DOI: 10.2298/CSIS110720007K

Experimental investigation of the quality and productivity of Software Factories based development

Andrej Krajnc[1], Marjan Heričko[1], Črt Gerlec[1], Uroš Goljat[1] and Gregor Polančič[1]

[1] University of Maribor,
Faculty of Electrical Engineering and Computer Science,
Smetanova ulica 17, SI-2000 Maribor, Slovenia
{andrej.krajnc1, marjan.hericko, crt.gerlec, uros.goljat, gregor.polancic}@ uni-mb.si

Abstract. Software organizations are always looking for approaches that help improve the quality and productivity of developed software products. Quality software is easy to maintain and reduces the cost of software development. The Software Factories (SF) approach is one of the approaches to provide such benefits. In this paper, the quality and productivity benefits of the SF approach were examined and evaluated with an experiment involving two treatments - the traditional and the SF approach. For the purposes of this experiment, the Goal – Question – Metric (GQM) approach was used. Participants were grouped into thirty-two teams. There were sixteen projects available. The results were evaluated and presented through quality and productivity criteria, which were used for the experimental study. The results showed that the Software Factories approach was significantly better than the traditional approach.

Keywords: software factories approach, benefits, quality, productivity, experiment.

1. Introduction

A continuous objective in software engineering is to develop high quality solutions within a short time [3, 6, 15]. This can be achieved with the use of known software development methods, or approaches, where the quality of solutions is provided [38]. In most cases the project stakeholders would like to evaluate the software development outcomes as well as the effectiveness and efficiency of the underlying software development approach.

Several approaches that help to decrease time and effort in software development activities exist [4, 5, 8, 9], whereas the most efficient way of creating software is not to develop it, but rather reuse it. The biggest motivation for reusing software assets is to decrease software development costs and reduce the time and effort needed for their development. Software quality can also be improved with software reuse [36]. When reusing software parts, it also improves maintainability, because of the use of already well tested software parts. When we discuss software reuse, we need to look at two different aspects of software reuse: developing for reuse and developing with reuse [36]. The first is important when something is developed for reuse, like a component or some software part, while the latter is important when such a component or part of the software code is reused. Over time, a large amount of different approaches and techniques for software reuse have been developed, including: software frameworks, software libraries, software generators, design patterns and software product lines.

Within these approaches, software frameworks and software product lines (abbreviated as SPL) have been established as one of the most successful approaches for software reuse, because their reuse is based on product families rather than on individual reuses [2, 18]. In relation to them, a new approach for successful software reuse has evolved over the past few years: Software factories (abbreviated as SF).

A SF is a pattern for an approach to software system development, in which instances of those systems share features, functionality and architecture [2, 3]. The underlying four concepts of SF are: SPL, architecture framework, automated guidance and Model-Driven Development (abbreviated as MDD). Leveraging these concepts, SFs provide knowledge in the following forms: asset-like architectural frameworks with common features, models to create parts of software patterns, and recipes and tools for helping the developer. These assets help to automate the delivery of members of an SPL. In other words, an SF can produce software solutions in a way analogous to the way an airplane factory produces airplanes. A certain SF can produce software products in a specific domain. If we have an SF for mobile applications and we want to develop a web portal, we have to use another SF.

The SF approach provides the following benefits [2, 3, 4, 6]: an increase in productivity, a decrease in the time to market, an increase in the level of reuse, the providing of automatic guidance, a higher level of abstraction and an increase in product quality. Some of these benefits can also be adopted from the SPL approach [6, 7].

1.1. Motivation for the study

As Lord Kelvin (1824-1907) noted [40] “To measure is to know” and “If you cannot measure it, you cannot improve it.” By applying this idea to the SF domain, we believe that the measurement of development approaches helps to control, estimate and improve development processes and consequently the organization.

