IT University
of Copenhagen

Architecture-Level Evolvability Assessment: Assessing Sustainability of Software Product Evolution

Hataichanok Unphon

IT University Technical Report Series TR-2010-128

ISSN 1600-6100 May 2010

ISSN 1600-6100

ISBN 9788779492196

Telephone:	+45 72 18 50 00
Telefax:	+45 72 18 50 01
Web:	www.itu.dk

Architecture-Level Evolvability Assessment: Assessing Sustainability of Software Product Evolution

Hataichanok Unphon

IT University of Copenhagen
DK-2300, Copenhagen S, Denmark
unphon@itu.dk

Abstract. This paper proposes a comprehensive architecture assessment method, a so-called Architecture-Level Evolvability Assessment (ALEA). The ALEA method aims at assessing how well the current architecture of software products is able to accommodate future uses and business contexts without jeopardizing the continuous software development. The ALEA method offers not only to broaden prospects of architectural changes, but also to assess the impact of changes on sustainability. In order to assess the sustainability, the ALEA method employs an evolvability framework consisting of sufficient contexts to propagate the effects of the architectural changes. The key element of the ALEA method is the involvement of a 'walking architecture' — a person or a group of people who carries most if not all the architectural knowledge and makes design decisions — throughout the assessment period. Based on empirical evidence of implementation and validation of the ALEA method (on a case study), the ALEA method is applicable for software product line development and agile software development approaches.

Keywords: software architecture analysis method, qualitative empirical research

1 Introduction

The success of many IT-related businesses is critically dependent on software products. Businesses need to be increasingly flexible and responsive to changes in the marketplace, and to develop and market a new products and services in a timely manner. In order to meet new business opportunities and accommodate for rapid modification and enhancement, software needs to be as flexible as possible. When the software evolves, the initial program structure is changed. Without being aware of the impact of changes, the initial program structure is often corrupted by a series of changes over time, which leads to dead-end evolution. Moreover, the software tends to get disconnected from its context of use. This raises the question of how to ensure that software is able to bring immediate success and support long-term evolution, or, in other words, how to evaluate the technical adaptability of the software as well as its sustainability with respect to the use and business contexts as well as the development organization and practice.

The main contribution of this paper is to propose a method, so called Architecture-Level Evolvability Assessment (ALEA). Note in this context that the terms architecture analysis, architecture assessment, architecture evaluation and architecture review are used interchangeably throughout this paper. ALEA has been developed, implemented, and validated during a cooperative project with a company developing product line architecture for surface water modelling systems. Due to the development environment, user practices, and business vision, the architecture needs to allow for intensive tailoring and continuous development. The ALEA method provides the necessary elements for analyzing the architecture. A framework proposed in [27] successfully complements a keystone of ALEA from a socio-technical perspective. Compared to existing methods of architecture evaluation, the ALEA method puts more emphasis on sustainable development, meaning balancing the needs of the short and the long term. The artefacts examined in the ALEA method cover both design architecture and code architecture. ALEA is a light-weight architecture evaluation method which can be integrated into an agile development cycle. However, there are some challenges to ALEA which should be further refined.

This paper is outlined as follows. Section 2 presents the case description. Section 3 explains the research method. Section 4 introduces terms and definitions. Section 5 describes the ALEA method, shows an implementation of the ALEA method on the case study, and suggests improvements to ALEA. Section 7 is discussion and related work. Section 8 draws conclusion and looks at future work.

2 Case Description

DHI Water Environment Health (DHI) is a pioneering organisation that develops software applications for hydraulic modelling [1]. In 1972, System 11 and System 21 were two of the first computational modelling systems developed at DHI to simulate water flow patterns with the help of one-dimensional and two-dimensional models. A three-dimensional simulation was developed in the 1980s. Originally, the organisation focused on hydraulic research, not on software engineering. Software development and software maintenance were challenges only on a small scale. All simulation programs were built in a similar way, i.e., an engine implementing differential equations changes the data in a set up model for one time step per simulation loop. In the late 1980s, DHI released the MIKE 11 and the MOUSE software products. Both products originated from System 11 following the requests of different usages, i.e. open channels and pipe networks. MIKE 11 and MOUSE are standalone Windows-based applications. The main users of these products are consultants who do simulations of hydraulic conditions, i.e. water level and flow, and analyse the hydrological effects of environmental change. Due to different market needs, ownership was split into different consultancy departments and in the last decades MIKE 11 and MOUSE have been developed and maintained in parallel. Released in 2005, MIKE URBAN followed requests to have a more complete and integrated modelling framework for both water supply and wastewater systems.

Through decades of successful use and development, the requirements of the software have evolved as well. In particular, the software is used in a more general setting, e.g. scheduled forecasts. The company was faced with the challenge of identifying and developing a kernel for data handling, simulation setup, and graphical interaction with simulations and their results. The first re-engineering project started with the MIKE 11 engine in 2006. Later, the MOUSE engine was merged into the MIKE 11 re-engineering project. Meanwhile, the organisation was changing. DHI set up a software product department in order to strengthen the software development process and the design. The software product department has taken development activities and ownership of DHI's software products. As a consequence, the department decided to re-engineer the core computational parts of some of the one-dimensional simulation software products, i.e. MIKE 11, MOUSE and MIKE URBAN, in a project called MIKE 1D. The project is estimated for 360 man weeks of implementation.

