Sequencing Intelligent Components through Releases as a Risk Reduction Strategy: a Smart-city Example

Pablo Valenzuela-Toledo

Department of Computer Sciences and Informatics
University of La Frontera
Temuco - Chile 4811230
email: pablo.valenzuela@ufrontera.cl

Carlos Cares

Center of Studies in Software Engineering
University of La Frontera
Temuco - Chile 4811230
email: carlos.cares@ceisufro.cl

***Abstract*—In Component-Based Software Engineering, the process of selecting software components is under several risk factors. Traditionally, these have been identified or mitigated with software project management techniques. However, the new demand for intelligent systems has added complexity to the process. Despite the success and technological advances of this type of systems, their development in an environment ready for production remains a challenge. There is a considerable number of technical issues that limit their adoption, and their selection determines the introduction of new risk factors, different from traditional ones. In this paper, we present the following idea: given a set of requirements for one component - intelligent behaviour, for example - we propose to sequence and replace different components through evolving releases as a risk reduction technique, instead of choosing the option of only one "right" component. Using a systematic mapping literature review, we gather the main risks of intelligent components. Then, we present a formalization of the risk-based component selection technique. Finally, we offer an example to illustrate our approach using a sequence of intelligent software components in the context of an air pollution forecasting system.**

Keywords–Intelligent components; Component Selection; Component-Based Software Engineering; Risk Management.

I. INTRODUCTION

The Component-Based Software Engineering approach is based on the idea that software systems can evolve by selecting and aggregating appropriate software components [1]. Some of the recognized advantages of this approach are: (1) faster development, since assembling new applications through existing components reduces development time; (2) easier to maintain, since managing one component at a time makes maintenance easier; (3) improved quality, since each component is tested before releasing it; (4) easier to create applications variants and upgrades, since changing or upgrading each component separately is simpler; and (5) lower overall development cost, since the development cost is reduced by handling or upgrading information systems separately [2].

When a software-intensive system is evolving, the process of deciding which component to use involves different available sourcing options, such as internal development, outsourcing, buying a commercial component or adopting some open source component [3]. Several factors have already been identified, in order to select a specific component: size, cost, maturity (years, versions), compatibility, and adherence to standards, among many other nonfunctional requirements [3]–[5].

Risk management is a classical area in Project Management discipline covering not only projects but also programs and portfolios [6]. Risk management implies to manage potential events which would (negatively) impact long-term strategic objectives and projects' objectives, i.e., cost, time and scope. In software engineering, risk management has been a topic of growing relevance through time, and different risk factors have been identified, such as analysis, design, coding, testing, planning, control, contracts, teams, clients, policies and structure [7]. In relation to Component-Based Software Engineering, the risk is moved from classical waterfall stages and their management to component-based stages and their management, i.e., to component seeking, selecting, and testing.

While the field of Component-Based Software Engineering has identified ways to help select a software component using prioritization factors, there are new considerations and challenges to overcome due to the intelligence software era [8]. This intelligence software era brings systems that are known as systems that can automatically improve through experience [9]. The successes of the artificial intelligence field are visible, for example, in domains such as computer vision (e.g., object recognition [10]), natural language processing (e.g., information extraction [11]), and sound analysis (e.g., voice recognition [12]). Diverse applications became part of products of big and famous companies, such as Facebook, Google, and Apple, producing a closeness effect between people and artificial intelligence which has brought a new set of demands to software production.

Therefore, despite the success and technical advances in intelligence systems, its development in a production-ready setting still remains challenging. There is a lack of tools and software engineering practices for building such systems, especially if the company/organization does not have an experienced machine-learning research group and a data-oriented supporting infrastructure [13]. For example, let us consider the selection of a machine learning component to enable a weather forecasting functionality. This component may include specifications about hardware (e.g., Graphics Processing Unit (GPU) models), platforms (e.g., machine learning, deep learning library dependencies), source code (e.g., prepossessing, glue code), configuration (e.g., model features configuration), training data (e.g., sample period), or model state (e.g., version of training model) [14]. This set of attributes differs from traditional prioritization factors, adding complexity to the selection process, and therefore risks.

To address the challenges presented above, in this article, we present an approach based on software components, which allows managing the risk involved in selecting complex software components, such as those that provide intelligent

 ISBN: 978-1-61208-730-6

behaviour. The contributions of this work are the following: (1) we identify the intelligent component risks by conducting a systematic mapping literature review; (2) we introduce a formalization of the problem of risk in the process of software components selection; (3) we present a selection sequence technique, from low to high risk, with a growing scenario of requirements; and (4) we illustrate the proposal by presenting the case of a plan for implementing an air pollution forecasting system as part of a Smart-city project.

