Developing Architecture in Volatile Environments

Lessons Learned from a Biobank IT Infrastructure Project

Jarkko Hyysalo, Gavin Harper, Jaakko Sauvola, Anja Keskinarkaus, Ilkka Juuso, Miikka Salminen, Juha Partala
Faculty of Information Technology and Electrical Engineering
University of Oulu
Oulu, Finland
e-mail: jarkko.hyysalo@oulu.fi, gavin.harper@oulu.fi, jaakko.sauvola@oulu.fi, anjakes@ee.oulu.fi, ilkka.juuso@ee.oulu.fi, miikka.salminen@ee.oulu.fi, juha.partala@ee.oulu.fi

***Abstract*—The architecture specifies how the system should be designed and built. Several architecture frameworks exist for implementing the architectural design process. However, shortcomings are identified in current architectural design processes, especially concerning volatile domains like healthcare. We claim that an iterative architectural design process is required, where the technical concerns are separated from the non-technical ones. Furthermore, a strong guiding vision is required. Based on our experiences from a biobank IT infrastructure process, we present a Continuous Renewability architectural design process that is modular, interoperable, controlled and abstracted, thus being capable of handling complex systems with severe uncertainties.**

Keywords- Architecture; design; lessons learned; post-mortem; process.

I. INTRODUCTION

Software systems are becoming ever more complex. Consequently, software and systems development has become increasingly challenging and intellectually demanding [1][2]. Therefore, it has been proposed that coherent and comprehensive modelling approaches be applied. Subsequently many approaches are developed in the field of systems architecture modelling [3].

Defining the architecture is an activity that specifies how a system is to be designed and implemented. Several architectural frameworks are available providing guidance on how to enact the architectural design process. However, in domains that are not established or stable, there exist variables that may cause changes and unexpected events that require non-routine solutions. The wider the scope of the project and the more stakeholders that are involved, the more difficult the architecture definition is [3].

Moreover, if the development problem is not well structured, it becomes increasingly more challenging to address and communicate [4]. Healthcare is one such domain that is constantly evolving. There exist several stakeholders from different domains, various laws and regulations are in effect, some of which are still emerging, e.g., General Data Protection Regulation [5], services and service models are still being refined. It is then easy to see the inherent volatility within this particular domain. Hence, we claim that an incremental and iterative process is necessary, where the outcome is built gradually. These facilitates observations of the evolution of the design and implementation over time, and to better understand its requirements and potential– gradually gathering feedback and incorporating it into the development. However, not only design and implementation, but also system use and renewability has to be acknowledged in the architecture.

Furthermore, there is a need to have a strong guiding vision towards which the architecture development efforts can be compared to. In order to define such a process, we propose the following research question: *What form of architectural design process is suitable for volatile environments?* To address our research question, we used a post-mortem analysis to study the process of building a biobank IT infrastructure. As a result, we propose a Continuous Renewability approach to architectural design process.

The remainder of the paper is organised as follows. Section II studies the background. Section III presents the research approach. Section IV presents the architectural design process and the empirical experiences from healthcare domain. Section V evaluates our approach. Section VI discusses the results and implications. Section VII summarises this work.

This paper is an extended version of [6] including, e.g., more detailed literature study, extended description of the proposed approach, and evaluation of the approach.

II. BACKGROUND AND RELATED WORK

The healthcare domain is one example of domains that are constantly evolving by means of new technological innovations, new requirements for efficiency and cost and new regulations being introduced. There also exists the continued interaction and dependency on legacy systems and data formats. The design reality of healthcare IT architecture is sketched in Figure 1.

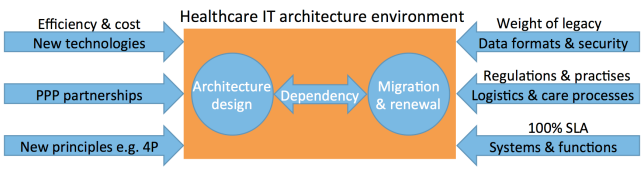

Figure 1. Design reality of healthcare IT architecture.

Legacy systems and data formats that are widely utilised in the medical domain create challenges by means of potentially isolated and non-interoperable systems. In particular, legacy issues arising from the use of existing data formats, processes, applications and service-level agreements (SLA) increase the level of complexity involved in designing a unified technical solution. Thus, there exists a need for migration and renewal strategies in addition to strategies that enable complying with the legacy systems.

Architectural design is heavily guided by requirements regarding efficiency and cost. Emerging technologies may provide better and more efficient solutions to current challenges, public-private partnerships (PPP) funded by a partnership of government and a number of private sector companies and new principles like 4P medicine, referring to preventive, predictive, personalised, and participatory medicine [7]. 4P medicine is also sometimes referred to as personalised or precision medicine. It can be seen as the tailoring of medical treatment to the individual characteristics, needs, and preferences of a patient during all stages of care, including prevention, diagnosis, treatment, and follow-up. It will also include enhancing the awareness about lifestyles and preventive lifestyle changes. The goal is to enhance the health outcomes with integration of evidence-based medicine and precision diagnostics into clinical practice.

Through this holistic approach, in combination with several divergent stakeholders and new technologies, it is easy to see that the design environment may become fragmented and volatile. Furthermore, the healthcare sector is examining new strategies and business models, such as PPP, where the strategic and business drivers are diverse. The gradual evolution of legacy systems towards new solutions must enable the continuing use of existing systems integrated into the current environment. This may potentially result in a complex environment with combination of both legacy systems and applications with brand new solutions [8]. The ongoing evolution through changing legislation, regulations and improvements in medical practices creates an environment that is constantly changing.

