Information and Software Technology xxx (2008) xxx–xxx

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Evaluating legacy system migration technologies through empirical studies

Massimo Colosimo [a], Andrea De Lucia [a], Giuseppe Scanniello [b,*], Genoveffa Tortora [a]

[a] Dipartimento di Matematica e Informatica, University of Salerno, Via Ponte Don Melillo, 84084 Fisciano (SA), Italy
[b] Dipartimento di Matematica e Informatica, University of Basilicata, Viale Dell'Ateneo, Macchia Romana, 85100 Potenza, Italy

A R T I C L E I N F O

Article history:
Received 1 February 2008
Received in revised form 5 May 2008
Accepted 6 May 2008
Available online xxxx

Keywords:
Software migration
Empirical studies
Controlled experiments
Case studies
Technology transfer

A B S T R A C T

We present two controlled experiments conducted with master students and practitioners and a case study conducted with practitioners to evaluate the use of MELIS (Migration Environment for Legacy Information Systems) for the migration of legacy COBOL programs to the web. MELIS has been developed as an Eclipse plug-in within a technology transfer project conducted with a small software company [16]. The partner company has developed and marketed in the last 30 years several COBOL systems that need to be migrated to the web, due to the increasing requests of the customers. The goal of the technology transfer project was to define a systematic migration strategy and the supporting tools to migrate these COBOL systems to the web and make the partner company an owner of the developed technology. The goal of the controlled experiments and case study was to evaluate the effectiveness of introducing MELIS in the partner company and compare it with traditional software development environments. The results of the overall experimentation show that the use of MELIS increases the productivity and reduces the gap between novice and expert software engineers.

1. Introduction

Legacy systems typically form the backbone of the information flow within organizations and are the main driver to consolidate information on their business. In case one of these systems stops working, the business might be dramatically influenced. In the literature different solutions have been proposed to replace such systems [9,32,40]. Typical solutions include: discarding the legacy system and building a replacement system, freezing the system and using it as a component of a new larger system, and modifying the system. Changes may range from a simplification of the system through a reduction of size and complexity, to preventive maintenance operations such as redocumentation, restructuring, and reengineering, to an adaptive maintenance process entailing interface modification, wrapping, and migration [32]. These alternatives are not mutually exclusive and the decision of the approach to use is generally based on an assessment of the quality and business value of the system [4]. Often other non-technical factors influence this decision, as for an example the need of software enterprises to move their legacy systems to a modern infrastructure in order to remain competitive in the global market [3]. In this case, system migration is the only viable alternative [3], since the risk of replacing a legacy system might be unsustainable [39].

The migration of a legacy system is a complex task, which is influenced by several concerns (e.g. its decomposability, budget, technical and time constraints, etc...). In order to preserve the past investments, and reduce risks and development costs the encapsulation of legacy systems through wrapping technologies is a viable alternative [8,14,23,39,41]. Wrapping technologies are now mature and also include commercial solutions (e.g., WebSphere Studio Enterprise Developer [27] offered by IBM). Unfortunately, even when based on wrapping technologies the legacy system migration often requires ad hoc solutions that needs to be assessed.

To determine whether a new technology is successful, we should look at whether it solves a problem and it is commercially viable and at how long the innovation takes to become accepted in practice. Furthermore, it should be also useful to understand how to increase the odds that the technology choice is the right one [33]. Redwine and Riddle [36] have observed that the time needed for a software technology to mature to the point that it can be diffused to the practitioners of a given company ranges between 15 and 20 years. Nevertheless, this is a too long time in the software market, where the time-to-market is essential to keep competitive [33]. A consequence is that innovative technologies are often adopted before a clear evidence of their advantages. To overcome these difficulties experimentation is necessary [4,25] and should be performed with researchers, practitioners, and users to assess innovative technologies and tools. Experimentation requires a research plan that will take place over the years. Generally, experiments are formal, rigorous, and controlled investigations normally conducted in laboratory environment to provide a high

* Corresponding author. Fax: +39089963303.
E-mail addresses: massimo.colosimo@gmail.it (M. Colosimo), adelucia@unisa.it (A.D. Lucia), giuseppe.scanniello@unibas.it (G. Scanniello), tortora@unisa.it (G. Tortora).

0950-5849/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2008.05.012

level of control. In case it is not possible to randomly assign subjects to different treatments [11,44] quasi-experiments can be alternatively performed. Experimentation should be used to enable the identification of technologies ready to be transferred to particular organizations and market segments, so an evaluation of these technologies in typical situations or pilot projects is also required. Generally, such an evaluation cannot be conducted in a controlled way, so case studies have to be performed, where the activities concerning the use of the technology are observed through the collection and the analysis of relevant data [4,44].

In [16], we have presented a process and a supporting environment, named MELIS (Migration Environment for Legacy Information Systems), for the migration of ACUCOBOL-GT legacy systems to the web. The migration process and tool were developed within a technology transfer project conducted in cooperation with a small software company. The goal of the company was to become an owner of the technology in order to migrate their legacy systems according to the new technology and business change requirements of their customers. In that paper, we have also presented the results of a preliminary case study conducted on a legacy system of the partner company to validate the approach and to get feedbacks about the tool.

In this paper, we evaluate the effectiveness of using MELIS during the migration of COBOL legacy systems to the web, in particular concerning the productivity improvement. To this aim, we have conducted user studies to compare the support given to the migration process defined in [16] by MELIS against the support given by the COBOL development environment used within the partner company, namely ACUBENCH [1], combined with the web development environment integrated in MELIS, namely Lomboz [26]. It is worth noting that while the company has a long experience of COBOL development (in particular with ACUBENCH), web development is still in the future plans, together with the migration of the marketed COBOL legacy systems to the web. This means that there was not an established web development environment available in the company that we could compare with MELIS in the user studies and for this reason we decided to use Lomboz. In particular, we present here the results of two controlled experiments and a case study. The first controlled experiment was conducted using master students in Computer Science, while the replicated experiment was conducted within the laboratory of the partner company and involved professional programmers. The goal was to evaluate how much the use of MELIS improves the productivity of software engineers and reduces the gap between expert and novice programmers. As the nature of controlled experiments requires that migration tasks are completed in a few hours, we had to necessarily select for these tasks small COBOL programs. To evaluate the benefits of MELIS in a real environment we also conducted a case study, where the key practitioners of the partner company were asked to migrate two subsystems of a relevant system with the MELIS tool and the development environments ACUBENCH and Lomboz.

The remainder of the paper is organized as follows: Section 2 presents related work. The developed migration technology in terms of strategy, migration process, and supporting tool is highlighted in Section 3. The controlled experiments and the cases studies together with the achieved results are discussed in Sections 4 and 5, respectively. Final remarks and future work conclude the paper.

2. Related work

Internet is an extremely vital technology that is changing the way in which organizations conduct their business and interact with the partners and the customers [3]. To take advantage of the Internet open architecture, most companies discard the existing software systems and then develop new systems that meet the new needs of the company business. However, economic and technical constraints make the development of new software systems impossible in most cases. Therefore, the migration represents a successful activity to preserve the past investment [9,39] and concurrently to move the original system toward the new technology infrastructure (i.e., the web).

During the years, several approaches have been presented to migrate monolithic and procedural legacy systems to different distributed architectures, including client server architectures [10,12], distributed objects architectures [35,41], and web-based and service oriented architectures [2,3,7,13,30,42]. These approaches share some commonalities in the used reverse engineering and reengineering methods. Furthermore, due to the risks involved in reengineering and migration projects [3,9,39], the majority of these approaches are incremental. Typically, incremental migration approaches tries to decompose large legacy systems in smaller chunks, which are then wrapped and used within a new environment. Thus, in this scenario wrapping approaches become essential.

Wrapping is essentially the practice of implementing a software architecture using pre-existing components. The component can be a batch program, an online transaction, a program, a module, or even just a simple block of code [39]. Components are accessed via a wrapper that implements the interface that newly developed objects use to access legacy components or systems. Therefore, wrappers are responsible for passing input parameters to the encapsulated component, capturing the output data, and returning them to the requester. Different wrapping approaches have been presented in the literature, which differ mainly in how they support the migration strategy and in the technology they use to encapsulate the legacy software at different granularity levels. For example, Sneed [41] proposes tools and wrapping techniques, where online programs are transformed into data driven subprograms which process an XML-document. Differently, batch programs are adapted to read and write XML-documents. Finally, in subprograms parameters are set from an XML-document. Recently, Sneed in [42] has extended this approach to enable the integration of legacy systems into a Service Oriented Architecture. In [14], a CORBA-IDL (Interface Definition Language) wrapper is adopted to migrate legacy systems to client–server architectures. Wrapping is performed at both the procedure and program level. After receiving the requests from a remote client, the wrapper simulates a Job Control Language (JCL) procedure in the creation of input and output files and invokes the legacy system program. Lin et al. [23] propose to use a wrapper in order to reuse MS-Windows as a CORBA object. The wrapper implementation requires the wrapper to redirect the input/output data streams and to generate a CORBA interface. The windows task input channel is redirected using an event message simulation for mouse and keyboard devices, while the wrapped application saves the output data in the clipboard space using the output redirection mechanism. The CORBA-IDL is adopted to create the CORBA interface.

Wrapping is also used in [24] for integrating COTS MS-Windows applications in a distributed system using Java technologies. The authors propose a 3-phase process where the components are first encapsulated, constructing a server-side Java object by wrapping a MS-Windows application, secondly, by including a coordinator to integrate the wrapped applications, and thirdly by constructing a user interface implemented in Java to manipulate the integrated applications.

Encapsulating legacy systems usually requires reengineering the legacy user interface using technologies of the new environment in which the legacy system has to be accessed. User interface reengineering involves reverse engineering to abstract a user interface conceptual model and forward engineering to re-implement the user interface [28]. Several approaches based on data-flow analysis [27], State Transition Diagrams [8,43], and techniques derived from artificial intelligence [29] have been proposed in the literature. For example, Moore and Moshkina [30] describe an architectural restructuring aimed at migrating character-oriented user interfaces to a web-based front end.

