ReWaMP: Rapid Web Migration Prototyping Leveraging WebAssembly

Sebastian Heil(✉), Valentin Siegert, and Martin Gaedke

Technische Universität Chemnitz, Chemnitz, Germany
{sebastian.heil,valentin.siegert,
martin.gaedke}@informatik.tu-chemnitz.de

Abstract. Web Migration is a challenge, in particular for Small and Medium-sized Enterprises (SMEs). In previous collaborations with SMEs we noticed an initial resistance to migrate legacy desktop applications to the web, due to concerns about the risk and lack of developers with web expertise . This initial hurdle can be mitigated by the ability to rapidly create running web prototypes based on the existing desktop codebase and expertise of the developers. Therefore, we outline a rapid prototyping approach for Web Migration and present a solution architecture, process and supporting infrastructure based on WebAssembly. We describe challenges and report on an experiment applying WebAssembly on a scenario desktop application derived from real-world industrial code.

Keywords: Web migration · Prototyping · WebAssembly
Software reuse

1 Introduction

Migration of legacy systems (LS) to the Web is an important challenge for software developing companies. On the one hand, user expectations, advantages of platform-independent deployment [2] and lack of long term support for obsolete technologies provide a rationale for renewing legacy software as web applications. Though aware of these reasons for web migration (WM), Small and Medium-sized Enterprises (SMEs) find it difficult to commence a WM [4]. In several previous projects in collaboration with SMEs, we noticed initial resistance to migrate legacy desktop applications to the web.

Using LFA problem trees, we identified two main problems that contribute to the notion of risk which keeps SMEs from commencing a WM: *doubts about the desirability* and *doubts about the feasibility*. Desirability concerns are rooted in a lack of knowledge of current web technologies and thus possibilities and advantages. Feasibility concerns are mainly due to a lack of employees with web development expertise in companies which have focused on the development of desktop applications. Solving this problem means to demonstrate desirability and feasibility to these companies.

T. Mikkonen et al. (Eds.): ICWE 2018, LNCS 10845, pp. 84–92, 2018.
https://doi.org/10.1007/978-3-319-91662-0_6

To address similar concerns in (forward) web engineering, prototyping techniques are common. Widely adopted Agile methodologies have promoted prototype-first development. Influenced by Design Thinking methods [5], *rapid prototyping* and iterative development are used to involve stakeholders and gather early feedback. The prototypes allow to explore the solution space in the two dimensions of desirability and feasibility [5]. To demonstrate technical feasibility of the solution idea, required technologies and frameworks are explored in the prototype. Also, the prototype demonstrates core features of the solution idea. Thus, software prototypes are used as a means of communication, present main characteristics of the solution to stakeholders and allow to interactively gather feedback. In this paper, we explore how the rapid prototyping paradigm can be transferred from forward web engineering to the field of web migration.

The key characteristic of rapid prototyping is limited time and effort. Existing prototyping methods like [9] can be re-used to support parts of this process which require development from scratch. To meet the effort and time requirements of rapid prototyping, re-use of as much of the legacy code as possible is crucial. Technologies like Google Native Client, emscripten or especially WebAssembly[1] allow to run non-JavaScript programs in the browser. A re-use focused rapid prototyping approach can be primarily performed with existing staff experienced in the legacy technology base. We describe our approach in Sect. 2, detail our WebAssembly-based method in Sect. 3, report on our findings in a validation experiment in Sect. 4, provide an overview on related work in Sect. 5 and conclude with a roadmap in Sect. 6.

2 Approach

To address the initial resistance described in Sect. 1, we propose the *Rapid Web Migration Prototyping (ReWaMP)* approach (cf. Fig. 1) for enabling web engineers to quickly create running web application prototypes from legacy code.
The **legacy code** is the source code of the LS which should be transformed into a web application. ReWaMP assumes that the legacy source code is available, which typically is the case in enterprise software development contexts, where legacy systems are still actively operated, maintained and even developed.

