An approach for classifying design artifacts

Sébastien Adam, Ghizlane El Boussaidi, Alain Abran
Department of Software and IT engineering
École de technologie supérieure
Montréal, Canada

***Abstract*—Software designers have to deal with a large number of distinct software design artifacts (SDAs), including requirements, patterns, and tactics. This paper proposes a technique that systematizes the classification of SDAs, and a classification scheme (CS) which organizes the SDAs into a matrix, in a manner derived from the Zachman Framework for enterprise architecture. An instantiation of this CS is a traceability matrix called a software-structure map (SSM) that records the SDAs and their relationships. The approach is illustrated through the analysis of the Template Method (TM) design pattern as an example of a SDA.**

Keywords-software knowledge management, software artifacts, multi-dimension analysis, decision support systems

I. INTRODUCTION

During the development of a software system, the software designers deal with numerous software design artifacts (SDAs) such as goals, concerns, requirements, and design patterns. A SDA can be characterized using some related SDAs and issues that may threaten the success of a project. For instance, a design pattern [1] is a SDA that is characterized by a rationale, a solution, some consequences, and trade-offs. Somehow, the SDAs constitute the assets that embody decisions and trade-offs applied during the project. Several approaches propose a process or a technique aiming at managing the software artifacts (e.g., [3, 7, 8]). These approaches usually focus on a subset of the artifacts involved in the development process and on a specific development perspective. However, there is a lack of works that support a methodical management of the SDAs and their relationships.

The SAM (Software Architecture Mapping) framework [9] was proposed to manage the accumulated knowledge related to software design in an integrated and systematic manner. SAM enables to: 1) relate the SDAs to their factors of influence; 2) offer support to use the relevant SDAs and to appropriately solve their related issues; and 3) keep track of the adopted arguments and resolved issues. The SAM framework relies on a knowledge base that is populated by creating a set of matrices called *software structure maps* (SSMs). A SSM is a matrix that organizes software design artifacts and their relations. It is built using a classification scheme that is derived from the Zachman framework [8].

This paper presents the proposed classification technique that systemizes the creation of the SSMs (see Figure 1). The technique uses the classification scheme (CS) of the SAM framework for classifying the SDAs according to their descriptions in the literature – see Figure 2 [1. 2, 3, 10]. The technique is illustrated through the analysis of the Template Method (TM) design pattern as an example of a SDA.

The contributions of this paper are: 1) reusable specifications of the SDAs and their relationships based on a uniform SSM format; 2) a systematic technique for extracting and structuring the SDAs using the SSMs; 3) a flexible technique to transform textural descriptions to networks of SDAs. This paper is organized as follows. Section II presents an overview of the proposed classification technique. Section III introduces a case study to illustrate the classification technique. Section IV presents the related works and section V presents conclusions and future works.

II. OVERVIEW OF THE CLASSIFICATION TECHNIQUE

Figure 1 presents the proposed classification technique which aims at creating a SSM by extracting the verbs and nouns for structuring the SDAs and relationships that constitute the description of a style, a design pattern, or a tactic. The resulting SSM is a matrix of traceability that records design knowledge (DK) about the problem and solution spaces of a software design. The SSMs should be managed as part of the DK. A SSM captures DK about direct or indirect relationships between SDAs; it supports analyzing as presented in [9] how the SDAs impact the capacity of the software design to satisfy targeted objectives.

