


| | | |
|--|---|------------------------------------|
| ITC 2/49 Information Technology and Control Vol. 49 / No. 2 / 2020 pp. 249-259 DOI 10.5755/j01.itc.49.2.22893 | A Quantitative Approach to Analyze Modifiability in Software Architectural Design of Agile Application Systems | |
| | Received 2019/03/06 | Accepted after revision 2020/01/08 |
| |  http://dx.doi.org/10.5755/j01.itc.49.2.22893 | |

HOW TO CITE: Philip, M. M., Singhal, N., Ravi, R., Vijayakumar, B. A Quantitative Approach to Analyze Modifiability in Software Architectural Design of Agile Application Systems. *Information Technology and Control*, 49(1), 249-259. <https://doi.org/10.5755/j01.itc.49.2.22893>

A Quantitative Approach to Analyze Modifiability in Software Architectural Design of Agile Application Systems

Milu Mary Philip, Nishank Singhal, Raagashree Ravi, Vijayakumar B.

Department of Computer Science, Birla Institute of Technology and Science – Pilani,
Dubai Campus; P.O Box 345055, DIAC, Dubai, United Arab Emirates;
e-mails: p20120003@dubai.bits-pilani.ac.in; nishanksinghal20nov@gmail.com;
f20150277@dubai.bits-pilani.ac.in; vijay@dubai.bits-pilani.ac.in

Corresponding author: p20120003@dubai.bits-pilani.ac.in

The present-day software application systems require a high degree of agility during the development and operational phases due to the advancements in software technologies and also because of the need to support the variation points in software architecture. A single architectural style will not be adequate for the architectural design of such application systems. This paper uses a composite architectural style involving three different architectural styles, namely model view controller, pipes and filters and reflection architectural pattern. A metric to evaluate the extent to which the architectural design is modifiable is defined and formulated. The number of direct connections between the components and their modes of operation are the various factors that will determine the extent to which the architectural design is modifiable. The model has been tested successfully for a prototype document processing application system. The composite architectural design is quite generic and it can be used for any real-time application system where the three modes of operation – data stream mode, user interaction and dynamic invocation mode exist together.

KEYWORDS: Model View Controller, Modifiability, Pipes and Filters, Quality Attributes, Reflection.

1. Introduction

The present-day software application systems are increasingly driven by large-scale deployment of components and their interrelationships. The software architecture of an application system defines the overall composition and formation of an application system. It is described as a framework, representing the main components and their connections [8]. It is a baseline for the successful design and implementation of the application system. A robust and reliable software architecture ensures that the software application system will meet the essential functional and non-functional requirements. Modifiability is one of the essential non-functional attributes that an application system should possess. It is defined as the potential of a software application system to be improved or reorganized to fit to the needs and requirements of the customer. It molds the application system to mature to a better and complete product. The facility to add, delete or modify the components, or change the order of execution of the components are the variation points that the architectural design should support. This is considered as the main criteria to measure the modifiability attribute [1]. These variations to the requirements can be established at any of the stages of the software development lifecycle [16].

This paper uses a *composite architectural model* for handling data stream mode of operation, user interactivity and variation points at run time. The stream mode of operation is fulfilled using the pipes and filters architectural pattern; the interactivity is made possible using the model view controller pattern and lastly the reflection architectural pattern makes the application system adaptable to the different variations. The composite architectural model follows the *pattern within a pattern* concept of blending the primitive patterns. The P&F, MVC and reflection are the primitive patterns that have been used together. The *modifiability metric* has been defined and formulated, in order to evaluate the architectural design. This metric provides an estimate on the extent to which the architectural design is modifiable. The major contributions of the work in this paper are highlighted below:

- 1 A composite software architectural design is being used to provide architectural flexibility and extendibility to comply with the dynamic requirements of the user community.
- 2 A metric to evaluate the architectural design for the modifiability quality attribute is defined and formulated.
- 3 The extent to which the architectural design is modifiable is assessed using the modifiability metric.

This paper is organized as follows- Section 2 covers the existing literature on the different architectural styles and the architectural evaluation techniques and methods. It also highlights the contributions of the current work. Section 3 discusses the generic model for composite architectural design -involving data stream, interactive and dynamic modes of operation of the application system. Section 4 provides details on the computation of modifiability metric that can be used for the evaluation of the architectural design. Section 5 discusses the document processing system – the case study that is considered in this work. The algorithm and the representation of the composite architectural style for the case study in this work is also detailed in this section. The last section focuses on experimental set up, test scenarios involving various combinations of components and connectors and the evaluation of the architectural style for the modifiability attribute. The benefits of the composite architectural model are also highlighted.

