# Benefits of Using Domain Model Code Generation Framework in Medical Information Systems

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

Both in medical information system development and upgrade project, we often face with a challenge of creating large number of reports and data collection forms. In order to reduce our efforts in this segment of system development, we tried to use various code generation and reporting tools. The main problem with standardized tools were lack of flexibility. Thus, we decided to develop domain model based framework that consists of data modeling, inverse engineering, code generation and model interpretation libraries and tools. Data modeling tool is used to create domain specific model starting from the loaded meta model. Both code generation and runtime interpretation tools use domain specific model as a basic input, and together with visual templates and generation/interpretation classes form easily extendable and customizable system. In our medical information systems development and upgrade projects we use both approaches and tend to define their proper roles in overall information system life cycle – from requirement collection phase to later system upgrades. In this paper we present basic building blocks of our framework and compare the effects of its usage against development when no automatic generation component is applied as well as when only standardized code generation tools are used. We managed to reduce development time in some segments of the system using domain model based generation tools to about one third of usually needed. The presented framework and its components are developed and tested during last six years and tested in four different development projects, around ten upgrades and in more than 25 information system deployment projects.

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, Model interpretation

## 1. INTRODUCTION AND MOTIVATION

All major parts of information system's life cycle depend on a knowledge that come from many areas of computer sciences. At the same time, it requires the application of domain specific knowledge that should be incorporated into the system, in order to make developed software useful to its end users. Considering all levels of complexity, time needed for a system development is, pretty often, much longer than it is really necessary. Also, this period can be unacceptable for potential end users. The causes for this situation are very different, but in this paper we want to point one – time spend on a development of different components that share the same set of basic functionalities, but display different data – primarily data collection forms and reports.

In order to help developers solving this problem, software development environments offers different types of wizard-like tools that can load data model and then produce the form containing all required data entry fields and labels. Even though, these tools can support some complex views (master-detail or MVC), but when needed to be incorporated into the information system, developer must spend significant time to adapt their automatically generated logic before fitting the project. Furthermore, if generated forms need to support some translation mechanism, usually some time must be spent on this too. The similar story

This work is supported by the Ministry of Education and Science of Republic of Serbia (Project number #III47003).
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 534, 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, M. Heričko (eds.): Proceedings of the 4th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA 2015), Maribor, Slovenia, 8.-10.6.2015. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

can be told for reports. Even many reporting tools and engines can easily generate reports, their customization and later inclusion into the running software project can require significant amount of time.

The problem is not limited to the development phase, but also to later phases of upgrade and information system maintenance. Table 1 shows the statistic we extract from the project of medical information system (MIS) development for Primary and Ambulatory Care Center situated in Nis [Rajkovic et al 2009]. Nis is the biggest city in southern part of Serbia with population of about 250,000.Since it is a regional center, more than one and a quarter of a million of people from southern and eastern Serbia gravitates to our target institution for more complex medical examinations. The initial project has been running from 2009 till 2013, and during this period system was built incrementally. First departments started to use the system in second part of 2010 and system entered full service in all departments till the end of 2012. In this period, overall daily load of the system rose from about 3000 registered medical services daily to more than 13500. During 2013 and 2014 system was upgraded upon separate change requests. After the developed MIS was successfully deployed in Nis, the system was deployed in primary care centers of another 25 towns in southern and eastern Serbia. Furthermore, the extended and modified version was also used for Neurology Clinic in Nis supporting different set of initial system requirements [Milenkovic et al 2010].

Increased number of registered medical services was followed with the increase in number of requested reports and data collection forms. As it has been stated before, the primary care center has between 10 and 15 thousands of medical services (including examinations, therapeutic threatens and laboratory analysis) daily. Total number of different offered services is around 300, while around 250 are requested by government authorities. Many of them use standard data collection forms, but for significant number of services separate forms had to be developed. Next, medical personnel has to generate various reports and send them to Ministry of Health (MoH), insurance funds and other government agencies. Moreover, there are internal reports needed for covering internal business process as well as different medical research reports.

