Modernization from a Maintenance Process Perspective: Challenges and Lessons Learned

Aiko Yamashita
Department of Information Technology,
Oslo and Akershus University College of Applied Sciences, Oslo, Norway
Email: aiko.yamashita@hioa.no

***Abstract*—Modernization and migration initiatives are not limited to projects where complex legacy systems need to be phased-out. They include wider contexts, from the replacement of obsolete middle tiers, to the migration of algorithms from prototype-purpose platforms to commercial platforms. As the need of modernization continues to increase, we need to understand better what are the challenges to be addressed in relation to modern practices and processes. Aspects such as: decision-making on the migration of components/sub-systems, management of operations during the phase-out stages, critical knowledge and business logic transfer, they all impose demands on the development cycle and the way in which projects are planned and carried out. This paper presents a synopsis of challenges encountered during several modernization and migration initiatives within different industrial domains, across organizations spanning diverse countries. Some key lessons learned were: (1) work planning needs to be adjusted to handle better information uncertainty, (2) estimation practices need to be fine- tuned, e.g., by explicitly allocating information foraging activities prior estimation activities, (3) documentation and traceability cannot be neglected, (4) the phenomenon of 'role creep' should be avoided, and finally (5) clear processes need to be in place for the procurement of appropriate test data, and for enabling test automation.**

***Index Terms*—IT modernization, software migration, maintenance processes, industrial report, knowledge management**

I. Introduction

Modernization and migration (from hereon called just modernization) initiatives are not limited to projects where complex legacy systems need to be phased-out, but include wider contexts, from the migration of legacy services to the cloud, the replacement of obsolete middle tiers, to the migration of algorithms from prototype-purpose platforms to commercial platforms, etc.

Aside from the inherent technical challenges, these initiatives comprise a wide range of challenges with respect to *how* to best calibrate a given development or maintenance process. Aspects such as: decision-making on the migration of components/sub-systems, management of operations during the phase-out stages, critical knowledge and business logic transfer, all these aspects impose demands on the development cycle and the way in which projects are planned and executed.

With the popularization of agile methods, devOps, and continuous integration paradigms, it becomes necessary to observe more closely, the recurrent challenges found in modernization projects, to foresee the interplay between those challenges and the particularities of the different development paradigms.

While many studies have investigated modernization challenges, they often do so on an individual case basis. Thus, it is the position of this author that more discussion is needed in terms of identifying and describing recurrent patterns in modernization initiatives spanning different contexts and domains. That would provide a more complete picture, where the idiosyncrasies of *modernization initiatives* can be incorporated to the set of considerations for fine-tuning methodologies/processes, or for evaluating the suitability and/or need of new methods and tools for this particular "problem domain".

This paper is not meant as an exhaustive account of challenges within modernization initiatives. It is aimed at pro- viding a synoptic account on some of the prevalent challenges observed, which were extracted from personal experiences while working as a software consultant/advisor during the last 10 years. In that sense, this work constitutes more of a times-series experience report than an empirical study; thus its results should by not means be equated with outcomes from formal empirical studies.

Rather, the intention of this work is to initiate a more active discussion within the community on process-related challenges in modernization endeavors and examining those under the light of current development methodologies, such as agile. Such discussion would promote further effort at identifying, formalizing and verifying the interplay between the different socio-technical variables/aspects that form part of this complex picture that we call modernization.

The projects on which the observations are based upon, belong to the following domains: retails, medical, logistics, and services. The projects were performed in diverse organizations based in USA, Japan, Sweden, and Norway. While a formal cross-comparison of the particular challenges across different domains is out of the scope of this work, the summary presented includes challenges that were observed across at least two

Manuscript received December 28, 2016, revised May 14, 2017.

doi: 10.12720/jait.8.2.107-113

different projects. Based on the challenges and guidelines, sets of possible avenues of research are discussed. The reminder of this paper is as follows. In section 2, a short introduction to related literature is presented. In section 3, a brief description of the context from the projects is provided. Section 4 presents the summary of the perceived challenges in modernization initiatives. Section 5 discusses lessons learned and Section 6 provides concluding remarks and future work.

