AWSM
Agile Web Migration for SMEs

Sebastian Heil and Martin Gaedke
Technische Universität Chemnitz, 09107 Chemnitz, Germany

Keywords: Software Migration, Web Engineering, Agile Development.

Abstract: Migrating legacy desktop to web applications is an important and challenging task for SME software companies. Due to their limited resources, migration should be integrated in ongoing development processes. Existing research in this area does not consider recent paradigm shifts in web development. Therefore, our work is dedicated to supporting SME software providers in migrating to modern web applications while integrating this into ongoing development. This paper outlines our idea and presents a roadmap towards achieving this goal.

1 INTRODUCTION

Driven by the high number of ways in which users interact with recent web applications, changing user expectations pose new challenges for existing non-web software systems. Continuous evolution of technologies and fading away of support for obsolete technologies furthermore intensify the pressure to renew these systems. As web browsers are becoming the standard interface for many applications, web applications provide a solution to platform-dependence and deployment issues (Aversano et al., 2001). Existing software systems have been developed over a long time and contain valuable knowledge about models, rules and solutions in their application domain. Thus, it is often not practical to replace them by a newly developed web application. Instead, it is necessary to transform legacy into new software keeping the existing knowledge. This is the objective of software migration.

Existing migration approaches are hardly feasible for small and medium-sized enterprises (*SME*s), cannot be run in parallel to and combined with ongoing agile development and do not consider the specifics of web application development. Development teams of SME software providers are occupied with development and maintenance of existing applications. As the resources are rather limited, there is no development team available to exclusively dedicate to migration (Horowitz, 1998). Agile development is widely employed, but there is no approach which allows migration to be integrated into day-to-day agile development activities. Interactivity and ubiquitous mobile access require migration towards web applications. However, existing approaches do not consider web characteristics such as distributedness, statelessness, hypermedia, event-orientation, responsiveness etc. Also, recent advances in Web Engineering like re-use oriented composition approaches are not regarded.

Imagine e.g., a patient management system. This software is run as desktop application at doctors' offices. For adding interactivity like online appointment scheduling, migration to web is required. However, due to continuously changing regulations, SME development teams are busy updating and maintaining the system. This is where our approach for Agile web migration for SMEs (AWSM) comes into play.

As there are many SME software providers with legacy software and user expectations require transformation to modern web applications, enabling them to successfully perform this migration within given constraints will impact the competitiveness of this sector.

We outline AWSM in Section 2. We position our approach to related work in Section 3 and conclude the paper with a roadmap in Section 4.

2 THE AWSM APPROACH

As shown in Figure 1, AWSM contributes to four areas. For Requirements & Knowledge Discovery, we propose a meta-model of knowledge, an annotation platform and discovery tools. Migration Planning & Implementation are supported by a component-based target architecture, component repository and hybrid web applications. Requirements & Knowledge Discovery and Migration Planning & Implementation are

Heil, S. and Gaedke, M.
AWSM - Agile Web Migration for SMEs.
In *Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering (ENASE 2016)*, pages 189-194
ISBN: 978-989-758-189-2