The remainder of this paper is organized as follows. In Section 2, we collect information for establishing the risky points of intelligent components, by applying the systematic mapping protocol. In Section 3, we present a general framework for selecting a sequence of components in place of only one and the constraints for this choice. In Section 4, we apply the general proposal by showing a scenario of the selection of an intelligent component that enables an air pollution forecasting functionality. Finally, in Section 5, we conclude with benefits and limitations.

II. Open issues on selecting and adapting intelligent components

To investigate the risky points when developing, selecting or adapting intelligent software components, we analyze the state-of-the-art of related software engineering challenges. Our goal was to understand what are the software engineering challenges, how they have been addressed, as well as the context where they occur. To achieve that, we perform a systematic mapping review (following the guidelines of [15] [16]).

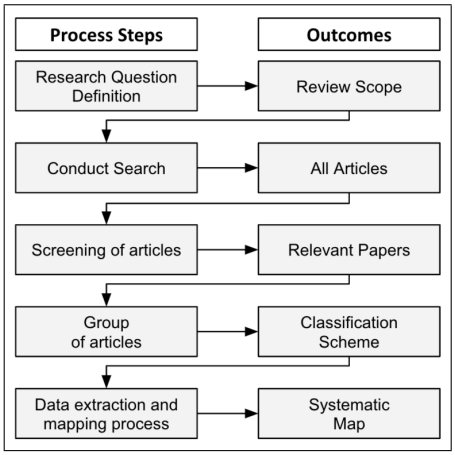

Figure 1. Systematic mapping review process. Adapted from [16].

Figure 1 gives an overview of how we developed the study. First, we formulated research questions keeping in mind the problem of unknown challenges when developing intelligent software components. Second, we searched for articles in digital libraries. The search included the search keywords, the temporal interval of papers and the type of bibliographic source. Third, we defined and applied the inclusion and exclusion criteria to select articles. Fourth, we described the classification scheme built with the selected articles. Finally, we ordered the selected articles and generated a systematic map.

A. Research question definition

We defined the following main research question (RQ): what are the software engineering challenges when developing, selecting or adapting intelligent software components? In our research scope, we considered answering this question by reviewing studies that present analysis, reviews or case studies that overview software engineering challenges related to the development of intelligent software components.

B. Executing the search

We executed the search by intersecting and joining keywords in the following search string: (‘machine learning’ OR ‘artificial intelligence’ OR ‘autonomous’ OR ‘deep learning’) AND (‘systems’ OR ‘applications’ OR ‘components’) AND (‘software engineering challenges’). We utilized four library sources: (1) Science Direct [17]; (2) IEEE Digital Library [18]; (3) ACM Digital Library [19]; and (4) Springer Link [20]. Complementary, we also looked for articles in Scopus bibliographic database [21] as a form of verification, and to expand the search spectrum. In all cases, we looked for articles between the years 2014 and 2019 (March). As a result, we gathered a total of 615 articles.

C. Screening of articles

To screen and review the gathered articles, we defined and applied inclusion and exclusion criteria. Our inclusion criteria considered primary and secondary studies published in chapters of books, journals and scientific conferences. As exclusion criteria, we did not consider articles with: (1) research context out of our scope; (2) a language other than English used; (3) specific application domain (e.g., telecommunication; energy, medicine, etc.); (4) duplicates and conference proceedings summaries; and (5) no full article text available. After applying the inclusion criteria, we selected a total of 9 relevant articles. Also, the exclusion criteria allow us to select a total of 4 articles. In this phase, we also included a snowballing article selection technique (following the guidelines of [15]). This technique gave us a total of 5 new relevant articles. We also decided to add 8 articles manually. These articles come from relevant conferences, like The First Symposium on Software Engineering for Machine Learning Applications [13] and domain experts from Google. Finally, we got a total of 17 selected articles.

TABLE I. SUMMARY OF GATHERED ARTICLES BY CRITERIA AND STAGE PART I.

	ScienceDirect	IEEE	ACM
Initial search	14	4	8
Inclusion criteria	0	2	1
Exclusion criteria	0	1	1
Snowballing selection	0	1	0
Total after criteria	0	2	1
Manually added	-	-	-
Final total	-	-	-

A summary of gathered articles by criteria is available in Table I and Table II (The data was divided in two tables for readability).

