Supporting Online Updates of Software Product Lines: A Controlled Experiment

Bartosz Michalik*†, Danny Weyns*‡, Nelis Boucké*, Alexander Helleboogh*
*DistriNet Labs, Katholieke Universiteit Leuven, Belgium
†Institute of Computing Science, Poznan University of Technology, Poland
‡School of Computer Science, Physics and Mathematics, Linnaeus University, Campus Växjö, Sweden
Email: {bartosz.michalik,nelis.boucke,alexander.helleboogh}@cs.kuleuven.be, danny.weyns@lnu.se

***Abstract*—The evolution of Software Product Lines (SPL) is challenging because stakeholders have to deal with both regular evolution and the co-existence of different products. Our focus of product evolution is on the tasks integrators have to perform to update deployed SPL products with minimal interruption of services. In case of Egemin, our industrial partner, the updates of SPL products is further hampered as a consequence of outdated and imprecise architectural knowledge of deployed products. To facilitate the updates of products, we have developed the architecture-centric approach which comprises two complementary parts: an update viewpoint and a supporting tool.**

In this paper we present an evaluation of the architecture-centric approach. The approach is compared with the Egemin's current update approach in a controlled experiment. In the experiment 17 professionals were asked to perform 68 updates of logistic systems. The results obtained from the experiment show that the architecture-centric approach significantly improves the correctness of updates and reduces the interruption of services during updates of Egemin's SPL products.

***Keywords*-SPL, software product line, on-line updates, experimentation**

I. Introduction

Improved productivity, enhanced quality, and reduced time to market are reported benefits of Software Product Line (SPL) adaptation [18]. Clements and Northrop [4] define an SPL as a set of software-intensive systems (products) that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and are developed from a common set of core assets in a prescribed way.

Many companies have successfully adopted an SPL. Nevertheless, managing the evolution of a product line and ensuring consistency of changes in the affected products remain key research challenges in the field [15]. Integrators have to cope with regular evolution, as well as with the co-existence of different products. In addition, architecture documentation of an SPL is often imprecise because the effort to manually update this documentation is often counter productive [8].

In a recent R&D project in collaboration with Egemin[1], an industrial manufacturer of logistic systems, we faced the problem with a lack of accurate architecture documentation in the context of the evolution of the company's SPL products. Our particular focus on product evolution is on the update tasks integrators have to perform to evolve one or more deployed products of an SPL. Unfortunately, due to the inherent complexity of logistic systems and the lack of explicit architecture documentation of the deployed products, Egemin's integrators are forced to use improvised update practices which not infrequently result in erroneous updates. Restarting a system in an incorrect configuration may lead to serious problems with an industrial installation which harm the reputation of the company. The problem grows with the number of managed industrial installations. Currently, Egemin maintains more than 200 deployed products, which require in total about 300 update tasks per year. Therefore, the company searches for a systematic solution that can improve their update practices.

To address this problem, we have developed an architecture-centric approach [20] to support SPL products updates. Our approach comprises two complementary parts: an update viewpoint and a supporting tool. The viewpoint defines the conventions for the construction and use of architectural views to deal with the stakeholders' update concerns, in particular correctness and availability. The supporting tool assists integrators with reconstructing missing architecture knowledge using data harvested from the deployed resources, and guiding them with performing the updates. In order to demonstrate the effectiveness of the architecture-centric approach, we designed a controlled experiment in which 17 professionals (subjects) perform 68 updates to logistic products (objects).

In this paper, we report on the evaluation of the approach effectiveness with respect to two quality requirements inherent to products updates. The first examined requirement is the *correctness of an update*, which means that a deployed system must be transformed to the new configuration without compromising its consistency. The second examined requirement is *availability of the system under update*. Availability refers to minimal interruption of services during updates which is key for 24/7 industrial installations. Next to these quality requirements, we analyzed the confidence level of the integrators on the correctness of updates. To reason about the effectiveness, the proposed solution is compared with Egemin's current practices (*baseline* approach) in a controlled experiment.

The remainder of this paper is structured as follows. Section II introduces the update problems with Egemin's SPL and briefly describes the proposed solution. In Section III, we explain the design of the experiment. Section IV elaborates on the analysis of the obtained results. Validity threats are discussed in Section V. Finally, we present related work in Section VI, and draw conclusions in Section VII.

[1] http://www.egemin.com

II. CONTEXT

A. Egemin's SPL

Egemin is an industrial manufacturer of logistic systems. Egemin's SPL evolved from previous separate solutions and over time the company widened the SPL scope by offering customers new features. A typical configuration of a logistic system is presented in Figure 1. The software is deployed on three hosts. Each logistic subsystem comprises a service and a client that make use of the distributed logistic platform. While a service offers the functionality of the subsystem, a client offers a graphical interface to access the service. The product includes a warehouse management system (E'wms® - Egemin warehouse management system) that is responsible for managing tasks in the system, and control software for the automated guided vehicles (E'tricc® - Egemin transport intelligent control center). The subsystems consist of multiple interrelated components that can communicate.

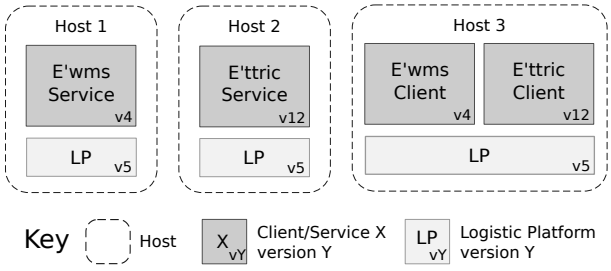

Fig. 1. Typical configuration of a product.

Logistic systems are long-lived; typically 10+ years. During this lifespan, they evolve as a result of maturation of SPL assets, changing customer requirements and environmental settings. The goal of an evolution is to migrate the deployed logistic system (*as-is*) to a new version (*to-be*).

To migrate a product, integrators in Egemin use installation bundles containing new versions of various submodules that comprise the update resources (e.g. executables and libraries).