Lately, the software product department officially promoted another project called the Decision Support System (DSS) Platform. The DSS Platform affords end users the leverage to customise ongoing water simulation using historical, current, and predictive data. The DSS Platform usually uses data that has already been gathered into persistent storage and occasionally works from operational data. The simulation it builds on has to be set up as well by developing the model of the water system.

3 Research Method

This work was conducted as qualitative empirical research aiming at providing a well-grounded and rich detailed description based of a case study rather than superficial measurement. The research cooperation with DHI addressed the introduction of product line architecture into product development. The basis for the research described here is the fieldwork which I have been involved in for two and a half years. I wrote a research diary documenting daily observations, interviews, and meetings. As a field worker, I was expected not only to observe, but also to influence the projects in which I participated. The research was designed as action research by following the cooperative method development approach (CMD) [12]. The research activities are summarised in Table 1. Due to a lengthy period of cooperation, research activities are chronologically divided into three cycles: 1.) MIKE 11 re-engineering project, 2.) merging of MIKE 11 and MOUSE re-engineering project, and 3.) MIKE 1D project. Note that the research activities in the second cycle were collected when the third cycle was ongoing. Each cycle consists of three phases, i.e., participant observation, deliberating change, and evaluation. Most empirical evidence presented in this paper is obtained from the last cycle.

4 Terms and Definitions

This section introduces terms and definitions used in this paper. Subsection 4.1 defines evolvability and evolvability framework. Subsection 4.2 explains socially embedded systems. Note, again, that the terms system and software are used interchangeably. Subsection 4.3 presents a 'walking architecture'.

Table 1. A summary of research activities

Cycle / Phase	**1.) MIKE 11 re-engineering project (Aug. – Nov. 06)**	**2.) Merging of MIKE 11 and MOUSE engines re-engineering project (Dec. 06 – Oct. 07)**	**3.) MIKE 1D project (Feb. 07 – Mar. 09)**
Participant observation	- Study functionalities and code architecture of MIKE 11 and MOUSE engines. - Compare between MIKE 11 and MOUSE engine source code. - Interview DHI staff members. - Found a striking similarity in the source code between MIKE 11 and MOUSE engines.	- Review of architectural documentation and online user references systems used at DHI. - Observe development practices and technical infrastructure of MIKE 11 and MOUSE engines. - Review off-the-shelf documentation generators. - Interview developers and internal users of MIKE 11 and MOUSE engines on how they can use the architecture document.	- Review off-the-shelf static code analysis tools. - Analyse MIKE 1D source code using the reviewed tools and identify the relative complexity of its components. - Compare the analysis with the previous cycle projects. - Join MIKE 1D project weekly meetings. - Interview MIKE 1D team members on the idea of assessing the architecture and how they can use of the architecture as an aspect of software development.
Deliberating change	- Present a poster highlighting identical code parts between MIKE 11 and MOUSE engines. - Present a talk on software architecture and product line architecture. - Participate in a subproject on developing data access module architecture for the MIKE 11 re-engineering project.	- Propose a layered architecture to represent architectural knowledge. - Compare documentation generators and recommend a suitable one. - Update architecture documentation. - Create a prototype of an online architectural knowledge system.	- Conduct a workshop on architecture discovery with MIKE 1D team members. - Introduce the basic idea of architectural conformity checking. - Recommend suitable static code analysis tools. - Present the “good” and “bad” parts of the source code from the static code analysis tools. - Present an empirical study on architecture evaluation in industrial practice, the concept of software evolvability, and evolvability framework. - Propose Architecture-Level Evolvability Assessment (ALEA). - Organise a workshop on MIKE 1D and DSS compatibility.
Evaluation	- Evaluate the flexibility of the data access module by looking at different change scenarios at DHI and their implications in terms of implementation efforts. - Found that organisation of software development influenced product line architecture development [23].	- Found that architectural knowledge was more visible in the discussion than in the document. - Found that the prototype of the online architectural knowledge system has been set up and used internally.	- Found that architectural analysis tools and techniques embedded in daily routine were welcome by the development team. - Found that the development team uses “build hierarchy” to check the compliance of their source code against the architecture’s structure when they build the software [24]. - Validate ALEA and evolvability framework with MIKE 1D team members.

4.1 Evolvability and Evolvability Framework

Belady and Lehman [7] first introduced and used the term *evolution* as ‘a sequence of changes to the system over its lifetime which encompasses both development and maintenance’. In today’s competitive software market, it would be too restrictive to limit evolvability to maintenance issues only. The growth dynamics of a system depend highly on the business context. To increase market share, it may be vital to add new features. Yet, a system that is used will be changed [19]. Unphon et al. [27] have further defined evolvability as ‘the adaptability of software in order to serve the needs of use and business contexts over time reflecting on its architecture’. Architecture represents a common abstraction of a system that many of the system’s stakeholders can use as a basis for mutual understanding, negotiation, consensus, and communication [6]. Architecture and other contexts around it must be adapted to accommodate the needs of use and business contexts. However, it would be somewhat misleading if architecture adaptation jeopardises other contexts that have brought a success to software. In this paper, evolvability is further refined into *technical adaptability as well as sustainability with respect to the use and business contexts as well as the development organisation and development practice*.

