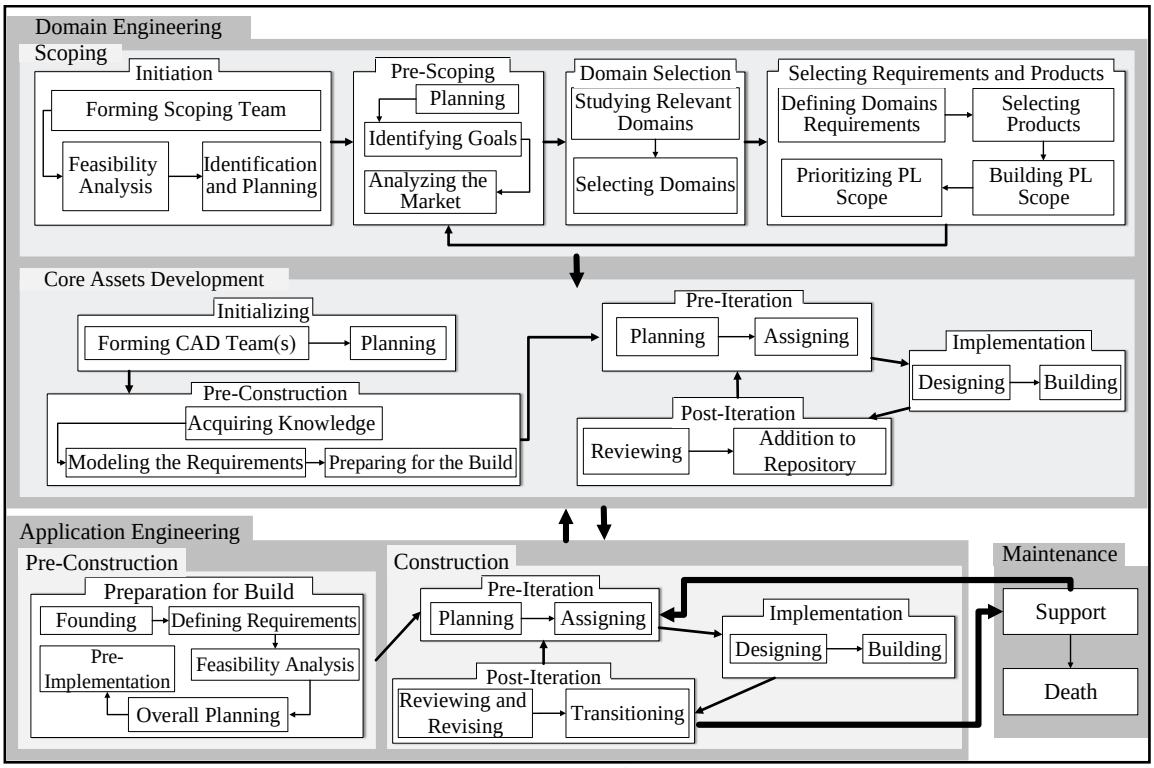

Figure 2. Process of GMAP

GMAP is described throughout the rest of this section by using a *process-centered* approach [5]: the focus is on describing the high-level *phases*, intermediate-level *stages*, and bottom-level *tasks* of the methodology; the roles involved and the work-products produced are seen as secondary to these process constituents.

1) Domain Engineering (DE) Sub-process

DE aims at building the PL infrastructure. AE team members are the main customers of DE, as they use the core assets developed in DE.

a) Scoping

The PL scope is defined in this phase. It continues in an iterative-incremental manner until the PL scope becomes stable enough for the core assets to be built based on it. The PL scope is then completed while new PL products are being developed. The constituent stages are explained below.

1) Initiation

The preliminary activities for identifying the PL scope are performed in three sub-stages:

Forming Scoping Team: The Scoping Team is formed by analyzing the experiences and skills required; team members include the product manager, a number of senior developers and developers, and selected customer representatives [7].

Feasibility Analysis: This stage is performed by the senior manager, product manager and senior developers of the scoping team. The results of analyzing the risks and constraints and estimating the required resources are recorded in the PL Vision. Finally, feasibility study is performed based on the PL Vision, and a Go/No-Go decision is made.

