Assessing the Suitability of Architectural Patterns for Use in Agile Software Development

Samira Seifi Jegarkandy, Raman Ramsin
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
e-mail: seifi@ce.sharif.edu, ramsin@sharif.edu

***Abstract*—The software industry is moving towards agile software development methods, as they accommodate rapidly changing requirements, and cope remarkably well with modern challenges of software development. On the other hand, it has long been recognized that software architecture has a major impact on the maintainability, scalability, and quality assurance of software systems, so much so that it is virtually impossible to produce high-quality software systems (which are inherently complex) without architectural design. Agile methodologies use lightweight architectural practices, and applying architectural patterns is a common practice in agile development. However, to this date, there has been no comprehensive study on the suitability of existing architectural patterns for agile development. We introduce a set of criteria for assessing the suitability of architectural patterns for use in agile approaches, and evaluate a set of prominent architectural patterns based on these criteria. Agile developers can use the results of this evaluation to assess the suitability of each pattern for application in their agile development projects.**

Keywords-agile software development; software architecture; software pattern; architectural pattern; criteria-based evaluation

I. INTRODUCTION

Agile software development methodologies have been gaining in popularity, among industry practitioners and researchers alike, since they emphasize rapid and flexible development [1]. On the other hand, software architecture, as a discipline, deals with modeling and managing a software system's structure, project blueprint, and communications among stakeholders, which are essential for achieving quality attributes such as usability and maintainability [2]. Architectural design has always been an important issue in agile approaches, even though these approaches have strived to keep their architectural design tasks as lean as possible [3]. It has been suggested that agile methods' support for architectural activities should be enhanced even further [4].

In software engineering, an architectural pattern is a structured description of a reusable coarse-grained architectural *solution* to a commonly occurring *problem* within a given *context* [5]. Architectural patterns mainly target the non-functional requirements of the product, and provide an overall structure for the target system in order to address these requirements [2]. Architectural patterns are widely used today in all development approaches, including agile development, for structuring software systems. Although several approaches have been proposed for applying architectural patterns in agile development (such as [6]-[9]), there is currently no comprehensive review of these patterns to assist developers in selecting agile-friendly patterns. This paper focuses on evaluating the suitability of architectural patterns for application in agile development. To this aim, we propose a criteria-based evaluation approach. The evaluation criteria used in our approach have been elicited from the Agile Manifesto and Principles [10] and the CEFAM evaluation framework [11]. We have used these criteria to evaluate existing architectural patterns; due to space limitations, however, only the patterns that are most prominent and relevant will be focused upon in this paper.

The evaluation results obtained in our research (reported herein) can be leveraged to identify a set of agile-friendly architectural patterns. Thus, our proposed set of criteria provides a valuable framework for assessing the suitability of architectural patterns for use in agile development, and also for warning against the use of architectural patterns that are not particularly suitable for agile development. Another potential benefit of our proposed criteria is the applicability of the evaluation results for improving architectural practices in agile approaches; this has indeed been our ultimate intention in this research: we intend to combine architectural patterns and agile methodologies in order to address architectural issues in agile development without any adverse effect on agility. Even if an agile method puts sufficient emphasis on software architecture, misusing the method can cause architectural problems; the individual criteria and their respective evaluation results can potentially alleviate this problem by helping to determine the appropriate time and situation for applying each pattern in an agile context, thus enabling the developer to address architectural design issues.

The rest of this paper is organized as follows: Section II reviews the architectural patterns used in agile development; Section III introduces the proposed evaluation criteria, based on which the evaluation results are presented in Section IV; and Section V presents the concluding remarks and discusses possible directions for furthering this research.

II. ARCHITECTURAL PATTERNS AND AGILE DEVELOPMENT

In this section, we will provide an overview of architectural patterns and their use in agile development.

 ISBN: 978-1-61208-465-7

A. Review of Architectural Patterns

Patterns have been defined for many different areas of software development. However, our focus here is strictly on patterns for software architecture, which are commonly used for shaping the high-level structure of a software system. The terms architectural *style* and architectural *pattern* are widely used for describing these reusable structures [2]. In this subsection, brief descriptions will be provided for major architectural patterns; it should be noted that in cases where there are several variants for a pattern, the variant that is more widely used has been included, even if it is an older (earlier) variant. In later sections, we will assess these patterns as to their applicability in agile development.

We have categorized the architectural patterns according to their application areas in order to better manage the complexity of the spectrum of patterns under review. Even though some of the patterns are usually categorized as design patterns, we have included them herein as they can also be used for solving architectural problems, e.g., the Decorator pattern [12] can be used for creating dynamically configured chains of subsystems, and can thereby offer a solution at the architectural level. Some patterns address several application areas; in such cases, one of these areas has been designated as the main application area. All such patterns have been categorized based on their main application areas**,** e.g., we have assigned the Architecture with Component-as-a-Service (CaaS) pattern [13] to the Mobile Software Development category, even though it belongs to the Distributed Systems Development category as well. The patterns have been briefly described in Tables I and II; these tables also show the main category to which each pattern belongs.

