ARCHITECTING IN THE CONTEXT OF AGILE SOFTWARE DEVELOPMENT: FRAGILITY VERSUS FLEXIBILITY

G. H. El-Khawaga [1], Prof. Dr. Galal Hassan Galal-Edeen [2], and Prof. Dr. A.M. Riad [1]

[1] Faculty of Computers and Information, Mansoura University, Egypt.
ghelkhawaga@acm.org
[2] School of Sciences & Engineering, American University in Cairo, Cairo, Egypt.
Galal@acm.org
[1] Faculty of Computers and Information, Mansoura University, Egypt.
amriad2000@mans.edu.eg

ABSTRACT

As the size and complexity of software systems increase, software development process couldn't be bound to just codifying some modules that serve needed functionality and finding the appropriate configuration of these modules. Instead, a growing need emerges to sketch a big picture of the whole system that not only identifies basic parts of functionality, but also harmonizes these parts internally, manages how these parts will provide needed functionality, and paves the way for future adaptation. The answer to this need was software architectures. The agile approach to software development wasn't about introducing a magical solution that will handle all development problems. However, the agile architecting approach is believed to be a source of new problems. Through this paper, we are going to explore agile architecting problems and what is needed to achieve an architecting approach that can be agile, while serving its purpose of producing a stable architecture.

KEYWORDS

Software Architecting, Agile Software Development, Agile Architectures, Fragile Architectures.

1. INTRODUCTION

Over years, software architectures are believed to be the first step on moving from problem space to solution space, as architectures provide an infrastructure for building scalable systems [1] which can handle several complexity forms such as geographic distribution, number of requirements and number of stakeholders [2]. Software architecture design not only encompasses the whole system's life but also it can sometimes affect other systems in case of product-lines development or while reusing parts of a system in other systems. Software architecture paves the way for achieving specific qualities of the system being developed, e.g. modifiability, performance, and these qualities are critical for the system's future as well as being critical for its implementation and operation. It forms a map characterizing relations between whole system's parts.

Regardless of the growing need of having a big picture of the system under development and despite the increasing criticism of agile methods for not offering suitable process for large-scale systems; agilists view architecting work in light of traditional development [3]. They suppose that architecting is a tedious work that requires upfront design and massive documentation [4]. Kruchten claimed that agile teams may consider agile values and good architecture practices to be contradicting [5]; while in their survey, Falessi et al. concluded that the principles of architecture-centric methods are believed to be supportive to agile values [2]. However, while agilists are always concerned with delivering value more frequently, they believe that architecture's value is not visible or at least is not achievable on the short term [5].

However, agilists tried to tame architecting to acquire an agile flavour; therefore, they provide some guiding architecting principles to consolidate agility in the resulting artefacts of the architecting process. Leffingwell et al. concluded the agile architecting principles into phrases like "*the teams that code the system design the system*", "*they build it, they test it*", "*system architecture is role collaboration*" which emphasize the importance of role collaboration and variation in experiences among team members to enhance self-organization among team members [6]. Leffingwell et al. added a principle through the phrase "*build the simplest architecture that could possibly work*" to cope with agilists' chase of simplicity and doing only work that would add tangible value frequently [6]. Leffingwell et al. added a principle calling that "*when in doubt, code it out*" which comes very consistent with agilists' belief that working software is the main measure of success [6]. And finally, the last principle of agile architecting Leffingwell et al. phrased shows that "*the bigger the system, the longer the runway*" [6]. This runway means architectural fundamental elements developed in earlier iterations that would play as an infrastructure for the coming parts of the system. It does not contain functionality but shows framework [7]. By having an architectural runway; agilists bid on attacking the right system requirements that are hard to incorporate late in the development cycle. The size of the system determines the amount of work required in developing its runway; and here comes the challenge for agilists' belief and understanding of this principle. The more the team and the project are larger, the more the effort required in developing an architecture's runway; then –in our opinion- the more agilists' are likely to dismiss this principle and ignore long-term expected benefits of constructing such runway in enabling incremental development and smooth accommodation of further changes. We think that agilists disregarded this principle through their chase of frequent delivery and short feedback cycles. Unfortunately, the agile architecting principles are inconsistent with what is already done in architecting software products using an agile approach.

Driven by their trend to maximize the work undone, minimize the work done upfront, defer decisions to the last responsible time, invest in code rework, and react to changes rather than design for them as Coplien & Bjornvig explained, we argue that agilists are more likely to quit investing in building the software architecture of a product and wait for it to emerge as the product grows incrementally [8]. Software architectures are believed to be important to learn about and preserve architectural knowledge. Jansen & Bosch showed that in absence of knowledge about design decisions themselves, there is more possibility for violating architectural decisions' constraints, and this may lead to architectural degradation [9]. They also confirmed the difficulty of tracing cross-cutting design decisions which are achieved through various components; and consequently, architecture change and maintenance will be a difficult and expensive process. Through sections two and three we are going to explore problems resulting from their trend in architecting, and proposed solution directions to them, and our insights into how to reach a balance between architecting while preserving process agility in section four.

