Case Study: Using Model Based Component Generator for Upgrade Projects

PETAR RAJKOVIC, University of Nis, Faculty of Electronic Engineering
IVAN PETKOVIC, University of Nis, Faculty of Electronic Engineering
DRAGAN JANKOVIC, University of Nis, Faculty of Electronic Engineering

To improve development of the medical information system components, we have introduced model based code generation tool. It was of great help when we needed to develop series of components sharing the same set of basic functionalities – predominantly data collection forms. Since we faced many functionality update requests, we tend to check the usability of our model driven approach in cases when legacy components developed under the older version of same programming framework need to be included in our system. In this paper we described the set of updates needed for our code generation components as well as two most important use cases – when existing functionality from legacy system has to be extended and when significant portion of legacy system needs to be adapted and included in new system. Overall conclusion is that model driven approach is still useful even with upgrade projects, but required side effort is higher, especially when the first component in a row is getting adapted.

Categories and Subject Descriptors: **H.4.0 [Information Systems]**: General; **H.5.2 [User Interfaces]**[1]**:** *User interface management systems (UIMS)*; I.6.5 **[Simulation and modeling]**: Model development

General Terms: Human Factors, Design

Additional Key Words and Phrases: Model driven development, Code generation, Upgrade projects

1. INTRODUCTION AND MOTIVATION

Working on the development of medical information systems (MIS) for a decade and a half, we got experience with many different kind of projects – from simple demonstrational pilots to complex upgrade and the integration with legacy systems. The projects that focus on the upgrade, migration and the integration with other systems brings a complete new set of organizational and technical problems. In this paper, we present the results of our case study focused on the usage of model driven development (MDD) approach in various types of MIS extension projects.

To improve overall MIS lifecycle, we introduced data modeling and generation tools [Rajkovic et al, 2015] that helped us mostly in development of the components sharing the same set of basic functionalities. In addition, we defined a framework around used software development methodologies to choose a proper approach in relation to the type of targeting project [Rajkovic et al, 2016]. Beside both suggested updates proved to be useful with new developments, the real challenges came with the upgrade projects [Gettinger 2012]. In the upgrade projects, we were often faced with the requests to integrate external pieces of software or even to extend them. In some cases, including legacy functionalities into our MIS was required. Since the targeting legacy projects can vary in terms of technology and standards, a lot of work was expected. Luckily in some cases, we noticed many common functionalities within legacy components and decided to try to use our existing data generator tool to help us with software upgrade. For specific parts of the upgrades, we had to develop the additional pieces of code to adapt targeting components to our MIS.

This work is supported by the Ministry of Education and Science of Republic of Serbia (Project number #312-001).
Authors' addresses: Petar Rajkovic, University of Nis, Faculty of Electronic Engineering – Lab 534, Aleksandra Medvedeva 14, 18000 Nis, Serbia, e-mail: petar.rajkovic@elfak.ni.ac.rs; Ivan Petkovic, University of Nis, Faculty of Electronic Engineering – Lab 523, Aleksandra Medvedeva 14, 18000 Nis, Serbia, e-mail: ivan.petkovic@elfak.ni.ac.rs; Dragan Jankovic, University of Nis, Faculty of Electronic Engineering – Lab 105, Aleksandra Medvedeva 14, 18000 Nis, Serbia, e-mail: dragan.jankovic@elfak.ni.ac.rs;

In: Z. Budimac (ed.): Proceedings of the SQAMIA 2017: 6th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Belgrade, Serbia, 11-13.09.2017. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

Since many of upgrades are based on *adapter* and *visitor* patterns [Kim et al, 2017] we realized that the set of classes that have to be developed will significantly differ from the windows forms which are the primary output of our code generation tool. We choose to extend our generator tool with template based approach and tried to use common templates as much as possible. The additional focus was put on testing phase. Since we needed to verify that both sides of the system work as expected, we conducted the initial tests using generated test vectors and unit tests. Generator tool used to take the same part of data model and generate both component and tests which can be run immediately (having in mind known potential problems with automatically developed tests [Palomba 2016]).