2. Background

This section presents the existing literature on architectural patterns and the evaluation of quality attribute – modifiability.

An application system provides support for user interactivity using a text or graphical based interface [19]. The functionality of the application system should be separated from the interface. This makes it easy to incorporate the changes in the interface, without affecting the working of the system. The model view controller (MVC) architectural pattern where the entire system is structured into model, view and controller, renders a solution for such application systems. The model component includes the data and the different functionalities of the system. The view acts as the user interface, where the user enters his requirements and the controller manages the user input [16].

The pipes and filters (P&F) architectural pattern facilitates the processing of input data in streams [21]. Here, the system is arranged in a sequence of processing stages. These stages are implemented using separate filter components. The filter components are created independently by the developer community [19]. This will ultimately speed up the time and effort required to build up the application system. Each component acts as a filter, taking the input from the previous stage, processing it and emitting the output to the next stage in a pipeline mode. The workflow management system [18] shows how P&F pattern can be implemented to serve this purpose. A layered software architectural design was developed for the development of a flexible and smart organic rankine cycle (ORC) [20]. This design allows coupling through IoT into smart systems based on the functionalities provided by the reflective operator and the cognitive operator present in the architectural design of the application system.

The reflection architectural pattern enables dynamic changes to the structure and behavior of the application systems [5]. These changes occur at runtime, when the application system is in operation. In practical systems, it is not always possible to foresee the runtime variations, especially those variations relating to the choice of components and their order of execution. In such scenarios, the architectural design of the application system is realized using – meta level, base level and meta object protocol. The meta level consists of the description of the objects relating to an application system, whereas the base level mainly focuses on the application logic. The presence of large collection of components, independent programs and application specific libraries help in building application systems with less development efforts. However, they pose great challenge with respect to choosing the correct combination of components as well as their sequence of operations. It is very much essential to evaluate the architecture for assessing the extent to which the quality attribute modifiability is supported. Software Architectural evaluation speculates the quality of a software product from a higher level design description [7]. The evaluation of the software architecture provides credible mechanisms to calculate the various features of the quality attributes and to distinguish the likely risks in the software architectural design [15]. The software architecture evalu-

ation determines if the current design satisfies all the requirements right from the analysis phase.

In the present age of agile software development, the requirements of user applications vary considerably in any of the phases of the software development lifecycle. Hence, the software architecture for such application systems should provide support for modifiability. The present-day software firms exist in a world in which variability in requirements is the major concern [12]. This makes it essential for the companies to deploy software application systems that are adaptable to ever-changing demands of the user community. A substantial portion of software development lifecycle costs deals with the development of the application system to meet the changing requirements. Hence, modifiability acts as a strong quality requirement for software architectural designs [4].

Architectural Tradeoff Analysis Method (ATAM), is one such method that uses scenarios for architectural evaluation. The scenario based evaluation methods are widely used for assessing the architecture before the development of the application system. It evaluates the architecture and also gives information on how well the quality objectives collaborate with each other [10]. It determines the outcome of architectural decisions with respect to the quality requirements. ATAM considers the evaluation of multiple quality attributes [22]. Software Architecture Analysis Method (SAAM) is yet another generic evaluation technique that evaluates the software architecture with respect to any quality attribute. The stakeholders play a very important role in such approaches as the scenarios are being developed by the stakeholders. The SAAM technique is used to explain and examine the software architecture of an application system. The steps in SAAM are as follows – designate a functional partitioning, link the functional partitioning to the structural decomposition of the architecture, selection of quality attributes to evaluate, scenarios to test the quality attributes and finally to estimate the degree to which the architecture provides the support [9].

Architecture Level Modifiability Analysis (ALMA) deals with change scenarios to analyze the modifiability of an application system [3]. The stakeholders present various scenarios that can emerge at a later stage in the software development lifecycle of an application system. The architectural design should be made adaptable to these change scenarios. The effect

of these scenarios are then determined. The components that should be modified for each of these scenarios have to be examined using architectural views. The aspects that these viewpoints cover, are a necessity for the application systems to be modifiable [12]. This evaluation technique finds its application in business information systems.

The above mentioned literature gives an overview of the different architectural patterns, their uses and the architectural evaluation techniques. The work in [17] clearly describes an architectural design where interactivity and data stream mode of operations coexist. The variation points were not handled in such an architectural design. The Conference Management System [14] uses the blackboard and reflection architectural patterns in combination. The major contributions of the work described in this paper is being highlighted below:

- 1 The metrics for the evaluation of the composite software architectural design are developed.
- 2 The level of support for modifiability in the architecture is assessed using the modifiability metric. The earlier works in the area of software architectural evaluation does not directly provide a metric for the modifiability attribute, though the coupling and complexity has been detailed.
- 3 The various factors that will determine the extent to which the architectural design is modifiable has also been pointed out.

