Estimating the Complexity of Architectural Design Decision Networks

CARLOS CARRILLO SANCHEZ1, RAFAEL CAPILLA2, AND MIROSLAW STARON3
1Departamento de Ingeniería Telemática y Electrónica, Universidad Politécnica de Madrid, 28031 Madrid, Spain
2Department of Informatics, Universidad Rey Juan Carlos, 28933 Madrid, Spain
3Department of Computer Science and Engineering, University of Gothenburg, 405 30 Gothenburg, Sweden
Corresponding authors: Carlos Carrillo Sanchez (carlos.carrillo@upm.es) and Rafael Capilla (rafael.capilla@urjc.es)

ABSTRACT The stability and longevity of software systems rely on the quality of design decisions over time. In modern software-intensive systems the number of design decisions taken, the dependencies between those decisions, and the number of design alternatives considered, complicate software maintenance and jeopardize the system’s longevity. Despite the existence of complexity metrics applied to code, there is a lack of metrics for design decisions. As estimating the complexity of a set of design decisions is needed for understanding the difficulty of software evolution, this paper proposes and validates a new metric to estimate the complexity of decision networks. The metric is based on decision topologies and provides a way to understand the complexity of decision sets and reason about the maintenance difficulty. We validate our metric empirically in two different ways: (i) evaluating the complexity of two service-based platform systems, and (ii) analyzing the evolution of complexity in four open-source projects and compare how the evolution of complexity affects to the architecture in one of the open-source projects. Our results show that certain network topologies are more difficult to maintain, so we provide a set of tactics to reduce the complexity of design decision networks.

INDEX TERMS Architectural decisions, architectural knowledge, complexity, evolution, software sustainability, stability.

I. INTRODUCTION
Architectural erosion and drift are major concerns that affect the maintenance and evolution of architectures and systems [1], and in order to estimate the “health” of complex systems, we need to measure their quality through different software metrics. Nowadays, most of the current metrics used to identify “bad smells” and anti-patterns rely on code and architecture measures, which are also useful to estimate different quality properties of systems.

Today, the trend to provide more sustainable software (i.e. the capacity of a system to endure overtime) easier to develop and maintain is the way to extend the longevity of systems. However, little research has been carried out to estimate the sustainability of architectural design decisions [2]. An architecture design decision (ADD) can be understood as a decision that can address architecturally significant requirements (i.e. those requirements that have a measurable effect on a system’s architecture) and capture the underpinning reasons that lead to high-quality architectures.

One of these quality indicators is complexity, which provides estimations about how complex a system is in terms of interrelationships between different code pieces and design elements. Therefore, the challenge to reduce the complexity of systems is often a matter to reduce the number of dependencies between the elements along with maintenance and evolution cycles. Although architecture metrics tend to analyze the complexity of architectural entities, to the best of our knowledge there are no approaches that attempt to measure the complexity of decision networks, mainly due to the vaporization of the significant Architectural Knowledge (AK) [3]. Design decision networks are composed of a set of nodes representing the significant architectural design decisions and a set of links connecting the nodes, which represent the interrelationships and dependencies between the decisions.

The goal of this research is to define and evaluate a metric for estimating the complexity of decision structures focused on software architecting [4]. To evaluate the metric, we study...
the evolution of the complexity of design decision networks in order to understand if our complexity metric can be used to estimate the evolution of the complexity of software architectures. We investigate the evolution of the complexity in four open-source projects in order to confirm our initial results, and we provide a set of tactics aimed to reduce the complexity of the design decision networks. Our main results provide evidence that we can use the metric to quantify and isolate simple decision network topologies from the complex ones.

The remainder of this paper is as follows. Section II introduces the related work on architecture knowledge and its relationship to software sustainability. In Section III we describe the modifications to an existing meta-model to measure different quality factors in architectural design decisions and we suggest a metric to estimate the complexity of decision networks. Section IV outlines the initial validation of the metric using different topology networks while the interpretation of the results and discussion are described in Section VI. In Section VII we provide some tactics to reduce complexity in decision networks and Section VIII discusses the limitations of our approach. Finally, we draw conclusions and future research in Section IX.

II. RELATED WORKS

Software complexity accrues over time as part of the natural evolution of the software development process, and sustainable software architectures need to evolve to achieve efficient maintenance change [2]. This section discusses related works about modeling decision networks, the role of sustainability in architecture and decisions as an indicator of its quality, and different approaches describing structural complexity metrics.

A. ARCHITECTURAL DECISION NETWORKS

Because software architecture is the consequence of the design decisions made, these decisions tend to form a network of decisions when the architecture evolves, as one decision may depend on previous decisions made. There are several design rationale representation forms proposed in the past. One of the earliest approaches is IBIS (Issue-based Information System) [5], [6], which forms a network representing (i) the Issues as design questions, (ii) Positions as answers to the questions, and (iii) Arguments as elements used to support or refute the positions. Similar to IBIS is the QOC (Question-Option-Criteria) decision model [7] where the design questions provide different options and the criteria a mechanism to weigh various options. Finally, a recent work describes [8] a framework for documenting architecture design decisions using four viewpoints and including a decision relationship viewpoint for documenting explicitly the links between decisions.

Apart from decision networks represented using IBIS or QOC models, current graph theory describes a graph data structure G as a set of N nodes and L links connecting the nodes. Normally, topological metrics estimate how close are the nodes of a network based on the distances between nodes, which are computed using different formulas, such as the shortest hop-count between two nodes (i.e. in graph theory, hop-count is the number of intermediate nodes between two different nodes) [9], the closeness of a node as the average hop-count from one node to all the others [10] or Girth of a graph, understood as the hop-count of the shortest cycle contained in the graph. Other graph metrics use distances to group similar or related nodes [11], or clustering the neighbors.

Also, the role of complexity is recognized in [12] where the authors propose a set of design quality metrics to assess the design quality of self-management capabilities and where complexity and decision complexity are software quality estimators used in the proposed approach. Nevertheless, the decision complexity formula is computed and used in a different way that the complexity estimator suggested in this research. Although the aforementioned metrics are useful to analyze the similarities between nodes (e.g. related decisions) and other quality factors, there is a lack of specific metrics to estimate the structural complexity of decision networks, such as we investigate in this research. In order to understand better the type of decision networks, we describe below an example using a mobile app we developed.

Example of a Decision Network: As an example we will use a Mobile app for traffic tickets developed between 2010-2011 at Rey Juan Carlos University and extended during 2015-2016. In the first version the mobile app included the first design decisions to the requirements in order to: (i) create a traffic ticket, (i) check driving license, (iii) check traffic plate, and (iv) list traffic tickets. Other design decisions were about sending the tickets to a web server and the selection of Android libraries for the location of the device and capturing the images with the mobile phone. During the next update of the app, we made new design decisions to improve the application. An example of a decision network is shown in Figure 1. Starting from the NewComplaint class (i.e. the striped box), we updated the software architecture supported by new design decisions as shown in the white boxes. The CreateTicket decision encompasses two alternatives about the way to store the data of the traffic tickets (i.e. XML and JSON). Also, this class connects with a decision aimed to create a method to StoreTicket in two alternative databases in the mobile phone, which are SQL Lite and Level DB, and where the decision made was in favor of the SQL Lite database. Tickets are stored in the mobile device when it loses the connection with the webserver. Moreover, the decision to support a method for the Geolocation of the mobile phone impacts on the libraries used by the Android device to invoke the Android LocationManager component. Finally, the SendTicket class in the architecture impacts on the decision to create a method to check the connectivity between the mobile and the webserver phone before tickets are sent, which also impacts on the decision to use the Android ConnectivityChange component. As we can see, the selection of specific components or design alternatives impacts new decisions to select specific libraries too.
B. ARCHITECTURE SUSTAINABILITY

The Karlskrona Manifesto suggests different sustainability dimensions [13], [14] (e.g., from social and environmental) but the focus on software sustainability refers to “as the capacity of a system to endure” [2]. From the software architecture point of view, the development of stable architectures and systems often rely on good and timely design decisions [15] (e.g., this is the case of reference architectures of long-lived systems or well-established domains like the AUTOSAR model for automotive systems).

