Software Modernization Revisited: Challenges and Prospects

Hugo Bruneliere, Inria—Mines Nantes—LINA
Jordi Cabot, ICREA—Universitat Oberta de Catalunya
Javier Luis Canovas Izquierdo, Universitat Oberta de Catalunya
Leire Orue-Echevarria Arrieta, Tecnalia
Oliver Strauss, Fraunhofer IAO
Manuel Wimmer, Vienna University of Technology

The authors discuss important factors to consider when migrating software to the cloud and offer recommendations to maximize the chance of success.

Organizations undertake more and more software modernization projects every day, mostly owing to rapid changes in the technological landscape pushing them to evolve their systems before those systems become obsolete. Such projects are sometimes taken too lightly and start more because of passing fads than because they're motivated by real technological limitations or system problems. In our experience, many managers have only a partial view of these projects' complexity and consequences.

A modernization project usually has three phases. First, reverse engineering provides an understanding of the system's purpose and current state. Many supporting approaches rely on model-based techniques to discover software models representing the system at a higher abstraction level. Examples of such models are UML (Unified Modeling Language) class diagrams, state machines, and workflows. Second, forward engineering analyzes those models and transforms them (if

0018-9162/15/$31.00 © 2015 IEEE

EDITOR **MIKE HINCHEY**
Lero—the Irish Software Research Centre;
mike.hinchey@lero.ie

necessary) to become the modernized system's specification. Finally, developers or automated code generation techniques (or a combination of both) use these models to produce the corresponding code for the targeted platform.

Tools can help in some of these tasks—for example, tools that discover UML models from Java code, or code generators for various platforms. But currently no global methodology exists that can guide software engineers through the whole process. Moreover, no real support exists for assessing project risks or final software quality.

On the basis of our experience in conducting and building tools for software modernization projects, we discuss here important factors to consider in such projects and offer recommendations to maximize the chance of success.

THE ARTIST PROJECT

We follow the approach taken by the ARTIST project (Advanced Software-Based Service Provisioning and Migration of Legacy Software; www.artist-project.eu). ARTIST is an ambitious European collaboration involving industry (the Athens Technology Center [ATC], Atos, Engineering Ingegneria Informatica, Sparx Systems, and Spikes) and academic institutions and research centers (Fraunhofer IAO, the Institute of Communication and Computer Systems [ICCS], Inria, Tecnalia, and the Vienna University of Technology). Part of the EU's Seventh Framework Programme for Research, the project started in October 2012 and is scheduled to last three years. It proposes a complete open-technology and vendor-neutral approach to software migration to cloud environments, including a generic methodology and base tooling support.

THE REAL FACE OF SOFTWARE MIGRATION

To get a representative vision of the factors impacting software modernization projects, we analyzed four case studies (from ARTIST industrial partners) of the migration of real software systems to the cloud:

- DEWS CCUI (Distant Early Warning System for Tsunamis Command and Control User Interface, by Atos), a Java system for tsunami early detection;
- LoB (Line of Business, by Spikes), a .NET solution for business process management;
- eGov (by Engineering Ingegneria Informatica), a J2EE (Java 2 Enterprise Edition) framework to support Italian citizens' interaction with their government; and
- News Assets (by the Athens Technology Center), a .NET news publication management system.

After initial analysis, we came up with six dimensions that helped us systematically evaluate each project's complexity:

- *Technical space (TS)*. A TS is a family of knowledge, tools, and technologies used to implement the system. For instance, some software artifacts belong to the grammarware TS (for example, source code files and the grammars they conform to), and others belong to the xmlware TS (for example, XML-based configuration files and their document type definition or schema). Each TS imposes a different reverse-engineering approach.
- *Origin*. Some software artifacts are generated; others are manually created (for example, the scaffolding of project files derived from specifications). It's important to filter out generated artifacts for which their abstract specification is available.
- *Purpose*. We identified main categories—for example, code or application, data, configuration, specification, and documentation. Having a fine-grained artifact categorization is fundamental for engineers to prioritize or filter artifacts during migration.
- *Architectural viewpoint*. We observed a typical four-layer classification: presentation, business logic, data, and communication. Again, this helps in problem decomposition and in identifying key company stakeholders who can assist during migration (depending on the layers to which the artifacts belong).
- *Environment*. External technological requirements and dependencies can strongly influence the process.
- *Size*. Size (for example, of memory) and the number of individual components are key elements to consider and study.

We circulated a survey among the case study companies and conducted interviews to evaluate each project regarding the six dimensions. Table 1 summarizes the results at the project level, even though our analysis was at the artifact level. Getting a better picture of the main project challenges (for example, each project had to treat and properly integrate a mix of TSs) benefitted both the companies and us. We recommend that companies perform a similar analysis before starting any migration project.

KEY SUCCESS FACTORS

From our experiences with the ARTIST project, we found that the following four key factors contribute to a more

TABLE 1. The four case studies according to the identified dimensions.