An approach to migrate character based user interface to any client devices is proposed in [8]. This approach exploits a black-box technique to identify the dynamic and static models of user interfaces based on characters and forms and reproduces them on client devices with the support of a software wrapper. Checks on the user interfaces are performed when all the input fields of a form are filled in. An extension of the approach to a service oriented architecture has been proposed in [13].

Many of the web migration strategies proposed in the literature are conceived for decomposable or semi-decomposable legacy systems [9]. For example, Aversano et al. [2,3] propose the main results of a migration project aimed at integrating an existing COBOL system into a web-enabled infrastructure. The original system was semi-decomposable with a client component (represented by the user-interface) and a server component (represented by the application logic and database). The migration approach that Aversano et al. [2,3] propose requires that the graphical user interfaces are manually migrated to Microsoft ASP technology, while the server components are wrapped using dynamic load libraries.

Also, Bodhuin et al. in [6] describe an approach to migrate a COBOL system easy to decompose towards a web-enabled architecture based on Model View Controller (MVC). The software components of the original legacy system are identified using slicing techniques. These components are restructured and then turned into java classes by using the PERCobol tool [22]. The same authors in [7] present an approach and a tool to migrate a non-decomposable LEGACY SYSTEM to two-tier web-enabled architecture through the use of a Screen Proxy. This approach is very close to our approach. However, the authors do not discuss problems related to embedded side effects and control flow in the user interface description.

Zdun [45] provides an overview of a reference architecture integrating the recurring components and the wide variety of technologies needed to bring a legacy application to the web. This architecture is based on legacy system wrapping and takes into account several other aspects, including authentication, session management, dynamic content creation, and presentational abstractions. In this way the author provides a conceptual understanding of which components are required for reengineering a larger system to the web.

Research in the empirical software engineering field aims at acquiring general knowledge about which process, method, technique, language, or tool is useful for whom to conduct which tasks in which environments [4,21,38]. Concerning the migration of legacy systems, strategies and supporting tools have been largely experimented in case studies conducted on real size software systems [2,7,10,30,35,41]. On the other hand, controlled experiments have been rarely used in the software maintenance field. The combined use of case studies and controlled experiments let software engineers better evaluate whether the migration technologies produced in research laboratories fulfill the industry needs [5,23,33].

3. Industrial context and migration strategy

In this section, we briefly describe the technology transfer project in which the user studies described in this paper have been conducted. In particular, we describe the software development context of the partner company, the results of the assessment of the legacy systems that the company needs to migrate, the defined migration strategy and process and the tool that we have developed to support this process. More details can be found in [16].

3.1. The context

The main goal of the technology transfer project was to define a migration strategy and process and the supporting technology to make the company's practitioners able to systematically migrate their legacy systems to the web, according to the customer requirements.

The partner company has been developing and maintaining standard business-oriented software packages for 30 years. It started with the development of COBOL systems for minicomputers in the '70s. In the '80s the company first moved its COBOL development to UNIX workstations and then to PCs with MS-DOS. The legacy systems developed in the previous decade were migrated to the PC in order to broaden their market segment of smaller users. As part of this migration, the partner transferred its software first to Micro Focus COBOL for MS-DOS in the '80s and then to ACUCOBOL for MS-DOS in the '90s. At the end of the '90s the ACUCOBOL development environment was upgraded to the ACUCOBOL-GT version, which supports the development of graphical user interfaces for Windows. The software development environment currently used is ACUBENCH [1].

As first step of the migration strategy definition we assessed the legacy systems with greatest business value developed and marketed by the partner company [16]. Among the legacy systems that the management identified as business critical for the company, in particular we selected the systems that the customers required to access on the web. The results of the assessment revealed that these systems had a very low level of decomposability with spaghetti-like code. For this reason, we decided to adopt an incremental migration strategy based on the reengineering of the user interface using web technology, on the transformation of interactive legacy programs into batch programs, and the wrapping of the legacy programs. The new web-based user interface and the wrapped legacy system communicate trough a middleware component.

3.2. Target environment and migration strategy

The target architecture resulting from the migration of a legacy system to the web is depicted by the deployment diagram of Fig. 1. The technologies used for the implementation of the software components of the target architecture were imposed as a constraint by the partner company.

At the end of the migration process described in Section 3.3, the interactive programs of the legacy system are converted into batch programs (Wrapped Legacy System) and all interactions with the user are redirected to the Middleware component,[1] which establishes a communication link between the migrated user interface and the application logic of the legacy system. The user interface is divided into two components, the Reengineered GUI and the GUI Deliverer. The Reengineered GUI includes the Java Server Pages replacing the COBOL SCREEN SECTIONs of the original system, while the GUI Deliverer includes the Java Servlets and Beans used to manage the control logic of the new web user interface and access the functionalities of the Wrapped Legacy System through the Middleware component. The GUI Deliverer accesses the Reengineered GUI

[1] User interfaces in COBOL are described using SCREEN SECTIONs, which define dialogs using the ACCEPT and DISPLAY statements. In particular, these statements are used to display output data on the user interface and to receive input data from the user interface, respectively. In a migrated program, these statements are wrapped and the dialog is redirected to the Middleware component.

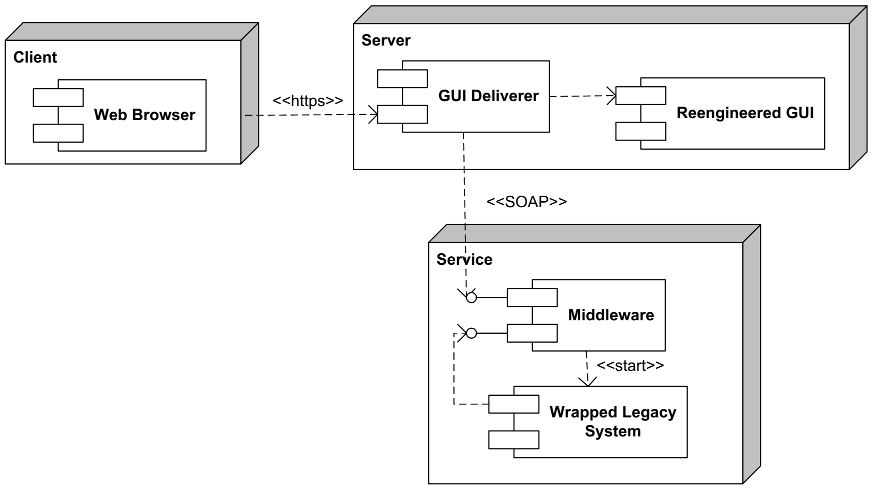


Fig. 1. The target software architecture.

to get the web pages required to accomplish a given function. The GUI Deliverer component is accessed by the web browser using the HTTPS (HTTP over Secure Socket Layer) protocol.

The Middleware was implemented as a Dynamic Link Library (DLL). This component runs on the same node as the Wrapped Legacy System. The DLL was developed using the programming language Delphi. This language was required by the partner company due to the experience of the practitioners involved in the project. To enable the communication between the GUI Deliverer and the Middleware component, we wrapped the latter using JNI (Java Native Interface). In case the Web server and the Middleware run on the same node, no further communication middleware is needed. Otherwise, when it is required that these components run on different nodes a communication middleware (e.g., RMI or SOAP) has to be used.

The Middleware starts the Wrapped Legacy System when required. Successively, it manages the communication between the reengineered user interface and the wrapped legacy system using a shared memory and semaphores. The Middleware component provides two different interfaces to the wrapped legacy program and the new user interface, respectively. The first interface is provided to the wrapped legacy program to set and get information from the web-based user interface. The second interface is provided to the GUI Deliverer software component to access the functionalities of the wrapped legacy system. More details on the communication between the components of the target architecture can be found in [16].

3.3. Migration process and tool support

The Middleware component of the target architecture shown in Fig. 1 is a generic component developed so that it can be used in the migration of any legacy program, thus enabling the software engineer to concentrate only on the wrapping of the legacy programs and on the reengineering of the user interface. The adopted migration process is shown by the activity diagram with object flow of Fig. 2. Ellipses are phases of the process, whilst rectangles are software artifacts. To fully support the migration of the legacy systems of the partner company according to the phases of the proposed strategy we decided to develop an Eclipse plug-in, named MELIS (Migration Environment for Legacy Information Systems).

The *Pre-processing* phase is automatically supported by MELIS and provides details on the types and attributes of each graphical object composing the SCREEN SECTIONs of a given ACUCOBOL-GT program. The structure of the SCREEN SECTIONs and the corresponding graphical objects are encoded in an XML file, which hierarchically organizes the SCREEN SECTIONs, the contained objects (ENTRY-FIELDS, LABELs, etc.), and the control checks of a given COBOL program. Control checks on the ENTRY-FIELDs of a SCREEN SECTION are expressed in terms of BEFORE and AFTER clauses. These clauses trigger the execution of COBOL paragraphs before and after the user has filled in the ENTRY FIELD on the screen. As discussed in [16], this control flow embedded in the user interface description of COBOL programs increase the difficulty of separating the presentation logic from the application logic and database access logic and make it almost impossible to completely automate the migration process.

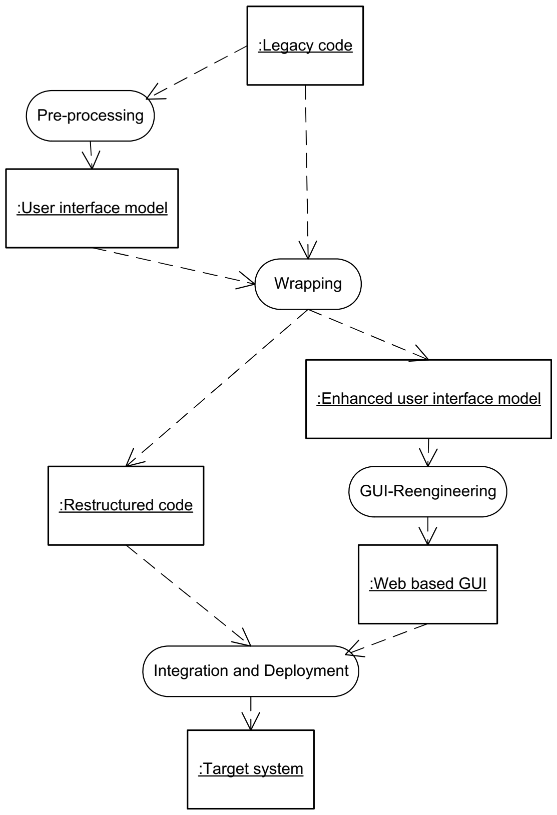


Fig. 2. The migration process.