As it has been stated, the proposed update relies on our existing model based generation tool which proved as efficient in modelling and implementing new components. It has been extended with the new set of templates and functionality. It was important not to improve only development, but also testing phases. As an evaluation, we will demonstrate the process and effects for two, from our point of view, common cases – extending existing functionality with new category of options and integrating parts of legacy project into our information system. It is important to say, that examined cases differ from our MIS in architecture and coding standards, but they used the older version of the programming framework, so they can be assumed as technologically compatible.

2. RELATED WORK

The problem with the integration of legacy systems is well known in the area of medical informatics. Choosing between the integration of new functionalities into legacy system, and including legacy functionalities into the new system cannot be assumed as a "happy flow" [Wiegers 2013], but as an exceptional case within software development. Many different approaches can be found in the literature, and many authors are aiming to contribute for future standardization. Current standards such are [HL7, 2013] and openEHR [Kalra 2005] are well descriptive in terms of entities and attributes, but there is not a clear streamline in the guidelines related to upgrade processes.

When the integration with legacy system is needed, a large group of authors focus on data exchange standards, such is [Khan et al, 2014] and [Sachdeva et al, 2012], avoiding full integration. They presented state of the art adaptive interoperability engine that ensures data accuracy. Generated data exchange models are designed in a way that can easily follow future changes and requirements. Introducing data exchange standards is a good way for a situation when end users tend to continue to use the old system, or even when they need to work with several standards [Schloeffel et al 2006].

Since we had requirements to integrate the old functionalities, we used their approach to map the data from the old system before they are merged to a new one. The additional problem with mapping is the fact that both databases are usually large enough that simple whole-dataset-copy approach requires a lot of time and cannot be used effectively [West 2013]. Thus, while working on mappings we used sample extraction method as suggested in [Pageler et al 2016].

Interesting project is described in [Duftschmid et al, 2013]. The authors tried to integrate some 27 new archetypes into legacy system. They managed to do so with 15 of them, but the amount of work needed was higher than expected. Initially we have an idea to first improve legacy software to match the current standard, but since the integration with the new system was required we decided to go with partial adaptation of existing functionalities and include them with the new software.

Eventually we decided to extract Entity-Attribute-Value items and use them as a data model for our modeling tool as suggested in [Kalra et al, 2007] and [Duftschmid et al, 2010]. For this purpose we used reverse engineering tool of our data modeling framework [Rajkovic et al, 2015]. After identifying main entities and their extension points, we were able to define proper mappings and adaptation scenarios.

3. EXTENDING DATA MODELING FRAMEWORK

Data modeling and code generation tools are the part of our information system development framework for many years. We have started with the development in 2009, and first results were published in [Rajkovic et al. 2010]. Further updates and usage overview are presented in [Rajkovic et al. 2014] and [Rajkovic et al. 2015]. So far, our data modeling framework proved as suitable for the new developments and prototype building. The usage in the upgrade projects was limited due to the lack of testing support. It leads to the set of updates shown in Figure 1. These updates include an additional code generation routine needed to support test generation and immediate automated testing of generated components. The idea was to update template based code generation. The concept is as following: the code generation tool will a load template code file where specific parts are represented by the placeholders. The code generation tool will then search the data model and replace the placeholders with mapped model attributes. This approach give less flexibility comparing with initially defined CodeDOM [Hinkel 2016] definitions, but due to its simplicity, we used to get faster results, especially in the cases when we needed to generate the adapter class various entities from the legacy project.