There are six pattern categories, as explained below:

- *Patterns for Mobile Software Development:* Various architectural patterns yielding different levels of qualities, especially as to performance and energy consumption, which are used for developing mobile systems and applications (shown in Table I).
- *Patterns for Cloud Systems Development:* Patterns for using cloud-platform services (shown in Table I).
- *Security Patterns:* Patterns that provide a high level of security (shown in Table II).
- *Patterns for Distributed Systems Development:* Patterns that define how distributed components collaborate with each other (shown in Table II).
- *Patterns for Agent-Oriented Systems Development:* These include patterns for developing systems in agent-oriented contexts (shown in Table II).
- *General-Context Patterns:* General patterns that do not belong to a specific application area or development context (shown in Table II).

TABLE I. ARCHITECTURAL PATTERNS FOR MOBILE SOFTWARE DEVELOPMENT AND CLOUD SYSTEMS DEVELOPMENT

Pattern Category	Name	Brief Description
Mobile Software Development	**Architectural Pattern for Mobile Groupware Platforms [14]**	Used for developing groupware platforms, providing separate functionality for three basic concerns: Collaboration, Communication, and Coordination. It divides systems into three separate layers: the collaboration layer consists of mobile groupware applications; the communication layer handles messages interchanged among mobile units; and the coordination layer provides the services required by applications to coordinate their operations on shared resources.
	Balanced MVC Architecture [15]	Used for supporting service-based mobile applications. This pattern is an extended Model View Controller (MVC) architecture where client and server systems embody the MVC pattern.
	External Customizer [16]	Focuses on adapting web content to mobile clients by creating a component that converts data from arbitrary mobile web information systems to a suitable format for potential clients.
	Internal Customizer [16]	Removes the need for external customization by providing the client with a response directly suitable for its manipulation. This pattern uses customization mechanisms in the design of mobile web information systems.
	Web Channel Broker [16]	Extends the Broker pattern with the capability to interact with the whole web while presenting only a subset of it.
	Application with External User Interface (UI) Elements [17]	Represents interactive applications with physically separated UI components. This pattern is an adaptation and extension of the MVC approach.
	Standalone Mobile Application [13]	Runs the entire expected functionality on a mobile device without referring to any external services or servers [13].
	Mobile Application with Full Offloading [13]	Offloads the whole application and its associated database to a server. The mobile device just transmits the required dataset to the server, and is not involved in any computations.
	Mobile Application with Partial Offloading [13]	Offloads parts of the application and the dataset to an external server.
	Architecture with Software-as-a-Service (SaaS) [13]	In this pattern, the client incorporates a simple web browser or dedicated client program, while all the functionality required is fulfilled by external services.
	Architecture with Component-as-a-Service (CaaS) [13]	Provides cloud services as finer-grained units of common and reusable functionality.
	CaaS-Based Architecture with Offloading [13]	Divides the required functionality into three parts: one part is fulfilled by the CaaS architecture, one part is offloaded to a dedicated server, and the remaining part is implemented in the client application.
	Extended MVC [18]	Extends the common MVC pattern with additional components and adaptations specifically intended for mobile application development.
Cloud Systems Development	**Cloud Policy Management Point [19]**	Controls security functions, including authentication, authorization, cryptography, and control of virtual machine images.
	Eventually-Consistent User Interface [20]	Ensures the eventual consistency of the user interface in cases where it is not possible (or desirable) to ensure the consistency of the cloud data stores that are used by the user interface.
	Loose Coupling [21]	Reduces dependencies among distributed applications and their components by using Brokers.
	Service Level Agreements Compliance Checking [22]	Provides a three-layered architecture for distinguishing probe concerns from monitoring data collection concerns according to the SLAs.