Three representative examples of update scenarios, observed in Egemin's SPL, are:

- A security submodule of the logistic platform must be updated to remove a problem with the cipher mechanism found in version v_5.
- A new crane device is bought from a 3^{rd} party provider by a customer and the control software of this device must be integrated with the existing configuration.
- A new statistical module for E'wms, that improves throughput, is offered to the clients. This module can be integrated with E'wms from version v_4 on.

As mentioned in the introduction, there are two important quality requirements related to these scenarios:

R1. *Correctness of the update*: The integrator should perform a correct sequence of update steps to bring the product from the as-is to the to-be version. Update steps include adding/removing/replacing resources and stopping/starting processes. Restarting an incorrect configuration may compromise the logistic system's consistency.

R2. *Availability of the system under update*: The integrator should minimize the total shutdown time of the various logistic subsystems. Logistic systems typically have to operate 24/7. Interruption of its services is costly and should be kept minimal.

B. Problems with product updates

Egemin's SPL adoption is an example of a Ploughed Fields Adoption of a SPL [4], or Reenginering-driven SPL adoption according to Schmid and Verlage's classification [16]. The SPL emerged from a set of products previously offered by the company. As a consequence, Egemin's SPL contains a lot of legacy code which is not fully documented. Hence, only a coarse-grained mapping between features and the software artifacts is maintained.

The subsystems are developed by different teams that are relatively independent. Moreover, most of the architectural knowledge is personalized among engineers from the development teams. In addition, due to ad-hoc updates, the architectural description of deployed systems drifts from the actual configuration thus the traceability between the SPL asset base and the product's components is imprecise. Therefore the knowledge of deployed products and their execution environment required to perform update tasks is not available to the integrators.

The lack of required knowledge of deployed products leads to improvised update practices that are error-prone. Moreover, uncertainty about the product configuration makes it difficult to achieve the correctness (R1) and availability (R2) requirements discussed above. To reduce the risk of incorrect updates, the integrators team consists of highly skilled engineers from the different development teams. This increases the cost of updates and diverts the most experienced employees from their daily tasks. Despite the costs, the approach does not guarantee correctness of the update. Being aware of the consequences of their current update approach, Egemin searches for a systematic solution to improve the updates of their products.

C. Architecture-centric Approach

The architecture-centric approach we developed for supporting online updates of SPL products comprises two complementary parts: an update viewpoint and a supporting tool. The viewpoint defines the architectural conventions for handling the updates of the SPL products. The tool reifies the viewpoint concepts for a particular SPL, supporting integrators with performing product updates.

1) Update Viewpoint: The ISO/IEC 42010 standard [11] defines an architecture viewpoint as "a work product establishing the conventions for the construction, interpretation and use of architecture views to frame specific system concerns." As such, the update viewpoint establishes the conventions for defining and using update views to deal with the correctness and availability concerns of SPLs.

The viewpoint defines four model kinds that deal with the previously mentioned concerns. As-Is (M1) and To-Be (M2) Product Deployment models allow stakeholders to browse the

structure of a deployed product (as-is) and the future product (to-be) respectively. The Update Procedure Model (M3) shows the update steps that integrators have to perform to update a deployed product, dealing with the availability and correctness concerns. Finally, the Update Inconsistencies Model (M4) shows inconsistencies of the product, dealing with the correctness concern.

The four model kinds are based on an integrated meta-model that defines the conceptual entities, their attributes and the relationships that comprise the vocabulary of these model kinds. Moreover, the integrated meta-model offers the basis for an architectural repository that we use to harvest the relevant information from which the models are derived (we further discuss this in following subsection). The detailed description of the update viewpoint is presented in our previous work [20].

2) Supporting Tool: The construction of the architecture models for supporting updates of the SPL products consists of four phases: (i) harvesting relevant architecture knowledge; (ii) storing the harvested knowledge; (iii) analyzing architectural knowledge; (iv) visualizing the architectural models in a comprehensive way for the stakeholders.

We developed a prototype tool to facilitate the update process for Egemin SPL. A coarse-grained architecture of the tool is presented in Figure 2. There are four main building blocks that encapsulates the functions required to implement the previously mentioned phases.

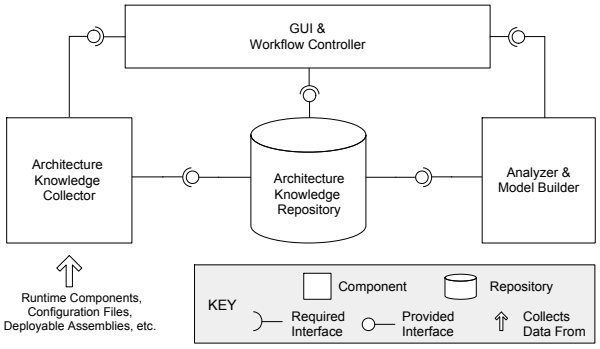

Fig. 2. Coarse-grained architecture the tool.

The Architectural Knowledge Collector. This module comprises a number of plugged-in harvester components that perform the actual knowledge gathering. The knowledge collector provides a common infrastructure to configure and manage this data collection. A harvester is a small subprogram that extracts architectural knowledge from specific sources. There are three harvesters used for the Egemin's products line that collect information about components, their static and dynamic dependencies, versions, and installation bundles. Egemin products are based on the .Net platform, therefore, the required knowledge is harvested from .Net assemblies, XML configuration files, and MSI installation bundles.

Architecture Knowledge Repository. The repository is populated with the data collected by the harvesters. The repository stores architecture knowledge that complies to the integrated meta-model defined by the update viewpoint [20]. We used the Eclipse Modeling Framework[2] as a basis for the repository.