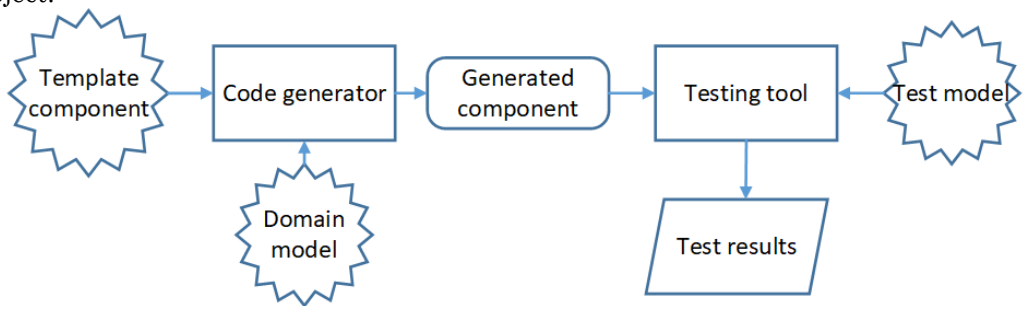

Fig. 1 Extended concept of code generation tool

```
public void ExecuteAddEntityTests(Type entityToTest, List<TestVektor> testVektori)
{
    object form = Activator.CreateInstance(entityToTest);
    IDocumentProperties dokument = form as IDocumentProperties;

    foreach (TestVektor vektor in testVektori){
        try{
            object pregled = dokument.GetPregled();
            object referenceToPregledCollection = pregled.GetType()
                .GetProperty("ParentCollection").GetValue(pregled, null);
            MethodInfo mi = referenceToPregledCollection.GetType().GetMethod("Add");

            mi.Invoke(referenceToPregledCollection, new object[] {pregled.GetType()
                                    .GetMethod("GetConvertedObject", BindingFlags.Public | BindingFlags.Static)
                                    .Invoke(null, new object[] {pregled})
                                });
            _context.SaveChanges();
        }
        catch (Exception ex) { MedisTestAssert.Fail(ex); }
    }
}
```

Fig. 2 Example of a generic test vector execution method

The next improvement we needed was the integration with testing tool. Since our MIS is developed under .NET environment, we rely on the standard test suites. Testing tool is a .NET application, designed as a part of our modeling tool that is able to generate test models and test classes and run tests. Tests are immediately executed and results can be prompted or stored and later evaluated. Testing phase itself is even more important when the legacy code has to be included into a new project. After including old functionality, we must ensure that both sides work as expected. Running initial set of tests will speed up component validation and detection of mapping related issues.

Testing tool is designed to run unit, integration and regression tests. Actually, it will execute the code from any loaded library containing test classes. The test classes will load the list of test vectors and run them in the sequence. The example of generic test method for adding new type of medical examination is displayed in Fig. 2. The testing class is instantiated with a reference to the list of test-vectors (variable *TestVektori*) and to the parent entities referencing test entities for patient and medical service. Variable *_context* references execution context of testing environment including all possible destinations where the data can be stored. For example, when *_context.SaveChanges()* is invoked, depending on configuration, data can end up either in database and/or in XML repository and/or as test result. In order to properly test generated or imported components, mentioned list of test vectors must contain representative values (Fig. 3). Values can differ significantly depending on data type and the scope. Also, data fields in the model can have various standard values which also have to be considered. Test vector generation process is based on the adaptive random [Shahbazi 2016] [Chen 2004] and model based testing [Jacky 2007] [ElFar 2001], but due to its complexity it will not be presented in details in the scope of this paper.