Identification and Planning: Organizational factors are identified by the senior manager, product manager, and senior developers of the scoping team; the organization's structure and its processes are explored and documented in the PL Vision. The product manager and the senior developers then determine an overall plan for the scoping phase.

2) Pre-Scoping

This stage, performed by the senior manager, product manager, and senior developers of the scoping team, consists of three sub-stages:

Planning: The iteration plan is elicited based on the overall plan.

Identifying Goals: Business and organizational goals are recorded in the PL Vision.

Analyzing the Market: The PL Vision is completed by studying the markets related to the candidate domains (as to their characteristics and success factors).

3) Domain Selection

The scoping team determines the target domains of the PL in two sub-stages, which are typically performed in tandem:

Studying Relevant Domains: The relevant domains are explored by the team members.

Selecting Domains: Domains are analyzed based on certain parameters that verify the potential of each domain for inclusion in the PL; domains are then selected based on this analysis and the parameters important to the organization.

4) Selecting Requirements and Products

The scoping team selects the PL products and requirements; sub-stages are as follows:

Defining Domains Requirements: The requirements of the selected domains are elicited. Requirements are then reviewed in order to resolve redundancies and ambiguities.

Selecting Products: Candidate products are defined for each domain based on domain requirements. Candidate products are then prioritized based on the parameters defined in the instantiated methodology, and PL products are selected.

Building PL Scope: The PL Scope is completed by relating products with requirements. Then, the customers review and validate the PL scope.

Prioritizing PL Scope: Reusable parts that will be implemented in the CAD phase are identified. We recommend two methods for this purpose: 1) The method used in [7] and [19], in which the goals are operationalized with the aim of defining certain metrics (the GQM method is typically used); prioritization is then conducted based on the derived metrics; and 2) The method used in [14] and [16], in which each stakeholder prioritizes the products and their requirements; stakeholders then discuss the priorities, reach a consensus, and select the core asset requirements.

b) Core Assets Development (CAD)

Reusable core assets are developed based on the results of scoping. Constituent stages include the following:

1) Initializing

This stage sets the stage for building the core assets. Sub-stages are as follows:

Forming CAD Team(s): Each CAD team is led by a senior developer, and is made up of several experienced developers. Developers are selected based on the knowledge, skills, and experience levels required.

Planning: Requirements prioritization is first conducted with the help of the product manager. Next, the overall plan is elicited and a subset of the requirements is selected for the next release. Also, a date for the next release and a duration for CAD iterations are determined.

2) Pre-Construction

The requirements of the next release are defined in a more detailed fashion, and the PL architecture is designed. Sub-stages are as follows:

Acquiring Knowledge (Optional): If the team requires more information on the requirements, knowledge acquisition is performed with the help of the product manager.

Modeling the Requirements: Requirements are modeled by specifying their commonalities and variabilities; requirements can be modeled in several formats: use-cases [20], acceptance tests [11], feature diagrams [21][22], features (as in the FDD methodology) [4][5], and a textual feature-based format [12]. The dependencies among the requirements are also identified, which can affect the selection of the requirements [22] and their implementation sequence.

Preparing for the Build: The architecture is designed, and architecture-level variabilities are specified; the architecture is then evaluated [4]. A list of implementation units is then produced, which can be based on the requirements [3], components [4][6], or any other relevant concept.

3) Pre-Iteration

An iteration plan is developed in the following sub-stages:

Planning: Prioritization is applied to the implementation units. Iteration planning is then performed, and a number of units are selected for the current iteration.

Assigning: If more than one team is involved, assigning to teams is performed. Assigning to developers is then conducted inside each team; this can be done in two ways: 1) the senior developer in each team assigns the units to the developers [4]; or 2) the developers choose what they intend to implement (common in self-organizing teams [23][24]).

4) Implementation

Detailed design and implementation is performed on the iteration's implementation units, in the following sub-stages:

Designing: Documents are studied and domain experts are interviewed in order to enrich the team's knowledge of the implementation units; these tasks are executed in tandem with other tasks of this stage, and may result in changes to the requirements model. Detailed design is then conducted, resulting in the design model. Finally, model notes [4] are added to document the design alternatives, and the reasons behind the design decisions.