3. Architectural Design Using Composite Patterns

The application systems in the present-day world with large number of component libraries, filter programs are highly complicated, that they cannot be designed using a single architectural style. Thus, a composite architectural style is being used in this work, with the three primitive patterns following the *pattern within a pattern* concept of blending the architectural styles. This section describes a generic model of the composite architectural design as shown in Fig. 1. It comprises of the architectural patterns MVC, P&F and reflection. The individual components that are used for the composite architectural design are discussed below.

3.1. Interactive Mode

The interactive mode of operation is realized using the MVC pattern. The view provides an interface to allow the user to enter the input details pertaining to the number of processing modules, their names, order of processing, and the mode of operation – parallel or sequential. For each and every application system, the filters for processing the input data are selected by the user. These processing filters are used to process the input data. It also consists of a facility where the user can enter any new filter according to the requirement. The controller handles and manages the user input and selects the required methods and procedures that have to be invoked for the processing of the input data. The model component consists of the *coreData* that has the input data and the *coreFunction* which has the different methods and procedures for the processing of the input data. The sequence of steps that are required for this subsystem are shown below:

- 1 The user chooses the required processing filters and their sequence of operation from the list of options provided in the interface. The new filter modules that are required by the user can also be entered in the user interface.
- 2 The controller takes the input details from the interface and invokes the corresponding methods present in the *coreFunction* of the model component.
- 3 The data that have to be processed are present in the *coreData* of the model component, and this will be modified.

3.2. Processing Data in Pipeline

The functionalities provided by the application system are present in the *coreFunction* of the model component and they operate in pipeline mode. This is made possible using the pipes and filters (P&F) architectural pattern. Thus, the model component is designed using P&F architectural design. This is the first instance, where the *pattern within a pattern* concept of blending the architectural styles is being realized. The processing modules in the P&F pattern are filters and the link between the filters is established using the pipes. The input data in the model component are taken up by the pipe and provided to the first filter for their processing. The sequence of operation of the filters can vary according to the entry made by

the user in the interface. For each and every change in the sequence of operation of the filters, there will be a corresponding change in the architectural design, thereby creating the flexibility in the design.

3.3. Dynamicity

Dynamic software product lines provide various ways to deal with the changes that occur at run time [6]. These changes occur at runtime, when the application system is in operation. In practical systems, it is not always possible to foresee the runtime variations, especially those variations relating to the choice of components and their order of execution. The software architectural design of such application systems is developed using the reflection architectural pattern [5]. The reflection architectural pattern enables dynamic changes to the structure and behavior of the application systems. It divides the entire software system into the meta-level, which contains the various properties of the system, and the base level, that consists of the entire logic of the application [11]. Andersson et.al [2], showed how a systematic approach using the reflection architectural pattern can be used for filling the gap between architectural design and its implementation by reifying the architectural features as meta-objects that are manipulated at execution time. The two causally connected layers of the reflection architectural pattern have been detailed below:

- 1 Base Level – The different services that are supported by the application system will be displayed in the user interface (view component) and form the base level. The base level will not be affected by any of the user requirement changes. Each component is implemented using third party filter programs and hence, the execution logic of the filters cannot be modified at any point of time. Any new filter can be introduced in this level.
- 2 Meta Level – The base level can be modified at runtime with the help of the meta-objects present in this level. The processing components and their order of processing are the variation points in this application. Based on the user/client requirement, the different processing components, which are the instances of the component ‘*Filter*’ can be added or deleted and the connectivity between them (which is established using the pipes, that are the instances of ‘*Pipe*’) can also be modified and thus, they are visualized as the meta level.

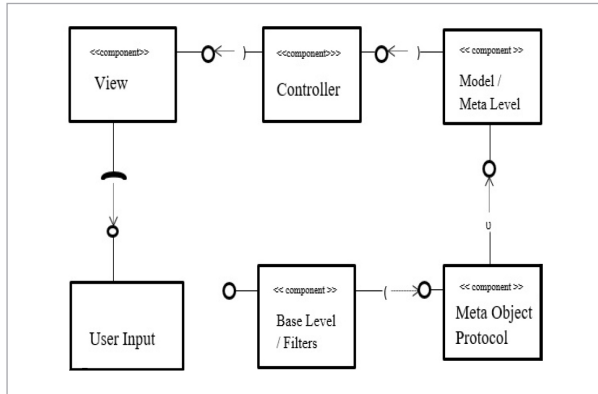
The meta-level makes the necessary changes in the different features present in the base level using an interface called as the meta object protocol (MOP). For any change in the filter components as well as in their order of processing (which is specified in the user interface), the MOP takes up the updated information and makes the required changes in the meta-objects accordingly. With each change in the meta-object, the required impact is reflected in the base level. Thus, the running architecture of the system is dynamically manipulated by the meta-level through the meta object protocol.