2. TROUBLESOME AGILE ARCHITECTING: ORIGINS AND IMPLICATIONS

Through the introduction we have presented how agilists communicate their beliefs about architecting. How the proposed agile architecting principles are translated into practices is where the problems reside. We argue that these practices haven't helped in coming up with an architecture that can introduce a form upon which the structure of the product to be built can emerge incrementally. Instead, these practices contributed to having a fragile architecture which would have the problem of being expensive to fix. These problems can occur when early, simple design decisions result in foreseeable changes that cause breakage in the design beyond the ability of refactoring to handle [10]. Through the next subsection we are going to explore these practices one by one and then we will discuss how they contributed into limiting the applicability of the agile software development approach.

2.1 Agile Architecting Practices

Agile architecting revolves around three basic practices; using refactoring, getting to a communicated metaphor about the architecture to be built, and developing spike solutions [11].

I. Architectural Refactoring

Agilists use refactoring as their main practice to accommodate changes as they come up during software development life cycle, as well as design improvement. Refactoring is defined according to Mens & Tourwe as: *"The process of changing a software system in such a way that it does not alter the external behaviour of the code, yet improves its internal structure"* [12]. As Moser et al. claimed, refactoring in the context of agile methods increases software understandability, improves software design, and accelerates the coding process and finding bugs [13]. Refactoring in higher levels of abstraction like architectural levels in called *architectural refactoring* [11]. Agilists use architectural refactoring to achieve quality attributes that are not provided by ordinary refactoring.

However, Barbacci et al. argued that not all quality attributes such as security can be accommodated later in implementation through refactoring [14]. They showed that some quality attributes' components and mechanisms must be designed early in the life cycle. Besides, frequent non-systemic modifications to requirements can make the architecture fragile [15], and can result in architectural degradation, which leads to a mismatch between the actual functions of the system and its original design [16], and subsequently upgrades and fixes become expensive to implement. This case is called architectural erosion [17]. Architectural erosion is defined as the regressive deviance of an application from its original intended architecture resulting from successive changes [18]. Architectural erosion leads to increasing resistance to change and subsequently high cost of maintenance [19]. Architectural degradation causes are mainly mapped to late-lifecycle changes, which are considered to be the most crucial, risky, and expensive when they are changes to requirements [16]. Therefore, the earlier to make changes is the better.

Buschmann mentioned some characteristics of refactoring such as that it can improve only developmental qualities such as maintainability of design, and it is not suitable for inserting new functions or improve operational quality attributes, because such refactorings would alter a system's behaviour [20]. This claim sounds reasonable as long as the main aim of refactoring is to alter internal structure without changing external behaviour, and it also raises critical questions about the viability of refactoring –in the context of agile development- to leverage a system's architecture and alter it later to insert missed quality attributes. Coplien & Bjornvig firmly emphasized the ineffectiveness of using refactoring to fix architectural problems, and Coplien & Bjornvig proved the correctness of his claim by citing reports, which conclude not only that Test-Driven Development (TDD) provided no architectural benefit of refactoring, but also TDD may cause the architecture to deteriorate [8]. This opinion contradicts Breivold et al.'s claim that the strength that TDD gives to architectural development, while not denying the increase in time and effort and the loss of design and architecture decision traceability while adopting TDD [21]. Bohem & Turner claimed that experiences where cost of change remained low over time were associated only with small applications and expert programmers who know where to refactor quickly and correct defects [10].

II. Metaphors

A metaphor is a shared story that guides development team on how the system should look like and how it works [3]. A metaphor is defined according to West as: *"A story that everyone – customers, programmers, and managers –can tell about how the system works"* [22]. Agilists used metaphors to describe the logical architecture of the system in form of some stories which are familiar for the development team [23]. A metaphor's effectiveness is evaluated after developing it. Agilists used metaphors sometimes to substitute lack of architecture [5].

Irit Hadar showed that despite creating metaphors to achieve concretion of abstract concepts, metaphors sometimes don't provide the desired clearance and cause confusion and meaning distortion [24]. Metaphors were argued to be a methodological weak point [22], because they usually don't give precise or definite meaning and they become subject to how each member in the team would understand its meaning and moral. In the study conducted by Tomayko & Herbsleb, they concluded that metaphors were proved not to be costly, but useless in achieving either communication of project's idea or guidance through the architecture development process [25]. This practice was widely ignored [25] because it has never been completely understood. Even agilists like Fowler claimed being not able to understand what is meant by a metaphor and how to use it [22]. Kent Beck, the godfather of XP, tried to explain the benefits of metaphors in guiding developers through the exploration of system architecture but suddenly he surrendered to his critics and re-explained a metaphor as a gained skill not as a practice [3], [26]. However, practitioners like West & Solano argued about the necessity of having system metaphors because not having them would enforce developers to employ default sets of metaphors and this would impact the resulting design negatively [26], because there would be a demanding need to continue in refactoring the resulting design to include the desired requirements. Despite their attempts to increase advocacy of metaphors' creation, we think that they provided weak propositions which don't qualify metaphors to substitute for lack of architectures.