II. Background and Related Work

According to Barbier & Recoussine [1], *software modernization* is just one of the many terms that refer to the transition from outdated systems to newer ones (other applicable terms are: replacement, migration, renovation, recasting, revamping). They state that software modernization can be understood with in a variable-geometry sense, defined by the techniques behind the modernization, the underlying intentions, and the scope/need of the modernization. The term "modern" implies often that the resulting system(s) are moving away from obsolete languages/platforms/standards/tools, and are supported/surrounded by perennial and more efficient ones. Previously, the term modernization and migration have been used interchangeably, and it is the intention in this paper to keep a rather wide interpretation, alongside the view of Barbier & Recoussine. The work by Seacord [2] and Ulrich [3] constitute two seminar books when it comes to modernization of legacy systems and architecture-driven modernization, with concrete examples involving COBOL systems. Tilly et al. [4] explore the challenges on the early stages of Service Oriented Architecture (SOA), when Simple Object Access Protocol (SOAP) was gaining popularity. More recent research on modernization initiatives report experiences from projects within SOA, but with the incursion of Cloud Computing [5]–[9]. For example, Ali Babar & Chauhan [7] discuss the challenges of cloud computing, mostly from infrastructure and architectural perspectives. Stavru et al., [9] in the other hand, addresses concretely the topic of *challenges* for modernization. They assert that the extraction of business processes demand high costs whiles constituting an essential prerequisite for the specification, business design and implementation of services. They also pinpoint those modernization initiatives within complex SOA implies governance issues, unless clear frameworks are established to identify and define roles and responsibilities. Also, the complexity of SOA implies also a wider dependency to external systems and third parties, and consequently, risk mitigation becomes an important area of concern. Whiles Stavru et al., provides a catalogue of agile components to deal with some of these challenges, the analysis remains at a prescriptive level and does not include an in-depth assessment on the degree of suitability of these techniques.

Other work [10]–[12] describes how model-driven processes and technologies can support modernization initiatives. One example is by Fleurey et al., [10] who present a case on how Model-Driven Engineering (MDE) constitutes a cost- effective alternative to out-sourced manual re-development. They actually describe a process followed the migration of a large-scale banking system from a COBOL based mainframe to J2EE. They discuss two main challenges of Model-driven migration: commercial limitations imposed by preliminary costs and tasks (such as business process extraction), and the cost of testing, which represented 45% of the total migration cost (they argue this last challenge is applicable to any software migration in general). They add that the underlying reason of these high costs is because test tasks were mostly handled manually.

Teppe [13] reports on a migration project of a flight booking system written in SPL[1] to C++. Teppe echoes the findings of Fleurey et al., by reporting that testing played an important role and consumed a significant part of the resources. Two major reasons were: the difficulty of establishing repeatable, realistic test scenarios, given the complexity of the states and conflicting modes (e.g., booking a seat on an airplane cannot be repeated with the same seat, as it would appear occupied the second time), and the comparison of results, due to the non-deterministic sequences in which sub-tasks would be processed (e.g., the order of the incoming messages). This last aspect would demand occasionally manual evaluation. Coordination was an aspect deemed important, given the high complexity in terms of involved stakeholders. On those respects, a central point of coordination and decision-making was deemed vital to ensure an adequate prioritization of tasks and resource allocation. This last point was deemed as a major risk, given that projects with more immediate benefits would often overthrow the migration initiative. Teppe asserts that this leads to *"resource conflicts and excessive implementation times and is a major reason why long running migration projects are abandoned."*

Teppe also discussed the importance of knowledge transfer. The staff working with the mainframe environment needed to be "migrated" and retrained in the new platforms/technologies. This was done in order to integrate them to the new teams/projects, as they held the key to the application and production know-how, which constitutes a major asset to the company. According to Teppe, "*process knowledge is more important to a user organization than technical knowledge, e.g. the command of a particular programming language or operating system.*" From the cited related work, is possible to observe that the focus on technical challenges and suggestions for methodologies/tools is the most prevalent; and the actual challenges from a maintenance/evolution process perspective need to be distilled individually from the set of industrial case studies. Moreover, the interplay between methodological aspects of modern development methods (e.g., agile, continuous development) and the concrete challenges within migration initiatives *from a process perspective* has not yet been investigated in detail. But for that to happen, a more comprehensive

[1] SPL is a procedural programming language to support system level programming under the operating system BS2000 [13].

compendium of modernization challenges from both process and technical perspectives need to be in place.