 ISBN: 978-1-61208-465-7

TABLE II. ARCHITECTURAL PATTERNS RELATED TO SECURITY, DISTRIBUTED SYSTEMS DEVELOPMENT, AGENT-ORIENTED SYSTEMS DEVELOPMENT, AND GENERAL CONTEXT

Pattern Category	Name	Brief Description
Security	**Secure MVC [23]**	Shows how several fundamental security patterns must be applied to MVC components in order to provide secure access/modification of the information residing in the Model component.
Distributed Systems Development	**Component-Based Architectural Style [24]**	Decomposes the application into reusable components (functional or logical), which are location-transparent and expose well-defined communication interfaces.
	Service-Oriented Architectural Style [24]	In this style, some of the components provide services to other components.
	Event-Based Integration Style [25]	Removes the need for identities on the connector interfaces in order to reduce the coupling among components. This style is also known as the Implicit Invocation or Event System style.
	Client/Server Architectural Style [24]	Provides distributed systems consisting of separate clients and servers and a connecting network. There are variants such as Client-Queue-Client and Peer-to-Peer (P2P).
	Distributed Publish/Subscribe [26]	Decouples the publishers of events from those interested in them.
	Enterprise Service BUS [26]	Integrates a variety of distributed services and related components. Within this architectural pattern, various components connect to a service bus via their service interfaces.
	Broker [5]	Achieves better decoupling of clients and servers through providing indirect, location-transparent access to distributed services by handling message calls to the appropriate objects.
	A-3 Style [27]	Defines a structure for coordinating distributed components. This style adopts the concept of "group" as an abstraction for organizing an application into semi-independent slices, providing a single and coherent view of these aggregates, and coordinating them.
AO Systems Development	**Layered Agent [28]**	Provides a structure for supporting the behavior of agents. This pattern decomposes agents into layers. All agents do not have the same layers.
	AO-Broker [28]	This pattern is a special kind of Broker specifically customized and extended for use in agent-oriented systems [28].
	Presentation-Abstraction-Control (PAC) [5]	Provides a structure for interactive systems. This pattern defines the system as a set of cooperating agents, each of which is responsible for a specific aspect of the system's functionality.
General Context	**Model-View-Controller (MVC) [5]**	Divides an interactive system into three interconnected, highly specialized, and loosely coupled components: Model, Views and Controllers. The model component encapsulates core data and functionality, view components display information to the user, and controllers handle user input.
	Model-View-Controller-Context (MVCC) [29]	An extension of the MVC pattern that also incorporates a context component, which is solely responsible for handling context-awareness concerns.
	Zone [30]	Provides flexibility in changing the logical and physical architecture of the processing unit, and the resources the processing units need to accomplish their tasks.
	Microkernel [5]	Provides mechanisms for extending the software system with additional and/or customer-specific functionality, thus making systems adaptable and extensible. In this pattern, the most important core services of the system are encapsulated in a microkernel component.
	Reflection [5]	Supports extension of applications and their adaptation to evolving technology and changing functional requirements. This pattern splits the system into two levels: a base level defines the application logic, and a meta level makes the software self-aware by providing information on its essential features.
	Façade [12]	Provides a unified interface to a complex subsystem. This pattern shields the components of a subsystem from direct access by their clients.
	Blackboard [5]	Useful for combining patchy knowledge to arrive at solutions, even if they are sub-optimal or not guaranteed. This pattern tackles problems that do not have any deterministic solution strategies.
	Component-Based Framework [31]	Used for developing component-based systems. This pattern provides a mixture of fixed and flexible elements that maximize the scalability and extensibility of systems.
	Configured Handler Method [32]	Performs event handling by using metadata, thus avoiding proliferation of empty methods or reduction in class cohesion. Also known as Metadata-based API and Metadata-based Invoker,
	Layers [5]	Divides the system into distinct layers where each layer is at a particular level of abstraction and handles a specific concern of the system.
	Pipes and Filters [5]	Used for processing date streams. This pattern divides the tasks of a system into several sequential processing steps (filters) that form a pipeline. Data is passed between adjacent filters through pipes.
	Adapter [12]	Used for translating calls between two different interfaces. This pattern converts the interface of a class into the interface that clients expect.
	Decorator [12]	Additional responsibilities can be dynamically attached to an object by using this pattern.
	Command [12]	Encapsulates a request as an object, thereby letting users parameterize clients with different requests, queue or log requests, and support undoable operations [12].
	Command Processor [5]	Complements the Command pattern [12] by addressing the management and scheduling of Command objects.
	View Handler [5]	Manages all the UI views that are provided by the system.

B. Using Architectural Patterns in Agile Development

Architectural concerns have always been addressed in agile development methodologies; the system "metaphor" used in XP is a prominent example [1]. However, there is a growing interest in further extending agile methods with architectural approaches [3][4]. One way to improve architectural design in agile software development is to use architectural patterns. These patterns should make architectural tasks more agile or add architectural tasks to

agile activities [7]. Research is ongoing on combining agile development with architectural patterns in order to improve agile processes; MASAM [8] and Mobile-D [9] are two such methods that use architectural patterns in agile approaches in order to capture architectural knowledge. References [33]-[35] provide examples of the use of architectural patterns in agile software development; these patterns have reportedly improved the quality of the systems produced, and have provided a holistic view of the system, without which agile projects could face serious impediments.