Also, evaluating the quality of the designs often relies on technical debt estimations as to identify “bad architecture smells” [16], [17], and other architecture anomalies [18], [19] aimed to reduce the number of anti-patterns that may lead to architecture erosion and drift [20], [21]. Moreover, during evolution, the inability of systems to change or add new features may affect also to the stability of the architecture, so the evaluation of instability and change proneness metrics help to understand the effects of design patterns use as an example of good design decisions, such as described in [22]. This relationship between architecture metrics, evolution, and sustainability indicators is also stated by Le at al. [23], while [24] reports an industrial experience about a catalog of architectural metrics to assess architectural fitness and deviation from recommended practices during architecture reviews. The proposed catalog of metrics is organized from different viewpoints including the architectural design decision perspective. Nevertheless, few approaches focus on the relationship between architectural design decisions and the stability of the AK that should lead to good architectures. The work described in [25] suggest a meta-model with explicit trace links between design decisions and other software artifacts, and focus on the impact of the evolution of decisions, while a configurable and extensible metamodel [26] describes explicit relationships between design decisions and quality indicators.

C. STRUCTURAL COMPLEXITY MEASURES

One of the many software metrics that can be used to analyze software maintainability is complexity. For years, several works have proposed methodologies for validating software metrics [27] or process models for software measurement methods [28]. In addition, McCabe’s seminal work [29] suggested a way to estimate software complexity in architectural design through the cyclomatic complexity (i.e. “a measure to control the number of paths through a program”) as a form of structural complexity derived from a flow graph and computed using graph theory. Hence, understanding code and other forms of software complexity imply that higher levels of complexity involve more elements and interconnections between them [30]. Some works [31] suggest that software’s structural complexity depends on the hierarchical organization of the code base but also on the nature of the dependencies between the different interconnected elements. Complexity measurement frameworks like the Structure101,\(^1\) measure cyclic dependencies between packages and the excessive dependencies among packages and classes in order to offer a comprehensive view of software complexity. Also, the work from Reddy et al. [32] describes dependency

\(^1\)Structure 101 is a toolset aimed to estimate the complexity and structure of a software system using the dependencies between modules and modularity indicators. Available at: https://structure101.com/
oriented complexity metrics to improve the quality of designs after refactoring by reducing the degree of coupling between OO artifacts.

A brief history of research on complexity metrics can be found in [33] including a number of measurement criteria for object-oriented software. Ma et al. [34] provide a comprehensive treatment of a hybrid set of complexity metrics for large-scale open-source OO systems to measure complexity at different levels of granularity. The authors highlight the absence of graph-level metrics and network topologies as influencing factors to estimate complexity in OO relationships as indicators of structural defects using the traditional out-degree and in-degree ratio. In the work described in [35], the authors discuss graph measures and the role of numerical weights to highlight the importance of the nodes in a graph but also how cyclic dependencies between nodes affect software quality negatively. However, other recent works like [36] that use graphs in software architecture recovery techniques do not consider weighted dependencies as a way to improve the quality of the recovered architectures.

Other works analyze the structural complexity of feature models, as these kinds of trees could be considered somehow comparable to decision networks described using a Question-Option-Criteria (QOC) tree shape. Broader studies [37] investigate the role of different software metrics such as complexity and stability to measure the understandability of architectural structures and the granularity level (e.g. architecture, package, module) in which they work, but only two of the approaches mentioned deal with design decisions. In another survey [38] the authors report design quality attribute metrics while (cyclomatic) complexity is ranked in the top-frequency positions as a very relevant attribute impacting software maintenance. More recent approaches like [39] describe a suite of complexity metrics to evaluate object-oriented (OO) software projects. The authors suggest five metrics to evaluate class, methods, and code complexity among others to inspect the outer and inner structure of an OO system and to understand the inheritance of levels of the code complexity. Some of the proposed metrics include weights to reflect the cognitive importance of basic control structures to understand software design. Also, the works discussed in [40], presents CodeScene, an open-source tool for predictive analyses and visualization aimed to assess technical debt in automated tests. The proposed tool provides a complexity analysis trend useful to detect candidates to be refactored.

All in all, to the best of our knowledge none of the related works explore the complexity of architecture design decision networks, as this is important to maintain large sets of architectural design decisions but also use these to analyze how complex the resultant architecture could be, as this affects system’s evolution cycles. All previous proposals investigated the complexity in source code and one of them suggests the use of weights but related to code and with a different goal than our approach. Therefore, estimating the complexity of systems and architectures using decision networks has been not addressed by previous works.

Table 1 summarizes a comparison of the most important metrics and approaches related to the estimation of complexity in source code and graphs.

### III. MAINTAINABILITY OF ARCHITECTURAL KNOWLEDGE

Because the granularity of the decisions captured (e.g. a decision could be made to create a class or just an attribute) clearly impacts on the number of decisions captured and dependencies between these decisions, in a recent work [26] we defined a framework describing different criteria that define explicit relationships between quality attributes and metrics that can be used to estimate sustainability indicators both for maintenance and evolution practice areas. One of the consequences derived from capturing the significant architectural design decisions is the relationship between them (a.k.a, internal traceability between design decisions), so we define a decision network as “a network of connected decisions where a change in one decision clearly impacts on other decisions”. Hence, as the architecture evolves, the decision network evolves too and the complexity can increase over time (e.g., like the number of dependencies between packages in Linux kernels [41]).

---

**Table 1. Comparison of the most important software design quality metrics.**

| Approach | Year | Scope | Goal |
|----------|------|-------|------|
| [29] | 1989 | Source code | McCabe’s seminal complexity metric |
| [10][11][9] | 2004, 2004, 2007 | Graphs | Distances between nodes |
| [31] | 2008 | Source code | Software’s structural complexity |
| [32][34][39] | 2009, 2010, 2018 | Source code | Object-oriented complexity metrics |
| [35] | 2012 | Graphs | Estimation of dependency cycles and weights in nodes |
| | 2015 | Architectural structures | Complexity and stability |
| | 2015, 2017 | Design decision networks | Meta-models relating qualities to estimate the evolution of design decisions |
| [37][30][36] | 2017 | Code | Detect interconnections between elements |
| [36] | 2018 | Graphs | Architecture recovery techniques |
| [40] | 2018 | Code | Complexity analysis trend for predictive analysis |
| [12] | 2019 | Graphs | Estimation of decision complexity of self-management capabilities and similarities between nodes |
TABLE 2. Relating the granularity and the structural complexity of design decisions (a more complete version is described in [26]).

| Quality characteristics | Sustainability criterion | Sub-quality characteristic | Metrics |
|-------------------------|--------------------------|----------------------------|---------|
| Maintenance. Keep under control the decisions and trace links to make the decision network maintainable | Granularity of decisions and trace links. Limiting the granularity of decisions to be captured avoids finer-grained decisions and reduces the number of links if the number of alternative decisions is under a reasonable range (e.g., between 1-8) | Complexity. Limiting the number of decisions and trace links can reduce the complexity of decision networks. We analyze the impact on connected and isolated decisions as part of the structural complexity. | NodeCount (number of decisions made), v (number of options or alternative decisions) |
| Stability. The structural stability of a decision network should not be affected by changes in the description of the decisions as we don’t introduce new nodes and dependencies between decisions. | | | Decisions changes per unit of time. |

One way to keep this complexity under control and understand how stable the decisions are is to estimate its complexity based on the number of nodes (i.e., the decisions) and edges (i.e., a relationship between two or more design decisions).