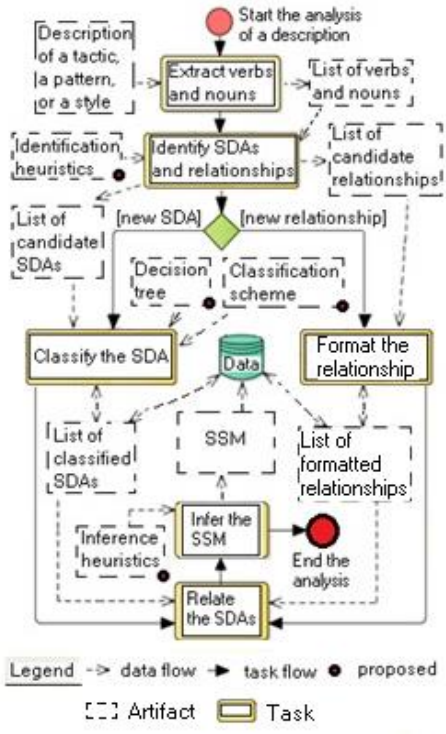

Figure 1. The classification technique of the SAM framework

DOI reference number: 10.18293/SEKE2015-226

	Rationale (Why)	Context (When)	Driver (What)	Structure (Which)	Behavior (How)	Allocation (Where)
Select objectives	needs, expectations, goals	organizational risks, politics, business model, situational factors	requirements, constraints, business rules	structures of domain objects	processes, activities, tasks, procedures	allocation of domain objects
Identify knowledge artifacts	architectural concerns	application domain standards, regulations, conventions	architectural properties	patterns and tactics	patterns of interactions	patterns of allocation
Define architectural artifacts	architectural design rationale	architectural risks, assumptions	scenarios	structural fragments	behavioral fragments	allocation fragments
Specify system artifacts	detailed design rationale	system's risks, assumptions	operation contracts	structures of modules	behaviors of components and connectors	allocations of elements
Describe architectural views	views descriptions	external entities, scopes, vocabularies, symbols	viewpoints	structural views	behavioral views	allocation views
Evaluate software structures	acceptance / assurance criteria	external and in-use metrics	internal metrics	structural evaluation records	behavioral evaluation records	physical evaluation records

Figure 2. The classification scheme of the SAM framework

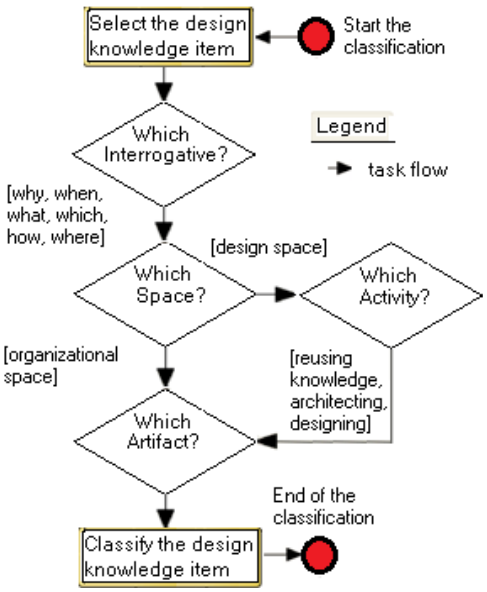

Figure 3. The decision tree of the SAM framework

A. *The tasks of the classification technique*

Six tasks constitute the proposed classification technique: extract verbs and nouns, identify SDAs and relationships, classify the SDA, normalize the relationship, relate the SDAs, and infer the SSM. The first and second tasks aim at identifying the candidate SDAs and relationships from the analysis of the description of a style, a design pattern, or a tactic using the identification heuristics. Then, the third and fourth tasks aim at classifying the SDAs using the decision tree and the classification scheme, and formatting the relationships using a list of formatted relationships. The fifth and sixth tasks aim at relating the SDAs and inferring the SSM by using the relationships and inference heuristics.

B. *The proposed identification heuristics*

For guiding the identification of the SDAs, we use in the SAM framework a set of identification heuristics. We consider that a SDA is: 1) less specific than implementation artifacts, i.e. implementation may be selected, within a particular technological context, to accomplish the intent of a SDA; 2) more enduring than implementation artifacts, i.e. a SDA should be described in a way that allows multiple implementations; 3) typically discovered or abstracted from practice and should have some correspondence with best practices such as styles, design patterns, and tactics; 4) coherent with more general or specific artifacts; 5) precise enough to be capable of analysis; and 6) related to one or more SDAs.