III. PROPOSED EVALUATION CRITERIA FOR ASSESSING SUITABILITY OF ARCHITECTURAL PATTERNS FOR AGILE DEVELOPMENT

As mentioned before, the suitability assessment proposed herein is based on suitability criteria. To this aim, we propose a special set of qualitative criteria based on the agile traits outlined in the Agile Manifesto and Principles [10], and the CEFAM Framework for evaluating agile methodologies [11]. This approach is based on the rather obvious observation that a pattern that violates these characteristics and damages agility cannot be used in agile development.

Our proposed set of evaluation criteria will be introduced throughout the rest of this section. The criteria, listed in Table III, are divided into two categories according to the type of evaluation results obtained through applying them:

- *Simple form:* The evaluation results for these criteria are of the "Yes/No" type, denoting the satisfaction or non-satisfaction of the criterion. "Need for formalism" is the only criterion in this category.
- *Scale form (multilevel):* The result of applying a Scale-form criterion is selected from among a number of predefined discrete levels. The levels are numbered in descending order of satisfaction; in other words, level 1 signifies the highest degree of satisfaction of the criterion. To provide a more precise evaluation, two of the criteria ("Reusability" and "Complexity Management) have been further divided into finer-grained sub-criteria.

In order to show the validity of the proposed criteria for their ultimate purpose (i.e., assessment of agile-friendliness), Table III also depicts the Agile Principles [10] that underlie (are addressed by) each and every criterion. The proposed criteria have also been assessed based on the validity metacriteria of [36]; this assessment shows that the proposed criteria are valid in that they are: 1) *General* enough to be used for evaluating all architectural patterns as to their suitability for application in agile development; 2) *Precise* enough to help discern and highlight the similarities and differences among architectural patterns as to their agile-friendliness; and 3) *Comprehensive* enough to cover all significant features of architectural patterns as pertaining to their suitability for application in agile development.

IV. SUITABILITY OF ARCHITECTURAL PATTERNS FOR USE IN AGILE DEVELOPMENT: EVALUATION RESULTS

In this section, we provide the results of assessing the reviewed architectural patterns based on the proposed criteria. The evaluation results, as assessed by the authors, are presented in Table IV.

Assessing the overall suitability of an architectural pattern for use in agile development can be a matter of opinion, as many patterns are strong in some of the criteria, but weak in others. Deciding the overall suitability of a pattern is therefore subjective, and depends on the priority of the criteria in the mind of the assessor. The last column of Table IV shows the overall suitability of each pattern as judged by the authors: "✓" denotes "Overall Suitable", and "×" signifies "Overall Unsuitable". In our opinion, the criteria and sub-criteria pertaining to "Reusability", "Decomposability", and "Complexity Management" are more important than others in assessing the overall agile-friendliness of the patterns. We have therefore given more weight to these criteria when giving our final verdicts.

To better understand the nature of the assessments made in this section, Table V illustrates how the proposed criteria have been used for assessing the MVC architectural pattern. Overall, based on the evaluation results, the MVC architectural pattern has been deemed as a suitable pattern for use in agile software development.

The selection of the appropriate pattern depends on the results of applying the whole set of criteria, not just one criterion; this means that the final evaluation result might be the same for all the assessors. Yet, even if the assessors do not concur on the final result, the evaluations are still valuable in that they help identify the strengths and weaknesses of the patterns under review; this knowledge can be used for comparing alternative patterns and improving the use of the patterns in agile methods. As an example, consider comparing MVC to Layers. Comparing the evaluation results shows that MVC fares better than Layers in most of the criteria; therefore, if these criteria are deemed crucial in a project, MVC would be preferable to Layers in that particular project situation.

V. CONCLUSION AND FUTURE WORK

The software industry is becoming increasingly keen on using agile methodologies to achieve rapid and flexible development. On the other hand, software architecture has evolved into a vast, essential discipline in software engineering; architectural design has become indispensable, especially when reusable, distributed, and maintainable software systems are required. Agile methodologies are in need of improvement as to their support for architectural design, and architectural patterns seem to be a promising means for addressing this need. This has been our ultimate goal in this research: to use architectural patterns for enhancing architectural design in agile methodologies.

As the first step towards this goal, we have evaluated existing architectural patterns as to their suitability for use in agile development. A set of qualitative criteria have been defined for evaluating existing methodologies. The results of criteria-based evaluation reveal that not every architectural pattern is suitable for use in an agile context; therefore, if an application requires the use of an architectural pattern that has been deemed as agile-unfriendly, using an agile approach for its development would be considered risky (at best).