Various architectural frameworks have been proposed to address the different design realities, including standards, such as ISO/IEC/IEEE 42010. In addition a general model for architectural design is presented in [9]. Architectural frameworks have been analysed extensively [1][10][11], and a recurring theme across the frameworks is that each describes the role of the architecture in the product development process as a "*systematic analysis and design of related information to provide model for guiding the actual development of information systems*" [10]. The architectural design process is a one that guides the definition of a given system architecture, however, there is no general solution for the representation of a system's architecture [10]. Many architecture frameworks discuss the architecture creation process yet few focus on the process [11]. The value of the processes is shown in the literature and it has been suggested that processes ensure that activities in an organisation are performed consistently and reliably [12]. The architectural design process should provide a structured approach to architecture activities in the product development process [10]. Furthermore, it is also important to acknowledge the phases of system in use and renewability. Thus, architecture should cover: 1) design, 2) implementation, 3) deployment, 4) usage, and 5) renewability. The lack of proper planning for items usage and renewability can often lead to problems for customers, because evolution and renewability is expensive or impossible, system use may be restricted, and new processes are not supported. Maintenance can also suffer if developers only do design, implementation, and deployment.

Several frameworks exist for modelling architecture. While different frameworks have different content and target a different audience [11], they aim to provide structure and systematic processes for systems design [10]. Examples of well known and established architecture frameworks are the Zachman Framework for Enterprise Architecture [13], 4+1 View Model of Architecture [14], Federal Enterprise Architecture Framework (FEAF) [15], Reference Model for Open Distributed Computing (RM-ODP) [16], The Open Group Architectural Framework (TOGAF) [17], DoD Architecture Framework by the US Department of Defense (DoDAF) [18], and a general model of software architectural design by Hofmeister et al [9].

While each architecture framework is suitable for different environments, they may result in similar outcomes based on their architecture goals and viewpoints. Viewpoints are an important feature of architecture frameworks as they represent the goals and focal points that the architecture framework emphasises like business, information, software and technical architectures. The analysis revealed that only three of the frameworks, FEAF, TOGAF and DoDAF, provided explicit support for the architectural design process, RM-ODP provided partial support, ZF and 4+1 View provided no support. ZF and TOGAF have a focus on enterprise architecture, 4+1 View and RM-ODP on software systems (typically distributed), FEAF is primarily a framework for architecture planning and DoDAF focuses on enterprise architecture related to defence operations and business operations and processes. It is thus typically quite domain specific. The general model by Hofmeister et al. is based on synthesis of several existing approaches. [3][9][10][11]

Evaluations of architecture frameworks are presented, e.g., in [3][9][11]. While there are pros and cons for each method, common deficiencies in the architecture frameworks can be identified [10]: 1) The level of details required in models is not specified enough, 2) Rationales are not considered in models, thus no verification is possible, 3) Non-functional requirements are not considered in all frameworks, 4) Software configuration is not considered in all frameworks.

There are also more recent approaches to systems design that aim at tackling the challenges of modern development environments. Palladio Component Model (PCM) is one such approach; among other benefits it enables the analysis of different architectural design alternatives (i.e., optimisation) and aims to address the challenges during the early development stages, thus avoiding costly redesigns

[19]. Software architecture optimisation has also been studied to help the search for optimal architectural design, e.g., Aleti et al. [20] performed an extensive systematic literature review focusing on software architecture optimisation.

Even with software architecture optimisation efforts, architectural design still involves complex trade-off analyses that may require expertise in several domains or the environment may be more variable and dynamic than current process can support. It is even possible that not all stakeholders are known or they may not already know what the intent for the product to accomplish. Thus, such uncertainty may exist that the guiding vision for the product is impossible to be fully defined during the early stages of development. Instead, it is suggested that it is built incrementally.

In conclusion, there is a need for an architectural design process that addresses the identified shortcomings, including: volatile environments, the availability of specific details, design rationales, non-functional requirements, and software configurations.

III. RESEARCH APPROACH

The results are based on experiences gathered during a biobank IT infrastructure development project. The research consisted of studying several organisations related to biobank activities. The purpose of this was to define architecture for a biobank and implement a functional infrastructure. Managing the large number of stakeholders and constantly changing environment requires carefully considered architecture approach, thus creating the need and basis for this work.

During the project several challenges were identified. A post-mortem analysis was conducted to analyse these findings and to identify the shortcomings and improvements for the architectural design process. A post-mortem analysis is a study method that may be used to gather empirical knowledge. The benefits of a post-mortem analysis include revealing findings more frequently than other methods, such as project completion reports. It is beneficial to conduct post-mortem analyses after important milestones and events in addition to the end of a project. Post-mortem analysis can be used as a project-based learning technique [21][22]. In addition to finding the impediments of the development process, post-mortem analyses may be used to improve methods and practises [23]. During this research project, a post-mortem analysis was used to study our development process to facilitate identifying potential sources for improvement or optimisation.

Our post-mortem analysis follows the general iterative post-mortem analysis proposed by Birk et al. [21], as shown in Figure 2.

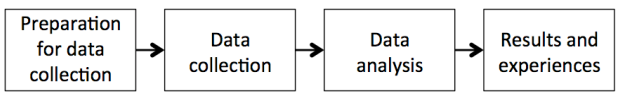

Figure 2. General post-mortem analysis process [21].

TABLE I. VARIOUS STAKEHOLDERS IN BIOBANK DOMAIN

Stakeholder	Input
Valvira (National Supervisory Authority for Welfare and Health)	- Biobank permission - Supervision
Sample donor, person	- Consent - Samples
KELA (The Social Insurance Institution of Finland)	- Information systems service
THL (National Institute for Health and Welfare in Finland)	- Architecture - BBMRI-ERIC: Obligations
BBMRI (Biobanking and BioMolecular resources Research Infrastructure)	- Common methods
Registry	- Source data
Service provider	- Service
Health care units	- Sample and data - Support services
Research	- Sample and data
National ethical board (TUKIJA)	- Reports
STM (The Ministry of Health and Social Affairs)	- National biobank overall architecture

Our research started with an initial preparation stage where we carefully identified the key participants involved with our effort and selected viable methods and procedures.