One of the main objectives of software engineering is to continuously improve the quality of outcomes as well as the efficiency of engineering activities [38]. As stated in the previous section, several approaches that leverage these objectives do exist; however, if the effectiveness of these approaches is not objectively analyzed, we cannot generalize assumptions out of them.

The above statements are also valid in the SF domain, with many stated benefits (see the previous section) and researchers have reported that there is a lack of empirical investigations [41]. There have been some studies related to measuring SF benefits [4, 5, 18], but their results have been primarily focused on quality characteristic, like number of defects, and reusability. Another motivation for our study was to test the theoretically [2, 3] stated quality and productivity benefits of the SF approach in an empirical way. Doing such a study was also a test to see if the SF approach as such was mature enough to be used later on in real projects in the industry. Another motivation for this study was to motivate participants to use more advanced development approaches to develop solutions with higher quality.

The goal of this study was to empirically evaluate the SF approach and to investigate if it is more effective and efficient when compared to traditional development.

In our research, we have addressed and evaluated SF in terms of their quality and productivity, compared to a "traditional" software development approach. The traditional approach has been defined as *"a software development approach where the whole software product is built by developers from scratch."* This means that no additional tools that help generate source code and no explicit design patterns are used. There is also no reuse of any already available code.

For our study we set up a following research question: "*Does the SF approach deliver better quality code and does it increase the productivity of the development team compared to a traditional approach?*"

According to the research question, we organized the paper as follows. This section further investigates software quality and productivity. Section 2 describes research foundation for this work; section 3 describes work related to the object of the investigation; section 4 describes the goals of our study, the hypotheses associated with the study and the design of the experiment. The results are presented and interpreted in section 5. In section 6, we have listed our conclusions together with the limitations, as well as the theoretical and practical implications.

2. Research foundation

In the following subsections, research foundation regarding software quality and productivity is presented.

2.1. Software quality

The primary objective of software development projects is to fulfill either the stated or implied user requirements, which are commonly conceptualized in terms of software quality [38, 39]. To achieve high quality, it is important to use proper measurements for software solutions [38]. International standards, like ISO 9126, emphasize the need for measurements for assuring product quality [30]. It is important to measure software from the beginning of development until the end of the product's lifecycle. Software quality is divided into internal and external quality [30]. Internal quality is the totality of characteristics of the software product from an internal view [30]. Internal quality is measured and evaluated against the internal quality requirements. The details of software product quality can be improved during code implementation, reviewing and testing [30]. External quality is the totality of characteristics of the software product from an external view [30]. It is the quality when the software is executed, which is typically measured and evaluated while testing in a simulated environment with simulated data using external metrics [30]. Despite the fact that new software development methods and approaches have been applied in software development, there are still problems with the quality of software products [38]. Quality is judged according to different characteristics [38], which is given different significance for different stakeholders. Therefore, as previously mentioned, different views of software quality can be observed and analyzed. There are differences in analyzing quality from the point of view of customers or users on one hand, or from the point of view of the development team on the other [30]. As mentioned, we are firstly interested in external quality and subsequently in internal quality (Fig. 1).

When measuring internal quality, we use different software metrics [38] that cover large aspects of object-oriented development, like complexity, inheritance, coupling, and cohesion [10, 16]. It is important to have goal-oriented measurement; there you can define clear objectives that you want to achieve with a measurement [12, 38].

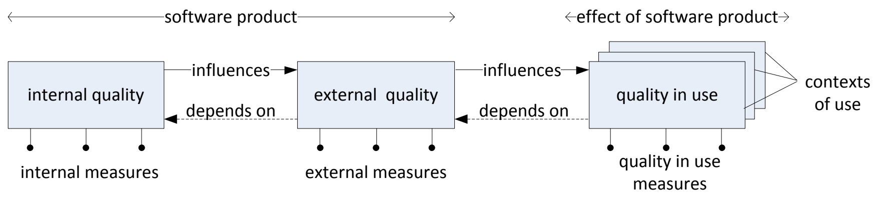

Fig. 1 Relationships between types of software quality [30]

In research about the SPL and SF approach, there are many cases regarding improved quality [3, 4, 6, 29, 37]. These cases are primarily related to external quality measurement (like the number of defects) or they theoretically state the quality benefits of such an approach. However, researchers should also focus on the internal quality of the SF approach and its derived software code, as well as its quality.