A. QUALITY DECISIONS MODEL

Because maintainability indicators are often a combination of more than one metric and in order to obtain meaningful values, we address in this research an estimation of the complexity of a decision network using two different indicators: NodeCount and EdgeCount [42]. NodeCount and EdgeCount are indicators of size as a form of complexity measures [30]. Metrics for AK maintenance refer to granularity of a decision network, size of the decision model, and the number of decisions captured, while for AK evolution we suggested: ripple effect and instability indicators, decision volatility and timeliness. Based on previous works [26], [43], we describe in Table 2 our view relating the granularity of the design decisions with the complexity of the decision network. In the table, high-level quality attributes like maintenance of design decisions, are delimited by different criteria. One of these criteria is to define the granularity of the design decisions in order to avoid an excessive number of irrelevant decisions to be captured (e.g. a design decision represents the creation of a class attribute or method). Although as maintenance can be quantified using different sub-quality characteristics, we focused on complexity to understand how the size and relationships between design decisions can impact on maintaining the relevant AK. In our approach the connected design decisions are represented as a graph network, so we used the NodeCount and EdgeCount metrics shown in the last column of Table 2 to compute the number of nodes (i.e. the decisions) and their links between them.

As we state in Table 2, the stability of decisions is important for architecture maintenance and evolution cycles. However, even if the complexity of the architecture is constant, the maintainability can change if the stability (e.g. number of changed decisions or frequency of changes per decision) varies. Such stability is often measured via instability metrics in the design. Based on the meta-model described in [26], we provide a simplified and customized version consisting of the UML class-package diagram shown in Figure 2. In the upper boxes, the Architecture model package encompasses basic architecture descriptions and viewpoints according to the standard ISO/IEC 42010 elements, while the DD Core package encompasses a set of classes for capturing the significant design decisions, their rationale, the status of the decisions and dependency types between them. As we didn’t change either one of the packages in the refined version we only show in Figure 2 the empty UML packages. The DD Extensions package contains a reduced version of the classes and attributes described in [26] where we show some of the attributes used by configurable templates for capturing the relevant AK according to the granularity of the decisions.

Finally, the Quality Decisions Model package (lower left part of Figure 2) has been refined to handle a variety of quality attributes described as an enumeration list in the QualityAttributes class. These attributes include complexity and those affecting the evolution of a decision network. This package contains methods to evaluate some quality aspects of design decision networks and particularly those that can be used to estimate the complexity. The ADDQualityMetrics class contains attributes related to the number of design alternatives which can be limited using the maxNumberADDAlternatives attribute to keep under control the number of nodes while the attributes ADDCount and relationshipCount are used to compute the number of nodes (i.e., decisions) and edges (i.e., relationships between decisions) such as we described in Table 2. In addition, the isolationADDWeight attribute represents a factor between 0 and 1 that specifies how relevant isolated decisions are, as this may have a clear impact when we estimate the structural complexity as isolated decisions are expected to have a limited impact when the connected decisions change. The method calculateComplexity() computes the complexity of the network based on NodeCount and EdgeCount and combined with a weight for isolated decisions, as we explain in next section.
B. A COMPLEXITY METRIC FOR ARCHITECTURAL DECISION NETWORKS

According to the ISO/IEC 25010 Std. [44], a quality measure is defined “as a measurement function of two or more values of quality measure elements”, understanding by a quality measurement element “a measure defined in terms of an attribute and the measurement method for quantifying it, including optionally the transformation by a mathematical function”. Thus, in our approach, we define a formula to measure the complexity of design decision networks considering these as a set of decision network as a graph of connected design decisions. Therefore, we suggest a new formula to estimate the complexity of decision networks based on: (i) decisions that can be connected to others, (ii) the number of children (i.e. design alternatives) for a given decision, and (iii) the importance of different types of decisions based on their weights. The rationale for this new formula was based on similar ones used to estimate the complexity of source code but we suggested a new variation of the existing formulas normalizing the results in a range between [0..1] to make the results more manageable as complexity normally tends to increase across evolution except in those cases where a major refactoring is carried out. In addition, and differently from previous formulas, we introduced the notion of weighted decisions to highlight the importance of some nodes in the decision networks. This part of the formula is different from previous approaches estimating only the complexity of the source code.

The protocol we followed to propose the complexity formula given by equation (1) is as follow: (i) we built basic network topologies as those shown in Figure 5 of Section IV, (ii) we produced different variations of the topologies as those shown in Figures 6, 7 and 8 to cover all the different cases including cycles, (iii) we used this formula to calculate the complexity values of the topologies and then we analyze the results to refine the formula, (iv) the last step is aimed to normalize the results of the complexity values relative to the total number of nodes and edges.

\[
\text{Complexity} = \frac{\sum_{j=1}^{DD_{noCh}} \text{noCh}_j + \sum_{i=1}^{DD_{withCh}} \text{Ch}_i}{T_T + DD_T} \quad (1)
\]

The definition of the elements of equation (1) is as follows:
\[
\sum_{j=1}^{DD_{noCh}} DD_{noCh} j_i \text{: Represents the sum of connected decisions without children (i.e. decisions without an alternative).} \\
\sum_{i=1}^{DD_{withCh}} DD_{withCh} i_j \text{: Represents the sum of connected decisions with children.} \\
T_f \text{: Represents the total number of edges connecting the decisions in a graph.} \\
DD_T \text{: Represents the total number of decisions in a graph.}
\]

1) NUMBER OF CHILDREN
Just as structural complexity formulas rely on the number of if constructions, classes or packages, and their dependencies, we compute the number of connected decisions and the number of paths from one decision to another, as an indicator of the complexity in terms of alternative decisions and cycles. Also, instead of considering the length of the paths in the graph (as a way to compute the shortest path between two decisions), we use the outdegree of each decision to estimate such complexity.

2) NORMALIZATION OF COMPLEXITY VALUES
Finally, we normalize the results between [0..1] dividing by the total number of decisions plus edges. Some complexity approaches like McCabe’s formula computes complexity in terms of absolute values but in our case, we preferred to normalize the result to better compare the complexity for different network sizes.

3) WEIGHTED DECISIONS
A decision network can contain different categories of decisions with different degrees of importance. For instance, we could have business decisions (e.g. related to cost of hardware), strategic decisions (e.g. the preferred programming language), or technical decisions (e.g. selection of a platform or software service). Therefore, the software architect can value each decision category with a different scale. Another way to assign weights to the decisions according to their importance or a different scale could be how weakly or strongly decisions are connected. For instance, decisions could be weakly connected (e.g. the selection of a component and the change of a decision would have a limited impact on other decisions). Also, decisions could be connected with a greater degree of importance (e.g. the selection of an API or a protocol can impact on several decisions), and finally, those decisions that are strongly connected because the change in one decision mandates a careful review or change in other related decisions (e.g. change of programming language or a library implies the replacement of the middleware). These examples can be used to understand different difficulty levels when changing decisions and hence, to assign different weights to these categories expressing the importance of connected decisions. For example, an architect could define these levels into simple (i.e. weak connected decisions where a change in one decision does not imply changes to other decisions), medium (i.e. a decision may change if another decision changes) and strong (i.e. a decision changes when other connected decisions changes too). Consequently, we can define a configurable parameter name \( \alpha \) to indicate different degrees of importance for different types or categories of decisions.

IV. VALIDATION
In order to understand the complexity of design decision networks and its impact on architecture, we raised the following research questions:

- **RQ1.** What is the complexity in architectural design decision networks formed by a combination of nodes and edges?
- **RQ2.** How can the complexity of a decision network be used to estimate the complexity in software architecture?

We tested our formula in two different ways. First, we ran several trials using different decision network topologies and accordingly to different degrees of complexity and, second we used two sample decision networks belonging to an architectural decision model of a service-based platform system [45], [46] where a variety of design decisions based on 29 patterns where captured alongside their relationships between the decisions. These patterns were used to describe alternative (exclusive-or) and complementary (inclusive-or) design practices during the decision-making activity [27]. The decisions captured include options that may lead to design solutions and decisions that can trigger other decisions forming all of them a decision network. Both case studies encompass different levels of decision-making (e.g., architecture, platform, integrator, application) and were made at different stages. Figure 3 shows the overview picture of the conversion process aimed to visualize the service-based platform decision networks. We used the data of two sample service-based platforms containing the design decisions and we converted the data into ASCII and CYPHER format so they can be visualized using two different libraries (i.e. Neo4j and JGraphT are implemented in Java). In the second case, we combined the complexity algorithm prior to the visualization using JGraphT to represent the complexity values of both sample networks.

