

A template-based code generator for web applications

Burak UYANIK^{1,2}, Veysel Harun ŞAHİN^{3,*}

¹Department of Computer and Information Engineering, Institute of Natural Sciences, Sakarya University, Sakarya, Turkey

²Turkish Management Sciences Institute (TÜSSİDE), TÜBİTAK, Kocaeli, Turkey

³Department of Software Engineering, Faculty of Computer and Information Sciences, Sakarya University, Sakarya, Turkey

Received: 08.10.2019

Accepted/Published Online: 25.12.2019

Final Version: 08.05.2020

Abstract: The importance and usage of web applications grow every day. Today from small businesses to large-scale corporations, many institutions prefer web applications for both their internal and external services. Code size and complexity of these kinds of applications grow rapidly. This brings up the question of how to improve the development process of web applications. A solution can be to use code generators. This paper introduces a template-based code generator to improve the development process of web applications. The code generator was developed and integrated into a real-life web application. Today, the web application together with the code generator is actively used in industry. This proves that an effective integration of a template-based code generator into a real-life large-scale web application can be achieved. In addition, the effectiveness of automatic code generation to manual implementation was shown with experimentation. Throughout the experiments, bug-free code generation was observed. Also, 98.95% improvement in average development time, 93.97% improvement in average test run count, and 49.37% improvement in average code size was achieved.

Key words: Software engineering, template-based code generation, web architecture, web application

1. Introduction

As computer technology becomes prevalent, our expectations from computer systems increase, especially in terms of functionality. Developed software grows both in terms of quantity and code size. Software development practices become more important. Researchers study new approaches for the design and development of software systems. There are several different goals of these efforts like easing the design and development of software projects, decreasing the number of bugs, providing development efficiency, and lowering costs.

One of the fast-growing areas of software development is web applications. Web architectures and thus web application architectures are still being studied actively. Multilayer and service-oriented architectures became the preferred choice of web application developers to ease the design, development, integration, and access. Detailed information about web services architecture can be found in the World Wide Web Consortium (W3C) working group's note.¹ A messaging standard between web services was defined in the W3C Recommendation on Simple Object Access Protocol (SOAP) Messaging Framework.² Fielding and Taylor introduced the Representational State Transfer (REST) architectural style in 2002 [1]. Today, REST is one of the most

*Correspondence: vsahin@sakarya.edu.tr

¹W3C (2004). Web Services Architecture [online]. Website <https://www.w3.org/TR/ws-arch> [Accessed: 04-Sep-2019].

²W3C (2004). SOAP Version 1.2 [online]. Website <https://www.w3.org/TR/soap12> [Accessed: 04-Sep-2019].

popular web architectures. Readers can get more information about web architectures from [2, 3]. Because the functionality and code size of web applications are growing, getting help from code generators alongside architectural structures in the development process also becomes widespread.

Before going any further we should mention that in this study we use the terms “code generator” and “code generation” in the context of automatic programming [4, 5]. These terms are also commonly used in compiler research [6], which is not the target of this paper. In our study, code generation, or automatic programming, refers to the process of generating software automatically by using inputs from users and the environment. The inputs can include parameters, models, schemas, templates, etc. The generated software can include several kinds of files (e.g., source code files, configuration files) as well as database objects and records.

There are several advantages of using code generators. One of them is increasing the productivity of developers [7]. Another one is improving the quality of code by means of reducing errors [8]. The benefits of using code generators in web application development are listed as follows:

1. In web development many similar codes (e.g., database operation codes) are written. Code generators can prevent the rewriting of replicated code repeatedly by developers.
2. In web development projects, generally two types of developers work together: front-end developers and back-end developers. Code generators can create both front-end and back-end codes. Thus, the project costs can be lowered by assigning fewer developers.
3. Code generators save developer time by reducing the need to write code.
4. Because the code writing activity is decreased, the number of bugs may also decrease.

In this study we develop a code generator that uses a template-based code generation approach. In this approach, a code generator takes inputs (required data and templates) and produces software by synthesizing those inputs [5].

As the name implies, a template-based code generator works with the help of templates. In our study we chose to use the templatization method [9] to create templates. The templatization method is also called the code migration method [10]. In this method, first a reference implementation of the target software is created. Then the required templates are developed by using that implementation. The code generator can then use the templates and other inputs if necessary and creates the target software as similar as possible to the reference implementation.

In this paper we introduce the above-mentioned template-based code generator for web applications. After the development, the code generator has been successfully integrated into the Turkish Management Sciences Institute (TÜSSİDE)³ Integrated Management System (TBYS), which is a large-scale real-life web application. TBYS is an enterprise resource planning (ERP) system. Today ERP systems are widely used in many organizations. Also, there is active research on all aspects of ERP systems [11, 12]. TBYS has been developed in a modular fashion to enable smooth integration into different kind of corporations. TBYS is an extensible application to easily add new functionality depending on the needs of users. As the reader might guess, the main helper of this extensibility property is the code generator.

The main contributions of this paper are as follows:

³TÜSSİDE (<http://tusside.tubitak.gov.tr/en>) is a research and development unit of the Scientific and Technological Research Council of Turkey (TÜBİTAK) (<https://www.tubitak.gov.tr/en>).