Project history was examined with the key participants involved in the project (primarily project managers and system architects) and project documents were studied. Then our goal for the post-mortem analysis was determined–to understand the needs for the architectural design process as well as identify potential sources for improvement and optimisation.

Data collection involved gathering relevant project experiences from team members and key stakeholders (Table I). Participants of our data collection as well as data analysis session were project managers (1), system architects (2) and developers (2). A decision was made to conduct a lightweight post-mortem analysis. KJ sessions [24] with thematic analysis [25][26] were utilised to gather and organise ideas and data.

In the analysis phase, findings and ideas were organised into groups based on their relationships. Post-It notes were used to record the ideas and findings and related notes were then grouped together. Based on our results, we present Continuous Renewability architectural design process. Table I presents the key stakeholders that participated in the definition of the biobank. The organisations were chosen as they could each provide potential data related to the research question. Experts and managers from different organisational levels were involved. Examples of input by the sources are also presented.

IV. EXAMINING THE ARCHITECTURAL DESIGN PROCESS

A. Preparation

In the planning phase, project results were studied. These results included meeting memorandums, requirements, company materials and the results produced. The focus of the post-mortem analysis was decided to be to understand and improve the current processes, to find out what challenges regarding to architectural design process exist and where we succeeded. Post-mortem analysis participants were

informed of the procedures and schedules were agreed upon and the goal of the post-mortem analysis was determined. A lightweight post-mortem analysis was selected as it fits the project size best [27].

B. Data Collection and Data Analysis

In this step, a summary of project history was explored with key members of the project to better understand the history of the project. Then we gathered the relevant project experience, and participants were asked to provide their views on the development process and practises. The views were documented using KJ sessions.

Participants were given a set of Post-It notes and they were asked to write down one issue on each note including both challenges and successes. Each note was then attached to a whiteboard and the person was asked to explain why the issue is important. When all the notes were on the whiteboard, they were discussed thoroughly and then they were organised into thematic groups and each group was named, see Table II. Grouping the findings revealed nine themes. These groups indicate the main challenges or needs that were encountered in the development of the biobank IT-infrastructure. These challenges are issues that may often be met in the architecture development in volatile environments.

Data analysis was done in the same session as data collection.

TABLE II. SUMMARY OF THE MOST IMPORTANT FINDINGS

Theme	Finding
Abstraction	+ Abstracting the system design is useful
Change and uncertainty	- Changing requirements, components and environment - Unclear responsibilities - Ongoing efforts that affect the work disruptively
Communication	- Understanding stakeholders from other domains is challenging - Various general communication issues are met - Unclear stakeholders complicates the communication - Critical information not available + Constant communication within the development team was useful + Common vision and shared understanding within the development team was helpful + Building trust between the stakeholders enabled the communication channels to be build
Controlled	+ Planning and decision-making built within the process
Guiding vision	+ First architecture draft providing a guideline + Defining the basic data flows before trying to integrate with the hospital systems
Interoperability	- Numerous interfaces to existing systems - Vast number of systems and applications, including legacy systems
Iterative approach	+ Iterative process builds the outcome gradually
Modularity	+ Following system architecture principles allowing for modular system
Separation of concerns	- Non-technological issues complicating technological issues (politics, rigid processes, etc.) - Complex operational environment requires examination of the system from different views + Identifying new separation opportunities in existing architecture enabled development of isolated domains

After the views and ideas were recorded, they were discussed in detail. A root-cause analysis was conducted to find out why those items occurred. Identifying root causes of the identified issues included consideration of how general these issues are and whom they concern.

Analysis suggests that the most important issue affecting the architectural design was change and uncertainty in addition to communication issues and complexity in the system and environment. Together these hindered development efforts and may potentially affect quite severely the quality of the product. However, we also found ways to tackle these issues. For example, applying good communication practices, iterative development and having a guiding vision are all suggested.

In summary, our architectural design process was iterative in nature. This allowed us to build a shared understanding of the work to be done, to shape the goals, and to react to numerous changes and uncertainties as well as the knowledge gaps between different stakeholders. The work started with a stakeholder analysis to find the relevant stakeholders and their viewpoints regarding biobank IT infrastructure. The key problem in gathering stakeholder views was their wide array of potential wishes and then implementing them on a technical level. Several stakeholders were not technically oriented in their background, thus they did not know the technical restrictions that may exist in such an environment.

Similarly from both a legal perspective and a technological perspective, the various stakeholders had difficulty grasping adequately other domains than their own. Especially challenging were the legal issues regarding sensitive and personally identifiable information. Getting the stakeholder views and mapping them to a technical level in addition to ensuring compliance with laws and regulations is time consuming since there needs to be a consensus amongst the stakeholders. Multiple requirements were identified ranging from very abstract to very concrete. Based on this analysis we were able to come up with an architectural design process for volatile environments.

C. Experiences

Here we summarise our experiences from the architectural design process conducted in order to build a biobank IT infrastructure. The aim is to provide hands-on experience on architectural design process and to provide guidance on how to define architectures for systems that exist in volatile environments, and as with many other frameworks to control the complexity of development by abstracting the system design and modelling the intended system at different abstraction levels.

We suggest an iterative approach in the form of Continuous Renewability, where the work is done iteratively and incrementally with feedback loops throughout the process improving communication. At each level, there are discussions about what is required, and what is already available. Frequently, comparisons to previous levels are done. As the process progresses, the need for changes and their associated effects grows smaller.

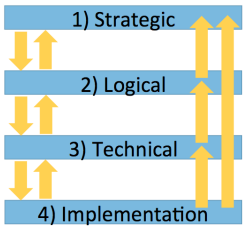

Figure 3. Four levels of abstraction for describing software architectures.