The evolvability framework proposed in [27] is used for reviewing the effects of architectural changes. The proposed framework presents interaction between architecture and the six contextual dimensions, i.e., business, use, software engineering organisation, software engineering practice, technical infrastructure, and technical selection. Each contextual dimension is defined and illustrated as follows:

Business context is the context or environment to which the system belongs. For example, DHI software is a commercial software product and sold as licensed.

Use context relates the system to the work practices of the intended users. For example, hydraulic engineers use DHI software for water flow modelling, wave simulation, or flood forecasting.

Software engineering organisation is the organisational context in which the software development is carried out. For example, DHI software is developed in Denmark, the Czech Republic, and China. The DHI software product department employs the Microsoft Solutions Framework (MSF) team model [3]. MIKE 11 and MIKE URBAN software products were developed by different departments.

Software engineering practice refers to the work practices of the system developers. For example, the development process at DHI is a mixture between iterative/incremental processes and agile methods. The core computational simulation developers are educated in hydraulic engineering, but the graphic user interface (GUI) developers are computer scientists. Most if not all MIKE 1D developers are highly educated in water and environmental engineering, not software engineering.

Technical infrastructure lists the hardware and basic software assets backing the system, focusing on the design as it is now. For example, MIKE 1D components are implemented in the C# programming language. The MIKE 1D project has unit test, nightly build, and build hierarchy as development infrastructure. DHI software only supports the Microsoft operating system.

Technical selection is part of a suggested design and relevant to design implementation. It needs to be seen in the context of existing and planned systems, as well as in the context of other systems that are part of the same design. For example, a common data access module handles setup data of MIKE 11 and MOUSE.

Others have used the notion of context or contextual factors before. Kensing [17] proposed a conceptual framework that IT designers should be aware of when they design applications for a specific organisation. The framework addresses 1.) project context, separating into design and implementation; 2.) use context, dealing with work practice and strategy; and 3.) technical context, interacting with system and platform contexts. Kensing does not apply the framework to concrete design proposals. Dittrich and Lindeberg [11] developed Kensing's framework further by mapping out contextual factors in order to understand the suitability of a less technically advanced design for a specific industrial setting. This work further develops this framework to support architecture-based analysis when planning to evolve software products.

4.2 Socially Embedded Systems

Socially embedded systems [27] are 'systems that can be modelled intensively according to the environment and practices of its end users'. ERP systems, e-government applications, virtual office software, and decision support systems are examples of socially embedded systems. Design decisions of the socially embedded systems underline the importance of human interaction with (and cooperation via) the software in terms of societal activities. According to Lehman [20], an *E*mbedded program (*E*-program) is a part of the world which it models. This implies a constant pressure for change. The usability of the system is the main concern of *E*-programs. Close cooperation between end users, people working with the systems on a daily basis, and developers throughout the entire development process is strongly recommended for capturing the contexts and qualities of use that cannot be fully anticipated in the initial phase. In use-oriented design, Participatory Design (PD) is regarded as a method for improving usability [18]. Socially embedded systems often allow users to tailor the software to specific needs. Examples of end user tailoring categories are customisation, composition, expansion, and extension [13].

Socially embedded systems also evolve over time, as do technically embedded systems[1]. But evolving socially embedded systems is not just constrained by interfaces to hardware or the mechanical specification; it is also constrained by use and business contexts as well as development practice and development organisation. Floyd et al. [14] have already emphasised bringing the social contexts along with the technical in the essence of software development. However, they did not explicitly explain how to do that with respect to software architecture practice. Evolving socially embedded systems, one has to balance social and technical requirements while maintaining a consistent pace for supporting short and long term requirements. This work suggests a systematic method to mediate those requirements (Section 5).

[1] The concept of technical embedded systems, or embedded systems as defined in [28], refers to any computer that is a component in a larger system and that relies on its own microprocessor, e.g., telephone switches, hybrid cars, and printers.

4.3 Walking Architecture

The concept of ‘walking architecture’ was coined in [26] as a representation of architectural knowledge, that is to say, architecture is alive with a walking architecture. The walking architecture is a key person, or a number of key persons, who maintain and update the structure of the software, and are involved in discussions of change motivated in the development, or by new requirements, and who introduce newcomers to the structure of the software. The role of the walking architecture can be seen in that of the chief architect. However, not all companies have a chief architect. All product development teams have a person or group of people acting in that role, even though their title might be different, like chief technology officer (CTO), senior developer, product manage, project leader, or system architect.

Architectural issues arise from inside as well as outside the development team, they cover technical and social aspects of software development, and require domain, as well as software engineering expertise. In order to solve the architectural issues, the chief architect interacts with technical and business people, establishes tools and practices, and recruits or train team members for that expertise, etc. Because architecturing is not only a matter of technical design, but also of juggling the social contexts of software development, which makes it almost unable to automate. Whatever methods and tools software engineering research proposes, they need to be aligned with the practices of knowledge-sharing by, and with, the walking architecture.