During the *Wrapping* phase the XML file representing the SCREEN SECTIONs is enhanced and the original code is wrapped to become a batch program. This is the most challenging phase of the migration process and is semi-automatically supported by MELIS. The main issue of this phase consists of identifying the checks that can be or not migrated to the client. In particular, the checks on the entry fields of the SCREEN SECTIONs have to be classified as: format check operations, which are migrated to the client, and field database access or application logic operations, which are left on the wrapped COBOL program and invoked using the Middleware component. This classification is suggested by MELIS, thus providing a semi-automatic support.

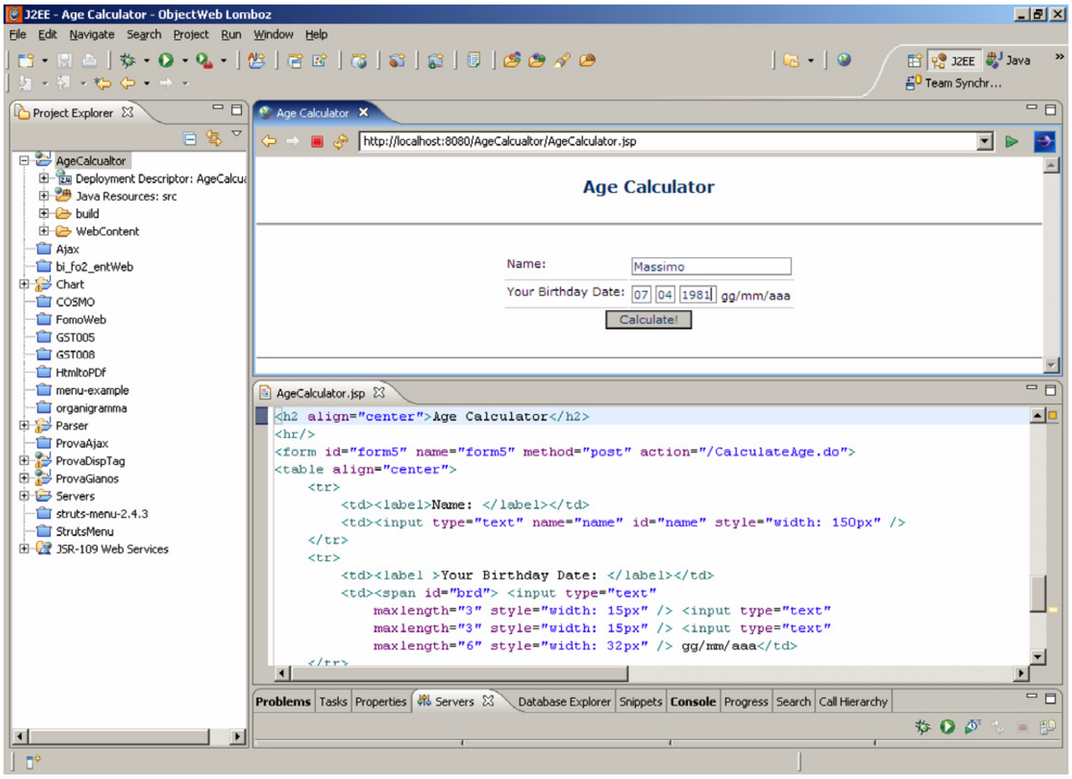
The format checks that can be migrated to the client are embedded in the XML file in terms of javascript functions. To further support the developers, MELIS also provides a library of reusable and customizable javascript functions. Concerning the checks that cannot be migrated to the client, MELIS automatically comments the statements DISPLAY (used to visualize a SCREEN SECTION) and ACCEPT (used to get data from a SCREEN SECTION) and then replaces them by calls to the Middleware component.

The *GUI-Reengineering* phase produces the web-based graphical user interface starting from the intermediate SCREEN SECTION representations. MELIS automatically generates a JSP page for each SCREEN SECTION of a given ACUCOBOL-GT subsystem. Fig. 3 depicts a reengineered web-based graphical user interface as shown in the MELIS plug-in. In particular, this figure shows the reengineered user interface of the ACUCOBOL-GT program used within a migration task of the controlled experiments described in Section 4. To enable the communication between each JSP page and the wrapped legacy code MELIS also generates a Servlet and a Java Bean. The Java Bean contains the fields of the SCREEN SECTIONS and is built from the XML description contained in the enhanced user interface model. It is worth noting that to modify the generated JSP pages MELIS integrates Lomboz [26], an Eclipse plug-in available under GPL license for J2EE development.

Finally, the *Integration and Deployment* phase enables the integration of the wrapped legacy code and reengineered web-based graphical user interface. First the wrapped legacy code is compiled using the ACUCOBOL-GT compiler integrated in MELIS, and then the reengineered web-based graphical user interfaces and the compiled legacy code are deployed on the *Web Server* and on the *Application Server* nodes, respectively. MELIS fully supports the compilation of the wrapped legacy code, the deployment of the reengineered user interface, and the execution of the migrated program (see Fig. 4). In this phase the software engineer should also perform regression testing to prove the functional equivalence between the original legacy system and the migrated system. Test case specifications can be derived from the original legacy system and used to exercise the target system in order to identify possible differences in behavior. Let us note that MELIS does not provide specific support to automatically derive test cases and check the functional equivalence.

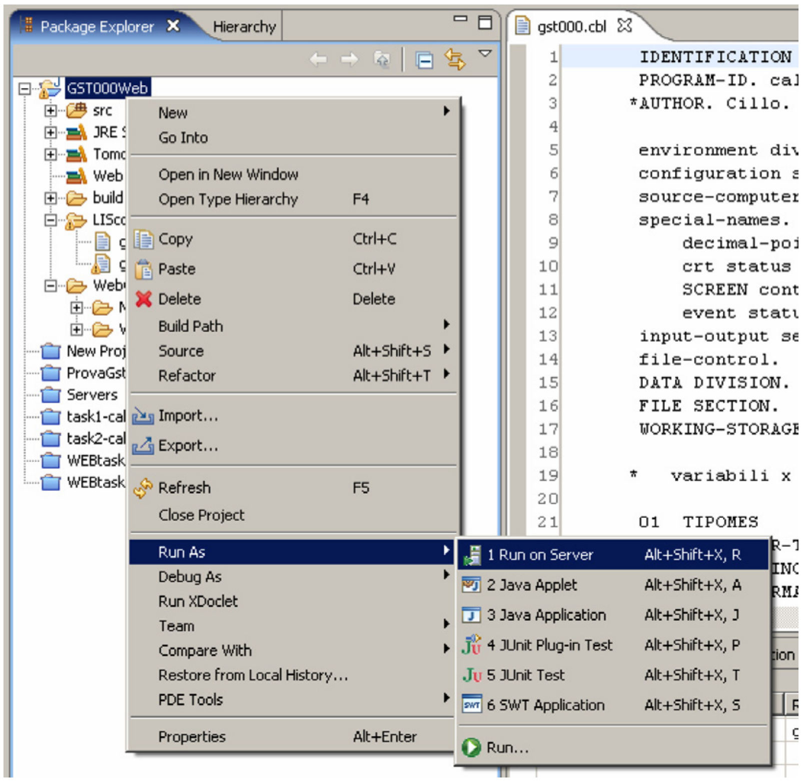
4. Controlled experiments

In this section, we report on the results of two controlled experiments, which have been conducted with master students in Computer Science at the University of Salerno and professional programmers of our partners company, respectively. These experiments aimed at assessing the effectiveness of using MELIS in the migration of ACUCOBOL-GT programs compared to ACUBENCH, the software environment used within the partner company for COBOL development, and the Lomboz Eclipse plug-in for web development. While the company has a long experience of COBOL development (in particular with ACUBENCH), web development is still in the future plans, together with the migration of the mar-



```
<h2 align="center">Age Calculator</h2>
<hr/>
<form id="form5" name="form5" method="post" action="/CalculateAge.do">
<table align="center">
    <tr>
        <td><label>Name: </label></td>
        <td><input type="text" name="name" id="name" style="width: 150px" />
    </tr>
    <tr>
        <td><label >Your Birthday Date: </label></td>
        <td><span id="brd"> <input type="text"
            maxlength="3" style="width: 15px" /> <input type="text"
            maxlength="3" style="width: 15px" /> <input type="text"
            maxlength="6" style="width: 32px" /> gg/mm/aaa</td>
    </tr>
```

Fig. 3. Reengineered user interface.



```
  IDENTIFICATION
  PROGRAM-ID. cal
 *AUTHOR. Cillo.

  environment div
  configuration s
  source-computer
  special-names.
      decimal-poi
      crt status
      SCREEN cont
      event statu
  input-output se
  file-control.
  DATA DIVISION.
  FILE SECTION.
  WORKING-STORAGE

 *   variabili x

  01  TIPOMES
```

Fig. 4. MELIS support to run the migrated program on the server node.

keted legacy systems to the web. This means that there was not an established web development environment available in the company that we could compare with MELIS in the user studies and for this reason we decided to use Lomboz.

In the following, we present the design and the results of the two controlled experiments according to the template suggested by Wohlin et al. [44].

4.1. Experiment definition

The goal of the two experiments was to evaluate the improvements in productivity achieved when using MELIS for the migration of COBOL programs with respect to more traditional software development environments. To this aim, we defined two migration tasks that could be performed in 3 h. The two tasks required to migrate small ACUCOBOL-GT programs to the web according to the target architecture shown in Fig. 1 and the migration process shown in Fig. 2, and verifying the functional equivalence of the original and the migrated programs. To avoid biasing the experiments by using different test case selection methods, we provided the subjects with the test case specifications to be used during regression testing. The difference between the two tasks was given by the programs to be migrated:

- T_1: migrating an ACUCOBOL-GT program that computes the basic arithmetic operations, i.e., addition, subtraction, multiplication, and division (composed of 196 LOCs);
- T_2: migrating an ACUCOBOL-GT program that computes the age of a person using his/her birthday (composed of 238 LOCs).