D. Grouping the articles

Once we finished the previous phases, we defined a classification scheme based on three subjects related to software engineering challenges: (1) Software Life Cycle Phase, which concern the software life cycle phases involved. Here, we included specification, development, verification and validation, and evolution phases. Also, we included project management as a way to categorized related project development issues; (2) Proposals, to investigate how the challenges have been

addressed. That includes development approach, project management, adjustment, measurement strategy, tool development, team diversity or no strategy; (3) Context, to investigate where the challenges have happened. We considered the academy, big industry (e.g., Google or Apple) or small industry (e.g., online or startup companies).

TABLE II. SUMMARY OF GATHERED ARTICLES BY CRITERIA AND STAGE PART II.

	Springer	Scopus	Total
Initial search	45	544	615
Inclusion criteria	0	6	9
Exclusion criteria	0	2	4
Snowballing selection	0	4	5
Total after criteria	0	6	9
Manually added	-	-	8
Final total	-	-	17

E. Data extraction and mapping process

We completed the systematic mapping review by generating a summary study map (Figure 2). We were able to identify three categories with associated papers: (1) development; (2) evolution; and (3) project management. In each of these categories, there exist software engineering challenges that we summarize in Table III.

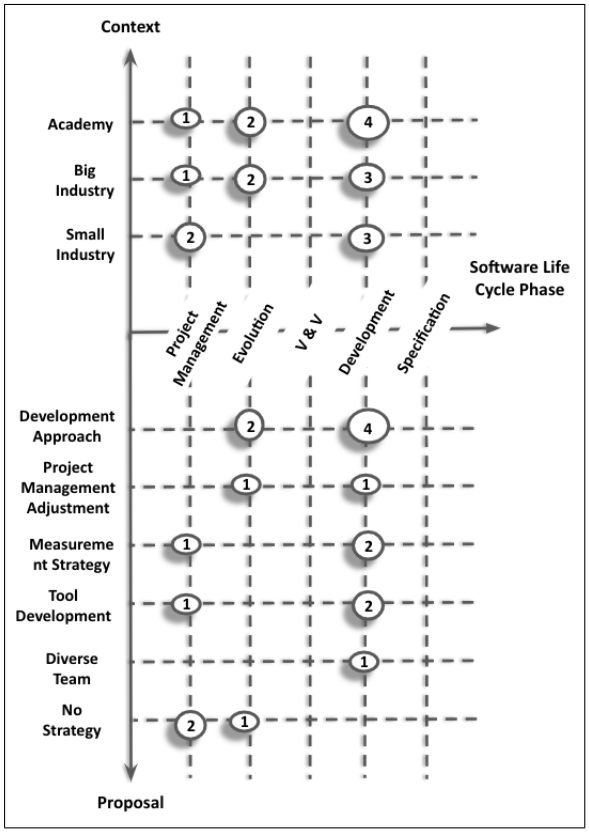

Figure 2. Summary study map.

The Development category includes the following software engineering challenges (or open issues): (1) Experiment Management, to manage a large number of experiments performed when identifying an optimal model. In this experimental process, it is necessary to guarantee reproducible results [14] [22] [23]; (2) Transparency of Machine Learning Models. Complex models like large neural networks used in fields, for example, computer vision or natural language processing are difficult to explain, and thus, transparency is traded for accuracy [13] [14]; (3) Difficulties in estimating the results of intelligent software components before they have been trained and tested [24] [25]; (4) Resource Limitation, concerning specific requirements of distributed systems to manage large volumes of data, the computational needs for extracting and transforming data, training and evaluating a model, and serving the model in production [14] [24]; (5) Testing, related to the need for tools which allow test static data and production data, machine learning models and production-ready components [22] [24] [26]; (6) Data Processing, when working with distributed systems, adds complexity in several dimensions. It requires additional knowledge, time to operate the systems, management and resources associated with hardware and software [14] [24]; and (7) Development Team, concerning how diverse a software development team must be. This list covers different aspects of developing an intelligent component [23] [27] [28].

The Evolution category includes: (1) Issues with Dependencies, that is data, hardware, machine learning frameworks or models dependencies [28] [29] [30]; (2) Hidden Feedback Loops; this phenomenon happens when the data adapt the model and not backward, especially in production-ready systems [14]; (3) Monitoring Deployed Systems that refers to the need to maintain a deployed machine learning system over time. Usually, teams fail to recognize the effort needed [14] [23]; and (4) Glue Code and Supporting Systems that refers to systems with a small part of the code belonging to the intelligent component. Here, the rest is "glue code" that interacts with supporting systems, and thus, the system becomes hard to test [14] [31] [32].