Building: Test design is first conducted to produce test-cases for the current iteration's implementation units. Coding and refactoring are then performed [6]. Testing is performed continuously throughout this stage. Code inspection is the final task, which can be done in two ways: 1) the senior developer of each team inspects the code [4], or 2) the developers inspect one another's code [6].

5) Post-Iteration

The activities required for concluding the iteration are performed. Sub-stages are as follows:

Reviewing: Testing is performed with the cooperation of AE team members, and acceptance tests are run on the implemented units. A review meeting is then held to conduct regular review activities. The requirements model, implementation units list, and architecture (and if necessary, the PL scope document) are updated. The team and the product manager then discuss the AE team's requests for extending the PL scope (implementing product-specific parts as core assets); if they decide to implement certain parts as core assets, the PL models are changed as required.

Addition to Repository: Core assets are added to the repository (as directed by the core assets manager) for the implemented units and their corresponding requirements and tests.

2) Application Engineering (AE) Sub-process

This sub-process's goal is to build the target products by reusing the core assets built in the CAD phase. An important undertaking in this sub-process is to send requests to the CAD team for extending the PL scope. Three approaches are recommended for this purpose: 1) *Request-In-Advance*: requests are sent prior to starting the development of the product [15] (before forming the implementation units list, as the result may affect this list), and also at the beginning of each iteration (due to possible changes in requirements); the tasks corresponding to this approach reside in the *Pre-Implementation* and *Planning* sub-stages; 2) *Request-During-Implementation*: requests are sent when product development is underway in the *Building* sub-stage [10]; and 3) *Request-In-Retrospect*: requests are sent at the end of each iteration (for the units implemented in the iteration); this approach is implemented in the *Reviewing and Revising* sub-stage. AE phases are explained throughout the rest of this subsection.

 ISBN: 978-1-61208-590-6

a) Pre-Construction

The activities required for launching a new product development project are performed. The only stage of this phase is explained below.

1) Preparation for Build

This stage mainly focuses on analysis activities. Sub-stages are as follows:

Founding: AE teams are formed, with the same structure as in CAD.

Defining Requirements: The AE team elicits the requirements and builds the product's Requirements Model. Two methods are recommended for this task: 1) the requirements available in the core assets repository are provided to the customer, who then selects a subset of them according to his/her requirements [13] (we recommend interviewing the customer for eliciting the requirements that are not present in the core assets); and 2) the customer expresses his/her requirements from scratch [3], and the team matches these requirements with the core assets requirements. The product manager and the AE teams' senior developers then compare the product's requirements to the PL scope to decide whether this product is an instance of the PL; if it is, the product's information is added to the PL scope, and the AE team and the core assets manager extract the core assets related to the product.

Feasibility Analysis: This stage is only performed if the project requires certain resources that are not needed for other PL instances, or if certain risks or constraints are involved. The senior manager and the AE teams' senior developers cooperate in this stage. Analyzing the risks and constraints is first performed, followed by estimating the required resources (based on the amount of core assets that can be used in developing the product); the results of both tasks are recorded in the Project Vision. Finally, feasibility study is performed.

Overall Planning: The overall project plan is produced; again, the amount of core assets usable in developing the product is a crucial factor.

Pre-Implementation: Extension of the PL is requested if the "Request-In-Advance" approach is selected; if the DE team approves the implementation of product-specific requirements as core assets, it develops new core assets. The next task is designing the product architecture: if a PL architecture is available, it is instantiated; if not, a product-specific architecture is developed. Finally, a list of implementation units is produced.

b) Construction

This phase is executed in an iterative-incremental manner. The constituent stages are explained below.

1) Pre-Iteration

This stage is the starting point for development iterations. Sub-stages are as follows:

Planning: The first task, requesting for PL extension, will be performed if "Request-In-Advance" has been chosen. After prioritizing the implementation units, iteration planning is conducted. If any bug-fixes or new requirements are received from the Support Team, they are checked; if they are related to AE, they are added to the iteration plan; if not, they are relegated to the CAD team.

Assigning: Implementation units are assigned to developers (as in CAD).