Table 1. Increase of the number of required reports and data collection forms per year

| Calendar year | Medical examinations (daily) | External reports | Internal reports | Data collection forms | Required medical services |
|---|---|---|---|---|---|
| **2011** | 2990 | 17 | 7 | 14 | 170 |
| **2012** | 7322 | 27 | 40 | 63 | 194 |
| **2013** | 13599 | 35 | 73 | 74 | 231 |
| **2014** | 13696 | 39 | 139 | 91 | 245 |

At the moment, our MIS dedicated for primary care facilities supports around 100 different data collection forms and almost 200 reports. Having in mind previous experience in information system development, and facing the constantly rising number of requested data collection forms and reports we tried two different approaches for making GUI development process more effective – use standard generation components that come with development environment and use domain model based custom built code generation and model interpretation tools. We are going towards a definition of a domain model-based framework that will help us when needed to automatically generate, test and deploy series of similar visual components. The framework, which was developed for initial project for Nis Primary Care Center, was intensively used to support specific requests for deployments in other medical facilities. The framework and its components were used in 4 development project (primary care MIS, hospital MIS, laboratory IS and radiology IS), around 10 upgrade and in more than 25 deployment projects.

In this paper we will presents the basic design elements of our framework and show its effect on the information system development process. Concepts of specific domain model creation, domain model extension and update are presented as basic building blocks of the framework. Furthermore automatic code generation and model interpretation are presented in following chapter. At the end, the comparison of results is presented and discussed.

## 2. RELATED WORK

The main elements of our model driven development framework are tool for domain specific data modeling, database structure check and inverse engineering library, code generation module and code interpretation library. Literature related to model driven engineering is pretty voluminous, but we can point out the paper [Meixner 2011] that elaborates and compares different approaches in model based GUI development along with past and current trends. Looking at the definitions, our approach could be categorized as holistic model driven GUI development process.

The base component of our framework is a modeling tool. It can be used for creation new and validation of existing models. Validation is based on design space exploration (DSE) principle and concepts based on dependency analysis described in [Hegedus et al 2011]. Our modeling tool is extended with reverse engineering library used to extract entities from legacy systems. In [Ramon 2014] the authors were focused on finding implicit layout and then generate appropriate GUI model. We follow this philosophy, but applied it on data definition level and the used generated model to generate not only GUI but various other components. We extended the inverse engineering tool with the component able to generate database administration application used for better understanding the structure of a legacy database. Our approach is based on environment named Teallach [Barclay et al 2003], which interface generation and environment customization concepts are partly reused. When working with legacy databases one problem that we often faced were vendor-specific functions and structures for databases. To overcome this problem we used the approach described in [El Akkaoui 2011]. This paper describes model driven framework that supports ETL (extract-transform-load) processes for data warehousing projects. The major part that we acquired were approach for creating vendor-independent sub-models and transformation based on a set on common interfaces.

Our code generation approach is based mostly on [Badreddin et al 2014], while widget based code interpretation was influenced by [Wanderman-Milne et al 2014] and [Ren et al 2010].Comparing code generation and interpretation approach, the biggest drawback of interpreted component is lack of standardized test cases [Schlegel 2010], but the comparative advantage is their flexibility. Thus, we believe that both approaches have the right place in system development and can be used for different purposes.

## 3. DATA MODELING FRAMEWORK

Our data modeling framework is developed around initial meta model appropriate for MIS development. As a starting point we used OpenEHR meta model. Next, we adapted it and define extension points – entities that can be used as heading elements for newly defined items. Next step was to develop modeling tool that could be easily used both by developers and potential end users. The importance of the user friendly modeling tool is not only technical, but also project management related. This can help involving future users from the beginning of development process which eventually will lead to better results during system acceptance process.

The starting point in our project was to develop the extensions of domain specific model in a modeling tool. Modeling tool was used to define model extensions that will corresponds to future entities that will represent separate medical examinations or reports. Initially, we identified entities in starting model that can be further specialized (and call them extension points) and then use the modeling tool to define derived entities. The extension points can be configured including base entity name and specifying extension rules. These model extensions will be used as an input for generation tool that will be described in next section.

Another dimension to the modeling process is brought by legacy software and legacy data. In many cases, clients already have large amounts of data collected over the years which want to integrate into a new system. To complete the set of needed developing components, reverse engineering tool is a next that is needed (Fig. 1). Its main aim is to analyze existing data structures and automatically expand the model. Reverse engineering tool will initially load legacy data structure and examine it against existing meta model and extension point definition. Furthermore, reverse engineering tool can generate database administration application that helps in visualizing data structures in legacy database and helps in understanding relations between data entities.