2.2. Software productivity

Productivity is one of the most important benefits, when considering the evaluation of a new software approach. Due to the ever-increasing demand for new software, it is important to use an approach that increases the productivity of the development team and shortens the time that is necessary for solutions to get to the market [35]. Productivity also depends heavily on different aspects, such as the experience of the developers, the approach used, the development environment, and what domain the solutions will be developed in (as well as the knowledge the developers have of such a domain), etc [1, 5, 29]. Productivity has been widely researched and analyzed in different contexts and approaches.

Measuring software productivity has been discussed in different ways [1, 15, 32, 39]. Different metrics have been proposed and used [39], like using size related metrics - Lines of Code (LOC) and Number of Classes (NOC), when using object-oriented development. The most important measure of software productivity is to measure the effort needed for the development of a certain project, product or functionality [38, 39]. Effort is mostly presented as the time, in hours or minutes, required for development [15].

Another view on the SPL or SF productivity benefit deals with the economics of such an approach and the economics of reusability. There are several models that discuss software product line economics. Most decisions about which products to include in the SPL and how to organize and structure the development of the products are economic decisions. One of the most known models is the model SIMPLE [31]. The model provides a set of functions that account for the expenses and benefits of building the product line and operating the product line organization [31]. It helps an organization decide if it should adopt the software product line strategy to build products through the costs and benefits related with the use of the SPL approach. Poulin [35] presented a model for estimating the financial benefits of software development with SPL. The model was used to calculate the “Product Line Return on Investment (ROI)” metric. In this analysis, Poulin also used an LOC metric and the percentage of the reuse code in each project. The authors in [32, 33] present the economic impact on adoption of an SPL approach. Their findings were related to the increase of quality, and improved productivity. With their model, they present a top-down approach to evaluate the SPL process of development. An important part of their study was to study the reuse effort. When talking about reuse effort, there are different approaches to measuring reusability, such as the level of the reuse in organizations, reuse Return on Investment (ROI) metrics and the effort needed for the development for reuse [34]. These approaches are focused mostly on improving productivity and better quality when reusing software code or components.

3. Related work

In this section, related work will be presented. Related work is considered from different areas of software development, from software factories, software product lines and software frameworks.

3.1. Software factories

Software factories were largely presented by Greenfield, Short [2] and Lenz, Wienands [3]. The authors addressed and stated the benefits that this approach delivers with its adoption.

Menendez [17] performed a study of SF-based projects in the aerospace industry. He demonstrated a 35% productivity improvement in his research. In our research, we want to show that improvements can also be made for smaller projects. A limitation of this research is that the study was performed for demonstration purposes only and was only applicable for the relatively small domain of navigation in the aerospace industry.

Aoyama [19] presented an evolution of the SF approach at Fujitsu. A model for using the SF approach was adopted. Their productivity improvement was about 30% higher than the development process that was used before the SF approach. Some other benefits, including higher productivity, have also been gained, such as the incremental delivery of products, lower total costs and a shorter development cycle.

Matsumoto [28, 29] presented SF which was established at Toshiba. In Japan, several companies have used the SF approach, which helped to reduce the cost of software development and increased the quality of software. Each of them, especially Toshiba, achieved high levels of productivity and improved quality of software. For example, Toshiba achieved 0.2 detected errors per 1000 LOC.

3.2. Software product lines

Knauber et al. [11] defined several hypotheses related to SPL, where for the scope of our research, the following hypotheses are interesting:

- “SPL decrease the development effort per product.” We will adopt this hypothesis and change it a little bit to use with our productivity criteria.
- “SPL decrease the time to market per product”, which will also be adopted into our productivity hypotheses.