**Preparation of the Data:** We specified the decisions of the two sample models both in ASCII and CYPHER (i.e., a declarative language that enables the creation of graphs and enact queries over the network graph) so they can be visualized using both JGraphT and Neo4j libraries respectively. While JGraphT uses an adjacency matrix to visualize
graphs and enables the allocation of the spatial position of the nodes, Neo4j automatically displays the nodes and their relationships. Both components provide similar visualization facilities for small and large networks. Figure 4 shows an example using CYPHER to create two nodes (i.e. two decisions) and a relationship between them.

### A. DECISION NETWORK TOPOLOGIES

Before testing the formula in the service-based platforms, we enacted some trials with sample networks according to different network topologies. Hence, we created 12 small networks, as representative of typical topologies, including isolated and connected decisions with cycles and alternative paths between the decisions. Our first set of decision topologies shown in Figure 5 displays very elementary decision networks including connected decisions with one or several paths and one isolated decision.

**FIGURE 5. Models 1-2-3 represent connected decisions including one or multiple paths.**

Decision networks are often much more complex than the previous models as they include different paths to reach a given decision. Therefore we could have cyclic topologies, depending on how decisions are connected to others. A typical example of a cyclic topology is a decision that impacts on decisions previously made (e.g. the selection of a specific platform implies the use of a given programming language but after, if new requirements demand supporting another programming language and upgrade to the decision about the platform implies a dependency to the first decision). In figure 6 we represent several examples of cyclic topologies.

The striped circles represent changes from the previous topology when a new link between two decisions is added or changed. Thus, in the figure, model 5 is an evolution of model 4, and model 6 is an evolution of model 5. In the lower part of the figure, model 8 evolves from model 6 adding a new dependency represented by the striped circles.

Normally, in decision-making, one decision may trigger other decisions as design alternatives. In this case, it could happen that instead of having only cycles and alternative paths, the topology adopts a tree-like structure where one decision may have different leaves as design alternatives or even as final decisions made (i.e. a final decision has no alternatives for a given time-frame). This is typical of the adoption of QOC decision models. Consequently, we represent in figure 7 non-cyclic directed graphs and tree decision networks.

### B. RESULTS FROM SAMPLE NETWORK TOPOLOGIES

In order to answer RQ1 we analyze the complexity of sample networks, which are categorized into two different sets and two sample decision networks belonging to an industrial case study. First, we show the estimation of the complexity of the cyclic topologies without isolated decisions. As we can observe in table 3, as the number of cycles and alternative paths increases, the complexity of the network increases too.

Second, we analyze the results of the same networks but including isolated decisions. Table 4 displays the complexity results belonging to those tree-like topologies corresponding to models 9 to 12 and without considering isolated decisions. When we have more final decisions and design alternatives, the decision network turns more complex. In case of networks with multiple paths, our algorithm computes twice or more times a node that can be reached from two or several paths, so the complexity value is a bit high. This makes sense in this kind of topologies as extra paths reaching a given decision increase the number of possibilities and therefore the complexity of the network.

---

**TABLE 3. Complexity of cyclic topologies without isolated decisions.**

| Model | Description                              | Complexity |
|-------|------------------------------------------|------------|
| Model 4 | Network with 1 cycle                     | 0.56       |
| Model 5 | Network with 2 cycles and 2 alternative paths | 0.59       |
| Model 6 | Network with 3 cycles and 3 alternative paths | 0.61       |
| Model 7 | Network with 4 cycles and 2 alternative paths | 0.61       |
| Model 8 | Network with 4 cycles and 6 alternative paths | 0.63       |

---

2In case of networks with multiple paths, our algorithm computes twice or more times a node that can be reached from two or several paths, so the complexity value is a bit high. This makes sense in this kind of topologies as extra paths reaching a given decision increase the number of possibilities and therefore the complexity of the network.
increase the number of design choices. Therefore, maintaining a large number of design alternatives in QOC models could increase the maintenance effort of all the decisions which must be revisited when a decision is modified.

Equation (2) provides an example of how the value of the complexity metric defined in formula (1) is calculated for Model 10 (see Figure 8)

\[
\text{Complexity} = \frac{\sum_{j=1}^{DD_{\text{noCh}}} \text{noCh}_j + \sum_{i=1}^{DD_{\text{withCh}}} \text{Ch}_i}{T_T + DD_T} = \frac{(C + G + H + I) + (\text{Ch}_A + \text{Ch}_B + \text{Ch}_D + \text{Ch}_E + \text{Ch}_F)}{4 + (2 + 2 + 1 + 3 + 2) + 10 + 9} = \frac{14}{19} = 0.74
\]

From our initial experiments, we got significant results of the trends in complexity for different decision networks and we learned how weighted isolated decisions may impact the proposed complexity formula. However, in order to confirm the early results using small sample decision networks, we describe in the next section the outcome using two real case studies that provide bigger and more realistic networks from an industrial case study.

FIGURE 6. Models 4-8 represent more complex decision networks with multiple cycles and alternative paths.

FIGURE 7. Models 9-12 representing non-cyclic direct graphs and tree-like topologies.

TABLE 4. Complexity of directed graphs and tree-like decision networks without isolated decisions.

| Model   | Description                                      | Complexity |
|---------|--------------------------------------------------|------------|
| Model 9 | Directed graph with 3 final decisions            | 0.68       |
| Model 10| Directed graph with 4 final decisions and 1 decision with 3 alternatives | 0.74       |
| Model 11| Tree network with 5 final decisions              | 0.76       |
| Model 12| Directed graph with 5 final decisions and 1 decision with 3 alternatives | 0.79       |

FIGURE 8. Model 10 details.
C. RESULTS FROM TWO SERVICE-BASED PLATFORM MODELS

Finally, we report the results of the complexity of two sample networks belonging to an industrial case study of two service-based platforms (SBP-1 and SBP-2) [45], [46], that we described at the beginning of the validation section. We used the Neo4j library to visualize the SBP-1 model such as we see in figure 9, which encompass 61 nodes and 70 relationships. The second model, not shown in the figure, encompasses 33 connected nodes and 50 relationships. The complexity values are shown in Table 5.

In order to test the impact $\alpha$, we defined two categories of decisions: those with 1 or 2 alternatives with an $\alpha$ equal to 0.4 and those with 3 or more alternatives with an $\alpha$ equal to 0.8, as a way to reflect its importance in the decision network because changing decisions with more number of alternatives in QOC topologies should be more complex than those deci-

| Model | Description | Complexity |
|-------|-------------|------------|
| SBP-1 | Network with 61 vertexes and 70 relationships | 0.67 |
| SBP-2 | Network with 33 vertexes and 50 relationships | 0.71 |

3We selected the values of $\alpha$ to reflect two different levels of importance of the decisions according to their number of alternatives.
sions with a fewer number of alternative decisions. Therefore, for model SBP-1 we found 56 decisions with 1 or 2 alternatives and 5 with 3 or more, and for model SBP-2, we found 27 decisions having 1 or 2 leaves and 6 decisions with 3 or more alternatives. Consequently, the complexity values for SBP-1 and SBP-2 using the weights, are 0.41 and 0.45 respectively. The explanation of the new elements in the complexity formula (equation (3)) is as follows.

\[
\text{Complexity} = \frac{\sum_{i=1}^{G} DD_{\text{noCh}}}{T + DD_T} + \frac{\sum_{k=1}^{G} (\alpha_k * \sum_{i=1}^{G} DD_{\text{withCh}} Ch_{ki})}{T + DD_T}
\]

\( G \): Describes the number of categories of design decisions. Each category has its own value of \( \alpha \).

\( \alpha_k \): Refers to the importance of a category of design decisions and has a specific weight or value associated with its importance.

\( \sum_{i=1}^{G} DD_{\text{withCh}} Ch_{ki} \): Represent the sum of connected decisions with children belonging to a specific category.