Analyzer & Model Builder. The analyzer uses the collected data to provide models M3 and M4 of a product. Model M3 results from the comparison of the as-is and to-be models [19]. Model M4 is obtained by verifying whether there are any unresolved dependencies between system resources. In addition, company-specific rules can be defined that are checked during the analysis. E.g., one of the rules defined for Egemin's SPL requires that the version of a given assembly should be identical at all locations of a logistic system. The model builder visualizes models M1-M4 using the data stored in the repository and obtained from the analysis. We have used Eclipse graphics libraries[3][4] for the implementation of this module.

GUI & Workflow Controller. This module provides the end-user interface for using the tool and browsing the models. Figure 3 shows a snapshot of the GUI, in which the Update Procedure model (M3) for a concrete project is shown. The top-left box shows the different locations on which the product is deployed. The box on the right hand side shows the installation bundles (product installers) that have to be deployed on the selected location. The most important information is presented in the bottom box, which shows an update script that lists the subsequent steps to realize the update.

An integrator can access the other models using the navigation bar visible in the top of the figure. The As-Is Product Deployment model enables the integrator to navigate the structure of the currently deployed product and examine the dependencies between assemblies. The To-Be Product Deployment model allows the integrator to browse the structure of the installation packages that must be used in the update. Finally, the Update Inconsistencies model provides the information about missing assemblies and version problems of the current installation.

An interaction with the tool starts with the configuration of the harvesters. During the configuration, the locations of a deployed product and installation bundles have to be indicated. Next, an integrator can run a harvesting process and navigate through the presented models. The harvesting step can be executed several times during an update, for example, to monitor the progress of the update.

III. Design of the Experiment

To evaluate the effectiveness of the architecture-centric approach for online updates and support the transfer of the tool to Egemin's practice, we decided to perform a controlled experiment. For the design of the experiment, we followed the guidelines provided by Wohlin et al. [21]. The evaluation is an in-vitro supervised experiment with a paired comparison design, where supervision has been performed by the authors of this paper. In this paper, we give a rigorous overview of

[2]http://www.eclipse.org/emf
[3]JFace (http://wiki.eclipse.org/index.php/JFace)
[4]SWT (http://www.eclipse.org/swt)

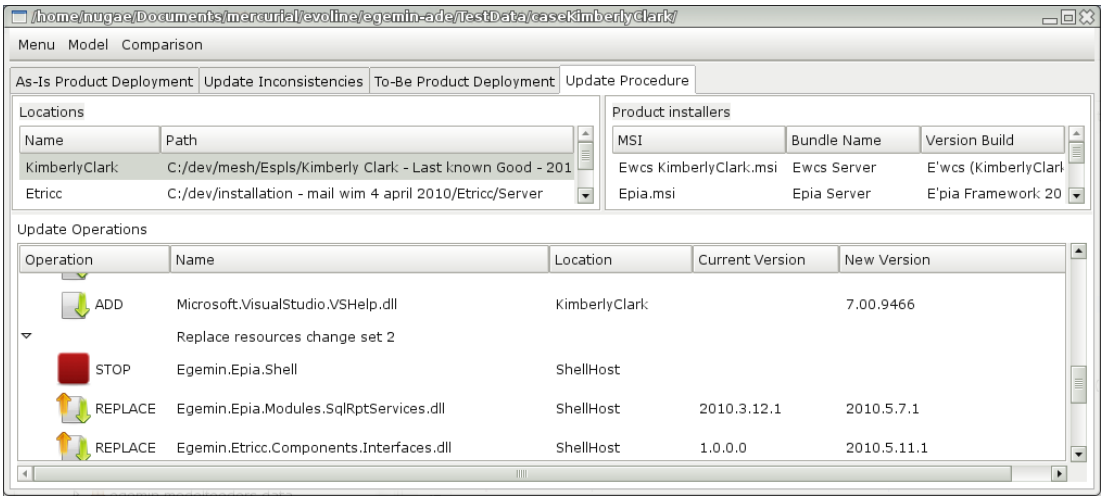

Fig. 3. Snapshot of the tool. An example of the Update Procedure model for Kimberly Clark configuration.

the experiment. For a complete description of the experiment with all the detailed results, we refer the reader to [14].

A. Goal

The goal of this study is to *evaluate the effectiveness of the architecture-centric approach with regard to the correctness (*R1*) and availability (*R2*) requirements. Additionally, we aim to investigate the integrators' confidence level on the correctness of updates with the tool in comparison to Egemin's current update practice.*

B. Questions

To evaluate the effectiveness of the architecture-centric approach and assess the integrators' confidence level on the correctness of updates, we want to answer the following concrete questions:

Q1 Is there a difference in correctness of the updates between the architecture-centric and the baseline approaches?

Q2 What is the difference in availability of the system under update when the architecture-centric and the baseline approaches are used?

Q3 Is the integrators' confidence affected by the approach that is used to perform updates?

C. Tasks Description

During the experiment, subjects have to update logistic systems derived from the Egemin's SPL. The goal of an update is to bring a system from its current (*as-is*) state to the expected new state (*to-be*). To update the system a subject has to stop and start relevant services, and remove, replace and add resources (.Net assemblies and executables). During an update, either the architecture-centric approach or the baseline approach is used.

The updates are performed on variants of the Kimberly Clark (KC) logistic system. The system is installed in the standard location (Program Files/Egemin) and is operational. The KC logistic system consists of an E'tricc service, an E'wms service, an E'pia service (logistic platform), and the associated clients (see Figure 1).

For the purpose of the experiments we have defined three kinds of the update scenarios:

S_0 Learning scenario kind - these are easy scenarios which require simple changes that affect one of the subsystems. The aim of the learning scenarios is to familiarize subjects with the system under update and the usage of our tool.

S_A Simple scenario kind - in these scenarios, the E'wms or E'tricc services and corresponding client screens are object of the update. The number of changes is limited. To perform the correct update two processes must be restarted.

S_B Complex scenario kind - in these scenarios, the whole logistic system is affected by the update. Most of the running processes have to be restarted to perform the update correctly.