Different approaches can be used, as there are different needs, requirements, environments, etc. In our case, Continuous Renewability approach was required. Biobank data does not become old or obsolete. Instead, the amount of data grows over time. Similarly, the architectural design must be continuous to account for changes and new events and to have scalability and potential for upgrade while still keeping the whole system interoperable.

In the Continuous Renewability model, four levels of abstraction are identified each representing a distinctive view of the architecture from enterprise level to technical details of the system/software architecture, see Figure 3. Each level implements the level above with more detailed technologies and descriptions. Each level is also a phase in our architectural design process. Moreover, each level corresponds to a set of stakeholders, and together the different views form the complete architecture specification.

1) Strategic architecture is the starting point providing the overall description of the development problem, defining business views, business processes and rules, and performance goals. It also defines the conceptual architecture that connects the architecture effort with the visions, organisational strategies, business drivers and goals in addition to processes and functional perspectives.

One of the main contributions is to communicate the vision and define the rationale–why things are required. External input can come from multiple sources including but not limited to various communities, laws and regulations.

At this level, a strategic architecture is defined with the following output: A semantic model defining the relationships of business entities and business processes. Most of the external requirements and customer feedback come through this level as this level is typically closest to the customer interface of an organisation. It must have a solid understanding of customer requirements and it should communicate these requirements in addition to their rationale to internal stakeholders. A strategic architecture is also influenced by the business strategies and other high-level visions. Furthermore, it also receives feedback in an iterative manner from the whole architecture development cycle. It was noted that frequently this strategy consumes a disproportionate amount of time and effort, as it must be strictly representative of reality in order to provide a good basis for further actions.

2) Logical architecture defines the functions and various resources or components of the system including their relations and how information flows throughout the system. Furthermore, this level defines the qualities of the system, i.e., gives the measurements on how to achieve the business goals specified at the strategic level. External input is the inventories of available building blocks for the system.

The results from level 1 are further examined and developed in level 2, where the logical architecture is defined. The output of level 2 provides a logical data model that defines the relationships of data entities.

3) Technical architecture implements the logical level and provides a foundation by defining the technical architecture including technology platforms, information system environments, hardware, software, network components, interfaces, platforms, etc. External input comes from sources including standards, non-functional requirements (NFR) (like redundancy, security, availability, scalability and interoperability). The output of level 3 is to provide a technical architecture that defines the physical data model and a technological architecture.

4) Implementation architecture focuses on details, such as hardware and software, operating systems and middleware in addition to interoperability and data definitions. External input comes from sources including configuration documents, technical constraints and application requirements. The output of level 4 is an implementation architecture that defines the implementation details, such as components, applications and software and hardware configurations. This implementation architecture takes into account all the technical constraints. Furthermore, it also has to communicate back to level 1, e.g., the weight of legacy (like data formats and applications), which will affect planning, system interoperability, business decisions, etc. The weight of legacy is a critical factor in system design in the healthcare domain where there may not exist alternatives that are readily available to replace existing systems.

If changes are made at any level, it may have an effect on any other level. The Master Architecture is defined to guide the development effort, structure and scope of the process. This is illustrated in Figure 4.

The Master Architecture maintains the up-to-date specification in addition to a specification document produced for each level. Defining the Master Architecture can start from the current architecture or current standards and infrastructures.

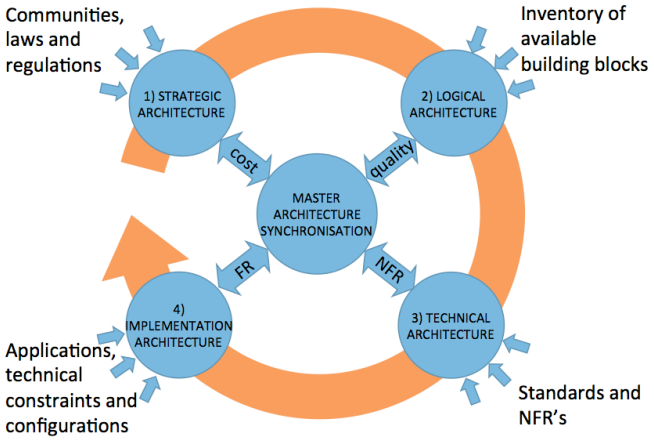

Figure 4. Continuous Renewability architectural design process.

First, a high level architecture is used which will then be further specified as the architectural design process progresses. The Master Architecture may also illustrate a set of use-case scenarios that can be referred to at each level to understand requirements of the system to be designed.

The model has to account for 1) reality, 2) methodologies, 3) design models, and 4) design functionalities.

Each phase has to correspond to the Master Architecture to verify the feasibility and progress of the development towards the set criteria. For example, phase 1 relates to synchronisation of costs and trade-offs, phase 2 to quality aspects, phase 3 to NFR's and phase 4 to Functional Requirements (FR). Similarly, the produced documents and items are verified against the Master Architecture.

The resulting architecture from this process is available at [28].

V. EVALUATION

Our Continuous Renewability approach is an amalgamation of several best practices found in other approaches and design methods. Table III presents a comparison of several commonly used approaches with our approach, and consider how they address the challenges and needs encountered in the architecture development in volatile environments. The themes captured in post-mortem analysis provide a good starting point for identifying requirements for architecture development in volatile environments. These themes are also recognised in the literature. The approaches are evaluated towards the identified requirements. The evaluation is literature based. Each approach is checked if they fulfil the requirement fully, partially or not at all. The Continuous Renewability approach (CR in the Table III) is built to address the identified requirements fully.

It should be noted, however, that DoDAF is limited in scope and it does not address the relevant views for implementing the system as a software architecture as well as the other approaches [29].