III. Architectural Spikes

Agilists are seeking to eliminate possibilities of being mistaken or choosing the wrong parts to implement. Spiking is one of their practices to assess risks resulting of design choices. Spikes are defined according to Tomayko as: *"A rapid development of a prototype that answers a single question about requirements"* [27]. A spike is a throw-away code that helps in estimating the difficulty of a certain task and assessing technical risks resulting of advocating a certain architectural design decision. Spikes are created to explore technically difficult portions of the system that would be programmed and solved, while prototypes are used to validate the system to be built [28]. We argue that architectural spikes can't serve their purpose of ensuring the suitability of architectural design decisions to achieving needed requirements. Mainly basic architectural design decisions are made to satisfy driving requirements, especially quality attribute-related requirements. So, constructing an architectural spike with a single user feature can help measure an architecture's achievement level of only design or quality attribute concerns localized in the implemented feature, which can't be used as a determinant of the whole architecture achievement level of certain quality attribute. This can be explained by believing in the fact that an architecture's achievement level of a certain quality attribute is the cumulative achievement level of different concerns related to this quality attribute across all the modules interweaved together to form the structure fitting into the form introduced by the evaluated architecture. Thinking that an architecture's achievement level of a certain quality attribute can be measured through a number of single architectural spikes and then generalized to the whole architecture is like thinking that a vertical slice of a not-uniformed soil can be used as a determinant of characteristics of a vast desert.

2.2 Agile Architecting as Agile Development Applicability Hinderer

We believe that agilists' trend in architecting has resulted in many problems, some of them show on the short term, and others show on the long term. For example, agilists tend to do the simplest work upfront and not to include requirements which are not acquired explicitly by customers, so as not to stick into developing not needed features. Agilists' tendency has caused them to ignore even foreseen changes [10], like these changes needed to add quality attributes to a product under development. Adding quality attributes through a software system's life cycle introduces new requirements, thus it can be considered some sort of perfective changes. As cited by Mohagheghi et al., Lientz et al. reported that 60.3% of the maintenance effort was categorized as perfective [29]. The challenges accompanying quality attributes' accommodation -whether these challenges are in general or are attributed to the usage of agile methodologies in software development- have resulted in having perfective changes to be of the highest percentage of the total maintenance efforts. Mockus & Votta's study revealed that perfective changes -as well as being the highest to add more lines of code- are more time consuming than adaptive and corrective changes [17].

Methodologists and process practitioners have highlighted some situations where agile methods offered limited support. We have categorized these limitations according to: (1) their effects on personnel involved in the project, including developers, stakeholders and customers and how to organize the work environment for a project; and (2) their effects on choosing applications for which agile methods would be suitable. We will not go through the details of the first category because this type of limitations is out of scope for this study. The second category is about *application-oriented limitations.* These limitations restrict the applicability of agile methods to certain applications. Certain characteristics should be present in an application to ensure that using an agile method for developing it would be a viable option. However; agile architecting problems are believed to be the main stimulus of these limitations. Examples on these limitations are [30]:

- *Limited support for building reusable artefacts.* Many agilists emphasized the need to set up for the next development level where extending system or integrating it with another one is important [31]. However; agile proponents have what they consider a golden design rule of "YAGNI" meaning "You Are not Going to Need It" [3], [31]. This slogan implied that they do not model, or implement what they are not going to need for the current time. They regarded mainly specific-purpose applications while not giving the right share of attention to general-purpose applications [30]. As a result, they did not produce reusable artefacts. Upcoming development efforts are costly and complicated in the absence of reusable artefacts. Bohem & Turner emphasize that YAGNI both throws away valuable architectural support for foreseeable requirements and frustrates customers who want developers to believe their priorities and evolution requirements are worth accommodating [10].

- *Limited support for developing safety-critical software.* Safety-critical systems are believed to have tight constraints and many risks. Developing such systems requires high quality control mechanisms, more order, in-depth requirements specification, more planning, and more modelling and documentation [30]. Safety-critical systems demand high level performance and reliability. As we've mentioned above, agile methods deal with quality attributes as functionalities which can be added during iterative development. Whereas; these systems won't be easy to modify during implementation. Agile critics argued in many positions that agile processes won't be suitable to develop safety-critical software [10], [30] where an error would cause direct damage to human lives.