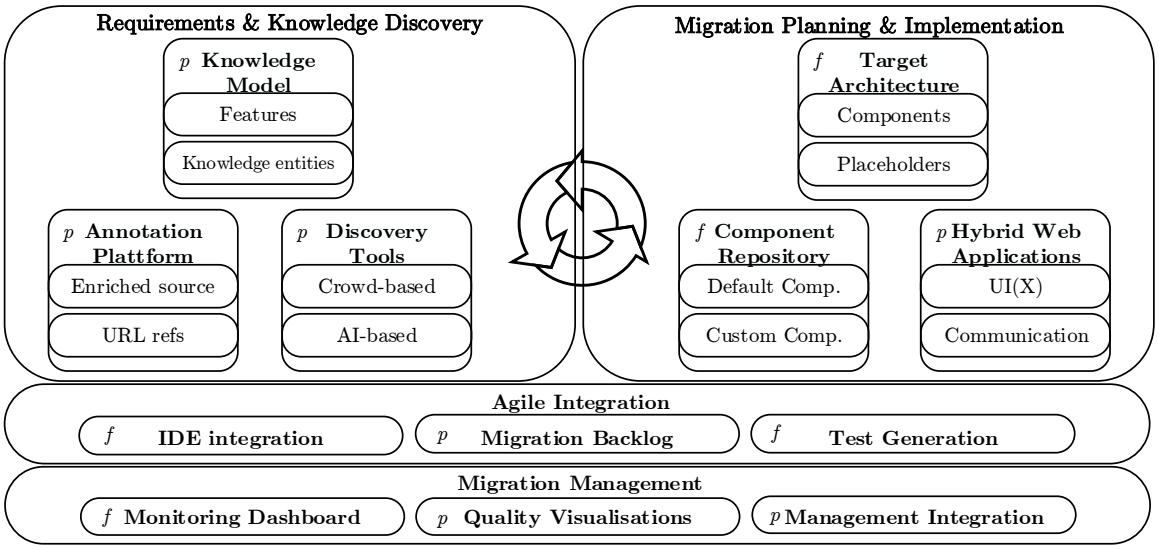

Figure 1: AWSM Overview.

iterative. Agile integration and migration management intersect both areas. In Figure 1, a *p* denotes work in progress and an *f* denotes future work.

Simple transformation approaches are not sufficient, as migrating to the web requires a fundamental paradigm shift involving a distributed, multi-language target environment. Thus, assisted re-development taking into account these characteristics is required. Here, the most important challenge is to avoid loosing existing knowledge. Therefore, at this early stage we focus on Requirements & Knowledge Discovery.

Requirements & Knowledge Discovery. Legacy system understanding is an art, but a necessary first step (Horowitz, 1998). To discover, document and prepare the knowledge which is implicitly contained in legacy source code for migration AWSM introduces a *knowledge model*. Existing work, distinguishes three types source code: presentation/user interface, application logic and database access/persistence (Canfora et al., 2000; Horowitz, 1998; Ping et al., 2003). OMG's Knowledge Discovery Meta-Model (KDM)[1] specifies a meta-model for software systems. The source package of KDM describes source code files and areas in the source code. However, KDM does not provide a model for expressing the knowledge contained in the source itself. This is too coarse-grain and therefore we work on an ontology for knowledge in source code. The knowledge ontology will be described using OWL[2] and compatible with KDM.

Legacy system understanding is about understanding what the software does and how the software does it (Horowitz, 1998). Thus, our current draft distinguishes between two types of *code information*: features and knowledge entities. *Features* refer to code which implements functional requirements that are directly visible to users. *Knowledge entities* comprise the traditional categories of presentation, application logic and persistence, but also business rules, configuration, deployment, algorithms and explanations in comments.

Based on the knowledge ontology, the challenge is to support developers to discover and document the knowledge and to make it usable for migration. Comprehensive re-documentation approaches like (Corbi, 1989; Kazman et al., 2003) are neither feasible for SMEs nor can they be integrated into day-to-day agile development activities. Based on (Rivero et al., 2014), we introduce the concept of annotations to this problem. Annotations can be applied as an additional layer of description over the code. An annotation expresses that a selected area of source code is related to a feature or knowledge entity. Different code information can be annotated independently and can overlap. One code information can appear in several places in the code, represented by several annotations. AWSM supports developers by providing an *annotation platform* which allows to enrich code with annotations. Unlike traditional re-documentation approaches, our web-based annotation platform provides a URL for every annotation and knowledge entity. During re-development, this enables developers to refer to the knowledge, e.g. in emails, wikis, task descriptions etc., and jump to their definition and location in the source. The discovered features define the requirements for re-development. A screenshot of the annotation platform can be seen in Figure 2.

[1] http://www.omg.org/spec/KDM/1.3/
[2] http://www.w3.org/TR/owl2-overview/

To lower effort for the annotation work, we are currently investigating possibilities of executing this classification task by means of machine learning and crowdsourcing. Our *discovery tools* are tested and their effectiveness is evaluated in cooperation with our industry partner, a software development SME.