III. Context and Technologies

Due to space limitations and disclosure restrictions, is not possible to provide in-depth details of the contexts the observations have been drawn from. However, this author hopes that the information given in this section would give enough detail to judge the relevancy and representativeness of the projects. As mentioned previously, the domain areas consist of *retails* (ranging from books to consumer electronics), *medical* (i.e., medical devices), *logistics* (maritime and land based), and *services* (i.e. purchase and tracking of services, as opposed to goods). The companies involved comprised a rather diverse set: USA, Japan, Sweden and Norway. The source technologies involved were also diverse, ranging from COBOL, VisualBasic, ASP, and PHP programming languages, to obsolete frameworks such as Java Swing [14], and prototype programming languages such as LabView[2]. The target technologies involved J2EE, JPA[3], Jersey[4], .NET C#, and JavaScript (including different JavaScript frameworks) amongst others. The final solution implemented by the target technology displayed different architectural styles such as: micro services architecture, microkernel architecture and a mix of event-based and layered architectures. The size of the systems (both source and target) differed substantially across projects, ranging from 5~10 KLOC to thousands of KLOC. The great majority of the projects used some form of agile method (mostly Scrum), and deliverables were made in an iterative fashion.

IV. Challenges Observed

The catalogue of challenges described in this chapter encompass four main areas: (a) Reworking of solution, (b) Drivers for defects, (c) Time consuming (manual) tasks, and (d) Work planning.

A. Reworking of Solution

This challenge relates to reworks/reimplementation that is often performed during a project. Miscommunication and lack of a complete overview on the requirements, of- ten led to this problem. For instance, misinformation often leads to the introduction of incorrect data structures, which can be considered in a way as a 'Technical debt', leading to continuous refactoring/restructuring that could have been avoided. This challenge reflects that high volatility of code due to uncertainties on the requirements and business logic can lead to sub-optimal designs, in a self-reinforcing problem.

Many times, although the business-flow at a high level or requirement level is more or less clear, the technical details of a specific platform obscured the business logic. Often in projects developers who were experts in the legacy platform or the source platform were not involved or not available. This compromised the accuracy and/or correctness of the assumptions, which played an important role in the early solution design stages and work planning. In later stages of some projects, new information was often uncovered or made available (e.g., a COBOL developer dropped by and found that the Java developer had completely misunderstood a logical flow). This situation is also reflected by Teppe [13] who asserts that experts in the legacy technology are often the 'domain experts', and as such, they should be closely involved in the initial stages of a modernization project.

B. Drivers for Defects

This challenge is associated to the perceived root-cause for faults, failures and wrong implementation that is experienced in a project. This includes: a) wrong or missing information (requirements, business logic, etc), b) ripple effects or side effects after changes (due to for example, system complexity), and c) difficulties by developers to assess the potential implications of the changes (i.e., concept location, impact analysis).

In many cases, drivers responded to an underlying pressure to focus on 'functionality' rather than for instance, keeping the documentation updated, which lead to more misinformation and wrong assumptions. This last problematic is also reported by Teppe [13] who discuss the internal struggles for delivering 'useful functionality' rather than focusing on internal proper- ties or artifacts that are important for migration purposes.

An example of wrong or missing information is when developers when moving from one platform to another make wrong assumptions. For the medical system, the use of incorrect casting operations (e.g., truncating decimals instead of rounding) leads to costly defect fixes.

It was found after that conversion in LabView is implicitly managed, in contrast of C#, since the latter requires a specific conversion method. In the same way, misinterpretations of algorithmic elements specified in prototype languages (e.g., declaration of global variables and static values interpreted as local and dynamic variables) manifested in accuracy-related defects.

