Change-Impact driven Agile Architecting

Jessica Díaz, Jennifer Pérez, Juan Garbajosa, Agustín Yagüe

Abstract

Software architecture is a key factor to scale up Agile Software Development (ASD) in large software-intensive systems. Currently, software architectures are more often approached through mechanisms that enable to incrementally design and evolve software architectures (aka. agile architecting). Agile architecting should be a light-weight decision-making process, which could be achieved by providing knowledge to assist agile architects in reasoning about changes. This paper presents the novel solution of using change-impact knowledge as the main driver for agile architecting. The solution consists of a Change Impact Analysis technique and a set of models to assist agile architects in the change (decision-making) process by retrieving the change-impact architectural knowledge resulting from adding or changing features iteration after iteration. To validate our approach, we have put our solution into practice by running a project of a metering management system in electric power networks in an i-smart software factory.

1. Introduction

Software architecture is a key factor to scale up Agile Software Development (ASD) in large software-intensive systems. Several works propose the coexistence of software architectures and ASD [1] [3][5][13][14][19], and a few approaches present successful cases of *agile architecture* [18] or *iterative architecture* [8]. Agile architecture can defined as *"the one that develops with the system, and includes only features that are necessary for the current iteration or delivery"* [8]. However, how to perform this iterative architecture refinement is still a challenge [1]. This challenge is addressed in this paper.

Aligning fruitfully software architectures and ASD requires leveraging the inherent qualities of software architectures (e.g. abstraction, communication, analysis) while complying with agile principles (e.g. open to change). This alignment can be achieved as long as practitioners are able to count on mechanisms for enabling: (i) Incremental design of features, i.e. flexible construction of the architecture by adding small increments[1]. (ii) Accommodation of new features or customizations on existing features. We refer to both of them as *agile architecting*, because although conceptually different, require the same mechanisms to carry them out. The reason is that, in both cases, these mechanisms must be able to cope with change, though in the first case the change is planned (feature increment), and in the second case the change is unplanned (feature evolution).

It would be highly convenient and desirable that the mechanisms for enabling agile architecting would assist and guide agile architects, specifically in (i) the decision-making process of implementing changes in each agile iteration, and (ii) the maintenance of the architecture integrity, i.e. the preservation of earlier architectural design decisions iteration after iteration. Regarding the former, the knowledge about the effects of a change upon the architecture provides architects with information that can be advantageously deployed to reason about how and where to implement that change. It also allows architects to make better evolution decisions based on risks, cost or viability of the change. Regarding the latter, the continuous process of architecting should never result in the software degradation as a consequence of intentionally or accidentally violation of earlier design decisions or constraints. In this sense, agile architects need knowledge about dependencies between design decisions, constraints, tradeoffs, etc., which can assist them in countering or even avoiding several well-known negative effects of software evolution such as architectural erosion and degradation [25].

This paper presents the novel solution of using *change-impact* architectural knowledge as the main driver for agile architecting. This solution provides agile architects with knowledge to (i) assist and guide them in the *change (decision-making) process*, and

[1] An increment is often smaller than a feature —prominent or distinctive user-visible characteristic or quality of a software system.

(ii) favor the preservation of the architecture integrity during the iterative architecting process. This knowledge results from analyzing the impact that changes —feature increment and/or evolution— introduce into the architecture, iteration after iteration in an agile process. The solution consists of a *Change Impact Analysis* (CIA) technique and modeling artifacts for: (i) documenting architectural knowledge —the design decisions and rationale driving the iterative architecture solution—, and (ii) tracing architecturally significant features with their realization in the architecture. These models are traversed using the proposed CIA technique to retrieve the architectural design decisions and architectural components and connections that are impacted as a consequence of changing features. This solution is implemented in a modeling framework called FPLA[2].

The novelty of this paper is to prove how the output from a CIA technique can be effectively used to assist and guide agile software architecting. This CIA technique was deployed in the agile method Scrum [29] and built on the results from previous works [23][11] that provide flexible mechanisms to design iteratively and incrementally software architectures.

To empirically validate our approach we have conducted a case study in an i-smart software factory, combining both academic and industry efforts. The results show that our approach for agile architecting is viable in an industry project in the energy power networks domain, and effectively assists and guides architects in the tasks of making-decisions about changes and maintaining the architecture integrity.

The structure of the paper is as follows: Section 2 describes the background. Section 3 discusses related work. Section 4 presents the CIA technique, and supporting mechanisms, which drive agile architecting in the Scrum process. Section 5 describes the case study. Finally, conclusions and further work are presented in Section 6.

2. Background

2.1 Agile Architecting

The role of software architecture in ASD has been a highly controversial issue in the last few years. There are many advocates for and opponents against giving to architectures the importance in ASD that it has in other development approaches. Advocates of the architecture's key role in the software process have their doubts about the scalability of any development approach that does not pay sufficient attention to architecture [1], specially for achieving quality goals when developing large-scale software-intensive systems. In fact, Cockburn [10] showed some data about the unfeasibility of using agile methods in large size projects and life-critical systems. The reason is that the benefits of software architecture are missing and agile teams completely depend on tacit knowledge. The work of Falessi et al. [14] found that agile practitioners perceive software architecture as relevant on the basis of aspects such as communication and understanding of software systems, rationalization of previous design decisions, documentation of rationale necessary to evaluate design alternatives, scaling of agile practices to large projects, documentation of points of flexibility within the system to support future requirements, and system planning and budgeting.