- A template-based code generator for web applications is introduced.
- The design, development, integration, and work flow process of the template-based code generator for a large-scale web application (specifically an ERP system) are described.
- Personal experiences regarding the development of a real-life system are shared. Also, the achievements gained from the project are given.
- Evaluation of the developed code generator is realized using experimentation [13]. The experimentation process and results are shared and discussed.

The rest of the paper is organized as follows. Section 2 explains the technologies and development process of the system. The code generator is presented in Section 3. In Section 4 we explain the experiments that were used to assess the code generator and discuss the results. After giving the related work in Section 5, the paper concludes with Section 6.

2. Development

In this section we give information about the development of the code generator. First, we summarize the main technologies used by the code generator. Then we describe the development process.

2.1. Technologies

Stored procedures [14] are routines that are stored in the system catalog of a database management system application. There are several advantages to using stored procedures like increased performance, decreased network traffic, and code reuse. Because of their advantages, they are usually preferred by web developers. Stored procedures are also used in TBYS. Therefore, the code generator makes heavy use of this technology.

In TBYS, create, update, read, and delete (CRUD) operations are handled by the help of automatically created stored procedures. Stored procedure templates were developed by using the Microsoft Text Template Transformation Toolkit (T4) templating language.⁴ The code generator uses the T4 template system to create stored procedures from templates.

Another template system used by the code generator is handlebars.⁵ The HTML (interface) templates for the user interfaces of TBYS are developed by using the handlebars templating language. The transformation of HTML templates to HTML files is performed with the help of the handlebars template system.

HTML files are mainly composed of user interface components like buttons, checkboxes, etc. To be able to create user interface components, the code generator uses plugin technology. There are two types of plugins: HTML plugins and JavaScript plugins. HTML plugins are used to determine the view properties. HTML plugins reference predefined cascading style sheet (CSS) files in the system. JavaScript plugins are used to determine the action properties of user interface components. Some of the user interface components and the file names of the related HTML and JavaScript plugin templates are shown in Table 1.

RESTful web services [1] are also used extensively in TBYS. Therefore, the code generator creates web service codes that use RESTful architecture. For communication in the RESTful architecture the JavaScript Object Notation (JSON) [15] format is actively used by TBYS and the code generator. In addition to web

⁴Microsoft (2016). API Reference for T4 Text Templates [online]. Website <https://docs.microsoft.com/en-us/visualstudio/modeling/api-reference-for-t4-text-templates?view=vs-2019> [Accessed: 04-Sep-2019].

⁵Handlebars (2019). Handlebars homepage [online]. Website <http://handlebarsjs.com> [Accessed: 04-Sep-2019].

Table 1. HTML and JavaScript plugin templates.

Component	HTML plugin	JavaScript plugin
Button	button.html	button.js
Button Menu	buttonmenu.html	buttonmenu.js
Check List	checklist.html	checklist.js
Date Picker	datepicker.html	datepicker.js
Date Range Picker	daterangepicker.html	daterangepicker.js
Form	form.html	form.js
Image Viewer	image_viewer.html	image_viewer.js

services, the code generator uses JSON format also for its internal processes. The action and routing information of HTML plugins are stored in a template file in JSON format.

2.2. Development process

In the development process we first decided to create a template-based code generator. As mentioned above, we chose the templatization method for template preparation. After deciding the code generation type and template preparation method, we created a requirement list.

Like all other software systems, code generators also have their own requirements [16]. The determined requirements for the code generator of this study are listed below:

- Reference implementation, templates, and inputs of the system should have well-defined clear syntax.
- Reference implementation and templates should not have bugs.
- There should be functions and arguments to validate the generated code.
- Generated code should be well structured and well documented.
- Generated code should be easily traceable based on the original specification.
- The templates and all of the required inputs should be readily available before starting the code generation process.

During the implementation of the code generator, the C# programming language and ASP.NET MVC platform were used. Templatization is performed by the help of both T4 and handlebars templating languages. After the implementation, the code generator is integrated into TBYS, which is also an ASP.NET MVC web application.

3. The code generator

Two main duties of the code generator are to perform the creation of new modules (code generation process) and to integrate the newly created modules into TBYS. In this section the structure of the code generator and this workflow are explained in detail.

The activity diagram of the code generator is shown in Figure 1 and the stages of the workflow are listed below. Each stage of the workflow is explained in detail in the following sections.