C. *The proposed classification scheme*

Figure 2 presents the proposed classification scheme (CS) of the SAM framework. The CS organizes the SDAs extracted from our analysis of the descriptions of styles, design patterns, and tactics. The CS captures the SDAs about the design problem and solution spaces, and about explicit or implicit relationships between the SDAs. The CS captures only the SDAs that influence the life cycle of a system.

The CS organizes the SDAs into a matrix that is based on the Zachman Framework for enterprise architecture [8]. The matrix classifies the SDAs according to their descriptions and relationships, as described in [1, 2, 3]. More specifically, the CS uses a matrix where the rows represent the activities of the software design process and the columns, the interrogatives (why, when, what, which, how, and where). The outcomes of the following activities occupy the row labels: select the objectives, identify the knowledge that has been successful in achieving similar objectives, and define, specify, describe, and evaluate the software architecture. The problem space is split into the interrogatives why, when, and what. The rationale (WHY issues) provides reasoning about the problem. The context (WHEN issues) describes the environment and hypotheses that influence the solution space. The drivers (WHAT issues) define the problem. The solution space is split into the interrogatives: which, how, and where. The domain objects and architectural elements have roles (WHICH issues) in realizing the solution. The execution of their behaviors (HOW issues) at the assigned locations (WHERE issues) shall satisfy the objectives for which a SSM is done. The SDAs in the top row of Figure 2 define the problems and solutions from an organizational perspective. The ones in the five lower rows do the same from a design perspective. Each lower-row contains artifacts for refining the interrogatives of the row that is above it, from the general objectives to the specific system artifacts.

D. *The proposed decision tree*

We propose to use the decision tree in Figure 3 for classifying the SDAs, and the following questions for supporting the classification task. The questions begin with the prefix "Does the SDA describes". Each question relates to one of the four main questions presented in the decision tree: which interrogative, space, activity, and artifact best render the meaning of the SDA in the context of a SSM?

- Which interrogative?
  - why: "… a reasoning for the SSM?"
  - when: "…a contextual information for the SSM?"
  - what: "… a target for a solution?"
  - which: "… the element of a solution?"
  - how: "… the behavior of an element?"
  - where: "… the allocation of an element?"
- Which space?
  - organizational: "… the organizational space? "
  - design: "… the design space? "
- Which activity?
  - reusing knowledge: "… an information that is part of the design knowledge base?"
  - architecting software: "… an information about a design fragment?"
  - designing software: "… an information about a design structure?"
- Which artifact?
  - use the SDAs' descriptions

E. The proposed SDAs descriptions

For classifying an artifact, we propose to use the SDAs described in Tables I to III. We extracted the proposed SDAs' descriptions from our review of the literature. Because of the lack of space, we describe only some SDAs that relate to the top four rows of the classification scheme.

TABLE I. THE DESCRIPTIONS OF SOME SDAS RELATED TO THE WHY INTERROGATIVE

Why: These SDAs provide reasoning for the SSM
Architectural / Design concern: an area of interest specified with respect to a goal in terms relevant for architecting / designing
Architectural rationale: a statement of reasons for a design fragment (e.g., isolate each layer from changes in other layers)

TABLE II. THE DESCRIPTIONS OF SOME SDAS RELATED TO THE WHAT INTERROGATIVE

What: These SDAs provide the targets for the solution space
Architectural property: a condition about a property of the elements or relations of a design fragment (e.g., performance)
Scenario: a description of how a software product should respond to a stimulus

TABLE III. THE DESCRIPTIONS OF SOME SDAS RELATED TO THE WHICH INTERROGATIVE

Which: These SDAs provide the elements of the solution space
Design pattern: a description of how the elements of a design fragment relate to each other in order to address a design concern
Structural fragment: a set of elements and relationships of a design fragment (e.g., instantiation of the template method)

F. The proposed relationships description format

We identified some relationships between the SDAs from the literature [1, 2, 3, 4, 5, 6, 7, 10] – see Table IV. The SAM framework proposes to format each relationship using a unique identifier, a description of the relation, and the SDAs between which the relationship applies, as example:

