Applicability of Quality Attribute-Driven Architecting In the Context of Agile Software Development: A Case Study

G. H. El-Khawaga
Information Systems Department,
Faculty of Computers and Information,
Mansoura University,
Mansoura, Egypt.
Email: Ghada.elkhawaga {at} ieee.org

Galal Hassan Galal-Edeen[1], A. M. Riad[2]
[1]Information Systems Department, Faculty of Computers and Information, Cairo University, Cairo, Egypt.
[2] Dean of Faculty of Computers and Information, Mansoura University, Mansoura, Egypt

Abstract— **Understanding and grasping the philosophy of architecting and considering the true purpose of it can help in driving a conclusion that architecting –if done carefully- can be an agility-enabler stage, which demands an agile mindset while analyzing and specifying its drivers; and enables acting in an agile way through implementing them, while keeping business values and easing frequent accommodation of changes. Through this paper, a framework for architecting in the context of agile software development is presented. This framework is built to overcome problems of current trends in agile architecting. Integrating architecture-centric practices into an agile process and managing effectively how and where these practices would be inserted into the process, and what actions and interactions will be between these practices and other development ones, enable the proposed framework to realize these aims, and a case study was held to help explore how this framework achieved these goals.**

Keywords- Software Architectures, Agile software development, Quality attributes, Architecting Framework

I. INTRODUCTION

Methodology practitioners believe that the amount of architecting done in the design phase of an agile process is not enough to produce a flexible but not fragile architecture. Agile architecting problems are believed to be the main reasons for accusing agile methods of resulting in architectures whose quality is suspected. If a deeper look is to be made into these problems, it will be found that they are interrelated. Agilists chase simplicity; this chase affected their view of architecting and in some cases limited it in their view of metaphors as a way for expressing and sketching architectures. Their chase of simplicity frightened them of doing Big Design UpFront (BDUF) and drove them into ignoring even foreseen changes –quality attributes –that would attack them on their way through the development lifecycle of a project. It is thought that targeting quality attributes would contribute to having a solution or at least eliminating side effects of the three other problems.

For addressing these issues, there is a need for spending some time planning architecture upfront. Architects should advocate a development culture that values design decisions based on careful analysis of requirements and give a due care to quality attribute requirements in advance, especially that they do not change as rapidly as functional requirements [1]. There is also a need for analyzing resulting architecture carefully to assess its adoption of needed quality attributes, and to deal with conflicts between several qualities at the earliest possible development level. By designing for including quality attributes right from the beginning, an architecture will be shaped around a long term goal. By including quality attributes, planning will be held for a basic infrastructure of a system that will change through development lifecycle. By designing architectures while regarding quality attributes from the beginning; agile methods would be more qualified for developing safety-critical systems where performance and reliability are a must. By spending time upfront in building a software architecture, and basing this architecture on quality attribute requirements, agile methods will be more qualified for building scalable software systems and manage complexity. Planning for quality attributes in advance prevents problems of missed quality attributes and implications of redesigning a system to incorporate these quality attributes.

Through this paper, a framework for Architecting Practices Integration into Agile Software Development (APIASD) -while regarding quality attributes upfront and working on preserving the flexibility and agility of an architecture- and a case study on its applicability, are presented. In section two, APIASD framework is illustrated, and it is applied on a case study which is presented in section three. After this, the experience and results after applying this framework are highlighted in section four, to conclude and sum up the experience as a whole in section five.

II. APIASD IN A GLANCE

APIASD aims to represent a structured yet flexible process that can result in an explicit architecture ensuring agility of the

product to be built. While forming APIASD, there were certain aims to achieve, and these aims were inspired by criticism of architecting in the context of agile software development previously presented [2]. The aims of APIASD can be summed up as follows:

- **Providing guidance and practical steps** that can help bridge the gap between requirements and software architecture from both sides. Architectural drivers –especially quality attributes- should be clearly identified in definitive forms so as not to leave the decision of accommodating them for afterthought or chances. Mappings are offered between business goals and user requirements –in all of their forms- and the resulting architectural artefacts.
- **Having** a **balance between architectural concerns and context**. While adhering to context more than concerns, more time and effort can be wasted, because a project's quality attribute goals need to be analyzed may be less than those defined through the organization's process. On the other hand, if quality attributes' specification is extracted only as concerns more than being driven by context, gains of experience reusability will be missed, and much time may be spent in analysing quality attributes while valuable quality goals can be obtained from considering contextual information available.
- **Having the rationale** of architectural decisions is necessary not only to trace decisions back to their reasons, but also to leverage the learning curve of a team of a project's team members.
- **Offering change impact information**. The effect of changing a component, a connector, or a relationship between them, or inserting a new component in response to either a functional or quality requirement should be highlighted ahead; so as not to have a design that apparently can accommodate all changes as they come up, while in fact it suffers severe mess in its architecture which degrades gradually.
- **Having an explicit architecture** –even if an initial incomplete version of it- is necessary for a process adopting an incremental and iterative development trend, like an agile software development process. In this context, an architecture provides insights into what the next step or the next chunk to develop will be.
- **Guiding software architecture's decomposition into increments** –while keeping business value as the main motive of the decomposition- is also a basic aim of this framework.

To begin illustrating how APIASD was supposed to reach these aims, figure 1 presents APIASD components. As shown in figure 1, APIASD consists of five basic architecting phases over three levels of agile development. The architecting phases are the result of integrating practices from several architecting methods, specifically they are: Global Analysis (GA) [3], Qualty Attribute Workshop (QAW) [4], Attribute-Driven development (ADD) [5], and Architecture Tradeoff-Analysis Method (ATAM) [6]. In a glance, an illustration of each phase is presented.

A. *Phase 1: Value Directions' Analysis*

Studying and exploring values that may have a global effect on the entire software system is inevitable to formulate architectural drivers and enable change impact analysis by searching for conflicting directions that provide more potential for changes through the software's lifecycle. Steps of this phase take place after an initial understanding of values and concepts is available, so as to act as a directive for architectural drivers' identification. In the context of agile development, these value directions are open to modifications and changes whenever a clearer understanding of software's goals and requirements is reached. The first step of this phase takes Vision, elevator statement, and product highlights as its inputs. Through the first step, development team and stakeholders work on considering the purpose of the software and critical business needs and identifying cross-cutting business values that should be targeted. The output list of value directions is analyzed through the second step of the same phase.

This second step of this phase is held to establish relations, and preference criteria of value directions identified. These criteria work to enable change impact analysis, and locating change influenced directions whenever a change can affect any of value directions identified. The development team and stakeholders – with the guidance of product owner and onsite customer- gather to identify how a value director is likely to change during or after development and to identify to what extent this value director is negotiable or critical from business stakeholders' viewpoint. Also they identify the impact of a direction on others, so as to enable identifying conflicts between them and making a decision whenever a change hits by the software.

B. *Phase 2: Quality Attributes' Analysis*

The goal of this phase is to transform quality attributes captured through value directions into tangible form to guide architecture creation. Through the first step of this phase, development team members brainstorm to obtain an explicit form of quality attributes and to provide an operational definition of quality attributes to be used to get a clue of how to achieve these attributes and how to measure their level of satisfaction. Scenarios generated are identified through defining scenario parts, which are: stimulus, environment, response, response measure, source of stimulus, and the artefact affected by a stimulus. Scenarios generated fit into quality attribute-related feature cards. A team leader makes sure that each quality attribute has at least one scenario concretizing and representing it. An architect is responsible for separating system-related concerns from software-concerns whenever found in scenarios generated.

Through the second step of this phase, quality attribute-related feature cards are consolidated and prioritized. Quality attribute-related feature cards consolidation helps condensing development team efforts into necessary quality attribute-related feature cards only. Quality attribute-related feature cards prioritization helps in selecting the right portion of the software to begin architecture development, based on business value and impact on the architecture to be created.