The concrete scenarios were prepared by Egemin's senior architect, who did not participate in the experiment as a subject. The proposed scenarios are similar to the updates Egemin integrators perform in practice. We asked the subjects to perform two updates for each scenario kind, one with the architecture-centric approach, the other with the baseline approach. We prepared pairs of similar scenarios to ensure objectivity and avoid interference between treatments.

D. Selected Variables

In order to analyze the impact of the used approach on the update process we considered the following depended variables:

I1 Correctness of an update - a variable that expresses the result of an update, i.e., an update is correct when a subject performed all the required file modifications for a given scenario. For the analysis, we use a binary metric that indicates whether the update is performed correctly or not.

I2 Number of unique process shutdowns - the number of processes that are stopped at least once during a given

update task. For the analysis, we use the difference between the optimal and the actual number of unique shutdowns as a metric.

I3 Number of process shutdowns - the total number of the process shutdowns that are performed during the update task. For the analysis, we use the difference between optimal and actual number of shutdowns as a metric.

I4 Time: duration of an update (expressed in minutes) - the time measured between the moment a subject finishes reading a scenario and the moment the update is completed.

I5 Confidence level - a questionnaire-based self-assessment of a subject's confidence about the correctness of an update. We used a Likert 5-point scale [12] in the questionnaires.

I6 File modifications ratio - the number of file modifications made during an update task. In the statistical analysis, we use the ratio between optimal and actual number of file modification as a normalized metric.

In addition, we defined two independent variables: an approach (architecture-centric or baseline) and the complexity level of the scenario. For the experiment output analysis, we used only the data collected from the simple and complex scenarios.

E. Subjects

Seventeen highly qualified professional volunteered in the experiment. That includes six integrators from Egemin and eleven experienced research fellows from the DistriNet group.

F. Preparation and Materials

We performed a pilot experiment with three researchers from the DistriNet group. The pilot participants were not involved as subjects in the actual experiment. The goals of the pilot study were: (i) to discover all possible problems related to the execution of the update tasks; (ii) to get some indicators for the complexity and the time needed to complete the tasks; (iii) to examine the clarity of the update scenario descriptions and the questionnaires.

The findings from the pilot study helped us to improve the materials and the experimentation procedure. First, we removed some ambiguities from the scenario descriptions and questionnaires. Second, as we discovered that switching scenarios was time consuming and could lead to mistakes, we decided to create automation scripts to prepare the update tasks. Third, we have standardized and written down the guidance for supervisors.

As previously mentioned, we prepared a presentation to introduce the SPL update problem to the subjects. The first part of this presentation explains the logistic system, the installation structure, and the quality requirements for a product update. The second part introduced the tool. The interested reader can access the presentation, and the update scenario descriptions at the project web-page[5].

In addition, we prepared six virtual images of the different variants of the KC system to ease the setup of the experiments. We developed a simple tracing tool, that runs in the background, to automatically count file modifications and track time stamps of events. Finally, we provided the scenario descriptions and notebooks for subjects. The questionnaires for all updates were handled via an on-line application.

[5]http://people.cs.kuleuven.be/bartosz.michalik/evoline/

G. Execution

To collect the data, we have organized 17 experiment meetings. In each meeting, one subject and one supervisor participated. During the meeting of 2.5 hours, a subject has to perform six different, supervised updates to a logistic system. The supervisor prepared the experiment materials and setups, took note of the subjects actions relevant to the experiment, and assisted the subject when required.

The following description of the experiment is common for all 17 subjects. At the beginning of the meeting the subject is given an introductory presentation about Egemin's logistic system, the architecture-centric approach for performing updates, and the tool. Next, the subject is asked to perform the updates described in two learning scenarios. The objective of this phase is to familiarize the subject with both, the logistic system's structure and the tool. Prior to the execution of an update, the supervisor prepares the environment, starts and configures the tool. The subject then receives the description of the first update scenario and is asked to perform the update using the tool. After completion of the update, the subject fills out a task related questionnaire. The questionnaire covers task difficulty, level of confidence and additional comments. This procedure is repeated for the second update scenario. However, this time the subject uses the baseline approach. An identical questionnaire is used for the second scenario. During the execution of the learning scenarios, the subject can consult the supervisor whenever needed.

After completion of both learning scenarios, the controlled part of the experiment begins. Subjects perform four different update tasks using both approaches for simple and complex scenarios. A subject performs a random sequence of scenarios, such as: (i) the second simple scenario using the architecture-centric approach, (ii) the first complex scenario using the baseline approach, (iii) the second complex scenario using the architecture-centric approach, (iv) and the first simple scenario using the baseline approach ; or any other combination.

In the controlled part of the experiment, the following steps are applicable to each of the update tasks:

a) The subject reads the description of an update scenario.
b) The supervisor prepares the update setup. This step includes starting the logistic subsystems and the tracing tool. Next, he configures the update tool and performs an initial harvesting. This step applies only to the scenarios in which the tool is used.
c) The subject performs an update of the logistic system. The supervisor keeps track of the relevant actions of the subject.
d) The subject fills out the scenario questionnaire.
e) Relevant data is collected for further analysis, including the state of the logistic system after completion of the update task.

In the final step the subject fills out an evaluation questionnaire for the complete experiment.

H. Data Analysis

Three data sources are important in our study: the supervision reports, automatically collected data, and survey results.

Relevant data include logs of the subjects actions with timestamps, including the number of process shutdowns and restarts, automatically collected logs of file modifications and scenario errors, and data of on-line surveys about the level of confidence of the update correctness.

We decided to treat all subjects as a single group based on the results of several Kolmogorov-Smirnov tests [9]. Then, we have checked whether there is no relationship between the observed variables using graphical methods and Person's correlation matrices. Next, we checked whether the data comes from a normally distributed population using the Sharpio-Wilk's test [17]. As we have not observed any relevant dependency between the variables and the collected data was not normally distributed according to performed tests, we decided to use the Wilcoxon signed-rank test [10] for further inferential analysis.

In all our statistical tests, we accept a 5% probability of committing type-I-error. To analyze the collected data we used the open source R package[6].