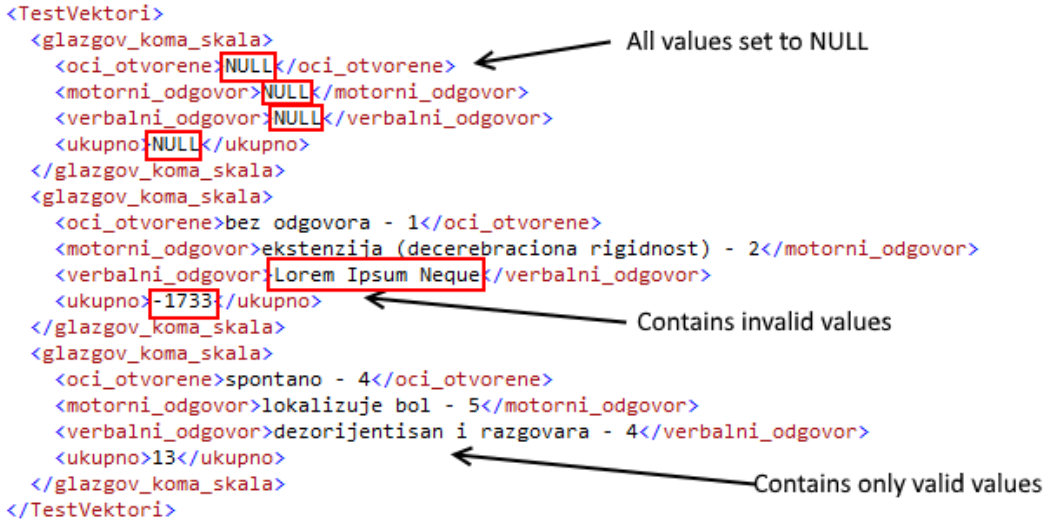

Fig. 3 Example of generated test vectors

4. CASE A: UPGRADING LEGACY COMPONENT WITH ADDITIONAL FUNCTIONALITY

One of the common requests we faced was upgrading the legacy components. These requests were usually not only about including legacy components into our project, but to extend them by supporting some of the features common for our MIS. One of these common functionalities is action-level configuration. In many cases, old MIS had not any role-based access control. Since new MIS developed by our research group (and called Medis.NET) supports possibility for defining accessibility for each action per single user, the feature was often incorporated into the update.

In this particular case, the users of our targeting institution already had an experience with a piece of software supporting pediatricians. The software allowed them not only options related to medical documents, but also the whole set of administrative actions. The mentioned set consists of actions such are defining new medical records for any department, updating demographic data or even change the existing insurance setup. After standardized MIS systems are introduced in Serbian primary care, this kind of actions were not allowed any longer for medical personnel. Since our client insisted to keep existing legacy functionality running, we needed to integrate it into the environment of our MIS. This kind of upgrade cannot be understood as code-generation friendly and it is easy to

run into the situation that more manual work is needed than in the case when a new form is built. The positive side of this specific case is that target form is developed in the same technology as the base application and can be directly imported to a project.

To achieve this kind of upgrade, we had to go with following steps:

- Define meta model of configuration parameters
- Include target form in the project and map actions with configuration parameters
- Add additional extension points in target form
- Run generator tool to generate configuration class and to update target form

Defining meta model of configuration parameters can be considered as the standard part of the process expected by our model based framework. The class containing configuration parameters would later be used to generate a configuration class whose instances would be loaded later by the MIS configuration tool (

Fig. 4). Mapping target form with the additional tags and extension points that would be used by code generation tool is the step which requires the most of manual work and time. In this case the initial design of the form helped a lot.

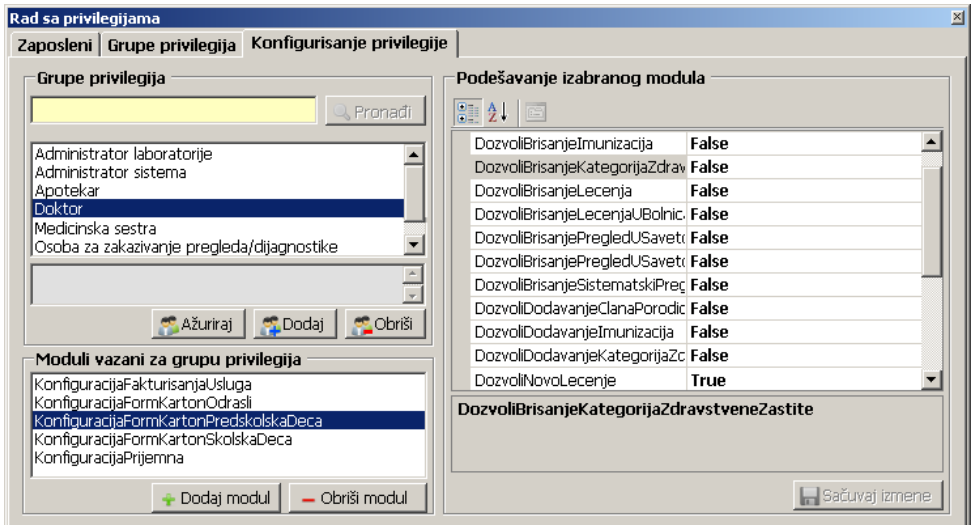

Fig. 4 The list of generated privileges within configuration tool

Since every add/edit/delete action was triggered by the button, the direct mapping between Boolean configuration parameter and properties making action button Visible/Enabled can be established. Another part needed for the form that need to be included is the new constructor allowing the form to fit in the new project. Benefits in this case are that end users do not request the migration of only one form, but rather series of similar ones. Then, the adaptation of each of the successive forms will use, as the basis, the same set of templates and analytics tools defined for the first in the row. In our presented case we had total five forms to include. In 4 out of 5 cases we used defined templates without modifications.