On the contrary, hard opponents perceive the effort in architecture as wasted effort, equating it with big upfront (BDUF) —a bad thing-leading to massive documentation and implementation of *you ain't gonna need it* features [1]. A common belief within the agile community is that *"If you are sufficiently agile, you don't need an architecture — you can always refactor it on the fly"* [10]. However, Kruchten states that architectural refactoring often becomes prohibitively costly very quickly if certain considerations have been neglected early in the process (excerpted from [13]). Kruchten [19] and Booch [5], among others, propose the iterative and incremental evolution of the architecture to reduce the big upfront design and keep the system in sync with changing conditions.

2.2 Change Impact Analysis

Change impact analysis (CIA) determines the potential effects upon a system resulting from a proposed change [2]. CIA can be used to predict the effects of a change before it is implemented, possibly giving an estimate of the effort/cost to implement the change [27], as well as the potential risk involved in making the change [21]. This analysis can be then used to make better evolution decisions such as whether or not the change should be carried out based on economic viability of software evolution or other risks such as degradation of software systems. In fact, there is an extensive work in CIA to support software evolution [7][9], although Mens at al. [21] identified change impact as one of the future challenges (timeframe of 2015 and beyond).

3. Related Work

Advocates of a balance between architecture and agility propose that the architecture emerges gradually iteration after iteration, as a result of successive small

[2] It is available on https://syst.eui.upm.es/FPLA/home

refactoring [1][5][19][22]. Most of these approaches invest in a first architecture —zero-feature release [4]; i.e. *"getting an architecture sufficiently right early without necessarily resorting to big upfront design"*[19]. This means that it will take longer to get to code, i.e. in a zero-feature release the architecture is in place but no user-visible features are delivered to the customer [22]. Conversely, other authors believe in *continuous architectural refactoring* starting on simplicity and flexibility [5].

This paper does not focus on identifying whether it is better to invest in a first architecture or to rely on continuous architectural refactoring. This paper presents the mechanisms to have flexibility at the time of defining software architectures and change-impact knowledge in order to support the change decision-making process and preserve architecture integrity.

Change impact analysis has not been previously applied to agile architecting as we propose in this paper. Moreover, it is not only novel the fact of applying change-impact knowledge to drive agile architecting. The CIA technique that we use in this paper covers several of the lacks of current CIA approaches. Most CIA approaches analyze the source code and few approaches do the architecture [30][16]. Even fewer approaches consider architectural knowledge, that is design decisions and rationale driving the architecture solution, to aid change impact analysis [31]. To cope with these lacks, a previous work [11] defined a CIA technique in the domain of Software Product Lines (SPL [26]). As discussed in Section 4, agile architects can take advantage of this technique to support the change decision-making process and try to preserve the architecture integrity.

4. Agile architecting guided by change impact

This paper presents CIA as the main driver for agile architecting. To that end, we have defined a CIA technique, supported by architectural models, that assists architects during the agile architecting process. These models promote communication between individuals and agile teams working on the system, and support (semi-)automatically reasoning over the space of architectural knowledge. They are described below.

4.1 Flexible-PLA Metamodel

Our solution is supported by the definition of software architectures conforms to the Flexible-PLA Metamodel [24]. It was defined in a previous work to explicitly specify the architectures that realize SPL. This metamodel and their underlying concepts allow one to iteratively and incrementally construct and evolve software architectures based on two properties that they provide: flexibility and adaptability [23][12].

The main concept underlying Flexible-PLA Metamodel is the concept of Plastic Partial Component (PPC [24]). The PPC concept was originally defined for specifying variability inside components. The variability of a PPC is specified using *variability points*, which hook fragments of code to the PPC known as *variants*, and *weavings* which specify where and when extending the PPCs using the variants (see Figure 1.a). As variability facilitates the planned evolutionary software development [15], agile architects can take advantage of the PPC primitives for incrementally and iteratively refine the architectural components that compose a *working architecture*[3] [23]. The PPC variability mechanism is the backbone to support incremental development of architectural components through the incomplete specification of components, and their extension by hooking new variants. As a result, working architectures can be incrementally and iteratively designed and evolved in each iteration by weaving/unweaving extensions, and/or by modifying the architecture configuration through optional components and connectors.

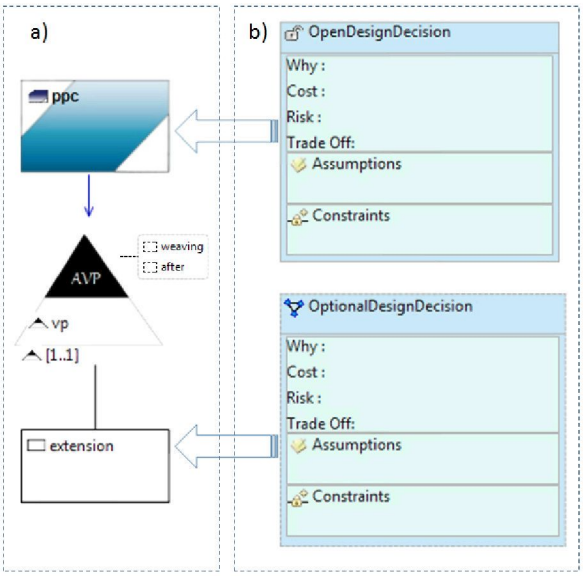

Figure 1. a) Flexible-PLA model & b) PLAK model

4.2 PLAK Metamodel

The documentation of architectural knowledge (AK) supports the rationalization of architectural decisions taken during the solution design. The rationalization of early design decision may help to evolve the architecture while preserving its integrity.

