TALE: Tool for Application Logic Extraction

Kamil Rybiński, Sławomir Blatkiewicz, Norbert Jarzębowski, Wiktor Nowakowski, Michał Śmiałek

Warsaw University of Technology, Warsaw, Poland

Abstract

Developing a new software system based on legacy software can be quite hard and labor-intensive. The main issue is to preserve the essential application and business logic – normally resolved in a manual process. The presented tool suite automates capturing essential knowledge on the application logic of legacy systems by recording their observable behaviour. The recovered knowledge is represented with use case scenarios having precise sentences describing user-system interactions. The tool suite has a heterogeneous architecture and is composed of several Eclipse-based applications. The central idea is to use a standard test automation system to capture test scripts and then translate them into constrained natural language scenarios that are machine processable. This allows for further automatic transformations even down to code. The paper presents the tool design details and experience gained during its development.

1. Introduction and related work

In this paper we present a tool for extracting application logic information from legacy systems to facilitate further easy migration into new system design. Application logic carries information about the user-system dialogue in relation to domain-specific data processing and platform-specific user interface appearance. In our solution, such information can be extracted from any legacy system by determining its observable behaviour. This information can then be stored in the form of requirements-level models written in the Requirements Specification Language which has precise formal specification of its syntax [1]. These models can be then transformed into architectural and design-level models or even into code.

Figure 1: Overview of the recovery process

The recovery process is illustrated in Figure 1. The legacy system is subject to "UI ripping" (recording the observable behaviour) as available in test automation tools. This produces test scripts in the XML format. The scripts are processed by the TALE tool developed within the REMICS project (www.remics.eu). This automated process results in RSL models that can be further manually refined in the ReDSeeDS tool (www.redseeds.eu). This process was described in detail by Nowakowski et al. [2]. It can be also compared to that proposed by Hungar et al. [3]. However, this solution generates low-level state models and is limited to recovery of telecommunication systems and concentrated on generating test cases. Here we present the details of the TALE tooling environment that is suitable for a wide range of business systems which exhibit high intensity of user-system interactions. The models generated by TALE are suitable for further transformation into modern technology code, at the same time preserving the application logic of the legacy system.

It can be noted that most solutions to knowledge recovery from legacy systems concentrate on retrieving code or detailed design artifacts. This was formalised through the introduction of the Knowledge Discovery Metamodel [4], with MoDisco (www.eclipse.org/modisco) and Netfective Blu Age (www.bluage.com) being one of its first implementations. Other approaches include data analysis solutions like Data Reverse Engineering [5] or Database Reverse Engineering [6]. According to our best knowledge there are no similar solutions where application logic is recovered based on observable behaviour. Most work regarding reverse engineering of (graphical) user interfaces concentrate on testing purposes (see eg. work by Memon et al. [7]) and direct migration into new user interface technology (see eg. Stroulia et al. [8]). In the presented solution, the application logic is recovered in the form of requirements-level scenarios that facilitate discussion on the possible changes in functionality. By contrast, there seem to be no other advanced tools that focus on requirements recovery. Sparse examples include work by Fahmi and Choi [9] and the RETR initiative [10]. However, there seems to be no toolkit that supports these ideas. Recently, an approach by Repond et al. [11] proposes to formulate the recovered system knowledge in the form of use cases. Though, in contrast to the TALE tool, it does not offer any means to auto-