Both programs contained two SCREEN SECTIONs and the only difference between them concerned the presence of embedded control flows in the user interface of the second program (three controls to check the validity of the inserted month, day, and year). However, we do not expect that this slight difference might have an effect on the productivity both when using MELIS or the other development tools. In other words, the selection of a program with embedded control flows only served to make a difference in the two migration tasks, not to analyze the effect of task complexity on the productivity. Indeed, although the assessment of the legacy systems conducted in [16] revealed that the presence of embedded control flows in the SCREEN SECTIONs of an ACUCOBOL program is quite frequent and is one of the main obstacles to the automatic migration of legacy systems, the goal of these two controlled experiments was only evaluate how much the semi-automatic support to legacy system migration provided by MELIS improves the productivity compared with more traditional development tools. Further experimentations should be devoted to analyze this factor.

Each subject involved in the two experiments had to perform the two tasks across two laboratory sessions, in one case with the support of MELIS and in the other case with the support of the software development tools ACUBENCH and Lomboz. Within the migration process, ACUBENCH was used to change and wrap the COBOL programs, while Lomboz was used to reengineer the user interface. The different tool support used in the migration process is the main factor that we want to investigate. It is worth noting that there is no other commercial or open source tool available for the migration of ACUCOBOL-GT programs to the web (and this was the reason why MELIS was developed [16]), so the only possible alternative to MELIS was a combination of ACUBENCH with a web development tool.

4.2. Experiment context

To evaluate how much the use of MELIS reduces the gap between expert and novice programmers we first conducted a controlled experiment with master students and then we replicated it with professional programmers.

4.2.1. Academic context

The first controlled experiment was conducted within the Software Engineering laboratory at the University of Salerno with volunteers. In particular, the volunteers were 28 students of the Advanced Software Engineering course of the master program in Computer Science. One of the main topics of the Advanced Software Engineering course was software maintenance and reengineering. All students had a comparable level of background, as they were all graduate students with basic software engineering knowledge, good web development experience, and no COBOL programming knowledge. However, as part of the advanced software engineering course the students attended some tutorials and lab sessions on COBOL programming before conducting the controlled experiment. It is worth noting that while MELIS provides a good support to the automation of the migration process, the software development environment used within the partner company does not provide such a support, so migrating a COBOL program to the web with these tools requires to manually change and wrap the COBOL programs. Due to the lack of COBOL programming experience, there was a risk of abandonment of the experiment that we mitigated by making the students work in pairs: the 28 students were then randomly grouped in 14 teams, each composed of 2 students.

4.2.2. Industrial context

The replicated experiment was conducted within the laboratory of our partner company with four professional programmers and four academic researchers (i.e., research fellows and Ph.D. students). The practitioners had similar COBOL programming experience level, but no experience of web technologies. For this reason, we also used academic researchers with good experience of web development and no COBOL programming experience. The academic researchers attended some tutorials on COBOL programming as well as the master students of the first experiment, while the practitioners attended some tutorials on web development and J2EE technology. The four practitioners and the four academic researchers were randomly combined in four teams, each composed of a practitioner and an academic researcher. In such a way the risk of abandonment of the experiment was mitigated.

4.3. Hypotheses

The controlled experiments aimed at confirming that on average the use of MELIS reduces the migration effort (and then increases the productivity) with respect to the use of traditional development tools. Therefore, we formulated the following null hypothesis:

- H_{n1}: the use of MELIS does not significantly affect the effort to migrate an ACUCOBOL-GT program to the defined target environment; The alternative hypothesis is:
- H_{a1}: the use of MELIS significantly affects the effort to migrate an ACUCOBOL-GT program to the defined target environment;

Moreover, we wanted to investigate whether the use of MELIS reduces the gap between software engineers with different experience level. We consider the subjects of the replicated experiment (industrial context) having high experience than the subjects of the first experiment (master students) and we consider subjects of the same experiments as having the same experience level. In particular, we expect to find no significant difference in the effort produced in the two experiments when the migration tasks are performed with MELIS, while we expect to find a significant difference when MELIS is not used. Therefore, we formulated the following null hypotheses:

- H_{n2}: the subjects' experience does not significantly affect the effort to migrate an ACUCOBOL-GT program to the defined target environment, when MELIS is used;
- H_{n3}: the subjects' experience does not significantly affect the effort to migrate an ACUCOBOL-GT program to the defined target environment, when ACUBECH and Lomboz are used; The corresponding alternative hypotheses are:
- H_{a2}: the subjects' experience significantly affects the effort to migrate an ACUCOBOL-GT program to the defined target environment, when MELIS is used;
- H_{a3}: the subjects' experience significantly affects the effort to migrate an ACUCOBOL-GT program to the defined target environment, when ACUBECH and Lomboz are used;

4.4. Selected variables and experiment design

In order to properly design these experiments and analyze the achieved results, the following independent variables were considered:

- *Tool:* this variable indicates the factor on which the study is focused, i.e., using MELIS as tool support in the migration process (denoted as *PL*) or using ACUBENCH and Lomboz (denoted as *NOPL*);
- *Task:* the migration tasks, denoted as T_1 and T_2, respectively;
- *Lab:* the subsequent laboratory sessions, denoted as *Lab1* and *Lab2*, respectively;
- *Exp:* the first controlled experiment (conducted in the academic context with subjects with low experience) denoted as Exp_1 and its replication (conducted in the industrial context with subjects with high experience) denoted as Exp_2.

According to the defined null hypotheses the dependent variable to consider in our experiments was the time required to perform the tasks. This dependent variable includes both the effort to comprehend the original program and the effort required to perform the phases of the migration process described in Section 3.3, including regression testing. Different dependent variables, such as the number of produced LOCs, have not been considered, due to the small size of the selected COBOL programs and to the fact that the target architecture and the migration process are the same both when MELIS and the traditional development tool are used.

Although, we designed the experiments in such a way to avoid the task effect on the dependent variable, *Task* has to be considered as a variable of our controlled experiments and tested to evaluate that it does not impact on the productivity. The considered treatments of the controlled experiments are then represented by the possible combinations of the factors *Tool* (PL and NOPL) and *Task* (T_1 and T_2). To avoid results to be biased by group ability, each group experienced both tools and both tasks over the two laboratory sessions. As the organization of the laboratory sessions in general might affect the results of an experiment, we also needed to consider the *Lab* factor as independent variable and assessed its influence through statistical tests. A significant influence would denote the presence of a learning effect that might bias the results of the experiment. In order to minimize the learning effect, half of the teams worked in *Lab1* with MELIS and half of them without MELIS. Finally, the variable *Exp* was selected to assess the effect of the subjects' experience on the effort to perform the tasks with the different tool support.

Table 1 summarizes the design of the two experiments. In the controlled experiment performed with master students, we randomly assigned four teams to the groups A and C in Table 1, while three teams were randomly assigned to the groups B and D. On the

Table 1
Experiment design

Subject's group	Tool	
	PL	NOPL
A	T_1, Lab1	T_2, Lab2
B	T_2, Lab2	T_1, Lab1
C	T_2, Lab1	T_1, Lab2
D	T_1, Lab2	T_2, Lab1

other hand, in the replicated experiment conducted with the practitioners one team was randomly assigned to each group in Table 1.

4.5. Preparation

As mentioned in Section 4.2, the subjects of the first controlled experiment as well as the academic subjects of the replicated experiment attended some tutorials on COBOL programming, ACUCOBOL-GT programming language, and ACUBENCH while the practitioners attended some tutorials on web development, J2EE technology, and Lomboz. All subjects of the experiments successively attended a training session aimed at presenting detailed instructions on the target environment, the migration process, and the goals of migration tasks (and related COBOL programs). Finally, to give subjects more confidence with the considered tools some examples (not related to the tasks to avoid biasing the experiments) were also presented.

To assess the clarity of the migration tasks, the adequacy of the time to perform the laboratory sessions, and other related questions, as well as the perceived usefulness and usability of the different tools, the survey questionnaire shown in Table 2 was adopted. All the questions expected closed answers according to a five-point Likert scale [31]: from 1 (strongly agree) to 5 (strongly disagree). The questionnaire was the same for all treatments and each migration team filled it in at the end of each laboratory session.

Regarding the preparation of the PCs involved in the controlled experiments, we installed MELIS, ACUBENCH, Lomboz, and the technological infrastructure of the target environment.

4.6. Material, execution, and data analysis

To carry out the controlled experiments each team was provided with a folder containing a pencil, some white sheets, and the following material in hard copy:

- the introductory presentation of the experiment;
- an introductory guide of the ACUCOBOL-GT programming language and J2EE technology containing the contents of interest for the migration tasks;
- a document containing a detailed description of the reference migration process and the components of the target architecture;
- the user manuals of MELIS and the development environments ACUBENCH and Lomboz;
- the source code of the ACUCOBOL-GT programs, to be migrated in the two tasks;
- the survey questionnaires to be filled in at the end of the two laboratory sessions.

During the experiments the supervisors monitored the teams and collected the time spent while they performed the migration tasks with the development environments ACUBENCH and Lomboz. On the other hand, the effort required to migrate the ACUCOBOL-GT programs with MELIS has been automatically traced by the tool. MELIS stored the required effort in a log file.

Once the laboratory sessions of both the controlled experiments were accomplished (without time limit) the migration teams filled the survey questionnaire in and the supervisors collected the produced software artifacts (i.e., the wrapped ACUCOBOL-GT programs and the reengineered graphical user interface). The supervisors also collected the log files containing the information traced by MELIS during the execution of the migration tasks.

Statistic tests have been used to analyze the collected data. In particular, as the design of the two experiments is paired, we have used the Wilcoxon signed-rank test [15] to test the null hypothesis H_{n1}. Furthermore, we have planned to employ a two-way Analysis of Variance (ANOVA) [18] to verify the effects of other variables on the means of various groupings of a single dependent variable. The interactions between the factors as well as the effects of individual factors are tested as well. To confirm the results of two-way ANOVA the Friedman non-parametric test has also been employed. On the other hand, to verify the null hypotheses H_{n2} and H_{n3} the Mann Whitney non-parametric test [20] has been used, due to the fact that the observations are unpaired (i.e., the number of subjects within the academic and industrial context was different). This test was also used to assess the correlation between the answers to the survey questionnaire of the controlled experiment conducted within the academic context with the *Task* factor. Let us note that the results of the tests are intended as statistically significant at $\alpha = 0.05$.

