Applying the ReMiP to Web Site Migration

Torsten Gipp
University of Koblenz-Landau
56070 Koblenz, Germany
tgi@uni-koblenz.de

Andreas Winter
Johannes-Gutenberg-University Mainz
55128 Mainz, Germany
winter@uni-mainz.de

Abstract

Web sites serve to publish information, both locally in intranets as well as on a global scale. Like all software systems, they have to cope with changing requirements and evolving technologies. The reference process model for software migration*, ReMiP, provides a generic process model for software migration in general. The paper introduces ReMiP and summarises the application of a tailored ReMiP towards migrating a static HTML-based web site to a content management system.*

Keywords: web site migration, process model.

1 Introduction

Once a web site is created, it evolves over time and has to be adapted to changing requirements and new technologies. This includes providing newer and more contemporary presentation views, improving the underlying infrastructure, or extending the web site's functionality.

One strategy to cope with these changes is to design the web site for good maintainability from the very beginning. This prospect is delivered by *(web) content management systems*, which provide a means to store, maintain, and deploy web content. By separation of content and visualisation, they enable easy modification of the content without having to consider its visualisation [10]. However, many legacy web sites exist that are *not* driven by technologies that ensure maintainability for a longer period of time. Maintaining static web sites consisting of various, more or less well-structured files, requires additional effort when adding similarly structured content.

If existing content is still newsworthy and well structured, it makes sense to transfer legacy web sites into a new environment. In addition to the advantages of maintaining the current content, further functionality can be added.

Transferring a static web site to a content management system can be viewed as *software migration*. Here, migration is considered as a transformation of software systems into a new environment without changing its functionality [8].

The reference process model for software migration, ReMiP [1, 2], presents a generic and adaptable process for software migration. This paper discusses the application of ReMiP to migrating the web site on GXL[1], a static HTML-based web site, into a content management system.

The setting is sketched in section 2 by describing the migration objective, the legacy and the target system. Section 3 embeds web site migration into general software migration. The generic reference process model for software migration, ReMiP, is introduced in section 4.1. Applying the taylored ReMiP to the GXL web site migration is presented in section 4.2. Section 5 evaluates the application of ReMiP for web site migration and concludes the paper.

2 The Setting

The GXL web site now exists for more than seven years. The site was set up during the development of GXL and was changed and extended according to the maturation of GXL. Migrating the GXL site became necessary because students maintaining the web site left university. Content management systems were not at hand when work on the GXL site started, so various helpers in form of makefiles and some scripts were developed to ease maintenance, but knowledge on using these helpers and on their interdependencies left with their creators.

Since the GXL site presents a set of well-structured data, it was decided to base the new GXL site on a content management system. The new web site was also envisioned to take advantage of embedding further components already developed for the university's content management system. These components include the bibliography database with extensive retrieval mechanisms and a general download mechanism for software artefacts.

2.1 Legacy System: Old GXL Site

The legacy web site introduces the *GXL – Graph eXchange Language*. A screenshot of an example page can be seen in figure 1. GXL is an XML-based standard exchange format for sharing data between tools. GXL has been ratified by reengineering and graph transformation research communities and is supported by various software (re-) engineering and graph transformation tools [9].

The web site is structured in five main sections. *Overview* gives introductory background information about the language, including examples, a list of frequently asked questions, and a list of related publications. The examples section has a sophisticated navigation structure that allows to browse through examples and along (meta) schemas for each example. In *Definition*, the GXL syntax is defined and *Schemas*

[1] http://www.gupro.de/GXL/

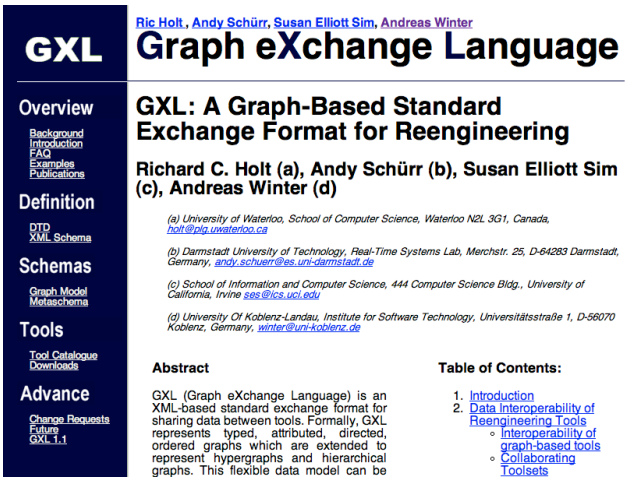

Figure 1. Example page from the old GXL site

shows the underlying graph model. *Tools* lists third party tools including related links and/or downloads, and *Advance* collects change requests.