Abstraction is one of the requirements for architecture description languages [30], furthermore, abstraction is necessary to enable the examination of architecture from different perspectives. Zachman, FEAF and DoDAF address the abstraction requirement, while TOGAF has very limited views [31]. Abstraction is addressed in 4+1 at least partially with different views.

Changes and uncertainty. In real life, work has many variables, changes and unexpected events are met, vast amounts of data must be handled, and innovative solutions are needed [12][32][33]. Readiness for changes is necessary to adapt to future situations [9], furthermore, uncertainty and changes are also often met during the development work. Flexibility minimise the impact of changes. Change and uncertainty is addressed in Zachman, DoDAF and Hofmeister. FEAF accommodates changes at least partially through flexibility of methods, work products and tools [30]. TOGAF has a flexible process and accommodates changes and promotes change management [10][34]. RM-ODP does not consider the future needs or evolution of the architecture [10]. 4+1 does not address system evolution [10].

Communication is a mediating factor in coordinating and controlling the collaborative work. Software development requires a vast amount of communication, especially when dealing with complex infrastructures. These issues have been reported to decrease both the frequency and quality of communication, and ultimately, productivity. To mitigate these issues, tools, processes, and methodologies are required. [35][36]

Communication is addressed in Zachman (through abstraction, simplification and common vocabulary) and Hofmeister, but not explicitly. FEAF provides a common language and facilitates communication [34]. DoDAF address communication at least partially through extensive documentation [10]. 4+1 address communication requirement fully. RM-ODP provides a framework for defining the languages for the viewpoints to be used as a dictionary for architecture description [29].

Controlled refers to rigid processes and best practices that the architectural design process is based on. It also overlaps with the guiding vision, as control also comes from the ability to evaluate constantly the results towards set targets, ensuring the correct architectural decisions [9].

FEAF (partially) and TOGAF (fully) provides process support [10][31][34]. FEAF measures success [34], while TOGAF lacks the continuous evaluation or validation. 4+1 provides partial support through the validation of the architectural design, while DoDAF defines the process and evaluation [10][31][34] and Hofmeister provide full support for controllability. RM-ODP does not describe the architectural design process [10].

Guiding vision provides a common goal to guide the development and harmonise the practices. Guiding vision also acts as a baseline towards which the development can be verified.

TABLE III. COMPARISON OF APPROACHES (0=NO SUPPORT, 1=PARTIAL SUPPORT, 2=FULL SUPPORT)

Requirement	Zachman	FEAF	RM-ODP	TOGAF	DoDAF	4+1	Hofmeister	CR
Abstraction	2	2	1	2	2	1	1	2
Change & Uncertainty	2	1	0	2	2	0	2	2
Communication	1	2	1	0	1	2	1	2
Controlled	0	1	0	1	2	1	2	2
Guiding vision	0	1	0	2	2	2	2	2
Interoperable	0	2	2	2	2	0	0	2
Iterative	0	1	0	1	2	2	2	2
Modular	0	1	1	0	0	2	0	2
Separation of concerns	1	2	1	0	1	1	2	2

Evaluation is ensuring that the architectural decisions are the correct ones [9]. DoDAF provides description of the intended product, with guidance and rules for consistency [18]. TOGAF and 4+1 guides the organisation with an architectural vision. In FEAF each segment has a guiding vision [34].

Interoperability is one of the drivers that contribute to the success of the product and it is necessary to address interoperability already in the architectural design [39].

FEAF, RM-ODP, TOGAF and DoDAF explicitly promote interoperability [10].

Iterative development is emphasised in [9], it was also revealed as one of the success factors in our post-mortem analysis. FEAF, TOGAF and 4+1 are iterative, however, FEAF and TOGAF do not explicitly propose iterations after each phase, but only after the whole process [9][34]. DoDAF is iterative [18].

Modularity is recognised as a crucial attribute in software architecture [40][41]. FEAF uses autonomous partitions to manage complexity [34]. 4+1 supports modularity to promote ease of development, software management and reuse as well as addressing environmental constraints [10].

Separation of concerns provides several benefits, such as reduced complexity, improved reusability and simpler evolution [37][38]. Zachman, RM-ODP, DoDAF and 4+1 partially address this requirement through views. Hofmeister address the complexity and separation of concerns. FEAF address this, as it is built on segments and enterprise services, which can be seen as views to development [34].

VI. DISCUSSION

Volatile environments present many non-trivial challenges for architectural design and specification. A post-mortem analysis was conducted on a biobank IT infrastructure project to understand the architecture based on real problems and attempted solutions. A post-mortem analysis is a tool that can be used to learn from the experiences of previous iterations or completed projects. It can also be used to improve and adapt current software development processes [22]. Here, our aim was to learn from our experiences and then suggest improvements for architectural design processes, especially for volatile environments.

Our Continuous Renewability approach to architecture is a) modular, b) interoperable, c) controlled and d) abstracted. This way we can handle complex systems with significant inherent uncertainties. The design philosophy is that when designing the architecture, the requirements were separated into technical and non-technical requirements whereby the non-technical requirements included requirements specialised to the biobank domain, like sample management and identification and table structures. These are requirements that do not affect the design of data flow, as we only need to know that the data exists and will be in some form that can be trivially read from, written to and transmitted securely over an encrypted socket-based connection directly to the next stage. The specific content of the data is largely irrelevant in most cases.