- *Limited support for developing large, complex software.* While emphasized in many positions that agile development methods are not suitable for developing large and/or complex systems [10], [30]; Holler had a contradictory opinion about this issue [32]. Holler claimed that the difficulty of handling large projects is common to all methodologies and not an agile approach-specific problem, because as the size grows, the complexity, dependency and massiveness of the components of a system increases [32]. Agilists depend on refactoring to modify their architectures during development, while researchers emphasize that refactoring on large-scale would affect resulting software quality [1]. Also, in large scale systems, refactoring proved to be expensive as size grows [4], [31]. Scalability is an important issue for large software systems, reusability is an important issue for product line systems. However, to obtain scalability or reusability, quality attributes should be considered right from the beginning. To have a big picture of a software system, and to consider quality attributes from the beginning and to have techniques and forming structure to address these quality attributes; time should be spent ahead to develop an architecture for the system to be developed. Large projects need traditional planning and specification to deal with increased project complexity [10], [4]. Agile methods were proved to be not suitable as the level of complexity and size of project scales.

It is clear now how the agile approach to architecting affect not only the agile software development approach applicability on certain types of applications, but also on the short term on percentage of maintenance spent in fixing what could have been handled if time was spent ahead to get an initial version of the architecture of a software system to be developed. Through the next section, we are going to highlight efforts exerted to redirect agilists and agile software development towards getting a suitable approach to architecting software systems to be developed.

3. WORKING AROUND AGILE ARCHITECTING PROBLEMS

Agilists and methodologists worked together to increase agile methods' applicability and to overcome application-oriented limitations while preserving benefits and advantages agile methods can offer. Methodology practitioners believe that the amount of architecting done in the design phase of an agile process is not enough to produce a flexible architecture -which would begin as an initial one and evolve incrementally- and not a fragile architecture- which emerges by chance in an unplanned manner and hence can't provide a big picture of the software's evolution and change accommodation options through its lifetime. Through the coming subsections we are going to discuss efforts exerted to come up with an architecting approach that suits flexibility demanded by the agile approach to software development.

3.1 Altering the Existing Agile Architecting Approach

Methodology practitioners tried to insert more architecting practices into the design phase. These practices aim at overcoming architecting problems especially those caused by not considering quality requirements right from the beginning in the architecture.

I. Continuous Architectural Refactoring (CAR) & Real Architecture Qualification (RAQ)

As a way to discover inefficiencies that could mislead the system's architecture toward an unmanageable and unsuitable shape unless resolved as soon as possible; Sharifloo et al. suggested the *Continuous Architectural Refactoring (CAR)* as a practice for identifying architectural smells and deciding solutions to remove them. CAR is a practice that is applied in parallel with agile iterations [11]. During an iteration, each team builds a model for the tasks they have already finished. An architect will integrate models received with more recently received models and analyzes the resulting diagram carefully to identify smells and make decisions to solve them. CAR aids well in supporting developer-related quality attributes while it cannot efficiently analyze user-related quality attributes [11]. Sharifloo et al. suggested another practice to handle user-related attributes; *Real Architecture Qualification* (RAQ) [11]. RAQ is applied at the end of all iterations to test the working system and enables collaboration with stakeholders to identify user-related quality attributes like performance. RAQ is a kind of a brainstorming session in which the final architectural model is discussed and evaluated, and the result of this discussion is reflected through refactoring decisions made.

CAR is advantageous of easily identifying architectural smells and improving developer-related quality attributes, however it depends on the architect solely and this trend violates communication and collaboration practices. Also, we argue that this practice requires extra modelling efforts. Modelling remains an extra work, which may overload the development team and require highly-skilled personnel. Also, this practice produces redundant models because developers create models for the tasks they have already finished, while the more cost efficient is to modify the original models according to what have been already achieved rather than creating new models. RAQ seems to be a time consuming practice when considering the amount of time consumed in holding brainstorming sessions and their subsequent sessions.

Unfortunately, in CAR & RAQ, there is no design for quality attributes in advance, so more refactorings are triggered and this may be costly as the development process proceeds. Also, even if the authors claim that CAR & RAQ are not time consuming practices because they are held in parallel with the main development process [11], actually this parallelism could be disturbing and dispersing of the development team. Jeon et al. tackled CAR & RAQ from another side [33]. They argue that these methods don’t provide guidance on how to analyze quality attributes. We are totally with this opinion.

II. Quality Attribute Workshop (QAW) & Attribute-Driven Design (ADD)

Practitioners like Nord et al. argued about the necessity and viability of integrating architecture-centric practices into agile methodologies, especially XP so as to insert and force designing for quality attributes inclusion from the beginning, and to create the architecture for a software product rather than depending on metaphors to have one, and to evaluate the architecture for its strategies’ ability to achieve the required level of quality attributes addressed [34]. Up to the moment we are writing this, there are no studies reporting applying QAW & ADD in the context of an agile software development approach, but there are recommendations and arguments about their applicability.