Dimension	Case study project and company*			
	DEWS CCUI, Atos	LoB, Spikes	eGov. Engineering Ingegneria Informatica	News Assets, Athens Technology Center
Technical space	Source code (Java, Python), XML	Source code (C#, PowerShell JavaScript, HTML, CSS [Cascading Style Sheets], ASP XAML [Active Server Pages Extensible Application Markup Language), XML, plain graphics	Source code (Java, OWL [Web Ontology Language], WSDL [Web Services Description Language]), XML, plain text, plain graphics	Source code (C#, JavaScript, HTML, CSS), XML
Origin	Mainly manual code, some code generation	Mainly manual code, some code generation based on domain-specific languages (DSLs)	Balanced (a partly generative approach for code)	Mainly manual code, some code generation
Purpose	Application, data	Application, configuration	Application	Application, data, configuration
Architectural viewpoint	Presentation, business logic, data	Presentation, business logic, data	Presentation, business logic, data	Presentation, business logic, data, communication
Environment	Eclipse Platform (Java), Linux	Microsoft Visual Studio, SQL Server (.NET), Windows	Eclipse Platform (Java), Protégé (ontologies)	Microsoft Visual Studio (.NET), Oracle relational database management system
Size	Medium	Medium for GPL parts, rather small for DSL parts	Large for ontology parts, rather small for the rest	Large for the application, medium for the rest

* DEWS CCUI: Distant Early Warning System for Tsunamis Command and Control User Interface; LoB: Line of Business.

unified, focused, goal-oriented, and guided migration.

One format to rule them all

Model-based migration, in which the system is represented by interrelated models (showing different system views), lets us more easily deal with the TSs' heterogeneity. Specialized components (discoverers) inject the content of artifacts (from different TSs) into the common model-based migration platform (into the modelware TS). Discoverers for several kinds of grammar-based and XML-based artifacts are publicly available. For the generated models to be interoperable, the modeling platform shares a common meta-metamodel (provided by the Eclipse Modeling Framework in our case).

We've benefited most from an appropriate mix of modeling languages such as

- UML,
- SysML (Systems Modeling Language),
- BPMN (Business Process Model and Notation),
- KDM (Knowledge Discovery Metamodel, part of the Architecture-Driven Modernization initiative; http://adm.omg.org), and
- some domain-specific languages (DSLs).

We also rely on corresponding model-based techniques, such as model transformation or text and code generation, to process models generically.

Different views for different stakeholders

Modeling frameworks separate the visualization of modeled information (the concrete syntax) from its content (the abstract syntax). This lets engineers process model content efficiently and provides several visualizations (possibly using alternative notations) of a model or model fragment. This distinction is especially useful for general-purpose modeling languages such as UML or ODP (Open Distributed Processing) that predefine different viewpoints for different stakeholders. It lets engineers compute several views of a system to emphasize or disregard certain aspects.

From our experience, viewpoints are beneficial in migration projects, and not only for standard system design. During reverse engineering, viewpoints help improve system understanding (for example, separating a system's structure from its behavior, or the architecture from a detailed implementation). In forward engineering, viewpoints are fundamental for making decisions on software modifications.

Nonfunctional properties as first-class citizens

Although migration projects traditionally focus on systems' functional aspects, they also deal more frequently with nonfunctional aspects. This is particularly true for software migration to the cloud. Often, the base platforms and programming languages don't change during migration. However, cloud environments might introduce features or capabilities that can help improve nonfunctional properties that are important for the system and its owner. In such projects, nonfunctional properties are the main driver for migration and must be taken into account in all phases. A concrete example is the design and realization of refactorings tailored to improve performance and scalability.

Migration as a process

Besides representing software artifacts, models also help explicate knowledge of the migration process. Engineers can exploit this to enact a well-defined migration process for their projects.

We've established a systematic road map for achieving software migration by providing a reference process (as an explicit process model) to be customized for specific migration projects. The available tooling supports this customization and allows following the specified process step-by-step. So, defined processes not only concentrate on actual code migration activities but also address premigration (for example, specifying migration goals and identifying a target environment to satisfy expressed requirements) and postmigration (for example, evaluating initial migration goals against the finally migrated system).

