EXPERIENCES INITIATING SOFTWARE PRODUCT LINE ENGINEERING IN SMALL TEAMS WITH PULSE

Hans Martin Mørsk-Møller, Bo Nørregaard Jørgensen
The Maersk Mc-Kinney Møller Institute, University of Southern Denmark
Campusvej 55, 5230 Odense M
Denmark
hmmm@mmmi.sdu.dk, bnj@mmmi.sdu.dk

ABSTRACT
Small teams of software engineers are represented both in small companies and semi-independent fractions in medium and large companies. Even though some results and experience papers have been published in the context of Small- and Medium-Sized Enterprises (SMEs), there is a lack of experience papers and collective results on Software Product Line Engineering (SPLE) in the context of small teams.

This paper remedies this situation by providing experiences from a successful approach of applying software product line engineering in a small team.

We conduct a transition from single-system production to Software Product Line (SPL) in the domain of greenhouse climate control systems. The domain contains inherent variabilities and extensive commonalities between the products in scope, which makes it a prime candidate for SPLE - besides having good business prospects.

The transitioning to SPLE is generally made more difficult due to the inherent characteristic of being a small team – resulting in the requirement of keeping the overhead of the transitioning process to an absolute minimum. Another requirement in our case was keeping the production online. We targeted these requirements by tailoring an existing methodology, PuLSE™, drew advantages of NetBeans Rich Client Platform, and based the product line on the existing application.

KEY WORDS
Software Engineering, Reusability, Software Methodologies, Software Product Lines, Software Design

1. Introduction

The starting point of our approach was the legacy application Climate Monitor¹, which was an application for displaying arbitrary climate data in the domain of greenhouses. Climate Monitor was implemented using the NetBeans Rich Client Platform² (referred to as NetBeans from here on) and developed in a modular fashion. Because of its modularity and easy extensibility the Climate Monitor became basis for experiments in other areas. It was extended with modules for control of climate computers, durability test analyses of ornament plants, and photosynthetic optimization analyses capabilities among others. Over time the Climate Monitor became over-featured, complex and its architecture eroded.

Five new product ideas were conceived and financed as part of another project, Project DynaLight. Similarities in functionality and domain with Climate Monitor were initially recognized and attention was drawn to harvest benefits from reuse. Due to promising results and a profound interest in the field of Software Product Line Engineering (SPLE), an analysis was started to investigate the possibilities of developing an SPL.

The expectations to the reuse through adoption of an SPL were:

- accelerated development
- higher quality applications
- reduction of maintenance cost

Specific requirements to the transitioning process and further maintenance and evolution were set:

- keeping ongoing production online
- feasible by one to five software engineers
- keeping the transition cost to a minimum

These requirements were mainly caused by the current and future number of involved software engineers, besides the current deployments of the Climate Monitor.

¹ http://ecosoc.sdu.dk/coe/Climate_Monitor
² www.netbeans.org
The structure of the paper is as follows: Section 2 State of the Art presents the current research related to the topic of the paper. Section 3 Our Approach presents the ideas behind our approach. Section 4 Software Product Line Engineering is structured accordingly to the technical components PuLSE-BC, PuLSE-Eco, PuLSE-CDA and PuLSE-DSSA as presented in the PuLSENTM methodology. It contains Section 4.1 Baseline and Customization creating an overview and overall customization, Section 4.2 Scoping presenting the process of scoping, Section 4.3 Domain Analysis presenting the construction of the domain model, and Section 4.4 Software Architecture Design our approach to development of the software architecture. In Section 5 Results and Discussion the approach and results of our findings are discussed. Section 6 Conclusion summarizes the results and concludes on the findings.

2. State of the Art

A preliminary investigation was conducted in search for detailed guidance for the transitioning process in our particular context. Our main concern was to find a feasible approach to SPLE for a small number of software engineers; hence a light-weighted, tangible and specialized guidance was sought. The result of the preliminary investigation showed a lack of methodologies addressing small teams in particular.

Thus we turned to investigating methodologies that address closely-related environments; hence our attention was drawn to results of applying SPL in Small- and Medium-sized Enterprises (SMEs). Several of the SMEs have the same amount of developers as our case, thus lacking extensive personnel resources for heavy rigid SPL processes and cannot afford halting the production during transitioning.

The remaining part of this section reviews the state of the art of SMEs, especially in relation to the requirements identified as essential for small teams.