[3] The one that is delivered after each iteration, together the working product, as a result of the agile architecting process.

The main types of AK are the design decisions driving the architecture solution, their dependencies and rationale. We have previously defined the concept of *Product-Line Architectural Knowledge* which is formalized through the PLAK Metamodel [11]. This paper extends the PLAK Metamodel and their modeling primitives (rationale, constraints, assumptions, etc. [11]) to capture the knowledge of adding feature increments or changing features in each agile iteration. This extension consists of the following primitives:

- *Closed design decisions* (Closed DDs) are completely closed (or bound) in a given iteration and support the realization of those features that can be completed in one iteration and that architects considered unchanging in time.
- *Open design decisions* (Open DDs) are intentionally left open (or delayed) and support the realization of those features that cannot be completed in one iteration and that architects plan to complete iteration after iteration (see the design decision with the open lock of Figure 1.b). Open DDs consist of a set of optional design decisions.
- *Optional design decisions* (Optional DDs) support each of the increments of an Open DD (see the optional design decision of Figure 1.b).
- *Alternative design decisions* (Alternative DDs) support the alternative realization of Closed and Open DDs, respectively.

These four types of DDs offer a complete support for documenting the knowledge derived from the agile architecting process. The PLAK Metamodel supports the documentation of flexible and adaptive architectures through decisions which can be intentionally left delayed, and later closed in following iterations.

DDs completely or partially realize features, affecting multiple architectural components and connectors, and often become intimately intertwined with other DDs [6]. In this regard, the PLAK Metamodel supports dependencies between DDs and traceability between features and their realization into architectures. Traceability links are created around the DD in such a way that they turn into the links between feature and architecture models. Additionally, the PLAK Metamodel defines the *linkage rules* that comprise the semantics to bridge the gap between feature and architecture primitives.

4.3 CIA Technique

Changes in features impact the system architecture and can lead to ripple-effects which are not obvious to detect. In this work, we adapt the CIA technique that we previously defined for SPL [11], to analyze the impact of adding or changing features in each iteration of the agile life-cycle. This CIA technique consists of a *traceability-based algorithm* and a *rule-based inference engine,* which traverse Flexible-PLA and PLAK models (see Section 4.1 and 4.2, respectively) based on a set of traceability links and propagation rules. The process, that this CIA technique implements, consists of two main steps described below:

(1) Given a change in features (adding, deleting or updating), the *traceability-based algorithm* determines (i) the *first-order* design decisions that are involved with the feature to be changed, (ii) the *n-order* DDs that depend on the first-order DDs, and (iii) the first-order architectural elements (PPCs, components, and connectors) that are involved in each (first and n-order) DD. The algorithm traverses the traceability links that bridge features and architectural elements, and the dependency relationships between design decisions.

(2) Given a change in the working architecture that realizes the change in features, the *rule-based inference engine* fires propagation rules to obtain the change propagation in the working architecture. Namely, when a modification over the working architecture is applied to, propagation rules are fired to simulate the effects on the rest of the working architecture. We thereby obtain the *n-order* architectural elements that are impacted by the change.

As previously mentioned in Section 4.1 and 4.2, this paper relies on the use of the concept of variability to iteratively and incrementally construct/evolve software architectures, as long as this variability is documented and traced through Open and Optional DDs. The CIA technique described within this section, fits perfectly the needs of analyzing the impact of agile architecting, as it traverses Flexible-PLA and PLAK underlying concepts which support architectural variability. The output of the CIA technique provides change-impact architectural knowledge that may be useful for reasoning about a proposed change in features and guiding the change decision-making process, as well as may help to preserve the architecture integrity iteration after iteration of the agile life-cycle.

4.4 Agile Architecting process

Our solution of using CIA as the main driver for agile architecting is deployed in the agile method Scrum [29]. Figure 2 shows a tailored Scrum development process in which agile architecting is considered as a key activity to prepare the iteration (aka. sprint). Briefly, the process is described below.