 ISBN: 978-1-61208-465-7

TABLE III. PROPOSED EVALUATION CRITERIA FOR ASSESSING THE SUITABILITY OF ARCHITECTURAL PATTERNS FOR AGILE DEVELOPMENT

Criterion		Description	Possible values	Underlying agile principles*
Reusability	**Encapsulation and abstraction**	How does the pattern promote abstraction and encapsulation? Abstract modules are more reusable by nature, and encapsulation enhances reusability by setting up barriers among modules and reducing coupling [5].	1: At the level of components/classes; 2: Only at the level of subsystems/layers; 3: Addressed implicitly; 4: Not addressed, but not adversely affecting reusability; 5: Reusability reduced due to violation of encapsulation or lack of abstraction.	CR; CS; FD; FWS; GD; S
Reusability	**Separation of concerns and high cohesion**	How does the pattern support separation of concerns and promote high cohesion in modules? Separation of concerns and high cohesion go hand in hand, and enhance reusability by encouraging non-complex, specialized modules.	1: At the level of components/classes; 2: Only at the level of subsystems/layers; 3: Addressed implicitly; 4: Not addressed, but not adversely affecting reusability; 5: Reusability reduced due to clustering of functionality and execution of non-related work at the level of layers/components.	CP; CR; FD; GD; R; TP
Decomposability		How is the structure decomposed by the pattern so that each individual piece is small enough to be developed in an agile manner?	1: Explicitly addressed for the entire system; 2: Explicitly addressed for part of the system; 3: Addressed implicitly; 4: Not addressed, but not adversely affecting decomposability; 5: Decomposability is adversely affected by the pattern.	CP; CR; CS; CW; FD; FWS; R
Complexity Management	**Coupling and change propagation**	How does the pattern promote low coupling and prevent the propagation of change?	1: At the level of components/classes; 2: Only at the level of subsystems/layers; 3: Addressed implicitly; 4: Not addressed; 5: Coupling and change propagation is adversely affected by the pattern.	CP; CR; CS; FD; GD
Complexity Management	**Modularity**	How does the pattern provide a meaningful decomposition of the software system into subsystems and components? How does the pattern indicate how to physically package the entities that form the logical structure of the system?	1: At the level of components; 2: Only at the level of subsystems/layers; 3: Addressed implicitly; 4: Not addressed.	CR; R; S; TP
Hiding of implementation details		Does the pattern hide implementation details?	1: Only provides the overall system architecture; 2: Shows class structure; 3: Shows the classes and interfaces needed to create the elements; 4: Implicitly implies implementation details; 5: Explicitly states implementation details.	FTFC; SOT; TP
Removal of extra/duplicated work		Does the pattern pay special attention to removing extra/duplicated parts, thereby enhancing the simplicity of the design and avoiding unnecessary development work?	1: Addressed; 2: Addressed, but needs extra effort when applying the pattern; 3: Addressed implicitly; 4: Not addressed, but not adversely affecting simplicity; 5: Extra/duplicated work is introduced by the pattern itself.	CS; FD; FWS; GD; S
Application costs		Are the time, cost, and effort required for applying the pattern justifiable?	Application costs are: 1: Lower than the "before" state; 2: Reasonable; 3: Acceptable, because the pattern solves important problems; 4: High, because the problems solved are not important.	CS
Explicit definition		Does the pattern define the architectural solution (structure of the system and the relationships among its constituent elements) in a detailed and explicit fashion?	1: Explicit definitions of structure and relationships are provided; 2: Explicit definition of structure and implicit definition of relationships are provided; 3: Implicit definition of structure and explicit definition of relationships are provided; 4: Implicit definitions of structure and relationships are provided; 5: Not addressed.	CW; R
Need for modeling		Does applying the pattern require modeling (analysis/design)?	1: The modeling required can be supported by all agile methodologies (e.g., in a "metaphor" document); 2: The modeling required can be supported by some (but not all) agile methodologies; 3: The modeling required cannot be supported by agile methodologies, as it can have an adverse impact on agility.	CS; FTFC; FWS
Need for Formalism		Does applying the pattern require formalism? If yes, the pattern is not recommended for use in agile development.	"Y": Yes; "N": NO.	CR; FTFC; FWS

Legend:

* CP: Consistent Pace; CR: Changing Requirements; CS: Customer Satisfaction; CW: Collaborative Work; FD: Frequent Delivery; FTFC: Face-to-Face Conversation; FWS: Focus on Working Software; GD: Good Design; R: Reflection; S: Simplicity; SOT: Self-Organizing Teams; TP: Trust in People.