2.1 Guidance

Development of a Software Product Line while keeping software production online is a complex task that requires guidance. The task requires both knowledge of SPL-specific aspects, communication skills and technical expertise. The transition is very context dependent, both with respect to domain, process, resources, organization and technology [1]. Bosch [2], Clements and Northrop [3], Pohl et Al [4] describe the transition in their works. Their level of abstraction is kept above implementation specifics and does not provide concrete tangible guidance for small teams. The works are, besides that, very extensive to cover a variety of cases and mostly aimed at larger organization [5].

The founders of the PuLSENTM [6] methodology address the implementation level by providing concrete tangible examples and reports. The methodology provides concrete guidance through well-defined process steps and limits its extensiveness by customizing the methodology to its application context. Having designed the methodology to be customizable makes it easier to adopt. The PuLSENTM methodology is divided into coherent process elements, which can be applied independently and incrementally.

Some process elements of the PuLSENTM cannot be applied directly by using the published material exclusively. They are dependent on support documentation. These support documents are to the best of our knowledge kept as internal documents, as they provide business value to the IESE, and are not available to the outside. In the following, we will try to fill-in the blanks, using the public available high-level descriptions of these process elements.

2.2 Validity (Evidence)

Successful applications of SPL methodologies exclusively focused on small teams have, to the best of our knowledge, not been presented. Thus we investigated and evaluated experiences closely related to our context (i.e. SMEs) as mentioned in Section 2.

There are few successful experiences published on Small and Medium-sized Enterprises (SMEs) [7]. Clement and Northrop’s work ([3]) includes an experience report where several conclusions are drawn on both advantages and disadvantages of SPLs in small organizations [8]. Several results on the introduction of SPLs in SMEs have been published by Fraunhofer Institute for Experimental Software Engineering (IESE) validating parts of the PuLSENTM methodology [7][9].

From an evidence-oriented perspective, the work of IESE comes closest to dealing with SPL in resource-constrained settings, particularly their work with developing and validating the PuLSENTM methodology on SMEs [7].

2.3 Special Issues

There are problems specifically regarding SPLs in small companies [10]. These are among others: difficulties in finding support for the transition, finding a champion, and difficulties in introducing new technologies. A set of proposed solutions are given, recommending e.g. that the approach is kept simple and adapted to future needs, that the SPL approach is applied first to a well understood sub-
domain, and that the processes are described explicitly for the SPL maintenance and the core asset evolution. The identified problems and proposed solutions are from a project adapting SPL methodologies to small and medium companies that could not afford huge investments or risks introduced by heavier methodologies [10].

2.4 Reducing Transitioning Cost

There are identified three different approaches for adopting SPLE in an organization: Proactive, Extractive, and Reactive [11]. Focusing on less expensive and faster transition to an SPL, the two last mentioned are in scope.

The Proactive Approach is characterized by having a high upfront development cost for creating all reusable assets from which all scoped product members later can be instantiated. This process is considered a heavy adoption approach [11].

The Extractive Approach initializes the SPL by reusing existing common and varying assets, thus it is applicable only in case of existing applications. The high level of reuse can enable a quick transition and reduce the cost.

The Reactive Approach can be bootstrapped on the Extractive Approach; the principle of the Reactive Approach is to incrementally extend the SPL to new requirements.

The Extractive Approach is also known as the reengineering-driven entry point in PuLSE™, and the Reactive and Proactive as the Evolutionary and Revolutionary approaches, respectfully [12].

2.5 Call for Experiences

Kircher et Al identify in [5] a need for finer grained and detailed guidance for starting a software family (ed. product line), but does not delimit the problem space of their need. Guidance and experience reports are also called for in [13], whilst the context is delimited to SMEs.

3. Our Approach

Aiming specifically on keeping the level of effort for our transition as low as possible, production online and approach manageable by a few software engineers, several thoughts were incorporated in our approach from single products to one generic software architecture.

3.1 Lightweight Guidance

We adapted PuLSE™ to a condensed lightweight version, i.e. using only a minimal subset of the whole methodology. The adaption can be divided into the four parts taken from PuLSE™ (see Figure 1). The specific adaption of each element will be treated in their respective subsection in Section 4. In general only work products providing highest value should be maintained due to limitation of personnel resources.

![Figure 1 - Process Elements](image)

3.2 Extractive/Reactive Approach

Based on rationale and the ability of the Reactive approach to lower up-front cost associated with SPL adoption (see [14][15]), this approach was chosen.