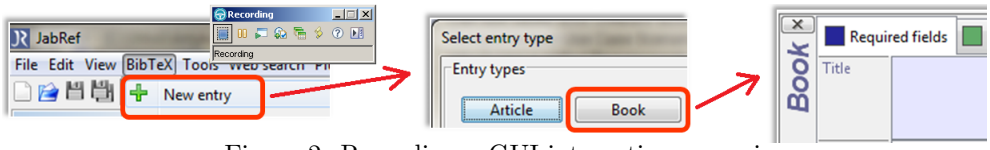

Figure 2: Recording a GUI interaction scenario

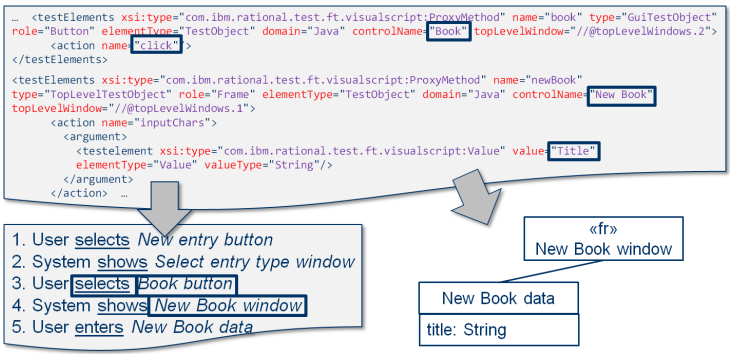

Figure 3: Translating from a GUI recording to requirements in RSL

mate the recovery of scenarios. Some approaches, like the one by Bertolino et al. [12] propose synthesizing behaviour protocols that contain similar information to that of use case scenarios. However, this is based on using the existing implementation artifacts or design models, while our approach is independent on any "internals" of the legacy system.

2. Tool suite requirements and pre-existing components

To capture legacy application logic we need to process and store information on all the significant paths through the user interface, including exceptional behavior (eg. entering invalid data, operation cancellation). Thus, the important requirement of the new tool suite is to be able to record all the possible user-system interaction paths of a legacy system. One of such paths is illustrated in Figure 2. Here we can see a short scenario for entering a new book entry using the JabRef reference manager. We would thus want to "play-out" many such scenarios through normal usage of the system and record their steps and data exchanged with the user (cf. UI ripping). It can be noted that such recording is present in typical test automation tools.

The above requirements led to choosing Rational Functional Tester (RFT) [13]. Its main purpose is automation of functional and regression testing. Capturing and simulation of user actions can be performed for various user interface styles and technologies. The captured functionality ("test scripts")

is editable and presented together with the UI screens. RFT uses an object map between the script and the application under test, thus providing detailed information about data objects engaged in the interactions. The test scripts and objects can be exported into machine processable scripts in XML. This is illustrated in Figure 3 (upper half) where the XML file contents reflect the scenario and the data (the book title) from Figure 2.

There were also made attempts to utilise other test automation tools, but none of them met the presented requirements as fully as RFT. They capture only some details of the user-system interaction or store the captured information in a form difficult to process. HP Unified Functional Testing software [14] and SmartBear TestComplete [15] are some examples of the analysed tools. Unfortunately, the extent and form in which the captured objects are stored by these tools were not satisfactory for the recovery process. There were also additional problems with the range of supported languages and technologies. For example, we could not use tools such as Selenium [16] because they only support browser-based applications.

Test scripts recorded by RFT need further processing. Their purpose is not to capture application logic units but to capture linear paths through the system behaviour for further repeated automatic test execution. Thus, the new tools should be able to translate and merge such scripts into coherent human-readable high-level models representing units of application logic. It can be noted that this can be very well packaged into familiar use cases with their potential to be processed by the techniques of Model Driven Engineering (see eg. work by Astudillo et al. [17]). What is also needed is means to store the use case scenarios as models. This is offered by the Requirements Specification Language which is unique through its very detailed metamodel. Moreover, RSL has a comprehensive and mature tool suite - the ReDSeeDS Engine [18]. Thus, the new tools should be able to create RSL models based on the recorded test automation scripts. This is illustrated in Figure 3. The scenario from Figure 2 is now translated from an XML file into a use case scenario containing 5 simple subject-verb-object sentences. This is supplemented by a domain model containing information on windows (e.g. "New book window") and associated data (e.g. "New book data").