Alltogether, the published web site currently consists of 450 static HTML documents, images, downloads and one CSS file. A dedicated documentation does not exist.

The web site is completely static, no calculation takes place when the user accesses one of the pages. Most content is stored in XML files, which are tranformed to static HTML by XSLT. Whenever the site content is changed, a generation step, defined in a sprawled makefile, is executed. Additional content is directly represented and rendered in HTML.

Analyzing the *technical quality* of the legacy system showed various syntactic errors in the HTML documents [7]. Most errors were related to repeatedly misusing the <a>-tag and not closing open tags. These errors were easily repaired in a separate restoration step.

2.2 Target System: New GXL Site

It was decided to migrate the GXL site into the university's web content management system *UniCMS* based on *Plone* [13]. Plone provides a user-friendly web interface for comfortable WYSIWYG and form-based editors.

Plone follows *content-oriented* management of data, whereas the old system follows a *page-oriented* style. Content-orientation refers to managing and publishing *content objects* which are presented dynamically. Each content object is associated to a content type that prescribes the object's properties and their types. In contrast, the page-oriented legacy system consists of structured XML files and some bare HTML documents without formal structure.

The *presentation* of content types in Plone is defined by page templates [17]. Page templates contain *expressions* that access content objects, and call macros that define the overall layout of content visualization. Thus, presentation and content are clearly separated in Plone.

2.3 Migration Challenge

Focussing on the differences between the legacy and the target system, three challenges had to be overcome in order to do the migration: (i) devise a mapping from XML/HTML documents to content types, (ii) design a Plone-based web site structure, functionally equivalent to the legacy site, and (iii) achieve a clear separation of content and visualisation.

3 Web Site Migration

The *life cycle of web sites* follows the software life cycle [14]. After being developed (*development*), web sites evolve over time, and have to be adapted to changing user needs and technological enhancements (*evolution*). After various evolutionary iterations, web sites reach a *servicing* stage, where maintenace becomes complex and expensive and only minor adaptations are possible. Like software systems, the servicing stage is followed by the *phase out* and *close down* of web sites. Extending a web site's lifetime requires reengineering activities like migration during the evolution stage to avoid transition into the servicing stage.

The current realisation of the GXL web site is near transition to the servicing stage. To keep this site in evolution, migration into a content management system is necessary.

Software Migration describes the process of converting an existing software system into a new environment. In ideal cases, software migration projects aim at technical conversions preserving the functionality of given systems and only address changes in non-functional requirements.

The legacy GXL web site (cf. section 2.1) is a static web site, only consisting of HTML and XML files. Layout is embedded in HTML files. Except of XSLT scripts, responsible for converting XML files to HTML files during publication, no further programs are used. The intended content-management-system-based target system provides a clear separation of content and layout. Thus, the GXL web site migration should be viewed mostly as a *data migration*.

Following the objective of the GXL web site migration, the target system is based on a new runtime environment, provided by Plone. Here, the development environment changes from plain XML/HTML files to Plone content types including their maintenance via Plone's user interface. Thus, in this aspect, the GXL web site migration addresses the migration of *development environments*.

Furthermore, the underlying architecture changes from simple file management to data management in an object-oriented database. Architectural issues also reflect the design of the web site structure in conjunction with the modeling philosophy provided by the runtime system. Parts of the legacy system, which were defined explicitely in HTML, are represented implicitly in Plone: Navigation by HTML tags has to be transferred to an implicit navigation according to Plone's folder structure. Here, the GXL web site migration has to be viewed as *architectural migration* (cf. [8]).

4 Migration Process

Process models in software engineering assure a controlled development of software systems. Various process models for software development were designed in the last decades. Except of being iterative, these process models do not support maintenance activities. Comprehending legacy systems, defining evolution strategies, developing intermediate architectures, transforming systems, and substituting legacies by migrated systems are ignored [8].

There are also specialized process models for software evolution and migration. For instance, the *Little Chicken approach* [6] and the *Migration phases* [16] address software migration in general. Process models focused on data migration are given by *MIKADO* [3] and the *Butterfly approach* [5]. Sneed [15] also addressed migration to object-oriented software systems and *SMART* [12] provides a process model for migration to service oriented architectures. These process models address software migration processes. However, they do not make use of established techniques used in general forward engineering oriented process models.