If we regarded QAW independently, it is believed to enhance customer collaboration by involving stakeholders in quality attributes definition process. It also bases planning and story generation processes on business goals; this helps deliver more business value. The architect can use resulting stories to design the architecture including quality requirements. This would lessen the number of refactorings and help plan for foreseen changes. If we regarded ADD independently, we would find that it prepares the architecture to accommodate quality attributes right from the beginning, and this would reduce the number of refactorings through the system development life cycle. Also, ADD encourages incremental development as it decomposes the system into smaller elements. QAW/ADD lessens required refactorings and does not handle quality attributes as functionalities. It is also worth mentioning that ADD contains a verification step which takes place at the end of the ADD, so whenever a change occurs, the design can be adjusted in the verification step. QAW/ADD conforms to agile values in facilitating change accommodation by continuous analysis and planning, increases customer engagement and gained feedback, maximizes business value, and enables incremental development.

3.2 Replacing Currently Adopted Architecting Approach

In this solution path, agilists suggested replacing the whole design process with a modified subprocess that inherits agile methods' advantages accompanied with another software development approach and at the same time avoids shortcomings of the standalone approaches.

Agile Model Driven Development

AMDD is the agile version of Model Driven Development (MDD) [35]. While the moral of MDD contradicts with the value of agile development, which confirms that working software is the primary artefact; Ambler explained that agile models are those which are barely good enough [35]. This means that AMDD's trend is not to go far in extensive modelling but it is to create models which are at the most effective point they could possibly be at. AMDD's target is to guide developers and stakeholders throughout an effective design process. In his introduction to AMDD, Ambler advocated providing a big picture at the beginning of each release through the envisioning activity [35]. However, this big picture would be at the level of a project release. While claiming that AMDD is a critical strategy for scaling agile software development beyond the small, co-located team approach seen in the first stage of agile adoption [36], lack of a big picture at the project level would cause challenging integration problems [37] if applied for larger, more complex projects.

While MDD is referred to as a set of approaches in which code is automatically or semi-automatically generated from more abstract models [38], Ambler has argued that using modelling tools would require modelling skill set and specialized expertise [39]. However, Zhang & Patel showed through a case study that the productivity in number of source code lines per staff month increased in case of high percentage of automatically generated code compared to hand coding [40]. Also, Ambler argued that with AMDD, a little bit of modelling is done and then a lot of coding [36]. This declaration seems to be a clear violation of the basic idea of MDD, which aims at moving the development efforts from programming to the higher level of abstraction and concentrate efforts on modelling and generating needed code from these models [38]. AMDD can also be affected by MDD's problems such as requiring high expertise, moving complexity rather than reducing it and other problems [38], provoked by using UML as the standard modelling language enabling MDD.

4. DEFINING ELEMENTS OF AGILE ARCHITECTING

Now, we can collect the elements of a desired agile architecting approach. Let's first talk about what is needed to have an *agile* approach. An approach to develop software architecture should follow both of top-down and bottom-up approaches. It should adopt minimalist approach to architecture while the highest priority architectural drivers are identified are identified and the simplest design effort is done to achieve them. An initial architecture is achieved at the beginning to serve as the base for a final form that will emerge through continuous evolution and as a result of growing and accumulated understanding of business goals and user requirements that either come up or change through the project's lifetime. An initial version of the architecture enables project management to organize work assignments, and configuration management to setup development infrastructure; and the product builders to decide on the test strategy [41]. It should also enable further changes to affect the architectural level as the whole product development lifecycle proceeds. Agilists believe that nothing can be done right from the beginning [31], so this rule applies also to architecting. If the resulting architecture won't be right and at its final form from the beginning as Isham believed, then incremental architecting won't be a waste of time [42]. Instead; incremental architecting would enable synchronizing the system with changing conditions rather than adopting a single straight direction from the beginning [4]. To be agile, an architecting trend should adopt simplicity in design, and waiting till more uncertainty about user requirements is reduced. This way, it enables keeping architecture design decisions to the minimum while ensuring meeting user requirements. Changes in user required functionalities are not to be constrained or prohibited. Therefore, changes in quality attributes are to be managed properly through building an architecture where insights into changes' impacts on quality attributes –if any- can be provided. Concentrating on driving the resulting architecture by quality attribute requirements reduces the need for architectural refactoring to include quality attributes. An *agile* architecting approach should praise and encourage individuals and interactions among them through brainstorming sessions and joint decision making facilitated by discussions and voting. It should also produce the minimal amount of documents, provided that ideas and needed values of design decisions are communicated among team members.