A limitation of their work is that their findings are based on theoretical conclusions, rather than on empirical data.

Ajila and Dumitrescu [13] conducted research about SPL evolution in the form of changes. The original goal of this research was to study the economic impact of market repositions on the product line and the identification of metrics that can be used to record changes in the product line. One of the goals was to measure the lines of code (LOC) metric in each period of development. They also measured the efficiency of the product line in the case of developing products. The results showed that the use of the product line approach eases the integration of a new product. In addition to this, the efficiency also increased. The limitation of this research was that the authors only used the Lines of Code (LOC) metric. No metrics for code complexity were used.

In [20], Chen presented an SPL process simulator. He simulated the process of the development life cycle in terms of time-to-market. The simulation results showed that time-to-market can be reduced. The data gathered in the simulation was compared to theoretical data in [11]. According to our research, where the real projects' data was used, their data was gathered from a simulator and the data input in a simulator was selected from randomly distributed numbers within a certain range.

3.3. Software frameworks

Object-oriented frameworks have been largely researched. Because of their important relation with SPL and SF, research work made on the productivity and quality of object-oriented frameworks is relevant.

Polančič et al. [21] presented an empirical examination of application frameworks success. They did a survey regarding several important factors. One of the factors also covered productivity and quality. In a survey with 389 participants, the average answer with regard to productivity improvement using frameworks was “agree” while the same amount were in agreement with quality improvement. Both marks are on Likert scales of 1-7 with end points of “strongly agree” and “strongly disagree”. The limitation of this paper was its research method compared to our research. The use of a survey can sometimes get objective answers from participants.

Basili et al. [15] did an experiment to better understand the benefits of reuse in an object-oriented framework. They conducted a four-month long experiment, in which a new project was developed. The results showed an approximately 34% reuse rate. Their findings were that productivity improved with the increase of the reuse rate. Productivity was presented as an equation between size and effort time. In the paper, no data about quality was presented.

Morisio et al. [22] presented an empirical study in an industrial context on the production of software using a framework. They tested hypotheses regarding productivity and quality. They made a direct comparison with the traditional approach. The limitation of this research is the development process, where all projects were developed by the same programmer. Also, quality was measured in the relationship between development effort and the rework effort required to correct the code. Productivity was markedly better at about 50%. It should be emphasized that really small projects were used; the largest had 2,673 lines of code. The limitation of this work is also that all projects were developed by a single developer.

Mamrak and Sinha [23] did a case study of productivity and quality gains using an object-oriented framework. They implemented several input forms using the framework. The effort they needed to generate and implement a new application using a framework was reduced by 23% in terms of the average lines of code written. By comparison, only the LOC metric was used and the quality factor was presented through reused code.

In [25] authors presented a study to assess the impact of experience and maturity on productivity in software development. Two projects were measured, one using initial and one using subsequent development. First project was developed using new platform. For the quality measurement software metrics have been used. The project developed with new platform had about four times higher development effort. On the other hand, quality measures were both, more or less equal and there was no significant difference. A limitation of the study was that the participants were not familiar with developing with a new platform.

Table 1. Summary of related work.

Author	Research area	Methods	Factors	Results
Menendez[17]	SF	Case study	Productivity	35% productivity improvement
Aoyama [19]	SF	Case study	productivity, shortened development time	productivity improved by about 30%
Matsumoto [28, 29]	SF	Case study	productivity, quality	higher level of productivity achieved, quality presented as less number of defects
Knauber et al.[11]	SPL	Theoretical study	productivity, shorten time to market	hypothetical graphs of improved productivity
Ajila and Dumitrescu [13]	SPL	Case study	size of product code (productivity), changes on the product line	lower time-to-market for products, integration of new product is easier and more efficient