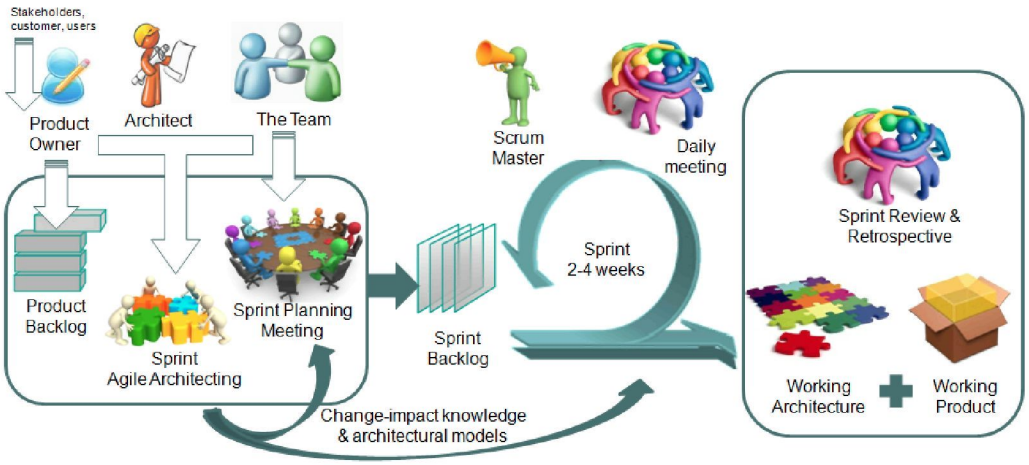

Figure 2. A tailored Scrum with Agile architecting

The first step consists of capturing the requirements of the *Product Owner* from the product vision (features). Features may be decomposed into a list of *user stories* (US) known as *product backlog*. Then, US are prioritized, based on business value, and assigned to *sprints*. Scrum implements an iterative lifecycle based on these sprints. Sprints start with *sprint planning meeting* in which the Product Owner and Team plan together what has to be done. In this tailored Scrum, the *agile architecting* tasks are developed in conjunction with the sprint planning meetings (see Figure 2). The abovementioned modeling artifacts and the CIA technique drive agile architects as follows:

- The Flexible-PLA Metamodel provides agile architects with primitives to iteratively and incrementally construct and evolve *working architectures*. Changes are realized by using the extension mechanism of PPCs or/and changing the architecture configuration.
- The PLAK Metamodel provides agile architects with primitives to document design decisions, dependencies, constraints, tradeoffs, etc. as well as to trace features to working architectures.
- The CIA technique is applied to the *working architecture* of the previous sprint (except for the first sprint). It provides agile architects with the change-impact knowledge resulting from the changes planned for the sprint. From the impacted components and connections, the architects can reason about where and how implementing that change. From the impacted design decisions, the architects are aware of the effects of that change over previous constraints, tradeoffs, risks, and hence they have knowledge to preserve the architecture integrity.

As a result of this tailoring, agile architects interact with the rest of the team in planning the features to be done by tracking architectural concerns —constraints, risks, viability, etc.— and balancing them with business priorities. The output of sprint planning meetings consists of: the *sprint backlog* and tasks that must be performed to achieve the sprint goal, and the working architecture models (Flexible-PLA and PLAK models) to be implemented during the sprint (see Figure 2). At the end of each sprint, a *working product* and a *working architecture* are delivered. In the sprint *review meeting* the Product Owner assesses the working product to validate that US were met or introduce changes into the US.

5. Case study

With the purpose of validating our approach for agile architecting, we have conducted a case study within a project in an experimental i-smart software factory (iSSF) [20]. Following subsections briefly introduce some information about the environment for conducting the case study, i.e. the iSSF, and describe the case study according to the guidelines for conducting and reporting case study research in software engineering of Runeson & Höst [28].

5.1 Context: Running Environment

The iSSF is a software engineering research and education setting in close cooperation with the top industrial and research collaborators in Europe [20]. Indra Software Labs[4] leads this initiative at the corporate level in Spain in conjunction with the Technical University of Madrid (UPM). The iSSF facility continuously runs projects in eight-seven week cycles. In this case study we focused in a project to develop a *metering management system in electric*

[4] http://www.indracompany.com/en

power networks. In total, 10 people participated in the case study: four developers, two product owners, one scrum master (who performs both the tasks of the Scrum master and the tasks of an architect at part-time), one full-time architect and two observers. The observers had access to all project information and collaborated directly with product owners and fellow team members.

5.2 Research questions

The goal of the case study is to search evidence in order to answer the following two research questions: **RQ1**: Is the CIA algorithm effective in locating the impacted architectural design decisions and elements resulting from a proposed change in features? **RQ2**: Does the CIA algorithm assist and guide agile teams in reasoning about the changes and preserving the architecture integrity while agile architecting?

To answer RQ1 the architects analyzed change impact as follows: First, architects manually analyzed change impact without the CIA technique and the supporting models that this paper presented. Then, the architects analyzed change impact with the assistance of the FPLA modeling framework that implements the CIA algorithm and the Flexible-PLA and PLAK primitives. This procedure allowed to verify if the CIA algorithm had determined impacts that architects manually did not, or vice versa if the architects had manually determined impacts that the CIA technique did not. From this procedure, it is possible to define the dependent variable for quantitatively measuring RQ5: the percentage of impacts that the CIA algorithm automatically determines that are not manually determined by architects. The potential independent variable that might have an influence on the dependent variable is the total number impacts that exist given the proposed change(s) and the architects experience.

To answer RQ2 two properties were measured: the assistance and the guidance in making decisions about the changes and in maintaining the architecture integrity during the iterations of an agile process. These properties are hard to measure and especially hard to quantify. Therefore, they are estimated by qualitatively analyzing a questioner that architects filled and comments expressed by the architects while they were realizing the architecture.

5.3 Data collection

The collection methods used to gather quantitative and qualitative data are described as follows:

- Observation. Two observers attend the agile architecting, planning and review meetings which are video recorded, transcribed, and analyzed.
- Interview. The architects are interviewed following a questionnaire open to the discussion. These interviews are video recorded, transcribed, and analyzed using the *constant comparison method* [17].
- Archival data. The information about the project is collected in Redmine[5].
- Analysis of work artifacts. The data of Flexible-PLA and PLAK models generated with the FPLA framework, the CIA algorithm running also under FPLA, and the code under subversion[6] are gathered.
- Metrics. The metrics captured by Sonar[7] are collected at the end of each sprint.