Although these two examples lean more towards technical issues, they also respond to the inability or the challenge to foresee potential conflicts across platforms, versions or system modules, and this often responds to the lack of a complete overview on the system. Also, complete overview cannot be achieved when technical knowledge (related to the old and new platform) and the domain knowledge are fragmented. In situations when only few people within the team or an organization count with this overview, they are often 'overloaded' with questions from their peers, making it difficult for them to perform their own tasks, in addition to the difficulties faced by those who do not have adequate knowledge to complete the tasks.

Side effects are natural in evolving systems (both at soft- ware and hardware levels). However, achieving adequate test data coverage to identify these side-effects constitutes a great challenge in modernization initiatives, in particular if the product is rapidly evolving, or if the development pace is very fast (as many times advocated

[2] http://www.ni.com/labview
[3] http://tinyurl.com/cpkwyef
[4] https://jersey.java.net

by agile). Often this was very difficult in transactional systems interacting with multiple external and third party systems, because in order to recreate an entire test scenario, it was required to 'reset' the values/states in those external systems as well. In many situations, combinatorial explosion of possible system states and the lack of understanding from the organization w.r.t. "where do we stand" in terms of test data coverage and adequacy constituted two critical drivers for 'unexpected surprises'.

This situation also relates to the case described by Stavru et al., [9] who states that the inherent complexity (in terms of third-party and external systems) in SOA represents a critical issue in migrations. Finally, impact analysis is extremely challenging because of the abovementioned complexity, which obscures the details on the implementation or logic of the system as a whole. The correctness criteria then becomes also difficult to assess, as also mentioned by Teppe [13], and also highly related to the previous discussion with regards to the lacking an adequate overview of the system.

C. Time Consuming Tasks

This challenge relates mainly to activities or tasks (mostly manually performed) that require a lot of time, or are deemed as very inefficient/difficult. Tasks involving technical details that are obscure, or tasks requiring developers to use un- known technologies are often time consuming. The lack of appropriate development infrastructure (e.g., servers take too long to deploy/update changes, inadequate debugging tools for JavaScript code) is also factors that lead to time-consuming situations.

In large organizations, developers often needed to navigate across the organization to for example, identify and clarify the correctness criteria that could enable them to implement and test a given functionality. This 'foraging' of details in order to complete a task can be often very time consuming. This situation manifested as well at the testers level, where the lack of clarity in descriptions of both functionality and defect reports made defect reproduction a time-consuming task.

Another challenge is to 'cut loose' from inefficient or sub- optimal designs rooted in the legacy systems that the new systems need to interact with. This task is not easy due to the dependencies between the new system and some of legacy modules that were yet not planned for migration. Often developers and architects are 'arm-bended' when sub- optimal solutions were forced upon them, later on causing different problems and time consuming restructurings. Finally, brittle task scheduling can also considered a driver for time- consuming tasks, as often developers/testers require longer time to re-learn the context (after not working on a given task/goal for a while) to complete the tasks.

D. Work Planning

Work Planning has to do with the sequence on which different tasks/implementation are conducted and the usage of different resources in the project at different points in time. Work Planning can be closely intertwined with delays in the project and time consuming tasks, and in some projects, this work planning was not optimal. There appears to be a minimum set of information that is needed before each sprint/iteration planning that would allow making better technical decisions when designing a solution. In those respects, the consolidation of the *required information* constitutes an essential part of this challenge. Such consolidation encompasses identifying information needs, identifying the key stakeholders, or sources of information; and should ideally be considered as a separate item in a project plan. Work planning in terms of allocation of resources constitutes an equally important aspect. In one particular project, it was possible to observe how the Project Manager, who was already overloaded with multiple roles, should have designated a 'functionality expert' with a technical background to 'forage' for critical information prior sprint/iteration planning meetings.

V. Lessons Learned

Some lessons can be extracted from the observations dis- cussed in the previous section. These are: (1) work planning needs to be adjusted to handle better information uncertainty, (2) estimation practices need to be fine-tuned, e.g., by explicitly allocating information foraging activities prior planning activities, (3) documentation and traceability cannot be neglected, (4) the phenomenon of 'role creep' should be avoided, and finally (5) clear processes need to be in place for the procurement of appropriate test data, and for enabling test automation.