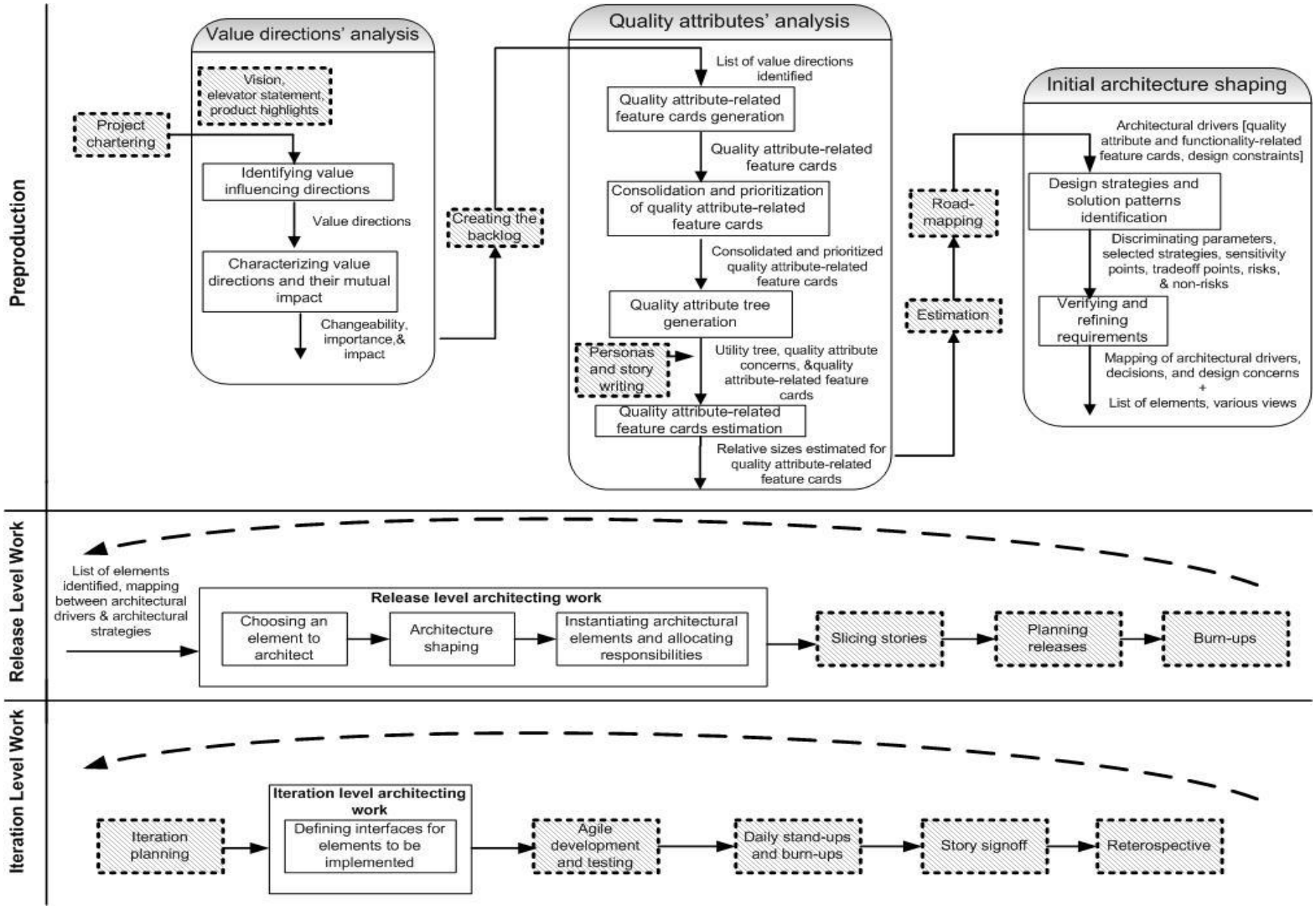

Figure 1: A framework for Architecture-centric Practices Integration into Agile Software Development (APIASD)

The development team votes on pairs of quality attribute-related feature cards to be merged, in a session managed by the team leader. Then, the architect works on identifying priority of quality attribute-related feature cards based on their relative impact on the architecture to be developed, these prioritizes assigned are based on back tracing to associated value directors impact. Prioritization scale is (H/M/L), where (H) denotes High, (M) denotes Medium, and (L) denotes Low.

The goals of the third step are to identify quality attribute concerns and to group related quality attribute-related feature cards by quality attributes and quality attribute concerns. Identifying quality attribute concerns paves the way for identifying solution strategies addressing each quality concern through upcoming stages. Grouping related quality attribute-related feature cards by quality attributes and quality attribute concerns helps in identifying conflicts between a scenario and other related to the same quality attribute or those related to other quality attributes. The architect works on building an initial version of the utility tree with quality attribute concerns defined through it. This version is discussed by the development team to get to a common understanding about quality concerns' naming and tenet. After identifying functionality-related feature cards, the team should identify dependencies between these feature cards and quality attribute-related ones.