3.4. Combinational Architectural Model

The view presents an interface, where the user can enter the input details. The controller accepts the user entries and passes the input details to the model component. The model component encapsulates the core data of any application system and its associated core functions. The *model* corresponds to the *meta level* of the reflection architectural pattern. The meta level mainly consists of the information of the various fields, methods and function calls used in the application system. The information from the meta level is taken to the base level through the meta object protocol (MOP). Based on this input, the required base level filter objects are created. The *base level* consists of the individual *filters* and the *pipes* that are supported by the application system. This level provides the necessary operations pertaining to the application logic. Each operation is realized using filters. The filters can be arranged in any particular order as per the requirement of the application system. The link between the filter components is established using the pipes. The processing of the input data is carried out in stream mode of operation. The required interface and provided interface for the different components are realized using the UML2.0 notation.

The variation points (inclusion, deletion of processing modules and their order of processing) are the modifications that can happen to the application system. These changes will be known at run time, when the user enters the input in the user interface. The corresponding base objects are thus created only at run-time. The binding between base object and meta object occurs at run-time, and hence this follows the *run-time reflection*. At compilation time, the individual filter objects or base objects will not be created,

Figure 1
Generic Model of the Combinational Architecture



as the system does not know what entry the user will make during the execution. The individual filters or the base objects will not be able to make any changes to itself. This architectural design adapts itself to variation points and its variants. The design should be evaluated for the modifiability quality attribute, which is described in the subsequent section.

4. Computation of Modifiability Metric

The software architectural evaluation mainly deals with the assessment of the architecture, to determine whether it meets the quality requirements [22]. The software architectural design described in the previous section, mainly handles the variation points in the architecture, expressed in terms of components and their ordering. Hence, modifiability acts as an important quality attribute to realize the variation points, thereby providing flexibility in the development of application systems that evolve over a period of time.

The architectural factors like coupling, cohesion and complexity are used to measure the modifiability in the architectural design. Coupling can be described as the dependency between the different filters that make up the application system. It is one of the fundamental properties of the software architectural design. The interdependencies between the modules makes it difficult to understand and modify an application system. It also gives rise to ripple effects which causes errors and changes from a given module to one or more dependent modules. Cohesion is defined as the extent to which the modules in a filter collabo-

rate with each other to serve the functionality of the particular filter. In the present work, third party filter programs are being used to serve the application system. It is considered that the operation of these filter programs is not modified. Therefore, as cohesion is the property of each of the filter, this factor is not considered in this work, to measure the modifiability. Here, adding new filters, deleting existing ones and changing the sequence of operation of the filters are the variation points, to be considered to measure the modifiability attribute. The presence of large number of components with high coupling and low cohesion is an indicator of highly complex architectural design. Such a design is not desirable for evolving systems, since it provides limited support for modifiability.

The different filters are connected to each other, which increases the coupling. With every increase in the number of filters, and also with every link between the filters, the complexity of the application system also increases. Modifiability is calculated with respect to the coupling as well as complexity value. The following parameters are considered in the computation of modifiability metric:

N - the number of filters in the application system considered

p - the number of direct links between filters

q - the number of indirect links between filters

k - an integer constant indicating a set of filters having a relation

The coupling can be defined as the ratio of the number of direct links to the total number of links possible for the filters:

$$\text{Coupling} = p / (p + q). \quad (1)$$

Cyclomatic Complexity [13] is defined as

$$CC = E - N + 2. \quad (2)$$

where, E is the number of edges (link between filters), N is the number of nodes (filters)

$$p + q >= k. \quad (3)$$

Substituting Eq. (3) in Eq. (1) gives,

$$\text{Coupling} <= p / k. \quad (4)$$

The risk assessment threshold value [13] for a simple module without any risk has a complexity value in the

range of 1-10. To calculate the cyclomatic complexity, the number of edges ' E ' corresponds to the number of direct links between the filters. Hence,

$$CC=p-N+2. \quad (5)$$

Substituting Eq. (4) in Eq. (2) gives,

$$\begin{aligned} \text{Coupling} * k - N + 2 &<= 10 \\ \text{Coupling} &= (8+N) / k. \end{aligned} \quad (6)$$

An architectural design can support a higher level of modifiability for any evolving application system, when the coupling factor is low between the filters. As the coupling between the filters increases, the chances of making any changes to the filters become more complex. Thus, it can be deduced that,

$$\begin{aligned} \text{Modifiability} &\propto (1 / \text{Coupling}) \\ \text{Modifiability} &= M * (1/\text{Coupling}), \end{aligned} \quad (7)$$

where M is a proportionality constant and is assumed to be 1 in this study.