5 Architecture-Level Evolvability Assessment

Socially embedded systems evolve to support uses and business needs, which may not exist when the systems were designed or developed initially. Walking architecture needs to envision architecture for an intermediate success while being mindful of the change effects for the long-term evolution. Architecture-Level Evolvability Assessment (ALEA) is proposed as a tool for walking architecture to evaluate adaptability and sustainability of the architecture. ALEA answers not only the question of ‘how the envisioned architecture will look?’ or ‘how that affects quality factors?’, but also the question of ‘how the architecture can be evolved in a sustainable manner?’. ALEA promotes interaction between business and technical stakeholders of the systems, e.g., end users, developers, and, more importantly, walking architecture. It is significant that the walking architecture involves and participates throughout the assessment period. Subsection 5.1 presented the concise ALEA method description. Subsection 5.2 shows the implementation of the ALEA on the DHI case. Subsection 5.3 suggested improvements to the ALEA.

5.1 The Description of ALEA Method

The ALEA method is divided into 3 stages: *elicitation*, *assessment*, and *reporting*. Each stage is elaborated below.

The first stage, *elicitation*, aims to prepare necessary elements for the assessment stage. The elements are *existing architecture*, *quality factors*, *assessment goal*, and *assessment items*. *Existing architecture* can be elicited from architectural documentation or walking architecture. *Quality factors* [22] represent behavioural characteristics of a system: correctness, reliability, flexibility, testability, maintainability and reusability. *Assessment goal* describes a purpose of the assessment. If the goal is not specifically identified, it will lead to involving unnecessary stakeholders and cause difficulties in identifying assessment items. *Assessment items* can be seen as new requirements, use scenarios [9], change issues, etc. Each item should come from the stakeholders who tell what is expected to happen rather than assuming change or predicting use. If the items are identified, they must, subsequently, be prioritised in such a way that high-priority items are assessed before low-priority items.

The second stage, *assessment*, aims at reviewing both *architecture adaptation* and *sustainability assessment* of the adaptation for each assessment item. *Architecture adaptation* includes: evaluating existing architecture with respect to assessment items, envisioning architecture, and assessing the envisioned architecture with respect to relevant quality factors. The envisioned architecture can be seen as the existing architecture with new components added, adding new interfaces to existing components, changing existing components, or changing existing interfaces. The analysis of the envisioned architecture with respect to quality factors will provide a solid basis for making an objection decision in case of design trade-offs. *Sustainability assessment* addresses the envisioned architecture with respect to the evolvability framework. If need be, some contexts might be adapted to support the envisioned architecture or the assessment item, or the envisioned architecture has to be refined. Because the envisioned architecture will 'inhabit' the same context as the existing architecture, it is vital to be mindful of what the root context of an assessment item is, which contexts could potentially be effected, and how they could be adapted.

The third stage, *reporting*, not only aims at documenting the whole assessment, but also entails a mechanism of follow-up. It is absolutely essential that all findings in the architecture evaluation are backed by evidence. The mechanism of follow-up makes the design decisions visible to responsible stakeholders. The mechanism is not to make decisions immediately, but to broaden the perspective and inform the basis on which decisions are eventually made. For example, if the stakeholders are aware of what they gain from a possible solution, will they favour that solution or will they find another solution? If there are multiple solutions for the same assessment item, the stakeholders will see which quality factors or evolvability contexts are affected by each solution.

5.2 Implementing ALEA at the DHI

This subsection presents empirical evidence in which ALEA was first implemented at the DHI. When the MIKE 1D project was well underway, I proposed ALEA to the MIKE 1D team in order to analyse whether the MIKE 1D ongoing development aligned with the DHI business vision. The idea was welcomed by the MIKE 1D team members. The members suggested a number of assessment goals, one of which was that the MIKE 1D and DSS Platform compatibility was carried out because it could be assessed between two in-house projects, and without hiring any external hydraulic and environmental consultants. When we prompted a workshop on MIKE 1D and DSS Platform compatibility, we elicited quality factors of the MIKE 1D architecture, but the current picture of MIKE 1D architecture was given from the main developer. We divided the workshop into two parts: the first part aimed at eliciting assessment items from the DSS Platform project; the second part aimed at analysing the architecture with respect to the assessment part of the ALEA method.

Due to limited funding, we arranged the first part of our workshop as a lunch meeting[2]. Participants were not only team members of the MIKE 1D project and the DSS Platform project, but also all the interested stakeholders from the DHI consultancy departments. I also invited an architecture expert to participate in this workshop. The workshop started with the MIKE 1D main developer presenting the current MIKE 1D design architecture, and ideas of how the DSS Platform could work with the MIKE 1D architecture. Then, the workshop participants gave direct input or assessment items to the MIKE 1D team, which were discussed in the second part of the workshop. In the first part, there were ten participants. An architecture expert and I observed and recorded the discussion. Total time spent on the first part was one hour.

In the second part of the workshop, there were five participants: three MIKE 1D team members, an architecture expert and myself. I was a modulator for the second part. We went through the given input or assessment items from the first part, and discussed by following the ALEA method. There were two assessment items discussed in the second part. In the end, we reflected on the ALEA method and the evolvability framework. Total time spent on the second part was two hours. After the workshop, I documented the discussion in a report, while the MIKE 1D team members followed up with the issues raised in the discussions.

Fig. 1 shows the workshop report. The report has two parts: PART I captures goal, quality factors, existing architecture, and assessment items; PART II captures a set of assessment items along with architecture discussions, envisioned architecture, related quality factors, sustainability discussions, conclusions and action plans.