A. Reworking of Solution

Lesson A.1 *– Early overview.* Effort should be focused on having a better idea on early stages of the project or even before the project starts what should be implemented, what are the dependencies, which are the main key players; identify the major obstacles, challenges, and risks. By identifying the dependencies timely, it is possible to agree and allocate resources accordingly. In several projects, the agile mantra of 'embracing the change' can lead to unnecessary fixes later in the project, and unwanted delays. Legacy technology experts are needed from day 1, whenever making decisions on the technical solution of the system. 'Active' assessments of the uncertainties and risks are definitively needed on top of the normal agile practices.

Lesson A.2 *– Let the right people in.* The right/relevant people should be involved in each sprint/iteration in a development cycle so that they could provide feedback on how to implement the solution/how to do the work. They should be involved before the sprint planning meeting to identify the concerns and aspects to be taken into account and to identify the risks. This largely reflects the particular needs or conditions imposed by modernization projects where information does not lay within the team, and external, obscure dependencies are imposed over the new systems.

Lesson A.3 *– Prepare enough well before each sprint/iteration planning.* A good idea in modernization initiatives is that an extra workshop/meeting before each sprint/iteration planning meeting was important, to invite

the relevant players (business and technical domain experts), to identify the challenges, responsibility areas, and risks of the upcoming sprint. An 'extra-step' within a sprint/iteration should be put in place, in order to assess the uncertainty and devise a plan to reduce and manage uncertainties and risks stemming from those.

B. Estimation Practices

Lesson B.1 – *What to do before estimation?* Under the context of modernization projects, there is need of better preparation before performing the estimation (make sure there is enough information from technical/business perspectives to make realistic estimates). For instance, tasks need to be better specified before the meeting as well as the breakdown of the user stories (or use cases). As mentioned previously, more focus is needed on identifying unknowns (data, components), and hidden cost drivers (e.g., extra activities). For instance, if technical debt is identified timely, this can help to identify risks during the estimation, and improve accuracy on the effort estimates for the sprint/iteration. Also, impact analysis should be incorporated for better estimations. Models/diagrams de- scribing the dependencies between the different components involved can facilitate understanding the implications/effects of the changes and to identify risks.

Lesson B.2 – *What to do during estimation?* Estimation sessions across different projects varied, some were long and inefficient, and somewhere fast and effective. The difference between the effective ones and the ineffective ones laid on the fact that the team members already understood the problem and required tasks fairly enough before the estimation meeting, therefore the task laid ahead constituted on achieving a group understanding and consensus on the effort required for each of the tasks. Beyond group consensus, the rationale for the estimation should also be somehow registered, as it could help to convey the top management the risks and unknowns beforehand. In that way, if delays occur, they would not be caught by surprise.

Lesson B.3 – *What to do after estimation?* Teams being self-reflective about their own estimates are very rare, and this situation is often related to inaccurate estimates, in particular when large or unclear functionality was implemented. Consequently, more focus should be placed on improving the actual estimates by evaluating the estimation accuracy after iterations, (e.g., has it been accurate and why yes/no?) and by identifying which kind of tasks are harder to estimate or more prone to inaccurate estimates.

C. Documentation and Traceability is Critical

Lesson C.1 – *Just talking to each other is not enough.* In situations where developers needed to 'forage' for information to perform their tasks, there was often no formal processes established for knowledge accumulation. Thus, things that were found during the project by one person were 'lost' or unknown for the rest of the team. In other situations, technical decisions may have been done during the implementation of a change, but they would not be documented properly in any common knowledge repository. If technical decisions remain undocumented, developers who later take over a given module would wonder on the rationale of the implementation choices. The lack of a protocol for knowledge consolidation, and complexity in terms of system and team size, as well as lack of code ownership could lead to fragmented knowledge across a migration initiative. On those terms, I will allow myself to quote a revelatory comment by a former colleague: "We communicated well, the problem is that we did not always document what we discussed".