TABLE III. OPEN ISSUES/SOFTWARE ENGINEERING CHALLENGES

Category	Challenges / Open Issues
Development	Experiment Management
	Transparency of Machine Learning Models
	Difficulties to Estimate Results
	Resource Limitation
	Testing
	Data Processing
	Development Team
Evolution	Issues with Dependencies
	Hidden Feedback Loops
	Monitoring
	Glue Code and Supporting Systems
Project Management	Effort Estimation
	Privacy
	Data Workflows

Finally, the Project Management category includes: (1) Effort Estimation. When developing intelligent software components, the goals might be unclear, and an expected-results definition is challenging. Also, other properties like acceptable performance definition are hard to set beforehand [28] [33] [34]; (2) Privacy of the Knowledge of a System, that refers to how to store the data across the weights of a machine learning model. This information is hard to manage when there is a lack of model understanding. It forces companiesto have terms of service agreements and to use anonymized or aggregated statistics of the user data [13] [14] [23]; and (3) Data Workflows that required to handle the volume, variety, and the velocity of a large amount of data [33] [35] [36].

The final list of software engineering challenges involves a risk every time that there is a lack of adequate associated techniques. However, to present our proposal, we formulate a formalization in the next section.

III. SEQUENCING COMPONENTS AS RISK REDUCTION STRATEGY

In this section, we present a theoretical point of view of the problem of the selection of several components in place of selecting only one. Regarding this topic, the main focus of the literature is on how to select one component for a specific set of requirements. Here, the methods and techniques for decision making are qualitative approaches [5] [37] [38]. To the best of our knowledge, there are no proposals considering the redundancy of components for the same set of requirements, and there are not proposals considering components' risks. In this context, we consider that most of the quality factors are uncertain factors because the expected quality occurs in the best case, once the component has been successfully integrated. This perspective manages not only quality factors but traditional uncertainties, such as total time and budget.

In this proposal, we keep one of the main assumptions of other authors, that is, we decide what components to select before the test and try cycle. Risk perspective accepts that the uncertainty of a project is higher at the beginning and diminishes as it progresses due to proactive planning and pertinent decision-taking [6]. Therefore, the focus of the proposal is on the worst high risks case scenario, i.e., at the beginning of the project.

In the following, we present a set of definitions for building a conceptual framework which enables the consideration of risks as a manifestation of uncertainties associated with components.

Definition 1: The nth-dimension Risk of Using a Component c, as part of a software solution s, or simply RUC, is a vector $\vec{r}_s^{\,c} = (r_1, r_2, ..., r_n)$ where $r_i \in [0,1] \subseteq \mathbb{R}$.

Definition 2: The nth-dimension risk-evaluation of a set of components $C = \{c_1, c_2, ..., c_m\}$ for a software solution s is denoted as the relationship $R_s^C \subseteq C \times [0,1]^m$:

$$R_s^C = \{(c_1, \vec{r}_s^{\,c_1}), (c_2, \vec{r}_s^{\,c_2}), ..., (c_m, \vec{r}_s^{\,c_m})\} \quad (1)$$

where the second element $\vec{r}_s^{\,c_i}$ is the nth dimension RUC corresponding to the component c_i in the software solution s.

Definition 3: A risk-based prioritization function over a set of components $C = \{c_1, c_2, ..., c_m\}$ is a function $\rho : [0,1]^n \longrightarrow [0,1] \subseteq \mathbb{R}$. The resulting value of $\rho(\vec{r}_s^{\,c_i})$ will be called the total risk of a the component c_i under ρ.

Definition 4: The functionality of a software component respecting the set of requirements $Req = \{r_1, r_2, ..., r_F\}$ is a vector $\vec{f}_{Req}^{\,c} = (f_1, f_2, ..., f_k)$ where $f_i \in \{0,1\}$. It will be said that the component c accomplishes the functionality f_i iff $f_i = 1$. In contrast, it will be said that the component c does not accomplish the functionality f_i iff $f_i = 0$.

Definition 5: The functionalities of two components c_1 and c_2 respecting the set of requirements $Req = \{r_1, r_2, ..., r_F\}$ are the same iff $\vec{f}_{Req}^{\,c_1} = \vec{f}_{Req}^{\,c_2}$ and it will be said that c_1 and c_2 have different functionality iff $\vec{f}_{Req}^{\,c_1} \neq \vec{f}_{Req}^{\,c_2}$.