The coupling factor is expressed in a scale of 0 to 1 [7]. A coupling value of 0 means that there are no dependency between the filters in the architectural design, whereas a value of 1 refers to a maximum level of coupling between the filters. A coupling value of 0.5 can be considered as an acceptable level of coupling. When coupling = 0, the modifiability will be too high and hence this can be considered as the best architecture with respect to the modifiability attribute. When coupling = 0.5, and when the proportionality constant M is 1,

$$\text{Modifiability} = 1/0.5 = 2.$$

Thus, the modifiability metric can be tabulated as shown in Table 1.

Table 1
Modifiability Metric

| Coupling Factor | Modifiability Threshold | Architectural Evaluation |
|-----------------|-------------------------|--------------------------|
| [0] | Too high | Excellent |
| (0-0.5) | >2 | Good |
| [0.5] | 2 | Highly Acceptable |
| (0.5-1) | 1<modifiability<2 | Acceptable |
| [1] | 1 | Not Acceptable |

5. Case Study: Document Processing System

As a case study, the document processing application system has been considered. The variation points are introduced in this architecture in one or more of the following ways – adding new filters, deleting the existing ones and changing the processing sequence of filters. The document processing application system reads a collection of input documents in a continuous stream mode, applies appropriate filter programs in a given order and finally transforms them into an output collection. The filter programs considered here include *pdf2word*, *Search* and *word2pdf*. The software architectural design detailed in the subsequent section gives more insight into how the variation points in architectural design are handled by the application system.

5.1. Representation of Combinational Architectural Design for the Case Study

The architectural design for the case study described in the previous section is being detailed here. The application system starts its working with the interactivity. The interactivity is designed using the model view controller architectural pattern. The view component requires the following details from the user:

Number of filters – 3
 Names of filters – pdf2word, search, word2pdf
 Order of Processing– pdf2word, search, word2pdf
 Mode of Operation – Sequential

These input details are provided to the controller. The controller then selects the required methods *pdf2word()*, *search()* and *word2pdf()*. These methods as well as the input documents that are to be processed are present in the model or the meta level.

Here, the application system is examined about its properties, its fields and methods and then makes the changes depending on what has been found out. The properties of the application system are realized using its input data, as well as the functionalities provided by it. The *pdf2word()*, *search()* and *word2pdf()* are invoked in this component. Whenever the data are updated or modified, the controller takes up the updated information from the model and passes it back to the view, so that the user can view it as required.

The base level consists of the application logic- which implies that the processing of the input documents is taken place here. The function calls from the model/ meta level are taken up by the MOP. Finally, the input data is processed in the base level. The processing of the input data in the base level is carried out in stream mode of operation. The individual processing modules are called as filters. The connectivity between the filters is established using the pipes. The final processed output can then be given to the view component, using which the end user can view it. Each of the components in the architectural design and the corresponding required and provided interfaces have been detailed in Table 2. This corresponds to the generic model of combinational architectural style in Fig. 1.

Table 2

Representation of Combinational Architectural Design for Document Processing System

| Component Name | Required Interface | Provided Interface |
|---|--|--|
| View / User Interface | User Entered Details: - Number of Filters - 3 - Filter Names (<i>pdf2word</i> , <i>search</i> , <i>word2pdf</i>) - Order of Processing - Mode of Operation | Input Details |
| Controller | User Input Details from User Interface | Selects the required methods - (<i>pdf2word()</i> , <i>search()</i> , <i>word2pdf()</i>) |
| Model / Meta Level | Output from controller | Invoke the methods or function calls |
| Meta Object Protocol | Output from Meta Level | Invoke the filters <i>pdf2word</i> , <i>search</i> , <i>word2pdf</i> present in Base Level |
| Base Level - Filter (<i>pdf2word</i>) | Output from the meta object protocol | All the PDF documents are processed in stream to WORD using the <i>pdf2word</i> filter. |
| Base Level - Pipe (P1) | Processed WORD documents from <i>pdf2word</i> | Provide the WORD documents to the <i>Search</i> filter |
| Base Level -Filter (<i>Search</i>) | Input documents in WORD format | The required keyword is searched |
| Base Level - Pipe (P2) | Lines in which the keyword that is searched is present | Output passes on to <i>word2pdf</i> filter |
| Base Level - Filter (<i>word2pdf</i>) | Searched input documents in WORD form | Process the input documents |
| Base Level -Pipe (P3) | Converted PDF documents | The final processed output |

5.2. Algorithm: composite_Arch_Design

This section proposes a generic algorithm for the combinational architectural design described earlier.

