Guiding Flexibility Investment in Agile Architecting

Carlos Fernández-Sánchez, Jessica Díaz, Jennifer Pérez, Juan Garbajosa
Technical University of Madrid (UPM), CITSEM, E.U. Informática, Madrid, Spain
{carlos.fernandez, yesica.diaz}@upm.es, {jenifer-perez, jgs}@eui.upm.es

Abstract
Agile software development pursues to deal with continuous change. But software product architectures without enough flexibility can restrict how products cope with change. However, designing for flexibility often entails high costs and risk that comes with the assumption that change will happen. Actually, in software architecture the flexibility investment decision making problem has become challenging. This paper presents a process to assist architects in MAKing dEcisions about Flexibility investment in Software Architecture (MAKEFlexi). MAKEFlexi is based on technical debt and real options approaches. Technical debt allows for estimating the additional cost derived from the lack of flexibility in software architectures, whereas the real options valuation allows for estimating the value of the flexibility that a design option could provide. MAKEFlexi has been applied to an industry project for smart grids to assist architects in making decisions about designing for flexibility to vary data storage technologies.

1. Introduction
Turbulence in the current business climate makes change inevitable if a company is to remain competitive. Change adoption is part of the business environment, and the capacity to respond to this situation by harnessing change, rather than avoiding it, is especially critical in the software industry [11]. Therefore, a key issue for software companies is that their people and the software products they produce must be open to change. Agile values aim “to respond to change over following a plan” [4]. However, such kind of flexibility in the process could be insufficient if our software products are not open to change. Software architecture is one of the main drivers in software product development and evolution since it is considered to be the bridge between requirements and implementation [26]. The software architecture of products open to change must be ready for change, so that architecting can be aligned with the welcome to change agile principle. That has motivated a growing recognition of the importance of having a good architecture design in agile software development [1].

Software architecture can be considered as a set of architectural design decisions [5]. These decisions are based on information that is current at the time the decisions are made, however decisions makers can also consider possible future changes. Increasing flexibility is one of the decision drivers for coping with change in software architecture [6]. Flexibility is defined as the ability to deal with changes that can be anticipated or expected [7] i.e., changes that can come or not come with some probability over some period of time. Designing for flexibility involves the design of an architecture that facilitates change [6], usually by defining variation points [7]. Therefore, flexibility is an important factor for success because it may reduce the product development time and the cost of incorporating expected changes into the business over the lifetime of the software.

However, making the software architecture flexible often entails high costs in terms of time and effort [30]; it also entails a risk associated with the assumption that change will happen. This means that designing for flexibility is an investment in a trade-off between short-term costs versus longer-term benefits. In this regard, it is necessary to evaluate when the Return on Investment (ROI) of designing for flexibility would be positive. Toward that end, it would be desirable to have models that allow one to decide whether or not it is profitable to invest in designing for flexibility at any given time. In this paper, we refer to “invest in flexibility” as the design/refactoring of architecture in order to add variation points to support expected changes.

This paper presents a process to obtain a decision tree model that assists architects in MAKing dEcisions about investments in design for Flexibility (which we refer to as MAKEFlexi); i.e., MAKEFlexi assists architects in deciding when to invest (or not) in flexibility to support expected changes. MAKEFlexi is based on technical debt [33] and real options [16] approaches. Technical debt makes the cost of quality weaknesses in software visible. Specifically, MAKEFlexi uses technical debt to make visible the additional cost derived from the lack of flexibility to accommodate expected changes over some period of time. In addition to debt,
it is necessary to consider the value that flexibility could provide to architecture design solutions to accommodate expected changes. In this regard, MAKEFlexi uses the real options valuation [16] to estimate this value as a function of the (i) uncertainty or probability that changes could happen, and (ii) the period of time in which the need to introduce changes could happen. This value is used to decide if to invest in flexibility, and when to do so. This means that, depending on the expected changes, it could be more profitable to delay the investment in flexibility—i.e., take technical debt—or invest in flexibility in order to pay off the debt.

We have applied the MAKEFlexi process in a joint project with industry in the area of power networks grids. The resulting decision tree model has effectively guided architects in the tasks of making decisions about flexibility.