Similarly to [17], to assess the overall quality of the source code (i.e., the wrapped COBOL program, the JSP pages, the servlets, and the Java beans) produced by the subjects in the migration tasks we planned to adopt an inspection process based on the Fagan's method [19]. The inspection team was composed of three academic researchers moderated by one of the authors. The inspection was blind, i.e., the inspectors were not aware of the tools used to perform the migration tasks in the different cases. The goal of the inspection process was mainly to verify whether the source code quality of the target systems is (positively or negatively) affected when MELIS is used to migrate a COBOL program to the web.

4.7. Results of the controlled experiments

In the following subsections we present and discuss the results of the controlled experiments conducted within the academic and the industrial context. The impact of the tool used in the migration tasks on the subjects' experience is described as well.

4.7.1. Academic context

The descriptive statistics of the dependent variable for the experiment conducted within the academic context are shown in Table 3. The first two rows show the statistics in case of different tool usage without taking into account the task difference, while the remaining four rows show the statistics for different combinations of tool and task. The results show that the subjects spent on

Table 2
Survey questionnaire

Id	Question
Q1	I had enough time to perform the assigned task
Q2	The objectives of the assigned task were perfectly clear to me
Q3	The task I had to perform was perfectly clear to me
Q4	The material given to me provided enough information concerning the task to perform
Q5	I believe that wrapping the ACUCOBOL-GT program was simple
Q6	I believe that that the reengineering of the original graphical user interface was simple
Q7	I believe that the integration and the deployment of the migrated ACUCOBOL-GT program was simple

Table 3
Descriptive statistics of the experiment performed in the academic context

Task	Tool	Min	Max	Mean	Median	Std. Dev.
All	NOPL	174	335	262.3571	296.5	52.59220
	PL	25	72	46.8571	48	14.44809
T_1	NOPL	174	335	250.5714	227	58.96569
	PL	25	64	45.4286	49	15.37159
T_2	NOPL	192	314	274.1429	290	46.81677
	PL	34	72	48.2857	47	14.53403

Table 4
Two-way ANOVA results on tool and task (R^2 = 89.9%, R^2(adj) = 88.7%)

Source	Type III sum of squares	df	Mean square	F	p-Value
Tool	325,081.75	1	325,081.75	212.601	0.000
Task	1222.321	1	1222.321	0.799	0.380
Interaction	750.893	1	750.893	0.491	0.490
Residual	36,697.714	24	1529.071		
Total	1,033,047	28			

the average about 47 min to perform the migration task using MELIS, while they spent on average about 262 min without using it. Therefore, the time to migrate a program using traditional development tools is on average more than five times the time required to migrate the same program with MELIS. Moreover, let us also note that the average time to perform the task T_2 (with embedded control flow) is higher than the average time to perform the task T_1 (without embedded control flow) when either MELIS or ACUBENCH and Lomboz are used.

The time to perform the migration tasks according to the considered treatments are summarized by the boxplots in Fig. 5. This figure does not present outliers and shows a good distribution of the time to accomplish the tasks T_1 and T_2 with or without the MELIS support. Let us note that the boxplots of the treatments NOPL are more skewed than the boxplots of the treatments PL, while the efforts to perform the task T_1 have a distribution more symmetric than the distribution of the efforts to perform the task T_2.

The Wilcoxon paired test reveled that the hypothesis H_{n1} can be rejected. This means that the use of the tool significantly affects the effort required to perform the migration tasks (p-value = 0.000). Thus, according to the descriptive statistics shown in Table 3 we can conclude that the use of MELIS reduces the migration effort (and then increases the productivity).

The two-way ANOVA test (see Table 4) confirmed that there is very highly significant effect of the *Tool* factor, while the *Task* factor and the interaction between the variables *Tool* and *Task* are not significant. The results of two-way ANOVA have been confirmed by the Friedman non-parametric test, which revealed that the *Tool* variable and the effort to perform the migration tasks are related (p-value = 0.000), while there is not a statistically significant relation of the distributions of the effort to perform the migration tasks and the variable *Task* (p-value = 0.109). Although, as expected the

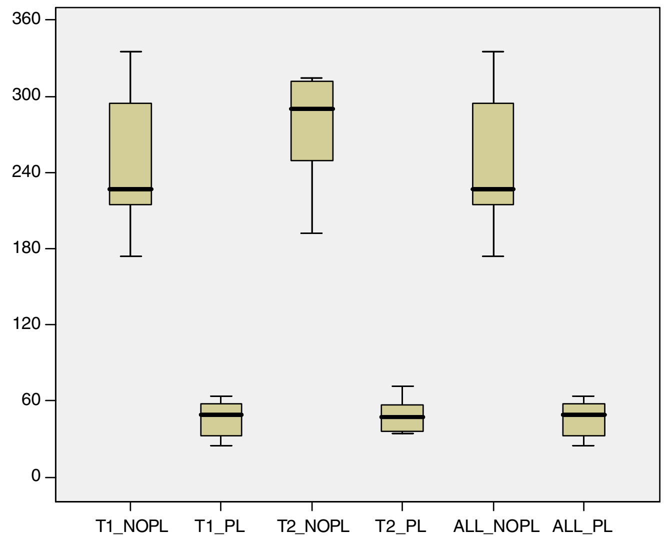


Fig. 5. Box plots of the raw times.

selected tasks do not significantly impact on the productivity, Table 3 shows that the average time to perform the task T_2 is higher than the average time to perform the task T_1, whatever tool support is used. This means that the presence of embedded control flows in the user interface might affect the effort required to migrate a COBOL program. Future work need to be devoted to this issue and to evaluate the interactions between the task size and complexity and the tool used in the migration process.

Further ANOVA analyses also revealed no significant dependence of the migration effort on the teams, demonstrating that group ability did not influence the results. Also, ANOVA revealed no dependence of the migration effort on the *Lab* sessions, indicating the absence of learning effect.

The data analysis revealed that the use of MELIS effectively supports the subjects and significantly reduces the effort required to perform the migration tasks. However, this result does not say anything about the quality of the code produced with different tool support. The inspection revealed that there is no difference in the quality of the source code in the two cases, probably due to the small size of the migrated programs and to the fact that the same migration process is used independently of the tool support.

To analyze the data collected from the survey questionnaire we considered the answers provided by each subject according to the factor *Task*. The boxplots on the left hand side and on the right hand side of Fig. 6 visually aggregate the answers of the survey questionnaire for the tasks T_1, and T_2. Fig. 6 shows that the agreement level was quite concordant and adequate for the two tasks. However, a particular consideration is deserved for the question Q7. In fact, the Mann–Whitney non-parametric test showed (with p-value 0.004) that the subjects felt the integration and the deployment of the migrated programs simpler for the task T_1 than for the task T_2. It is likely that this is due to the control checks embedded in the user interface description of the task T_2. It is worth noting that there is not a statistically significant difference for the other questions.

4.7.2. Industrial context results

The descriptive statistics of the dependent variable for the replicated experiment carried out in the industrial context are shown in Table 5. The time spent for the migration tasks was on average 48.75 min when using MELIS and 191.5 min when using ACUBENCH and Lomboz. Therefore, the effort required to migrate a COBOL program using the traditional development tools is on average four times the effort required to migrate the same program with MELIS.

The distribution of the time required to perform the migration tasks is summarized by the boxplots in Fig. 7. Let us note that the boxplots do not present outliers and show a good distribution of the time to accomplish the migration tasks with different tool support.

The Wilcoxon paired test reveled that the null hypothesis H_{n1} can be rejected. Similarly to the first experiment and according to the statistics of Table 5, we can claim that the use of MELIS reduces the migration effort also in the industrial context.

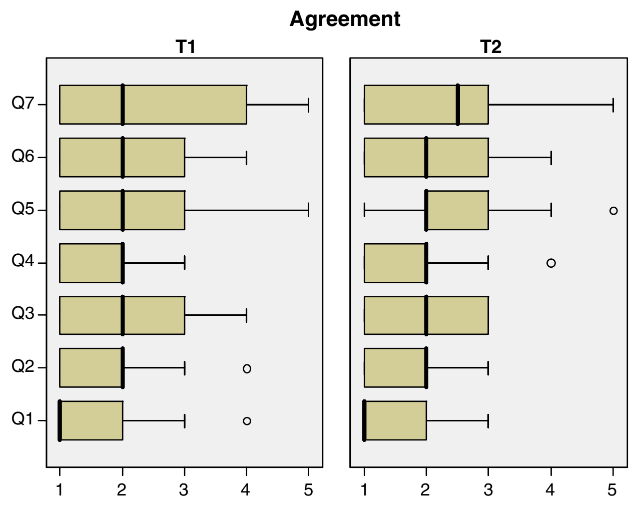


Fig. 6. Boxplots of the survey questionnaire.

Table 5
Descriptive statistics of the experiment performed in the industrial context

Task	Tool	Min	Max	Mean	Median	Std. Dev
All	NOPL	180	205	191.5	190.5	11.56143
	PL	30	69	48.75	48	17.23127
T_1	NOPL	184	205	194.5	194.5	14.84924
	PL	30	56	43	43	18.38478
T_2	NOPL	180	197	188.5	188.5	12.02082
	PL	40	69	54.5	54.5	20.5061

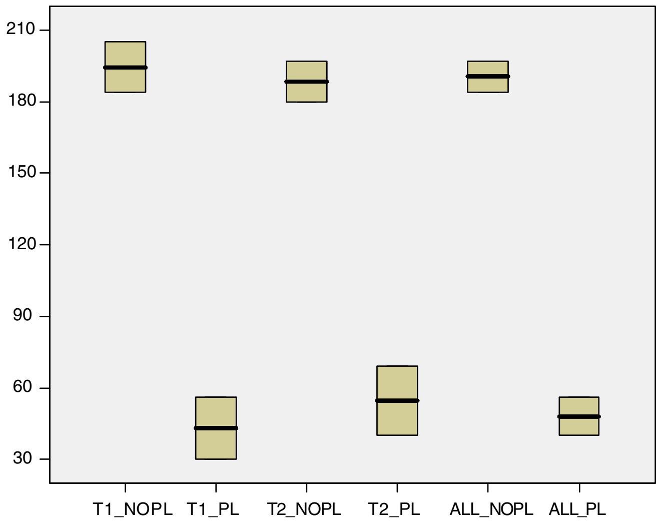


Fig. 7. Box plots of the raw times.