UI mockups are visual sketches of the user interface. They allow to get early feedback from the target users on the understandability and information structuring of the user interface and can help to identify problems in the process or the conceptual abstractions of the underlying domain model. Mockups are traditionally created manually, as hand-drawn rough sketches. Since the LS is still a running software, UI mockups can be created from screenshots of the legacy UI with minimal effort in the first iteration of our approach. Over time, the screenshot mockups can be replaced and incrementally improved towards a modern web UI, taking into account similarity aspects in order to smoothly transition from the legacy layout to a potentially different web UI layout [3].

[1] http://webassembly.org/.

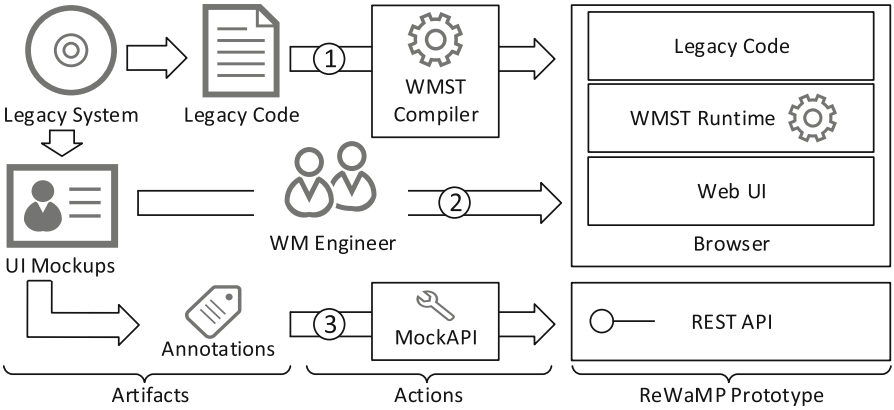

Fig. 1. ReWaMP overview

Based on the initial artifacts legacy code and UI mockups, in ReWaMP, the migration engineer (ME) creates the web application prototype by three **actions**:

1. Leveraging a suitable web migration support technology (WMST), he creates an adapted version of the legacy code artifact which can be run in the browser.
2. Based on the UI mockups, he implements a web UI using HTML, CSS & JS.
3. By annotating the UI mockups, he generates a REST API backend (cf. [9]).

This paper focuses on action 1. We provide details on this action in Sect. 3. For action 2 we are currently investigating automatic transformations of legacy UIs to web-based UIs, from absolute layouts to grid-based layouts under similarity constraints [3] using optimization algorithms. Action 3 integrates our previous work [9] from forward web engineering contexts into this migration prototyping scenario. The resulting prototype combines legacy code with a generated REST API and a newly developed web UI. It demonstrates feasibility and desirability of a WM with limited effort. From this interim stage, the application can be incrementally transformed into a real web application by migrating the business logic contained in the legacy code towards client- or server-side web programming platforms. The architecture allows re-use of legacy view behavior code at the client side, in related work the re-use at server side has been described [7]. Thus, ReWaMP is a rapid prototyping approach for WM.

3 WebAssembly-Based Rapid Web Migration Prototyping

WebAssembly (WASM) is an open standard currently designed by the W3C of a format for compilation to the web aiming at portability and size- and load-time-efficiency. The group includes all major browser providers. WASM combines experience from previous work on emscripten and Native Client. Using parts

of the emscripten toolchain, C/C++ code can be compiled to WASM. As of 01/2018, WASM has reached cross-browser consensus with preview implementations in all major browsers. In assessment of WMSTs, WASM has the fewest limitations, is compatible with major platforms, is the most current and actively developed technology and sees wide support from major companies due to being an open standard. Therefore, we showcase in this section our prototypical implementation of a ReWaMP toolchain based on WASM for action 1 of Sect. 2. For this, we describe the architecture of the resulting web migration prototype and detail the ReWaMP process.

3.1 Architecture of ReWaMP Prototypes

The client side of ReWaMP prototypes consists of:
HTML/CSS. The HTML/CSS component describes the layout and content structure of the Web UI as created in action 2 of Fig. 1.