\( DD_{\text{withCh}} Ch_{ki} \): Represent the number of connected decisions with their children decisions belonging to a specific category.

We show in figure 9, the decisions, and relationships of the SBP-1 service-based platform. Questions are named as Q1, Q2, and so on, and the green arrow indicates the available options for each question. The yellow arrows mean a relationship between the option selected and the potential solution chosen while the grey arrows belong to triggers activated by a decision or a question. Also, the red nodes belong to design patterns and the grey ones to decisions expressed using a QOC model.

One interesting consequence is that complexity of graph networks does not depend exclusively on the size of the model but also on how each particular graph is modeled. Compared to the sample topology of Model 11, the complexity of both service-based platforms networks decreases because the value of \( \alpha \) assigns less importance to decisions with 1 or 2 alternatives, which are the majority in both examples. However, this weight is arbitrary and can be assigned by the software engineer in case he or she wants to reflect on a different importance for certain groups or categories of decisions. In addition, we visualize in figure 10 one of the service-based platforms (SBP-1) using the JGraphT library to show the differences of visualization and providing additional details about the type of the trace links, which in this case belongs to a QOC decision model. The figure offers better details about the type of dependencies between decisions rather than using Neo4j.

**D. COMPARISON TO OTHER METRICS**

As to the best of our knowledge, there are no previous experiences in estimating the complexity of design decision networks, it becomes hard to compare our results to previous metrics. Nonetheless, the authors in [47] assess on modularity and stability indicators in software architecture-level decisions, and they describe a suite of architecture-level metrics to automate quantitative stability and modularity. A more recent study [48] discusses the detection of architecture smells as hotspot patterns that can be detected in complex systems using the notion of Design Rule Spaces (DSRs) which contain a set of files and their dependencies. In the proposed approach the authors suggest formulas to analyze five hotspot patterns to detect overly complexity in open-source systems. In our approach it gets more difficult to compare to other complexity formulas because (i) we focus on decision networks instead of the complexity of source code structures even if both can be described in terms of nodes and edges, (ii) we use weights for isolated nodes, and (iii) we normalize the results to the size of the model instead of reporting absolute numbers of complexity like McCabe’s formula [29]. However, estimating the complexity of different topology networks using, for instance, McCabe’s formula produces a similar trend in the way that tree-like structures are more complex according to McCabe’s results than cyclic structures, which confirms the validity of the proposed formula, but normalized to the size of the model. We didn’t observe outliers when the number of elements increases, so the trend between McCabe’s formula and ours is similar.

Moreover, in order to provide additional confirmation of our results we simulated four sample decision networks estimating their complexity using the Structure101 framework. Structure101 measures Fat (i.e. too much complexity in functions, classes and methods) and Tangles as indicators of excessive complexity (XS) of source code structures by extending McCabe’s cyclomatic complexity. As a result, we couldn’t measure the tangles using our sample decision networks (i.e. models 4, 5, 11, and 12) because Structure101 uses invocations between packages and methods to analyze the existing cycles and in our decision networks we define relationships between the nodes. In the case of Fat metrics, Structure101 analyzes such complexity in the design, leaf packages, classes, and methods.

In addition to the two research questions investigated in this paper, we wanted to compare the trends of the complexity in our sample topologies with Structure101. The tool Structure101 defines two threshold values (e.g. Fat(method) equal to 10-15 and 120 for XS) as the norm from the industry to skip those cases of low complexity, so we chose Fat(method) equal to 10 to reduce the bias of not complex bifurcations in our sample networks. Our results showed that with Structure101 the XS value in terms of Fat(class) complexity of models 4 and 5 is 38 and 73 respectively (i.e.: 0.56 and 0.59 using our formula) which confirms the trends that model 5 is more complex than model 4. With respect to models 11 and 12, the results provided by Structure101 are 48 and 72 respectively, as confirmation that model 12 results more complex than model 11 (i.e. 0.76 and 0.79 using our formula). In both cases, models 11 and 12 are more complex than model 4, and only model 12 exhibits a similar complexity than model 5 using.
Structure101. As a remark, the order of the magnitudes of complexity produced by Structure101 and our algorithm is different, as our formula normalizes the values between 0 and 1. However, we just wanted to show that the complexity trend is almost the same using two different ways to measure the complexity values.

In Table 6 we compare for four different sample topologies the results of the complexity estimated using our metric and the values provided by McCabe and the Structure101 frameworks. Although the scales of complexity are different, the trend of increasing complexity is similar in the three cases.

**TABLE 6.** Comparison of the complexity values estimated using different metrics and our approach.

| Topologies used | McCabe’s complexity [29] | Structure101 framework | Complexity values using our approach |
|-----------------|---------------------------|-------------------------|--------------------------------------|
| Model 4         | 2                         | 38                      | 0.56                                 |
| Model 5         | 3                         | 73                      | 0.59                                 |
| Model 11        | 5                         | 48                      | 0.76                                 |
| Model 12        | 5                         | 72                      | 0.79                                 |

**V. RESULTS FROM OPEN-SOURCE PROJECTS**

Additionally to the industrial SBP-1 and SBP-2 networks, we validated our complexity formula with four
TABLE 7. Evolution of the complexity of open-source projects.

| Repository | Version | Year    | Vertexes | Relationships | Size (MB) | Complexity |
|------------|---------|---------|----------|---------------|-----------|------------|
| Catroid    | 0.5.0a  | May 2011| 211      | 890           | 1.76      | 0.8420     |
| Catroid    | 0.8.0   | Jun 2013| 1025     | 5009          | 12.5      | 0.8775     |
| Catroid    | 0.9.10  | Aug 2014| 1157     | 5966          | 27.1      | 0.8794     |
| SonarQube  | 5.0.1   | Feb 2015| 1000     | 3816          | 1031      | 0.8569     |
| SonarQube  | 5.2     | Nov 2015| 1035     | 3962          | 1212      | 0.8523     |
| SonarQube  | 5.3     | Jan 2016| 1036     | 3964          | 1233      | 0.8524     |
| SonarQube  | 5.5     | May 2016| 1017     | 3829          | 1282      | 0.8498     |
| SonarQube  | 6.0     | Aug 2016| 1018     | 3839          | 1392      | 0.8501     |

- **dex2jar**: 0.0.9.5 (Jan 2012) Vertexes: 899, Relationships: 3708, Size (MB): 1.82, Complexity: 0.8821
- **dex2jar**: 0.0.9.7 (Jan 2012) Vertexes: 807, Relationships: 5454, Size (MB): 1.68, Complexity: 0.8869
- **dex2jar**: 0.0.9.10 (Oct 2012) Vertexes: 837, Relationships: 5642, Size (MB): 1.79, Complexity: 0.8866
- **dex2jar**: 0.0.9.12 (Dec 2012) Vertexes: 839, Relationships: 5669, Size (MB): 1.79, Complexity: 0.8868
- **dex2jar**: 0.0.9.15 (Jun 2013) Vertexes: 825, Relationships: 5626, Size (MB): 1.77, Complexity: 0.8881
- **hadoop**: 0.12.0 (Mar 2007) Vertexes: 511, Relationships: 3609, Size (MB): 24.7, Complexity: 0.9036
- **hadoop**: 0.13.0 (Jun 2007) Vertexes: 583, Relationships: 4214, Size (MB): 26.8, Complexity: 0.9066
- **hadoop**: 0.20.0 (Apr 2009) Vertexes: 1316, Relationships: 11504, Size (MB): 118, Complexity: 0.9175
- **hadoop**: 1.0.0 (Dec 2011) Vertexes: 2034, Relationships: 18010, Size (MB): 155, Complexity: 0.9185
- **hadoop**: 1.1.0 (Oct 2012) Vertexes: 2178, Relationships: 19665, Size (MB): 182, Complexity: 0.9193

FIGURE 11. Complexity between different versions of open-source projects including SBP-1 and SBP-2.

open-source projects. Table 7 shows four different open-source projects and their versions over time. For each project, we run ARCADE using the source code as input to compute the number of vertexes (i.e. origin and destination nodes in the graph) and dependencies between them and removing the duplicate elements. After, we exported the results in an appropriate format to be used by our complexity formula to provide the complexity values for each release analyzed. However, as we can see in the complexity results in the last column, we didn’t find projects with complexity values lower than 0.84. In the table, we show the size of each project per version in MBytes according to the package downloaded. The column named Vertexes displays the number of the classes retrieved (i.e. assuming we consider a class like a node in a decision network). We only took into account the source code of each software project but we excluded from our analysis supplementary code like test cases or web pages.