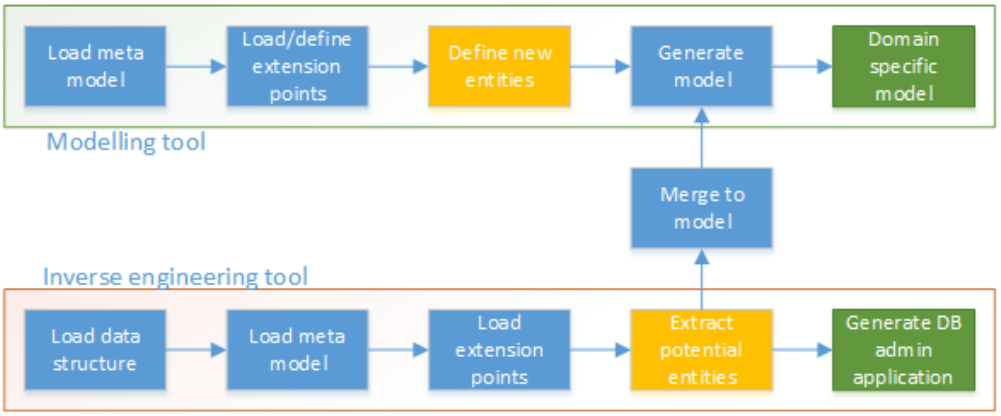


Fig. 1. Main functional block of data modeling and reverse engineering tool

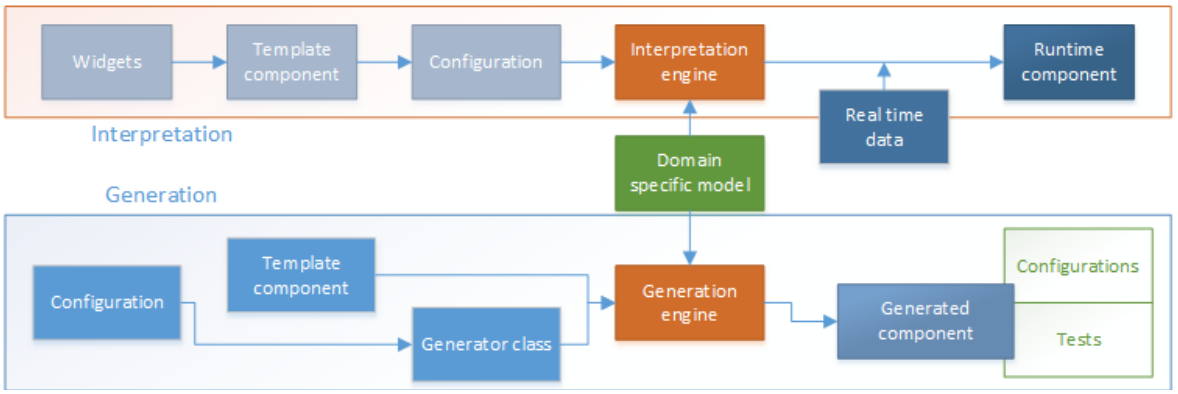


Fig. 2. Comparison of interpretation and generation processes based on domain specific model

When identify data tables that can satisfy extension point conditions, it will generate new entities that can be integrated in domain specific model. Before the extracted entities can be generated for domain specific model, they will be compared to the entities that already exist in order to find potential conflicts. Potential conflicts are examined on two cases, named Case A and Case B. Case A is a situation when two entities of the same name are found (one in the existing model, and another in the set of generated entities) – then the structure will be further examined. Case B is a situation when two entities of the same structure, but different names are found.

In case A, the tool will first check if the entities are defined under the same extension point. If so, user will be prompted with merging tool where can decide if should discard new version of the entity (extracted from legacy database), overwrite the existing entity definition with new version or pick which properties from the original and which from the final model should be included in a merge result. In case B, user must change the name of one of the entities to maintain naming convention in end model. At the moment there is a constraint that all entities must have different names.

Automatic conflict resolution can be enabled, and then it can be defined per extension point. Available strategies for merging in case A are:

- Keep existing entity. In this scenario changes from loaded entities will be ignored.
- Overwrite with loaded entity. Loaded entity will overwrite existing one.
- Merge properties. Properties that appear only in loaded entity will be added to existing entity.

In case B, only one automatic merge scenario is supported – change the name of loaded entity by adding the prefix that corresponds to extension point name.

Generated model is used then as a pivot element both for code generation and component interpretation engines (Fig. 2). Both of them use domain specific model as the main input and combine its structure with already prepared configurations and templates in order to produce final component.