This phenomenon has been reported partially by Ghobadi and Mathiassen [15], who describe 'Project setting barriers' as one of the context-related obstacles to knowledge sharing in agile projects. It is the opinion of the author of the present paper, that documentation and the knowledge management process needed to be examined and improved within migration initiatives, in particular those involving agile methods. For instance, if a task required updating some logic data, the corresponding change should be updated in Wiki, and there should be 'mechanisms' or routines in place for ensuring that. Similarly, technical decisions and choices taken on the implementation should be documented once the decision is made. This implies that a team would have to agree upon which guidelines to use, what a documentation should contain, and who is the responsible for the quality of the artifacts, etc.

Lesson C.2 – *Make it visual!* Another important aspect in migration initiatives is the use of visual aids depicting how the new system communicates with other systems (e.g., interface, dependency diagrams), and on complex business rules or business logic (e.g., state machine on the possible statuses of a product). These aids could be particularly helpful in situations where complexity is high from a component perspective (e.g., number of items, components involved), but not from a cognitive perspective (e.g., complex algorithms, calculations). Moreover, a protocol for creating/updating re- quirements and the adequate level of involvement of key actors in the requirements specification process are two key aspects that need to be focused upon.

Lesson C.3 – *Agree on a light-weight documentation protocol.* Finally, a protocol that includes keeps control over the quality of documentation over user tests, (by including more detailed) and defect reports seems essential in modernization initiatives. This last aspect is critical to allow developers to do resolve defects more independently without causing a communication overhead with the responsible(s) for testing.

D. The 'Role Creep' Phenomenon and Specialized Roles

Lesson D.1 – *Do not overload a role.* In several projects, there was a general perception that the project manager (or a person with a similar role) suffered of 'role creep' syndrome, in terms of having multiple responsibility areas besides the management of the project. In those situations in particular, the rest of the team would have very limited/restricted roles (e.g., limited to doing programming tasks). The complexity of teams or organizations often leads to a centralized

communication schema, often being the Project Manager, 'the central link'. This situation should not be problematic, as long as there is not an overload on multiple roles and responsibilities on a single person.

Lesson D.2 – *Maybe specialized/dedicated roles as 'feature manager'?* This role is considered of great importance in modernization projects, from the perspective of knowledge management (see for instance work by Pilat & Kaindl [16]), where there are many unknowns stemming from the legacy applications, and where information lays hidden across a complex organizational structure. A feature manager should ideally have technical background and be also in charge of knowledge management: speaking ahead with the right people, stating expectations on the data required to plan the project, etc. The feature manager should ideally count with technical savvy to generate/arrange adequate test data to ensure that all critical cases were covered during testing. An example of why this role is important is the following: A developer considered that the code was well tested due to high code (branch) coverage. However, data coverage was very low, as the code was only tested with a couple of different parameters. This situation occurred because the developer lacked of an overview of the possible data points for that particular functionality (that would require expertise in both business and technical domains), and lead to costly bug fixes.

Lesson D.3 – *Quality Monitoring and Testing.* A chain of responsibility should be assigned with respect to the quality of the different elements/parts involved in a project. This does not imply that the responsible should be fixing/correcting everything, but that he/she should make sure that it is done. For instance, one person could be in charge of reporting the quality of the overall system (test coverage, technical debt, etc.) The architecture team can for instance, keep an eye on how the system adheres to architectural standards established by the organization. They could provide guidance on how the solution can be implemented, without doing the actual implementation, and making sure that things are done properly. Also dedicated resources for testing is vital in modernization initiatives. Beyond unit tests, more sophisticated mechanisms are necessary to test complex scenarios, in particular when legacy systems and the new systems need to interact.

E. A Dequate Testing Methods and Test Data is Essential

Lesson E.1 – *Get adequate test data and test cases.* This aspect is closely related to the challenges mentioned in the previous section. Better processes for test data procurement and maintaining a stable test data is important. In situations where business logic is unclear, the universe of the possible parameters is unknown and potentially large. This situation is also well known in the context of agile modernization projects [17]. This reinforces the view on the importance of a person who can be made responsible for 'foraging' the right information in the organization, as expressed by a colleague: "*...If you have spoken to the right people, it could have been easier to get the right data for the test...however, the test data has gradually got better, in particular when we started getting fresh data from production systems, which were more realistic...*" Finally, a complete definition of test cases is critical (i.e., have we covered all the most important cases? what should be tested? how should be tested? what are the pre-conditions/post-conditions? what is the correctness criteria?).