Definition 6: Given two components c_1 and c_2, the set of requirements $Req = \{r_1, r_2, ..., r_F\}$, the functionality of c_1 as $\vec{f}_{Req}^{\,c_1} = (f_1, f_2, ..., f_m)$ and the functionality of c_2 as $\vec{f}_{Req}^{\,c_1} = (g_1, g_2, ..., g_m)$, then it will be said that c_1 has less or equal functionality than c_2, denoted as $\vec{f}_{Req}^{\,c_1} \preceq \vec{f}_{Req}^{\,c_2}$ iff $f_k \leq g_k \forall k$.

Lemma 1: Given a set of components C, the binary relation $\preceq$ for functionalities of components imposes a partial order on C. Therefore, C is posed under $\preceq$.

Definition 7: The functional impact of a RUC $\vec{r}^{\,c}$ is a function $\lambda : [0,1]^n \longrightarrow \{0,1\}^k$. The resulting value $\lambda(\vec{r}^{\,c})$ will be called the loss of functionality of $\vec{r}^{\,c}$. It will be said that there is no loss of functionality iff: $\lambda(\vec{r}^{\,c}) = \vec{f}^{\,c}$ and it will be said that there is a total loss of functionality iff $\lambda(\vec{r}^{\,c}) = \vec{0}$.

Definition 8: Given a component c, its RUC $\vec{r}^{\,c}$, its corresponding evaluation of its loss of functionality $\lambda(\vec{r}^{\,c})$ and ρ a risk-based prioritization function, then the probability of occurrence of that loss of functionality will be $\rho(\vec{r}^{\,c})\lambda(\vec{r}^{\,c})$.

Definition 9: A risk-appraised situation for a component selection stage, as part of a component-based software process for developing the software s, is a quintuple $A_s = <C, Req, Rsk, \rho, \lambda>$, where, C is a non empty set of components, Req is a non empty set of requirements, Rsk is a particular nth-dimension risk-evaluation R_s^C, ρ is a risk-based prioritization function over C, and λ is a functional impact function applicable to risks in R_s^C.

Definition 10: A risk-appraised situation for a component selection stage, as part of a component-based software process $A_s = <C, Req, Rsk, \rho, \lambda>$, is called coherent iff

$$\vec{f}_{Req}^{\,c_i} \preceq \vec{f}_{Req}^{\,c_j} \Rightarrow \rho(\vec{r}_s^{\,c_i}) \leq \rho(\vec{r}_s^{\,c_j}) \;\; \forall c_i, c_j \in C, \quad (2)$$

When this constraint is not satisfied, then it is called an incoherent risk-appraised situation.

Lemma 2: Any risk-appraised situation $A_s = <C, Req, Rsk, \rho, \lambda>$, having just one element in C, is coherent.

The proof is trivial because there is only one element in C and ρ is a function. Therefore, the total risk of the component c_i is equal to itself.

Definition 11: A set of components D is called disposable by risk of a risk-appraised situation $A'_s = <C \cup D, Req, Rsk, \rho, \lambda>$ iff A'_s is incoherent, $A_s = <C, Req, Rsk, \rho, \lambda>$ is coherent and both situations accomplish the condition of equal functionality, i.e.,

$$\bigcup_{c_i \in C} \vec{f}_{Req}^{\,c_i} = \bigcup_{c_k \in C \cup D} \vec{f}_{Req}^{\,c_k} \quad (3)$$

Lemma 3: Given an incoherent risk-appraised situation $A_s = <C, Req, Rsk, \rho, \lambda>$ then, by definition

$$\exists c_i, c_d \in C \vec{f}_{Req}^{\,c_d} \prec \vec{f}_{Req}^{\,c_i} \quad (4)$$

and

$$\rho(\vec{r}^{\,c_d}) \geq \rho(\vec{r}^{\,c_d}) \quad (5)$$

then $\{c_d\}$ is disposable-by-risk of A_s.

Lemma 4: Given a coherent risk-appraised situation $A_s = <C, Req, Rsk, \rho, \lambda>$ and $\exists c_i, c_d \in C$

$$\vec{f}_{Req}^{\,c_d} \prec \vec{f}_{Req}^{\,c_i} \quad (6)$$

and

$$\rho(\vec{r}^{\,c_d}) \geq \rho(\vec{r}^{\,c_i}) \quad (7)$$

then $\{c_d\}$ is disposable-by-risk of A_s.

IV. THE ARAUCANÍA DIGITAL SMART-CITY EXAMPLE

In order to illustrate our proposal, we present an example using the Araucanía Digital Smart-city Project (ADSP) context, that includes air pollution forecasting functionality. The ADSP goal is to develop smart cities systems in the region of La Araucana, Chile. The project is funded by the Inter-American Development Bank (BID) and is executed by industry and academy actors [39].