The structure of the paper is as follows: Section 2 describes the background upon which the MAKEFlexi process is based. Section 3 describes MAKEFlexi. Section 4 presents the case study in which the process has been validated. Section 5 discusses related work. Finally, conclusions and further work are presented in Section 6.

2. Background

This section describes the two approaches upon which MAKEFlexi is based: technical debt and real options.

2.1. Technical debt

Technical debt is a metaphor that refers to the consequences of weak software development. It was first introduced by Cunningham [9]. This metaphor has been used during the past several years as a means of making the intrinsic cost of internal quality weaknesses visible in software. Quality weaknesses in software can be reworked to reduce this debt. A review of the literature about technical debt shows that it has several dimensions [33]:

- Code debt: technical debt that manifests in the form of poorly written code.
- Design and architectural debt: technical debt caused by shortcuts and shortcomings in design and architecture.
- Environmental debt: technical debt that manifests in the environment of an application; i.e., development processes, hardware, infrastructure, and supporting applications.
- Knowledge distribution and documentation debt: technical debt caused by lack of knowledge transmission or the absence of well written documentation.
- Testing debt: technical debt in the form of insufficient automated tests resulting in a waste of time due to the use of manual tests. Insufficient test coverage is another way to incur testing debt.

All these dimensions are managed by analyzing technical debt items, also known as instances [14]. We will use the term item in this paper. A technical debt item represents an “incomplete task” or a task that is not perfectly performed and one that might cause problems in the future. These are considered to be quality weaknesses. Two main concepts are widely used to make technical debt items visible: principal and interest [10]. Principal is the cost of eliminating—or reducing—the impact of a technical debt item at a given time in a software system, whereas interest is the recurring cost of not eliminating a technical debt item over some period of time.

This paper focuses on architectural debt, i.e., technical debt caused by the result of sub-optimal upfront architecture design solutions, or solutions that become sub-optimal as technologies and patterns become superseded [33]. An example of architectural debt item is an immature design architecture solution that could cause future additional cost when the system has to grow or be changed. The principal of this technical debt item can be estimated as the cost of improving the design by increasing software architecture flexibility (e.g., the cost of refactoring the software architecture to enable change), whereas the interest is the additional cost derived from the lack of flexibility (e.g., the additional cost of implementing changes if the software architecture has not been previously designed to be open to changes).

2.2 Real options

Real options valuation is a technique that arises from financial options valuation. “A financial option gives the holder of the option the right to do something, but the holder does not have to exercise that right” [16]. Financial options valuation assesses the value of those options. Real options valuation is an extension of financial options that uses financial valuation methods to assess capital investment opportunities in real assets under uncertainty [16]; that is, to estimate the value of options to determine whether or not to make the investment. Real options valuation has been successfully applied to solve software engineering problems [2] [12] [15] [23] and has been previously proposed to manage technical debt [6]. This paper uses a similar approach to make decisions about flexibility in software architecture.

Real options valuation is based on the estimation of the investment value evolution over some period of time by using probabilistic mathematical methods.
Therefore, the real options approach makes use of estimated values, such as the investment cost, the value added by the investment, and probabilities of investment evolution, as inputs. In this regard, the real options approach considers the uncertainty of that evolution as well as the period of time in order to exercise the investment option. Estimated values and uncertainty are mainly obtained from the experts’ opinion and from historical data.

There are different kinds of real options [16]. The MAKEFlexi process focuses on the expansion option, i.e., an option to make further investments and increase the output if conditions are favorable, specifically, investments to improve the architecture of a software system. In addition, it is important to mention that the MAKEFlexi process uses the decision tree method [32] to assess the value of real options. As an example, Figure 1 graphically shows the estimation of the value of an option by using a decision tree. The option is the investment in improving the flexibility of a software architecture to deal with a set of expected changes. Specifically, the software product has to undergo two expected changes over some period of time, and its software architecture should be ready to deal with those changes.