To explore what is needed to have an architecting approach to be inserted into an agile software development process, we need to define elements of *architecting* and insist on their existence in a light yet value-producing form. To have an agile *architecting* approach, architectural drivers – especially quality attributes- should be clearly identified in definitive forms so as not to leave the decision of accommodating them for afterthought. Architecting to include quality attributes from the beginning offers a long-term value concerning lessening the number of refactorings conducted through software development lifecycle, and enables scalability of a system provided that the effects and interactions between quality attributes, concerns, and user preferences are known ahead. Besides, guidance is desired as to how to come up with an architecture that satisfies all architectural drivers present up to the moment in a flexible design. This need can be achieved by offering mappings between business goals and user requirements, and the resulting architectural artefacts which are supposed to be reflecting these goals and drivers, to make sure that a development team is always having business goals as their main reference to develop an architecture and to adhere to agile development mindset that always keeps business value as a main motive. Another aim is to have a balance between architectural concerns and context. From agilists' tend in handling quality attributes, it can be concluded that agilists consider the context while specifying quality attributes more than spending time in analyzing concerns. This represents a clear violation of their beliefs that every project is a unique case that besides having an associated contextual and domain knowledge, there are also concerns specific to the case in hands, and these concerns need to be carefully considered. There is a need for achieving an architecture design based on a set of drivers derived while relying on considering context as well as concerns.

Having the rationale of architectural decisions is an aim to be pursued. A clear obvious rationale 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. Also, offering change impact information in advance is an aim to be achieved, so as to enable developers to assess a change's associated risks and to identify the suitable techniques to deal with changes based on these changes' similarities especially those changes affecting architecture and quality attributes. 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 clear ahead; so as not to have a design that apparently swallowing all changes as they come up, while in fact it suffers severe mess in its architecture, which degrades gradually. Obtaining change impact information is considered to be cost beneficial because assessing the effect of changes on the architectural level solely reduces the costs associated with the whole software evolution process as many implementation details need not to be considered are eliminated [43].

Having an explicit architecture, even if an initial incomplete version of it, is necessary for a process adopts 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. Architecting practices used to produce this initial architecture should be integrated into the development process so as to ensure workflow between development steps, always tracking business value and being driven by customer needs, and tracking architectural decisions implications and rapidly handling conflicts between requirements especially architectural drivers. The architecture of a software product should be put in place to act as a firm foundation for responding to upcoming changes.

5. CONCLUSION

As agility is more about a mindset more than being about practices; once the morals and value behind each architecting activity are reached, architecting activities can be tackled in numerous ways and can serve their purposes without violating agile values. Architecting activities can be lightened and can yield architecting artefacts that can build a stable yet flexible architecture. While the battle between agilists and architects is going harder, rationalists from both camps believe that agile development and architecting are not at odds. Our basic concern through this paper was to highlight problematic trends in currently adopted agile architecting and provide insights into how to come up with an approach which is truly agile and can truly yield an architecture that can accommodate changes while not being eroded.

REFERENCES

[1] Ramakrishnan, S., (2010), “On Integrating Architecture Design into Engineering Agile Software Systems”, proceedings of the Informing Science & IT Education (InSITE) conference, Cassino, Italy, 21-24 June, Informing Science Institute (ISI), pp. 9-25.

[2] Falessi, D., Cantone, G., Sarcia, S. A., Calavaro, G., Subiaco, P. & D’amore, C., (2010), “Peaceful Coexistence: Agile Developer Perspectives on Software Architecture”, IEEE Software, vol. 27, no. 2, pp. 23-25.

[3] Jensen, R. N., Moller, T., Sonder, P. & Tornehoj, G., (2006), “Architecture and Design in eXtreme Programming; Introducing Developer Stories”, proceedings of Extreme Programming and Agile Processes in Software Engineering, 7th International Conference (XP 2006), Oulu, Finland, 17-22 June, Springer Verlag, pp. 133-142.

[4] Erdogmus, H., (2009), “Architecture Meets Agility”, IEEE Software, vol. 26, no. 5, pp. 2-4.

[5] Abrahamsson, P., Babar, M. A. & Kruchten, P., (2010), “Agility and Architecture: Can They Coexist?”, IEEE Software, vol. 27, no.2, pp. 16-22.

[6] Leffingwell, D., Martens, R. & Zamora, M., (2008), “Principles of Agile Architecture”, Leffingwell, LLC . & Rally Software Development Corp.

[7] Faber, R., (2010), “Architects as service providers”, IEEE Software, vol. 27, no. 2, pp. 33-40.

[8] Coplien, J. O. & Bjornvig, G., (2010), Lean Architecture: for Agile Software Development, Wiley Publishing, Indianapolis, Indiana, USA

[9] Jansen, A. & Bosch, J., (2005), “Software Architecture as a Set of Architectural Design Decisions”, proceedings of the 5th IEEE/IFIP working conference on software architecture (WICSA 2005), Pittsburgh, PA, USA, 6-10 November, Nord, R., Medvidovic, N., Krikhaar, R., Stafford, J. & Bosch, J. (Eds.), IEEE, pp. 109-120.