 ISBN: 978-1-61208-730-6

First, we define our risk vector, following Definition 1, that considers the risk associated with intelligent components and with traditional software development projects (we present a scenario with already evaluated risks, and thus, we do not review how the values were generated). The risks for the intelligent components are those already identified in Section II: (1) Experiment Management (EM), $p = 0.05$; (2) Difficulties to Estimate Results (DER), $p = 0.1$; (3) Development Team (DT), $p = 0.15$; (4) Glue Code (GC), $p = 0.15$; and (5) Data Workflows (DW), $p = 0.05$. We also include the following 2 risks related to traditional software development: (6) Budget Limitations (BL), $p = 0.15$; and (7) Timeline Restriction (TR), $p = 0.15$ [40].

Second, we define a set of software components that provide air pollution forecasting functionality. Each component considers an associated algorithm previously used in another context for this purpose [41]. We name every component based on the associated algorithm-name as follows: (1) Support Vector Machine (SVM); (2) Artificial Neural Networks (ANN); (3) K-Means (KM); (4) K-Nearest Neighbors (KNN); and (5) Regression Models (RM).

Third, we present a resume of the seven risks evaluation related to the five intelligent software components (all mentioned above) (Table IV). Also, we set the values according to the context that the software development team and the project give us (we stand that we do not consider the methodology to calculate the risk value, and thus, we assume that this example does not represent a case study with appropriate empirical rigour). Here, the team is one project manager, two senior software developers, one junior data analyst, and one senior software engineering scientist. It is crucial to notice that we do not have a machine learning specialist in our team (this context represents a particular case that may change with a different team or a different project).

TABLE IV. RISK EVALUATION OF AN INTELLIGENT COMPONENTS SET

R_s^C	EM	DER	DT	GC	DW	BL	TR	ρ
SVM	0.5	0.5	0.6	0.5	0.6	0.5	0.4	0.41
ANN	0.4	0.6	0.6	0.5	0.8	0.6	0.6	0.47
KM	0.3	0.4	0.4	0.5	0.6	0.4	0.3	0.47
KNN	0.3	0.4	0.4	0.6	0.6	0.4	0.3	0.34
RM	0.2	0.2	0.2	0.6	0.6	0.3	0.1	0.24

In order to evaluate how many software requirements accomplish the functionalities of a set of software components, we present a resume in Table V. The collection of software requirements is defined as follows [26]: (1) Air Pollution Index Forecasting (APF), i.e., the prediction of the future value of the polluting particle; (2) Interpolation of the Current Pollution value (ICP), i.e., constructing new data points within some geographical areas; (3) Automatic Fail Identification (AFI), i.e., automatic identification of "not a number"(NaNs) values or infinities appearing in the model during the execution of the system; (4) Editable Model (EM), that is, the component supports hyper-parameters updating; and (5) Stale Model Aware (SMA), that is, the model allows an automatic stale identification.

In Table VI, we present the loss of functionality of components due to the risks mentioned above. To do this, we use the λ function on each functionality defined in the previous step.

Following the definitions, we have a risk-appraised situation as we describe in Definition 9. Additionally, there exists an incoherent risk-appraised scenario because, for example, SVM

TABLE V. THE FUNCTIONALITY OF AN INTELLIGENT SOFTWARE COMPONENT WITH RESPECT TO A SET OF REQUIREMENTS

$\vec{f}^{\,c}_{Req}$	APF	CCD	AFI	EM	SMA	ρ
SVM	1	0	0	0	0	0.41
ANN	1	1	1	1	0	0.47
KM	0	1	0	1	0	0.47
KNN	1	1	0	1	0	0.34
RM	1	1	1	0	0	0.24

TABLE VI. LOSS OF FUNCTIONALITY OF COMPONENTS

λ	$\lambda(APF)$	$\lambda(CCD)$	$\lambda(AFI)$	$\lambda(EM)$	$\lambda(SMA)$	ρ
SVM	1	0	0	0	0	0.41
ANN	1	1	1	1	1	0.47
KM	0	1	0	1	0	0.47
KNN	1	1	0	1	0	0.34
RM	1	1	0	0	0	0.24

provides less functionality than RM, and its risk is higher than RM. Therefore, SVM is risk disposable. Similarly, KNN offers better functionality than KM, and its risk is lower than the risk of KM. Therefore, KM is risk disposable too. Removing the disposable components, we have the following situation that $\preceq$ imposes to the component set:

$$\vec{f}^{\,RM}_{Req} \preceq \vec{f}^{\,ANN}_{Req}; \vec{f}^{\,KNN}_{Req} \preceq \vec{f}^{\,ANN}_{Req} \quad (8)$$

Finally, we selected these three components for the forecasting functionalities (RM, KNN, ANN). The potential loss of functionality due to risks, by using the logic operator "AND", gives us additional security on functionality AFI. To select a sequence, we first select the less risky component to ensure some functionality on early releases. But, due to the team size, we may support only prototype development at the moment. Therefore, we first plan RM and KNN then ANN. Note that risks are going down from 0.47 to 0.24*0.34*0.47.