4.1 Reference Migration Processes

ReMiP was developed as an adaptive framework to tailor software migration processes based on a reference process model [1]. ReMiP includes best practices from iterative software engineering process models and from specialized software migration process models. The complete ReMiP is documented in the Rational Process Workbench[2].

4.1.1 ReMiP Phases

In ReMiP, migration processes are structured by four iterative phases. A *Preliminary Study* includes economic and technical evaluation of the legacy system. *Conceptualization and Design* comprises the legacy analysis and defines the migration strategy and the intended target system. *Migration and Transition* contains iterative transformation, testing, and delivery of migrated packages. Depending on the legacy's quality, software renovation may be required before migration in each migration package. The *Closing down* ends the migration, so that only the target system is operational.

4.1.2 ReMiP Disciplines

While phases give an organizational embedding of projects, iterations drive the project disciplines to fulfill the project's aim. All *disciplines* in ReMiP can be accomplished in every iteration. Depending on the project status, these disciplines are performed in different intensity (cf. [11]). ReMiP comprises the following disciplines from software engineering and software evolution process models:

Requirements Analysis: Since functional requirements are already specified by the legacy, requirements analysis in migration projects only deals with eliciting non-functional requirements. Functional requirements are elicited in *Legacy Analysis*. Migration projects usually require only cursory descriptions of the functionality. But both, functional and non-functional requirements, have to be managed to ensure adaptation to changing (non-functional) requirements.

Legacy Analysis: Analysis of legacy systems aims at understanding legacy systems. It also covers system assessment to estimate the legacy's quality and economic value. These estimations decide on additional renovation, help to rate the reusability, and control the migration strategies.

Target Design: The target design defines architectures, the data (base) structures and user interfaces of the new system. The target design has to ensure an economic transformation of the legacy. This may also require to define intermediate architectures, data representations or user interfaces to provide using the legacy and the target system in parallel. Next to the definition of target structures, transformations from legacy to target artefacts and wrappers to encapsulate legacy functionality have to be defined.

Strategy Selection: Depending on target design, technical and economic quality of the legacy system, strategies to perform the migration and to deploy (parts of) the migrated system have to be defined. Usually the legacy system is divided into various migration packages which are transferred independently. Each package can be migrated by redevelopment (i. e. re-implementation), by wrapping (i. e. providing an interface to the legacy code), or by conversion (i. e. transformation into the new environment).

In contrast to incremental deployment, *big bang* migrations define transformations which are deployed completely in one step. Incremental deployment often requires additional effort for design and development of intermediate structures, whereas big bang migrations are easier to fail.

Implementation: According to the selected migration strategies, the migration has to be performed for each migration package. Migration by conversion usually uses certain transformation tools. If the legacy is still under enhancement, additional delta migrations have to be considered.

Test: Testing validates functional equivalence between legacy and target system. The legacy system already constitutes an approved system providing the required functionality, which is used as test base in regression tests.

Deployment: Deployment deals with delivering all relevant artefacts including the replacement of (parts of) the legacy system by (parts of) the migrated system. Here, the deployment strategy defines whether deployment follows an incremental or big bang approach. It also specifies when migration packages are rolled out. Deploying the last iteration also includes the final shutdown of the legacy system.

4.2 Applying ReMiP

Web site migration is a special application of software migration. ReMiP interweaves disciplines and activities from

[2] http://www.ackermann-wolf.de/remip/process/ovu_proc_remip.htm

general forward engineering and specialized software evolution process models into an integrated process framework for migration processes. In the following, a tailored ReMiP is applied to the migration of the GXL web site.

4.2.1 ReMiP Phases

One result of the *Preliminary Study* was that the topic *Advance* could be omitted. The *Downloads* part was not migrated, because it will be used until the *UniCMS* component is operable. During *Conceptualisation and Design*, it was decided that the best migration strategy was a big bang strategy. Migration packages were defined according to a dependency analysis of the different parts of the legacy web site. Due to the small size of the GXL web site, the actual *Migration and Transition* could be handled in one single iteration. Since the legacy was not changed during migration, no freeze and no delta migrations were necessary. The data migration was achieved by partly using automated transformers.

At the present time, the legacy web site is still on-line, because the new system lacks a proper visual design. As soon as this is implemented, the project can be concluded with the *Closing down*. Changing the visual design is comparably easy, as it does not affect other system components.