A CONCRETE APPROACH TO CLOUD MIGRATION

Using the principles and guidelines we've been describing, we developed a concrete methodology as part of ARTIST (see Figure 1). As summarized graphically, it has three phases:

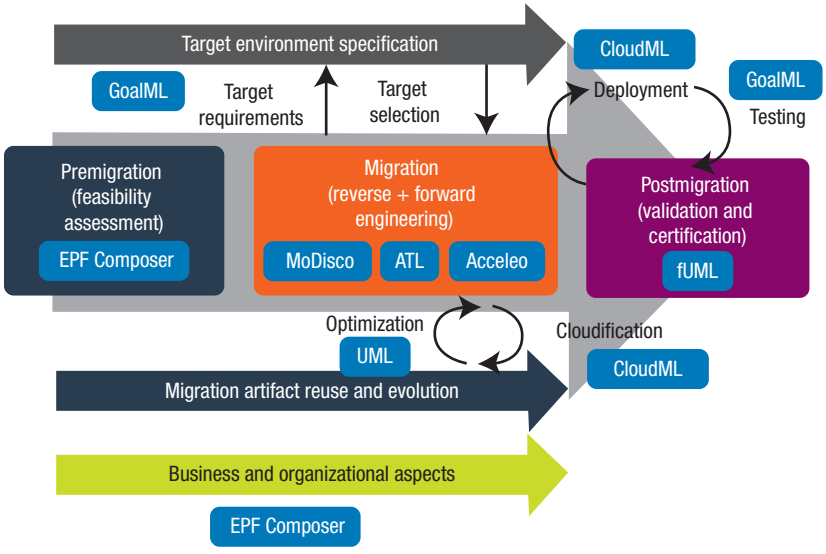

Figure 1. The ARTIST (Advanced Software-Based Service Provisioning and Migration of Legacy Software) process. This migration approach intends to be an answer to the challenges discussed in this article, and a concrete application of the success factors we've identified.

- Premigration evaluates the existing software, notably in terms of migration feasibility from both a technical and business perspective (for example, checking cloud compliance on the basis of standards such as the Topology and Orchestration Specification for Cloud Applications [TOSCA] or drafts such as the Cloud Computing Reference Architecture [CCRA]).
- Migration covers both reverse- and forward-engineering activities (including possible optimizations in the migrated software). It also includes selection of the target cloud provider.
- Postmigration addresses the migrated software's verification and validation. It also addresses the potential certification of the newly produced (cloud-based) pieces of software, notably in regard to cloud standards and current best practices.

We've implemented two versions of the tooling to support this methodology. One relies on the Eclipse framework and handles Java/J2EE cases. The other is based on Sparx Systems Enterprise Architect and handles C#/.NET cases. Throughout the process, and particularly during migration and optimization, we extensively use UML models (sometimes extended with some profiles) to represent the system's required aspects. This also facilitates exchanging models between tools and building several viewpoints. All used or produced models can be stored in or retrieved from the ARTIST repository at any time.

We employ other modeling languages and techniques in more domain-specific tasks. For example, we use GoalML (Goal Modeling Language) during premigration to precisely determine the migration's scope (for example, regarding nonfunctional properties) and to help determine the migration's feasibility. During postmigration, fUML (Foundational UML) behavioral models of the deployed system are automatically checked against the initial goal models to validate that the corresponding objectives are satisfied. The target specification and deployment process depend on

CloudML (expressed as a UML profile) to represent the required cloud infrastructures.

This is made easier by the reuse of open source modeling solutions (thanks to the shared modelware TS), mainly from the Eclipse ecosystem. In particular, we employ the Eclipse Process Framework (to enact and customize the ARTIST Methodology Process Tool [MPT] to follow the various ARTIST phases), the MoDisco model-driven reverse-engineering framework (for code-to-model discovery), ATL (for model-to-model transformation), and Acceleo (for model-to-code generation).

In ARTIST we elaborated on a complete migration methodology and base supporting tooling. But obviously, aspects of the different steps of the process still need improvement. So, we're creating an ARTIST Club, which will initially involve the project's partners and later be open to external participants. We want to make this collaborative effort a good opportunity to continue capitalizing on the ARTIST results, in terms of future research achievements (for academics) and further commercial exploitation (for industrial partners).

HUGO BRUNELIERE is a research engineer and project manager on the AtlanMod Team, a joint Inria, Ecole des Mines de Nantes, and LINA (Nantes Atlantic Computer Science Laboratory) research team. Contact him at hugo.bruneliere@inria.fr or hugo.bruneliere@mines-nantes.fr.

JORDI CABOT is an ICREA (Catalan Institution for Research and Advanced Studies) research professor at Universitat Oberta de Catalunya (UOC) and previously led the AtlanMod Team. Contact him at jordi.cabot@icrea.cat.

JAVIER LUIS CANOVAS IZQUIERDO is a postdoctoral fellow at UOC. He previously was on the AtlanMod Team. Contact him at jcanovasi@uoc.edu.

LEIRE ORUE-ECHEVARRIA ARRIETA is a project manager and the leader of the Cloud Technologies Team in Tecnalias's ICT-European Software Institute Division. Contact her at leire.orue-echevarria@tecnalia.com.

OLIVER STRAUSS is a research engineer on Fraunhofer IAO's Software Technology Competence Team. Contact him at oliver.strauss@iao.fraunhofer.de.

MANUEL WIMMER is a senior researcher in the Vienna University of Technology's Business Informatics Group. Contact him at wimmer@big.tuwien.ac.at.