V. CONCLUSION

In this paper, we have presented a solution proposal to the problem of intelligent software component selection. First, we have conducted a systematic literature review that allows us to identify a collection of risks that we classified into three main categories: (1) Development; (2) Evolution; and (3) Project Management. This review differs from previous related ones because our goal has been found lack of software engineering, instead of identifying a specific way to deploy intelligent systems. Second, we have formalized our proposal by presenting a conceptual framework and risk reduction strategy to support the component selection process. To the best of our knowledge, there is no work addressing the selection problem in the way that we have been presenting it. Third, we present the scenario of risks and how to select more than one component instead of just the best one. In this way, the advantage of this proposal is that it allows looking for not only the optimal option but a set of available best ones. Fourth, we have illustrated this scenario with a case of software planning in the context of an air pollution forecasting functionality as part of a Smart-city project. Fifth, as a limitation, while we considered that our presented example illustrates our proposal, under no circumstances we pretend to present this as a case study or an empirical evaluation. Thus, our future work is: (1) update our systematic literature review; (2) improve the proposal by formulating the component selection problem as a search based problem. With this approach, we expect to prove the methodological framework supporting our idea; and (3)

conduct a study case, using the frame of The Araucanía Digital Smart-city Project.

ACKNOWLEDGMENT

The authors would like to thank to ATNME16393CH SMART CITY IN A BOX project 2019, Temuco, Chile.

REFERENCES

[1] G. Pour, "Component-based software development approach: new opportunities and challenges," in Proceedings. Technology of Object-Oriented Languages. TOOLS 26 (Cat. No. 98EX176). IEEE, 1998, pp. 376–383.

[2] H.-I. Jeong, C.-S. Lee, C.-H. Kim, C. Park, and H.-C. Woo, "Design of a software component bank for distribution," Journal of systems integration, vol. 10, no. 3, 2001, pp. 223–237.

[3] P. Chatzipetrou et al., "Component selection in software engineering-which attributes are the most important in the decision process?" in 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, 2018, pp. 198–205.

[4] M. Borg et al., "Selecting component sourcing options: A survey of software engineerings broader make-or-buy decisions," Information and Software Technology, vol. 112, 2019, pp. 18–34.

[5] C. Alves, X. Franch, J. P. Carvallo, and A. Finkelstein, "Using goals and quality models to support the matching analysis during cots selection," in International Conference on COTS-Based Software Systems. Springer, 2005, pp. 146–156.

[6] H. Sanchez, B. Robert, M. Bourgault, and R. Pellerin, "Risk management applied to projects, programs, and portfolios," International journal of managing projects in Business, vol. 2, no. 1, 2009, pp. 14–35.

[7] H. R. Costa, M. d. O. Barros, and G. H. Travassos, "Evaluating software project portfolio risks," Journal of Systems and Software, vol. 80, no. 1, 2007, pp. 16–31.

[8] E. Woods, "Software architecture in a changing world," IEEE Software, vol. 33, no. 6, 2016, pp. 94–97.

[9] M. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," Science, vol. 349, no. 6245, 2015, pp. 255–260.

[10] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg, "Cloud-based robot grasping with the google object recognition engine," in 2013 IEEE International Conference on Robotics and Automation. IEEE, 2013, pp. 4263–4270.

[11] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, "Very deep convolutional networks for natural language processing," arXiv preprint arXiv:1606.01781, vol. 2, 2016, pp. 1107–116.

[12] J. R. Bellegarda, "Spoken language understanding for natural interaction: The siri experience," in Natural Interaction with Robots, Knowbots and Smartphones. Springer, 2014, pp. 3–14.

[13] F. Khomh, B. Adams, J. Cheng, M. Fokaefs, and G. Antoniol, "Software engineering for machine-learning applications: The road ahead," IEEE Software, vol. 35, no. 5, 2018, pp. 81–84.

[14] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software engineering challenges of deep learning," in 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, 2018, pp. 50–59.

[15] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in Proceedings of the 18th international conference on evaluation and assessment in software engineering, Citeseer. ACM, 2014, p. 38.