IV. Results and Discussion

In the following section, we provide answers to the questions from Section III-B. The discussion is supported with the analysis of the data collected during the controlled updates.

A. Q1. Update Correctness

To address question Q1 we use the *Correctness of an update* metric (I1).

TABLE I
DESCRIPTIVE STATISTICS FOR THE UPDATE CORRECTNESS.

		correct	incorrect
S_A	architectural	17	0
	baseline	14	3
S_B	architectural	17	0
	baseline	3	14

I1 - Correctness of an Update: Table I shows the frequencies of correct and incorrect updates for simple S_A and complex S_B scenarios. All updates supported with the architecture-centric approach were performed correctly. For baseline supported scenarios 82% of simple and 18% of complex tasks were correct.

We used two statistical tests to check the relevance of the observed results:

$H_{0-simple}$: There is no difference in the correctness of updates between the baseline and architecture-centric approaches for simple scenarios.

[6]http://www.r-project.org/

$H_{0-complex}$: There is no difference in the correctness of updates between the baseline and architecture-centric approaches for complex scenarios.

The statistical tests reveal that the measured difference is statistically significant for complex scenarios ($p - value = 0.000105$). For simple scenarios the hypothesis cannot be rejected ($p - value = 0.07446$).

Discussion: Correctness of updates is one of the most important requirements for integration tasks. It is important to notice that the update tool does not automate the resources manipulation phase. With both approaches, all resource manipulations had to be performed manually. Therefore, the chance of mistakes in an update task due to an incidental resource modification are similar for both approaches. The data analysis reveals that the usage of the architecture-centric approach improves the correctness of the updates, in particular for complex scenarios in which the updates affect multiple subsystems. Whereas all tool supported updates were performed correct, only three participants were able to perform all updates correct using the baseline approach.

B. Q2. Logistic System Availability

The number of unique process shutdowns (I2), total process shutdowns (I3), and the time for performing updates (I4) are used to address question Q2.

I2 - Number of Unique Process Shutdowns: For simple scenarios (S_A), 65% of the subjects performed all required shutdowns with the baseline approach. Nevertheless, the mean value for this metric is 0.41 as shown in the Table II. In contrast, the number of unique process shutdowns is optimal for all simple updates performed with the architecture-centric approach.

For complex scenarios (S_B), the value for metric I2 decreases for both approaches. We observed that 41% of the subjects performed the optimal number of unique shutdowns using the baseline approach, whereas 96% of them performed the optimal number of shutdowns using the architecture-centric approach. In addition, the average number of shutdowns for baseline is 0.94, which means that in almost all update tasks the subjects performed unnecessary shutdowns. To compare, the average value for the architecture-centric updates is 0.06.

TABLE II
DESCRIPTIVE STATISTICS ON UNIQUE PROCESS SHUTDOWNS.

		mean	median	std.dev	range
S_A	architectural	0.00	0.00	0.00	0.00
	baseline	0.41	0.00	0.62	2.00
S_B	architectural	0.06	0.00	0.24	1.00
	baseline	0.94	1.00	1.03	3.00

We statistically tested the significance of the results with the following hypotheses:

$H_{0-simple}$: There is no difference in the number of unique shutdowns between the baseline and architecture-centric approaches for simple scenarios.

$H_{0-complex}$: There is no difference in number of unique shutdowns between the baseline and architecture-centric approaches for complex scenarios.

The results show that the difference in the number of shutdowns are significant for both simple ($p - value = 0.01313$) and complex ($p - value = 0.002367$) scenarios.

I3 - Number of Process Shutdowns: Table III shows the differences in the average number of shutdowns and the variation of results between the approaches both for simple (S_A) and complex (S_B) scenarios.

In addition, with the baseline approach, the subjects performed the exact number of process shutdowns for updating the products in 41% of the simple and 24% of the complex scenarios. The usage of the update tool resulted in 88% and 82% of exact shutdowns respectively.

TABLE III
DESCRIPTIVE STATISTICS ON PROCESS SHUTDOWNS.

		mean	median	std.dev	range
S_A	architectural	0.18	0.00	0.53	2.00
	baseline	1.47	1.00	2.03	6.00
S_B	architectural	0.24	0.00	0.56	2.00
	baseline	3.12	3.00	2.96	11.00

Although the difference in number of shutdowns between the approaches is clearly visible, we tested the observations with following hypotheses:

$H_{0-simple}$: There is no difference in number of shutdowns between the baseline and architecture-centric approaches for simple scenarios.

$H_{0-complex}$: There is no difference in number of shutdowns between the baseline and architecture-centric approaches for complex scenarios.

The statistical tests reveal that the difference between the methods is significant for both simple ($p - value = 0.01051$) and complex ($p - value = 0.0007364$) scenarios.

I4 - Time: The overall time spent on the updates is summarized in Figure 4. When an intervention of the supervisor was needed (e.g. to restore the initial state of the system, which we allowed in the experiment), we excluded this time from the total update time. The average time spent on baseline updates is 11 minutes for simple scenarios (S_A) and 23 minutes for complex scenarios (S_B). Similarly, the average times of architecture-centric updates are 7.5 and 14.5 minutes respectively. The variability (in S_B) of the results with the baseline approach is approximately four times the variability of the results with the architecture-centric approach.

To check the significance of the observed results we performed two statistical tests:

$H_{0-simple}$ There is no difference between both update approaches in the time needed to complete the simple update tasks.

$H_{0-complex}$ There is no difference between both approaches in the time needed to complete the complex update tasks.