The decision tree method consists of two main steps. The first step involves the construction of a tree that represents (i) the evolution of the software product due to expected changes over some period of time, and (ii) the decisions as to whether or not to invest in preparing the architecture to accommodate the expected changes. Figure 1 shows the decision tree over a two-month period. This tree is characterized by three kinds on nodes: decision nodes, change nodes, and end nodes. Decision nodes, which are represented by rectangles in Figure 1, are points where a decision on architecture has to be made. Change nodes, which are represented by circles in Figure 1, are points where expected changes of the software product could come. Finally, end nodes, which are represented by triangles in Figure 1, show the final state of the software product after analyzing all possible scenarios resulting from the expected changes that may or may not occur and decisions that may or may not be made. These nodes make up the branches of decision trees that have an associated probability (e.g., the probability used in this example is 0.33 for all branches, which in this example is assumed that is given by the opinions of experts; see Figure 1).

The second step in this process consists of the assessment of the value of the option (i.e., the value of the option of investing in flexibility). The value of the option is the investment’s expected net value. This investment’s expected net value is computed by folding back the decision tree, starting with the end nodes and moving toward the root. This step is composed of two activities. The first activity consists of estimating the net value of end nodes (triangles)—i.e., the value of the software product after implementing the expected changes of the branch under analysis minus the cost of implementing those changes. For example, in Figure 1 considering that experts have estimated that the value added by an expected change is €10K, the cost of investing in flexibility is 2K, the cost of a change without a previous investment in flexibility is 3K, and the cost of a change with a previous investment in flexibility is 6K (see the Figure 1 caption, the end node marked with A has a net value of €12K [= (€20K = 10K * 2 changes) – 2K – (6K = 3K * 2 changes)], while the end node marked with B has a net value of €8K [= (€20K = 10K * 2 changes) – (12K = 6K*2 changes)]. The second activity consists of estimating the expected net value of the decision nodes and the change nodes. The estimation of the decision nodes (rectangles) consists of selecting the branch with the highest net value, while the estimation of the change nodes (circles) is the sum of the net value of the branches weighted by its probability. For example, in Figure 1 the decision node marked with C has a net value of 5K = max (5K; 4K) while the change node marked with D has an expected net value of €4290 = (8K*0.33) + (5K*0.33) + (0*0.33). This activity ends by estimating the value of the root of the tree (a decision node). This value is used to decide whether or not to invest in flexibility, and when to do so. In this particular example, the value of the root is €4950 = max (€4950; €4290). This means that the best decision is to invest in flexibility from month 1 and not to delay the decision.
3. The MAKEFlexi Process

This section presents the MAKEFlexi process used to obtain a decision tree model to make decisions regarding flexibility investment in software architecture. MAKEFlexi is performed as a part of the Backlog grooming sessions [18]. Backlog grooming sessions give agile teams the opportunity to look further into the future of the product, and can alert them to technical challenges. The process consists of a set of steps to create a model, based on decision trees to estimate when to design for flexibility. Each step is described as follows.

- **Step 1**: Analysis of expected changes. In this step, the architect identifies the expected changes that are to be supported through flexibility. In addition to the expected changes, it is necessary to determine: (i) the time frame in which the expected changes could happen, (ii) the probability that the expected changes could happen, and (iii) the value added by implementing the expected changes. This information has to be provided by experts. Depending on the expected changes different actors can be implied in the estimation of the probability of change. If the changes will depend on new contracts, then the responsible of such contracts will be who has the information. If the changes will depend on the adoption of a new technology, then the responsible of such technical decision will be who knows the needed information. Therefore, this probability of change will be subjective. Also, it is possible to use historical data about similar expected changes to estimate how often changes are made in the software.

- **Step 2**: Design of alternative architecture solutions to support the expected changes. In this step, architects propose alternatives with different degrees of flexibility (i.e., different amounts of investment) that are to be analyzed.