Through the forth step of this phase, quality attribute-related feature cards are estimated. Quality attribute-related feature cards' estimation is a practice held the same way as for functionality-related feature cards. Estimating a feature size is a group effort that represents the collective mind of a team's members and increases their sense of ownership for the project they are working on. Through this step, development team agrees on scale for weighting and estimating feature cards (Fibonacci scale for example). By the end of this step and after estimating the functionality-related feature cards, the product backlog is divided into two similar sections; one for quality attribute-related feature cards and the other section for functionality-related feature cards.

C. Phase 3: Architecture Shaping:

An initial architecture is obtained through the first step of this phase. This initial version is driven by highly ranked architectural drivers to ensure being driven by business value. The resulting initial architecture evolves and grows incrementally to incorporate further features and quality attribute concerns as changes hit by the product to be developed. The resulting architecture is built upon the highest priority quality attribute concerns and features, and the remaining ones are inserted into the architecture through conducting further iterations of the same phase presented in this subsection. The architect works on identifying discriminating parameters of each design concern inspired from its associated quality attribute-related feature cards and scenarios. Discriminating parameters serve as comparison criteria for the development team to make decisions about which architectural strategies to choose. The development team also associates chosen architectural strategies with their implications concerning sensitivity points, tradeoff points, risks, and non-risks to provide insights into change impacts in case a change hits by the software.

Then the team has a second step to ensure that all architectural drivers were satisfied through mapping them to architectural strategies and decisions proposed to address them. The development team works together on matching each architectural driver with its related design concerns and architectural strategies selected to satisfy them. The development team members also collaborate to decide how patterns relate to each other and give insights into new element types that emerge as a result of combining patterns. Through this step the initial form of an architecture is visualized through constructing necessary views decided according to which quality attributes are highly required and according to user perspectives. The development team also identifies integration types between functional and quality attribute requirements, whether a quality attribute overlaps, overrides, or wraps a functional requirement.

D. Phase 4: Architecture-Related Release and Iteration Work:

Incremental and iterative nature of this framework is emphasized and highlighted through steps of this phase. The initial architecture generated through previous phases provides a roadmap of basic elements of a product to be, and all what the development team need to do –through the first step of this phase- is to choose an element of the system to work on in the coming release. Choosing which element to be architecturally explored and technically implemented through a release is done according to business values addressed by this element, risks and difficulty associated with developing it, current knowledge available about dependencies of other elements on this one, or through organizational criteria like skills improvement plans. Then the team members pick related architectural drivers associated with the chosen element, especially highly-ranked architectural drivers that apply to it.

Through the second step of this phase, development team members work together on defining basic responsibilities implied by employing architectural strategies chosen. They also work on defining basic functionalities implied by architectural drivers allocated to the current release. Responsibilities allocation should fulfill certain criteria such as functional coherence, locality of responsibility, grouping similar patterns of behavior, and grouping similar patterns of abstraction. The team leader makes sure that all dependencies between release allocated elements are transformed into responsibilities allocated to certain elements to fulfill. After this, team members distribute functionalities among all elements assigned to be developed through the current release. This step is held to give an initial idea about different elements' responsibilities which can be used as a basis for release planning and iteration work allocation.

The third step of this phase complements the development team negotiates an element's interfaces from the perspective of the different views available. An interface describes the PROVIDES and REQUIRES assumptions a software element makes about others. A team's experience of the selected architectural strategy can accelerate defining interfaces between software elements and facilitate this step. The outputs

of this step may include interfaces between elements to be implemented through the current iteration; interfaces between elements to be implemented through the current iteration and other elements to be implemented through other iterations; external interfaces –if any- between elements from other systems and elements to be implemented through the current iteration; and dependencies and expectations between elements to be implemented.