1. Developer creates a table in the database for the module.

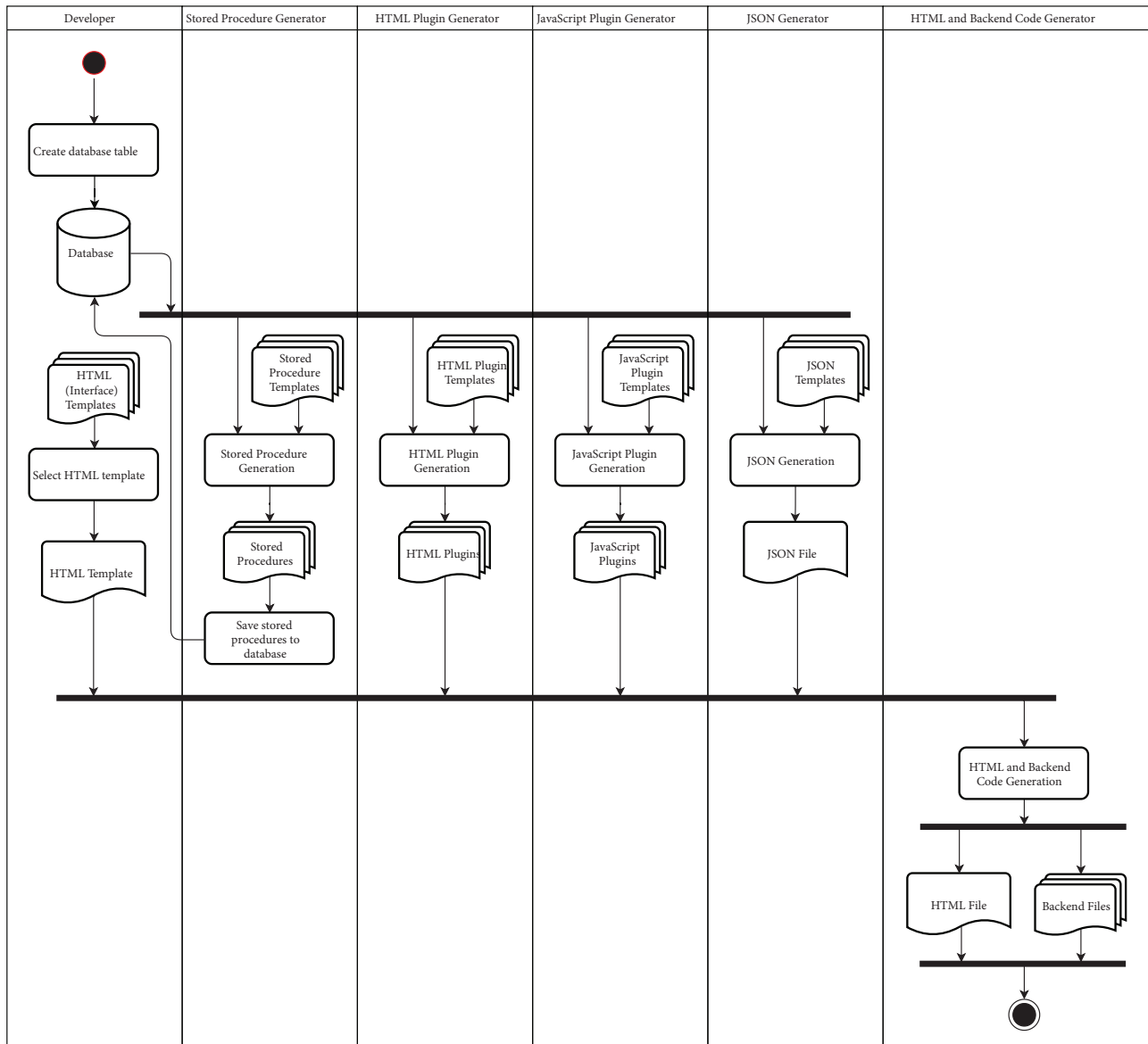


Figure 1. Activity diagram of the code generator.

2. The code generator automatically creates the stored procedures for the operations on the new table.
3. Developer selects an HTML template for the interface.
4. The code generator creates HTML plugins based on the columns of the table.
5. The code generator creates JavaScript plugins for the events of the HTML plugins.
6. The code generator creates JSON schemes based on JSON templates.
7. The code generator creates web services (ASP.NET Web APIs) for server-side communication.
8. The code generator creates the HTML file based on the selected HTML template and previously created HTML plugins, JavaScript plugins, and JSON schemes.

9. The developer and the code generator integrate the newly created module into TBYS.

3.1. Table design and creation

Code generation is started by the developer. In this phase the developer creates a table in the database to store the data of the target module of TBYS. This module can be anything related to the system. For example, the supervisor can request an “annual leave management module” to be developed and integrated into TBYS. In this paper we will give our examples based on this example of “annual leave management module” for clarity and uniformity.

The developer first identifies the required data for the target module. Then, based on this information, the columns, data types, and constraints of the table are determined. For the code generator to handle the table properly and integrate it into TBYS, primary key naming of the table should follow a pattern. The name of the primary key should be the concatenation of the table name and “ID” word. For example, if the table name is determined as `AnnualLeave`, the name of the primary key should be `AnnualLeaveID`.

3.2. Stored procedure generation

Subsequent to the table creation, the code generator generates stored procedures based on this table. During this operation it uses the stored procedure templates developed by the T4 templating language. The names of the stored procedures are given using the “SP\$TableName_FunctionalityName” pattern.

In this pattern, SP is the acronym of the stored procedure. TableName represents the name of the table on which the stored procedure will operate. FunctionalityName represents the type of the operation. For example, the stored procedure that performs the `SELECT` operation on the `AnnualLeave` table will be named `SP$AnnualLeave_SELECT`.

Stored procedures are used for all kinds of CRUD operations on the table. The operations and their descriptions are shown in Table 2.

Table 2. Stored procedure operations and descriptions.

Operation	Description
CREATE	Insert new record
SELECT	Select record
UPDATE	Update record
ONLYVALUE UPDATE	Update only one value
SAVE	Insert new record by controlling duplicated records
DELETE	Delete record
LIST	Select all records without any conditions
LOCK	Grant operation
UNLOCK	Revoke operation

3.3. HTML template selection and plugin generation

Following the creation of stored procedures, the developer selects a HTML template for the target module. The screenshot of the HTML template selection interface is shown in Figure 2. A sample code of a HTML template is given in Listing 1. In the listing, information about the “Button” component is seen.