Identifier	Description	SDA-to-SDA
Generalize	A SDA generalize another SDA	Structure-to-Structure

TABLE IV. THE FORMATTED RELATIONSHIPS OF THE SAM FRAMEWORK

Relationship	Description
Mandatory	A SDA requires the presence of another SDA
Optional	A SDA optionally implies another SDA
Constraint	A SDA constraints another SDA
Encapsulate	A SDA encapsulates another SDA
Generalize	A SDA generalizes another SDA
Specialize	A SDA specializes another SDA
Realize	A SDA realizes another SDA

G. The proposed SSM's inference heuristics

Due to the lack of space, Table V presents only some of the inference heuristics we propose for inferring a SSM using the classified SDAs and the normalized relationships. These inference heuristics aim at controlling the level of cohesion between the SDAs of a SSM. Only one SDA drives the cohesion of the SSM. All SDAs within a SSM shall be cohesive with the driver SDA.

TABLE V. THE INFERENCE HEURISTICS FOR THE SDAS RELATED TO THE WHY INTERROGATIVE

SDAs	Inference heuristics
Architectural concern, Design concern	- Part of the design knowledge base - Describe concerns for the SSM's design space - Influence all SDAs of a SSM's design space - Relate to a goal in the SSM
Architectural rationale	- Set rationale for elements of a design fragment - Relate to an architectural concern in the SSM
Design rationale	- Set rationale for elements of a structure - Relate to a design concern in the SSM

III. CASE STUDY - APPLYING THE CLASSIFICATION TECHNIQUE

This section presents an overview of the case study selected for applying the classification technique of the SAM framework. We analyzed the descriptions of multiple architectural tactics in [3], design patterns in [1], and architectural style in [2] for creating their SSMs using the proposed classification technique.

A. Mapping for the Template Method design pattern

Table VI presents the SSMs of the Template Method (TM) design pattern described in [1]. The TM design pattern is used for providing reusability and extensibility of algorithms in object-oriented software. It aims to implement the skeleton of an algorithm in a base class, and calls primitive methods that subclasses override to provide concrete behavior. The base class interface declares the algorithm as a template method, which calls abstract primitive methods that represent the algorithm's variation points. The subclasses implement the primitives to specialize the algorithm. As a result, the algorithm's structure is written only once and indirectly specialized in subclasses, which reduces duplication of code and enforces class interface stability. Also, the template method allows the addition of instrumentation in the base class, and lightens users' duty since it is no longer required to call a primitive.

TABLE VI. THE SSM OF THE TEMPLATE METHOD DESIGN PATTERN

SDA	Description
Concern	Avoid code duplication
Concern	Control subclasses extension
Concern	Localize changes
Concern	Prevention of ripple effect
Rationale	Fix the steps of the algorithm and ordering
Rationale	Let subclasses define the steps of the algorithm
Rationale	Maintain the algorithm's structure
Rationale	Limit extension points
Rationale	Provide default behavior
Rationale	Control access to the operations
Situational f.	Multiple kinds of primitive operations
Convention	Naming convention
Symbol	UML notation
Property	Object-oriented paradigm
Property	Object-oriented programming language
Property	Reusability
Property	Extensibility
Operational.	Define an abstract base class
Operational.	Define a template method
Operational.	Define a concrete child class
Operational.	Define hook operations
Viewpoint	Class diagram
Viewpoint	Sequence diagram
Pattern	Template Method
Tactic	Abstract Common Services
Fragment	Class library
Structure	Abstract class definition
Structure	Concrete class definition
Behavior	The TM controls the order of execution
Behavior	The hook operations do nothing by default

IV. RELATED WORKS

To take full advantage of the accumulated design knowledge, the designers need frameworks and tools not only to manage this knowledge but also to relate it to the decisions taken and artifacts produced during the design activity. However, most of models, methods, and tools provide limited views into this knowledge base [2, 5, 6, 7, 10]. Many approaches were proposed to support the design process [3, 5, 6, 7], but few approaches support the designers in managing and keeping track of the accumulated knowledge during the design process. One of the most used approaches is the Attribute-Driven Design method (ADD) [3]. The focus of ADD is the process of architecting systems in order to satisfy a set of quality attributes and to manage tradeoffs between these attributes (quality dimension). Our approach can be used to analyze and keep track of the artifacts and knowledge produced by the ADD.