Migration Planning & Implementation. Migration planning defines the *target architecture* based on components, focusing re-use and maintainability. Components can combine all three application tiers, but also represent only one such as user interface components. Each component is described in terms of features it provides.

The *component repository* contains a set of generic components, such as displays, editors and lists of data entities. If no suitable component is available, the architecture uses a placeholder component requirements description.

Hybrid web applications are an intermediate migration step. They consist of both newly implemented components in HTML, CSS Javascript and also of slightly adapted pieces of legacy code. Technologies like Native Client[3] or Emscripten[4] allow to execute legacy source code within the browser with only minor changes. While migration to a ”pure” web application is the objective, a hybrid web application can decrease the initial hurdle to migrate.

Agile Integration. The work of annotating a piece of code can easily be done while working on this code. When reading code during regular development or maintenance activities and noticing valuable knowledge, it can be annotated and thus be made explicit. This is in line with the ”refactor as you touch code” policy devised by many companies employing agile development. To make this very easy for developers, we will integrate our platform with an IDE.

Based on the annotated source, a *migration backlog* consisting of migration stories is used as implementation plan. It is set up from implementation stories for placeholder components and integration stories for connecting the components as defined in the target architecture. To integrate with regular development activities, migration stories are successively included into iteration planning such as Scrum sprint backlogs. To ensure functional equivalence we will apply test generation approaches. From the re-discovered features of the legacy system, acceptance *tests* for the web-based system are created.

Migration Management is important in enterprise contexts. We provide a *dashboard* to monitor the progress of the migration. The annotated source allows for analysis and *visualization* of structure and quality

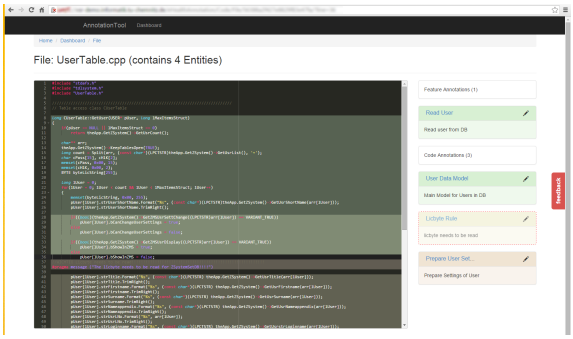

Figure 2: AWSM Annotation Platform.

issues. For instance, we display the interlacing of different features occurring in the same locations using sankey and chord diagrams. Our migration platform will be integrated with development management tools like Team Foundation Server. A set of migration stories forming a migration package can then be moved into the development management tool. This allows to implement them like stories within the SME’s regular development process.

A live demonstration of our migration tool can be found at[5].

3 RELATED WORK

There is some work in the area of web migration. Code-oriented approaches like (Aversano et al., 2001) analyse existing systems by call-graphs and decouple architecture layers. Using a call-graph based decomposition, the presentation layer is decoupled from the legacy system, reverse engineered and redeveloped with web technologies. Legacy Application logic and database components are left unchanged and wrapped, presentation is migrated to HTML/CSS. The authors state that this can only be a short-term solution as it does not improve structure and maintainability of the system. It does not focus on improving end-user visible features. For instance, the resulting UI is very similar to the legacy UI. It disregards current web design, UIX and navigation patterns and responsiveness. Also, nowadays web browsers are no longer restricted to handling only presentation. Enabled by technologies like Web Workers and client-side MVC frameworks, more application logic is moved to the browser.

(Horowitz, 1998) describes web migration in three steps: understanding the legacy software, designing the architecture and testing and tuning. Distribution of application logic to client and server is addressed. In the architecture design, the decision of what to exe-

[3]http://gonacl.com/

[4]https://github.com/kripken/emscripten

[5]https://vsr.informatik.tu-chemnitz.de/demos/awsm

cute on the server or client side is essential. Besides UI, client side issues include how much application logic is performed on the client and how the communication between client and server is achieved. The resulting web application is based on HTML and Java on the client side. The main focus is on communication in distributed applications via Corba, DCOM or RMI. Since then, the web application environment has changed. Browser-plugin-based technologies are vanishing. Components with communication via Corba, DCOM or RMI are superseded by services and SOAP or RESTful APIs.