4.2.2 ReMiP Disciplines

This section describes the application of the ReMiP disciplines, also mentioning their intensity in the different phases.

Requirements Analysis: The needs of the major stakeholders were elicited in a structured manner. The most important requirements dealt with providing an evolvable system, integration of the university's bibliography database, compliance to the UniCMS conventions, and better web browser compatibility.

The execution intensity of the requirements analysis discipline was high in *Preliminary Study* and at the beginning of *Conceptualisation and Design*. Small adaptations to the requirements were necessary during *Conceptualisation and Design*, because some requirements could not be realised in the target system in the given timeframe. One example is the visual design, which was decided not to re-implement in favour of the –albeit inferior– Plone default design.

Legacy Analysis: Legacy analysis was current throughout the whole process. First, the quality analysis was performed, as described in section 2.1. The economic value justified a migration. Small renovation activities like updating dead links were triggered during quality analysis.

The main analysis work consisted of reengineering a conceptual model that captured the data structure of the legacy. Additional analysis was needed at a later stage when the implications of the different navigation structure in the target system became clear.

Target Design: Plone was already fixed as the target system. A dedicated intermediate architecture was not necessary in our case. The transformation could directly address the internal Plone architecture. The necessary data structure was modelled as a class diagram, which was automatically converted to Plone content types by the *ArchGenXML* tool [4]. The identified concepts according to the conceptual model of the legacy system were mapped to these content types. This helped to meet the general requirement of not changing the functionality, because the data model could be re-used almost unaltered.

The navigation structure in Plone is different from that in the legacy system, so an adequate mapping was defined. Additional navigation features were gained as well. As an example, the legacy web site mentioned persons and their e-mail addresses in several places. During migration, these data were collected and proper person objects were made available in the target system. This lets the user see all person entries on one page by simply showing the contents of the respective folder.

Strategy Selection: It was found that the GXL web site could be best migrated by conversion. Conversion was feasible because the legacy data in the XML files was sufficiently well structured and the mapping to content types was straight-forward.

The deployment could be done using the big bang strategy. The big bang strategy was adequate because the legacy system would not change during the migration project, and the project itself was not time-critical.

Implementation: Due to the big bang strategy, the system could be isolated and transformed as a whole. Data migration was mainly done using automatic transformers. For each migration package, according to the content types, the necessary migration steps were implemented by Python modules. They converted XML and HTML documents into content type instances, using the Plone API. Since there was only a small amount of bibliography entries not already available in the university's database, the missing entries were added manually.

Since the GXL migration was a pure data migration, there was no need to implement further functionality.

Test: The web pages were checked for equivalence in both systems. The criteria were: completeness, link and HTML consistency, comprehensibly of the data presentation, and browser compatibility. For each criteria, manual or automatic tests were defined and successfully passed (cf. [7]).

The transformers were tested by continuous introspection. The implementation can surely be improved in terms of efficiency and maintainability. However, it is not necessary to improve the transformers since they will not be used anymore after completing the migration project.

Deployment: Except for the visual design, the target web site is finished. It was decided to deploy the design in the next iteration.

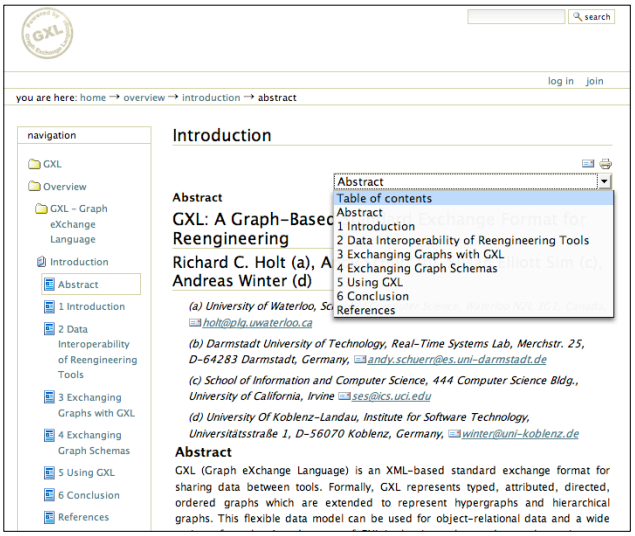

Figure 2. Example page in the new GXL site

5 Evaluation and Conclusion

A tailored ReMiP was successfully applied to migrate the static HTML-based GXL web site into a Plone-based content management system. Figure 2 presents the corresponding page to figure 1 in the target system[3].