2) Implementation

AE teams build the product by reusing the core assets. Sub-stages are as follows:

Designing: If the team needs to complete its knowledge of the requirements, customer interviews are conducted. This task is performed in tandem with detailed design, which is performed by instantiating the Domain Design Model (if available) and adding the product-specific parts.

Building: Test design is performed, and the tests in the core assets base are reused. If suitable core assets are available, a partial configuration of the product [10] is produced by assembling them. Product-specific parts are then built, either by adding the product-specific parts to the partial product configuration, or by building the product from scratch; if "Request-During-Implementation" is selected, requests for extending the PL Scope are sent to the CAD team during this task: if the product-specific parts are to be built as core assets, the CAD team designs their interfaces, based on which the AE team develops the product in parallel with the actual implementation of the assets by the CAD team [10]. Code refactoring is then applied [6]. Testing is performed continuously, and can be augmented with code inspection.

3) Post-Iteration

The AE team conducts review activities for finishing the iteration. Sub-stages are as follows:

Reviewing and Revising: Testing is conducted and acceptance tests are run on the implemented units (customer involvement is crucial). The next task is holding a review meeting, in which feedback on the usage of the assets is also recorded and conveyed to the CAD team [15]. The requirements, architecture, PL scope document, and implementation units are then updated. For new or changed requirements, relevant core assets are elicited with the help of the core assets manager. PL extension is requested if "Request-In-Retrospect" is selected. Finally, if there is a product-specific requirement that has recently been implemented as a core asset (as the result of a request for PL extension), the product is re-instantiated so that it includes this requirement as a core asset. The core assets manager ensures that the re-instantiation is done completely.

Transitioning: After the support team is trained, it prepares the training documents. The software product is deployed into the user environment, and conversion is applied. System testing is then conducted, and users are trained.

3) Maintenance Sub-Process

This sub-process spans maintenance and post-mortem activities in the phases explained below.

a) Support

Bug-fixes and new requirements are sent to the AE team. The AE team sends the requests related to the core assets to the CAD team: after applying the changes, the products that include the changed assets are tested and re-instantiated so that the changes are committed. Maintenance is performed via repeating the development iterations.

 ISBN: 978-1-61208-590-6

b) Death

This phase is carried out when a system is not maintainable anymore. Post-project activities are performed: the support team and the senior manager perform the legal, financial, and social activities related to closing the project. Post-mortem activities are then conducted, and the lessons learnt from the project are recorded for use in future projects.

V. EVALUATION

GMAP has been evaluated through two different approaches, as explained in the following subsections.

A. Criteria-based Evaluation

As mentioned before, we have used the criteria introduced in [2] as a basis for developing GMAP; in other words, these criteria have helped us identify the deficiencies and the strengths of previous APLE methodologies, which were then used for constructing GMAP. These criteria can also be used for evaluating GMAP to show that it does indeed address the deficiencies of previous methodologies. The results of evaluation based on criteria related to PLE characteristics are presented in Table III, the results of evaluation based on a set of general methodology-evaluation criteria are illustrated in Table IV, and the results of evaluation based on criteria related to agility characteristics are given in Table V. In all these tables, results are presented for a set of existing methodologies as well as GMAP, so that the results can be compared. In order to produce the results, we searched each methodology for mechanisms that satisfied each criterion. It should be noted that due to lack of space, we have only included the major criteria introduced in [2]. In these tables, "N/A" denotes "Not relevant to the context or properties of the methodology".

B. Evaluation by Instantiating the Proposed Method

To show that GMAP has the capability to be instantiated into a concrete and practicable APLE methodology, we have built a concrete method by instantiating GMAP. The concrete methodology was developed through a PLE project at a major Iranian utilities company. The Customer Management (CM) system used by this company, specializing in purification and distribution of water throughout the country, was the target of this project. The first author worked for three months at this company to help in performing the activities of the concrete methodology, and also in producing its work products. The project served as an effective testbed for improving and validating the methodology in the field.