This result has been further confirmed by a two-way ANOVA test. As shown in Table 6, there is very highly significant effect of the *Tool* overall, while the *Task* factor is not significant as well as the interaction between the factors *Tool* and *Task*. The results of the two-way ANOVA were confirmed by the Friedman non-parametric test that revealed how the *Tool* variable and the effort to accomplish the task are related (p-value = 0.046), while the *Task* factor is not significant (p-value = 1). Again, as expected the selected tasks do not significantly impact on the productivity. However, differently from the first experiment, the average time to accomplish the task T_2 is higher when MELIS is used and lower when ACUBENCH and COBOL are used. This result might be due to the fact that the software engineers in this case are more experienced with the use of traditional development tools or more simply to the fact that in this experiment we have a limited number of observations. Again, this result calls for future investigations of this issue.

Table 6
Two-way ANOVA results on tool and task (R^2 = 97.3%, R^2(adj) = 95.3%)

Source	Type III sum of squares	df	Mean square	*F*	*p*-Value
Tool	40,755.125	1	40,755.125	145.101	0.000
Task	15.125	1	15.125	0.054	0.828
Interaction	153.125	1	153.125	0.545	0.501
Residual	1123.500	4	280.875		
Total	157,487.000	8			

A further two-way ANOVA analyses was performed to investigate the absence of learning effect. This test revealed no significant dependency of the time to accomplish the migration tasks on the *Lab* factor. ANOVA also reveled no significant dependence of the migration effort on the teams, demonstrating that group ability did not influence the results.

Similarly to the controlled experiment conducted within the academic context, an inspection process has been performed to assess the quality of the source code produced in the laboratory sessions. The inspection based process revealed that the use of MELIS does not affect the quality of the software artefacts produced within the industrial context. In particular, we observed no difference in quality between the source code produced by using MELIS and ACUBENCH and Lomboz. Again, this could be due to the small size of the migrated programs.

The data collected from the survey questionnaire that each subjects filled in at the end of the laboratory sessions are visually summarized by the boxplots of Fig. 8. In particular, this figure visually aggregates the answers of the survey questionnaire according to the tasks T_1, and T_2. The agreement level considering the answers of the survey questionnaire according to the *Task* factor can be generally considered concordant as boxplots of Fig. 8 show. However, differently form the controlled experiment conducted within the academic context, the task T_2 was considered clearer (see boxes of the question Q3). The subjects also found the wrapping, the integration, and the testing simpler when they migrated the COBOL program of the task T_2 to the web (see boxes of the question Q5 and Q7). The different results obtained within the replicated experiment could be due to the low number of involved subjects and to

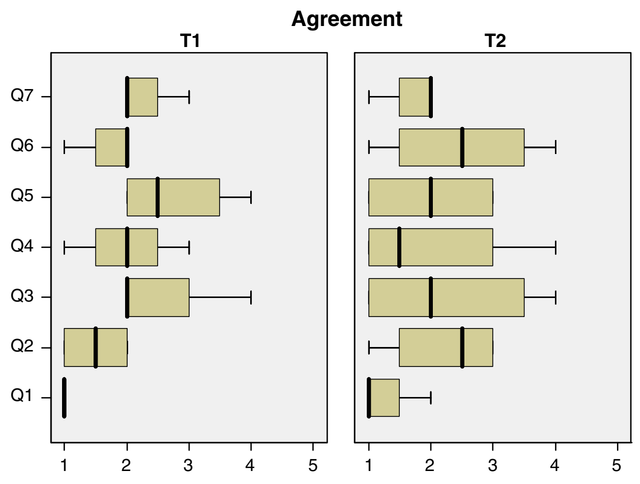


Fig. 8. Boxplots of the survey questionnaire of the replicated controlled experiment.

the higher experience of the subjects with embedded control flows. In all other cases more positive judgments were expressed for the task T_1. However, the differences cannot be considered statistically significant.

4.7.3. Evaluating the effect of the experience

In this subsection, the null hypotheses H_{n2}, and H_{n3} have been verified to investigate the effect of the subjects' experience on the effort to migrate an ACUCOBOL-GT program to the defined target environment. In order to reject these null hypotheses we investigated the effect of the factors *Tool* and *Controlled Experiment* on the dependent variable.

The Mann–Whitney test reveled that the null hypotheses H_{n2} cannot be rejected, as there is no significant relation between the experiment and the dependent variable *Time* when MELIS is used (p-value = 0.832). Indeed, as we can see from the second rows of Tables 3 and 5, the performances of the master students are very similar to the performances of the more experienced subjects of the replicated experiment. On the other hand, a significant interaction was observed (p-value = 0.019) in case the development environments used were ACUBENCH and Lomboz, thus enabling us to reject the null hypothesis H_{n3}. This means that the subjects' experience influences the effort to perform a migration task in case traditional development environment are used. In particular, as we can see from the first rows of Tables 3 and 5, less expert subjects needed more time than experienced practitioners when using traditional development environments. From these two results we can conclude that the use of MELIS reduces the gap between novice and expert subjects, or in other words less expert subjects benefit more of the use of MELIS with respect to the use of traditional tools.

4.8. Threats to validity

The threats to validity that could affect both the controlled experiments (i.e., internal, construct, external, and conclusions validity threats) are described in this section. Generally, the internal validity is only relevant in studies that try to establish a causal relationship. Thus, the internal validity threats are relevant for our study as we aimed at concluding that MELIS made a difference in the migration of COBOL programs according to the migration strategy proposed in [16]. The internal validity threats are mitigated by the experiment design, since each group of both the controlled experiments worked, over the two laboratory sessions, on two different migration tasks and with the support of the tool MELIS or the software development tools ACUBENCH and Lomboz. However, this result was confirmed by the two-way ANOVA test, which did not reveal a significant learning effect of the subjects across the laboratory sessions. This test also revealed no significant dependency of the time to accomplish the laboratory sessions. We also observed no significant dependency between the time to accomplish the tasks and the *Task* factor (i.e., T_1 and T_2) using or not the MELIS tool. This result is due to the selection of similar tasks with low differences in terms of size and complexity, although the descriptive statistics in the two experiments calls for future investigations on the effect of embedded control flows in the user interface on the effort required to migrate a COBOL program. The surveys also revealed that the subjects involved in both the experiments found clear everything regarding the migration tasks. Although the subjects applied the migration strategy and adopted the target architecture, they knew neither the objective of the experiment nor its hypotheses. Finally, we did not evaluate the teams on their performances within the laboratory sessions.

The construct validity threats that could be present in this experiment, i.e., the interactions between different treatments, were mitigated by a proper design of the experiments that allowed separating the analysis of the different factors and of their interactions. In fact, depending on the treatment, the measurement of the dependent variable was performed either analyzing the log files produced by MELIS or considering the times gathered by the supervisors of the experiments. Finally, the survey questionnaire was designed using standard ways and scales [30].

External validity refers to the approximate truth of conclusions involving generalizations. This kind of threat is always present when students are used as subjects. Nevertheless, last-year master students have a very good analysis, development and programming experience, and they are not far from junior industry programmers. To mitigate the external validity threats the controlled experiment was replicated within an industrial context. Before conducting the replicated experiment, we identified the population we would like to generalize the MELIS usage (i.e., the employees of our partner company), and then we drew a fair sample from that population and conducted our experiment with subjects belonging to this sample. Subjects were professional developers, while the academic subjects had a very good analysis, development and programming experience. Due to their experience academic subjects are not far from professional programmers. Moreover, none of the subjects of both the controlled experiments abandoned the experiments. To further confirm or contradict the obtained results, it will be worth replicating the experiment within different professional development environments and with a larger number of practitioners. Furthermore, in the future we need to investigate the effect of COBOL programs with different size and complexity, possibly selected from the ACUCOBOL-GT legacy systems of the partner company (provided that they can be migrated within a laboratory session).

The conclusion validity is the most important of the four validity types because it is relevant whenever someone is trying to decide if there exists a relationship in the considered observations. A definition of conclusion validity could be: the degree to which conclusions we reach about relationships in our data are reasonable. Regarding our experiments proper tests were performed to statistically reject the defined null hypothesis. Non-parametric tests were used in place of parametric tests where the conditions necessary to use parametric tests were not satisfied. Also, ANOVA results were confirmed by non-parametric tests (the Friedman test).

5. Case study

To evaluate how much MELIS improves the productivity of software engineers with respect to traditional development tools in a real migration context, we performed a case study. In particular, we selected the most business and critical legacy system of the partner company. This system is the oldest legacy system of the partner company, developed, evolved, and marketed over the last 30 years and it is used by many important customers of the partner company to support payroll, tax, and social security management, as well as all other accomplishments which regulate the employment relationships. The partner company delivered in 1978 the first release of this system, which was written in COBOL for a UNIX workstation. Successively, the character based user interface was migrated to Micro Focus COBOL for MS-DOS in 1986. In 1995 the system was migrated to ACUCOBOL for MS-DOS and in 2002 to ACUCOBOL-GT. The legacy system is a multi user system and has a two-tier client–server architecture with presentation and application logic running on the client and a centralized database implemented as ISAM (Indexed Sequential Access Method) files on the server. The synchronization of concurrent accesses to ISAM files is directly handled by the ACUCOBOL-GT runtime environment. The analysis of the source code evidenced that the system was not decomposable in software layers, while it was structured in loosely coupled subsystems implementing different functionalities and communicating through global variables and files. Finally, the interactions between the user interface and the application logic and database component were very intensive as also the high number of BEFORE and AFTER statements revealed. Further details on the quality assessment of the selected legacy system can be found in [16].