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

[11] Sharifloo, A. A., Saffarian, A. & Shams, F., (2008a), “Embedding Architectural Practices into Extreme Programming”, proceedings of the 19th Australian Conference on Software Engineering (ASWEC'08) Perth, WA, Australia, 26-28 March, IEEE, pp. 310-319.

[12] Mens, T. & Tourwe, T., (2004), “A survey of software refactoring”, IEEE Transactions on Software Engineering, Vol. 30, No. 2, pp. 126-139.

[13] Moser, R., Abrahamsson, P., Pedrycz, W., Sillitti, A. & Succi, G., (2007), “A Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team”, Proceedings of the second IFIP TC 2 Central and East European Conference on Software Engineering Techniques, (CEE-SET’07), Poznan, Poland, 10-12 Oct., Meyer, B., Nawrocki, J. R. & Walter, B. (Eds.), Springer-Verlag Berlin, Heidelberg, pp. 252-266.

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

[15] Khan, S. S., Greenwood, P., Garcia, A. & Rashid, A., (2008), “On the Impact of Evolving Requirements-Architecture Dependencies: An Exploratory Study”, Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE'08), Montpellier, France, 16-20 June, Bellahsene, Z. & Leonard, M. (Eds.), Springer, pp. 243-257.

[16] Williams, B. J. & Carver, J. C., (2007), “Characterizing Software Architecture Changes: An Initial Study”, proceedings of the First International Symposium on Empirical Software Engineering and Measurement, (ESEM'07), Madrid, Spain, 20-21 Sept., IEEE, pp. 410-419.

[17] Mockus, A. & Votta, L. G. (2000), “Identifying reasons for software changes using historic databases”, Proceedings of the International Conference on Software Maintenance, San Jose, CA, USA, 11-14 Oct., IEEE, pp. 120-130.

[18] Taylor, R., Medvidovic, N. & Dashofy, E., (2009), Software Architecture: Foundations, Theory, and Practice, Wiley publishing, Indianapolis, Indiana, USA.

[19] Perry, D. & Wolf, A., (1992) “Foundations for the study of software architecture”, ACM SIGSOFT Software Engineering Notes, Vol. 17, No. 4, pp. 40-52.

[20] Buschmann, F., (2011), “Gardening Your Architecture, Part 1: Refactoring”, IEEE Software, Vol. 28, No. 4, pp. 92-94.

[21] Breivold, H. P., Sundmark, D., Wallin, P. & Larsson, S., (2010), “What Does Research Say about Agile and Architecture?”, Proceedings of the Fifth International Conference on Software Engineering Advances (ICSEA), Västerás, Sweden, 22-27 Aug., IEEE, pp. 32-37.

[22] West, D., (2002), “Metaphor, Architecture, and XP”, proceedings of the third International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2002), Sardinia, Italy, 26-30 May, pp. 101- 104.

[23] Khaled, R., Barr, P., Noble, J. & Biddle, R., (2004), “Extreme programming system metaphor: A semiotic approach”, proceedings of the 7th International Workshop on Organizational Semiotics, Setúbal, Portugal, 19-20 July, pp. 152-172.

[24] Mancl, D., Hadar, E., Fraser, S., Hadar, I., Miller, G. R. & Opdyke, B., (2009), “Architecture in an Agile World”, proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPLSA 09), Orlando, Florida, USA, 25-19 October, ACM, pp. 841-844.

[25] Tomayko, J. & Herbsleb, J., (2003), “How Useful Is the Metaphor Component of Agile? a preliminary study”, school of Computer Science, Carnegie Mellon University.

[26] West, D. & Solano, M., (2005), “Metaphors be with you”, proceedings of Agile Development Conference (ADC'05), Colorado, USA, 24-29 July, IEEE, pp. 3-11.

[27] Tomayko, J. E., (2002), “Engineering of Unstable Requirements Using Agile Methods”, International Workshop on Time-Constrained Requirements Engineering (TCRE'02), Essen, Germany, 9 September.

[28] Dögs, C. & Klimmer, T., (2004), “An Evaluation of the Usage of Agile Core Practices , Software Engineering and Computer Science, Blekinge Institute of Technology, Ronneby, Sweden.

[29] Mohagheghi, P. & Conradi, R., (2004), “An empirical study of software change: origin, acceptance rate, and functionality vs. quality attributes”, Proceedings of the 2004 International Symposium on Empirical Software Engineering, (ISESE '04), Redondo Beach, CA, USA, 19-20 Aug, IEEE, pp.7-16.

[30] Turk, D., France, R. & Rumpe, B., (2002), “Limitations of Agile Software Processes”, the third International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2002), Sardinia, Italy, 26-30 May.

[31] Cockburn, A. (2006), Agile software development: The Cooperative game, 2nd Edition, Addison-Wesley, Boston, US.

[32] Holler, R., (2010), “Five myths of agile development”, VersionOne, http://pm.versionone.com/5-agile-development-myths-whitepaper.html, last access: 26 March 2013.