Approaches like (Canfora et al., 2000) deal with program decomposition as preliminary step for migration. Three types of components are distinguished: user interface, application logic and database components. For migration to a client-server architecture, user interface components are migrated to the client, database components to the server and application logic can be on both. The condition is that the legacy system is structured in distinct subroutines of the three types and database is not calling UI. Like most program decomposition algorithms, (Canfora et al., 2000) exploits graph representations. The slicing algorithm identifies UI components based on control flow information in a system graph colored according to the types. An interactive tool allows software engineers to mark application logic statements and extracts a new subroutine for the server. An optional description of the corresponding business rule can be added. Our approach distinguishes code at a higher granularity than the three types found in most publications. For identification, we investigate human and machine learning classification, not only for UI. The interactive tool is similar, but our annotation tool allows to reference annotated code information by its own URL.

(Gu and Lago, 2010) presents a survey of service identification methods. Inputs can be knowledge at enterprise level such as business processes, requirements, goals etc. or existing systems in terms of source code, data etc. Only few approaches combine both. The target output can be distinguished by service type. Services are grouped into business or IT perspective services. A wide range of strategies for service identification exist. Among less common approaches are component-based and UI-based methods, UI-based are regarded as innovative. The knowledge re-discovery of AWSM uses a combination of knowledge on enterprise level and existing systems. To create the target architecture, AWSM employs a component-based strategy in combination with UI-based.

(Ping et al., 2003) acknowledges the rapid changes in web technologies and presents a transformation framework for adapting existing web applications to MVC web applications. The approach has three phases. First database and presentation are separated by extracting db functionality and encapsulating it into java beans using syntax analysis. Then, presentation is transformed from HTML to JSP. In a last step, a controller-centric architecture is achieved. The focus is on database access encapsulation and control flow transformation whereas AWSM addresses all aspects. This work starts from static HTML web pages and migrates to MVC web applications in a J2EE context. By contrast, AWSM support migration of non-web legacy applications to web applications.

Even though the development of web applications has changed significantly – consider e.g. REST, client-side MVC, mobile devices etc. – research has not addressed this. Instead, the focus of web migration has shifted. Recent areas of interest include migration to service-oriented architecture (SOA) (Khadka et al., 2013), migration to cloud environments (Krasteva et al., 2013) and evolution of web systems (Kienle and Distante, 2014). Surveys like (Khadka et al., 2013; Razavian and Lago, 2010) provide an overview of research in SOA migration. It comprises general phases like legacy and target system understanding as well as SOA specific ones such as service identification. Service identification and identification of service rich areas is similar to component identification in AWSM. Open challenges in SOA migration include definition of a common process model, automation, language independence and decomposability. While we can build on the findings in general migration phases, migration towards interactive web applications differs. A mayor difference lies in the impact of migration on end users. Migration to SOA can be accomplished in the backend without end users noticing. In contrast, migrating legacy GUI applications towards interactive web applications impacts the way in which end users use the system.

Approaches like (Krasteva et al., 2013; Babar and Chauhan, 2011) focus on migration to cloud environments. REMICS (Krasteva et al., 2013) proposes the application of Model-Driven Modernization to the Software-as-a-Service delivery model. As AWSM, REMICS is based on OMG's KDM(Object Management Group, 2011). It focuses on migration process by defining seven activity areas such as requirements, recovery, migration and validation. Distinction between requirements and recovery of knowledge - traditionally combined as legacy analysis - can also be found in AWSM. (Babar and Chauhan, 2011) reports on experiences in migration of a web application to cloud. It focuses on architecture modifications required by the cloud environment. Two main aspects are stated. Separation of persistence and business logic facilitates

usage of cloud-provided storage services. Implementation of an orchestration layer is necessary when components are distributed. Moreover, loose coupling of components via HTTP-based APIs provides the possibility to move components to different cloud providers.