Listing 1. HTML template code sample.

```

<div class="actions action-form-buttonset">
  {{# each Plugins}}
  {{#if_eq this.Description.Name 'form'}}
  <div class="btn-group">
    {{#each this.Plugins}}
    {{#if_eq this.Description.Name 'button'}}
    {{#button this}} {{/button}}
    {{/if_eq}}
    {{/each}}
  </div>
  {{/if_eq}}
  {{/each}}
</div>

```

Select form you want to create and code template

Form Name

Form Key

Form Production Template

Classic Card entity.form	Classic List basic.list	Map Worker entity.form	EntityForm entity.form
EntityList entity.list	Map Form entity.form	Entity Panel entity.panel	Help Worker help.panel
Image Card image.card	popovers worker popovers.form	Preview List Template preview.list	Entiy_Dashboard Entiy_Dashboard
Custom Context Custom Context	ReportWorker ReportWorker	widget worker Widget	Abstract Form entity.form
Process Request Form entity.form	Process JobPackage Form entity.form		

Visualization Template Name

Save

Figure 2. HTML template selection interface.

After HTML template selection, the code generator creates HTML and JavaScript plugins. Plugins define the view properties and functions of the HTML components that will be placed in the user interface. As stated above there are two types of plugins: HTML plugins and JavaScript plugins. HTML plugins define the view properties. There are also CSS files embedded in the system for each type of HTML plugin. The CSS files are not autogenerated. They are referenced by HTML plugins for view properties. JavaScript plugins define the functionality of the HTML component. There are templates for both types of plugins in the system. Based on these templates the related plugins are created in two stages.

In the first stage the plugins for the common HTML components (like Save button, Delete button, etc.) of the selected HTML template is created. For example, if the template is of type “form”, the code generator

creates a “Save” button (HTML plugin) and its onclick event (JavaScript plugin). The plugins are named by concatenating the table name and functionality name (e.g., `AnnualLeave_save.html`, `AnnualLeave_save.js`).

In the second stage the plugins for the table-specific HTML components are created. During this stage, for each column of the table an HTML component is generated. The type of HTML component is chosen depending on the data type of the related column. For example, for a column of type `date`, a `datepicker` component is created. Naming of the plugins is performed based on the name and the data type of the related column. For example, if the name of the column is “address” and the data type of the column is “string”, the name of the plugins will be assigned as `address_input.html` and `address_input.js`. A screenshot of a generated plugin list is shown in Figure 3.

Html-JS plugins list of connected to form




















Form Plugins	
Name	Template Name
	form 
	save button 
	lock button 
	clear button 
	delete button 
	unlock button 
FaultRequestID	identity input 
FaultRequestDescription	text input 
FaultRequestDate	datepicker 
FaultAppointedDate	datepicker 
FaultAppointedDeadlineDate	datepicker 
FaultNecessarySale	icheck 
FixBeforeExplanation	text input 
FixAfterExplanation	text input 
WorkBeginDate	datepicker 
WorkEndDate	datepicker 
BirimTanimID	selectbox 
FaultRequestUser	hidden input 
PersonelID	selectbox 

Figure 3. Generated plugin list sample.

3.4. JSON generation

In the code generator JSON is also vastly used. There are JSON templates for each type of HTML template. A part of the structure of a JSON template is shown in Figure 4.

The JSON files are created from JSON templates by the code generator based on the selected HTML template, and the table. The created JSON files are used for two purposes: (i) providing communication