The company's CM system (operated in most Iranian cities) has certain features common to all the cities. In addition, each city demands its own specific features. This system has long been deemed suitable for development as a product line; however, it has not yet been implemented as such. The main problem is that the commonalities and variabilities among the systems of different cities have not been managed systematically; instead, a common system encompassing the features needed in all the cities has been developed. This has spawned other problems as well: 1) many unusable parts exist for each city; developers try to fix this problem at the code level, but this solution itself has resulted in unreadable code; and 2) a nontrivial modification in the system propagates throughout the whole system. These problems have motivated the company's CM system supervisor to consider the systematic development of a software product line. Since the PL and its instances need to be developed rapidly, it was concluded that an APLE approach would work best for the company. At the beginning of this project, the first version of the concrete methodology was developed by instantiating GMAP based on the project's initial requirements. The concrete methodology was then used at the company, and was gradually configured to better fit the company's needs. At the end of the project, the CM system's manager reported that the concrete methodology had indeed been capable of addressing their problems.

VI. CONCLUSION AND FUTURE WORK

APLE is a new paradigm that has emerged as the result of the need for managing changes in requirements, reducing time-to-market, promoting product quality, and decreasing development costs in software organizations. This approach can be applied to real projects only if adequate guidelines are provided on the activities, people, and work-products involved in the project. A software development methodology can satisfy this need; thus, several attempts have been made to propose practical APLE methodologies. After studying and analyzing these methodologies, we have sensed the need for an APLE methodology that possesses the strengths of existing APLE methodologies while addressing their weaknesses. To this aim, we have defined GMAP, a generic APLE methodology that spans the activities defined in all the studied APLE methodologies and also possesses the desirable features of PLE and agility. This methodology is abstract enough to be instantiated to produce a concrete bespoke methodology. Although adequately abstract, it is detailed to the task level, and provides suggestions as to ways for applying the tasks. The results of criteria-based evaluation of GMAP show that it satisfies the targeted APLE requirements and is indeed superior to existing APLE methodologies.

We aim to continue this work by reporting on the GMAP instance (concrete methodology) that was mentioned in Section V, and also by exploring the potentials of GMAP in addressing diverse APLE requirements. The research can be furthered by applying GMAP to a variety of project situations with different characteristics.

 ISBN: 978-1-61208-590-6

TABLE III. RESULTS OF EVALUATION BASED ON CRITERIA RELATED TO PLE CHARACTERISTICS

Criterion	Possible Values	CDD [4]	de Souza & Vilain [6]	RiPLE-SC [7]	Díaz et al. [3]	A-Pro-PD [10]	Ghanam & Maurer 2008 [11]	Ghanam & Maurer 2009 [13]	Ghanam et al. [12]	da Silva [14]	Carbon et al. [15]	Noor et al. [16]	GMAP
Presence of PL-Specific Activities													
Coverage of DE Activities	"S": Scoping; "A": Reference architecture; "CA": Core assets development.	A-CA	A-CA	S	A-CA	N/A	A-CA	N/A	A-CA	S	N/A	S	S- A-CA
Coverage of AE Activities	"R": Matching product requirements & core requirements; "A": Reference architecture instantiation; "CA": Core assets selection; "V": Binding of variation points to variants; "P": product-specific parts development.	N/A	R-CA	N/A	A-CA-V	CA-P	R-CA	R-CA- P	N/A	N/A	R-A-CA-V-P	N/A	R-A-CA-V-P
Product Line Characteristics													
Extensibility of PL Scope	Yes/No	N/A	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Production and Adherence to Reference Architecture	1: Not produced; 2: Produced, but not adhered to; 3: Produced, and adhered to.	2	2	N/A	3	1	1	1	3	N/A	3	N/A	3
Techniques for Performing PL-Specific Activities													
Prescription of Specific Method for Identifying Core Assets & Commonalities/Variabilities (C/V)	Yes/No	N	N	Y	N	N/A	Y	N/A	Y	Y	N/A	Y	Y
Prescription of Specific Method for Documenting C/V	Yes/No	Y	Y	Y	Y	N/A	Y	N/A	Y	Y	N/A	N	Y