WASM. The WASM Code component is derived by compilation of the legacy view behavior code to WASM. For each user interface, the corresponding WASM file includes all UI functionality and semantics. To enable communication with other components, it is bound to the JS glue code.

JS Glue Code. The JS glue code is created at the compilation of legacy code to WASM by the emscripten compiler. It provides the JS API of WASM, and thereby standard functionalities like WASM memory management and serialization of strings for message exchange.

ReWaMP JS Code. The ReWaMP JS code provides an infrastructure for the communication with the server side and for abstracted access to the generated Web UI. It therefore provides web migration prototyping-specific API extensions for the JS API of WASM required to control the behavior from legacy code compiled to WASM.

3.2 ReWaMP Process

The ReWaMP process (cf. Fig. 2) defines the necessary steps for creating a web migration prototype according to the architecture introduced above. It creates the WASM Code, JS Glue Code and ReWaMP JS Code for each view of the legacy system. This is achieved through extraction of relevant information from the legacy code, transformation in a WASM-compilable version and compilation to the WASM target. While the compilation of C++ code to WASM is supported by emscripten, the transformation is not trivial. To support the rapid creation of WASM files, we developed a prototypical WASM Transformator (WASM-T) for legacy C++ code bases using the MFC framework. The responsibilities of the migration engineer and the WASM-T are represented as lanes in Fig. 2.

To start, the ME (1) extracts the source files describing view behavior from the legacy code and (2) configures WASM-T, specifying file location and the main file per view. Comparable to main functions in C++, main files are those which

directly interact with a dialog or a window, containing event handlers for UI elements. To make it compilable to WASM, WASM-T transforms the extracted legacy code. Since automated semantic analysis of source code is complex and the code can reference dependencies which are not available (e.g. binaries), a semi-automated process is required. In (3), WASM-T performs pre-processing by deleting/rewriting GUI framework-specific parts of code and extracting UI coupling information from the legacy code for introducing the Web UI.

After pre-processing, the ME re-engineers code parts WASM-T was not able to transform. He does that either via expert changes in the code (4) or within a separate expert file (5). Expert changes replace or remove complex constructs to resolve missing dependencies. The expert file contains missing declarations and definitions of classes, variables and functions. For providing this information, not much knowledge about the legacy code is required, since it can be found in the related files. The engineer mocks classes and functions, since the classes require only DTO-like (data transfer object) versions and the bodies of server-side functions are filled by WASM-T in (6). Full automation of these tasks is out of scope for this exploration of WASM in web migration prototyping. WASM-T completes the transformation (6) by generating the bodies of empty functions in the expert files and adding WASM/JS API code, thus code ensuring both communication directions. Then, the code is compiled to WASM (7).

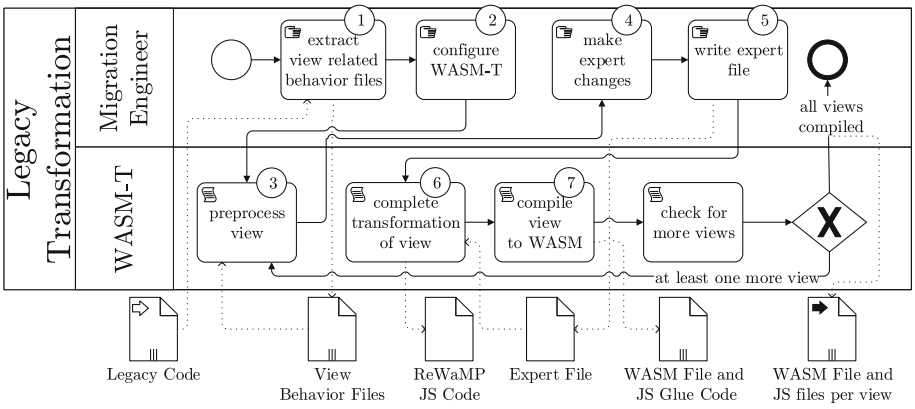

Fig. 2. ReWaMP process

4 Validation

In this validation, we answer the following questions through experimentation with ReWaMP: How much time and effort are required? What expertise is required by the migration engineer? What are the longest/most difficult activities in ReWaMP? How does WASM perform in the context of WM? Based on a