- **Step 3**: Estimation of technical debt for each one of the alternatives. The goal of estimating technical debt is to calculate the cost of implementing or not implementing the flexibility. To estimate the technical debt we use the concepts of principal and interest. In this case the principal of a technical debt item derived from the lack of flexibility is the cost of improving flexibility, usually through the definition of variation points. To estimate the principal it is necessary to detect the weaknesses in the system. Static code analysis has been successfully used for that task. For example, tools such as Sonar (http://www.sonarsource.org/) and PMD (http://pmd.sourceforge.net/) can detect architecture anti-patterns in software projects. Knowing the weaknesses, experts can estimate the cost of solving them by applying commonly used cost estimation techniques such as COCOMO or Function Point. The interest of a technical debt item derived from lack of flexibility is the additional cost of implementing the expected changes—additional Cost of Change (CoC)—if the item is not previously eliminated; i.e., if the software architecture is not flexible enough to introduce the change. However, estimating this interest is not easy because there is no reference for an ideal CoC; i.e., a solution with zero interest (see ideal solution line in Figure 2). The interest of an architecture design solution is the difference between the CoC of the ideal solution and the CoC of the alternatives (see “a” and “a + b” in Figure 2). Since the CoC of the ideal solution cannot be easily estimated, it is possible to use the comparison between the CoC of the different software architecture alternatives (see “b” in Figure 2). Therefore, the MAKEFlexi process uses the CoC for estimating the interest of the different alternatives of change for architectural support. Knowing the current architecture anti-patterns in the project, historical data can be used to compare the effort needed to change components implementing anti-patterns, with the effort needed to change components without such weaknesses. If historical data are available, it is possible to estimate the difference between the needed effort to perform changes in component with anti-patterns and component without them. As a result, the different alternatives can be compared when choosing which option is the most profitable. Finally, Step 2 and Step 3 can be performed iteratively in order to architects can propose architectures that solve the weaknesses detected in Step 3.

- **Step 4**: Construction of the decision tree. In this step the architect performs the construction of a tree that represents the evolution of the software system due to expected changes, such as the tree shown in Figure 1. To do that, the following inputs are used:
The expected changes, their probability, the time frame, and the value that these changes add to the software product under analysis (Step 1).

- Alternative design architecture solutions (Step 2).
- The constraints imposed by deadlines, resources, and development time needed to implement the expected changes and prepare the architecture to support the expected changes (i.e., designing for flexibility). This means that alternatives that cannot be performed in the time frame, given such constraints, are not considered in the decision tree. For example, in Figure 1 an end node marked with B has been modeled to represent a scenario in which sufficient time is not allotted to both invest in flexibility and implement the expected changes.

The construction of the decision tree requires one to determine the decision nodes, change nodes, and end nodes.

- **Step 5**: Assessment of the expected net value of the flexibility investment option. To make this assessment, the following inputs are used:
  - The estimations of the principal and the interest of each of the alternative architecture design solutions (see Step 3).
  - The decision tree (see Step 4).

In this step the model computes the investment’s expected net value by folding back the decision tree, starting with the end nodes and moving toward the root (see Section 2.2). This computation consists of: (i) estimating the net value of the end nodes, and (ii) estimating the expected net value of the decision nodes and the change nodes, until arriving at the root node.

- **Step 6**: Make the decision. Finally, the final decision is the investment option with the higher than expected net value from the root node. The value of the root node is used to decide whether or not to invest in flexibility, and when to do so. In this regard, it is possible that the model suggests delaying the decision until more information is available.

## 4. Case Study

A case study was conducted to provide empirical evidence that validates that it is feasible to use the MAKEFlexi process to guide flexibility investment in software architecture in an industry project. Case study research is a technique that consists of the investigation of contemporary phenomena in their natural context to search for evidence, gain understanding, or test theories by primarily using qualitative analysis [27]. In addition, we provide in this section guidance for how to use the MAKEFlexi process so it can be reproduced in other projects.

The case study was conducted in an experimental i-smart software factory (iSSF) [20]. The iSSF is a software engineering research and education setting utilized by the top industrial and research collaborators in Europe [20]. Indra Software Labs leads this initiative at the corporate level in Spain in conjunction with the Technical University of Madrid. The case study was performed within a project to develop a family of metering data management systems in electric power networks for smart grids [21], called Optimeter. This project was developed using Scrum methodology [28]. This project is part of two European ITEA2 projects: IMPONET1 and NEMO & CODED2.

The following section presents the case study based on the guidelines for reporting case study research in software engineering by Runeson et al.3 [27].

### 4.1. Case study description

Smart grids require the ability to manage different resources, from renewal or traditional energy producers to energy consumers. The case study performed on the Optimeter project consisted of the development of a family of software systems that manage meter data from a huge number of these resources. A metering management system captures meter data from telemetering systems or batch processes, loads these data into a database, supports data querying and processing, and provides these data to other systems for billing, forecasting, or purchasing. The customer stated that the high performance of data access and storage was essential to the success of the product. In order to manage meter data with high performance, it was necessary to evaluate several of the large data storage technologies available in the market (see Table 1).