We also display graphically in Figure 11 the complexity of SBP-1 and SBP-2 networks compared to the complexity of open-source projects. As we can see, the high number of relationships between the nodes in OSS projects is the reason why the complexity values are so high, and also why the number of relationships is not displayed in the figure for SBP networks. We can also observe in Figure 11 the evolution of complexity values (grey and black lines superseding the bars). However, we didn’t find OSS projects with values of complexity between 0.71 and 0.85. Finally, for a given
open-source project we noticed that the evolution of the complexity shows similar values between different versions.

**Impact of Complexity in the Architecture:** In order to answer to **RQ2** we investigated the impact of complexity in the architecture when decisions change. From the four OSS projects analyzed, we selected one of the four projects, which is **Catroid** [49], to study the complexity of a subset of its architecture. Catroid is an open-source visual programming language and environment that allows young users to manipulate and develop their own animations and games using Android phones or tablets. Catroid enables wireless control to external hardware via Bluetooth as well as Parrot’s drone via WiFi. In order to extract the number of vertexes and relationships of the open-source projects, we used the
FIGURE 13. Evolution of package Brick of Catroid version from 0.5.0 to 0.9.10.

ARCADE tool [50], [51] to find the dependencies between the architectural elements of the open-source projects (shown in Table 7) and use these dependencies as input for our instability algorithm. In addition, we used Visual Paradigm\(^4\) to represent graphically the reversed architecture. Figure 12 shows a UML class-package diagram containing a subset of the architecture reversed using ARCADE and displayed using the Visual Paradigm tool. For space reasons, we only show the names of the classes. The classes, packages and their dependencies shown in Figure 12 are motivated by the underlying design decisions, which can be modeled as a decision network such as the example shown in Figure 10.

We focused on the evolution of two different versions (i.e. 0.5.0a and 0.9.10) of Catroid and specifically in the Brick package which belongs to the Content package. On the left side of Figure 13 we can see a subset of version 0.5.0 where the classes displayed in white color will disappear in version 0.9.10 while the classes in black will remain in version 0.9.10. On the right side of Figure 13 we display the new classes in grey color that appear in this version but only those connected to the Brick package. As the number of vertexes (i.e. classes considered as new design decisions) and their relationships increase significantly between both versions the complexity increases too from 0.8420 to 0.8794. Although the order of the magnitude of the difference in the complexity values seems to be not big (i.e. 0.0374), the number of classes, and hence design decisions, have increased. We are aware that one design decision could lead to the creation of several classes (e.g. the use of a design pattern represented by several classes). However, we need to clarify that part of the exponential growth of vertexes and relationships between the two versions analyzed is due to the use of extra packages in the repositories which increase the number of classes. Therefore, the estimation of the complexity between both versions could be different if we only consider the essential components for each version.

\(^4\)Visual Paradigm is a UML editor available at https://visual-paradigm.com

FIGURE 14. Evolution of complexity in decision networks.

VI. INTERPRETATION OF RESULTS

There are several conclusions we can distill from the interpretation of the results estimating the complexity of decision models. From the results in Tables 3 and 4 belonging to the sample decision models, we summarized the complexity values and the trend of complexity in Figure 14. In the figure,
the complexity starts increasing as we add more connected decisions but we have the first point of inflection in Model 6 when we add our first cycle. As long as we add more cycles or alternative paths the complexity increases because we add more connected decisions and cycles. From the perspective of the topology, directed graphs without cycles and tree networks (i.e., Models 9, 10, 11, 12) exhibit higher values of complexity than cyclic structures (i.e., Models 4, 5, 6, 7, 8) as they have more final decisions. Hence, the maintainability of certain topologies regarding its complexity varies when more design alternatives are added. Also, the complexity decreases when we eliminate relationships between the decisions as this reduces the number of cycles (e.g., Model 8 with 4 cycles versus Model 5 with only 2 cycles and Model 4 with 1 cycle).

VII. TACTICS TO REDUCE COMPLEXITY

We summarize in this section a set of tactics derived from our experiments as guidelines to reduce the complexity of decision networks.

A. TACTIC 1. NUMBER OF ALTERNATIVE DECISIONS

Keep the number of alternative decisions under control plays in favor of better maintainability and hence a reduction in the complexity. Having less tree-like topologies and less number of final decisions will be beneficial for the reduction of complexity as well. Hence, decreasing the number of children (i.e., links to alternative decisions) in the formula tends to reduce the complexity values of the decision network.

B. TACTIC 2. NUMBER OF CYCLES

Reduce the number of cycles in the graph as a way to have a fewer number of alternative paths. Similarly to tactic 1, a reduction in the number of alternatives in the formula leads having a fewer of paths and therefore decreases the complexity of networks with cycles.

C. TACTIC 3. REDUCE TREE NETWORKS AS QOC MODELS

Topologies using tree networks (e.g., those using the Question-Option-Criteria – QOC- approach) seems more complex to maintain than those networks formed by cycles or alternative paths with few leaves.

D. TACTIC 4. REMOVE OBSOLETE DECISIONS

We suggest to remove the decisions that are outdated (i.e., packages and classes not used) as this will reduce the number of dependencies.

VIII. LIMITATIONS

We believe that complexity is one of the many factors that impact on the maintainability of design decision models. As stated in [52] we discuss the following limitations to our work. The complexity formula was defined in terms of nodes and edges and normalized to the size of the model. However, the definition of the $\alpha$ factor representing different levels of importance of categories of decisions can be a distorting factor when the value of $\alpha$ increases.

Although we believe there is a causal relationship between the variables of the study, i.e. the number of nodes and edges to determine the complexity of a decision network, we are not totally sure how the size of the models influences the complexity results. We tried a significant variety of topologies in order to cover the majority of the cases having different shapes, so from this side, we believe the design of the experiment was correct in order to mitigate the threat of considering only one type of decision networks. However, one factor that can distort some of the results could be the influence of different types of relationships between the decisions as this requires a deeper investigation. Some limitations of the metric, as rare cases, could appear in very small decision networks where the complexity values are high as possible anomalies.

At present, there is no practical way to define whether one should document finer grain or unimportant decisions, so the results of the complexity formula might be subject to interpretation. Therefore, independently of the size of the decision networks, we suggest documenting only the significant decisions. As few studies provide real decision models it becomes challenging to get additional case studies with a larger number of decisions. In addition, the lack of versioning of the decision models makes it difficult to analyze the evolution of the complexity over time, so this is why we studied open-source projects.

Finally, the complexity values of decision models in open-source projects could be distorted as we estimate such complexity including all the packages and libraries imported by an open-source project. Therefore, it would be better to provide a more accurate estimator using only the real number of classes and packages needed for that project. Also, the lack of open-source decision networks led us to use open-source projects and model our network based on the classes extracted from each open-source project.

IX. CONCLUSION AND FUTURE RESEARCH

This seminal experiment estimating the complexity of architectural decision models is relevant to estimate the complexity of architectures based on early design decisions made. Therefore, understanding how this complexity evolves over time we can also predict how complex an architecture evolves too. From observations, we need a combination of more metrics to provide more meaningful indicators about the complexity of decision networks and the system’s architecture. Also, we didn’t evaluate the complexity of the decisions per se, but this shouldn’t affect the complexity of the topology we want to maintain. One validity threat may happen if software architects can create different topologies using the same decisions and hence, end up with a different complexity measure. However, designing software architectures is a creative task for which more than one design solutions are possible.