1. Input: a collection of 'n' input elements
2. Output: collection of processed input elements.
3. Procedure:
 - 3.1 Read the required user definable details from the user interface.

Action:

- 3.1.1 The user enters the number of filters, the filter names as well as the required order of processing of the filters and their mode of operation.
- 3.1.2 If any new filter has to be added, the user enters the filter name in the space provided and go to step 3.2
- 3.1.3 The submit button is then clicked.
- 3.2 Insertion of a new filter

Action:

- 3.2.1 If there is any new filter that has to be added, check whether the class-file or the executable of the particular filter is provided by the user.
- 3.2.2 If it is not provided, throw error message as facility not available.
- 3.3 Invoke the appropriate processing filters at runtime to process the input elements.

Action:

- For every object obj,
- 3.3.1 Create the class object.
Class c = obj.getClass();
 - 3.3.2 Using the class object from 3.3.1,get the constructors of the specific class.
Constructor cs = c.getConstructors();
 - 3.4 Provide an external view of processed elements through the user interface.

The above algorithm has been successfully implemented for the document processing and analysis system.

6. Results and Discussion

The algorithm *composite_Arch_Design* has been successfully implemented for a document processing system. The experimental setup included JAVA under UBUNTU 14.04 OS. The main menu in the user interface has the provision to include the appropriate document processing filters. In the current work, *ConvertPdf2Word*, *ConvertWord2Pdf* and *Search* for a particular keyword in the input file are the filter programs that are being used. The user has the option to select the desired filters, their processing sequence and mode of operation. The input documents collection is comprised of files of type PDF, with a total size of 100 MB. The overall execution time for processing this document collection, with the 3 filters in sequential mode of operation was around 21.2 seconds. The time taken by each individual filter solely depends on the number of input documents and the size of the input files considered for a single run.

The modifiability threshold value is then calculated for the following scenario. This is a case of sequential mode of operation:

ConvertPdf2Word----Search----ConvertWord2Pdf

Here,

Number of Filters (N) = 3

Number of Direct Connections (p) = 2

Number of Indirect Links Possible (q) = 1

As per Eq. (1),

Coupling = $2/(2+1) = 2/3 = 0.66$.

According to Eq. (7),

Modifiability = $M * (1/ \text{Coupling})$.

M is a proportionality constant that is assumed to be 1.

Thus, Modifiability = $3 / 2 = 1.5$.

According to Table 1 in Section 4 described above, a value of 1.5 for modifiability threshold refers to an *acceptable* architectural design in terms of modifiability.

6.1. Modifiability and the Number of Direct Connections Between the Filters

If N is the number of filters taken into consideration, and p is the number of direct links between the filters, then,

Case 1: $p > (N-1)$ - there are too many interdependencies between the filters, which means coupling would be high. As there exist an inverse relationship between the coupling and modifiability, the modifiability will be very low. This implies that the architectural design is *not very good*, with respect to modifiability.

Case 2: $p = (N-1)$ - This will be the case of moderate modifiability, as the coupling factor will be in the acceptable range of 0.5 in such scenarios. Hence, modifiability will have a value of 2. This is in the *highly acceptable* range as per Table 1 shown in the previous section.

Case 3: $p < (N-1)$ - This refers to a scenario where all of the filters supported by the application systems are not interconnected. This means that the coupling factor would be very low. This is an example of an ideal architectural design for supporting the modifiability quality attribute.

6.2. Modifiability and the Different Modes of Operation of Filters

Both sequential as well as parallel modes of operation are supported by the architectural design. The sequential mode of operation is the case where one filter will complete its processing of the input data and then passes the processed data to the next filter. In the case of parallel connections between the filters, the execution of those filters connected in parallel, start at the same time.

For the scenario shown in Fig. 2, there are 6 filters that are to be considered for the processing of the application system. The typical sequential and parallel modes of operation are shown separately. Here, F1 – F6 are filter modules, M is taken as the number of parallel connections. When $M = 1$, it implies that the processing of the modules should be sequential and that all the filters are directly connected. For $M = 2$, it implies that there will be two filters that should be executed at the same time. In Fig. 2, when $M = 2$, the filters F1 and F3 should start their execution together. The number of links with no direct connections are the ones between (F1 and F5), (F1 and F6) and so on. It is observed that as the number of rows (value of M) increases, the coupling will be less and hence there will be an increase in modifiability. In this case, there will be no depen-

dependency between F1 and F3 and also between F1 and F4. Therefore, it is easy to make changes in this scenario.