The goal of this workshop is to assess the MIKE 1D architecture and DSS Platform compatibility. MIKE 1D quality factors are maintainability, usability, and integrity. Note that unrelated quality factors are omitted here. The MIKE 1D existing architecture is shown in Fig. 1. The MIKE 1D design architecture consists of four layers: *Application*, *Controller*, *Data Access*, and *Utilities*. Each layer is comprised of a number of components. Note that the components of each layer are not presented in this paper. The solid arrows show the ‘use’ relationship of components, layers, and products. For example, a component in *Application* layer uses *MU Proxy* components. The design decision to have a *Data Access* layer is an example of how the MIKE 1D architecture promotes its quality factors. Through the *Data Access* layer, a component in the *Controller* layer, *MIKE View* product, *MU Proxy* component, and, possibly, the

[2] At DHI, the lunch meeting is considered as an internal meeting in which the host shall not spend extra budget for any participants because it is considered as part of common contribution. Thus, holding such a meeting means economics collaboration between different in-house projects or departments.

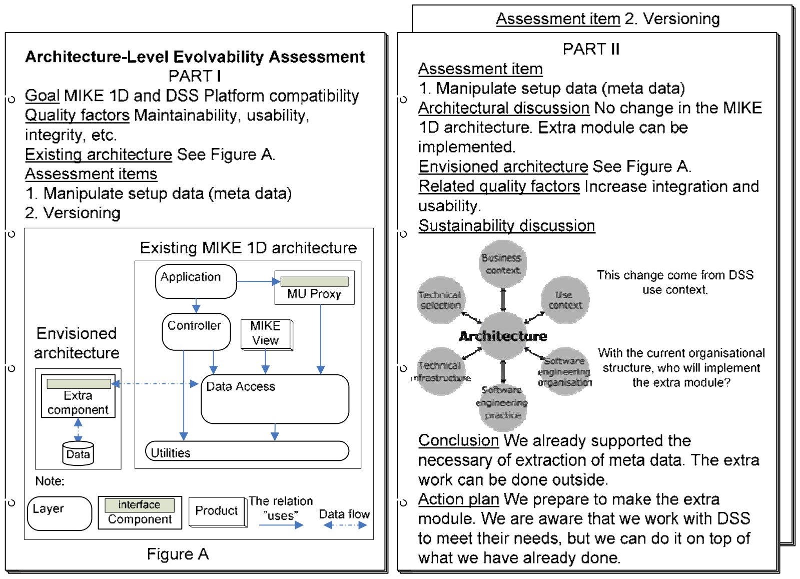
Architecture-Level Evolvability Assessment
PART I
Goal MIKE 1D and DSS Platform compatibility
Quality factors Maintainability, usability, integrity, etc.
Existing architecture See Figure A.
Assessment items
1. Manipulate setup data (meta data)
2. Versioning

Figure A

Assessment item 2. Versioning

PART II
Assessment item
1. Manipulate setup data (meta data)
Architectural discussion No change in the MIKE 1D architecture. Extra module can be implemented.
Envisioned architecture See Figure A.
Related quality factors Increase integration and usability.
Sustainability discussion

This change come from DSS use context.

With the current organisational structure, who will implement the extra module?

Conclusion We already supported the necessary of extraction of meta data. The extra work can be done outside.
Action plan We prepare to make the extra module. We are aware that we work with DSS to meet their needs, but we can do it on top of what we have already done.

Fig. 1. A report summarising the MIKE 1D and DSS Platform compatibility workshop

third-party users can handle setup of a water model in a straightforward manner without accessing any persistent storage directly. If a user wants to simulate a specific water model from a specific file, the user creates a specific file reader for that file and populates a component in the *Data Access* layer. Then, the user can perform a simulation without changing any component in the other layers.

A member of the DSS Platform project, who works with an operational flow forecasting system, addressed new requirements relating to the setup of data manipulation. One functional requirement of the DSS platform is to handle 'what if' situations. End users are free to change the setup data, e.g., water inflow during a simulation cycle. To forecast the next simulation, the DSS Platform needs to know 1.) which setup data to use; 2.) the results from previous simulations; 3.) manual input or calibrated setup data, e.g. when starting the forecast after a computing failure.

In order to support this requirement, MIKE 1D team members suggested that one should create a 'wrapper' around the *Data Access* layer. The wrapper is a minimal interface component that gives high-level functionalities to the *Data Access* layer. The wrapper would get data from the previous simulation, the calibrated setup data, or another persistent storage (a database or a result file from another simulation system). In this way, the wrapper would require metadata for transforming data appropriately. To create such a wrapper, nothing would change in the existing MIKE 1D design architecture. The architecture already supported the necessary extraction of the metadata. But an envisioned architecture would add the wrapper beside the *Data Access* layer, which can be done outside the MIKE 1D architecture. Fig. 1 shows the envisioned architecture locating on the left side of the existing MIKE 1D architecture.

In the sustainability assessment, the MIKE 1D team members saw that the manipulation of setup data originated from the context of the DSS Platform use. The envisioned architecture pointed out the challenge of software engineering organisation and business context, i.e., “*With the current organisational structure, who should implement the wrapper? The MIKE 1D team, the DSS Platform team, or someone else?*” and “*Will the wrapper be one of DHI’s saleable components? If so, who will take the lead on that?*”. Both the MIKE 1D and the DSS Platform team members are potential candidates for developing a wrapper. However, both teams were no able to make the decision. Therefore the follow-up plan is to report this to the head of the development group.