real world legacy codebase by our industrial project partner, we abstracted characteristics and created a sample application to answer these questions. Table 1 lists abstracted LS characteristics and their concrete instantiation in our scenario, a VC++ medical appointment management application. Appointments have a related patient, doctor and room. As business logic beyond CRUD, the system can determine the next time slot available for an appointment with a given combination of doctor and room. The following non-functional characteristics are represented: GUI is implemented using a desktop GUI framework, MFC in our scenario. Third party module dependencies are loaded at runtime, in the sample application as DLLs via assembly loading. The application consists of several components which exchange messages via MFC SendMessage. Files are used to store configuration data e.g., opening hours, and appointments are stored in an MS SQL database via ODBC. The scenario codebase comprises 3373 LOC, including 187 functions and 31 classes.

Table 1. Abstract characteristics of legacy software and scenario instantiation

Abstract characteristics	Instantiation in scenario
Legacy language	Visual C++
CRUD functionality	CRUD for medical appointments
Complex business logic	Find next available time slot
Desktop GUI	Microsoft Foundation Class (MFC)
Component communication	Window-to-Window via MFC SendMessage
Third-party dependencies	DLLs via assembly loading
File and data base persistence	Files and MS SQL Server via ODBC

4.1 Procedure

The scenario codebase was created by two researchers. Two other researchers conducted the validation experiment: one defined the experiment, measurements and observed, the other researcher was the test subject taking the role of the migration engineer. The test subject followed the steps described in Fig. 2. First he extracted the view behaviour files, then configured WASM-T and finally made the expert change for each view. To extract the files and perform the expert changes, Visual Studio was used to navigate the code base.

Since rapid prototyping requires quick and easy creation of prototypes, we measure time and effort in the experiment. **Time** has two main components, $t_M = t_M^{12} + t_M^{45}$ the time required by the ME to perform actions 1&2 and 4&5, and t_T the time required by the supporting infrastructure to analyze and transform the legacy code. t_M was measured by stopwatch, t_T was measured using python's time library. Time measurement started at the start of manual tasks after preparation of the tools, was interrupted if WASM-T started or resumed

processing and ended by finishing the last expert changes. t_T is dependent on the system environment. We used Intel i7 930 CPU (2,8 GHz), 14 GB RAM, Windows 10 x64. During measurement 87 other processes were active, and an average load of 13,8% was determined with in performance monitor. **Effort** evaluates the amount and extent of changes to the existing legacy code required by ReWaMP. We measure the effort in terms of e_T - the lines of code (LOC) added/changed/ deleted by WASM-T and e_M - the LOC added/changed/deleted manually by the ME. e_M was measured by observing the test subject's actions, and e_T by comparing the inputs and results of one continuous task set after the migration.

4.2 Results

The resulting ReWaMP prototype is available online[2]. Table 2 shows the measurements. We were surprised to see that the related files extraction did not take that much time (t_M^{12}). Here, expertise in the legacy UI framework is advantageous. We expected a lower ratio of e_M to the overall effort $e_M + e_T$, which is at 29%. This was caused by low decomposability of one view in the scenario, comprising DLL loading and debug code. The experiment showed that C++ experience and knowledge of the legacy UI framework are important. Basic web engineering expertise is helpful but not required. The most difficult task were the expert changes due to required legacy code comprehension. An interactive web prototype based on the legacy code could be created with reasonable time and effort through WASM-enabled code re-use. While a similar prototype can be created by an experienced web engineer in comparable time, the contribution of ReWaMP is to allow developers without web engineering expertise from the existing SME staff to achieve similar results.

Table 2. Time (in hh:mm:ss) and effort

t_M	t_M^{12}	t_M^{45}	t_T	e_M	e_T
01:17:42	00:21:31	00:56:11	00:00:72	253LOC	597LOC

5 Related Work

The large body of research in the web migration field [2,6] focuses on program decomposition, language transformations and legacy to SOA. With WebAssembly still being relatively new and a draft standard under development, there are no relevant publications available.