5. CASE B: INCLUDING THE SET OF LEGACY FUNCTIONALITIES INTO TARGETING MIS

Including the set of legacy functionalities is another request we had multiple times to deal with. The main problem with this kind of requests is that the set of action during the integration with legacy system cannot be precisely defined in advance. The systems can be developed differently and ability to reuse the existing functionality can be limited.

In this case, we had a request to bring the functionality of existing MIS dedicated for neurologists to our Medis.NET. The existing solution was developed in the manner of document management

systems with clear distinction between layers. For example, all forms were just used for displaying data, business logic for processing and data layer for interaction with database. Since we had to migrate data and to integrate into Medis.NET by keeping complete business logic unchanged, we had to find the best possible way to use our model based data generator and reduce the time needed for development.

The following set of actions were required here:

- Use reverse engineering tool to extract all the entities related specific medical examinations
- Establish connections to entities from Medis.NET and generate new segments of data access layer
- Adapt business logic to match the new entities
- Adapt presentation layer to match the new entities

The alternative for adaptation of business logic and presentation layer would be generation of new components and referencing necessary parts from legacy code. But in some cases, end users would not be willing to accept such solution.

The result of mentioned set of actions is a new project (called Medis.Neuro) referencing legacy libraries. It was built as separate component. This component can be then loaded in Medis.NET using standard approach such as inversion of control and supporting Spring library.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- modules declarations -->
<objects xmlns="http://www.springframework.net">
    <object type="Spring.Objects.Factory.Attributes.RequiredAttributeObjectPostProcessor, Spring.Core"/>
        <!-- Komunikacioni servis -->
    <object type="Medis.Communication.Service, Medis.Communication" singleton="true" id="MedisCommunicationService"/>
        <!-- Osnovni modul-->
    <object type="Medis.Core.Runtime.CoreRuntimeMsgListener, Medis.Core.Runtime" singleton="true" id="MedisRuntimeModule"/>
        <!-- Administracija -->
    <object type="Medis.Common.Kadrovska, Medis.Common" singleton="true" id="Kadrovska"/>
    <object type="Medis.Common.IzborLekara, Medis.Common" singleton="true" id="IzborLekara"/>
    <object type="Medis.Common.ZdravstvenaKnjizica, Medis.Common" singleton="true" id="ZdravstvenaKnjizica"/>
    <object type="Medis.Common.RadnoVreme, Medis.Common" singleton="true" id="RadnoVreme"/>
        <!-- Ambulanta-->
    <object type="Medis.Primar.Ambulanta, Medis.Primar" singleton="true" id="AmbulantaPrimar"/>
    <object type="Medis.Primar.Specijalisticki, Medis.Primar" singleton="true" id="AmbulantaSpecijalisticki"/>
    <object type="Medis.Dental.Modul, Medis.Dental" singleton="true" id="Dental"/>
        <!-- Stacionar-->
    <object type="Medis.Common.Stacionar, Medis.Common" singleton="true" id="Stacionar"/>
    <object type="Medis.Bolnica.IstorijaBolesti, Medis.Bolnica" singleton="true" id="IstorijaBolesti"/>
    <object type="Medis.Neuro.Pregledi, Medis.Neuro" singleton="true" id="Neuro"/>
</objects>
```

Fig. 5 Configuring Medis.NET with newly generated module

One possible configuration of running instance of Medis.NET is shown on Fig. 5. Beside the fact that generated components should be tested and have lower number of potential bugs, all kind of testing has even higher significance than with regular development. Just for the illustration, basic instance of Medis.NET running for primary care institutions has around 160 different data collection forms supporting different medical examinations. The set of functionalities included from old solution for neurologist consists of 146 different forms and significant business logic supporting many different decision making routines and calculations. So, after such an extension get introduced into information system, it is not simple unit and integration testing that must be done, but also performance check on the amount of resources used.

6. RESULTS AND DISCUSSION