TABLE IV. RESULTS OF EVALUATION BASED ON GENERAL CRITERIA FOR EVALUATING METHODOLOGIES

Criterion		Possible Values	CDD [4]	de Souza & Vilain [6]	RiPLE-SC [7]	Díaz et al. [3]	A-Pro-PD [10]	Ghanam & Maurer 2008 [11]	Ghanam & Maurer 2009 [13]	Ghanam et al. [12]	da Silva [14]	Carbon et al. [15]	Noor et al. [16]	GMAP
Lifecycle														
Coverage of Generic Lifecycle Phases		"D": Definition; "C": Construction; "M": Maintenance	D-C	D-C	D	D-C	D-C	D-C	D-C	D-C	D	C	D	D-C-M
Coverage of Design Activities		Yes/No	Y	Y	N/A	Y	N	N	N	N	N/A	Y	N/A	Y
People														
Definition of Roles and Their Responsibilities		1: No; 2: Roles yes, responsibilities no; 3: Roles and responsibilities defined.	3	1	3	1	1	1	1	1	1	1	3	3
Usability														
Well-definedness	Completeness of Methodology Definition	"L": Lifecycle; "A": Activities; "R": Roles; "P": Products; "RL": Rules; "TP": Techniques/Practices; "U": Umbrella Activities; "ML": Modeling Language.	L-A-TP-R-P-U-RL-ML	L-A (Partial)-PT-P-U (Partial)-RL	L-A-PT-R-P-U (Partial)-RL	L-A-P-U (Partial)	L-A-PT-U (Partial)-P	L-A-PT-P	L-A-PT-P	L-A-PT-P	L-A-PT-U (Partial)-P	L-A-PT-U (Partial)-P	L-A-PT-R-U (Partial)-P	L-–TP-R-P-U-RL
	Management of Definition Complexity	Yes/No	Y	Y	Y	N	N	N	N	N	N	N	N	Y
	Attention to Detail in Definitions of Phases/Tasks	Details provided for: 1: none; 2: some of the phases/tasks; 3: all the phases/tasks	3	2	3	1	1	2	3	3	2	2	3	3
Process Manipulation	Configurability of Process (at the start of the project)	1: No; 2: Possible, but not addressed explicitly; 3: Explicitly addressed	2	2	1	1	2	1	1	1	1	2	2	3
	Flexibility of Process (while running the project)	1: No; 2: Possible, but not addressed explicitly; 3: Explicitly addressed	2	2	1	1	1	1	1	1	1	1	2	3

 ISBN: 978-1-61208-590-6

TABLE V. RESULTS OF EVALUATION BASED ON CRITERIA RELATED TO AGILITY CHARACTERISTICS

Criterion		Possible Values	CDD [4]	de Souza & Vilain [6]	RiPLE-SC [7]	Díaz et al. [3]	A-Pro-PD [10]	Ghanam & Maurer 2008 [11]	Ghanam & Maurer 2009 [13]	Ghanam et al. [12]	da Silva [14]	Carbon et al. [15]	Noor et al. [16]	GMAP
Attention to Customer														
Support for Active User Involvement		Yes/No	Y	N	Y	Y	Y	N	Y	Y	Y	Y	Y	Y
Support for Continuous Customer Feedback		Yes/No	N	N	Y	Y	Y	N	Y	Y	N	Y	N	Y
Teams														
Support for Self-Organizing Teams		1: Not discussed; 2: No; 3: Yes.	2	3	1	1	1	1	1	1	1	1	3	3
Support for Face-to-Face Conversation		Yes/No	N	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
Product														
Support for Continuous Integration		Yes/No	Y	Y	N/A	N	Y	N	Y	Y	N/A	Y	N/A	Y
Process														
Support for Iterative-Incremental Development		Yes/No	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Prescription of Common Agile Practices		Yes/No	Y	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y
Degree of Agility	Support for Rapid Development of Products	1: No; 2: To some extent; 3: Yes.	2	3	2	3	3	2	3	3	2	2	2	3
	Support for Leanness Factors	Yes/No	Y	Y	N	Y	N	N	N	Y	N	N	Y	Y
	Support for Learning (from previous iterations/projects)	1: No; 2: Yes, implicitly; 3: Yes, explicitly.	2	3	1	1	1	1	1	1	2	2	1	Y
	Support for Responsiveness (provision of process feedback)	Yes/No	Y	Y	Y	Y	Y	N	N	Y	Y	Y	Y	Y

REFERENCES

[1] G. K. Hanssen and T. E. Fægri, "Process fusion: An industrial case study on agile software product line engineering," Journal of Systems and Software, vol. 81, no. 6, pp. 843–854, 2008.