5.1. Case study design

We designed a balanced case study and selected two subsystems that summarize most of the problems that could be encountered during the migration of the legacy systems of the partner company. The main COBOL files of these two subsystems were named GST005.cbl and GST008.cbl, respectively, thus in the following we will refer to them as GST005 and GST008. Statistics concerning these subsystems are reported in Table 7. The GST005 and GST008 subsystems were accompanied by test case specifications to be used for regression testing and produced by the practitioners of the partner company involved in the maintenance and evolution of these subsystems.

Both GST005 and GST008 contained calls to subprograms. However, the only interactive programs are the two main programs, whereas all the subprograms are batch programs (i.e., they contain no SCREEN SECTION). It is worth noting that GST005 contains more screen LOCs than GST008. This is due to the fact that GST005 contains more SCREEN SECTIONs, entry fields, and labels with respect to GST008. On the other hand, GST008 contained a larger number of embedded controls (i.e., 8 BEFORE and 8 AFTER).

Four practitioners with comparable background and not involved in the development and maintenance of the subsystems GST005 and GST008 were randomly grouped in two migration teams (i.e., A and B). The migration of the legacy subsystems was organized in two phases (i.e., Phase1 and Phase2). In Phase1 the practitioners were asked to migrate a subsystem using the MELIS plug-in (i.e., PL), while in Phase2 they were asked to migrate a different subsystem using the development environments ACUBENCH and Lomboz (i.e., NOPL). In particular, the team A first migrated GST005 using MELIS and then migrated GST008 without using it. Similarly, the team B was asked to migrate GST008 and GST005 with and without the tool support, respectively. The design of the case study is summarized in Table 8. It is worth mentioning that in both Phase1 and Phase2 the migration teams were asked to migrate the subsystems using the target environment and migration process shown in Figs. 1 and 2, respectively. The practitioners of the partner company were also asked to prove the functional equivalence between the original and the migrated subsystems, using the test case specifications provided with the subsystems. Thus, a migration task was considered concluded when the migrated subsystem passed the test.

To provide the teams with the same knowledge on the technologies of the target architecture, on the migration environment, and the companion methods we asked them to attend some tutorials before starting the empirical investigation.

5.2. Case study results

The statistics on the subsystems GST005 and GST008 migrated with MELIS and ACUBENCH and Lomboz are reported on Table 9. This table shows the number of total LOCs of the migrated subsystems in the first row. The second row reports the automatically added LOCs, while the number of LOCs that the practitioners manually added is shown in the third row. Let us note that the number of LOCs of the manually migrated subsystems is larger with respect to the number of LOCs of the same subsystems migrated using MELIS (see Table 9). This is probably due to the fact that MELIS generally optimizes the code required to migrate ACUCOBOL-GT programs to the web.

The effort required to migrate the two subsystems with and without MELIS (expressed in terms of person/hours) as well as the effort distribution among the different phases of the used migration process is shown in Table 10. In particular, the first row shows the effort required to comprehend the original subsystem, while the effort required to wrap it is reported in the second row. The number of person/hours that the migration teams spent to reengineer the graphical user interfaces is shown in the third row. The fourth row reports the effort to integrate the wrapped legacy code and the reengineered user interface, to test the migrated subsystems, and to perform regression testing. Finally, the last row shows the total number of person/hours required to migrate each subsystem.

The effort required to migrate GST005 with ACUBENCH and Lomboz is almost seven times the effort required when MELIS is used (see Table 10), while the effort required to migrate GST008 without MELIS is eight times the effort required with MELIS. This result shows that in the case study the subjects achieved even more benefits from the use of MELIS than in the controlled exper-

Table 7
Statistics of the subsystems GST005 and GST008

	GST005	GST008
LOC	4502	3756
Number of Screen LOC	633	78
Number of SCREEN SECTIONs	4	3
Number of Label	116	20
Number of EntryField	117	10
Number of BEFORE	3	8
Number of AFTER	3	8
Number of CALL	71	23
Number of GOTOs	65	34
Number of DISPLAY	18	17
Number of ACCEPT	8	3

Table 8
Pilot project design

	Tool	Team	
		A	B
Phase1	PL	GST005	GST008
Phase2	NOPL	GST008	GST005

Table 9
Statistics on the subsystems migrated using MELIS and ACUBENCH and Lomboz

	MELIS		ACUBENCH and Lomboz	
	GST005	GST008	GST005	GST008
Total LOC	7772	4802	7821	5097
LOC automatically added	3232	1025	0	0
LOC manually added	35	21	3319	1341

Table 10
Effort required to migrate GST005 and GST008 using MELIS and ACUBENCH and Lomboz

	MELIS		ACUBENCH and Lomboz	
	GST005	GST008	GST005	GST008
Comprehension	3	3	3	3
Wrapping	1	0.5	26	24
GUI reengineering	0.5	0.5	16	14
Integration and testing	1.5	1.5	3	3
Total effort	7	5.5	48	44

iments. In other words, the results show how the advantages achieved with MELIS increase with the size and complexity of the migrated programs. However, to generalize these results further investigations are required.

5.3. Threats to validity

The threats to validity that could affect this empirical investigation are described in this section. The internal validity threats are relevant for the empirical investigation presented in this section as we aimed at concluding that MELIS made a difference in the migration of legacy systems. The internal validity threats are mitigated by the experiment design, since each team composed of professional programmers has first migrated a legacy subsystem with the MELIS support and then with ACUBENCH and Lomboz (i.e., with a reduced support to the automation of the tasks). Furthermore, the interview of the involved professional programmers revealed that everything regarding the case study was clear. Finally, the practitioners were not evaluated on the performances concerning the migration of the selected subsystems.

The interactions between different treatments (i.e., construct validity threats) were partially mitigated by the design of the experiment. Let us note that the effort required to migrate the legacy subsystems was manually gathered by the professional programmers as the case study has been performed within the company and no control was possible. A further threat to the construct validity could be due to the learning effect. However, the learning effect could only negatively condition the effort required to migrate the legacy subsystems with MELIS as the subsystems were migrated first using MELIS and then without it.

The external validity threats could be present in this experiment. However, the developed migration technologies have been experimented on real legacy systems. The subjects were also representative of the population where the result should be generalized. Furthermore, none of the subjects considered the migration complex or abandoned the experimentation. It is also worth mentioning that the subjects carried out this empirical investigation without time limit. Nevertheless, it will be worth assessing the proposed migration strategy and tool within different professional development environments with subjects with different competences.

For case studies, conclusion validity is generally hard to achieve. Confounding factors might interact with the factor under study and threaten internal validity. In the case study presented in this paper, the majority of the presented data have been directly collected by the subjects. However, conclusion validity has been mitigated by the fact that we have also performed a controlled experiment with professional programmers of the productive environment where the defined migration technologies will be adopted (see Section 4.8). Also the selection of the involved professional programmers has mitigated the conclusion validity threat. In fact, they had similar background and programming experience and are also representative of the population where we would like to introduce the MELIS tool and the accompanying methods.

6. Conclusion

In this paper, we have presented the results of two controlled experiments and a case study to evaluate the use of MELIS (Migration Environment for Legacy Information Systems) for the migration of legacy COBOL programs to the web. MELIS has been developed as an Eclipse plug-in within a technology transfer project conducted with a small software company [16]. The partner company has developed and marketed in the last 30 years several COBOL systems that need to be migrated to the web, due to the increasing requests of the customers. The goal of the technology transfer project was to define a systematic migration strategy and the supporting tools to migrate these COBOL systems to the web and make the partner company an owner of the developed technology. The main goal of the controlled experiments and case study was to evaluate the effectiveness of introducing MELIS in the partner company and compare it with the traditional development environments ACUBENCH [1] and Lomboz [26]. ACUBENCH is used by the partner company for COBOL development, while Lomboz is a widely employed open source Eclipse plug-in for web development that has been integrated in MELIS. We decided to use Lomboz since there was not an established web development environment available in the company that we could use in the assessment. In fact, while the company has a long experience of COBOL development (in particular with ACUBENCH), web development together with the migration of the COBOL legacy systems to the web are still in the future plans.

The motivation of the industrial partner for moving to the web was less a desire to be technically innovative than the need to accommodate the changing business requirements of its customers. It is the mostly the end users, who suggested the management of the company to migrated the original legacy systems towards the web. The software houses themselves are conservative. They only act when they are forced to [34,37]. Such a position of the partner company makes the technology transfer quite challenging. Accordingly, at the end of the technology development phase, that included the definition of the migration process and the development and testing of the MELIS environment (see [16] for details), we planned a two-phases technology transfer. In the first phase, we conducted controlled experiments and case studies to achieve evidence on the main improvements expected by the use of MELIS with respect to more traditional software development environments, in particular concerning the improvement of productivity and the reduction of the gap between experienced and novice software engineers. To have evidence of the experience gap reduction we performed two controlled experiments, one with master students and the replication within the industrial environment. These experiments reveled that the use of MELIS significantly reduces the time required for the migration tasks. Furthermore, we also observed that the software engineers' experience does not significantly affect the effort to migrate an ACUCOBOL-GT program, when MELIS is used. In fact, we observed that the software engineers of both the experiments spent on average the same time to accomplish the tasks using MELIS, while the effort to migrate an ACUCOBOL-GT program was significantly affected by the software engineers' experience, when using ACUBENCH and Lomboz. In particular, the practitioners of the replicated experiment required less effort to accomplish the migration tasks, with the traditional development environments. A further data analysis reveled that for the more expert software engineers the time required to conduct a migration task with MELIS is on average 1/4 of the time required when using traditional tools. On the other hand, for the master students more benefits were achieved from the use of MELIS, as this ratio is on average less than 1/5. The result is that the use of MELIS generally increases the productivity and reduces the gap between novice and expert software engineers.

We have also conducted a case study with two teams of practitioners on two subsystems of the most business critical system of the partner company with the aim of analyzing the advantages of using MELIS with the respect to the software development environments in real migration tasks. Again, the data analysis reveled that when MELIS is used the productivity improvement is seven or even eight times with respect to the use of traditional development environments. This result demonstrates how the benefit of MELIS increases with the size and complexity of the migrated programs.