5.4. Analysis procedure

In this case study, both quantitative and qualitative analysis were used to examine the data gathered. For quantitative data, this case study uses analysis of descriptive statistics. As qualitative data is typically less precise than quantitative data, it is important to use *triangulation* to increase the precision of the study. There are several types of triangulation [28], e.g. methodological, data source. The three types of triangulation were used in this case study.

5.5. Case study description

The case study consists of a project to develop a metering management system in electric power networks. The metering management system is part a larger ITEA project called *Intelligent Monitoring of Power NETworks* (IMPONET[8]) that focuses on supporting complex and advanced requirements in energy management, specifically electric power networks. IMPONET aims for (i) continuous monitoring and bi-directional communication with customers to promote sustainability, and (ii) prevention of congestions, faults, and peak loads in real-time. The metering management system captures and manages meter data from a huge number of distributed energy resources. The overview of the system functionality is as follows:

- *Meter capturing*: integrates all meter capturing processes that collect meter data at substations.
- *Meter processing*: perform operations for validating meter data according to a validation formula and calculating the optimal vector for a measuring point.
- *Meter providing*. It defines the interface with client systems to provide exchanging data (e.g. billing and settlements, energy demand forecast, etc).

[5] Redmine is web-based project management and bug-tracking tool.
[6] Subversion is an open source version control system.
[7] Sonar is an open platform to manage code quality.
[8] http://innovationenergy.org/imponet/

5.6. Case execution

The agile architecting process was performed as described in Section 4.4. Due to space reasons, we focus on the features and working architectures which were realized in five sprints (over eight sprints). These features are described as follows:

- *F1_Meter reading* consists of reading metering data associated to different energy resources, periods (quarterly, hourly, daily and monthly) and dates. Metering data is provided by text files.
- *F2_Meter storing* consists of a large data store running over an object-oriented NoSQL database (*Big Data Oracle* running over *BerkeleyDB)*.
- *F3_Meter data access* consists of the initial data loading of historical metering data of one month, and query of these data. Both loading and querying require to leverage a high performance through the use of clustering technologies (*Apache Hadoop*).
- *F4_ Meter data process* includes the algorithms for validating raw and optimal data, as well as calculating the optimal vector (integrated processing) of raw and optimal data.
- *F5_ Graphical interface* consists of the interface that provides the query of metering data.

These features were progressively decomposed into US and the backlog was created. **Sprint 1** focused on implementing feature F2 (Meter storing), i.e. the installation and configuration of the database manager (Berkeley DB), and several conceptual proofs were realized to create and access to the database (see component *DBManager* and *DD001* in Figure 3). **Sprint 2** focused on implementing features F1 (Meter reading) and F3 (Meter data access). At this time, the working architecture of Sprint 1 was in too early of a stage that the CIA algorithm did not provide any relevant result that could have assisted architects in the decision-making process of adding these features. Figure 3 shows the working architecture resulting from Srpint 2: The component *MeterCapturer* implements the reading text files of metering data and processing the previously read data to form pair of key/value. The PPCs *DataLoader* and *DataQuery* implement data loading and data query, respectively. As the functionalities for data loading and data query could not be completely implemented in this sprint, and increments in following sprints could refine these components, the architects decided to implement them using PPCs. The design decision *DD002* keeps the rationale behind reading metering data in pairs of key/value. Finally, between *DD002* and *DD001* there is a dependency that keeps the relation between the need of reading data in pairs key/value and the use of the database manager *BerkeleyDB*. **Sprint 3** focused on completing F3 and implementing F4 (Meter data process). The CIA retrieved the design decisions and components which could have been impacted as a consequence of adding the new features on the current working architecture —i.e. the working architecture from Sprint 2 (see Figure 3): the PPCs *DataQuery* and *DataLoader*. Figure 4 shows the working architecture resulting from Sprint 3: The extension *Hadoop MAP/REDUCE* implements the operations for clustering and distributing work around a cluster in order to improve the performance for data accessing (data loading and data query). The PPCs *DataLoader* and *DataQuery* were extended with this functionality through the variability point clustering (see Figure 4). The design decision *DD005* keeps the rationale behind clustering as well as a dependency with the design decision *DD001* (see Figure 4). Finally, the PPC *MeterProcessor* implements initial operations over raw and optimal data to calculate the optimal vector. As the functionality for meter processing could not be completely implemented in this sprint and increments in following sprints might refine this component, the architects decided to implement it using a PPC (see Figure 4). **Sprint 4** focused on completing F4. The CIA retrieved the design decisions and components which could have been impacted as a consequence of adding this increment on the current working architecture —i.e. the working architecture from Sprint 3 (see Figure 4): the PPC *MeterProcessor*. Then, the working architecture from Sprint 3, specifically the PPC *MeterProcessor*, was extended with the algorithms for validating metering data and calculating optimal vectors. Finally, **Sprint 5** focused on implement feature F5. This feature has no a priori dependencies with others, so the CIA algorithm did not retrieve any impact.

5.7 Analysis and interpretation

Quantitative and qualitative analysis was used to examine the data gathered as described in Section 5.3. The research questions RQ1 and RQ2 are answered as follows:

RQ1*: Is the CIA algorithm effective in locating the impacted architectural design decisions and elements resulting from a proposed change in requirements?* In Sprints 3 and 4, the CIA algorithm is 100% effective in locating the design decisions and components which are impacted by the feature increments that are planned during the sprint planning meetings. The complexity of the working architectures resulting from these first sprints is relatively low, in such a way that the architects also determined 100% of the impacted design decisions and components.