Such translation should be done automatically. Moreover, it should be possible to merge several similar scenarios into use cases. The tool should also support modification and correction of the translation results. This is particularly important in situations where the person recording the scripts did not properly perform recording actions (eg. made the wrong choice for

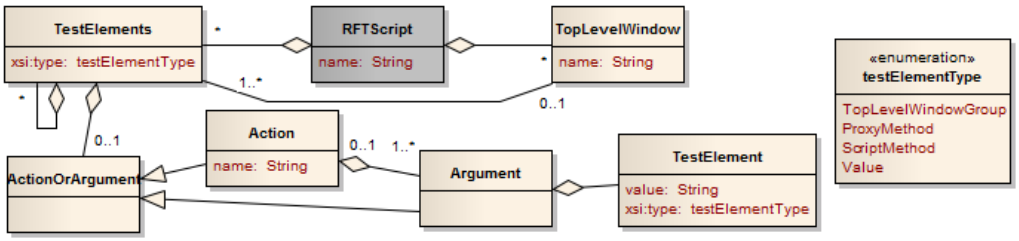

Figure 4: Elements of the script structure important for translation

the script beginning and end points). Therefore, the tool should provide features for dividing and merging scripts to properly reflect the use case goals.

In summary, the analysis resulted in selecting RFT and ReDSeeDS Engine as the first and last component in the recovery path. What was still necessary to develop was the tool to transform test-related scripts into application logic units (use cases with scenarios). This resulted in constructing TALE - Tool for Application Logic Extraction, as presented in the further sections. It should be noted that both RFT and ReDSeeDS were developed within the Eclipse framework, and thus it was a natural choice also for TALE.

3. TALE design issues

3.1. Interfacing with RFT

Understanding the RFT script structure is fundamental to proper extraction of information. Each script is an XML file containing activities grouped by the UI elements in which they have occurred. To understand its structure and prepare for transformation into RSL, the RFT script "language" has been reverse engineered into a metamodel. Its simplified structure is presented in Figure 4. The scripts are composed of test element groups (cf. **TestElements**). Every recorded window has its xsl:type set to **TopLevelWindowGroup**. Each element of this kind holds a reference to its description stored separately in a list of all windows (cf. **TopLevelWindow**). A window contains elements that the user has interacted with, typed as **ProxyMethod**s. Each such method has an **Action** - a click, a text input or key press. Actions have **Argument**s which refer to **TestElement**s that specify action attributes and values (e.g. entered text or pressed keys). At the window group or window element representation level, there can exist test elements typed as **ScriptMethod**s which represent calls to other scripts.

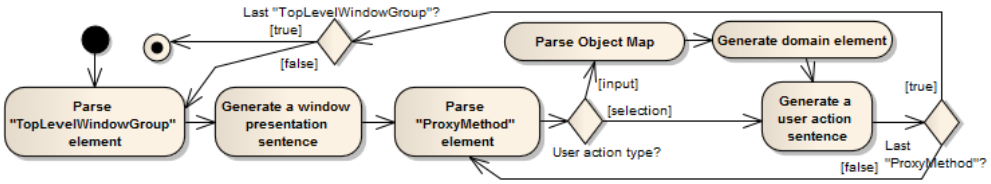

Figure 5: RFT to RSL transformation algorithm (simplified)

Test script recording also results in creating special structures (separate XML files, called object maps) containing objects with information on data elements and its types exchanged with the user. For brevity we will not present their structure here.

3.2. RFT script to RSL transformation

The recorded RFT scripts in XML format are parsed and transformed into a model compliant with the RSL metamodel. This model is stored within the ReDSeeDS repository (model storage) that is based on the JGraLab [19] technology. Since this repository is graph-based, this step involves creating a graph structure. From the user point of view, the translation results in a series of simple RSL sentences, user interface elements and domain concepts reflecting the contents of the XML file, as illustrated in Figure 3.

The transformation algorithm was coded in plain Java using a standard XML parser library (see Section 3.4) and the JGraLab API. Its overview is presented in Figure 5. The algorithm is based on gathering information from consecutive TopLevelWindow groups. For each of such groups, a window presentation sentence is generated (cf. sentences 2 and 4 in Fig. 3). Furthermore, each element under the current top level window is processed. The recorded user actions referenced by these elements result in generating either UI element selection sentences (cf. sentence 3), or data input sentences (cf. sentence 5). For the latter situation, the associated Object Map file is parsed and sought for associated data objects. Based on this, appropriate RSL domain elements (notions) are generated.