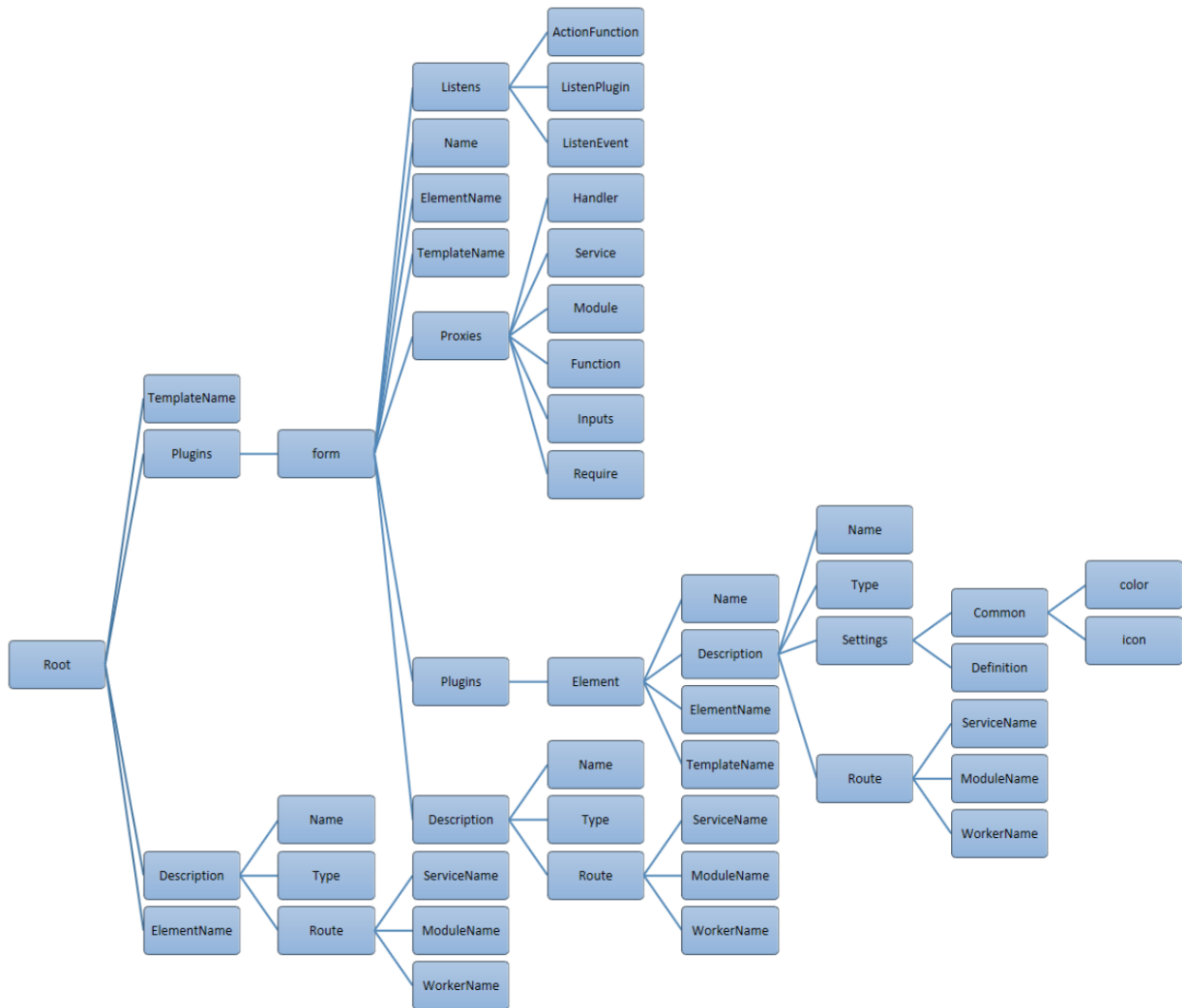


Figure 4. JSON template.

information between the HTML template and its plugins (HTML plugins and JavaScript plugins), and (ii) providing communication information between HTML plugins and their action methods in the ASP.NET application.

In Listing 2 and Listing 3, two code samples of a created JSON file are shown. The former listing shows the information of a “Delete” button and the latter listing shows the information of an “Input” component.

In both listings we can see the information related to the two purposes mentioned above. The general information about the components (name, type, etc.) are related to the first purpose, and the routing information (service, module, worker names, etc.) are related to the second purpose.

After generating the JSON file, a test run is performed. If no error is encountered, the JSON file is saved, and the code generation process continues.

Listing 2. Sample JSON code 1.

```

{
  "Name": "NewsRequest_delete",
  "Description": {
    "Name": "button",
    "Type": "delete",
    "Settings": {
      "Common": {
        "color": "btn-default"
      },
      "Definition": {}
    },
    "Route": {
      "ServiceName": "CorporationIntranet",
      "ModuleName": "NewsRequest",
      "WorkerName": "newsrequestform_13032019_www"
    }
  },
  "TemplateName": "button",
  "ElementName": "NewsRequest_delete"
}

```

Listing 3. Sample JSON code 2.

```

"Name": "NewsRequest_input",
"Description": {
  "Name": "input",
  "Type": "string",
  "Settings": {
    "Common": {
      "icon": "fa-pencil-square-o",
      "icon-type": "input"
    },
    "Definition": {}
  },
  "Route": {
    "ServiceName": "CorporationIntranet",
    "ModuleName": "NewsRequest",
    "WorkerName": "newsrequestform_13032019_www"
  }
},
"TemplateName": "input",
"ElementName": "NewsRequestTitle"

```

3.5. HTML and back-end code generation

The final part of the code generation process is the creation of the HTML file and the related back-end code. The code generator creates these files in several stages. These stages are listed below:

1. The plugins of each component of the HTML template are determined and mapped.
2. The information in the JSON file of each component of the HTML template are determined and mapped.
3. The action methods of the ASP.NET web application are created based on the information from the first two stages.
4. Depending on the information from the first three stages, ASP.NET web APIs are created.
5. The final HTML file is generated based on the information from previous stages.

Just like the JSON creation, a test run is performed after generating the HTML file. If no error is encountered, the HTML file is stored, and the code generation process continues.

3.6. Integration

After web services and HTML file are generated, they should be integrated into TBYS. For this purpose, the code generator stores the HTML file to the database and deploys web services to the server. Then the developer gives a name to the module, and the code generator adds related menu items in TBYS to access the newly created module. Thereafter, the new module can be accessed and used.

4. Empirical evaluation

To measure the effectiveness of the automatic code generation to manual implementation, several experiments were conducted in this study. For the experiments, we first determined the evaluation metrics to help compare the automatic code generation with manual implementation. Afterward, several development tasks were selected. The development tasks were the implementation duties of some modules in TBYS. Then a developer performed the development tasks both manually and by using the code generator. During the experiments we gathered information about each evaluation metric.

In this section, first we explain the above-mentioned experimentation process in detail. Second, we share and discuss the results of the experiments. Lastly, we explain the threats to the validity of our experiments and how we handled them.

The experimentation process was conducted in four stages:

1. Determine the evaluation metrics to compare manual implementation and automatic code generation.
2. Determine the development tasks for the experiments. In other words, determine the modules to implement.
3. Develop selected modules and integrate them in TBYS both manually and by using the code generator.
4. Gather data about evaluation metrics during and after the implementation.