5.3 Evaluation of and Improvements to the ALEA method

After experiencing ALEA, the MIKE 1D team members approved of the structure, the transparent decision-making process and the trade-off analysis. Before the ALEA method was implemented, the MIKE 1D team members analysed the architecture informally at the whiteboard. One of the members reported that “*When we do it (architecture assessment on the whiteboard), I think we get only half of the quality factors and half of the contexts (of the evolvability framework) because it is not structured. By getting this structure, we are able to make a more sound decision about what to do.*” Apart from that, ALEA endorses product-line architecture. One difficulty at DHI was thinking in terms of product line architecture. Often, a developer just came up with an idea to solve a problem. Due to this pragmatic decision, the developer often added his solution directly onto the source code without considering whether or not it could be usable for future projects. Through the assessment stage of the ALEA method, the developer is encouraged to consider not only the consequences of change on his own project, but also sustainable solutions for other projects. After the first implementation of the ALEA method, the MIKE 1D team members gradually learned the terms associated with the method as well as the connections between the architecture and its relevant contexts. The MIKE 1D team members planned to assess their architecture at the beginning of each milestone. “*It would be a good tool for a project leader,*” one of the MIKE 1D team members suggested. This feedback points out how to embed architecture assessment in the development process. Moreover, the ALEA method is aligned with the principles of Agile Manifesto [2]. MIKE 1D team members and other developers at DHI employ those principles into their software engineering practice. By sharing the same principles, ALEA can be seamlessly integrated into the work practice at DHI.

Positive feedback by MIKE 1D team members on the evolvability framework was comprehensiveness, illustrations, and visualisation. The evolvability framework was used in a sustainability discussion, as shown in Fig. 1. I found that the team members can visualise the consequence and propagate the effects of an envisioned architecture in a short period of time. What impressed me the most was the accuracy with which team members were able to predict the consequences of suggested changes to the architecture. When the evolvability framework was introduced, one team member questioned 1.) the difference between technical selection and technical infrastructure and 2.) how the framework relates to stakeholders. The answer to the first question is defined and illustrated in the Subsection 4.2. The answer to the second question is that the stakeholders belong to contextual dimensions. For example, based on an assessment item of the MIKE 1D and the DSS Platform compatibility, the DSS Platform team represents the use context.

On the other hand, the MIKE 1D team member had difficulties with the term 'evolvability' because it is a rather abstract and difficult concept to grasp. Some of them wondered why it was discussed. Eliciting quality factors was another challenge. The team members were not familiar with the term 'quality factors'. What this shows is that the ALEA method should be further refined in order to be comprehensibly conveyed in a novice environment. For example, addressing the importance of ALEA and simplifying the terms for the ease of communication.

During the first part of the workshop, one of the DSS Platform team members raised a well-known issue which the participants discussed in the first part of the workshop. When the MIKE 1D team discussed in the second part of the workshop, one of them complained: "*I don't know how they do that in practice actually.*" The elicitation of an assessment item should have been either described with extremely precise instructions or used the participatory design approach. However, this addresses the organisational relevance in architectural discussions, apart from the business or technical issues.

6 Discussion & Related Work

A summary of the ALEA method based on FOCSAAM [4] is presented in Table 2. Note that evolvability is the central 'quality attribute' of ALEA, while 'quality factors' are the elicited quality attributes of the existing architecture. The trade-off analysis of the quality factors are considered as part of the evolvability. E.g., in the continuous development of a system, the quality factors of the new version can wary tremendously from those of the old version. The artefacts examined in the ALEA method cover both design and code architecture.

ALEA provides essential features, as does most of the well-established architecture evaluation methods presented in [5], but on top of that, ALEA applies the evolvability framework as its tool to facilitate the sustainability assessment. Contextual dimensions of the evolvability framework have been discussed extensively in the other architecture evaluation methods, such as Architecture-Level Modifiability Analysis (ALMA) [8], Architecture Trade-off Analysis Method (ATAM) [15], and Architecture Reviews for Intermediate Designs (ARID) [10]. But, the dimensions are not structured for the analysis in these methods. The main difference between ALEA and ALMA — the evaluation method which is resembles ALEA the most — is that ALEA considers not only the modifiability, but also the sustainability with respect to an explicit evolvability framework.

ALEA provides 'effectiveness and usability' [16] as the other well-established architecture evaluation methods, according to the facts that 1.) the concept of architecture was 'concretised' or fully integrated in the development environment before the implementation of ALEA; 2.) ALEA is designed based on industrial practice; and 3.) ALEA gives precedence to socio-technical perspective. Moreover, ALEA is a light-weight architecture assessment method. ALEA requires half a day, excluding preparation and preliminaries, rather than three full days spent on ATAM, or one to two days spent on ARID. In this way, architecture evaluation can be performed more frequently.