It can be noted that the target sentences observe a simple subject-verb-object format (sometimes with two objects). The rule is that the subject is set to *system* for the window display sentences. In other cases the subject is set to *user*. The verb is set to *select* for the selection or button pressing sentences; *enter* is used for the data input sentences and *show* is used for the window display sentences. The sentence objects are set to either window,

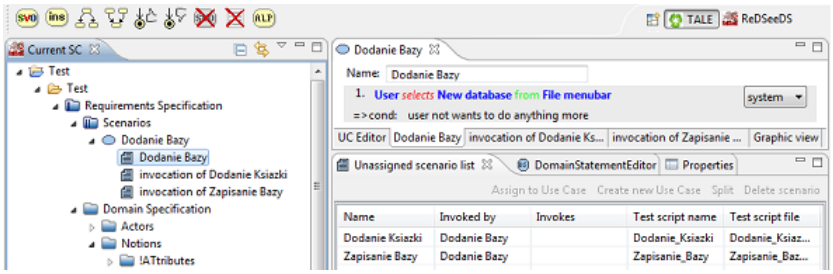

Figure 6: Sample window of the TALE tool

button or domain elements. The object name is taken from the source script and an appropriate postfix is added (cf. "Book window" vs. "Book data").

As mentioned above, before any data input sentence (cf. sentence 5) is added, a domain notion (cf. domain concept), representing this data, is created. All the input data element types are identified inside the proper Object Map file and marked with appropriate data types. They are then added to the domain specification (cf. the "Book data" element). All the associated primitive values from the map are set as the domain notion's attributes.

The full transformation results in creating a scenario for each of the RFT scripts. For all the scenarios, a single domain model is created that contains all the domain notions, window elements and button elements linked through associations (cf. Fig. 3, bottom right).

3.3. Scenario management

After RFT script processing, it should be possible to manipulate the RSL scenarios to construct complete and coherent use case models to reflect the observable functionality of the legacy system. This is performed in a special scenario editor, as illustrated in Figure 6. The recovered scenarios are displayed in the **Unassigned scenario list** (see bottom-right). These scenarios are not yet related to any use case. The TALE user can group them into use cases by creating new use cases or appending to already existing ones. When attaching to a use case, the user can choose a reference scenario and point to a correct joining place. This also adds condition sentences to both scenarios. The previously (possibly erroneously) attached scenarios can also be detached back to the unassigned scenario list. The user can also delete scenarios from the list, join them or split them. It is also possible to move scenarios between use cases, merge use cases or notions and automatically find common scenario fragments. This last feature uses in its implementation

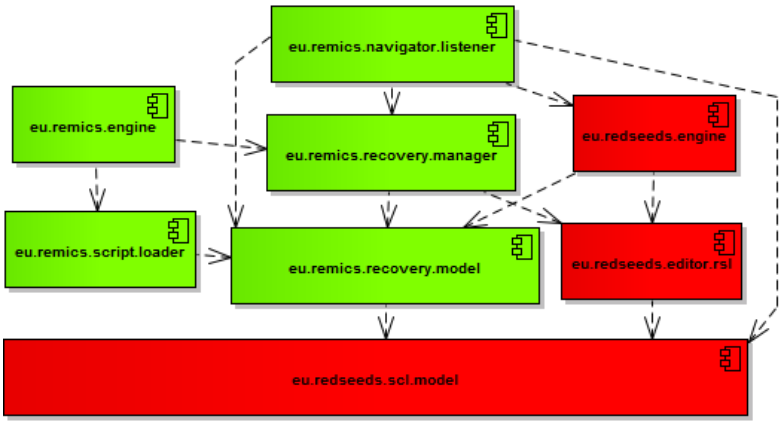

Figure 7: TALE main architectural components

the Rabin-Karp algorithm [20] for detecting same scenario fragments. Single sentences act as string patterns.

Figure 6 illustrates the package structure of the edited model, maintained within the tool. Some packages contain the created use cases, some contain the domain notions (see left). The scenarios and domain notions can be further edited and extended according to newly emerging requirements (see top-right). It can be noted that the example uses a real-life system from Poland, and the retrieved data is partially in Polish.