## 4. COMPONENT GENERATION AND RUNTIME MODEL INTERPRETATION

Domain model, extension points and template components are used as the input for the next important step – automated software components generation. For this purpose specific highly customized generation tool has been developed. Generation tool loads domain model, template component, generator class and by using them creates a new software component that can be included in a software project or compiled and immediately used as a library [Rajkovic et al 2014]. Beside it is based on general approach, the system is optimized for Microsoft .NET platform.

Along with domain specific model entity, another base input for generation tool are template components. Our approach is to take already developed and tested software component, such as windows form, and then identify the parts of the code that can be used as general. Then, replace form specific parts with the specially marked comments that will be replaced during component generation process with a code generated on the base of the entity taken from the model. Since its generation process is based on templates, it has been used within our projects for automatic creation of several different classes of components – Windows forms, value selection components, translation resources and access privilege lists. Another benefit of this approach is improvement in component testing. For automatically generated component, automatic tests can be defined. For each of the fields and actions, set of predefined tests can be included. This will be the addition to the initial set of the tests loaded with chosen template component. After generation and testing, newly created components can be used for the extension of existing applications. On the base of mentioned model extension, and previously developed template software components (such are forms and reports), our generation tool will generate GUI elements that can be incorporated into information system project.

As it has been stated before, one big advantage in this approach is if users are involved from the initial stages of the project they can much easier accept the developed software later. This is not a new conclusion, even in a case study [Linberg 1999] the lack of communication with future user can lead to a project fall. Further analysis can be found in [Agarwal 2006] and [MacLeod et all 2011]. Also, we published our results and observation on this topic in [Rajkovic et al 2013].

Model based component generation is not the only way of domain model usage used within our framework during information system development process. The next approach is runtime model interpretation. For this purpose, special set of Web components is developed (Fig. 2).

The model interpretation library uses basically the same approach as the component generator – it loads entity from the model and the template in order to create a component that will be displayed in the browser. The main difference between this and the approach with code generation is that users can have two additional tools that can use online – configurator and the template definition tool. In the template definition tool users can define the visual elements of the displayed components – its arrangement, positioning, colors fonts etc. Configuration tool lets user further specify the appearance and the content of particular elements of the chosen template.

## 5. RESULTS AND DISCUSSION

Both presented approaches can be effectively used during information system development process. We had an opportunity to test them on many different types of components so we are able to present some relevant results and to define guidelines when to use which approach. Immediate effects that we get from our framework is reducing the time needed for component development (Table 2). In the table below, we presented the time needed for specific steps in component development process in cases when no optimization is used, then when we used only standard components and at the end when we used model driven approach through our framework. The data that we presented are gathered as a result of surveying our development team members, so they cannot be marked as fully accurate and objective, but they are indicative enough to compare the results in different approaches. We have interviewed twelve

developers currently involved in development projects that actively use our framework. Interviewed colleagues have at least two years long experience in information system development, while some of them work more than a decade in this field.

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. All the other measurements are in correlation with mentioned T. After the creation of the database table, next step is defining a class in the object model. When using no optimization in development process, these two actions took the approximately the same time. But, in case when some object-relational model (ORM) is enabled in the project it is enough to create only one entity definition and it will be automatically used for creation both of the table and the entity.

Table 2. Comparison of time spent on developing single windows/web form using different approaches in development

| **Step** | **No optimization process applied** | **Using standard reports and component generators** | **Model driven dev.** |
|---|---|---|---|
| Database table definition | T | 0 | 0 |
| Defining class in object model | T | T | T |
| Developing visual form | 3T | 0.1T | 0.05T |
| Data validation methods implementation | 3T | T | 0.1T |
| Implementation of a logic specific for a form | 2T | 2T | 2T |
| Defining configuration parameters | 1T | 1T | 0.1T |
| Testing | 6T | 3T | 0.5T |
| Overall time | 15T | 8.1T | 3.75T |

Next step is developing of visual form. It is related to pure creation of visual elements of the form and their connection to values retrieved by ORM classes. Since a developer needs to define label translations, data displaying components and to develop/inherit the logic to connect the form with the rest of the system we assumed that needed time is around 3T. When using some generation tools, we would get instantly created form and some simple adjustment is needed then. In case when we use our generation/interpretation tools this time is even shorter since no changes in visual style are usually required.