The web page presents the introduction to GXL, presented on multiple pages. In the old web site, the navigation between the pages (next/previous) and the global navigation was implemented manually. By using the PloneArticle content type, this navigation presented in the drop down list and the navigation structure in the left column is generated automatically.

The new system consists of 280 Plone content objects. Comparing this to 450 single pages of the legacy system, the reduction is caused by the navigation in the *Examples* section. For each example, the old system had separate HTML files for the GXL source code, the graph visualisation, and the UML visualisation, whereas the new system stores these views in one content instance.

Since the case study only deals with data migration for a comparably small web site, not all features of the ReMiP were used. E. g., code and user interface migration, definition of temporary components, and synchronising multiple iterations, were not needed. Thanks to the adaptability of ReMiP, these activities could be safely omitted. ReMiP also showed its completeness since no additional activities were required.

ReMiP requires a significant amount of documentation. Analogously, comprehensive transformation rules were realised during case study. Considering the small size of the legacy system, less documentation and more manual transformations would have sufficed.

[3] http://wwww.uni-koblenz.de:14080/GXL

It can be concluded that applying the tailored ReMiP met the migration challenges (cf. section 2.3): (i) The data structure of the legacy was mapped by relating the conceptual model of the legacy system to corresponding content types. The instance data was then transformed according to this mapping. (ii) The navigation structure of the legacy system was transferred to appropriate Plone content types stored in appropriate folders. Plone automatically generates navigation means using this folder structure. (iii) By using Plone, layout/design and content is clearly separated. Replacing the currently used Plone default visualisation with a new visual design will be possible without touching the content.

It was shown that ReMiP could easily be tailored to the migration of small web sites, covering data-, development environment, and architectural migration. ReMiP was validated by successfully applying it to the migration of the GXL web site. Further evaluation is required to prove the general applicability of ReMiP in huge migration projects.

Acknowledgements: Thanks go to Ellen Ackermann for her valuable contribution of defining the ReMiP and to Johannes Caspary for migrating the GXL web site.

References

[1] E. Ackermann. Ein Referenz-Prozessmodell zur Software-Migration. Diplomarbeit, University of Koblenz, 2006.
[2] E. Ackermann, R. Gimnich, and A. Winter. Ein Referenz-Prozess der Software-Migration. *SWT*, 25(4):20–22, 2005.
[3] D. Aebi. Re-Engineering und Migration betrieblicher Nutzdaten. Phd-thesis, Universität Zürich, 1996.
[4] P. Auersperg, J. Klein, and R. van Rees. ArchGenXML code generator. http://plone.org/products/archgenxml, June 2007.
[5] J. Bisbal, J. Grimson, et al. Legacy Information Systems: Issues and Directions. *IEEE Software*, 16(5):103–111, 1999.
[6] M. Brodie and M. Stonebraker. *Migrating Legacy Systems*. Morgan Kaufmann, San Francisco, 1995.
[7] J. Caspary. Migration einer Website mit dem Referenz-Prozessmodell ReMiP. Thesis, University of Koblenz, 2006.
[8] R. Gimnich and A. Winter. Workflows der Software-Migration. *SWT*, 25(2):22–24, Mai 2005.
[9] R. C. Holt, A. Schürr, S. Elliott Sim, and A. Winter. GXL: A Graph-Based Standard Exchange Format for Reengineering. *SOCP*, 60(2):149–170, April 2006.
[10] G. Kappel et al. *Web Engineering*. Wiley, May 2006.
[11] P. Kruchten. *The Rational Unified Process*. Addison Wesley, Reading, 3rd edition, 2004.
[12] G. Lewis et al. SMART: The Service-Oriented Migration and Reuse Technique. Report CMU/SEI-2005-TN-029e, 2005.
[13] Plone Foundation. Plone. http://plone.org, March 2007.
[14] V. Rajlich and K. Bennett. A Staged Model for the Software Lifecycle. *IEEE Computer*, (2000):66–71, July.
[15] H. M. Sneed. *Objektorientierte Softwaremigration, Professionelle Softwareentwicklung*. Addison Wesley, 1999.
[16] H. M. Sneed, M. Hasitschka, and M.-T. Teichmann. *Software-Produktmanagement*. dPunkt, 2004.
[17] Zope Corporation. Zope Page Templates. http://www.zope.org/Documentation/ZopeBook/ZPT.stx, June 2007.