Besides architecture evaluation, the evolvability framework is a decision-making tool to find a sustainable way to improve software product line engineering. A BAPO-based framework (Business-Architecture-Process-Organisation) introduced in the Family Evaluation Framework (FEF) [21] is a model similar to the evolvability framework. Mapping between the BAPO model and the evolvability framework can be seen as: B-Business context, A-Architecture, P-Software engineering practice, and O-Software engineering organisation. The BAPO model identifies interrelationships among four independent software development concerns; applying changes in one concern induces changes in the other three concerns. Each has its own profile scale for benchmarking. However, an action that improves one of the profile scales may lead to a reduction in the values of the other scales. The profile scales in the BAPO model serves researchers for benchmarking the organisation against others rather than serving practitioners finding an optimum profile for their own organisation, because top marks for each dimension may not be optimal from a business and economic perspective. The BAPO model has been developed, mostly if not exclusively, from technically embedded systems. Therefore, the use context was not explicitly mentioned, as opposed to the evolvability framework, which was developed from socially embedded systems. The evolvability framework does not offer any scale for each contextual dimension, as explicitly stated in the BAPO model. But, the evolvability framework introduced in the ALEA method, essentially, helps practitioners be aware of changes in one context that induce changes in the other contexts, and involving 'the right people'.

Table 2. A summary of ALEA based on FOCSAAM

	FOCSAAM	**ALEA**
Component	**Elements**	**Brief explanation**
Context	Software architecture definition	Structure(s) of system which comprise software elements, the externally visible properties of those elements, and the relationships among them [6].
	Specific goal	Change impact analysis
	Quality attributes	Evolvability and other elicited quality factors
	Applicable stage	All stages of software life cycle
	Input & Output	Embedded in method description
	Application domain	Socially embedded systems
Stakeholders	Benefits	Continuous quality check and specific benefit according to the assessment goal
	Involved Stakeholders	'Walking architecture' and selected stakeholders depending on the assessment item
	Process support	Embedded in method description, participatory design (recommended)
	Socio-technical issues	Embedded in method description
	Required resources	Funding, person hours spent for elicitation, assessment and reporting
Contents	Method's activities	Three main stages: elicitation, assessment and reporting
	Software architecture description	Design architecture and code architecture
	Evaluation approaches	Based on change requirements, an expert evaluation
	Tool support	Evolvability framework
Reliability	Maturity of method	Developing and continuous validation
	Method's validation	Case study

In practice, architecture is often evaluated on an ad-hoc basis. One ongoing empirical study [25], which I took part in analysing interviews, reveals that non-technical issues (e.g., process, people, organisation, communication and finance) oftentimes is the cause of various architectural problems presenting a challenge to the architecture evaluation. Even if this finding cannot not be generally applied to all organisations, it provides a guideline for the deployment and implementation of ALEA. As a result, ALEA was welcomed at DHI because of it being simple, light-weight, and a cost-effective architecture evaluation method, which can be used by practitioners. The ALEA method promotes face-to-face conversation rather than documentation. ALEA requires the involvement of a 'walking architecture' throughout the assessment period. ALEA embeds the assessment of non-technical or socio-technical issues into its method in contrast to the other architecture evaluation methods.

However, there are questions of implementing ALEA into different contexts such as 'what if 50 people need to be involved throughout the whole assessment period?', 'how about open-source software?' or 'how about safety critical systems where the detail designs are documented?' Proportionately, a higher number of participants reduces the usability of the ALEA method. The ALEA method aims to promote discussions within small teams (2-5 people) situated at the same physical location. Although open-source software is developed at different locations, it sometimes has a focused development session lasting anywhere from a day to a week. When developers participate in such sessions, they, inevitably, prioritise the requirements, design, code, test, and, at the end of a day, release a new version of the software. In this way, ALEA can be integrated into the focused development session. With the walking architecture involvement, ALEA can be used even if there is a lack of explicit design documentation or no updated architectural documentation. The presence of detailed design documentation does not reduce the effectiveness of the ALEA method.

7 Conclusion and Future Work

This paper proposes a method called Architecture-Level Evolvability Assessment (ALEA). ALEA aims at evaluating how well the current architecture of a software product can accommodate future use and business contexts. What distinguishes ALEA from other architecture assessment methods is the sustainability assessment. The ALEA method consists of three stages: elicitation, assessment, and reporting. The elicitation stage aims at eliciting existing architecture quality factors, identifying an assessment goal, and identifying and prioritising assessment items. The assessment stage aims at evaluating architecture adaptation and sustainability for each assessment item. In the sustainability assessment, ALEA employs an evolvability framework in order to propagate the effects of changes. The reporting stage aims at documenting the whole assessment and follow-up the assessment by communicating it to 'the right people'. Comparing to other well-established architecture evaluation methods, ALEA gives precedence to socio-technical perspectives. The main requirement of employing

ALEA is the involvement of a ‘walking architecture’ throughout the whole assessment period.

The ALEA method and the evolvability framework were deployed, implemented and validated in a cooperative project with DHI. The case study shows that ALEA is applicable for evaluating product line architecture and able to integrate seamlessly with agile software development. After the first validation, there were some challenges to the ALEA method, which should be further refined and addressed. In order to support the evaluation more pragmatically, ALEA should 1.) concretise into a development cycle, 2.) integrate with the participatory design (PD) in the elicitation of assessment items, and 3.) simplify the terms (evolvability, quality factors, etc.) for the ease of communication. In term of maturity, ALEA should be performed with a wide range of assessment goals and items, and tried out with different organisations. Future works are expected to show whether and how 1.) ALEA can be applied in different context, 2.) comprehensible for a novice, and 3.) applicable to continuous development of software product lines.