There are several approaches for running legacy C/C++ code in the browser. Google Native Client by providing a runtime environment (Yee et al., 2009; Donovan et al., 2011), Emscripten by compiling into JavaScript subset asm.js (Zakai, 2011). WebAssembly aims at specifying a common bytecode for the web. In contrast to previous attempts, it seeks wide browser support without requiring plugins. Building on experience from Native Client and asm.js, it is a promising joint effort of all four major browser-providing companies (W3C WebAssembly Community Group, 2015). These approaches bear good potential for use in migration as they allow to run legacy code in a web environment. While this does not result in a true web application, it can be an intermediate step and lower the barrier to entry for migration. We are therefore evaluating their migration capabilities and requirements.

Agile development has been applied to the software migration domain. STDM (Abbattista et al., 2009) combines a user-story-based iterative process with automated acceptance testing. User stories and story tests take the place of traditional requirements documents. This improves sytem understanding and is a starting point for the migration plan. Based on user stories of the legacy system, acceptance tests for legacy and new system are created. The legacy story tests are iteratively migrated to the new system's environment. AWSM builds on this work. We also use annnotated features represented by user stories as a starting point for agile migration Also, STDM does not support feature discovery in legacy code. As stated by the authors, STDM acceptance tests are only reusable if legacy and target environment allow. While STDM was tested on a small-scale Java to Java, Web to Web migration, AWSM targets change in both programming language and paradigm. Legacy acceptance tests are therefore not reusable and are therefore left out in favor of acceptance tests for the new system. In (Krasteva et al., 2013), the authors describe an agile extension of REMICS. The process is based on Type-C Scrum and consists of several overlapping scrums for REMICS activity areas. Also, influences from XP, such as pair programming, are employed. In contrast to AWSM, (Abbattista et al., 2009; Krasteva et al., 2013) are agile migration methodologies that require entire teams dedicated to migration. AWSM, however, seeks to integrate migration work items into ongoing agile development. (dos Santos et al., 2013) highlights the importance of visualizing technical debt for communication within teams, with management. The dashboard in our approach follows the same idea.

Approaches like (Rivero et al., 2014) demonstrate how agile requirements engineering can be combined with model-driven techniques. The idea is to build on existing requirement artifacts, UI mockups in this case, and add an additional layer of description by enriching them with annotations. In AWSM we apply this idea to migration. The requirement artifact is the source code, annotations are added to describe migration-relevant properties. In (Rivero et al., 2014), RESTful API prototypes are then generated from standard components. This is similar to AWSM, but our components comprise all layers of the application.

4 CONCLUSIONS AND ROADMAP

To conclude, existing migration approaches are not feasible for SMEs, cannot be integrated with ongoing agile development and do not address characteristics of current web application development. Therefore, we outlined the scope and detailed areas of work for supporting SME software providers in migrating to modern web applications while integrating this into ongoing development. We briefly reported on the current state and presented the plan for future work.

In addition to the parts marked with *p* in Figure 1, we are currently conducting a structured literature review (SLR) of web migration approaches to provide a systematic overview of existing approaches and challenges. After prototypical completion of the three solution parts in requirements and knowledge discovery, the next step on our roadmap is to evaluate the tools with our industrial partner. We will observe how the tools are used by the developers, identify problems and iterate over the prototypes. Next step will be to improve the agile integration and migration management along with developers and product management. Based on the feedback, we will adapt our solution architecture and then address the parts marked with *f* in Figure 1, in particular focusing on migration planning and implementation.

ACKNOWLEDGEMENTS

This research was supported by the eHealth Research Laboratory funded by medatixx GmbH & Co. KG.

REFERENCES

Abbattista, F., Bianchi, A., and Lanubile, F. (2009). A storytest-driven approach to the migration of legacy systems. In Abrahamsson, P., Marchesi, M., and Maurer, F., editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 31 of *Lecture Notes in Business Information Processing*, pages 149–154. Springer Berlin Heidelberg.