Extraction was selected with the intension of accelerating the transition and lowering the cost through reuse of existing assets from the Climate Monitor.

3.3 Reuse over Reinvention

The rationale behind our customization of PuLSE™ is to promote reuse over reinvention. We aimed to reuse the legacy architecture from Climate Monitor, which is based on NetBeans modular design. The reuse of architecture would permit reuse of legacy modules with little or no adaption.

Using NetBeans also provide several other benefits, e.g. a mature component design, rules, design documentation, tutorials etc.

Variability Management in form of the NetBeans Module Management was used for coarse-grained variability at component level. This was done with the intention of removing the effort of designing and developing a new variability management scheme.

Code, documentation and other assets were mined and used for our approach instead of redeveloping these.
4. Software Product Line Engineering

4.1 Baselining and Customization

The intention of the baselining in PuLSE™ is to create a status overview of the current organization before transitioning to a software product line. PuLSE-BC defines a list of areas regarded of value for the forthcoming process steps of SPLE - these areas are called Customization Factors [1][16]. We have in our baselining approach used the Customization Factors as checklist for eliciting different information from the organization (see Table 1).

The information collected during baselining was used to customize our PuLSE™ further, e.g. by choosing modeling description language to UML and Feature Diagrams. This choice was based on the skills of the involved personnel, product portfolio and existing assets.

4.2 Scoping

The intention of the scoping process is to make sure the organization gets full advantage of the benefits from adoption SPLE in their development. This is done by evaluating the reuse potential and analyzing how the organization achieves the best results for the approach. Our customization is based on the Technical Component PuLSE-Eco v.2.0 [17] and mainly the use of it as described in the technical report GoPhone – A Software Product Line in the Mobile Phone Domain [16].

4.2.1 Main Features

The first step in scoping was to describe the main characteristics of the product members in a standardized way, enabling comparison and further elicitation of high-level commonalities and variabilities. This is called Product Line Mapping. Traceability to the source of information is kept to enable back-tracking of decisions and support evolution. Major functionality, release plan and market segment, among others, are elicited in this step as well.

4.2.2 Release Plan

Figure 2 - Product Line Genealogy Draft

The next step was to create a Product Genealogy that gathers the identified coarse-grained variability and commonality of the products from their Product Line Mapping. The Product Genealogy representational form was useful in the current product setting despite expected release dates was not available. A drafted version is shown in Figure 2. The main features of the products were derived from the Product Line Mappings and the order of development visible from the Product Genealogy. These major external visible features were structured in Product
Characterization template per product basis. Products and their main features together with a plan were developed in a coarse-grained way through the use of the aforementioned artifacts.

### 4.2.3 Technical Domains

Technical domains common between product members are identified based on descriptions captured in the developed artifacts of Product Line Mappings, Product Genealogy, and Product Characterization. The artifact designed for capturing the details on the technical domains is the Domain Identity Description. An example of a Domain Identity Description from our approach is illustrated in Table 2 and the Domain Identity Description template can be found in [16].

The Domain Identity Description moves the analysis from high business understanding level to expert descriptions of the technical domains, e.g. Data Source Controller, Climate Computer Communication.

### 4.2.4 System Structure

After the different technical domains were identified and described the different domains were put together in a so-called Domain Structure [16].

The Domain Structure is a compilation of the structures already persistent in the domain descriptions, where references to higher-, lower-, and sub-level domains were established. The Domain Structure Diagram visualizes the different technical domains and their interconnectivity. The intention is to get an overview and identify early deficiencies in the structure and design, also in the current composition into functional blocks. The two work products, the Domain Identity Descriptors and Domain Structure, were developed iteratively.