[33] Jeon, S., Han, M., Lee, E. & Lee, K., (2011), Quality Attribute Driven Agile Development, in proceedings of 2011 ninth International Conference on Software Engineering Research, Management and Applications (SERA), Baltimore, MD, USA, IEEE, pp. 203-210.
[34] Nord, R., Tomayko, J. & Wojcik, R., (2004a), Integrating Software-Architecture-Centric Methods into Extreme Programming (XP), Pittsburgh, Pennsylvania, USA, Software Engineering Institute, CMU.
[35] Ambler, S. W., (2007a), "Agile Model Driven Development (AMDD)", XOOTIC, Vol. 12, No. 1, pp. 13-21.
[36] Ambler, S., (2007b), "Agile Model Driven Development (AMDD): The key to scaling agile software development", available at: http://www.agilemodeling.com/essays/amdd.htm , last access: 26 March 2013.
[37] Elssamadisy, A. & Schalliol, G., (2002), "Recognizing and responding to bad smells in extreme programming", proceedings of the 24th International Conference on Software Engineering (ICSE'02), Orlando, Florida, USA, 19-25 May, ACM, pp. 617- 622.
[38] Picek, R. & Strahonja, V., (2007), "Model Driven Development – Future or Failure of Software Development?", proceedings of the 18th International Conference on Information and Intelligent Systems (IIS2007), Varaždin, Croatia, 12-14 September, pp. 407-414.
[39] Sharifloo, A. A., Saffarian, A. & Shams, F., (2008b), "Toward Empowering Extreme Programming from an Architectural Viewpoint", proceedings of the 9th International Conference on Agile Processes in Software Engineering and eXtreme Programming (XP 2008), Limerick, Ireland, 10-14 June, Springer Verlag, pp. 222-223.
[40] Zhang, Y. & Patel, S., (2011), "Agile Model-Driven Development in Practice", IEEE Software, Vol. 28, No. 2, pp. 84-91.
[41] Bass, L., Clements, P. & Kazman, R., (2003), Software Architecture in Practice, Addison-Wesley Professional, Boston, USA.
[42] Isham, M., (2008), "Agile Architecture IS Possible -You First Have to Believe!", proceedings of Agile 2008, Toronto, Ontario, Canada, 4-8 August, IEEE, pp. 484- 489 .
[43] Zhao, J., Yang, H., Xiang, L. & Xu, B., (2002), "Change impact analysis to support architectural evolution", Journal of Software Maintenance and Evolution: Research and Practice, Vol. 14, No. 5, pp. 317–333.

Author 1:

G. H. El-Khawaga
Teaching Assistant, Department of information systems
Faculty of computers and information
Mansoura University, Egypt.
ghelkhawaga@acm.org

Biographical note: Ghada Hesham El-Khawaga is working as a teaching assistant and researcher for more than 6 years. She is involved in the fields of information systems analysis and design, and software engineering with active participation and positive motivation. Her main interests are in the areas of software architecting, information systems development methodologies, and software engineering. She is a member of the ACM and IEEE. Ghada is seeking to be a contributor academic information systems' researcher on the short term and a consultant on the long term.

Author 2:

Prof. Dr. Galal Hassan Galal-Edeen
Computer Science Department,
School of Sciences & Engineering,
American University in Cairo,
Cairo, Egypt.
Galal@acm.org

Biographical note: "Professor Galal Hassan Galal-Edeen is currently a Professor of Computer Science at the American University in Cairo, where he is on secondment from Cairo University, Egypt. His main technical interests are in the areas of methodologies for software and information systems design, architectures and architecting. Prof.Galal-Edeen has consulted and trained widely in the areas information systems, software engineering, usability evaluation and innovation for clients in the UK, USA, the Middle East and Egypt. He holds a BSc in Management Sciences (Computing and Information Systems stream) with magna cum laude, from the Sadat Academy for Management Sciences, Cairo. He also holds a Master’s degree in Systems Analysis and Design, from the City University, London, a PhD in Information Systems Engineering from Brunel University, UK, a BA in Architecture from the University of Greenwich, UK and finally a Master in Advanced Architecture Studies from the Bartlett Graduate School, the University of London. He is a full professional member of the British Computer Society and the ACM. He can be corresponded with by email at: Galal@acm.org"

Author 3:

Prof. Dr. A.M. Riad
Dean of the faculty of computers & Information,
Mansoura University,
Mansoura, Egypt.
amriad2000@mans.edu.eg

Biographical note: "Professor Alaa El-Din Mohammed Riad is currently the Dean of the faculty of computers and information, Mansoura University in Egypt where he is a professor of information systems. His main technical interests are in the areas of information systems and software engineering. He has offered consultation services for many Commercial and Industrial Institutions, besides supervising more than 22 MSc researches, and four PhD researches.