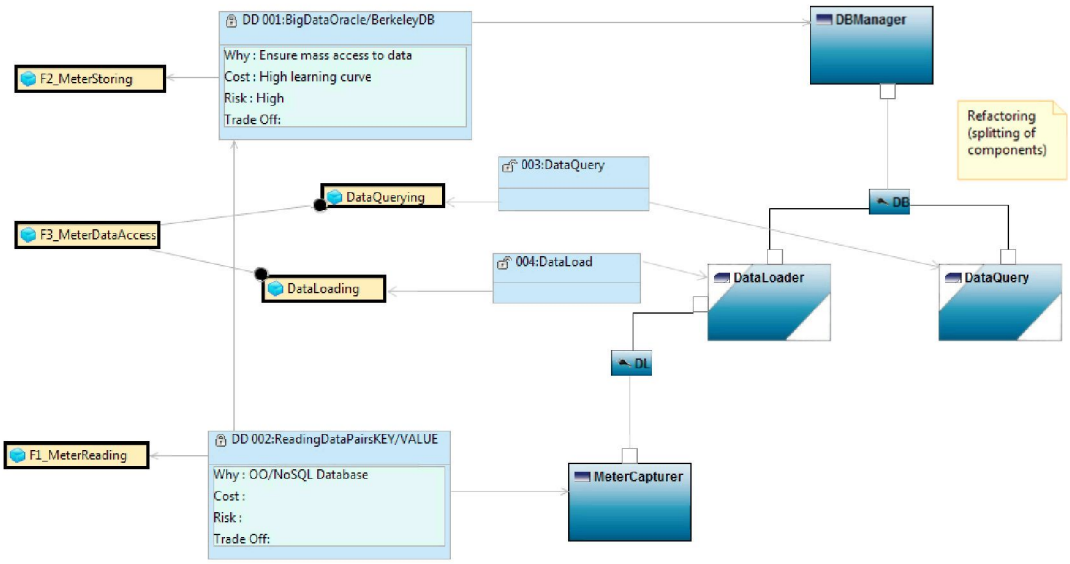

Figure 3. Sprint 2 (Flexible-PLA & PLAK models)

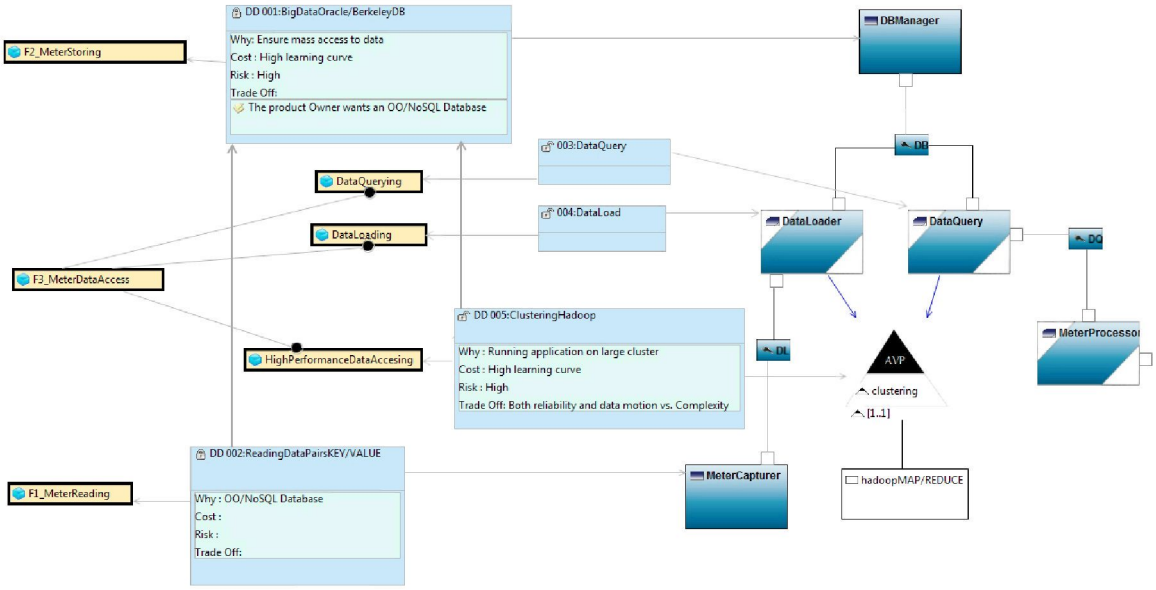

Figure 4. Sprint 3 (Flexible-PLA & PLAK models)

At that time, the project continued for more four sprints but two members of the Scrum team were substituted[9]. In Sprint 6, the product owner required the following conceptual proof: a change of the database manager with the aim of evaluating the performance of other solutions. Specifically, the product owner decided to evaluate *Oracle Real Application Clusters* (RAC[10]) over *Oracle 11g*. The architects ran the CIA algorithm to traverse the architectural model of Sprint 5 and retrieve the design decisions and components which could have been impacted by this feature change. The algorithm retrieved five impacts: the PPCs *DataLoader* and *DataQuery*, the variability point *clustering*, and the components *DBManager* and *MeterCapturer* (see row 2, Sprint 6 of Table 1). In this retrieval, two dependencies between architectural design decisions participate in the propagation of the change (see dependencies between *DD005* and *DD001*, and *DD002* and *DD001* in Figure 4). The manual analysis performed by the architects did not determine the propagation of the change. This is due to the fact that the dependencies between the *DD005* and *DD002* with *DD001* became hidden for the new architect, so that they only found that three components could have been impacted (see row 3, Sprint 6 of Table 1).