Regarding the granularity of the decisions, documenting very small decisions that have little impact on the
architecture only increases the complexity and maintainability of the topology. Complex topologies can reduce their complexity if we limit the number of alternative decisions and cycles. In this way, investigating the relative influence of cycles, alternatives, and size on complexity, and how the graph size and complexity are orthogonal, is also worthy to explore. Consequently, one clear benefit of estimating the complexity of architectural design decision networks is to uncover which topologies are more suitable and less complex to model architectural decisions. Another interesting outcome is to predict the complexity of the software architecture based on the evolution of the complexity of the decision networks, which affects software maintenance tasks.

As future research, we are experimenting with cost factors about the capturing effort of the AK for different software development strategies. We attempt to combine this approach with new metrics (such as stated in [23], [24]) based on ripple effect and instability measures to estimate the impact of changes in decision models, such as we described in a recent work [53] as a key factor to analyze the stability over time. We also plan to integrate the results of the complexity values with sustainability metrics in order to, for instance, measure the actual maintenance effort and use these results to argue for the precision of our metric and predict better the complexity of decision networks. Although in this research we only provided evidence of projects increasing in complexity, we plan to investigate also those cases where the complexity decreases significantly over time. Finally, exploring how to assign different weights to the decisions for capturing nodes with different importance, seems to be an interesting research path as, for instance, we can make decisions in different programming languages.

REFERENCES

[1] J. V. Gurp, J. Bosch, and S. Brinkkemper, “Design erosion in evolving software products,” in Proc. ELISA Workshop, vol. 134, 2003, pp. 1–6.
[2] C. C. Venters, R. Capilla, S. Betz, B. Penzenstadler, T. Crick, S. Crouch, E. Y. Nakagawa, C. Becker, and C. Carrillo, “Software sustainability: Research and practice from a software architecture viewpoint,” J. Syst. Softw., vol. 138, pp. 174–188, Apr. 2018.
[3] R. Capilla, A. Jansen, A. Tang, P. Agerviour, and M. A. Babar, “10 years of software architecture knowledge management: Practice and future,” J. Syst. Softw., vol. 116, pp. 191–205, Jun. 2016.
[4] M. Staron, “Automotive software architectures,” in Automotive Software Architectures. Cham, Switzerland: Springer, 2017, pp. 33–39.
[5] W. Rittel and H. Rittel, Issues as Elements of Information Systems, Berkeley, CA, USA: Univ. of California, 1970.
[6] H. A. Dutot, R. McCall, I. Mistrik, and B. Paech, Rationale Management in Software Engineering, Berlin, Germany: Springer-Verlag, 2006.
[7] A. MacLean, R. Young, V. Bellotti, and T. Moran, “Questions, options, and criteria: Elements of design space analysis,” Hum.-Comput. Interact., vol. 6, no. 3, pp. 201–250, Sep. 1991.
[8] U. van Heesch, P. Agerviour, and R. Hiliard, “A documentation framework for architecture decisions,” J. Syst. Softw., vol. 85, no. 4, pp. 795–820, Apr. 2012.
[9] L. D. Costa, F. A. Rodrigues, G. Travieso, and P. R. Villas Boas, “Characterization of complex networks: A survey of measurements,” Adv. Phys., vol. 56, no. 1, pp. 167–242, Jan. 2007.
[10] D. Koschützki, K. A. Lehmann, L. Peeters, S. Richter, D. Tenfelde-Podehl, and O. Zlotowski, “Centrality indices,” in Network Analysis: Methodological Foundations. Heidelberg, Germany: Springer, 2004, pp. 6–12.
[11] P. Doreian, V. Batagelj, and A. Ferligoj, “Generalized blockmodeling of two-mode network data,” Social Netw., vol. 26, no. 1, pp. 29–53, Jan. 2004.
[12] A. Jaleel, S. Arshad, M. Shoaib, and M. Awaiss, “Design quality metrics to determine the suitability and cost-effect of Self-* capabilities for autonomic computing systems,” IEEE Access, vol. 7, pp. 139759–139772, 2019.
[13] S. Betz, C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff, and C. Venters, “Sustainability debt: A metaphor to support sustainability design decisions,” in Proc. 4th Int. Workshop Requirements Eng. Sustain., 2015, pp. 1–10.
[14] P. Lago, S. A. Koçak, I. Crnkovic, and B. Penzenstadler, “Framing sustainability as a property of software quality,” Commun. ACM, vol. 58, no. 10, pp. 70–78, Sep. 2015.
[15] U. Zdun, R. Capilla, H. Tran, and O. Zimmermann, “Sustainable architectural design decisions,” IEEE Softw., vol. 30, no. 6, pp. 46–53, Nov. 2013.
[16] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic, “Identifying architectural bad smells,” in Proc. 13th Eur. Conf. Softw. Maintenance Reeng., Mar. 2009, pp. 255–258.
[17] D. Le and N. Medvidovic, “Architectural-based speculative analysis to predict bugs in a software system,” in Proc. 38th Int. Conf. Softw. Eng. Companion (ICSE), New York, NY, USA: ACM, 2016, pp. 807–810.
[18] H. Koziolek, “Sustainability evaluation of software architectures,” in Proc. Joint ACM SIGSOFT Conf.–QoSA ACM SIGSOFT Symp.–ISARC Qual. Softw. Archit.–QoSA Architecting Crit. Syst.–ISARC–QoSA–ISARC3. New York, NY, USA: ACM, 2011, pp. 3–12.
[19] I. Macia, R. Arcoverde, A. Garcia, C. Chavez, and A. von Staa, “On the relevance of code anomalies for identifying architecture degradation symptoms,” in Proc. 16th Eur. Conf. Softw. Maintenance Reeng., Mar. 2012, pp. 277–286.
[20] J.-L. Letouzey and M. Ilikiwicz, “Managing technical debt with the SQALE method,” IEEE Softw., vol. 29, no. 6, pp. 44–51, Nov. 2012.
[21] Z. Li, P. Liang, P. Agerviour, N. Guelfi, and A. Ampatzoglou, “An empirical investigation of modularity metrics for indicating architectural technical debt,” in Proc. 10th Int. ACM Sigsoft Conf. Qual. Softw. Archit. (QoSA). New York, NY, USA: ACM, 2014, pp. 119–128.
[22] A. Ampatzoglou, A. Chatziigeorgiou, S. Charalampidou, and P. Agerviour, “The effect of GoF design patterns on stability: A case study,” IEEE Trans. Softw. Eng., vol. 41, no. 8, pp. 781–802, Aug. 2015.
[23] D. M. Le, C. Carrillo, R. Capilla, and N. Medvidovic, “Relating architectural decay and sustainability of software systems,” in Proc. 13th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA), Apr. 2016, pp. 178–181.
[24] O. Zimmermann, “Metrics for architectural synthesis and evaluation—requirements and compilation by Viewpoint. An industrial experience report,” in Proc. IEEE/ACM 2nd Int. Workshop Softw. Archit. Metrics. Piscataway, NJ, USA: IEEE, May 2015, pp. 8–14.
[25] I. Malavolta, H. Muccini, and V. Smrithi Rekha, “Supporting architectural design decisions evolution through model driven engineering,” in Software Engineering for Resilient Systems, E. A. Troubitsyna, Eds. Berlin, Germany: Springer, 2011, pp. 63–77.
[26] C. Carrillo, R. Capilla, O. Zimmermann, and U. Zdun, “Guidelines and metrics for configurable and sustainable architectural knowledge modelling,” in Proc. Eur. Conf. Softw. Archit. Workshops (ECSAW). New York, NY, USA: ACM, 2015, pp. 63:1–63:5.
[27] N. F. Schneidewind, “Methodology for validating software metrics,” IEEE Trans. Softw. Eng., vol. 18, no. 5, pp. 410–422, May 1992.
[28] J.-P. Jacquet and A. Abran, “From software metrics to software measurement methods: A process model,” in Proc. IEEE Int. Symp. Softw. Eng. Standards, Jan. 1997, pp. 128–135.
[29] T. J. McCabe and C. W. Butler, “Design complexity measurement and testing,” Commun. ACM, vol. 32, no. 12, pp. 1415–1425, Dec. 1989.
[30] V. Antinayan, M. Staron, and A. Sandberg, “Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time,” Empirical Softw. Eng., vol. 22, no. 6, pp. 3057–3087, Dec. 2017.
[31] R. S. Sangwan, P. Vercellone-Smith, and P. A. Laplane, “Structural epochs in the complexity of software over time,” IEEE Softw., vol. 25, no. 4, pp. 66–73, Jul. 2008.
[32] K. N. Reddy and A. A. Rao, “A quantitative evaluation of software quality enhancement by refactoring using dependency oriented complexity metrics,” in Proc. 2nd Int. Conf. Emerg. Trends Eng. Technol., Dec. 2009, pp. 1011–1018.
[33] T. Honglei, S. Wei, and Z. Yanan, “The research on software metrics and software complexity metrics,” in Proc. Int. Forum Comput. Sci.-Technol. Appl., Dec. 2009, pp. 131–136.
[34] Y.-T. Ma, K.-Q. He, B. Li, J. Liu, and X.-Y. Zhou, “A hybrid set of complexity metrics for large-scale object-oriented software systems,” J. Comput. Sci. Technol., vol. 25, no. 6, pp. 1184–1201, Nov. 2010.