 ISBN: 978-1-61208-465-7

TABLE IV. EVALUATION RESULTS

Pattern		Encapsulation and Abstractness	Separation of Concerns, and High Cohesion	Decomposability	Coupling, and Change Propagation	Modularity	Hiding of implementation Details	Removal of Extra/Duplicated Work	Application Costs	Explicit Definition	Need for Modeling	Need for Formalism	Overall Suitability
Mobile Software	Architectural Pattern for Mobile Groupware Platforms [14]	2	2	3	2	2	2	2	2	1	3	N	×
	Balanced MVC Architecture [15]	1	1	2	2	1	1	1	2	1	1	N	✓
	External Customizer [16]	1	2	3	1	2	2	2	3	2	2	N	✓
	Internal Customizer [16]	5	4	4	4	3	1	4	3	2	2	N	×
	Web Channel Broker [16]	1	1	3	2	2	1	3	2	1	2	N	✓
	Application with External User Interface Elements[17]	2	1	2	2	1	1	2	2	1	2	N	✓
	Standalone Mobile Applications [13]	4	4	4	4	4	1	4	2	3	1	N	×
	Mobile Application with Full Offloading [13]	2	2	4	4	2	1	1	2	3	1	N	×
	Mobile Application with Partial Offloading [13]	2	2	2	2	2	1	2	2	3	1	N	✓
	SaaS [13]	1	1	2	2	2	1	1	1	3	1	N	✓
	CaaS [13]	2	2	2	2	2	1	1	1	3	1	N	✓
	CaaS-Based Architecture with Offloading [13]	2	2	2	3	2	1	1	3	3	3	N	×
	Extended MVC [18]	1	1	2	2	1	2	2	2	1	1	N	✓
Cloud Systems	Cloud Policy Management Point [19]	1	1	1	2	2	2	3	3	2	3	N	✓
	Eventually-Consistent User Interface [20]	3	3	4	4	4	4	3	2	4	1	N	×
	Loose Coupling [21]	1	1	2	1	2	1	3	3	1	1	N	✓
	SLA Compliance Checking [22]	2	2	3	2	3	1	2	2	3	2	N	×
Security	Secure MVC [23]	1	1	2	2	1	2	2	3	1	2	N	✓
Distributed Systems	Component-Based Architectural Style [24]	1	1	1	1	1	1	2	2	1	1	N	✓
	Service-Oriented Architectural Style [24]	1	1	1	1	2	1	1	1	1	1	N	✓
	Event-Based Integration [25]	1	2	1	1	1	1	2	3	1	2	N	✓
	Client/Server Architectural Style [24]	2	2	2	2	2	1	2	2	1	1	N	✓
	Distributed Publish/Subscribe [26]	2	2	2	2	2	1	2	2	1	1	N	✓
	Enterprise Service Bus [26]	2	1	1	1	2	1	2	2	1	1	N	✓
	Broker [5]	2	1	2	1	2	1	1	1	1	1	N	✓
	A-3 style [25]	2	2	2	2	1	2	3	2	1	2	N	×
AO Systems	Layered Agent [28]	1	1	1	2	1	1	2	2	1	1	N	✓
	AO-Broker [28]	1	1	2	1	1	1	1	1	1	1	N	✓
	PAC [5]	1	1	1	1	1	1	2	2	1	2	N	✓
General Context	MVC [5]	1	1	2	2	1	1	2	2	1	1	N	✓
	MVCC [29]	1	1	2	2	1	1	2	2	1	1	N	✓
	Zone [30]	1	2	2	2	2	1	2	2	1	3	N	✓
	Microkernel [5]	1	2	1	1	1	1	1	2	1	1	N	✓
	Reflection [5]	3	2	3	2	3	1	3	3	4	3	N	×
	Façade [12]	2	1	3	1	2	1	2	1	1	1	N	✓
	Blackboard [5]	3	2	5	3	2	1	3	4	2	2	N	×
	Component-Based Framework [31]	1	1	2	1	2	1	1	2	1	2	N	✓
	Configured Handler Method [32]	2	2	2	2	3	2	3	2	1	2	N	×
	Layers [5]	2	2	3	2	2	1	2	1	3	1	N	✓
	Pipes and Filters [5]	2	1	1	1	1	1	2	2	1	2	N	✓
	Adapter [12]	2	2	3	2	3	2	2	2	1	1	N	✓
	Decorator [12]	2	2	2	2	2	2	1	2	1	1	N	✓
	Command [12]	1	1	2	1	1	3	2	3	1	1	N	✓
	Command Processor [5]	1	1	2	2	1	3	2	3	1	1	N	✓
	View Handler [5]	1	1	2	2	1	2	2	2	1	2	N	✓

We have based our proposed evaluation approach on the Agile Manifesto and Agile Principles in order to ensure that agility requirements are properly and comprehensively addressed by the proposed criteria. The proposed criteria have also been validated through the application of assessment metacriteria.

Future research can focus on using the patterns that have been deemed as agile-friendly to improve specific agile software development methodologies. Architectural patterns can also be empirically evaluated by application to real-world development projects, the results of which can be used for enriching the results of criteria-based evaluation.