[9] One architect and one developer

[10] Clustering and high availability in Oracle db environments

Therefore, the CIA algorithm automatically retrieved two components which were not manually determined by architects. Namely, the percentage of impacts that the CIA algorithm automatically determined and that were not manually determined by the architects, is 40%. Hence, the 40% of DDs and components, which are impacted by the change, would have stayed hidden if the CIA algorithm had not worked effectively.

Table 1. Change-impact results

	Sprint 3	Sprint 4	Sprint 5	Sprint 6
Total number of impacts	2	1	0	5
CIA algorithm	100%	100%	100%	100%
Manual CIA	100%	100%	100%	60%

***RQ2**: Do PLAK and the CIA algorithm assist and guide agile team in reasoning about the changes and preserving the architecture integrity while agile architecting?* Analyzing the interviews to the architects, the following excerpts can be highlighted: <<The use of PLAK models was particularly useful to understand the system during staff turnover that took place between Sprints 4 and 5>> <<Without the knowledge provided by the PLAK model and the CIA algorithm, it would be extremely difficult to reason about the impact of changing the database manager>><<It's likely that several components had remained unchanged [...] we had quite likely implemented the necessary functionality in other or new components, so the former would had had "dead code">><<Dead code is an indicator of bad smell>><<This bad smell could have been identified after we had implemented the change by analyzing the dashboard of sonar platform>><<By using PLAK models and the CIA algorithm we were able to proactively determine all the impacts and to avoid this software degradation>><<we took previous design decisions into account among which there were dependencies>><<it did allow us to maintain the architecture integrity>>. These excerpts put in evidence that our approach for agile architecting assisted and guided the architects in the decision-making process about changes and in the tasks of maintaining the architecture integrity iteration after iteration of this Scrum development.

5.8 Evaluation of validity and limitation

Case studies are qualitative in nature. The objective judgment of the collected data of this kind is not possible. To improve the internal validity of the results presented, the independent variables that could influence this case study have been identified as follows: The architect experience has a great influence. This has been reduced because the expertise of the two architects who participate in the case study is very different (1 year vs. 7 years). However, the influence of project's size and architecture's complexity cannot be reduced due to the inherent nature of case studies, which normally focus on one project. Additionally, we have used triangulation of source data to increase the reliability of the results. In this regard, interviews were individually conducted with the two architects, although several questions were asked in a group setting to encourage discussion. However, the major limitation in case study research is concerning to external validity, i.e. "the generality of the results with respect to a specific population" [17], as only one case is studied. In return, case studies allow one to evaluate a phenomenon, a model, or a process in a real setting. This is something important in software engineering in which a multitude of external factor may affect to the validation results, and that other techniques such as formal experiments, although they permit replication and generalization, do not consider as they are conducting under controlled settings.

6 CONCLUSIONS

This paper presents a novel solution to drive agile architecting. Our solution assists and guides agile architects in the decision- making process at the time of adding or changing features by: (i) tracking the effects of changes upon the architecture —components and connections which are impacted by the changes—, and (ii) analyzing architectural concerns such as dependencies with earlier design decisions, rationale, constraints, risks, etc. which are also impacted by the changes. This analysis enables the preservation of the software architecture, as long as it helps to reduce the risk of unexpected consequences of changes. The analysis is instrumented by a CIA technique and deployed in the Scrum method.

The viability of our approach is proved by a case study which has been run in an experimental laboratory called i-Smart Software Factory. It combines both academic and industry efforts in R&D, being remarkable the facilities for tracking the projects' progress. The case study puts our solution into practice within the agile development of a metering management system in the electric power networks domain. The results prove that (i) the CIA algorithm is effective in locating the impact resulting from a change, and (ii) this change-impact helps architects to take better decisions, especially when architectural knowledge may be lost or vaporized as a result of a staff turnover. Hence, the case study proved that the CIA algorithm determined 40% of impacts more than architects manually determined. These promising

results did not interfere with other agile practices and did not incur a big upfront design, making the agile construction and evolution of architecture possible. One of the main conclusions of this research is that agile architecting is feasible so that ASD can be scaled up to large and complex software-intensive systems.

As future work, we are improving the FPLA framework, so that it can semi-automatically generate code by using *model-to-text* transformations. Its aim is to link architecture and code. The traceability between architecture and implementation may avoid another common problem in ASD when code drifts so far apart that it makes it much easier to erode the software system architecture.

ACKNOWLEDGMENT

The work reported in here has been partially sponsored by the Spanish fund (INNOSEP TIN2009-13849, IMPONET TSI-020400-2010-103, i-SSF IPT-430000-2010-038) and UPM (Technical University of Madrid) under their Researcher Training program.

10. References

[1] P. Abrahamsson, M. Babar, and P. Kruchten, "Agility and architecture: Can they coexist?" Software, IEEE, vol. 27, no. 2, pp. 16–22, 2010.

[2] R. S. Arnold, Software Change Impact Analysis. Los Alamitos, CA, USA: IEEE Computer Society Press, 1996.