The Optimeter project started with a proof-of-concept to evaluate Oracle Big Data (see data storage technology 1 in Table 1). Over a one-month period (two sprints, each sprint implied two weeks), the team developed a product that implemented a solution based on the data storage technology Oracle Big Data. Figure 3.a shows the architecture that had arisen over that product development. At that time, the architects needed to decide whether or not to invest in flexibility in order to support the variation of the data storage technology, i.e., to support a family of products that support the different variants regarding the data storage technology. This is due to there was a level of uncertainty regarding a future change of the data storage technology to satisfy the restrictions of data accessing

---

1. [http://innovationenergy.org/imponet/](http://innovationenergy.org/imponet/)

2. [http://innovationenergy.org/nemocoded/](http://innovationenergy.org/nemocoded/)

3. Not all the information of the report is described in this paper due to space limitations.
performance (see data storage technologies 2-4 in Table 1). In this regard, the architects considered the following expected-change scenarios: the possibility of varying one, two, or the three additional data storage technologies, as shown in Table 1. At that moment, the customer did not know how many of these candidate data storage technologies had to be evaluated in terms of the performance of massive data loading and querying; nor did the customer know what other business and commercial matters needed to be considered. Due to this uncertainty, it was necessary to obtain information about the ROI for preparing the architecture in order to vary the data storage technology (i.e., the value-generating power of flexibility).

4.2. Case study execution

During Optimeter execution, the MAKEFlexi process was applied as follows:

Step 1: Analysis of the expected changes. The expected change in the family of metering management systems under analysis is the variation of the data storage technology (Table 1). Hence, the family of metering management systems will implement three, two, one, or none of the data storage technologies variants. The time frame in which the expected change can happen is three months (the Optimeter project lasted four months, but one month had already been spent on implementing a solution based on the data storage technology 1, see Table 1). The probability of the expected-change scenarios was estimated on the basis of the customer’s knowledge. This knowledge was based on technical, business, and strategic factors, and because they have the last word about which technologies have to be supported. The probabilities can be seen in Figure 4. Finally, the value added by implementing each data storage technology variant was assessed by the customers. Therefore, customers estimated the value of each data storage technology that the family was able to support as €60,000. This value corresponded with the customer’s utility of having a family that allows them to vary the data storage technology for different data access performance requirements.

Step 2: Design of alternative architecture solutions to support the expected changes. To provide the software architecture of Figure 3.a. with the flexibility that facilitates the variation of the data storage technology, the architects proposed an alternative design architecture solution (see Figure 3.b). This architecture makes ready the software product to be flexible through two points of variation to support the four data storage technologies shown in Table 1. Because a data storage technology is composed of a data manager and a clustering framework, these variation points are defined through (i) three optional components that implement the data managers, and (ii) a Plastic Partial Component [24], i.e., a special type of component that can define variability points inside components, that implements the clustering frameworks. These components have been successfully applied in previous agile projects.

Step 3: Estimation of technical debt for each one of the alternatives. The principal, as defined in Section 3, is the cost of providing the architecture of Figure 3.a with the flexibility that facilitates the variation of the data storage technology (i.e., the cost of implementing the variation points shown in Figure 3.b). Additionally, PMD was used to detect current weaknesses (searching anti-patterns that have negative impact on flexibility) giving a total of one GodClass and 10 different coupling problems. The team estimated the effort of implement the new architecture that finally was valued at €22,000. However, if no investment in flexibility is made, then this cost is €0. The interest, as defined in Section 3, is the additional CoC of supporting a data storage technology variation derived from the lack of flexibility. The CoC was estimated in terms the dependencies and the coupling among the components that are affected by the expected variation. Dependencies were obtained from Sonar and the problems that were detected using PMD and, in terms of effort, they were estimated by the development team that already known the architecture and they have been working in the project the previous month. The CoC of the architecture of Figure 3.a was €29,500 for each data storage technology that was to be supported by the Optimeter family, while the CoC of the architecture of Figure 3.b is €15,000 for each data storage technology. Therefore, the difference in CoC between the two alternatives was €14,500 (i.e., the difference between €29,500 and €15,000).