III. CASE STUDY: AUTOMATIC QUOTATION SYSTEM

In this section, we are going to present how this case study was suitable for the application of APIASD, and how it was applied on it. In response to the company's HR request, the company's name and other identifying information won't be mentioned. The company's name will be denoted by **XYZ**. XYZ is a leading group of companies specialized in providing translation, localization, and content publishing services. XYZ has locations across USA, UK, Germany, and UAE. XYZ facilitates its clients' work through providing an online tracking system; called *Transparent*; which enables the clients to track and monitor their projects while ensuring privacy and efficiency. XYZ teams suffered before from problems related to late consideration of quality attributes into a software system, and hence the amount of implied rework which imposed –in some cases- redesigning the whole product under development. This encouraged the CEO and the software director to advocate applying APIASD on one of XYZ's projects. The project selected as a case study is a website for automatic quotation of translation services requested by a client. It is about a web-based system which enables a client to upload files to be translated; choose currency to pay with; choose a translation package from multiple packages with different facilities, like providing punctuated translations; customize delivery time; and track his/her requested translations progress. This product relies on *Transparent* in providing service packages' specifications, services' prices, and acceptance or rejection of translation tasks. The website allows clients to upload files in several formats and it is the software's responsibility to transform received files into RTF files.

Three members form the development team of this product; one senior developer, one tester, and a project manager who acts also as the architect in this project. CEO of XYZ recommended this project to be the company's pilot upon which the agile approach to software development and APIASD were to be employed for the first time in this company. *Automatic Quotation* was chosen to apply APIASD on it, because it has no external customers, it is a small project compared to others under development, and it is a project that hits all major software development phases.

A. Fitness Check

First of all, it was important to ensure that agile development is suitable for developing this project, as APIASD is targeting projects where an agile software development process is to be employed. Referring to Bohem & Turner's five critical dimensions [7] to decide the development approach suitable for a given project; the following notes were observed:

- **Size**: the product was expected to be developed within five months with a team of three persons. A small project is the ideal case for employing the agile approach.

- C**riticality**: it is a web-based project which provides quotation services of translation services. The customer of this project was the marketing department of the company. So, damage from undetected defects won't be irreplaceable money. Therefore, this product is not expected to be safety-critical, nor it is expected to cause loss of critical money. In this case, the agile approach won't be a risky choice if used in developing such a product.

- **Dynamism**: the basic challenge this team faces was frequent changes in customer requirements. These changes were not resulting from changes in the way this business goes. Instead, changes were mainly caused by having customers who aren't sure about what their needs were. Changes were also caused by having long delivery cycles through which the team can obtain customer's feedback. The project manager's basic complaint was about how to manage changes. Therefore, change responsiveness was a need that should be offered by the approach employed to develop this product.

- **Personnel**: there was only one senior developer available for developing this project, one tester, and the project manager who was from time to time aiding in programming tasks. The tester was willing to give hands on development, but not as a full-time participant. When applying skill levels introduced previously in chapter two, the result was:

  1- One developer of level 2 (the project manager with 14 years experience)
  2- One developer of level 1A (the senior developer with 5 years experience)
  3- One developer of level 1B

  Agile development requires, more than experience, a competent practitioner who is willing to learn and keep react to new situations innovatively. This was a clear characteristic of the developer and the tester who were so enthusiastic to migrate to using the agile approach to find a better way taking over their tasks. It was also noticeable that the project manager has the ability to tailor a process to fit a new situation.

- **Culture**: from interviews held with the software director and software department manager, and through questionnaires spread among team members; XYZ follows a plan-driven approach to software development, but there is no standard or documented process followed. Teams choose how they will tackle a specific project concerning design, programming, and testing. But, still a team is constrained by the project manager's decisions and views. The project manager declared he follows a mutual decision making style. Team members welcome having degrees of freedom in making decisions about learning new tools, technologies, following new trends in development, and deciding the order at which development of functionalities will proceed. Still, XYZ is considered to be an organization with traditional management style and

hierarchies, but people can feel comfortable and empowered by having more decentralized decision making process.

Based on the previous illustration, it was concluded that neither a pure agile approach nor a plan-driven one is advisable for developing such a project. Instead, a hybrid approach will be the most suitable for this situation. After conducting several discussions with team members, it seemed they were going to totally change their preproduction, planning practices, and how the product will be incrementally divided. What really matters is having APIASD being applied incrementally and iteratively. So, applying APIASD here was acceptable by the team as it was aligned with their beliefs to spend time till having an architecture for a product developed.

As APIASD is argued to be adhering to the values of agile development, people aren't considered to be resources. So, it was a must to ensure that the development team of *Automatic Quotation* has a background aligned with basic tenets and ideas introduced by APIASD. The background of the development team was characterized by three aspects through a number of questionnaires spread among them. These aspects are:

- **Beliefs** about what architecting is; these are beliefs gained through learning, communication, research, or any source of information a team member could have known about architecting from.