For software engineering, rapid prototyping has been acknowledged to increase the efficiency and effectiveness of software design [10]. By building and

[2] https://vsr.informatik.tu-chemnitz.de/demos/ReWaMP.

using a model of the system, representing the persistence, major business logic and user interface, it allows to gather feedback from users and to explore requirements. The availability of tool support for software prototyping plays a crucial role for its success [10]. In our work, we therefore seek to support the rapid web migration process by a WASM-based infrastructure. Appropriateness of rapid prototyping particularly includes new situations with limited experience to draw from [10]. For ReWaMP, the new situation is formed by the new technological environment of the web application, which significantly differs from the desktop application environment of the existing legacy system and bears uncertainties regarding feasibility and desirability.

It has to be noted that rapid prototyping as a methodology has often seen transfer into new domains. As a concept originating in the hardware development domain, it has been transferred into software engineering, especially in the context of information systems [1], UI Design [8] and even seen adoption for instructional design [10]. Likewise, in this paper we explore the application of rapid prototyping in the context of web migration, enabled by technologies like WebAssembly.

6 Conclusions and Roadmap

In this paper, we introduced ReWaMP, a novel rapid prototyping approach addressing the initial hurdle of web migration. ReWaMP enables migration engineers to quickly demonstrate desirability and feasibility. We outlined requirements and assessed technologies for supporting the approach, selecting WebAssembly as the most promising. Based on WASM, we presented a prototype architecture, creation process and supporting infrastructure for web migration prototyping, outlined challenges when applying WASM for WM prototyping and reported on the results of a validation experiment. Our next steps are investigation of automatic transformations of legacy UIs to web-based UIs under similarity constraints using optimization algorithms, the enhancement of automation of the supporting infrastructure and evaluation of the method in industrial context.

Acknowledgment. The authors would like to thank Thomas Blasek and Tobias Lang for their valuable contributions. This research was supported by the eHealth Research Laboratory funded by medatixx GmbH & Co. KG.

References

1. Alavi, M.: An assessment of the prototyping approach to information systems development. Commun. ACM **27**(6), 556–563 (1984)
2. Aversano, L., et al.: Migrating legacy systems to the web: an experience report. In: Proceedings of CSMR 2001, pp. 148–157. IEEE Computer Society (2001)
3. Heil, S., Bakaev, M., Gaedke, M.: Measuring and ensuring similarity of user interfaces: the impact of web layout. In: Cellary, W., Mokbel, M.F., Wang, J., Wang, H., Zhou, R., Zhang, Y. (eds.) WISE 2016. LNCS, vol. 10041, pp. 252–260. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48740-3_18

4. Heil, S., Gaedke, M.: Web migration - a survey considering the SME perspective. In: Proceedings of ENASE 2017, pp. 255–262. SCITEPRESS (2017)
5. IDEO.org: The Field Guide to Human-Centered Design. IDEO (2015)
6. Khadka, R., et al.: Legacy to SOA evolution: a systematic literature review. In: Migrating Legacy Applications, Chap. 3, pp. 40–71. IGI Global (2013)
7. Lucia, A., et al.: A strategy and an eclipse based environment for the migration of legacy systems to multi-tier web-based architectures. In: Proceedings of ICSM 2006, pp. 438–447. IEEE, September 2006
8. Nebeling, M., Leone, S., Norrie, M.C.: Crowdsourced web engineering and design. In: Brambilla, M., Tokuda, T., Tolksdorf, R. (eds.) ICWE 2012. LNCS, vol. 7387, pp. 31–45. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31753-8_3
9. Rivero, J.M., Heil, S., Grigera, J., Gaedke, M., Rossi, G.: MockAPI: an agile approach supporting API-first web application development. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 7–21. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39200-9_4
10. Tripp, S.D., Bichelmeyer, B.: Rapid prototyping: an alternative instructional design strategy. Educ. Technol. Res. Dev. **38**(1), 31–44 (1990)