[35] P. Bhattacharya, M. Biofotou, I. Neamtiu, and M. Faloutsos, “Graph-based analysis and prediction for software evolution,” in Proc. 34th Int. Conf. Softw. Eng. (ICSE), Jun. 2012, pp. 419–429.

[36] T. Litellier, D. Chollak, J. Garcia, L. Tan, D. Rayside, N. Medvidovic, and R. Kroeger, “Measuring the impact of code dependencies on software architecture recovery techniques,” IEEE Trans. Softw. Eng., vol. 44, no. 2, pp. 159–181, Feb. 2018.

[37] S. Stevanetic and U. Zdun, “Software metrics for measuring the understandability of architectural structures: A systematic mapping study,” in Proc. 19th Int. Conf. Eval. Assessment Softw. Eng. (EASE), 2015, pp. 21:1–21:14.

[38] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, M. Galster, and P. Avgeriou, “A mapping study on design-time quality attributes and metrics,” J. Syst. Softw., vol. 127, pp. 52–57, May 2017.

[39] S. Misra, A. Adewumi, L. Fernandez-Sanz, and R. Damasevicius, “A suite of object oriented cognitive complexity metrics,” IEEE Access, vol. 6, pp. 8782–8796, 2018.

[40] A. Tornhill, “Assessing technical debt in automated tests with code-scene,” in Proc. IEEE Int. Conf. Softw. Text., Verification Validation Workshops (ICSTW). Västerås, Sweden: IEEE Computer Society, Apr. 2018, pp. 122–125.

[41] I. Herraiz, G. Robles, R. Capilla, and J. M. Gonzalez-Barahona, “Managing libre software distributions under a product line approach,” in Proc. 32nd Annu. IEEE Int. Comput. Softw. Appl. Conf., Jul./Aug. 2008, pp. 1221–1225.

[42] T. Mens and M. Lanza, “A graph-based metamodel for object-oriented software metrics,” Electron. Notes Theor. Comput. Sci., vol. 72, no. 2, pp. 57–68, Nov. 2002.

[43] R. Capilla, E. Y. Nakagawa, U. Zdun, and C. Carrillo, “Toward architecture knowledge sustainability: Extending system longevity,” IEEE Softw., vol. 34, no. 2, pp. 108–111, Mar. 2017.

[44] Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (Square)—System and Software Quality Models, Standard 25010 International Standard, 2011, pp. 1–44.

[45] I. Lytra, S. Sobering, and U. Zdun, “Architectural decision making for service-based platform integration: A qualitative multi-method study,” in Proc. Joint Work. IEEE/IFIP Conf. Softw. Archit. Eur. Conf. Softw. Archit., Aug. 2012, pp. 111–120.

[46] I. Lytra, S. Sobering, H. Tran, and U. Zdun, “A pattern language for service-based platform integration and adaptation,” in Proc. 17th Eur. Conf. Pattern Lang. Programs (EuroPLoP). New York, NY, USA: ACM, 2012, pp. 1–27.

[47] K. Sethi, Y. Cai, S. Wong, A. Garcia, and C. Sant’Anna, “From retrospect to prospect: Assessing modularity and stability from software architecture,” in Proc. Joint Work. IEEE/IFIP Conf. Softw. Archit. Eur. Conf. Softw. Archit., Sep. 2009, pp. 269–272.

[48] R. Mo, Y. Cai, R. Kazman, and L. Xiao, “Hotspot patterns: The formal definition and automatic detection of architecture smells,” in Proc. 12th Work. IEEE/IFIP Conf. Softw. Archit. Montreal, QC, Canada: IEEE Computer Society, May 2015, pp. 51–60.

[49] W. Slany, “Catroid: A mobile visual programming system for children,” in Proc. 11th Int. Conf. Interact. Design Children (IDC). New York, NY, USA, 2012, pp. 300–303.

[50] D. M. Le, P. Behnamghader, J. Garcia, D. Link, A. Shahbazian, and N. Medvidovic, “An empirical study of architectural change in open-source software systems,” in Proc. IEEE/ACM 12th Work. Conf. Mining Softw. Repositories, May 2015, pp. 235–245.

[51] A. Shahbazian, Y. K. Lee, D. Le, Y. Brun, and N. Medvidovic, “Recovering architectural design decisions,” in Proc. IEEE Int. Conf. Softw. Archit. (ICSA). Apr. 2018, pp. 9509–9595.

[52] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” Empirical Softw. Eng., vol. 14, no. 2, pp. 131–164, Apr. 2009.

[53] C. Carrillo and R. Capilla, “Ripple effect to evaluate the impact of changes in architectural design decisions,” in Proc. 12th Eur. Conf. Softw. Archit., Companion, 2018, pp. 41:1–41:8.

CARLOS CARRILLO SANCHEZ received the B.Sc. degree in electronic engineering from the Universidad Politécnica de Madrid, Spain, in 1989, the M.Sc. degree in computer system from University Rey Juan Carlos, in Madrid, and the Ph.D. degree in computer science from the Universidad Politécnica de Madrid, in 2017. He is currently an Associate Professor with the Department of Telematic Engineering and Electronic, Universidad Politécnica de Madrid. He has served as a Reviewer for the Journal of Software Evolution and Process and the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING journal. His research interests include architectural knowledge, technical debt, and software quality among others. His professional activities concern with consultancy on forensics and digital evidence of security issues.

RAFAEL CAPILLA received the B.Sc. degree in computer science from the Universidad de Sevilla and the Ph.D. degree in computer science from Universidad Rey Juan Carlos, Madrid, Spain, in 2004. He is currently an Associate Professor of software engineering with Universidad Rey Juan Carlos. He is the coauthor of more than 100 peer-reviewed conference and journal articles and co-editor of the Springer book (2013) Systems and Software Variability Management: Methods, Concepts and Tools. He is a Regular Reviewer of several well-known journals (e.g., the Journal of Systems and Software, Information and Software Technology, the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, and IEEE SOFTWARE) and a Guest Co-Editor of more than ten special issues. His research interests include software architecture and architectural knowledge, software sustainability, technical debt, software product line engineering, and Industry 4.0.

MIROSLAW STARON received the Ph.D. degree from the Blekinge Institute of Technology, Sweden, in 2005. He is currently a Professor of software engineering with the University of Gothenburg. He specializes in software quality, measurements, safety-critical systems, and automotive software architectures. He has published over 200 articles and three books. His current research interest includes applications of artificial intelligence in intensive care. Since then, he has worked with a number of industrial partners large organizations, such as Ericsson, Volvo Cars, Volvo AB, SAAB Electronic Defense Systems, and RUAG Space.