In the first stage of the experiments, we determined the evaluation metrics to compare manual implementation and automatic code generation. The determined evaluation metrics are shown in Table 3. During this stage, we tried to select the metrics that we believed had an impact on the efficiency of the development process and the resource usage.

Table 3. Evaluation metrics and descriptions.

Name	Description
Development time	Time spent during the implementation
Test run	Number of test runs performed during the implementation
Bug count	Number of errors made during the implementation
Code size	Lines of code (LOC) count of the generated HTML file

The development time metric is directly related to the efficiency of the development process. In the literature, this metric was used by Akbulut and Toprak [8] and Possatto and Lucrédio [7]. The selection of the test run metric is because of its effect on the development time. It also has an impact on the resource (central processing units, etc.) usage. Like test runs, the bug count metric also affects the development time. In addition, decreasing debugging efforts may improve developer productivity. Selection of the code size metric is about the errors and the performance of the system. Increase in code size may result in more errors and hence more debugging efforts. This may reduce developer productivity. In addition, ERP systems tend to have

inclusive sets of modules [11, 12] for business processes. For example, in TBYS, there are over 1000 modules. Therefore, code size of modules has an impact on the efficient usage of storage units and network resources. The code size metric was used in the experimentation of a case study by Jugel and Preusner [17].

During the lifetime of an ERP system several different requests are made depending on the needs of the corporations, and these needs are implemented by the developers. In our experiments we tried to simulate this situation. Therefore, in the second stage we first examined the TBYS and identified the modules currently not implemented. As these modules are not present, they probably will be requested in the future.

Afterward, we selected some of these modules for implementation. During this phase we tried to choose different types of modules to better cover the different kinds of scenarios of possible needs. The selected modules are: (i) Fault Request List, (ii) Fault Notification Form, (iii) News Request Form, (iv) Vehicle Request Form, (v) Quality Proposal Form, and (vi) Employee List.

In the third and fourth stages, one developer implemented the selected modules both manually and by using the code generator. The developer who performed the tests is also the developer of the code generator. He is a senior developer. During and after the implementation, data about the determined evaluation metrics were gathered.

4.1. Results and discussion

The evaluation metrics and their data for each module are shown in Table 4. In this table, the columns with the “Man.” header include the values of implementation made manually, and the columns with the “Auto.” header include the values of implementation by using the code generator. The descriptions of the parameters in this table are shown in Table 3.

Table 4. Data gathered from the experiments.

Module name	Template type	Development time (min)		Test run (#)		Bug count (#)		Code size (LOC count)	
		Man.	Auto.	Man.	Auto.	Man.	Auto.	Man.	Auto.
Fault Request List	List	265	4	23	2	18	0	215	78
Fault Notification Form	Form	475	5	36	2	23	0	316	178
News Request Form	Form	545	6	40	2	32	0	422	218
Vehicle Request Form	Form	350	3	22	2	20	0	325	167
Quality Request Form	Form	512	4	37	2	38	0	485	257
Employee List	List	242	3	23	2	16	0	135	63

The development time values show the usefulness of the code generator clearly. In our system the code generator effectively decreased the development time. When using the code generator, most of the time was spent by the developer during the table creation process. After the table creation, the code generation was pretty straightforward and quick. However, the manual implementation took a lot longer since it was necessary to write all of the code by hand.

As stated above, developer productivity is one of the advantages of using code generators. The development time results of the experiments prove this argument. The average development time is 397.18 min during manual implementation and 4.17 min when the code generator is used. This corresponds to 98.95% improvement in average development time.

The second metric shows the number of test runs. We achieved 93.97% improvement in average test run count. When developing with the help of the code generator, two test runs are performed. One run is executed after JSON file generation, and another run is executed after HTML file generation. On the other hand, during manual implementation, a few dozen test runs were performed. This has two reasons: iterative development and bugs. In these experiments we chose to use an iterative development approach. Therefore, in several phases of the development, test runs have been performed to test the implementation of the related operation. Secondly, whenever a bug was discovered, it was fixed and another test run was performed.

As stated above, there are two main advantages of reducing the number of test runs. It decreases both the development time and the usage of computer resources (like central processing units, etc.). These values show that using code generators can help the utilization of computer resources as well as decrease development time.

Bug count is the third metric. As mentioned above, the code generator is a template-based code generator, and we prepared templates from validated reference implementation. Therefore, we encountered no bugs when using the code generator. However, a few dozen bugs were found and fixed during manual implementation. The data about bug count prove the advantage of code generators related to developer productivity.

The values of the code size metric show that the generated HTML file has fewer lines of code than manually created files. We observed 49.37% improvement in average code size. We believe that this is because of the templating method we chose. The code generator works on template files that are created by using validated and well-structured reference implementation. The code generator takes the template files and fills in the required places according to the inputs. On the other hand, developers, being human, have their personal experiences and opinions regarding software development. During the development, they reflect their viewpoints in their code. For example, one developer might prefer to write short, concise code and another developer might prefer longer, more descriptive code.

These results are consistent with the above-mentioned (Section 1) advantages and the benefits of code generators. In addition, template-based code generators require the standards to have been defined prior to the development. In this project this requirement helped the understanding and standardization of the business processes of the corporation.