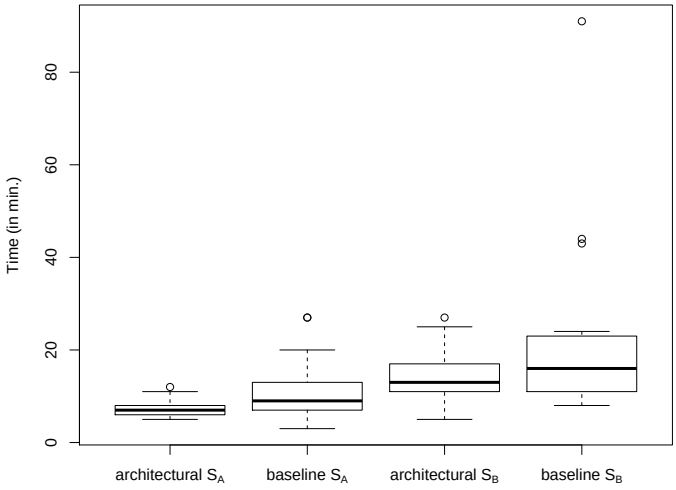

Fig. 4. Plots of the updates durations for simple S_A and complex S_B scenarios.

The results ($p - value = 0.02759$ for $H_{0-simple}$ and $p - value = 0.08153$ for $H_{0-complex}$) show that the differences in update times is only significant for simple scenarios.

Discussion: The number of unique process shutdowns expresses how accurate the subjects were in stopping the right processes during the update tasks. The number of process shutdowns can be used to estimate the total unavailability of the given subsystems. Both metrics accommodate the processes that had to be stopped to perform the update correctly but remained running. The fact that such a process was not stopped indicates an incorrect update. Such a situation occurred in 41% of the complex and 6% of the simple updates for the baseline approach.

The difference between the approaches is clearly visible in both metrics. We identified two possible explanations for these differences. First, we observed several "modify and verify" cycles with baseline updates. In other words, a subject modified the system and then checked correctness of the applied changes by simple end-user tests. This is a risky strategy, while running the system in an incorrect configuration may lead to data loses or worse. Second, we observed that subjects had a tendency to shut down more process as needed to perform an update with the baseline approach. Although, we observed superfluous shutdowns in case of architecture-centric updates too, these shutdowns were rare in comparison to the baseline updates.

We use the time metric to measure the duration of updates. Although, the architecture-centric updates at average took less time than the baseline updates, the difference is relevant only in case of the simple scenario. Due to the experimental settings, the time needed for the subsystems to restart was an order of magnitude smaller than in real system updates. Therefore, the number of process shutdowns seems to be a better estimator of the availability of the production system under update.

C. Q3. Confidence Level

We decided to check the confidence level in two ways: first directly by asking the subjects questions via questionnaires (I5), and send indirectly by looking at the file modification the subjects made during updates (I6).

I5 - Confidence Level: Confidence level is a metric that characterizes the belief of a subject that the performed update was correct. The 5-point Likert scale was used to examine the subjects' level of agreement with the statement "I am sure that I performed the update correctly". Results are presented in Figure 5. The bars indicate the number of responses for the different levels of agreement. The actual correctness of the updates is marked with two colors. The brighter part of a bar represents the number of incorrect updates, whereas darker part represents the number of correct updates. For example, for complex scenarios (S_B) performed with the baseline approach, three subjects state that they are confident that update were performed correctly. However, only one of subjects was right.

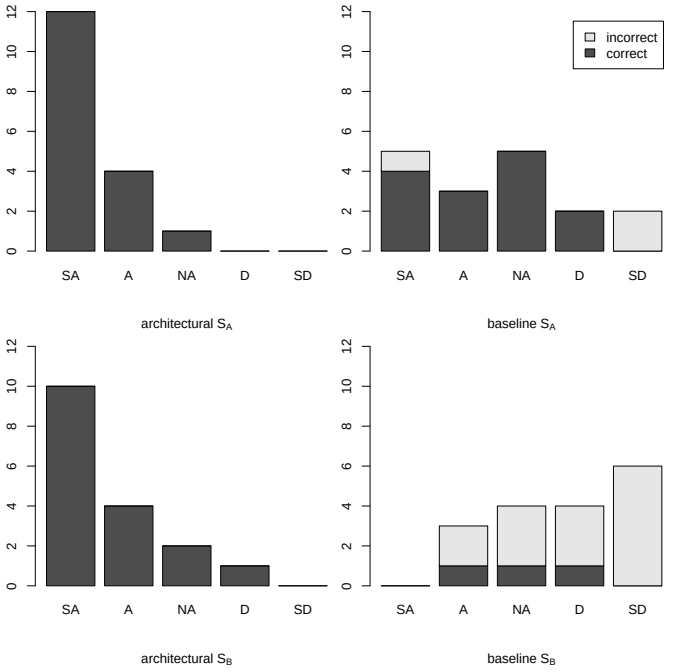

Fig. 5. Level of confidence about update correctness (SD - strongly agree, A - agree, NA - I am not sure, D - disagree, SD -strongly disagree). The actual correctness of updates is marked in colors (brighter - incorrect, darker - correct).

We checked the significance of the difference in the level of confidence using the following hypotheses:

$H_{0-simple}$ There is no difference in the subjects' level of confidence about the simple updates correctness between the architecture-centric and baseline approaches.

$H_{0-complex}$ There is no difference in the subjects' level of confidence about the complex updates correctness between the architecture-centric and baseline approaches.

The statistical tests confirm significance of the differences between the approaches for both scenario kinds (simple: $p - value = 0.002724$; complex: $p - value = 0.0002168$).

I6 - File Modifications Ratio: The file modification ratio expresses the relation between the optimal and the actual number of the file modifications for a given update task. The results for both approaches are plotted in Figure 6. The average value of the file modifications ratio for the architecture-centric approach is 1.02 for both simple (S_A) and complex scenarios (S_B). The value for the baseline approach is 10.04 for simple and 9.71 for complex scenarios. A high diversity of the file modification results is visible for update tasks performed with the baseline approach.