3.4. Technology and architecture

TALE was implemented as an extension for the ReDSeeDS tool, within the framework of Eclipse Rich Client Platform (RCP). The test script parsing algorithm implementation uses the Xerces Java Parser – an XML parser from Apache Xerces [21]. Worth noting is the possibility to switch between TALE and ReDSeeDS (arranged as perspectives) seamlessly since both tools are integrated within a single framework and they share the common RSL data model. Additionally, TALE uses GMF plug-ins for handling graphical diagrams with the underlying EMF model [22] modified to serve as a proxy layer for the JGraLab model.

The general structure of the tool components (Java plug-ins) is shown in Figure 7. The relevant ReDSeeDS components are marked red (darker) and the TALE plug-ins are marked green (lighter). Generally, the system is functionally divided into domain logic and application logic (combined with the UI). The domain logic is handled by two components: *redseeds.scl.model* implements the original RSL metamodel (part of a broader

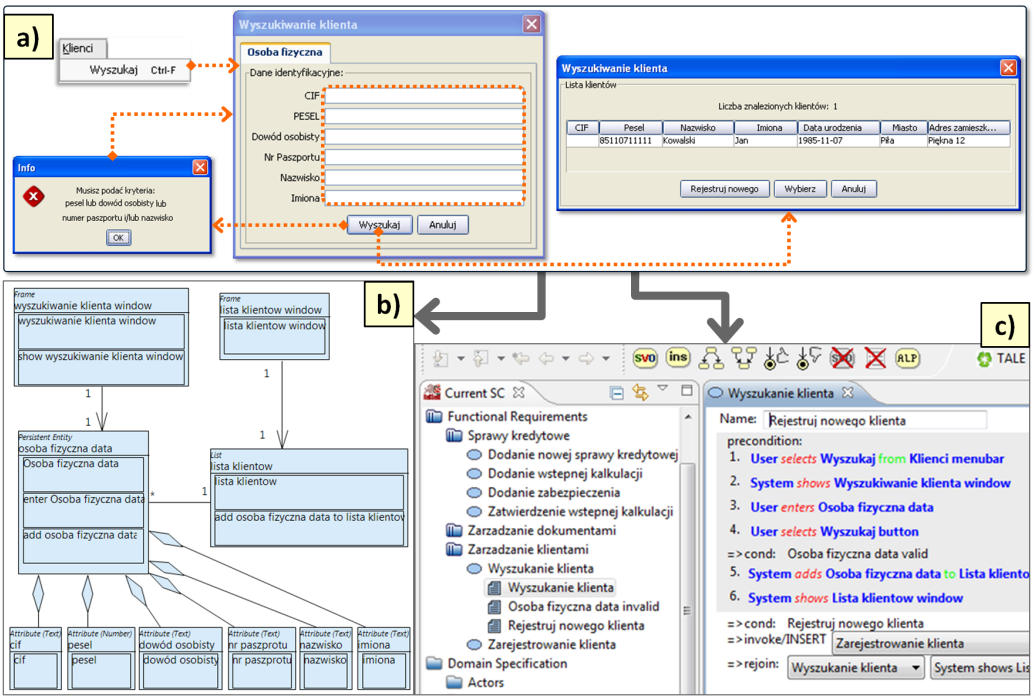

Figure 8: Illustration of three main steps of the recovery process

SCL metamodel); *remics.recovery.model* implements script processing and transformation, communicating frequently with *redseeds.scl.model*. The main TALE observable functionality, as presented in the previous two subsections, is contained in two application logic components: *remics.script.loader* and *remics.recovery.manager*. These two components are supported by the RSL editor (*redseeds.editor.rsl*) and the project tree manager (cf. *redseeds.engine* and *remics.engine*). The *navigator.listener* component supplements this functionality by reacting to changes in UI elements and updating the current model.

4. Case study example

The presented toolkit has been validated through recovering a non-trivial commercial system in the bank loan management domain. The system, called SZOK, has been developed by a Polish major software provider Infovide-Matrix, and was discontinued from further development a couple of years ago. The recovery process resulted in creating more than 50 full use cases, each with 2 or more scenarios. An example, pertaining to one of the use

cases, is presented in Figure 8.