Many architectural styles and patterns have been described and cataloged in the literature [1, 2, 3], but few approaches support the designers in extracting the design knowledge from textual descriptions. We believe that the proposed classification technique can be used to systematically analyze textual descriptions provided in the literature, organize the design artifacts, and to explicitly relate the artifacts used during the design process.

Finally, our work is closely related to Ovaska *et al.*'s work [5]. They proposed an approach to fully integrate quality requirements into the software design process. Their approach allows the architect to manage and track the quality attributes from the requirements specification to the architecture design. This approach focuses on finding styles and patterns using some quality attributes. While this is very useful, an architect still needs to keep track of the rationale, objectives and other constraints that led to choose these quality attributes. Our framework can be useful to manage these relationships into a SSM that relates in a finer-grained manner the artifacts of the problem space to the ones of the solution space, from the organizational goals to the specific system artifacts. We believe a SSM is a valuable artifact for providing an integrated view of the knowledge.

V. CONCLUSION

In this paper, we described a classification technique to populate a design knowledge base by extracting the software design artifacts and their relationships from the description of a style, a design pattern, or a tactic. We applied the classification technique for classifying the SDAs according to their descriptions and relationships, as described in the literature [1, 2, 3]. This work produced evidences that the multi-dimensional analysis approach introduced in [9] is a valuable step towards handling artifacts as an integrated set of factors of influence. The proposed classification technique can be customized to better support particular development process and systems' needs. In particular, the SAM framework may be adapted to sustain different CS. In the near future, we plan to propose a tool support and guidelines to support the process of creating a SSM, eliciting related arguments, and analyzing these arguments. The ultimate goal of this work is to build a reference model of SDAs and arguments linked formally and exploited by algorithms.

REFERENCE

[1] Gamma, E., Helm, R., Johnson, R., Vlissides, J., "Design Patterns: Elements of Reusable Object-Oriented Software", A.-Wesley, (1995)

[2] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J., "Documenting Software Architectures – Views and Beyond", Addison Wesley, Boston (2003)

[3] Bass, L., Clements, P., Kazman, R., "Software Architecture in Practice", Addison Wesley, Boston (2003)

[4] Kim, S., Kim, D.K., Lu, L., Park, S., "Quality- driven Architecture Development Using Architectural Tactics", Journal of Systems and Software 82, pp. 1211-1231 (2009)

[5] Ovaska, E., Evesti, A., Henttonen, K., Palviainen, M., Aho, P., "Knowledge Based Quality-driven Architecture Design and Evaluation", Journal of Info. and Soft. Tech. 52, 577-601 (2010)

[6] Shahin, M., Liang, P., Khayyambashi, M.R., "Architectural Design Decision: Existing Models and Tools", In: WICSA/ECSA 2009, IEEE, Cambridge, pp. 293-296 (2009)

[7] Parizi, R.M., Ghani, A., "Architectural Knowledge Sharing (AKS) Approaches: a Survey Research", Journal of Theoretical and Applied Information Technology, 1224--1235 (2008)

[8] The Zachman Framework, http://zachman.com/about-the-zachman-framework (2008)

[9] Adam, S., El-Boussaidi, G., "A multi-dimensional approach for analyzing software artifacts", 25th SEKE, June 27-29, Boston (2013).

[10] Standard, I.: ISO/IEC 42010 Systems and Software Engineering - Recommended Practice for Architectural Description of Software-Intensive Systems. ISOIEC 42010, (2011)