The component generation step is followed by implementation of data validation methods and the implementation of form-specific logic. While form-specific logic can hardly be replaced with automatically generated code (since it is a consequence of specific stakeholder requests) implementation of data validation method can be significantly tuned up. Standard component generators usually have check on the data type level, while our generation/interpretation tool can include also range checks that can be directly taken from used domain specific model.

Predominantly, our framework was used to create components for medical information systems, and many requirements specific for this area are included. We often have special requests to support data-field level configuration. In many cases the end users request the possibility to define which actions are possible for each field in some forms. Those requests lead to definition of specific configuration parameters. Depending on the number elements of the form, the process of defining and integrating configuration is process that lasts at least as the initial form definition. When our framework is used, we are able to define special generator class and template component so we could automatically generate configuration parameters at the same time while generating the form and therefore significantly reduce required time.

At the end of development process, testing and bug fixing lasted much longer when no generation component is used. Manual form building is process that many developers consider as less interesting and many different bugs came out when testing started. The most common problems are fields that are not connected to ORM properties and missing data validity checks. When using automated test generators, the time needed for this segment is halved. Using our framework, we managed to reduce this time significantly due to the fact that automatically generated and tested components are usually prone to mentioned errors. The segment of time that cannot be reduced is the one related to testing and fixing bugs from form-specific logic.

Table 3. Comparison the effects of different approaches in GUI components development on overall system development process

| Category | Standard reports and component generators | Generated components | Interpreted components |
|---|---|---|---|
| Number of user sessions per module | 4.33 | 2.76 | 2.08 |
| Iterations before accepted solution | 6.66 | 2.75 | 2.5 |
| Mockup generation time | 4 days | <1 day | <1 day |
| Average bug reports per module | 9.65 | 3.74 | 3.35 |

The improvements related to visual components generation are not the only benefit we got. The side effect was improvement of overall development process (Table 3). Before start using our framework we relied our development process on standard report and component generation tools. Using modeling tool both by us and our customers reduced number of requirement collection session with users. Without modeling tool we have 4-5 sessions before final agreement per module. With modeling tool we reduced that number to around 2. Using interpreted components with some default templates made us possible that we can show initial form overview even during the first session. Also, using functionality from our reverse engineering tool, we are able to deliver application mockup for less than a day. Mockup application usually have uniform visual style and it is based on a single display template, but its main aim is to verify defined data structures and help arrange them logically. So, initial forms will have default view, but will be able to display all needed data with corresponding types and ranges. After initial session, we are focused on developing GUI templates in order to have proper preview of future functionalities. During second session with potential users we are usually able to demonstrate them GUI components mockup and to finalize stakeholder document in the sections related to user interface and required data structure.

Considering this, our users initially know what to expect, so when the project come to system acceptance and testing the number of change requests and bug reports is significantly lower if our framework is used. Comparing our two approaches – generated vs interpreted components – we can state that users have better response to Web solution based on interpreted components than to Windows interface based on generated forms. The main reason is the fact that the users do need to install any additional piece of software (they need only Web browser). Also, with installed template designer, the advanced users can develop their own templates and extend the existing system. On the other hand, the advantage of Windows forms based solution is faster response than Web applications.

After few years of system exploitation we realized that both approaches have their place in overall medical information system lifecycle. Windows applications are used by doctors and nurses in ordination and next to the medical instrumentation – in places when system needs to collect and process data. On the other hand, Web based components look like more desirable when data access, formatting and presenting is needed – in Web application offering medical record overview and in report generation.

## 6. CONCLUSION

This paper presents domain driven approach in automatic code generation process. To support this approach we have developed specific framework. The framework is actively used within our development team. We interviewed team members in order to get an estimate of the process improvement using the framework. Following their responses we can assume that the best effect we have in requirement collection phase where overall time needed is reduced to roughly 30-40%. When the project reaches development phase overall estimate based on mentioned survey is that the total time needed could be reduced to 25-30% of the time needed initially. The significant effect is visible on GUI based components, but core system components still need to be programed. When the project comes to deployment phase, the number of reported bugs is significantly lower than with the approach when no component generation tools are used. The main reason for this is that developers have initially generated forms with trusted functionalities, so they can focus primarily on the form's specific logic.

Our automatically generated/interpreted components helped not only development but also whole information system lifecycle, from initial mockup solution generation, through the development to later system upgrades and finally to deployment and maintenance. In all of these steps automatically

generated components reduced needed time. In combination with usage of modeling tools that improve communication with the customers and keep them involved during system development, they make eventual system acceptance easier and reduces number of customer reviews before final goal is reached.