- **Experience-based lessons**; this aspect aims at getting an idea about changes to how a team member architects or participates to architecting a software product based on previous projects s/he has worked on.

- **What is done**, this aspect is concerned with getting to know whether the currently adopted architecting practices adopted by this team are aligned or may be aligned with practices introduced by APIASD.

To analyze the questionnaires' answers, Smith & Sidky's approach [8] to analyze a company's assessment of readiness - to adopt the agile approach- is used. The final results are presented in the following table.

TABLE 1: ALIGNMENT OF *AUTOMATIC QUOTATION* TEAM MEMBERS TO NEEDED BACKGROUND ASPECTS

Aspect	Alignment
Beliefs	Largely aligned (63.6 %)
Experience-based lessons	Largely aligned (58.25 %)
What is done	Largely aligned (57.2 %)

From the previous table, the development team's beliefs were largely aligned with the beliefs that were aimed to be induced by adopting APIASD. Actually, the *beliefs* aspect was the most important aspect, because it was to drive changes into the development team's adopted practices –*what is done*- to be more aligned with APIASD, and subsequently can affect *lessons learnt based on experience*. Also, it was encouraging to have results of questions related to *experience-based lessons* and *what is done* to be largely aligned with the ideas which were about to be brought in to the development environment by APIASD. To conclude, applying APIASD in such environment, by this development team, was suitable and expected to serve its aims.

IV. EXPERIENCE AND RESULTS

After applying APIASD in *Automatic Quotation* development, the details and influence of APIASD on the development of *Automatic Quotation* project need to be explored and analyzed. To capture these details and influences, several meetings were held with the team members; during which further guidance was provided to them, and their comments and suggestions were captured.

Because a single software product can have several architectures serving to achieve the same functionality with different levels of achievement of quality attribute requirements and different preferences, validating APIASD was not subject to resulting outputs. Instead, it was about how the development team tackled each step. A testimonial letter by the architect/project manager was provided. Through this letter, he provided an accumulative opinion encapsulating the whole teams' comments about their experience with applying APIASD. This letter provides insights, that are used through the coming subsections to highlight gains a team can have while applying this framework; possible threats to validity of this case study results; and further improvements that are implied by the experience of applying the framework on *Automatic Quotation* website development.

A. *Reflections from APIASD application on Automatic Quotation*

Adopting APIASD provided the team with an architecting-based smooth introduction to agile software development. This claim is justified by quotes –in italic- excerpted from the testimonial letter and team members' words told through meetings.

The *Automatic Quotation* team obtained the benefits of using "*a balanced approach for developing the architecture of a system using an agile process*", as the project manager characterized the framework in his letter. APIASD enabled the team to construct an initial version of a quality attribute-driven architecture. The initial architecture version is good enough to provide a form of elements' arrangements and relations between them based on architectural drivers serving a set of basic business needs. The team believes the resulting architecture "*will aid in aligning the product in the future with arriving changes*", because they believe that this can be enabled through the incremental and iterative manner, through which the initial architecture evolves till it reaches its complete form. The team was free to choose documentation type and dose, provided that this documentation is abstract but can serve its purpose. This way, architecture documents and artifacts are more likely to be updated, and requirements can be traced back whenever a change takes place. The development team agreed on some gains of using APIASD. These gains are highlighted in the following points.

- **Concentrating efforts on cross-cutting issues**. As declared by the project manager, "*value directions were valuable in guiding user requirements exploration*". Agreeing on some

general crosscutting values gave an overall perspective of the software to be developed and enabled the development team focus on its priorities. Deciding value directions helped the team pay attention to some contextual issues that should be regarded while developing the software needed. The team members liked the idea of value directions' identification, because it doesn't require in-depth diving into details. Identifying value directions' characteristics "*initiated discussions about risks*" related to frequency of changes of a certain value direction's related techniques, or organizational changes, and impact of these changes on other value directions.

• **Provision of a quality attribute-oriented approach to software architecting**. The software director and the project manager welcomed the framework's trend in enforcing handling of quality attributes right from the beginning, and having them considered and monitored till the implementation stage. The reason for this welcoming is that they experienced before situations, when not considering quality attributes from the beginning cost them much rework to be held. It was when they approached the end of developing an ERP system, when they got that the available servers don't enable developing required performance levels that they had to rebuild the system around using external servers. Therefore, the whole team was satisfied after using the framework and having quality attributes paid attention to at all stages. Mapping of decisions to their related quality attribute concerns "*was useful in tracking architectural drivers*" and "*useful of the team to capture the moral of implementing certain techniques*", as the project manager claimed in his letter. Also, quality attribute-related feature cards' priorities were in great manner inspired by stakeholders' preferences about value directions. So, the framework offered mappings to quality attributes throughout its steps.