Aversano, L., Canfora, G., Cimitile, A., and De Lucia, A. (2001). Migrating legacy systems to the Web: an experience report. In *Proceedings Fifth European Conference on Software Maintenance and Reengineering*, pages 148–157. IEEE Comput. Soc.

Babar, M. A. and Chauhan, M. A. (2011). A tale of migration to cloud computing for sharing experiences and observations. In *Proceeding of the 2nd international workshop on Software engineering for cloud computing - SECLOUD '11*, page 50, New York, New York, USA. ACM Press.

Canfora, G., Cimitile, A., De Lucia, A., and Di Lucca, G. a. (2000). Decomposing legacy programs: a first step towards migrating to client–server platforms. *Journal of Systems and Software*, 54(2):99–110.

Corbi, T. (1989). Program understanding: Challenge for the 1990s. *IBM Systems Journal*, 28(2):294–306.

Donovan, A., Muth, R., Chen, B., and Sehr, D. (2011). Pnacl: Portable native client executables.

dos Santos, P., Varella, A., Dantas, C., and Borges, D. (2013). Visualizing and managing technical debt in agile development: An experience report. In Baumeister, H. and Weber, B., editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 149 of *Lecture Notes in Business Information Processing*, pages 121–134. Springer Berlin Heidelberg.

Gu, Q. and Lago, P. (2010). Service Identification Methods: A Systematic Literature Review. In Nitto, E. D. and Yahyapour, R., editors, *Towards a Service-Based Internet*, volume LNCS 6481, pages 37–50. Springer Berlin Heidelberg.

Horowitz, E. (1998). Migrating Software To The World Wide Web. *IEEE Software*, 15(3):18–21.

Kazman, R., Brien, L. O., and Verhoef, C. (2003). Architecture Reconstruction Guidelines, Third Edition. Technical Report November, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Khadka, R., Saeidi, A., Idu, A., Hage, J., and Jansen, S. (2013). Legacy to SOA Evolution: A Systematic Literature Review. In Ionita, A. D., Litoiu, M., and Lewis, G., editors, *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments*, chapter 3, pages 40–71. IGI Global.

Kienle, H. M. and Distante, D. (2014). Evolution of Web Systems. In Mens, T., Serebrenik, A., and Cleve, A., editors, *Evolving Software Systems*, chapter 7, pages 201–228. Springer Berlin Heidelberg, 1 edition.

Krasteva, I., Stavru, S., and Ilieva, S. (2013). Agile Model-Driven Modernization to the Service Cloud. In *Proceedings of The Eighth International Conference on Internet and Web Applications and Services (ICIW 2013)*, pages 1–9, Rome, Italy. Xpert Publishing Services.

Object Management Group (2011). Architecture-Driven Modernization: Knowledge Discovery Meta-Model (KDM).

Ping, Y. P. Y., Kontogiannis, K., and Lau, T. (2003). Transforming legacy Web applications to the MVC architecture. *Eleventh Annual International Workshop on Software Technology and Engineering Practice*.

Razavian, M. and Lago, P. (2010). A Frame of Reference for SOA Migration. In Di Nitto, E. and Yahyapour, R., editors, *Towards a Service-Based Internet*, volume LNCS 6481, pages 150–162. Springer Berlin Heidelberg.

Rivero, J. M., Heil, S., Grigera, J., Robles Luna, E., and Gaedke, M. (2014). An extensible, model-driven and end-user centric approach for api building. In Casteleyn, S., Rossi, G., and Winckler, M., editors, *Web Engineering*, volume 8541 of *Lecture Notes in Computer Science*, pages 494–497. Springer Berlin Heidelberg.

W3C WebAssembly Community Group (2015). Webassembly design: Minmal viable product.

Yee, B., Sehr, D., Dardyk, G., Chen, J. B., Muth, R., Ormandy, T., Okasaka, S., Narula, N., and Fullagar, N. (2009). Native Client: A Sandbox for Portable, Untrusted x86 Native Code. In *2009 30th IEEE Symposium on Security and Privacy*, pages 79–93. IEEE.

Zakai, A. (2011). Emscripten: an LLVM-to-JavaScript compiler. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 301–312.