Incremental and iterative development is suggested as it allows observing the outcome and improving it as new information becomes available. Our proposal is a Continuous Renewability architecture model, which is intended to be general, such that it does not mandate how each level should be modelled. This allows several architectural styles and notations to be utilised. More important is that all the necessary views to a development are addressed. Furthermore, our architecture is modular to allow flexibility and extendibility. Modularity allows the reconstruction of any part of the system, such that an area-of-effect can potentially be localised to just those components directly connected to the modified region. In the case of the biobank, the system has been designed such that successive system component regions typically form a directional data flow through standardised and well-defined interfaces. Interoperability is similarly achieved through the specification of interfaces defining various domains with utilisation of open-standard communication protocols. Controllability comes from the rigorous process and from the Master Architecture that guides the development and verifies the outputs against the set targets. Architecture should also be highly abstracted. For example, there exist requirements that are irrelevant when designing a data flow because we only require knowledge that a given data exists and will be in some form that we can work with. The specific content of the data is largely irrelevant in most cases. This is highly beneficial in an evolving healthcare environment whereby the specific content of a data set in addition may frequently be in a state of flux while the laws and regulations surrounding the data set are interpreted. Increasing or decreasing the level of abstraction as required allows the examination of the system from different perspectives. The separation of technical concerns from non-technical concerns allows us to adapt to future needs, as the design is not relying on specific technologies or solutions.

In volatile environments, constant comparison to the Master Architecture is required. It allows for the verification of compliance for all the relevant inputs and design choices, even if those vary during development. The Master Architecture provides the goals towards which the effort is pushed as well as the guidelines that determine how those goals should be reached. Structure for the process is also provided. Design rationales guide the overall work and are kept up-to-date by continuous communication with the stakeholders who see the system being defined incrementally. Continuous communication also helps building trust between the stakeholders. This allows them to understand the rationale for design and implementation decisions better throughout the process as a consequence of context being more localised. Input is verified at each level and every iteration. Additionally, the Master architecture is updated accordingly. Comparing the results to the Master Architecture enables a constant feasibility analysis, and enables corrective actions if necessary.

We suggest an approach to architecture whereby domains are the fundamental units and the communication pathways between the domains indicate connectivity between domains. The internal structure of a given domain remains unspecified

in the highest level of abstraction. It is only specialised once the requirements for that domain are exhibiting some form of stability. For example, we can consider the anonymising encoding service of the biobank not as a part of the architecture, but as a specialisation of a domain for a specific task. Thus, if for example, the law changes or it turns out it was misinterpreted, the specialised components of the domain may be updated or replaced with minimal impact to the architecture assuming the new specialisation utilises the existing connection path and communicates using compatible data storage and communication formats. It then follows that any connected domains from which it receives from or transmits to must be able to accept that communication readily.

This is accomplished by initially designing the system at a high of abstraction, modelling the transformations that occur in a domain as a function with an argument type $\boldsymbol{T}$ that maps to some other type $\boldsymbol{U}$ where $\boldsymbol{T}$, $\boldsymbol{U}$ may have some structure or may represent a collection of different data types. It is also important to note that type identifiers, such as $\boldsymbol{T}$ or $\boldsymbol{U}$ are arbitrarily chosen and the label communicates only the preservation, or lack thereof, of the structure of the input data. The labelling of an input and output type is defined such that if the input and output types of a domain are identical as is the case in a mapping from $\boldsymbol{T}$ to $\boldsymbol{T}$ then the transformation that occurs is said to be structure preserving such that the output contains an identical structure to the input type. An example of this could be structured tabular data with given column headings. If the transformation does not modify this table structure, instead only reading the contents or modifying the table contents then it is said to be structure preserving. It is then possible once a directed graph of each transformation is obtained to perform algebra upon this graph. Such operations may include the simplification of the structure through composing transformations or identifying potential incompatibilities between domains through type mismatches. Each domain in the architecture is constructed from many transformations composed in such a manner that the functionality of a domain may be mapped as the composition of many functions.

In practise, many concepts may not map naturally to this model. Examples include data storage on disc and databases. In such cases, it is possible to map these as either state machines or simply as entities in the data flow that label a particular complex process. As requirements stabilise and become readily available, the intent is for an architecture defined in this abstract manner to reduce down to a traditional architecture specification.

Designing the system this way allows us to largely disregard the shifting external environment and design a system around the modelled data flow rather than the specific form of the transformations until such time that information exhibits stability. It is only required that information regarding what transformations are required exists. This way, the architecture is largely resilient against variation in both non-functional and many non-technical requirements as each domain is intended to be entirely self-contained with all state being local to that domain and any information that enters the domain is passed directly to it and the given output from a domain depends only upon information contained within that domain.

We propose that architecture specifies the interfaces to the various domains and utilise open standard protocols for communication. It is specified that all data be retained so any variation in requirements downstream can be trivially propagated through the signal chain or the entire data set can be rebuilt at any time if a failure occurs somewhere. Similarly, by defining the interfaces between domains, it is possible to enforce properties, such as strong and guaranteed cryptography on communications and storage in addition to simple topology modifications due to a standardised interface between domains. While this requires additional work in the implementation stage, by communicating through a unified routing system, it ensures that future software replacing legacy or unsuitable components may develop against a known, open communication protocol removing the possibility that proprietary vendor communication methods hamper third-party inclusion into the architecture. There are many benefits of this approach, as shown in Table IV.

With this in mind, we believe that this approach is not limited to a single field (pathology, genetics or similar) and does not depend on a single company. This is a general model that can apply to any domain of any size.