One disadvantage of the code generator is its inflexibility in needing different types of user interfaces. In our system, it is hard to create any user interface other than the defined HTML templates. If this kind of requirement emerges, two different paths can be taken:

1. All of the required files can be created by the developer manually. In other words, manual implementation is performed.
2. The HTML template file and all of the related templates can be prepared and added to the code generator.

Although this can be a disadvantage of the code generator, it should be noted that the development workload would not be very different from that of manual implementation.

4.2. Threats to validity

There are several threats to the validity of software engineering studies and experiments. Wohlin et al. [13] grouped these threats into four main categories: threats to conclusion validity, internal validity, construct validity, and external validity. Below we discuss these threats from the viewpoint of our experiments.

In terms of conclusion validity, reliability of measures may effect our study. Therefore, we selected evaluation metrics for the experiments from the viewpoint of validity in accordance to two criteria: (i) the evaluation metrics should represent the effectiveness of each approach, and they should be known, and used in literature, and (ii) we should be able to measure the values of each metric precisely during the experiments (like time, LOC, etc.). In other words, they should be objective metrics, and their measurement should be repeatable.

From the viewpoint of internal validity, maturation can be a threat. The outcomes of the experiments can vary over time by the effect of learning the system. The experiments in our study were conducted by one senior software developer who was also the developer of the system. He has a deep understanding of and knowledge about TBYS. Thus, learning bias was avoided.

There may be design threats to the validity of our experiments by means of construct validity. To avoid this threat, we tried to cover different kinds of scenarios of possible needs and properties during the module selection.

External validity tells us that the results of the experiments of software studies should be able to expand on industrial practice. We selected the modules for the experiments by considering possible real-life needs. In addition, the experiments were conducted on TBYS. By doing this, the effectiveness of the code generator was shown on a real-life ERP system.

5. Related work

One of the early works on code generation systems was presented by Balzer [16]. In his paper, Balzer explained the perspective and approach to the code generation process.

In 2013 Altiparmak et al. introduced a source code generation system for multilayered architectures [18]. Their system uses XML and XSLT technologies. In 2017 Akbulut et al. presented an automatic code generation tool for end user development [19]. A code generator framework that helps the development of Smart TV applications for different Smart TV platforms was introduced by Akbulut and Toprak [8].

In 2000 Harrison et al. introduced a method for generating Java implementation code from Unified Modeling Language (UML) diagrams [20]. Their method accepts UML diagrams specified at high levels and generates high-level implementation code. A more recent work on code generation based on UML specifications was presented by Sunitha and Samuel [21]. In their work, they proposed a metamodel to associate Object-Constraint Language (OCL) expressions with UML activity diagrams. By doing this, they aimed to improve code generation from UML models.

Possatto and Lucrédio introduced an automation method that aims to synchronize the templates based on the changes to reference implementation in a template-based code generator [7]. Another study on template-based code generation was presented by Jugel and Preusner [17]. In their paper, they gave a detailed description of a case study of C# API generation from Ecore models [22].

There are also code generators for different domains and purposes. For example, in 2016 Shulga et al. presented a code generator that generates program code from state machine patterns [23]. Another code generator that generates document type descriptors (DTDs) for a collection of XML documents was developed by Leonov and Khusnutdinov [24]. In 2019, Hu et al. introduced a template-based code generator for embedded real-time systems [25].

Besides these, there are numerous studies in the literature about all types of code generation. Interested readers can get more information about code generation studies from literature reviews and systematic mapping

studies. For example, Jörges explained different code generation techniques and state-of-the-art methods about code generation in a book chapter [26]. In 2018 Syriani et al. published a systematic mapping study on template-based code generators [5]. A different systematic mapping study about aspect-oriented model-driven code generation was presented by Mehmood and Jawawi [27]. Another systematic literature review study on code generation from state machine specifications was presented by Domínguez et al. [28].

6. Conclusion

In this paper a template-based code generator for web applications was introduced. The code generator was developed and integrated into a real-life web application, which is an ERP system (TBYS), successfully. In this paper the design, development, integration, and workflow processes of the code generator were explained. The experiences regarding this development were shared. We also explained the inner workings of the system. By using experimentation, we evaluated automatic code generation and compared it with manual implementation. The results were shared and discussed.

Currently the developed code generator is actively used in a production environment by TÜSSİDE. This project proves that code generators can be effectively used in ERP systems and large-scale real-life web applications. In addition, this study and the experimentation prove that code generators can increase developer productivity.

Today code generators are becoming more common in industry because of their many advantages mentioned above. Also, there is active research on this subject. In the future, we believe that different approaches to automatic code generation for different domains and programming paradigms will continue to be developed.

Acknowledgments

This project was developed by TÜBİTAK TÜSSİDE and this paper was published with the contributions of Sakarya University.