---

**Table 2 - Domain Identity Descriptor for Climate Computer**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Descriptors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domain Identity</strong></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Climate Computer Communication</td>
</tr>
<tr>
<td>Information Source</td>
<td>IntelliGrow Experts, Superlink 4 developers</td>
</tr>
<tr>
<td>Primary Function</td>
<td>Provide uniform communication interface to the different climate computer brands, types and versions.</td>
</tr>
<tr>
<td>Boundary Rules</td>
<td>In: Control different elements of the climate computers, e.g. window shades, window actuators, heating, supplementary light, in case the computer allows external control. Provide access to data from the climate computers, e.g. temperature measurement, moist level, CO2 level. Out: Manipulation of data is outside the scope of this element.</td>
</tr>
<tr>
<td>High Level Domains</td>
<td>Climate Computer Controller/Mediator</td>
</tr>
<tr>
<td>Lower Level Domains</td>
<td>--</td>
</tr>
<tr>
<td>Sub-Domains (Embedded Domains)</td>
<td>--</td>
</tr>
<tr>
<td>Functions</td>
<td></td>
</tr>
<tr>
<td>Function list and description</td>
<td>On request the domain should list the services the specific climate computer addressed. Based on reflection using this list and common standards for method-naming the user of the entity should be able to retrieve data or/and control specific functions offered by the climate computers.</td>
</tr>
<tr>
<td>Data/Objects handled and stored</td>
<td>ClimateComputerDataSet is handled by the Climate Computer Control entity – data is retrieved from the climate computer and made available to higher-level domains. ClimateComputerControlObject are configured according to the control of the given accessory</td>
</tr>
<tr>
<td>System Characteristic</td>
<td></td>
</tr>
<tr>
<td>Existing Assets</td>
<td>There is currently an implementation which uses the enterprise communication pattern, Database Communication, identified by Martin Fowler [18]. It is basically a driver written for the Superlink 4 communication to the SENMATIC climate computer; this driver pulls data from climate computer and inserts the retrieved data into a database. The driver is run as a Windows Service, i.e. completely separated from the program.</td>
</tr>
<tr>
<td>(Sub-)System Relationship</td>
<td>Climate Computer outside application boundaries.</td>
</tr>
<tr>
<td>Product Relationship</td>
<td>The Climate Computer Communication is included in all the products, but some products should not permit control but only data retrieval.</td>
</tr>
</tbody>
</table>
4.2.5 Product Mapping

Before the initial Product Map can be compiled, the product and technical domain descriptors need to be sufficiently stable. The Product Map should not be confused with the Product Characterization, as the latter only contains external main features for a single product, whilst the Product Map also contains internal technical details and covers all products in the SPL scope. Product line mapping consists of analyzing the reusability and benefits of including certain features. Each of the rows in the Product Map is evaluated by stakeholders – in our case the SPL owner and the project responsible did this evaluation. The result was a large matrix providing a valuable overview of the analysis results. These results were then evaluated further to specify which features should be developed as reusable assets and included in the Software Product Line Architecture (SPLA). Our case study differs from the published examples of PuLSE-Eco as our estimate of effort was more dependent on refactoring to reusable assets than to develop new. A simplified except from our Product Map is shown in Table 3.

Table 3 - Simplified Product Map

<table>
<thead>
<tr>
<th>Technical Domain</th>
<th>Feature</th>
<th>Products</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Manual</td>
</tr>
<tr>
<td>Output graphs</td>
<td>Hours with supplementary lighting</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td>Plant regeneration period (total darkness)</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td>Electricity price per hour (€/MWh) or (DKK/MWh)</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td>Total electricity price for using supplementary lighting</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Data sources</td>
<td>Weather forecast</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td>Electricity prices (Nordpool)</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td></td>
<td>Electricity distribution prices (Energinet.dk)</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Data charts</td>
<td>x-y plots</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Computational models</td>
<td>Time of sunrise and sunset (needed by plant regeneration)</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
</tbody>
</table>

✓=Included  ✓✓ Optional  blank = Excluded

4.3 Domain Analysis

The first step of the domain analysis is to elicit raw domain knowledge and specify this knowledge in standardized artifacts. This step is comparable to development of single systems. The second step is to model the artifacts into consolidated generic artifacts capturing commonalities and variabilities. These two steps are iteratively applied until the whole domain is captured. A decision model is developed in parallel which specifies how to transform the generic requirements back to being product specific. It should be noted that the domain model not only include scenarios, use cases, etc., but also artifacts like terminology and diagrams.

We customized our PuLSE™ instance in several ways: 1) There was no legacy information using a specific notation, thus UML, Use Cases, and Feature Diagrams were used. 2) Traceability was decided to be controlled manually until requirement for a tool was identified.

4.3.1 Customization

Most domain knowledge was derived directly from the Climate Monitor, its former developer and its user manual. The elicited use cases were then modeled into generic use cases. These generic use case templates extend the use cases templates described by Alistair Cockburn with Meta Tags for variability [16]. A snippet of a use case with Meta Tags is shown in Table 4 with three different alternatives in step 4 of the Main Success Scenario.
There are three different forms of variability [19]:

- **Optional Requirements** – requirements which should be part of an instance or not part of the specific instance.
- **Alternative Requirements** – a set of requirements of which only one or a subset of the requirements holds for a specific product instance.
- **Range Requirements** – requirements which specify a numerical range, instead of a specific value required by a single system.