Lesson E.2 – *Focus on better test infrastructure, explicit test protocol and test automation.* A better infrastructure for unit testing should be in place for modernization initiatives to test the behavior, and to make test code independent from implementation code (this, to avoid for instance test code refactoring after changes are made in the code).

Lesson E.3 – *Closer integration between testers/developers.* In many projects there was often a disassociation between developers and testers. People fulfilling the role of tester should be an integral part of the team, as there is a need of more feedback from testing to developers, and often testers understand well (or at least they should, in theory) the requirements to be tested.

VI. Conclusion and Future Work

Most of the projects used as basis for this synopsis used agile methods to a bigger or lesser extent. What was prevalent in all the projects was that intrinsically different working cultures with different knowledge or background had to interact. A primary lesson is that the requirements and knowledge management processes followed by agile teams did not always cope well with the level of uncertainty and continuous changes. Often in modernization projects, knowledge appears overly fragmented, constituting a challenge for both IT and business segments. In these situations, development processes (and maybe in particular agile methods) need to be properly adjusted to meet those challenges.

In particular, it seems, as the agile mantra of 'focusing on personal interaction and embracing the change' is rather difficult to embrace in projects focusing on legacy systems. A conjecture is that these mantras operate under the assumption that the team is compact (with a manageable size), that people know where the information lays within the organization, and that volatility is only latent at the requirements level[5]. When these assumptions are not applicable, the benefits of agile methods may not be fully reaped. Another assumption of agile is that the product owner is heavily involved in the process. This could be difficult in modernization projects, where multiple stakeholders are involved in the decision-making or when the resources allocated to the project are shared with the rest of the organization. Uncertainties (e.g., information missing), can lead to additional costs such as rework, introduction of defects, and delays. While these problems are not uncommon for other types of projects, the narrative on this work attempts to reflect that a naive/simplistic interpretation of agile principles i.e., high degree of flexibility towards change and lack of

[5] 'Requirements volatility' in contraposition to 'Fact volatility', where new pieces of information keep coming, changing the overall understanding of how different systems should interact.

focus on more formal processes (e.g., knowledge management, test data procurement) can potentially lead to, or aggravate these kind of issues.

Finally, *Work planning* was a recurrent challenge in the different projects, and mostly came intertwined with the issue of *knowledge management*. This activity is heavily dependent on the availability of correct and complete information, which is not always easy to achieve in modernization projects. The lessons learned from the observations reported can be summarized as:

- Focus on achieving an adequate understanding (and overview) of both the business rules and the most appropriate technical solution(s).
- Adequate technological frameworks and a common, agreed protocol to consolidate knowledge need to be in place.
- Identify uncertainties, dependencies and potential risks, to enable the allocation of resources at appropriate times in the project.
- Specify roles to avoid the phenomenon of 'role or responsibility creep'.
- Appoint a 'Feature Manager', i.e. a person with technical background, who could also build expertise and make explicit the business rules of the product or organization (e.g., by documenting the rules).
- Improve the identification of cost drivers and estimation inaccuracies in projects with high degrees of uncertainty.

A final observation of the author is that that within agile projects, there may be a latent 'fear' for relatively more formal or structured processes (perhaps stemming from the current mainstream understanding of agile, where formal processes are perceived as cumbersome and slow). Extremes of this fear could potentially hinder successful usage of agile in modernization projects. Hence, agile teams need to strive for an adequate balance between the degree of flexibility and formalism, as two sides of the same coin.

Future work consists of building a consolidated list of process-related software modernization challenges by means of literature screening (in particular focusing on requirements engineering and testing), validating those via surveys involving different constituent groups of software practitioners, and suggesting more concrete guidelines on how particular practices (e.g., SCRUM, Kanban, EVO, etc) could be calibrated in order to cope better with those challenges.

ACKNOWLEDGMENT

The author thanks all the anonymous colleagues and managers, with whom she could collaborate and discuss in the different projects, and who shared their impressions/insights. Their insightful discussions helped in a great extent to distill the present work.