References

- [1] Fielding RT, Taylor RN. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology* 2002; 2 (2): 115–150. doi: 10.1145/514183.514185
- [2] Tragatschnig S, Stevanetic S, Zdun U. Supporting the evolution of event-driven service-oriented architectures using change patterns. *Information and Software Technology* 2018; 100 (3): 133–146. doi: 10.1016/j.infsof.2018.04.005
- [3] Papazoglou MP. Service-oriented computing: concepts, characteristics and directions. In: *Fourth International Conference on Web Information Systems Engineering (WISE 2003)*; Rome, Italy; 2003. pp. 3–12. doi: 10.1109/WISE.2003.1254461
- [4] Rich C, Waters RC. Automatic programming: myths and prospects. *Computer* 1988; 21 (8): 40–51. doi: 10.1109/2.75
- [5] Syriani E, Luhunu L, Sahraoui H. Systematic mapping study of template-based code generation. *Computer Languages, Systems & Structures* 2018; 52: 43–62. doi: 10.1016/j.cl.2017.11.003
- [6] Cooper KD, Torczon L. *Engineering a Compiler*. 2nd ed. Boston, MA, USA: Morgan Kaufmann, 2011.
- [7] Possatto MA, Lucrédio D. Automatically propagating changes from reference implementations to code generation templates. *Information and Software Technology* 2015; 67: 65–78. doi: 10.1016/j.infsof.2015.06.009
- [8] Akbulut A, Toprak S. Code generator framework for smart TV platforms. *IET Software* 2019; 13 (4): 268–279. doi: 10.1049/iet-sen.2018.5157

- [9] Völter M, Bettin J. Patterns for model-driven software development. In: 9th European Conference on Pattern Languages of Programmes; Irsee, Germany; 2004. pp. 525–560.
- [10] Muszynski M. Implementing a domain-specific modeling environment for a family of thick-client GUI components. In: 5th OOPSLA Workshop on Domain-Specific Modeling; San Diego, CA, USA; 2005.
- [11] Peng GCA, Gala C. Cloud Erp: A new dilemma to modern organisations? *Journal of Computer Information Systems* 2014; 54 (4): 22–30. doi: 10.1080/08874417.2014.11645719
- [12] Charland P, Léger PM, Cronan TP, Robert J. Developing and assessing ERP competencies: basic and complex knowledge. *Journal of Computer Information Systems* 2016; 56 (1): 31–39. doi: 10.1080/08874417.2015.11645798
- [13] Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B et al. *Experimentation in Software Engineering*. Berlin, Germany: Springer, 2012.
- [14] Ramakrishnan R, Gehrke J. *Database Management Systems*. 3rd ed. Boston, MA, USA: McGraw-Hill, 2002.
- [15] ECMA International. *ECMA-404. The JSON Data Interchange Syntax Standard*. 2nd ed. Geneva, Switzerland: ECMA International, 2017.
- [16] Balzer R. A 15 year perspective on automatic programming. *IEEE Transactions on Software Engineering* 1985; SE-11 (11): 1257–1268. doi: 10.1109/TSE.1985.231877
- [17] Jugel U, Preusner A. A case study on API generation. In: Kraemer FA, Herrmann P (editors). *Lecture Notes in Computer Science*, Vol. 6598. Berlin, Germany: Springer, 2011, pp. 156–172.
- [18] Altiparmak HC, Tokgoz B, Balcicek OE, Ozkaya A, Arslan A. Source code generation for large scale applications. In: *International Conference on Technological Advances in Electrical, Electronics and Computer Engineering*; Konya, Turkey; 2013. pp. 404–410. doi: 10.1109/TAECE.2013.6557309
- [19] Akbulut A, Patlar Akbulut F, Köseokur H, Çatal Ç. Design and implementation of an automatic code generation tool for end-user development. *Dokuz Eylül University Faculty of Engineering Journal of Science and Engineering* 2017; 19 (55.1 Special Issue): 76–88 (in Turkish with an abstract in English). doi: 10.21205/deufmd.2017195532
- [20] Harrison W, Barton C, Raghavachari M. Mapping UML designs to Java. *ACM SIGPLAN Notices* 2000; 35 (10): 178–187. doi: 10.1145/354222.353184
- [21] Sunitha EV, Samuel P. Object constraint language for code generation from activity models. *Information and Software Technology* 2018; 103: 92–111. doi: 10.1016/j.infsof.2018.06.010
- [22] Steinberg D, Budinsky F, Paternostro M, Merks E. *EMF: Eclipse Modeling Framework*. 2nd ed. Boston, MA, USA: Addison-Wesley Professional, 2008.
- [23] Shulga TE, Ivanov EA, Slastihina MD, Vagarina NS. Developing a software system for automata-based code generation. *Programming and Computer Software* 2016; 42 (3): 167–173. doi: 10.1134/S0361768816030075
- [24] Leonov AV, Khusnutdinov RR. Study and development of the DTD generation system for XML documents. *Programming and Computer Software* 2005; 31 (4): 197–210. doi: 10.1007/s11086-005-0032-6
- [25] Hu K, Duan Z, Wang J, Gao L, Shang L. Template-based AADL automatic code generation. *Frontiers of Computer Science* 2019; 13 (4): 698–714. doi: 10.1007/s11704-017-6477-y
- [26] Jörges S. The state of the art in code generation. In: Jörges S (editor). *Construction and Evolution of Code Generators: A Model-Driven and Service-Oriented Approach*. Berlin, Germany: Springer, 2013, pp. 11–38.
- [27] Mehmood A, Jawawi DNA. Aspect-oriented model-driven code generation: a systematic mapping study. *Information and Software Technology* 2013; 55 (2): 395–411. doi: 10.1016/j.infsof.2012.09.003
- [28] Domínguez E, Pérez B, Rubio ÁL, Zapata MA. A systematic review of code generation proposals from state machine specifications. *Information and Software Technology* 2012; 54 (10): 1045–1066. doi: 10.1016/j.infsof.2012.04.008