 ISBN: 978-1-61208-465-7

TABLE V. DETAILED EXPLANATIONS FOR THE EVALUATION RESULTS OF THE MVC PATTERN

Criterion	Description	Value
Encapsulation and abstraction	The Model, View and Controller components defined in MVC are encapsulated and abstract.	1
Separation of concerns, and high cohesion	MVC separates the business logic from the presentation logic, so it supports separation of concerns. Constituent components are highly specialized and cohesive.	1
Decomposability	Model, View, and Controller components can be developed in different releases; but MVC is silent as to further decomposition of these components, especially the Model component.	2
Coupling and change propagation	MVC decouples the Model from Views and Controllers, so changes in the UI do not propagate to the system's core functionality (implemented in the Model). However, Views and Controllers are tightly coupled.	2
Modularity	MVC provides modularity by decomposing the system into Model, View and Controller components.	1
Hiding of implementation details	MVC is silent as to implementation, and just defines the overall architecture of the system.	1
Removal of extra/duplicated work	The system structure defined by MVC implicitly removes duplications and extra parts, and thereby precludes extra/duplicated development work.	2
Application costs	The large number of updates and runtime components increases the cost; however, this is controllable, and the cost of applying the pattern can be considered as reasonable.	2
Explicit definition	MVC provides explicit and detailed definitions for the system's main components and the relationships among them.	1
Need for modeling	MVC can be modeled in a "metaphor".	1
Need for Formalism	No Formalism is required.	N

Another strand of research can focus on defining detailed quantitative criteria for assessing the suitability of architectural patterns for use in agile development. This would enable developers to obtain a more rigorous assessment of architectural patterns.

REFERENCES

[1] R. Ramsin and R. F. Paige, "Process-centered review of object oriented software development methodologies," ACM Computing Surveys, vol. 40, no. 1, February 2008, pp. 1–89, doi:10.1145/1322432.1322435.

[2] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, 2nd ed. Addison-Wesley, 2003.

[3] S. Ramakrishnan, "On integrating architecture design into engineering agile software systems," Proc. Informing Science and IT Education Conference, June 2010, pp. 9–25.

[4] H.P. Breivold, D. Sundmark, P. Wallin, and S. Larsson, "What does research say about agile and architecture?" Proc. 15th International Conference on Software Engineering Advances, August 2010, pp. 32–37, doi:10.1109/ICSEA.2010.12.

[5] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-Oriented Software Architecture: A System of Patterns, vol. 1. Wiley, 1996.

[6] N. Harrison and P. Avgeriou, "Pattern-based architecture reviews," IEEE Software, vol. 28, no. 6, November 2011, pp. 66–71, doi:10.1109/MS.2010.156 .

[7] C. G. Álvarez, Overcoming the Limitations of Agile Software Development and Software Architecture. Master's Thesis, Blekinge Institute of Technology, September 2013.

[8] Y. Jeong, J. Lee, and G. Shin, "Development process of mobile application SW based on agile methodology," Proc. 10th International Conference on Advanced Communication Technology, February 2008, pp. 362–366, doi:10.1109/ICACT.2008.4493779.

[9] P. Abrahamsson, et al., "Mobile-D: An agile approach for mobile application development," Proc. 19th Conference on Object-Oriented Programming Systems, Languages, and Applications, October 2004, pp. 174–175, doi:10.1145/1028664.1028736.

[10] K. Beck, et al., "Manifesto for agile software development," Available online at http://www.agilemanifesto.org [retrieved: January, 2016].

[11] M. Taromirad and R. Ramsin, "CEFAM: Comprehensive evaluation framework for agile methodologies," Proc. 32nd Software Engineering Workshop, October 2008, pp. 195–204, doi:10.1109/SEW.2008.19.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[13] J. Kim, "Architectural patterns for service-based mobile applications," Proc. International Conference on Service-Oriented Computing and Applications, December 2010, pp. 1–4, doi:10.1109/SOCA.2010.5707181.

[14] A. Neyem, S. F. Ochoa, J. A. Pino, and D. Franco, "An architectural pattern for mobile groupware platforms," Proc. On the Move to Meaningful Internet Systems Workshops, November 2009, pp. 401–410, doi:10.1007/978-3-642-05290-3_52.

[15] H. J. La and S. D. Kim, "Balanced MVC architecture for developing service-based mobile applications," Proc. 7th International Conference on E-Business Engineering, November 2010, pp. 292–299, doi:10.1109/ICEBE.2010.70.

[16] W. A. Risi and G. Rossi, "An architectural pattern catalogue for mobile web information systems," International Journal of Mobile Communications, vol. 2, no. 3, September 2004, pp. 235–247, doi:10.1504/IJMC.2004.005162.