REFERENCES

[1] F. Barbier and J. L. Recoussine, *Software Modernization: Technical Environment*, Hoboken, NJ, USA: John Wiley & Sons, Inc., Jan. 2015.

[2] R. C. Seacord, D. Plakosh, and G. A. Lewis, *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[3] W. M. Ulrich and P. H. Newcomb, *Information Systems Transformation Architecture-Driven Modernization Case Studies*, ser, The MK/OMG Press, W. M. Ulrich and P. H. Newcomb, Eds. Boston: Morgan Kaufmann, 2010.

[4] S. Tilley, J. Gerdes, T. Hamilton, S. Huang, H. Muller, and K. Wong, "Adoption challenges in migrating to web services," in *Proc. Fourth International Workshop on Web Site Evolution*. IEEE Comput. 2002, pp. 21–29.

[5] A. Erradi, S. Anand, and N. Kulkarni, "Evaluation of strategies for integrating legacy applications as services in a service oriented architecture," in *Proc. 2006 IEEE International Conference on Services Computing*, IEEE, Sep. 2006, pp. 257–260.

[6] P. Mohagheghi and T. Sæther, "Software engineering challenges for migration to the service cloud paradigm: Ongoing work in the REMICS Project," in *Proc. 2011 IEEE World Congress on Services*. IEEE, July 2011, pp. 507–514.

[7] M. A. Babar and M. A. Chauhan, "A tale of migration to cloud computing for sharing experiences and observations," in *Proc. the 2nd international workshop on Software engineering for cloud computing - SECLOUD '11*. New York, New York, USA: ACM Press, 2011, pp. 50-56.

[8] J. Alonso, L. Orue-Echevarria, M. Escalante, J. Gorronogoitia, and D. Presenza, "Cloud modernization assessment framework: Analyzing the impact of a potential migration to Cloud," in *Proc. 2013 IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems, MESOCA 2013*, 2013, pp. 64–73.

[9] S. Stavru, I. Krasteva, and S. Ilieva, "Challenges for migrating to the service cloud paradigm: An agile perspective," *Lecture Notes in Computer Science*, vol. 7652, pp. 77–91, 2013.

[10] F. Fleurey, E. Breton, B. Baudry, A. Nicolas, and J. M. Jezequel, "Model-Driven engineering for software migration in a large industrial context," *Model Driven Engineering Languages and Systems*, vol. 4735, pp. 482–497, 2007.

[11] M. F. Wendland, M. Kranz, C. Hein, T. Ritter, and A. Garcıa Flaquer, "Model-based testing in legacy software modernization: an experience report," in *Proc. the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation - JAMAICA 2013*, New York, 2013, pp. 35-40.

[12] O. S. Ramon, J. S. Cuadrado, and J. G. Molina, "Model-driven reverse engineering of legacy graphical user interfaces," *Automated Software Engineering*, vol. 21, no. 2, pp. 147–186, Apr. 2014.

[13] W. Teppe, "The ARNO project: Challenges and experiences in a large-scale industrial software migration project," in *Proc. 2009 13th European Conference on Software Maintenance and Reengineering*. IEEE, 2009, pp. 149–158.

[14] D. M. Geary. *Graphic Java 2: Swing*, ser. Graphic Java 1.2. Sun Microsystems Press. 1999. [Online]. Available: https://books.google.no/books?id=4N1MGOsu1jEC

[15] S. Ghobadi and L. Mathiassen, "Perceived barriers to effective knowledge sharing in agile software teams," *Information Systems Journal*, vol. 26, no. 2, pp. 95-125, 2016.

[16] L. Pilat and H. Kaindl, "A knowledge management perspective of requirements engineering," in *Proc. Fifth International Conference on Research Challenges in Information Science*, IEEE, May 2011, pp. 1–12.

[17] B. Boehm and R. Turner, "Management challenges to implementing agile processes in traditional development organizations," pp. 30–39, 2005.

Aiko Yamashita holds an Associate Professorship at Oslo and Akershus University College of Applied Sciences. She has worked in Costa Rica, USA, Sweden and Norway in diverse organizations during the last 10 years. Her research interests include empirical software engineering, software quality, IT sustainability, and processes for high-tech transfer and innovation initiatives.