• **Condensed, yet comprehensive approach to requirements' gathering**. Before applying APIASD, teams in XYZ used to have requirements flowing down through the development hierarchy till a developer gets them. How the customer would get the needed requirements was up to how a developer understands them. Changes in requirements were not always resulting from changes in a customer's preferences; instead, changes were also resulting from misunderstandings about needed requirements. The *Automatic Quotation* team experienced feature cards' writing for the first time, and the project manager claimed that this practice "*helped a lot in gaining a clear understanding of user requirements and preserving them into a form that can be communicated easily*". Closely discussing value behind requirements through many sessions enabled the team members to reach mutual understanding of what a customer needs to get from this project, because they were able to figure out whenever shift of thoughts about requirements happened. The developer after holding a clear set of requirements in hands claimed that "*she has never had requirements of a project with that much clearance*". The team also pointed that the mapping between quality attributes and their addressing feature cards – provided through the utility tree generation practice- was "*a good way for organizing relations*".

• **Enhancement of architectural decision making process**. APIASD suggested many practices, where collaborative decision making is facilitated. The architect's trend in encouraging mutual decision making enabled making full use of these practices. Through meetings, the team members claimed that it was useful to begin considering implications of a chosen strategy on other quality attributes and factors that can affect the level of achievement of the quality attribute addressed by a certain quality attribute. Knowing sensitivity points, tradeoff points, and possible risks of using a strategy was considered by the team members as useful information that can help in discovering a strategy's or a certain pattern's contribution in evolving the architecture, besides providing potential to reuse these strategies or patterns in upcoming projects. Also, the team members noted that reusability of strategies from other projects eased the process of decision making.

• **Construction of a clear, but initial architecture**. Before applying APIASD, the team members had a belief that architecture should be clear and completely defined ahead. This belief was adjusted after experiencing the development of an initial architecture of *Automatic Quotation*, and having this architecture evolving incrementally and iteratively till reaching its final form by the end of the project. For example, while the development team could have initially architected the software for security, this was not a requirement of high priority. Therefore, it was valuable to consider it as its relevant feature cards become into consideration, at the suitable release. In the same time, the team allocated size to security-related feature cards, so as to consider their influence on needed development time and effort. For a requirement of high priority, like availability; the team placed elements for it at the initial version of the architecture and began detailing and exploring techniques to address availability needed concerns in the first release. Further exploration of availability techniques brought more specification into the initial architecture. The initial architecture was clear through constructing views, which "*helped in making the product parts more visual*", and defining elements which the project manager "*insisted on communicating them*" and interfaces between elements. Constructing views is the activity that "*was beneficial in clarifying relations between elements to be implemented*", as excerpted from the letter.

• **Utilizing on-purpose modelling**. The team constructed descriptive models, each one served to reflect a different aspect of the product; and these models worked jointly to present a big picture of the product; or "*a documented agreement on basic parts*". At the same time, as the development process went deeper through releases and iterations, relevant diagrams were used to give the team a close view of information flow, processes, and deliverables. The team believed that the constructed views can "*aid in locating which parts will be affected by changes*". This is because models helped them explore key elements and their relationships. Models helped the team in "*making the product parts more visual*" and this helped the team understand many situations and bring precision to the architecture description,

for example in the case where availability overlaps data services.

• **Doing just the work that adds value**. The application of APIASD enabled the *Automatic Quotation* team to achieve an understanding of the agile mindset and employing it in all practices suggested by this framework. This is because the framework's practices are "*applicable, simple, and understandable*". An example on gaining this mindset is the team's attitude in modelling and choosing which views to construct diagrams for. Another example is given by the project manager's statement about estimation, where he mentioned that estimation using story points "*was clear for its purpose, but unclear for how to decide relative sizes*". Therefore, the project manager claimed that for upcoming projects, he will use weighted factors –like cost, effort, and organizational effect- to decide relative sizes of both quality attribute and functionality-related feature cards.