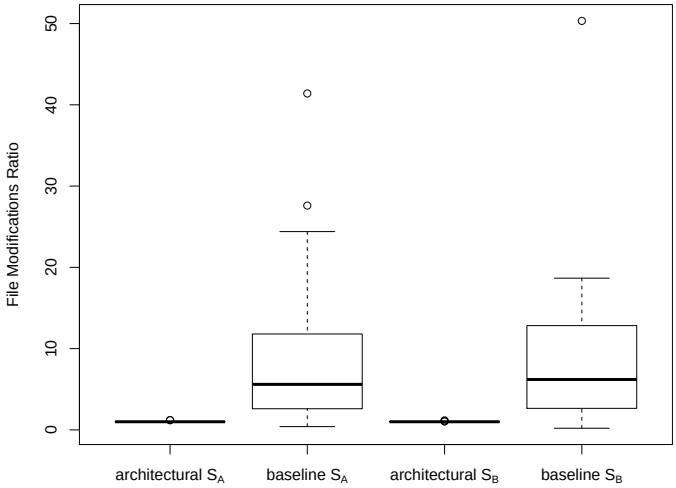

Fig. 6. Plots of the file modifications ratio for the two evaluated approaches both for simple (S_A) and complex (S_B) scenarios.

We used two hypotheses to test the significance of the differences in file modifications ratios:

$H_{0-simple}$ There is no difference in number of file modifications between the architecture-centric and baseline approaches for simple scenarios.

$H_{0-complex}$ There is no difference in number of file modifications between the architecture-centric and baseline approaches for complex scenarios.

The results, $p-value = 0.000274$ for simple and $p-value = 0.0004596$ for complex scenarios, shows that the hypothesis can be rejected for both cases.

Discussion: The results show that subjects are confident about the update correctness in updates performed with the tool (16 confirmatory answers for simple S_A and 14 for complex S_B scenarios). These results are aligned with the actual correctness of the updates, and further confirmed by the participants comments. For instance, one of the participants reported that the tool *"tells exactly what to do and where to find the files"*. The analysis of file modification ratio supports this finding. The results show that the subjects are guided by the instructions from the update tool, as the number of file modifications is optimal or close to optimal for all updates.

In contrast, the lack of required knowledge during updates with the baseline approach affects the level of confidence about the update correctness. The subjective opinions about the correctness of the update are diverged and in some cases

different from the actual state. For example, in a simple scenario one of the subjects was strongly convinced about the correctness of an update, but in reality it was performed incorrectly. In a production settings this can lead to the inoperative system configuration, or in extreme case, to the damages of an industrial installation.

V. Threats to Validity

In this section validity threats are discussed. We use the classification described by Wohlin et al. [21].

A. Internal Validity

1) Instrumentation: It can happen that an artifact used in the experiment may affect the observed results. To avoid this risk we ran three pilot experiments to check the suitability of the materials for the experiments. As we explained above, we use the insights from the pilot to adjust some of the material for the experiment. However, not all elements that can affect the results might be eliminated. Some subjects reported that the software running at a virtual machine was too slow. This might affect some of the observed results (e.g. metric I4). However, this issue will have a similar effect for all performed updates.

2) Maturation: In the study we used two experiments with paired design, where subjects performed four controlled updates. Therefore, the risk that the learning effect influences results of the treatments is high. To mitigate this risk we introduced the learning scenarios and randomized the order of the update tasks for all subjects.

B. External Validity

1) Interaction of Setting and Treatment: According to the taxonomy proposed by Zelkovitz [22] our experiment is an example of synthesis. The goal of the synthesis is to replicate a simple version of the technology to be able to measure the selected properties.

In our experimental settings we have used a real logistic system, but were unable to reconstruct the complete industrial installation. People might react differently when no real customers are involved or faulty updates to logistic machines cannot cause real damage. We tried to anticipated this threat by explaining the subjects that minimal shutdowns and correctness were key requirements for the updates.

In addition, the experiment did not allow the test person to interact with other people. In practice, integrators may contact colleagues during updates. Note that both approaches received an equal treatment. Moreover, it is known that fixing errors by calling colleagues only helps in a small number of cases.

C. Construct Validity

1) Mono-operation Bias: In the experiments we used only one logistic system, although with different variants. This introduces a risk that the Egemin's logistic systems were under-represented. However, the system used in the experiment contains most important subsystems Egemin's SPL. In addition, we examined the effectiveness of the approaches at two complexity levels that represent the majority of update scenarios of industrial installations.

D. Conclusion Validity

1) Reliability of Treatment Implementation: The subjects were supervised by different experimenters. Being aware of this fact, we have defined a detailed experimentation procedure which was followed in all experiments. In addition, we have automated the setup of an experimentation environment and by tracking several of the observed variables. Finally, three experimenters participated in the pilot experiments.

VI. Related Work

Despite the successes of SPL adoption, empirical verification of SPL related theories and processes is rather limited. Similarly to other domains [22], most of the findings result from case studies. Linden et al. [13] uses several industrial case studies to illustrate the benefits of SPL. Pöhl [15] summarizes experience reports related to SPL adoption in 15 organizations.

Case studies are also widely used in research, addressing an architecture-oriented view on SPLs. Bosh [3] reports on two industrial case studies involving Swedish organizations. The information from architects' reviews were analyzed to understand the state of the practice and identify issues related to the SPL architecture. Two other case studies are provided by Deelstra et al. [6], who report on product derivation issues of Thales Nederland B.V. and Robert Bosch GmbH SPLs. A simple case study on a product line of calculators was used to study the application of Maintainability Index to the SPLs [2].

In this paper we report on a controlled experiment with an architecture-centric approach to address update problems in the context of SPL evolution. Although our work is rooted in SPL, in this paper we focus on the updates of single products derived from a SPL.

A controlled experiment on migration of the COBOL software to the web is presented by Colosimo [5]. In the experiment, the effectiveness of the migration supported by the MELIS tool is compared to the more traditional approach. The focus of our work is similar to this paper. We compare the effectiveness of the product updates between the architectural and baseline approaches. We contribute with a comparative experiment which was performed in a professional settings, with respect to both, the subject and object of the experiment.