Our model based modeling and data generation tool proved to be useful when new parts of the software have been developed. We managed to significantly reduce used time especially until first prototype is done. In some cases we managed to reduce expected time to one third [Rajkovic et al, 2015]. Also, due to the generation of tested components we experienced lower number of bugs.

The story of using model based generation tools with upgrade projects is a bit different. Since significant adaptation is usually required, we could not achieve results as with newly developed. The realization of both presented cases was done with the help of MDD approach. Due the high level of customizability, running the process for the first component in the row will last even longer than standard development. We had an opportunity to test both cases of upgrade requests on several different types of components so we are able to present some relevant results.

The first of the effects that we get from our framework is reducing the time needed for component development (Table I). We are displaying the time needed for each of the steps in component development workflow. The same steps were applied both in cases when no optimization is used, and when we use MDD approach. The data that we present is gathered as a result of surveying our development team members. We cannot make them general, but they are indicative enough to compare the results with different approaches. As the basic measurement unit we define T, which is a time that needed to define the structure of one entity and to create corresponding database table.

Table I Comparison of time spent on developing single windows/web form using different approaches in development

Process	**Standard development**	**MDD – first in the row**		**MDD – with developed template components**		
Step	**new development**	**upgrade project case A**	**upgrade project case B**	**new development**	**upgrade project case A**	**upgrade project case B**
DB table definition	T	0	0	0	0	0
Defining class in object model	T	T	2T	0.05T	T	0.15T
Developing/Adapting visual form	6T	10T	12T	0.25T	1.35T	0.75T
Form specific logic	2T	0	0	2T	0	0
Defining tests	T	T	2T	0.2T	0.25T	0.2T
Testing	T	2T	4T	0.5T	T	T
Overall time	12T	14T	20T	3T	3.6T	2.1T

Pursuing new development of data forms without help of MDD takes a lot time to complete, as we estimated this to 12T. Using MDD, we estimated this the time reduction to 20-30% (we set it to 3T). The most significant time spent in this case is implementation of form specific logic which cannot be derived from other places. Theoretically, this can vary a lot, but we estimated it to 2T.

Working on upgrade projects and component integration is significantly time consuming with help of MDD we estimated necessary time to 14T for case A and 20T for case B. The most time is consumed while writing the adaptation classes and methods. Defining classes for object model takes more time than with regular development because developer cannot use results of reverse engineering tool directly, but needs to adapt retrieved list of fields introducing new key constraints and defining extension template. Extension template is used for any consecutive component to replace its key components. This is applicable for case B. For the case A it is not since we still need to define separate mapping for each of the forms.

We choose these two cases to be, by our opinion, representative. Time needed for adaption of any of succeeding forms could vary from case to case, but for these implementing the same interfaces it is much easier to reuse any of developed mappings. If we compare time needed to develop series of components, case B proved even better results than new builds after 10^{th} in the row (39T for new build, 38.9T for adapted). Case A always requires more time than new build since not only mapping on data is needed, but also unpredictable custom mapping on actions. Case B initially required the most of the time, but after 5^{th} adapted form in the row case B started to be more effective from the point of time consumption.

7. CONCLUSION

Including parts of legacy code in one's software project is scarce on every programmer's wish list. Dealing with code developed under different standards, architecture and even different technology is not an easy task to be accomplished. In this paper we wanted to show the overall positive effects we got from using our model based code generation framework in upgrade projects.