Figure 3. Flexibility investment option in the Optimeter project
€113K = 180K – 22K – 45K
45K is the cost of change with a probability of 0.1, given the node E, and as a consequence of deciding on “investing in flexibility” in the decision node A.

Step 5: Assessment of the expected net value of the flexibility investment option. In this step, net value is calculated as described in Section 3. As a result, the decision tree construction has been completed and reflects all of the possibilities that could unfold over time and shows the choices that should be made at each decision node. Table 2 details some results of nodes of Figure 4.

Step 6: Make the decision. The final decision was made on the basis of the expected net value obtained for the analyzed scenarios (see decision node A in Figure 4). The decision tree indicates that it is better to not invest in flexibility in month 2 because the value of node marked with F is higher than the value of node E; and therefore is more profitable to delay the investment because the uncertainty could decrease in month 3 (see Figure 4). This result is due to the value gained from waiting for more information about the project. That is, waiting to resolve uncertainties before deciding whether or not to invest.

4.3. Analysis, interpretation, and conclusions

During the development of Optimier, the MAKEFlexi process guided architects in deciding about when it is more profitable to invest in flexibility to support the variation of the three data storage technologies shown in Table 1. Therefore, after the execution of the case study, we can conclude that, in an industrial project, the MAKEFlexi process is a feasible way to make decisions about flexibility investments.

Several points about the feasibility of the study are detailed as follows: First, we have pursued a simplification of the analysis to gain an understanding of the MAKEFlexi process. This simplification has been achieved by estimating the value added by each new technology with the same value. Second, the estimation of the value added by the data storage technologies was facilitated by the experts’ know-how. Finally, we have selected a case study where the expected change was well-defined. As a result, the CoC could be easily estimated in terms of coupling and dependencies.

MAKEFlexi successfully assists architects in reasoning and explaining the motivation of decisions, as it provides a qualitative baseline for reasoning. Obviously, this baseline is based in estimations; the more precise are estimations, the more precise will be the conclusions from reasoning. The estimations used in the case study are mainly subjective. But it should be highlighted that they are intended to support qualitative reasoning, not quantitative; therefore they should be simply shown the right trend.

Table 2. Examples of decision tree nodes

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Net value</th>
<th>Value description</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>€113K</td>
<td>113K = 180K – 22K – 45K Where: 180K is the value added by three variants= €60K<em>3 22K is the cost of investment in flexibility 45K is the cost of change with a previous investment in flexibility=€15K</em>3</td>
</tr>
<tr>
<td>B</td>
<td>€68K</td>
<td>68K = max(68K,61K)</td>
</tr>
<tr>
<td>G</td>
<td>€53,150</td>
<td>53,150 = (68K<em>0.67)+(23K</em>0.33)</td>
</tr>
<tr>
<td>A</td>
<td>€47,145</td>
<td>47,145=max(54,500;47,145)</td>
</tr>
</tbody>
</table>
4.4. Evaluation of validity

To improve the internal validity of the results presented, the independent variables that could influence this case study have been identified as follows: the architects’ experience, the influence of the project’s size, the architecture’s complexity, and finally the complexity of the possible expected-change scenarios, which cannot be reduced due to the inherent nature of case studies, which normally focus on one project. In particular, we have reduced the complexity of scenarios to guarantee the understandability of the case study by using the same value to estimate the value added by each data storage technology. However, if we had estimated the value added by each data storage technology with different values, the expected-change scenarios would have been more complex, and consequently the scenario would be more difficult to be understood. Finally, it is important to emphasize that if the expected changes are not well-scoped, then it would be difficult to estimate the CoC.

However, the major limitation in the case study research concerns external validity because only one case has been studied. In return, case studies allow one to evaluate a phenomenon, a model, or a process in a real setting. This is important in software engineering in which a multitude of external factors may affect the validation results, and where other techniques, such as formal experiments, are not considered to be conducted under controlled settings, even though formal experiments permit replication and generalization.

5. Related Work

In the literature, several works about technical debt and the application of real options valuation to software engineering can be found.