TABLE IV. BENEFITS

- Since the creation of abstract domains is largely trivial and the communication between those domains follows open standards, each domain is fully knowable and may be audited. The system may then easily adapt by localising changes to only the affected domains.
- Adapting to future needs is made viable using this architecture, as it doesn't matter whether the software used to power a particular domain is open source or proprietary as long as it conforms to the open standard data storage formats and communication protocols, it can be replaced or upgraded.
- There is much less chance of a given software company creating a monopoly in the business domain by providing a large monolithic system that is proprietary and does not allow (or limits) the ability for third-parties to build upon or interface with it.
- There is opportunity for innovation because anyone can develop candidate solutions for domain specialisation without needing to invest effort in satisfying criteria regarding licensing other vendor APIs. It also allows for larger scale international collaboration.
- The organisation is free to choose any software, open source or proprietary to specialise each domain. We specify in our prototype biobank implementation architecture open source software because for our purposes existing solutions exist for many of the domain specialisations and it is possible to implement new functionality upon the existing code bases with relative ease. However, the client remains free to choose the software solutions they deem adequate. The only requirement is that the communication between domains follows open protocols with implementations provided either by an existing library or directly as part of the core infrastructure.
- It has a potential to be cheaper to maintain. For example, if there is a decision to go for an entirely open source system, not only does there not typically exist a license cost, there may exist multiple potential options regarding which organisation to hire for supporting and maintaining the system. That way they can receive quotes and optimise expenditure based on the value each quote offers.
- Since rigid software design processes may stagnate and impede innovation. By having a modular system, any organisation may be required to innovate whether it is by feature set or cost as there may not exist a possibility to implant a system at the project's inception and rely on the difficulty of switching to a competing product as a source of longevity in the deployed infrastructure.

This is where the novelty and innovativeness of this approach lies. We suggest a system design method that is resilient to changing requirements and constraints and is dependent only upon technological requirements, one that can adapt and grow to any scale and is both modular and knowable.

There do, however exist limitations to this approach. In this case, there is limited access to end-users as only a limited number of healthcare professionals were directly participating in the process.

We had to rely on application and service providers, who served as an intermediary between the researchers and the end-users. However, the application and service providers are established and well known in their domain and have a strong knowledge of the needs and requirements. We can thus rely on their experience for making informed decisions.

The results should interest both academics and practitioners as they provide an experience report on a generalised architectural design process for volatile environments. This is a method for designing a system in such a way that it bypasses many non-technical issues by separating the technical concerns from the non-technical concerns through modular design. The study also lays the groundwork for further scholarly inquiry, including validating the findings in practice.

VII. Conclusion and Future Work

This paper presented lessons learned from a biobank IT infrastructure project. A post-mortem analysis was conducted for biobank IT infrastructure process, where several challenges are encountered. Further challenges were presented by the strict requirements for privacy and anonymity as well as rigid processes involved with the patient data. We have identified several challenges and solution proposals.

Table V shows the overview of proposed solutions and summarises how the challenges may be addressed.

By following these proposals the resulting architecture will be a) modular, b) interoperable, c) controlled and d) abstracted. It is also suitable for volatile environments, thus addressing our research question.

Continuous Renewability approach is general by definition and should be easily adapted to other domains. However, the practical generalisability of our results is limited until the process is used in other domains. It is important to note that the viability of this approach will need to be verified through controlled experiment and observation. Though, the generalisability is one of our main design philosophies guiding the development, hence we believe that generalisability issues are likely to be negligible. This model should be refined as feedback from applications is received.

TABLE V. Overview of Proposed Solutions

- Changing requirements, components and environments are tackled with iterative process that builds shared understanding, shape the goals and allow reacting to changes. Furthermore, Continuous Renewability approach enables constant feedback and mitigates the effects of changes as the process progresses.
- Several communications related issues are tackled with iterative process, as it allow stakeholders to see the system grow, and their understanding improves along with the system. Improved communication practices also are necessary, starting from the planning to create a common vision on what to build and continuing through the whole development cycle. Constant communication also builds trust between the stakeholders.
- The Master Architecture provides the scope and guidance for the development work. First architecture draft is defined to provide a guideline for development. Then the basic data flows are defined. Master Architecture provides a checkpoint towards which the design can be verified, while designing the system around data flows mitigates the complexity as well as the effects of changing external environment.
- An iterative approach was adopted to build the outcome gradually. With the modular system architecture and abstracted systems design it allows for updating the design with minimal effort and minimal impact to other parts of the system. Abstraction and separation of concerns allows for adaptable design that accommodates the future needs and is scalable. It also is not reliant on certain technologies or solutions.
- Separation of non-technological issues from technological issues simplifies the design, as it isolates, e.g., the effects of politics and rigid processes from the technological concerns.
- The separation of the architecture into isolated domains connected through a common interface can serve to restrict the propagation of errors through the system in the event of component failure or modification. This in turn has the potential to offer greater flexibility and expansion of the system to meet future needs.
- Interfaces between the domains and to existing systems utilise the open-standard communication protocols. This ensures interoperability, as the components can be changed according to future needs.

References

[1] P. Robillard, "The Role of Knowledge in Software Development," *Communications of the ACM 42(1)*, pp. 87-92, 1999.

[2] F. O. Bjørnson and T. Dingsøyr, "Knowledge Management in Software Engineering: a Systematic Review of Studied Concepts, Findings, and Research Methods Used," *Information and Software Technology, vol. 50*, pp. 1055-1068, 2008.

[3] S. Leist and G. Zellner, "Evaluation of current architecture frameworks." In *Proceedings of the 2006 ACM symposium on applied computing*, pp. 1546-1553, 2006.

[4] J. C. Brancheau, L Schuster, and S. T. March, "Building and implementing an information architecture," *ACM SIGMIS Database 20(2)*, pp. 9-17, 1989.

[5] European Union, "General Data Protection Regulation 2016/679." 2016 URL http://eur-lex.europa.eu/eli/reg/2016/679/oj, 2017.07.19

[6] J. Hyysalo et al., "Defining an Architecture for Evolving Environments." In *Proceedings of SAC 2017*. In Press.

[7] C. Auffray, D. Charron, and L. Hood, "Predictive, preventive, personalized and participatory medicine: back to the future," *Genome Med 2(8)*, p. 57, 2010.

[8] F. M. Ferrara, "The standard 'healthcare information systems architecture and the DHE middleware," *International Journal of Medical Informatics 52(1)*, pp. 39-51, 1998.

[9] C. Hofmeister et al., "A general model of software architecture design derived from five industrial approaches" *Journal of Systems and Software*, *80*(1), pp. 106-126, 2007.