The architectural approach presented in this paper is inspired by research in the Software Architecture Reconstruction (SAR) domain. The general overview of SAR approaches in provided by Ducasse and Pollet [7]. However, differently from most code based approaches(e.g. [1]) we use the meta-data to build architectural models.

VII. Conclusions

Based on the findings of the experiment, we conclude that the proposed architecture-centric approach is a significant improvement comparing to Egemin's traditional update practices. The automatic constructed of the architecture models of deployment products is sufficient to *correctly* migrate Egemin's

products. We also observe a significant improvement in the *availability* of services during product updates.

By running the experiment and providing the results, we were able to convince the decision makers at Egemin that the update process can be successfully improved with the architectural approach.

The proposed solution has a potential to reduce the costs of the product updates. First, all subjects performed the updates correctly using the tool. This eliminates the potential costs of repairs which are required in case errors are revealed after updates. Second, the subjects were able to perform these updates without consulting their colleagues. One of the Egemin integrators mentioned that *"even the customer could perform updates this way [using the tool]"*. Third, the presented analysis of the data shows that subjects trust the architectural knowledge provided by the tool. Moreover, 15 of them reported that they have a strong preference for the architectural approach. With the architectural approach, the size of the integration team, as well as involvement of the key engineers in the updates can be reduced significantly.

Nevertheless, the proposed solution, as well as its evaluation have some limitations. The tool used in the experiment is a prototype, therefore some improvements are necessary before it can be used in a production environment. One of the most obvious improvements is (semi-)automation of the update steps. In addition, we have only examined the applicability of the solution for Egemin products. Although, we designed the tool with the extensibility to other domains in mind, the version of the tool used in the experiment is tailored to address the specific needs of Egemin. The cost of the architecture-centric approach adoption is twofold. First, the proposed meta-model might require some modifications to be applicable to another domains. Second, even without modification of the meta-model, the domain specific harvesters must be developed to allow for the data collection. Therefore, at this stage, the conclusions we draw cannot be generalized beyond Egemin's SPL context.

Although we successfully applied the architecture-centric approach to Egemin's SPL, several research challenges remain. First, the extensibility of the approach could be evaluated in a repeated experiment with a different SPL. Second, we plan to formally proof the correctness of the algorithms we use for deriving the update steps and checking consistency.

ACKNOWLEDGMENTS

We are grateful to professor Victor R. Basili for his valuable comments on the experiment design. We thank Kurt de Vocht for the inspiring discussions and his support in preparing the update scenarios. Finally, we thank the Egemin employees and our colleagues from DistriNet labs for their participation in the experiment.

REFERENCES

[1] S. Ajila and A. Kaba. Evolution support mechanisms for software product line process. *Journal on Systems and Software*, 81(10):1784–1801, 2008.

[2] G. Aldekoa, S. Trujillo, G. Sagardui, and O. Diaz. Quantifying maintainability in feature oriented product lines. In *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*, pages 243 –247, 2008.

[3] J. Bosch. Product-line architectures in industry: a case study. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pages 544 –554, May 1999.

[4] P. C. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, August 2001.

[5] M. Colosimo, A. D. Lucia, G. Scanniello, and G. Tortora. Evaluating legacy system migration technologies through empirical studies. *Information and Software Technology*, 51(2):433 – 447, 2009.

[6] S. Deelstra, M. Sinnema, and J. Bosch. Product derivation in software product families: a case study. *Journal of Systems and Software*, 74(2):173 – 194, 2005. The new context for software engineering education and training.

[7] S. Ducasse and D. Pollet. Software architecture reconstruction: A process-oriented taxonomy. *IEEE Trans. on Software Engineering*, 35(4):573 –591, 2009.

[8] J. Estublier, I. A. Dieng, and T. Leveque. Software product line evolution: the selecta system. In *Proceedings of the 2010 ICSE Workshop on Product Line Approaches in Software Engineering*, PLEASE '10, pages 32–39, New York, NY, USA, 2010. ACM.

[9] J. Gibbons and S. Chakraborti. *Nonparametric statistical inference*, volume 168. CRC Press, 2003.

[10] M. Hollander and D. Wolfe. *Nonparametric statistical methods*. Wiley-Interscience, 1999.

[11] ISO/IEC. Systems and software engineering - architecture description. *ISO/IEC standard, draft D8*, August 2010.

[12] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55, 1932.

[13] F. J. Linden, K. Schmid, and E. Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer, 1 edition, July 2007.

[14] B. Michalik, N. Boucke, D. Weyns, and A. Helleboogh. *Empirical Evaluation of EvoLine*. Katholieke Universiteit Leuven, 2011. TR. Available via http://people.cs.kuleuven.be/danny.weyns/EvoLineEvaluation.pdf.

[15] K. Pohl, G. Böckle, and F. Van Der Linden. *Software product line engineering: foundations, principles, and techniques*. Springer-Verlag New York Inc, 2005.

[16] K. Schmid and M. Verlage. The economic impact of product line adoption and evolution. *Software, IEEE*, 19(4):50 – 57, jul/aug 2002.

[17] S. Shapiro and M. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3-4):591, 1965.

[18] F. van der Linden, K. Schmid, and E. Rommes. *Software product lines in action: the best industrial practice in product line engineering*. Springer-Verlag New York Inc, 2007.

[19] D. Weyns and B. Michalik. Codifying Architecture Knowledge to Support Online Evolution of Software Product Lines. In *Sharing and Reusing Architectural Knowledge, 2011. SHARK'11. ICSE Workshop on*. ACM, 2011.

[20] D. Weyns, B. Michalik, A. Helleboogh, and N. Boucké. An architectural approach to support online updates of software product lines. In P. Kellenberger, editor, *2011 Ninth Working Conference on Software Architecture*. IEEE, June 2011.

[21] C. Wohlin, P. Runeson, and M. Höst. *Experimentation in software engineering: an introduction*. Springer Netherlands, 2000.

[22] M. V. Zelkowitz. An update to experimental models for validating computer technology. *Journal of Systems and Software*, 82(3):373 – 376, 2009.