During normal usage of the SZOK system, the flows of interaction were recorded using RFT. Illustration in Figure 8a shows an example of such interaction leading from selecting a menu option (Klienci → Wyszukaj; Clients → Search) to obtaining a list of matching clients (with a possible "detour" for invalid entered data). In the next step of the recovery process, the TALE tool has transformed the recorded scripts into an initial RSL model. This model was then manually "wired up" so that individual scenarios were connected into full use cases. Figure 8c shows the automatically generated and manually connected scenario, reflecting the user-system interaction illustrated in Figure 8a.

It can be noted that the TALE tool also re-creates the domain model containing domain notions and UI elements used in the recovered scenarios. What is important, the tool is able to extract information about the composition of notions. Such notions aggregate attributes for every entry field from the respective forms. This is illustrated in Figure 8b which contains an automatically generated domain and user interface element model recovered from the recorded interaction. For example, from the "Client search" (pol. "Wyszukiwanie klienta") window, the tool generates the "physical person" (pol. "osoba fizyczna") data notion that aggregates six attributes present in the form.

The final step is to refine the RSL model to cater for possible modifications and extensions to the system's functionality. Often, the domain model needs manual refactoring due to required renaming of recovered notion names. This is done in the ReDSeeDS perspective of our tool suite and is illustrated in Figure 8c (some notions renamed and several scenarios "wired" to compose for the "Wyszukanie klienta" / "Client search" use case).

5. Discussion and conclusion

The development of the TALE tool took around 20 man-months. The team consisted of the authors of this paper. The effort involved appropriate research and implementation of the RFT-to-RSL transformation algorithm and the scenario management editor. The tool is a crucial element of wider research to develop an effective method to migrate legacy systems to modern technologies. It has to be stressed that the requirements models generated and managed within the tool can be further used to generate code in a wide range of technologies [23]. The current results show that full (dynamic) code

of the upper layers (view, controller/presenter in MVC/MVP architectures) can be automatically generated. The only disadvantage is that the data processing layer (model in MVC/MVP) has to be migrated using other methods.

The effort associated with migrating legacy applications using TALE is concentrated in recording and merging user-system interaction scenarios. This would involve instructing regular users of the legacy system to cover all the paths that are intended for migration. Then, the recorded paths would need to be merged into use cases. It can be noted that in contrast to typical reverse-engineering methods, the above activities do not involve workforce with advanced skills. Further generation of the new system is fully automatic. In summary, the "manual" effort to migrate a system consists of three elements: playing out scenarios + merging scenarios into use cases + updating the generated system with data processing/storage algorithms. In case of lack of legacy documentation and/or lack of legacy source code, this approach seems to be the only economical solution. Also, in case when the legacy source code is available, its re-engineering might often be very difficult (cf. GOTO statements in legacy code etc.) and thus not economical.

In regard to implementation of TALE, several interesting observations can be emphasised. The plug-in architecture of RCP has significantly facilitated interfacing and reusing the ReDSeeDS components. Overall, the Eclipse environment provided a coherent workspace for the project, despite significant learning curve associated with its various elements. A prominent example is the GMF/EMF framework [22] used to develop the graphical model editors. It allowed us to quickly transform a metamodel (like the one shown in Fig. 4) into a rich graphical editor. Still, GMF/EMF lacks satisfactory documentation which leads to quite significant overhead associated with mastering this environment. Moreover, in the context of ReDSeeDS, we have experienced overhead due to incompatibility between the EMF and the JGraLab storage. On the other hand, JGraLab has proven to be a very efficient and easy to apply model repository system. It provides a very rich low-level API with additional high-level wrappers for common complex tasks.

The ultimate goal for the research around TALE is certainly very practical. It can be noted that the tool can recover the logic of practically any software system in respect to its observable behaviour. This makes the tool completely independent of the legacy system's internals (often very "twisted" and not recoverable by other means). It can be noted that TALE can be easily interfaced with other (G)UI ripping tools. This would necessitate changes only to the *remics.recovery.model* component that currently processes RFT

scripts (XML). This gives vast possibilities for recovering application logic for various types of user interfaces and technologies.