[17] A. Lorenz, "Architectural patterns for applications with external user interface elements," Pervasive and Mobile Computing, vol. 9, no. 2, April 2013, pp. 269–280, doi:10.1016/j.pmcj.2012.09.006.

[18] F. E. Shahbudin and F. F. Chua, "Design patterns for developing high efficiency mobile application," Journal of Information Technology & Software Engineering, vol. 3, no. 3, 2013, pp. 1–9, doi:10.4172/2165-7866.1000122.

[19] E. B. Fernandez, R. Monge, and K. Hashizume, "Two patterns for cloud computing: Secure virtual machine image repository and cloud policy management point," Proc. 20th Conference on Pattern Languages of Programs, October 2013, pp. 1–11.

[20] C. Fehling, et al., "Capturing cloud computing knowledge and experience in patterns," Proc. 5th International Conference on Cloud Computing, June 2012, pp. 726–733, doi:10.1109/CLOUD.2012.124.

[21] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications. Springer, 2014.

[22] A. Chazalet, "Service level agreements compliance checking in the cloud computing: Architectural pattern, prototype, and validation," Proc. 5th International Conference on Software Engineering Advances, August 2010, pp. 184–189, doi:10.1109/ICSEA.2010.35.

[23] N. Delessy-Gassant and E. B. Fernandez, "The secure MVC pattern," Proc. 1st International Symposium on Software Architecture and Patterns, July 2012, pp. 1–6.

[24] J. D. Meier, et al., Microsoft Application Architecture Guide, 2nd ed. Microsoft Corporation, 2009. Available online at https://msdn.microsoft.com/en-us/ee658086 [retrieved: January, 2016].

 ISBN: 978-1-61208-465-7

[25] R. T. Fielding, Architectural Styles and the Design of Network-Based Software Architectures. PhD Thesis, University of California at Irvine, 2000.

[26] E. Fernandez and N. Yoshioka, “Two patterns for distributed systems: Enterprise service bus (ESB) and distributed publish/subscribe,” Proc. 18th Conference on Pattern Languages of Programs, October 2011, pp. 1–10, doi:10.1145/2578903.2579146.

[27] L. Baresi and S. Guinea, “A-3: An architectural style for coordinating distributed components,” Proc. 9th Conference on Software Architecture, June 2011, pp. 161–170, doi:10.1109/WICSA.2011.29.

[28] E. A. Kendall, P. V. M. Krishna, C. V. Pathak, and C. B. Suresh, “Patterns of intelligent and mobile agents,” Proc. 2nd International Conference on Autonomous Agents, May 1998, pp. 92–99, doi:10.1145/280765.280781.

[29] H. Shams and K. Zamanifar, “MVCC: An architectural pattern for developing context-aware frameworks,” Proc. 11th International Conference on Mobile Systems and Pervasive Computing, July 2014, pp. 344–351, doi:10.1016/j.procs.2014.07.035.

[30] K. J. Rothenhaus, J. B. Michael, and M. Shing, “Architectural patterns and auto-fusion process for automated multisensor fusion in SOA system-of-systems,” IEEE Systems Journal, vol. 3, no. 3, September 2009, pp. 304–316, doi:10.1109/JSYST.2009.2022572.

[31] D. Parsons, A. Rashid, A. Telea, and A. Speck, “An architectural pattern for designing component-based application frameworks,” Software: Practice and Experience, vol. 36, no. 2, February 2006, pp. 157–190, doi:10.1002/spe.694.

[32] E. Guerra, C. Fernandes, and F. F. Silveira, “Architectural patterns for metadata-based frameworks usage,” Proc. 17th Conference on Pattern Languages of Programs, October 2010, pp. 1–25, doi:10.1145/2493288.2493292.

[33] S. Prakash, A. Kumar, and R. B. Mishra, “MVC architecture driven design and agile implementation of a web-based software system,” International Journal of Software Engineering & Applications, vol. 4, no. 6, November 2013, pp. 13–26, doi:10.5121/ijsea.2013.46021.

[34] R. Moser, A. Sillitti, P. Abrahamsson, and G. Succi, “Does refactoring improve reusability?” Proc. International Conference on Software Reuse, June 2006, pp. 287–297, doi:10.1007/11763864_21.

[35] R. Mordinyi, “Towards an Architectural Framework for Agile Software Development,” Proc. 17th International Conference and Workshops on Engineering of Computer Based Systems, March 2010, pp. 276–280, doi:10.1109/ECBS.2010.38.

[36] G. M. Karam and R. S. Casselman, “A cataloging framework for software development methods,” IEEE Computer, vol. 26, no. 2, February 1993, pp. 34–44, doi:10.1109/2.191987.

 ISBN: 978-1-61208-465-7