Figure 2

Scenario for Different Modes of Operation for the case of 6 filters

| In the case of 6 filters, | | |
|---------------------------|----------------------------------|---|
| 1. M=1 | F1---F2---F3---F4---F5---F6 | Number of Direct Links (p) = 5 Number of Indirect Links (q) = 10 |
| 2. M=2 | F1---F2---F5---F6 F3 ---F4 | Number of Direct Links(p) = 4 Number of Indirect Links (q) = 11 |
| 3. M=3 | F1---F2---F5---F6 F3 F4 | Number of Direct Links(p) = 3 Number of Indirect Links (q) = 12 |
| 4. M=4 | F1---F2---F6 F3 F4 F5 | Number of Direct Links (p) = 2 Number of Indirect Links (q) = 13 |
| 5. M=5 | F1---F6 F2 F3 F4 F5 | Number of Direct Links (p) = 1 Number of Indirect Links (q) = 14 |
| 6. M=6 | F1 F2 F3 F4 F5 F6 | Number of Direct Links (p) = 0 |

Figure 3

Relationship between Modifiability Attribute, Coupling Factor and Number of Parallel Connections between filters

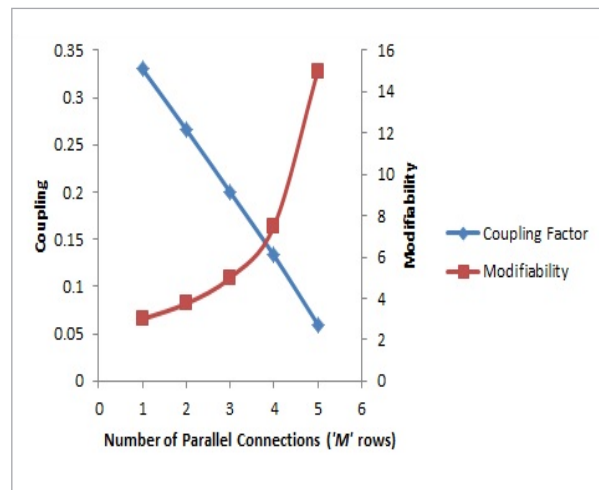


Fig. 3 shows the relationship between the number of parallel connections and the modifiability. The increase in the modifiability with the increase in the number of rows can be clearly observed. A similar relationship exists between modifiability and the parallel mode of operation between the filters, even if the number of filters N increases.

7. Conclusion

The software architectural design described in this paper, can be considered for any evolving application system. It uses a composite architectural style to design application systems in which three different modes of operation coexist - user interactivity, dynamism and stream mode of operations. Flexibility and adaptability are introduced to the application system, as the architectural design can handle the variation points leading to addition, deletion and changing the order of processing of the components.

It is highly essential to evaluate the architectural design before the development of the application system, in order to save the maintenance efforts and costs. The composite architectural design is evaluated for the modifiability quality attribute. A lower coupling factor indicates higher modifiability opportunities. Thus, the number of direct links between the filter, has an impact on the modifiability value. As the number of connections between the filter increases, it is observed that the chances of making the changes to the variation points becomes complex, which implies that the architecture does not support modifiability. The modifiability attribute is evaluated using the metric formulated in this work. The minimum threshold value for the architectural design to support modifiability fall in the range of 1-2. An analysis has also been performed on the effect of parallel connections between the filters on the modifiability attribute. It can be concluded that with every increase in the number of parallel connections, the coupling between the filters will be very low, which ultimately increases the modifiability scope.

The model used in this work, will significantly help the software architect in developing flexible software architectural design to support varying user requirements. Every architectural design can now be evaluated for the modifiability attribute using the modifiability metric computed in this work, and thus make the application system open to the ever-changing needs and demands of the user community.