[10] A. Tang, J. Han, and P. Chen, "A comparative analysis of architecture frameworks." In *11th Asia-Pacific Software Engineering Conference, 2004*, pp. 640-647, 2004.

[11] U. Franke et al., "EAF2-a framework for categorizing enterprise architecture frameworks." In *10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*, 2009, pp. 327-332, 2009.

[12] P. Mangan and S. Sadiq, "On Building Workflow Models for Flexible Processes." In *Proceedings of the 13th Australasian Database Conference*, pp. 103-109, 2002.

[13] J. Zachman, "A framework for Information Architecture," *IBM Systems Journal 38(2&3)*, pp. 454-470, 1987.

[14] P. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software 12(6)*, pp. 42-50, 1995.

[15] FEA, "Federal Enterprise Architecture Framework version 2." 2013 URL https://obamawhitehouse.archives.gov/sites/default/files/omb/assets/egov_docs/fea_v2.pdf, 2017.05.04

[16] J. Putman, "*Architecting with RM-ODP*," Prentice Hall, NJ, 2001.

[17] The Open Group, "The Open Group Architecture Framework (Version 9.1 "Enterprise Edition")." 2003 URL http://www.opengroup.org/architecture/togaf/#download, 2017.05.04

[18] Department of Defense, "Department of Defense Architecture Framework Version 2.02 - Vol 1 Definition & Guideline and Vol 2 Product Descriptions." 2010 URL http://dodcio.defense.gov/Portals/0/Documents/DODAF/DoDAF_v2-02_web.pdf, 2017.05.04

[19] R. Reussner et al., *The Palladio component model*. Technical report, Karlsruhe Institute of Technology, 2007

[20] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software architecture optimization methods: A systematic literature review," *IEEE Transactions on Software Engineering*, *39*(5), pp. 658-683, 2013.

[21] A. Birk, T. Dingsoyr, and T. Stalhane, "Postmortem: Never leave a project without it," *IEEE Software 19(3)*, pp. 43-45, 2002.

[22] M. Myllyaho, O. Salo, J. Kääriäinen, J. Hyysalo, and J. Koskela, "A review of small and large post-mortem analysis methods." In *Proceedings of the ICSSEA*, pp. 1-8, 2004.

[23] B. Collier, T. DeMarco, and P. Fearey, "A defined process for project postmortem review," *IEEE Software 13(4)*, pp. 65-72, 1996.

[24] R. Scupin, "The KJ Method: a technique for analyzing data derived from Japanese ethnology," *Human Organization, vol. 56*, pp. 233-237, 1997.

[25] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative research in psychology 3(2)*, pp. 77-101, 2006.

[26] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering." In *International Symposium on Empirical Software Engineering and Measurement*, 2011, pp. 275-284, 2011.

[27] T. Dingsøyr and N. B. Moe, "Augmenting experience reports with lightweight postmortem reviews." In *Product Focused Software Process Improvement*, pp. 167-181, 2001.

[28] J. Hyysalo, A. Keskinarkaus, G. Harper, and J. Sauvola, "Architecture Enabling Service-oriented Digital Biobanks." In *Proceedings of the 50th Hawaii International Conference on System Sciences (HICSS-50)*, January 4-7, 2017, Hawaii, pp. 3469-3478, 2017.

[29] N. May, "A survey of software architecture viewpoint models." In *Proceedings of the Sixth Australasian Workshop on Software and System Architectures*, pp. 13-24, 2005.

[30] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Transactions on software engineering*, *26*(1), pp, 70-93, 2000.

[31] L. Urbaczewski and S. Mrdalj, "A comparison of enterprise architecture frameworks," *Issues in Information Systems*, *7*(2), pp. 18-23, 2006.

[32] M. M. Kwan and P. R. Balasubramanian, "Dynamic Workflow Management: A Framework for Modeling Workflows." In *Proceedings of the 30th Hawaii International Conference on System Sciences (HICSS-30)*, pp. 367-376, 1997.

[33] M. Klein and C. Dellarocas, "A Knowledge-Based Approach to Handling Exceptions in Workflow Systems," *Computer Supported Cooperative Work 9*, pp. 399-412, 2000.

[34] R. Sessions, "Comparison of the top four enterprise architecture methodologies." 2007 URL https://msdn.microsoft.com/en-us/library/bb466232.aspx, 2017.05.04.

[35] M. Jiménez, M. Piattini, and A. Vizcaíno, "Challenges and improvements in distributed software development: A systematic review," *Advances in Software Engineering*, 2009 (3), pp 1-16, 2009.

[36] E. Carmel and R. Agarwal, "Tactical approaches for alleviating distance in global software development," *IEEE Software 18*(2), pp. 22–29, 2001.

[37] D. Soni, R. L. Nord, and C. Hofmeister, "Software architecture in industrial applications." In *Proceedings of the 17th International Conference on Software Engineering (ICSE 1995)*, pp. 196-196, 1995.

[38] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton Jr, "N degrees of separation: multi-dimensional separation of concerns." In *Proceedings of the 21st international conference on Software engineering*, pp. 107-119, 1999.

[39] D. Garlan and D. E. Perry, "Introduction to the special issue on software architecture," *IEEE Transactions on Software Engineering 21*(4), pp. 269-274, 1995.

[40] K. J. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen, "The structure and value of modularity in software design." In *ACM SIGSOFT Software Engineering Notes 26*(5), pp. 99-108, 2001.

[41] K. Sethi, Y. Cai, S. Wong, A. Garcia, and C. Sant'Anna, "From retrospect to prospect: Assessing modularity and stability from software architecture." In *Proceedings of the Joint Working Conference on Software Architecture & European Conference on Software Architecture (WICSA/ECSA 2009)*, pp. 269-272, 2009.