[16] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," Information and Software Technology, vol. 64, 2015, pp. 1–18.

[17] "Science direct," 2019, (Accessed on 13/06/2019). [Online]. Available: https://www.sciencedirect.com/

[18] "Ieee digital library," 2019, (Accessed on 13/06/2019). [Online]. Available: https://ieeexplore.ieee.org/Xplore/home.jsp

[19] "Acm digital library," 2019, (Accessed on 13/06/2019). [Online]. Available: https://dl.acm.org/

[20] "Springer link," 2019, (Accessed on 13/06/2019). [Online]. Available: https://link.springer.com/

[21] "Scopus," 2019, (Accessed on 13/06/2019). [Online]. Available: https://www.scopus.com/home.uri

[22] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ml test score: A rubric for ml production readiness and technical debt reduction," in 2017 IEEE International Conference on Big Data (Big Data). IEEE, 2017, pp. 1123–1132.

[23] J. Bosch, H. H. Olsson, and I. Crnkovic, "It takes three to tango: Requirement, outcome/data, and ai driven development," in Proceedings of the International Workshop on Software-intensive Business: Start-ups, Ecosystems and Platforms, vol. 2018, 2018, pp. 177–192.

[24] O. Hummel, H. Eichelberger, A. Giloj, D. Werle, and K. Schmid, "A collection of software engineering challenges for big data system development," 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2018, pp. 362–369.

[25] M. Felderer, B. Russo, and F. Auer, "On testing of data-intensive software systems," CoRR, vol. abs/1903.09413, 2019, in press.

[26] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "Whats your ml test score? a rubric for ml production systems." IEEE, 2016, pp. 19–26.

[27] M. Kim, T. Zimmermann, R. DeLine, and A. Begel, "The emerging role of data scientists on software development teams," in Proceedings of the 38th International Conference on Software Engineering. ACM, 2016, pp. 96–107.

[28] P. Bailis, K. Olukotun, C. Ré, and M. Zaharia, "Infrastructure for usable machine learning: The stanford dawn project," 2017.

[29] T. Menzies, C. Bird, T. Zimmermann, W. Schulte, and E. Kocaganeli, "The inductive software engineering manifesto: principles for industrial data mining," in MALETS '11. ACM, 2011, pp. 19–26.

[30] E. Breck, N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data infrastructure for machine learning," 2018, (Accessed on 13/06/2019). [Online]. Available: http://www.sysml.cc/doc/9.pdf

[31] D. Morgenthaler, M. Gridnev, R. Sauciuc, and S. Bhansali, "Searching for build debt: Experiences managing technical debt at google," in 2012 Third International Workshop on Managing Technical Debt (MTD). IEEE, 2012, pp. 1–6.

[32] K. M. Anderson, "Embrace the challenges: Software engineering in a big data world," 2015 IEEE/ACM 1st International Workshop on Big Data Software Engineering, 2015, pp. 19–25.

[33] T. Menzies, C. Bird, T. Zimmermann, W. Schulte, E. Kocaganeli, and T. Zimmermann, "The inductive software engineering manifesto: principles for industrial data mining," 2011, pp. 19–26.

[34] D. Sculley et al., "Machine learning: The high interest credit card of technical debt," in SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop), 2014.

[35] H. H. Olsson and J. Bosch, "Towards data-driven product development: A multiple case study on post-deployment data usage in software-intensive embedded systems," in International Conference on Lean Enterprise Software and Systems. Springer, 2013, pp. 152–164.

[36] G. Yenni et al., "Developing a modern data workflow for regularly updated data," in PLoS biology. Public Library of Science, 2019.

[37] R. Land, L. Blankers, M. Chaudron, and I. Crnković, "Cots selection best practices in literature and in industry," in International Conference on Software Reuse. Springer, 2008, pp. 100–111.

[38] A. Mohamed, G. Ruhe, and A. Eberlein, "Cots selection: past, present, and future," in 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07). IEEE, 2007, pp. 103–114.

[39] "Ufro smart city," 2019, (Accessed on 13/06/2019). [Online]. Available: http://smartcity.ufro.cl/

[40] L. Wallace, M. Keil, and A. Rai, "Understanding software project risk: a cluster analysis," Information & management, vol. 42, no. 1, 2004, pp. 115–125.

[41] C. Bellinger, M. S. M. Jabbar, O. R. Zaïane, and A. R. Osornio-Vargas, "A systematic review of data mining and machine learning for air pollution epidemiology," in BMC public health. BMC Public Health, 2017, p. 907.

 ISBN: 978-1-61208-730-6