Acknowledgments. This work was done while I was working at IT University of Copenhagen and DHI Water Environment Health.

References

1 DHI Water Environment Health, http://www.dhigroup.com
2 Manifesto for Agile Software Development, http://www.agilemanifesto.org
3 MSF Team Model v.3.1, Microsoft Solutions Framework (MSF) Team Model, http://-www.microsoft.com/downloads/details.aspx?familyid=C54114A3-7CC6-4FA7-AB09-2083C768E9AB&displaylang=en
4 Ali Babar, M., Kitchenham, B.: Assessment of a Framework for Comparing Software Architecture Analysis Methods. In: Kitchenham, B., Brereton, P., Turner, M. (eds.) Proceedings 11th International Conference on Evaluation and Assessment in Software Engineering (EASE). Keele University, UK (2 - 3 April 2007)
5 Ali Babar, M., Zhu, L., Jeffery, R.: A Framework for Classifying and Comparing Software Architecture Evaluation Methods. In: Proceedings Australian Software Engineering Conference (ASWEC). pp. 309–318 (2004)
6 Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, 2nd edn. (2003)
7 Belady, L., Lehman, M.: A Model of Large Program Development. IBM Systems Journal 15(1), 225–252 (1976)
8 Bengtsson, P., Lassing, N., Bosch, J., van Vliet, H.: Architecture-Level Modifiability Analysis (ALMA). J. Syst. Softw. 69(1-2), 129–147 (2004)
9 Carroll, J.M.: Making Use: Scenario-Based Design of Human-Computer Interactions. The MIT Press, 1st edn. (2000)
10 Clements, P.C.: Active Reviews for Intermediate Designs. Tech. Rep. CMU/SEI-2000-TN-009, SEI, Carnegie Mellon University (2000)
11 Dittrich, Y., Lindeberg, O.: Designing for Changing Work and Business Practices. In: Adaptive Evolutionary Information Systems, pp. 152–171. IGI Publishing, USA (2003)
12 Dittrich, Y., Rönkkö, K., Eriksson, J., Hansson, C., Lindeberg, O.: Cooperative Method Development. Empirical Softw. Engg. 13(3), 231–260 (2008)

13 Eriksson, J.: Supporting the Cooperative Design Process of End-User Tailoring. Ph.D. thesis, Department of Interaction and System Design, School of Engineering, Blekinge Institute of Technology, Sweden (2008)

14 Floyd, C., Keil-Slawik, R., Budde, R., Zullighoven, H. (eds.): Software Development and Reality Construction. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1992), illustrator-Weiler-Kuhn, C.

15 Kazman, R., Klein, M., Clements, P.: ATAM: Method for Architecture Evaluation. Tech. Rep. CMU/SEI-2000-TR-004, ADA382629, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (2000), http://www.sei.cmu.edu/publications/documents/-00.reports/00tr004.html

16 Kazman, R., Bass, L., Klein, M., Lattanze, T., Northrop, L.: A Basis for Analyzing Software Architecture Analysis Methods. Software Quality Control 13(4), 329–355 (2005)

17 Kensing, F.: Participatory Design in a Commercial Context - a Conceptual Framework. Participatory Design Conference, New York, USA (2000)

18 Kensing, F., Blomberg, J.: Participatory Design: Issues and Concerns. Computer Supported Cooperative Work (CSCW) 7(3-4), 167–185 (September 1998)

19 Lehman, M.: On Understanding Law, Evolution, and Conservation in the Large-Program Life Cycle. J. Syst. Softw. 1(3), 213–231 (1980)

20 Lehman, M.: Programs, Life Cycles, and Laws of Software Evolution. Proceedings of the IEEE 68(9), 1060–1076 (Sept 1980)

21 van der Linden, F., Bosch, J., Kamsties, E., Känsälä, K., Obbink, H.: Software Product Family Evaluation. Software Product Lines 3154, 110–129 (2004)

22 McCall, J.A., Richards, P.K., Walters, G.F.: Factors in Software Quality. Tech. Rep. RADC-TR-77-369, U.S. Department of Commerce, Washington, DC (1977)

23 Unphon, H., Dittrich, Y.: Organisation Matters: How the Organisation of Software Development Influences the Development of Product Line Architecture. pp. 178–183. IASTED International Conference on Software Engineering, Innsbruck, Austria (2008)

24 Unphon, H.: Making Use of Architecture throughout the Software Life Cycle - How the Build Hierarchy can facilitate Product Line Development. In: SHARK '09: Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge. pp. 41–48. IEEE Computer Society, Washington, DC, USA (2009)

25 Unphon, H., Babar, M.A., Dittrich, Y.: Identifying and Understanding Software Architecture Evaluation Practices (2009), ITU Technical report (almost finished)

26 Unphon, H., Dittrich, Y.: Software Architecture Awareness in Software Product Evolution (2009), submitted to J. Syst. Softw. (under revision)

27 Unphon, H., Dittrich, Y., Hubaux, A.: Taking Care of Cooperation when Evolving Socially Embedded Systems: The PloneMeeting Case. In: CHASE '09: Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering. pp. 96–103. IEEE Computer Society, Washington, DC, USA (2009)

28 Wolf, W.: What Is Embedded Computing? Computer 35(1), 136–137 (Jan 2002)