[3] M. A. Babar and P. Abrahamsson, "Architecture-centric methods and agile approaches," in Agile Processes in Software Engineering and Extreme Programming (XP), 2008, pp. 242–243.

[4] K. Beck, Extreme Programming Explained: Embrace Change. 2nd ed. Addison-Wesley Professional, 2004.

[5] G. Booch, "The defenestration of superfluous architectural accoutrements," Keynote Software Architecture Challenges in the 21st Century, USC, June 2009.

[6] J. Bosch, "Software architecture: The next step," in Software Architecture, ser. LNCS, vol. 3047. Springer, 2004, pp. 194–199.

[7] C.-Y. Chen and P.-C. Chen, "A holistic approach to managing software change impact," J. Syst. Softw., vol. 82, no. 12, pp. 2051–2067, 2009.

[8] H. Chivers, R. F. Paige, and X. Ge, "Agile security using an incremental security architecture," in Proceedings of the 6th International Conference on Extreme Programming, ser. LNCS. Springer, 2005, pp. 57–65.

[9] H. Cho, J. Gray, Y. Cai, S. Wong, and T. Xie, Model-Driven Domain Analysis and Software Development: Architectures and Functions. IGI Global, 2011, ch. Model-Driven Impact Analysis of Software Product Lines, pp. 275–303.

[10] A. Cockburn, Agile Software Development. The Cooperative Game. Second Edition. Addison-Wesley Professional, 2006.

[11] J. Díaz, J. Pérez, J. Garbajosa, and A. Wolf, "Change impact analysis in product-line architectures," in Proceedings of the 5th European Conference on Software Architecture, LNCS, vol. 6903. Springer, 2011, pp. 114–129.

[12] J. Díaz, J. Pérez, J. Garbajosa, and A. Wolf, "A process for documenting variability design rationale of flexible and adaptive PLAs," in On the Move to Meaningful Internet Systems: OTM 2011 Workshops, ser. LNCS, vol.7046. Springer Berlin / Heidelberg, 2011, pp. 612–621.

[13] H. Erdogmus, "Architecture meets agility," Software, IEEE, vol. 26, no. 5, pp. 2–4, 2009.

[14] D. Falessi, et al., "Peaceful coexistence: Agile developer perspectives on software architecture," Software, IEEE, vol. 27, no. 2, pp. 23–25, 2010.

[15] M. Galster and P. Avgeriou, "Handling variability in software architecture: Problems and implications," in Proceedings of the 9th WICSA '11, 2011, pp. 171–180.

[16] M. O. Hassan, L. Deruelle, and H. Basson, "A knowledge-based system for change impact analysis on software architecture," in Proceedings of Fourth International Conference on Research Challenges in Information Science, 2010, pp. 545–556.

[17] U. van Heesch, P. Avgeriou, and R. Hilliard, "A documentation frame- work for architecture decisions," Journal of Systems and Software, vol. 85, no. 4, pp. 795–820, 2012.

[18] T. Ihme and P. Abrahamsson, "Agile architecting: The use of architectural patterns in mobile java applications," Int. J. of Agile Manufacturing, vol. 8, no. 2, pp. 97–112, 2005.

[19] P. Kruchten, "Software architecture and agile software development an oxymoron?" Keynote Software Architecture Challenges in the 21st Century, USC, June 2009.

[20] J. L. Martin, A. Yagüe, E. Gonzalez, and J. Garbajosa, "Making software factory truly global: the smart software factory project," in Software Factory Magazine. Available on http://www.softwarefactory.cc/magazine F. Fagerholm, Ed., March 2010, p. 19.

[21] T. Mens, S. Demeyer, and T. Mens, "Introduction and roadmap: History and challenges of software evolution," in Software Evolution. Springer Berlin Heidelberg, 2008, pp. 1–11.

[22] R. L. Nord and J. E. Tomayko, "Software architecture-centric methods and agile development," IEEE Softw., vol. 23, no. 2, pp. 47–53, 2006.

[23] J. Pérez, J. Díaz, J. Garbajosa, and P. P. Alarcón, "Flexible working architectures: Agile architecting using PPCs," in Proceedings of the 4th European Conference on Software Architecture, LNCS, vol. 6285. Springer-Verlag, 2010, pp. 102–117.

[24] J. Pérez, J. Díaz, C. C. Soria, and J. Garbajosa, "Plastic partial components: A solution to support variability in architectural components," in Proceedings of the Joint WICSA/ECSA. IEEE Computer Society Press, 2009, pp. 221–230.

[25] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," SIGSOFT Softw. Eng. Notes, vol. 17, no. 4, pp. 40–52, 1992.

[26] K. Pohl, G. Bckle, and F. Linden, Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Germany, 2005.

[27] J. Ramil and M. Lehman, "Metrics of software evolution as effort predictors - a case study," in Software Maintenance, 2000. Proceedings. International Conference on, 2000, pp. 163–172.

[28] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," Empirical Software Engineer- ing, vol. 14, pp. 131–164, 2009.

[29] K. Schwaber and M. Beedle, Agile Software Development with Scrum.Prentice-Hall, 2002.

[30] A. Tang, A. Nicholson, Y. Jin, and J. Han, "Using bayesian belief networks for change impact analysis in architecture design," J. Syst. Softw., vol. 80, pp. 127–148, 2007.

[31] A. Tang, Y. Jin, and J. Han, "A rationale-based architecture model for design traceability and reasoning," J. Syst. Softw., vol. 80, pp. 918–934, 2007.