Each generic asset containing variability must have some form of meta mechanisms enabling the derivation of product-specific assets.

PuLSE-CDA describes elicitation of knowledge and modeling of knowledge as two separate process steps, while we experienced difficulties in keeping them separate. This could be an effect of using extraction as preliminary part of a reactive approach to develop the SPL.

---

### Table 4 - Use of Meta Tags in Use Cases

<table>
<thead>
<tr>
<th>Required database types supported (predefined set is visible in a drop down list)?</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ALT 1&gt; Weather forecast, Local weather, Greenhouse climate, Electricity prices (Nordpool), Electricity distribution prices (Energinet.dk) (Product Variant Climate Monitor)</td>
</tr>
<tr>
<td>&lt;ALT 2&gt; Weather forecast, Electricity prices (Nordpool), Electricity distribution prices (Energiner.dk) (Product Variants Light Sum and Photosynthesis)</td>
</tr>
<tr>
<td>&lt;ALT 3&gt; Electricity prices, Electricity distribution prices (Nordpool) (Product Variant Electricity Price)</td>
</tr>
</tbody>
</table>

### 4.3.3 Domain Decision Model

The decision model was structured hierarchically into three different levels: Lowest level treating variability within use cases, UML models, and Feature diagrams. Intermediate level variability on whole work product level, e.g. a complete use case, and highest level sets of work products directly manipulated by choices in the product map. An example fragment of our decision model is shown in Table 5. It should be noted that the resolution is bounded in sets (e.g. ID 2 in Table 5) – this is done to reduce complexity i.e. the amount of flexibility and choices in the decision model.

### Table 5 - Example Decision Model

<table>
<thead>
<tr>
<th>ID</th>
<th>Question</th>
<th>Subject</th>
<th>Resolution</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>How is the data displayed?</td>
<td>Hours with supplementary lighting</td>
<td>XY-Chart</td>
<td>Remove &lt;ALT 2&gt; from step 6 in UC1: Display hours with supplementary lighting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Alternative</td>
<td>Remove &lt;ALT 1&gt; from step 6 in UC1: Display hours with supplementary lighting.</td>
</tr>
<tr>
<td>2</td>
<td>What database types are supported?</td>
<td>Supported Data Resources</td>
<td>Weather forecast, Local weather, Greenhouse climate, Electricity prices, Electricity distribution prices</td>
<td>Remove &lt;ALT 2&gt; and &lt;ALT 3&gt; from step 4 in UC4: Add Data sources.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Weather forecast, Electricity prices (Nordpool), Electricity distribution prices</td>
<td>Remove &lt;ALT 1&gt; and &lt;ALT 2&gt; from step 4 in UC4: Add Data sources.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Weather forecast, Electricity prices (Nordpool), Electricity distribution prices</td>
<td>Remove &lt;ALT 1&gt; and &lt;ALT 2&gt; from step 4 in UC4: Add Data sources.</td>
</tr>
</tbody>
</table>

### 4.4 Software Architecture Design

The inherently more complex creation and validation of a generic architecture covering several systems requires additional guidance in comparison to development of single-system architectures. PuLSE-DSSA is used as base for our architecting approach, but the process was heavily influenced by the decision of using NetBeans and having it as software legacy asset from the Climate Monitor.

### 4.4.1 Architecture Model

The process consists of creating and selecting scenarios and iteratively using them to form the architecture. Most of our scenarios are directly traceable to the Domain Model, but some solution-specific scenarios were added, e.g. scenarios capturing non-functional requirements. A small part of the component architecture is shown in Figure 3. Special attention should be drawn to the <<OPT Variant>> tags specifying the components being optional. The bounding in sets enforced by the Architectural Decision Model is not apparent in the model.
<table>
<thead>
<tr>
<th>ID</th>
<th>Question</th>
<th>Subject</th>
<th>Resolution</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>Should the solution contain a Data Source module to add data sources to the application?</td>
<td>Existence of more data sources.</td>
<td>Yes</td>
<td>remove &lt;&lt;OPT Variant&gt;&gt; mark in the reference architecture on component IDXX, IDXX-Y1, IDXX-Y2, IDXX-Y3, IDXX-Y4, IDXX-Y5.</td>
</tr>
</tbody>
</table>

### 4.4.2 Architecture Decision Model

An Architecture Decision Model was developed in the same way as the Domain Decision Model to enable instantiation of composition of modules fulfilling the specific requirements of a product member. The ID on the elements shown in the Architecture Decision Model excerpt in Table 6 enables resolution of the variability from the Architecture Decision Model to the components - this enables the instantiation. From the developed reference architecture it was possible to instantiate all SPL members.