We examined two cases – when existing functionality from legacy system has to be extended (case A) and when significant portion of legacy system need to be included in our system (case B). In both cases, significant time is needed when first component is developed. During that time, all necessary custom templates are built allowing effective use of code generation tool for next instances.

Even with this approach, MDD significantly saved time in cases when many components with common functionality had to be generated. In cases when target components differ significantly, as well as have different usage, model driven cannot be of much help. The MDD approach has its value even for this kind of projects, and it can be used in more effective way with series of similar components.

REFERENCES

Ali Shahbazi and James Miller. 2016. Black-Box String Test Case Generation through a Multi-Objective Optimization. *IEEE Transactions on Software Engineering 42.4 (2016): 361-378.*

Andrew Gettinger et al. 2012. Transitioning from a Legacy EHR to a Commercial, Vendor-supplied, EHR. *Applied clinical informatics 3.4 (2012): 367-376.*

Dae-Kyoo Kim, Lunjin Lu, and Byunghun Lee. 2017. Design pattern-based model transformation supported by QVT. *Journal of Systems and Software 125 (2017): 289-308.*

Dipak Kalra and B. G. M. E. Blobel. 2007. Semantic interoperability of EHR systems. *Studies in health technology and informatics 127 (2007): 231.*

Dipak Kalra, Thomas Beale, and Sam Heard. 2005. The openEHR foundation. *Studies in health technology and informatics 115 (2005): 153-173.*

Fabio Palomba et al. 2016, On the diffusion of test smells in automatically generated test code: An empirical study. *Proceedings of the 9th International Workshop on Search-Based Software Testing. ACM, 2016. p. 5-14.*

Georg Duftschmid, Judith Chaloupka, and Christoph Rinner. 2013. Towards plug-and-play integration of archetypes into legacy electronic health record systems: the ArchiMed experience. *BMC medical informatics and decision making 13.1 (2013): 11.*

Georg Duftschmid, Thomas Wrba, and Christoph Rinner. 2010. Extraction of standardized archetyped data from Electronic Health Record systems based on the Entity-Attribute-Value Model. *International journal of medical informatics 79.8 (2010): 585-597.*

Georg Hinkel. 2016. NMF: A Modeling Framework for the. NET Platform. *Forschungszentrum Informatik, Karlsruhe, Germany, https://sdqweb.ipd.kit.edu/publications/pdfs/hinkel2016d.pdf*

Health Level 7. 2013. Normative Edition HL7 V3 2013 Care Provision Domain, *Ann Arbor HL7 International, 2013*

Ibrahim El- Far and James Whittaker. 2001. *Model- Based Software Testing. Encyclopedia of Software Engineering (2001).*

Jonathan Jacky et al. 2007. Model-Based Software Testing and Analysis with C#. *Cambridge University Press, 2007.*

Karl Wiegers and Joy Beatty 2013. Software requirements. *Pearson Education, 2013.*

Natalie Pageler et al. 2016. A rational approach to legacy data validation when transitioning between electronic health record systems. *Journal of the American Medical Informatics Association 23.5 (2016): 991-994.*

Petar Rajkovic, Ivan Petkovic, Aleksandar Milenkovic and Dragan Jankovic. 2016. Combining Agile and Traditional Methodologies in Medical Information Systems Development Process. *SQAMIA 2016. pp 65-72*

Petar Rajkovic, Ivan Petkovic, Dragan Jankovic. 2015. Benefits of Using Domain Model Code Generation Framework in Medical Information Systems. *SQAMIA 2015. pp. 45-52.*

Petar Rajkovic, et al. 2010. Software Tools for Rapid Development and Customization of Medical Information Systems. *Healthcom 2010, pp. 119-126.*

Petar Rajković, Dragan Janković, Aleksandar Milenković 2014, Improved Code Generation Tool for Faster Information System Development, *SAUM 2014, pp. 273 -276*

Peter Schloeffel et al. 2006. The relationship between CEN 13606, HL7, and openEHR. *HIC 2006 and HINZ 2006: Procs.: 24.*

Sean West. 2013. Need versus cost: understanding EHR data migration options. *The Journal of medical practice management: MPM 29.3 (2013): 181.*

Shelly Sachdeva and Subhash Bhalla. 2012. Semantic interoperability in standardized electronic health record databases. *Journal of Data and Information Quality (JDIQ) 3.1 (2012): 1.*

Tsong Yueh Chen, Hing Leung, and I. K. Mak. 2004. Adaptive random testing. *Advances in Computer Science-ASIAN 2004. Higher-Level Decision Making. Springer Berlin Heidelberg, 2004. 320-329.*

Wajahat Ali Khan et al. 2014. An adaptive semantic based mediation system for data interoperability among health information systems. *Journal of medical systems 38.8 (2014): 28.*