[2] F. Farmahini Farahani and R. Ramsin, "Methodologies for Agile Product Line Engineering: A Survey and Evaluation," Proc. International Conference on Intelligent Software Methodologies, Tools, and Techniques, 2014, pp. 545–564.

[3] J. Díaz Fernández, J. Pérez Benedí, A. Yagüe Panadero, and J. Garbajosa Sopeña, "Tailoring the Scrum Development Process to Address Agile Product Line Engineering," Proc. Jornadas de Ingeniería del Software y base de Datos, 2011.

[4] X. Wang, "Towards an Agile Method for Building Software Product Lines," M.Sc. Thesis, University of York, UK, 2005.

[5] R. Ramsin and R. F. Paige, "Process-centered Review of Object Oriented Software Development Methodologies," ACM Computing Surveys, vol. 40, no. 1, p. 3:1–89, 2008.

[6] D. S. de Souza and P. Vilain, "Selecting Agile Practices for Developing Software Product Lines," Proc. International Conference on Software Engineering & Knowledge Engineering, 2013, pp. 220–225.

[7] M. Balbino, E. S. de Almeida, and S. R. de Lemos Meira, "An Agile Scoping Process for Software Product Lines," Proc. International Conference on Software Engineering & Knowledge Engineering, 2011, pp. 717–722.

[8] A. Abouzekry and R. Hassan, "Software Product Line Agility," Proc. International Conference on Software Engineering Advances, 2011, pp. 1–7.

[9] "The Agile Unified Process (AUP)." [Online]. Available: http://www.ambysoft.com/unifiedprocess/agileUP.html. [Retrieved: August-2017].

[10] P. O'Leary, F. McCaffery, S. Thiel, and I. Richardson, "An agile process model for product derivation in software product line engineering," Journal of Software: Evolution and Process, vol. 24, no. 5, pp. 561–571, 2012.

[11] Y. Ghanam and F. Maurer, "An Iterative Model for Agile Product Line Engineering," Proc. International Software Product Line Conference, 2008, pp. 377–384.

[12] Y. Ghanam, D. Andreychuk, and F. Maurer, "Reactive Variability Management in Agile Software Development," Proc. Agile Conference, 2010, pp. 27–34.

[13] Y. Ghanam and F. Maurer, "Extreme product line engineering: Managing variability and traceability via executable specifications," Proc. Agile Conference, 2009, pp. 41–48.

[14] I. F. da Silva, "An agile approach for software product lines scoping," Proc. International Software Product Line Conference, 2012, pp. 225–228.

[15] R. Carbon, M. Lindvall, D. Muthig, and P. Costa, "Integrating product line engineering and agile methods: Flexible design up-front vs. incremental design," Proc. International Workshop on Agile Product Line Engineering, 2006, pp. 1–8.

[16] M. A. Noor, R. Rabiser, and P. Grünbacher, "Agile product line planning: A collaborative approach and a case study," Journal of Systems and Software, vol. 81, no. 6, pp. 868–882, 2008.

[17] T. Vale et al., "SPLICE: A Lightweight Software Product Line Development Process for Small and Medium Size Projects," Proc. Brazilian Symposium on Software Components, Architectures and Reuse, 2014, pp. 42–52.

[18] F. van der Linden, K. Schmid, and E. Rommes, Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering. Springer, 2007.

[19] J. M. DeBaud and K. Schmid, "A systematic approach to derive the scope of software product lines," Proc. International Conference on Software Engineering, 1999, pp. 34–43.

[20] H. Gomaa, Designing software product lines with UML: From use cases to pattern-based software architecture. Addison-Wesley, 2005.

[21] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Carnegie Mellon University, 1990.

[22] K. C. Kang, J. Lee, and P. Donohoe, "Feature-oriented product line engineering," IEEE Software, vol. 19, no. 4, pp. 58–65, 2002.

[23] K. Schwaber and M. Beedle, Agile Software Development with Scrum. Prentice Hall, 2001.

[24] K. Beck and C. Andres, Extreme Programming Explained: Embrace Change, 2nd edition. Addison-Wesley, 2004.

 ISBN: 978-1-61208-590-6