Requirements to the Architecture Decision Model originating from the Domain Model and Domain Decision Model are traceable by ID mapping. This gives the possibility to trace design decision to where it originated, and changes in the domain can be migrated to the solution space by following this form of bi-directional traces.

### 5. Results and Discussion

This experience paper has presented tangible examples of the processes and key work products from conception of the product line idea to a product line architecture covering the final member products, and underlined our adaption of the PuLSE™ methodology elements. The constraint of ongoing production has not affected the transitioning phase, due to our focus of incrementally introducing change and using the legacy application as seed for creating the SPL. The tangible outcome of our approach to reduce the adoption barrier in our resource-constraint small team was threefold:

- The approach used an existing methodology by adapting it to the requirements of small teams, in general reducing its extent and complexity.
- Extensive areas of the legacy application was extracted and reused as foundation for establishing the SPL. The SPL was maintained and extended using the Reactive Approach after its initialization phase where the Extractive approach was used. The combination of the Extractive and Reactive Approach led to a quick transition at low cost.
- Third-party modular design was used in form of NetBeans. Besides saving effort in designing and developing a custom application it also provided means of managing variability on module granularity, and provided general purpose application components, i.e. help menus, GUI, and wizards.

It is our hypothesis that using the combination of the Extractive and Reactive Approach especially helped to lower the adoption barrier for our organization. It seems rational to expect savings in effort by wrapping, refactoring and mining already existing assets – it should be noted that the existing modular design of the legacy
application facilitated our approach to reuse. Choosing a reactive instead of a proactive approach lowered the time-to-market of the first product members.

Our adoption of SPL focused on easing the transition and keeping the overhead to a minimum. Therefore we decided to choose PuLSE™, an existing methodology as our starting point. The rationale behind this decision was that methodologies already validated in closely related settings were likely to yield a successful transitioning in our case. Developing a new methodology from scratch could have required a lot of effort, thus customizing an existing and validated approach to our organization was assumed to lower the upfront investment. It is difficult to prove that these assumptions held, but we are convinced they effectively lowered the barrier, and constitutes one of the cornerstones in our successful transition.

NetBeans is developed to accelerate development of applications by providing extensive APIs to support common application features like GUI, automatic update facilities, component infrastructure, lookup implementation etc. – but it also provides guidelines for software architecture and learning paths through online documentation and community. The software architecture of NetBeans has a modular architecture, which has been matured over several years. The rationale for using NetBeans was that it would be less costly reusing a proved and mature design instead of developing our own component framework, besides an incitement was already there as it was basis of the legacy application.

The general idea behind our approach to SPL can be said to be reuse over reinvention – most importantly the reuse of methodology, legacy assets and NetBeans. Even though these elements of reuse attack different issues of the transitioning process; they build on the same principle of reuse and adaption. It is our opinion that the combination of practices incorporated in PuLSE™ truly eliminates the adoption barrier, lower the cost and accelerate the transition, not only in our case – but for all organizations in similar context planning to migrate to SPL.

6. Conclusion

The important contribution of this experience paper is a customization of PuLSE™ which allows small teams to make the transition from a single product to an SPL without too much overhead in the upstart phase. Tangible work product, concrete examples and recommendations to others have been provided, which hopefully can guide others in similar resource-constrained settings.

The result of our efforts was a software product line design covering the new portfolio and drawing significant benefits from legacy assets into account. The achievement of this result cannot be said to validate our approach in general, as one successful case study is not enough for that, but it adds to the empirical base.

Acknowledgments

Special acknowledgement goes to the researchers behind the PuLSE™ Methodology for publishing not only parts of their developed methodology, but also for contributing with experience papers, reports, and results from technology transfers.

The authors wish to acknowledge the European Regional Fund, Region South Denmark and University of Southern Denmark for the financial support through the project Intelligent Energy Handling in Greenhouses (project number 95-410-44060).

References


http://www.ghc2017.com/Intelligent_energy_handling/TheProject.htm
organization. A case study (Kaiserslautern, Germany: Fraunhofer IESE, 2001)