B. Threats to Validity

The main purpose of this case study is to examine whether applying APIASD can aid in developing software architectures driven by quality attribute requirements, while conforming to the agile mindset of software development. However, there are some threats of external validity of this case study, which affects the degree to which the results can be generalized. These threats are explained below.

• **Participants' experience**. Experience of team members is a critical success factor. An experienced architect is more likely to lead the team towards achieving better results and constructing clearer and more communicable artefacts of applying APIASD. The *Automatic Quotation* team had one architect, whose experience have helped in reusing knowledge from other projects and making good use of templates and guiding materials his team was supplied with.

• **Team members' experience with agile development.** The developer of *Automatic Quotation* had previous background about agile development without practicing it. However, this wasn't enough. Presence of previous experience of the agile approach in software development –among other benefits- could have helped the team in giving feedback about the effect of applying APIASD on the overall schedule compared to using a pure agile process without integrating APIASD. Also, a team with an experience with agile development could have provided feedback about the percentage of refactorings undone (reduced), as a result of applying APIASD.

C. Suggested Improvements to APIASD

After applying APIASD on the case study, many paths were highlighted that can provide potentials to leverage APIASD's effectiveness in achieving its goals. These improvements are suggested and explained as follows:

• **Addressing system-wise issues**. APIASD was about providing guidelines to construct software architectures; this was magnified through feature cards' writing sessions, where the architect was responsible for extracting software-related concerns from system-related ones. APIASD should be extended to handle system architecture-related issues, so as to fit within information systems' with their dimensions other than software. Incorporating guidelines to architect systems developed using the agile approach can help provide more realistic solutions to software architecting problems through putting software architecting into its wider context.

• **Applying APIASD on different projects with varying dimensions**. The project manager of *Automatic Quotation* gave an advice of that "*this framework should be applied on projects of larger scales and different contexts*". This can help in figuring out the influence of a product's size on the type of artefacts needed to represent a resulting architecture, and needed changes to be made on practices of APIASD to accommodate the increase in the number of requirements and/or the number of team members. It is also beneficial to identify the influence of a project's size on time and effort needed to develop an architecture for it. Also, there is a need to apply APIASD on projects with fast changing requirements to identify the percentage of changes implying architectural modifications to the total number of changes.

V. CONCLUSION

Agile architects should advocate a development culture that values making architectural design decisions based on careful analysis of requirements and give a due care to quality attribute requirements in advance, especially that they do not change as rapidly as functional requirements. APIASD is believed to provide guidelines and practices to develop a software architecture while adhering to both of agile development values and software architecting principles. APIASD was applied on a case study, whose analysis confirms the potential of APIASD to achieve and maximize returned business value on the long term. Still, a lot should be done to apply APIASD on more case studies to get to a vision about APIASD's effect on reduction in time and number of refactorings associated with software architecture development.

REFERENCES

[1] Nord, R. L., Tomayko, J. & Wojcik, R., (2004), "Integrating Software-Architecture-Centric Methods into Extreme Programming (XP)", CMU Software Engineering Institute, Pittsburgh, PA, USA.

[2] El-Khawaga, G. H., Galal-Edeen, G. H. & Riad, A. M., (2013), "Architecting in the context of agile software development: fragility versus flexibility", *International Journal of Computer Science, Engineering, and Applications (IJCSEA)*, Vol. 3, No. 4, 25-37.

[3] Hofmeister, C., Nord, R. & Soni, D., (2000), *Applied Software Architecture*, Addison Wesley Professional, Indiana, USA.

[4] Barbacci, M., Ellison, R., Lattanze, A., Stafford, J., Weinstock, C. & Wood, W., (2003), "Quality Attribute Workshops (QAWs)", CMU Software Engineering Institute, Pittsburgh, PA, USA.

[5] Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson, P., Nord, R. & Wood, B., (2006), "Attribute-Driven Design (ADD)", CMU Software Engineering Institute, Pittsburgh, PA, USA.

[6] Clements, P. C., Kazman, R. & Klein, M., (2002), *Evaluating software architectures: Methods and case studies*, Addison Wesley Professional, Indiana, USA.

[7] Boehm, B. & Turner, R., (2004), *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison Wesley Professional, Indiana, USA.

[8] Smith, G. & Sidky, A., (2009), *Becoming Agile in an imperfect world*, Manning Publications Co., NY, USA.