Baldwin and Clark use the real options approach to assess the value of design options created by mod-
ularity [3]. These options have value because they allow designers to independently investigate and replace designs. This work has been used in valuing modularity in object-oriented and aspect-oriented designs by Sullivan et al. [31] and Lopes and Bajracharya [19]. However, they do not use real options valuation to analyze the profitability of investments in flexibility.

It is important to emphasize that the literature notes that decision trees have been proposed as a suitable method to assess the value of real options in software [13]. However, the works of Ozkaya et al. [23], Bahsoon and Emmerich [2], and Gustavsson and Axellson [15] do not use this method. Ozkaya et al. assess the value of different architectural patterns and consider the maintainability and availability of the system. The implementations of some specific patterns are the real options. They use maintenance cost similarly we use cost of change. Additionally, they use the binomial valuation method as a way to assess the value of the options. However, binomial valuation restrict how the asset value—in this paper the asset is the investment—can evolve by using a lognormal distribution [8]. This restricts the change scenarios, existing in a software project, which can be represented. In contrast, decision trees can be easily adapted to represent more change scenarios. Bahsoon and Emmerich propose the real options approach to evaluate architectures in terms of their stability [2]. This approach makes the assumption of a fixed expiration date; the decision as to whether or not to execute the option can only be made at the end of the evaluation time. Based on this assumption, they use the Black-Scholes valuation method, which is not suitable for assessing the possibility of executing an option at any moment of the analysis time period. Finally, Gustavsson and Axellson use real options valuation to take into account the value of flexibility in automotive system design [15]. Gustavsson and Axellson use the binomial valuation method because it has an estimated volatility (volatility is a percentage that represents how the value of the investment can increase or decrease over the analyzed period of time). Decision trees can be used instead of volatility, which allows one to represent the possible situations that can happen (e.g., using the knowledge and experience of stakeholders and developers) during the analyzed period of time.

On the other hand, Erdogmus [12] uses decision trees to assess the value of the option of being able to continue a project or abandon it with iterative processes in opposition to a sequential process. He uses decision trees to show the differences related to flexibility between the iterative development process and the sequential development process (i.e., Erdogmus’ work is based on development process flexibility, but not on software product flexibility).

With regard to technical debt, several studies propose means to estimate the interest and/or principal of technical debt [10] [14] [22]. The use of real options valuation with technical debt has been suggested previously [17] [29]. Specifically, MAKEFlexi presents a solution that uses real options valued on decision trees and technical debt to deal with the flexibility investment decision making problem. The MAKEFlexi solution is a process that uses real options valuation to estimate the value of flexibility investment options and it uses technical debt to estimate the cost of the different degrees of flexibility provided by alternative architecture design solutions.

6. Conclusion

This paper has presented the MAKEFlexi process. It provides a solution for the flexibility investment decision making problem in agile architecting. The process helps architects to create models to decide whether to invest or not in flexibility, and when to do it, i.e., when is more profitable to delay the flexibility investment—i.e., take technical debt—or to invest in flexibility to manage the debt. Therefore, architects can make design decisions about flexibility based on economic considerations.

MAKEFlexi helps select the architecture design solution with the highest ROI of design for flexibility over the time frame under analysis. This decision takes into account the potential long-term benefit of flexibility, the short-term cost of each alternative architecture design solution as well as the uncertainty that the changes finally will not happen. Our experience in applying MAKEFlexi in the industrial project OPTIMETER was really satisfactory. MAKEFlexi allowed us to delay the decision of investing in flexibility at the second month of the project, so that we obtained a higher ROI. Additionally, we plan to evaluate the MAKEFlexi process in more case studies to reinforce that the process is feasible in other projects, and to determine in which situation is more profitable its application.

As future work, we plan to automate the MAKEFlexi process by developing a tool or extending one of the existing tools that support Real Options valuation. Also, it would be interesting to enrich the MAKEFlexi process with the storage and mining of architectural knowledge in order to be used as historical data. In long-term, we plan to integrate this tool with an architecture modeling framework with decision and architectural knowledge support in order to improve the design decision-making process.
Acknowledgements
The work reported in here has been partially sponsored by the Spanish fund: IMPONET (ITEA 2 09030, TSI-02400-2010-103), iSSF (IPT-430000-2010-038), NEMO&CODED (ITEA2 08022, IDI-20110864) and ENERGOS (CEN-20091048) and UPM (Researcher Training program).

References