Acknowledgment This research has been carried out in the REMICS project (`http://www.remics.eu`) and partially funded by the EU (ICT-257793 under the 7th Framework Programme).

References

[1] H. Kaindl, M. Śmiałek, P. Wagner, et al., Requirements Specification Language definition, Tech. Rep. D2.4.2, ReDSeeDS Project (2009).

[2] W. Nowakowski, M. Smialek, A. Ambroziewicz, N. Jarzebowski, T. Straszak, Recovery and migration of application logic from legacy systems, Computer Science 13 (4) (2012) 53–70.

[3] H. Hungar, T. Margaria, B. Steffen, Test-based model generation for legacy systems, in: IEEE International Test Conference (ITC), IEEE Computer Society, Charlotte, NC, 2003, pp. 971–980.

[4] R. Pérez-Castillo, I. G.-R. de Guzmán, M. Piattini, Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems, Comput. Stand. Interfaces 33 (6) (2011) 519–532.

[5] P. Aiken, Reverse engineering of data, IBM Systems Journal 37 (2) (1998) 246–269.

[6] J.-L. Hainaut, M. Chandelon, C. Tonneau, M. Joris, Contribution to a theory of database reverse engineering, in: Reverse Engineering, 1993., Proceedings of Working Conference on, 1993, pp. 161–170.

[7] A. M. Memon, I. Banerjee, A. Nagarajan, GUI ripping: Reverse engineering of graphical user interfaces for testing, in: Proceedings of the 10th Working Conference on Reverse Engineering, 2003, pp. 260–269.

[8] E. Stroulia, M. El-Ramly, P. Iglinski, P. Sorenson, User interface reverse engineering in support of interface migration to the web, Automated Software Engineering 10 (3) (2003) 271–301.

[9] S. Fahmi, H.-J. Choi, Software reverse engineering to requirements, in: Proc. Int. Conf. Convergence Inform. Technol., 2007, pp. 2199–2204.

[10] Y. Yu, J. Mylopoulos, Y. Wang, S. Liaskos, A. Lapouchnian, Y. Zou, M. Littou, J. Leite, RETR: Reverse engineering to requirements, in: Reverse Engineering, 12th Working Conference on, 2005, p. 234.

[11] J. Repond, P. Dugerdil, P. Descombes, Use-case and scenario metamodeling for automated processing in a reverse engineering tool, in: Proc. 4th India Software Eng. Conf., ISEC '11, 2011, pp. 135–144.

[12] A. Bertolino, P. Inverardi, P. Pelliccione, M. Tivoli, Automatic synthesis of behavior protocols for composable web-services, in: Proc. 7th ESEC/FSE '09, 2009, pp. 141–150.

[13] C. Davis, D. Chirillo, D. Gouveia, et al., Software Test Engineering with IBM Rational Functional Tester: The Definitive Resource, 1st Edition, IBM Press, 2009.

[14] HP Unified Functional Testing page, `http://www.hp.com/go/uft`.

[15] SmartBear TestComplete page, `http://smartbear.com/products/qa-tools/automated-testing-tools`.

[16] Selenium home page, `http://docs.seleniumhq.org/`.

[17] H. Astudillo, G. Génova, M. Śmiałek, J. Llorens Morillo, P. Metz, R. Prieto-Diáz, Use cases in model-driven software engineering, Lecture Notes in Computer Science 3844 (2006) 262–271.

[18] M. Smialek, T. Straszak, Facilitating transition from requirements to code with the ReDSeeDS tool, in: Requirements Engineering Conference (RE), 2012 20th IEEE International, IEEE, 2012, pp. 321–322.

[19] JGraLab project home page, `https://github.com/jgralab`.

[20] R. M. Karp, M. O. Rabin, Efficient randomized pattern-matching algorithms, IBM Journal of Res. and Dev. 31 (2) (1987) 249–260.

[21] Apache Xerces project page, `http://xerces.apache.org/`.

[22] R. C. Gronback, Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit, Addison-Wesley, 2009.

[23] M. Smialek, N. Jarzebowski, W. Nowakowski, Translation of use case scenarios to Java code, Computer Science 13 (4) (2012) 35–52.