## REFERENCES


Petar Rajković, Dragan Janković, and Aleksandar Milenković. 2013 “Developing and deploying medical information systems for Serbian public healthcare: Challenges, lessons learned and guidelines." Computer Science and Information Systems 10.3 (2013): 1429-1454.

Petar Rajković, Dragan Janković, Aleksandar Milenković 2014, Improved Code Generation Tool for Faster Information System Development, SAUM 2014, Nis, Serbia, 12 - 14 November 2014, pp. 273 -276, Conference Proceedings, ISBN: 978-86-6125-117-7.

Aleksandar Milenković; Petar Rajković; Dragan Janković; Tatjana Stanković; Miroslava Živković 2010: Software Module for Clinics of Neurology as a Part of Medical Information System Medis.NET, ICEST 2010, Ohrid, Makedonija, 23 - 26 Jun 2010; Proceedings, Vol. 1, br. 0, str. 323-326; ISSN: 978-9989-786-57-0;

Petar Rajković; Dragan Janković; Tatjana Stanković 2009: An e-Health Solution for Ambulatory Facilities 2009, 9th International Conference on Information Technology and Applications in Biomedicine, ITAB 2009, Larnaca, Cyprus, November, 2009; Proceedings, Vol. 1, br. 1, str. Fr. 1.5.1 1-4; IEEE Publishing 2009, ISSN: 978-1-4244-5379-5

Badreddin, Omar, Andrew Forward, and Timothy C. Lethbridge. 2014 "Improving Code Generation for Associations: Enforcing Multiplicity Constraints and Ensuring Referential Integrity." Soft-ware Engineering Research, Management and Applications. Springer International Publishing, 2014. 129-149.

Wanderman-Milne, Skye, and Nong Li. 2014 "Runtime Code Generation in Cloudera Impala." IEEE Data Eng. Bull. 37.1 (2014): 31-37.

Ramón, Óscar Sánchez, Jesús Sánchez Cuadrado, and Jesús García Molina. (2014) "Model-driven reverse engineering of legacy graphical user interfaces." Automated Software Engineering 21.2 (2014): 147-186.

Linberg, Kurt R. "Software developer perceptions about software project failure: a case study." Journal of Systems and Software 49.2 (1999): 177-192.

Agarwal, Nitin, and Urvashi Rathod. 2006 "Defining ‘success’ for software projects: An exploratory revelation." International journal of project management 24.4 (2006): 358-370.

McLeod, Laurie, and Stephen G. MacDonell. 2011 "Factors that affect software systems development project outcomes: A survey of research." ACM Computing Surveys (CSUR) 43.4 (2011): 24.

L. Ren, FengTian, X. Zhang and LinZhang, "DaisyViz 2010: A model-based user interface toolkit for interactive information visualization systems," Journal of Visual Languages and Computing 21, Elsevier, p. 209–229, 2010.

P. Barclay, T. Griffiths, J. McKirdy, J. Kennedy, R. Cooper, N. Paton and P. Gray, 2003 "Teallach — a flexible user-interface development environment for object database applications," Journal of Visual Languages & Computing, Elsevier, vol. 14, no. 1, p. 47–77, 2003.

Meixner, Gerrit, Fabio Paternò, and Jean Vanderdonckt. 2011, "Past, Present, and Future of Model-Based User Interface Development." i-com Zeitschrift für interaktive und kooperative Medien 10.3 (2011): 2-11.

T. Schlegel 2010, "An Interactive Process Meta Model for Runtime User Interface Generation and Adaptation," in Fifth International Workshop on Model Driven Development of Advanced User Interfaces, Atlanta, 2010.

Zineb El Akkaoui, Esteban Zimànyi, Jose-Norberto Mazón, and Juan Trujillo. 2011. A model-driven framework for ETL process development. In *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP* (DOLAP '11). ACM, New York, NY, USA, 45-52. DOI=10.1145/2064676.2064685 http://doi.acm.org/10.1145/2064676.2064685

Abel Hegedus, Akos Horvath, Istvan Rath, and Daniel Varro. 2011. A model-driven framework for guided design space exploration. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering* (ASE '11). IEEE Computer Society, Washington, DC, USA, 173-182. DOI=10.1109/ASE.2011.6100051 http://dx.doi.org/10.1109/ASE.2011.6100051