References

1. Altoyan, N., Perry, D, E. Towards a Well Formed Software Architecture Analysis. ACM European Conference on Software Architecture: Companion Proceedings, 2017, 173-179. <https://doi.org/10.1145/3129790.3129813>
2. Andersson, J., De Lemos, R., Malek, S., Weyns, D. Reflecting on Self-Adaptive Software Systems. IEEE/ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, 2009, 38-47. https://doi.org/10.1007/978-3-642-02161-9_2
3. Bengtsson, P., Lassing, N., Bosch, J., & van Vliet, H. Architecture- Level Modifiability Analysis (ALMA). Journal of Systems and Software, 2004, 69(1-2), 129-147. [https://doi.org/10.1016/S0164-1212\(03\)00080-3](https://doi.org/10.1016/S0164-1212(03)00080-3)
4. Breivold, H., Crnkovic, I., Larsson, M. A Systematic Review of Software Architecture Evolution Research. Information and Software Technology, 54(1), 2012, 16-40. <https://doi.org/10.1016/j.infsof.2011.06.002>
5. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. Pattern Oriented Software Architecture, Wiley India Pvt Ltd, New Delhi, 2010.
6. Coplien, J, O. Reflections on Reflection. In 3rd Annual Conference on Systems, Programming and Applications: Software for Humanity, 2012, 7-10. <https://doi.org/10.1145/2384716.2384721>
7. Dragomir, A., Lichter, H., Dohmen, J., Chen, H. Run-time Monitoring based Evaluation and Communication Integrity Validation of Software Architectures. In Software Engineering Conference (APSEC) 21st Asia Pacific, 2014, 191-198. <https://doi.org/10.1109/APSEC.2014.38>
8. Galster, M., Eberlein, A., Moussavi, M. Systematic Selection of Software Architecture Styles. IET Software, 4(5), 2010, 349-360. <https://doi.org/10.1049/iet-sen.2009.0004>
9. Kazman, R., Bass, L., Webb, M., Abowd, G. SAAM: A Method for analyzing the Properties of Software Architectures. In Proceedings of 16th International Conference on Software Engineering, 1994, 81-90.
10. Kazman, R., Klein, M., Clements, P. ATAM Method for Architecture Evaluation. Carnegie-Mellon Univ, 2000. <https://doi.org/10.21236/ADA382629>
11. Krupitzer, C., Roth, F, M., VanSyckel, S., Schiele, G., Becker, C. A Survey on Engineering Approached for Self Adaptive Systems. Pervasive and Mobile Computing, 17, 2015, 184-206. <https://doi.org/10.1016/j.pmcj.2014.09.009>
12. Lassing, N., Rijsenbrij, D., Van Vliet, H. Viewpoints on Modifiability. International Journal of Software Engineering and Knowledge Engineering, 11(4), 2001, 453-478. <https://doi.org/10.1142/S0218194001000591>
13. Mc Cabe, T. A Complexity Measure. IEEE Transactions on Software Engineering, 1976, 308-320. <https://doi.org/10.1109/TSE.1976.233837>
14. Molesini, A, F. Alessandro, C. Garcia, C. von Flach Garcia, V. B. Thais. On the Quantitative Analysis of Architectural Stability in Aspectual Decompositions. In IEEE Conference on Software Architecture, 2008, 29-38. <https://doi.org/10.1109/WICSA.2008.26>
15. Patidar, A., Suman, U. A Survey on Software Architecture Evaluation Methods. In 2nd International IEEE Conference Computing for Sustainable Global Development (INDIACom), 2015, 967-972.
16. Philip, M. M., Vijayakumar, B. Software Architectural Design for Image Retrieval System Involving Data Stream and User Interactivity. Proceedings of the World Congress on Engineering and Computer Science, 2014.
17. Philip, M., Vijayakumar, B. Design and Implementation of Combinational Software Architecture for Batch Image Processing System. International Journal of Software Engineering and its Applications, 11(6), 2017, 79-88. <https://doi.org/10.14257/ijseia.2017.11.6.07>
18. Scheibler, T., Frank, L., Dieter, R. Executing Pipes and Filters with Workflows. 5th International IEEE Conference on Internet and Web Applications and Services (ICIW), 2010, 143-148. <https://doi.org/10.1109/ICIW.2010.29>
19. Tisato, F., Savigni, A., Cazzola, W. Architectural Reflection Realizing Software Architectures via Reflective Activities. Engineering Distributed Objects, 2001, 102-115. https://doi.org/10.1007/3-540-45254-0_10
20. Wolff, C., Knirr, M., Priebe, Klaus-Peter., Schulz, P., Strumberg, J. A Layered Software Architecture for a Flexible and Smart Organic Rankine Cycle (ORC) Turbine - Solutions and Case Study. Information Technology and Control, 2018, 47 (2), 349-362. <https://doi.org/10.5755/j01.itc.47.2.19681>
21. Wulf, C., Wiechmann, C, C., Hasselbring, W. Increasing the Throughput of Pipe and Filter Architectures by Integrating the Task Farm Parallelization Pattern. ACM SIGSOFT Symposium on Component Based Software Engineering, 2016, 13-22. <https://doi.org/10.1109/CBSE.2016.21>
22. Yang, C., Liang, P., Avgeriou, P. A Systematic Mapping Study on the Combination of Software Architecture and Agile Development. Journal of Systems and Software, 2016, 111, 157-184. <https://doi.org/10.1016/j.jss.2015.09.028>