Future directions have also been identified as results of the data collected within this empirical investigation. For example, we have planned to investigate the effect of the tool usage on the number of defects introduced and possibly corrected during a migration task. In particular, due to the high level of automatic support provided by MELIS to the migration process, software engineers should introduce less defects when using MELIS than when using traditional development tools, such as ACUBENCH and Lomboz. In our studies the migration tasks were considered completed when the migrated programs passed the test. Although, we did not collect and analyze the number of defects introduced and corrected during the migration tasks, we expect that the higher effort required to migrate the COBOL programs with the traditional development environments was also due to the fact that the migration activities in this case were more error prone and then more defects had to be corrected during regression testing. For example, Table 10 shows that for both the migrated subsystems the effort employed in the integration and testing phase when MELIS is used is half of the effort required for the same phase when ACUBENCH and Lomboz are used.

Although the case study only involved the most motivated and key practitioners, they can also be considered as preliminary training on the job activities. In the second phase of the technology transfer project we have actually planned training courses to be attended by all the practitioners who will be involved in the migration of the COBOL systems of the partner company. In this phase we also aim at making the partner company an owner of the migration technology, by training the industrial researchers on the methodologies and technologies used to develop MELIS and involving them in the evolution of the tool. This part of the technology transfer project is actually the most challenging.

Acknowledgement

The authors thank Vincenzo Venezia, Giovanni Vildacci, as well as the programmers of MTSys s.r.l., our partner company, for the stimulating discussion and the precious suggestions. Special thanks are also due to the practitioners of the partner company, the academic researchers, and the students who, in one way or another, contributed to carry out the experiments presented in this paper.

The work described in this paper is supported by the project METAMORPHOS (MEthods and Tools for migrAting software systeMs towards web and service Oriented aRchitectures: exPerimental evaluation, usability, and technology tranSfer), funded by MiUR (Ministero dell'Università e della Ricerca) under Grant PRIN-2006-2006098097.

References

[1] ACUBENCH, Available at: <http://www.acucorp.com/solutions/datasheets/acubench/>.
[2] L. Aversano, G. Canfora, A. Cimitile, A. De Lucia, Migrating legacy systems to the web: an experience report, in: Proceedings of European Conference on Software Maintenance and Reengineering, IEEE CS Press, Lisbon, Portugal, 2001, pp. 148–157.
[3] L. Aversano, G. Canfora, A. De Lucia, Migrating legacy system to the web: a business process reengineering oriented approach, in: M. Polo (Ed.), Advances in Software Maintenance Management: Technologies and Solutions, Idea Group Publishing, USA, 2003, pp. 151–181.
[4] V.R. Basili, R.W. Selby, D.H. Hutchens, Experimentation in software engineering, IEEE Transaction on Software Engineering 12 (7) (1986) 733–743.
[5] V.R. Basili, The role of experimentation in software engineering: past, current, and future, Proceedings of International Conference on Software Engineering (1996) 442–449.
[6] T. Bodhuin, E. Guardabascio, M. Tortorella, Migrating COBOL systems to the Web by using the MVC design pattern, in: Proceedings of Working Conference on Reverse Engineering, IEEE CS Press, Virginia, USA, 2002, pp. 329–338.
[7] T. Bodhuin, E. Guardabascio, M. Tortorella, Migration of non-decomposable software systems to the web using screen proxies, in: Proceedings of Working Conference on Reverse Engineering, IEEE CS Press, Victoria, BC, Canada, 2002, pp. 165–174.
[8] D. Bovenzi, G. Canfora, A.R. Fasolino, Enabling Legacy System Accessibility by Web Heterogeneous Clients, in: Proceedings of European Conference On Software Maintenance and Reengineering, IEEE CS Press, Victoria, Canada, 2003, pp. 73–81.
[9] M.L. Brodie, M. Stonebraker, Migrating Legacy Systems, Morgan Kaufmann, San Francisco, 1995.
[10] J.G. Butler, Mainframe to Client/Server Migration, Computer Technology Research Corp., Charleston, South Caroline, 1996.
[11] D.T. Campbell, J.C. Stanley, Experimental and quasi-experimental designs for research on teaching, in: N.L. Cage (Ed.), Handbook of Research on Teaching, Rand McNally, Chicago, 1963, pp. 1–2.
[12] G. Canfora, A. Cimitile, A. De Lucia, G.A. Di Lucca, Decomposing legacy programs: a first step towards migrating to client–server platforms, Journal of Systems and Software 54 (2000) 99–110.
[13] G. Canfora, A. Fasolino, G. Frattolillo, P. Tramontana, Migrating interactive legacy systems to web services, in: Proceedings of the Conference on Software Maintenance and Reengineering, IEEE CS Press, Bari, Italy, 2006, pp. 24–36.
[14] C.C. Chiang, Wrapping legacy systems for use in heterogeneous computing environments, Information and Software Technology 43 (8) (2001) 497–507.
[15] W.J. Conover, Practical Nonparametric Statistics, third ed., Wiley, 1998.
[16] A. De Lucia, R. Francese, G. Scanniello, G. Tortora, Developing legacy system migration methods and tools for technology transfer. Software: Practice and Experience, 2008 (to appear), DOI: 10.1002/spe.870, Available from: <http://www3.interscience.wiley.com/cgi-bin/abstract/117922806>.
[17] A. De Lucia, M. Di Penta, R. Oliveto, F. Zurolo, Improving comprehensibility of source code via traceability information: a controlled experiment, in: Proceedings of 14th International Conference on Program Comprehension, IEEE CS Press, Athens, Greece, 2006. pp. 317–326.
[18] J.L. Devore, N. Farnum, Applied Statistics for Engineers and Scientists, Duxbury, 1999.
[19] M.E. Fagan, Design and code inspections to reduce errors in program development, IBM Systems Journal 15 (3) (1976) 182–211.
[20] M. Hollander, D.A. Wolfe, Nonparametric Statistical Methods, second ed., Wiley-Interscience, New York, 1999.
[21] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, IEEE Transaction on Software Engineering 28 (8) (2002) 721–734.
[22] LegacyJ, Available from: <http://www.legacyj.com/lgcyj_perc1.html>.
[23] J.M. Lin, Z.W. Hong, G.M. Fang, H.C. Jiau, W.C. Chu, Reengineering windows software applications into reusable CORBA objects, Information and Software Technology 46 (6) (2004) 403–413.
[24] J.M. Lin, Z.W. Hong, G.H. Fang, C.T. Lee, A style for integrating ms-windows software applications to client–server systems using java technology, Software: Practice and Experience 37 (4) (2006) 417–440.
[25] S. LinkMan, H.D. Rombach, Experimentation as a vehicle for software technology transfer – a family of software reading techniques, Information and Software Technology 39 (11) (1997) 777–780.
[26] Lomboz, Available from: <http://www.objectlearn.com/index.jsp>.
[27] IBM WebSphere software: Legacy modernization with WebSphere Studio Enterprise Developer, 2002. Availabel from: <http://www.redbooks.ibm.com/redbooks/pdfs/sg246806.pdf>.
[28] E. Merlo, P.Y. Gagn, J.F. Gilard, K. Kontogiannis, L. Hendren, P. Panangaden, R. De Mori, Reengineering user interfaces, IEEE Software 12 (1995) 64–73.
[29] M. Moore, User Interface Reengineering, Ph.D. Dissertation, College of Computing, Georgia Institute of Technology, Atlanta, GA, 1998.
[30] M. Moore, L. Moshkina, Migrating legacy user interfaces to the internet: Shifting dialogue initiative, in: Proceedings of Working Conference on Reverse Engineering, IEEE CS Press, Brisbane, Australia, 2000, pp. 52–58.
[31] N. Oppenheim, Questionnaire Design Interviewing and Attitude Measurement, Pinter Publishers, 1992.
[32] T.M. Pigoski, Practical Software Maintenance – Best Practices for Managing Your Software Investment, John Wiley & Sons, New York, NY, 1997.
[33] S.L. Pfleeger, W. Menezes, Marketing technology to software practitioners, IEEE Software 17 (1) (2000) 27–33.
[34] S.A. Raghavan, D.R. Chand, Diffusing software engineering methods, IEEE Software 6 (4) (1989) 81–90.
[35] M. Rahgozar, F. Oroumchian, An effective strategy for legacy systems evolution, Journal of Software Maintenance: Research and Practice 15 (2003) 325–344.
[36] S.T. Redwine, W.E. Riddle, Software technology maturation, in: Proceeding 8th International Conference on Software Engineering, IEEE CS Press, London, UK, 1985, pp. 189–200.
[37] E.M. Rogers, Diffusion of Innovation, fourth ed., Free Press, New York, 1995.
[38] D.I.K. Sjøberg, J.E. Hannay, O. Hansen, V.B. Kampenes, A. Karahasanovic, N. Liborg, A.C. Rekdal, A survey of controlled experiments in software engineering, IEEE Transaction on Software Engineering 31 (9) (2005) 733–753.
[39] H.M. Sneed, Encapsulating legacy software for use in client/server systems, in: Proceedings of Working Conference on Reverse Engineering, IEEE CS Press, Monterey, CA, 1996, pp. 104–119.
[40] H.M. Sneed, Risks involved in reengineering projects, in: Proceedings of the 6th IEEE Working Conference on Reverse Engineering, IEEE CS Press, Atlanta, GA, 1999, pp. 204–211.
[41] H.M. Sneed, Wrapping legacy COBOL programs behind an XML-interface, in: Proceedings of Working Conference on Reverse Engineering, IEEE CS Press, 2001, pp. 189–197.

[42] H.M. Sneed, Integrating legacy software into a service oriented architecture, in: Proceedings of the Conference on Software Maintenance and Reengineering, IEEE CS Press, Bari, Italy, 2006, pp. 3–14.

[43] E. Stroulia, M. El-Ramly, P. Iglinski, Paul Sorenson, User interface reverse engineering in support of interface migration to the web, Automated Software Engineering, vol. 3, Kluwer Academic Publishers, 2003, pp. 271–301. no. 10.

[44] C. Wohlin, P. Runeson, M. Host, M.C. Ohlsson, B. Regnell, A. Wesslen, Experimentation in Software Engineering – An Introduction, Kluwer, 2000.

[45] U. Zdun, Reenginering to the web: a reference architecture, in: Proceedings of 6th European Conference on Software Maintenance and Reengineering, IEEE Comp. Soc. Press, Budapest, Hungary, 2002, pp. 211–216.