Chen [20]	SPL	Simulation	effort reduction, time-to-market reduction	improved development effort after a number of products developed
Polančič et al. [21]	Software Frameworks	Survey	different factors for the acceptance of object-oriented frameworks, including productivity and quality	participants agreed with productivity and quality improvement when using software frameworks
Basili [15]	Software Frameworks	Experiment	productivity, level of reuse	as reuse rate in projects increases, productivity increases
Morisio et al.[22]	Software Frameworks	Experiment	productivity, quality	productivity improved by about 50%
Mamrak and Sinha [23]	Software Frameworks	Case study	productivity, quality	23% less LOC written
Tomaszewski and Lundberg [25]	Software Frameworks	Experiment	productivity, quality	productivity was improved about four times higher using new approach, quality measures show no significant difference
Our study	SF	Experiment	productivity, quality	Improves productivity and effort for about 14%, better quality of code

4. The empirical study

The objective of our study was to investigate the impact of the SF approach on software quality and software development productivity, as compared to the traditional approach. To achieve the objective, we used between-subjects based on an experimental method. The experimental research as described in this article was a part of a larger study in which we investigated different effects of the SF approach on the software development process. For the purpose of this part of the research, we have narrowed it down and focused on the quality and productivity of the code.

4.1. Experimental variables and hypotheses

As previously mentioned, for the purposes of this research, we defined the criteria that are included in the literature as the advantages/benefits of SF [2, 3] and SPL [6, 7]. The GQM approach [12] (Table 2) for defining factors and corresponding metrics from the stated objective of the research was used.

Table 2. GQM approach [12] for our research.

Goal	Purpose	Experimental investigation
	Issue	Quality and productivity benefit
	Object	Software Factories based approach
	Viewpoint	From the developers' viewpoint
Question	Does the SF approach increases *productivity* and decreases the time of development?	Does the SF approach deliver a better *quality* of code?
Metric	Effort Time, LOC, NOC	Quality index (QI)

4.2. Productivity hypothesis

One of the core benefits of SF is in alignment with the following statement: “The Software Factories approach increases the productivity of the development team and decreases the time needed for development [3]”. As mentioned in Section 2.2, for productivity, it is also important that the development be done with software reuse, because it shortens the development time. As stated in the motivation for the study, the goal was to measure if the development effort would be improved with the use of the SF approach. In our study, productivity was defined as the amount of work indicated with the sum of hours needed for the implementation of a project, Lines of Code (LOC) and the Number of Classes (NOC) metrics. According to this, we declared the following hypothesis:

H1: The time needed for the development of software projects using a traditional approach is greater than the time and effort needed for the development of software projects when using the Software Factories approach.

The corresponding null hypothesis states that there is no difference between the two groups of approaches (SF versus traditional approach) in light of the productivity.

4.3. Quality hypothesis

As introduced in Section 2.1, quality can be measured internally and externally. In our study, we have chosen to measure internal quality. With internal quality we can cover different aspects of software code and its quality.

Several software development quality metrics exist [24]. In our study we have decided to measure the quality of code with software metrics sets presented in [14].

We tested the code quality with both approaches (SF and traditional). The metrics used in this criterion provided numerical values. These values represent objective metrics, because the input in the metric function is data and the output from the function is a single numerical value. In this case, there is no impact on the results. On the other hand, if one looks at the subjective metrics, there can be some influence on the results. This is the reason why we decided to take software product metrics [10, 24], which were already tested and are statistically proven. Size-related metrics were used in the productivity phase of the measurement. Quality measurement was realized with object-oriented class-related metrics [1, 14]. The chosen metrics cover the coupling, complexity and maintainability of classes [16].

The Quality Index (QI) was proposed in [1, 14]. The authors defined the method (equation) that expresses the quality of code with different metric sets. In [14], interchangeable metric sets were evaluated to calculate QI. We used those interchangeable metric sets to evaluate the quality of code in both approaches.

The Quality Index is defined as $QI = \frac{\sum_{i=1}^{n} PMQR_i}{n}$

$$PMQR_i = f_i(m\gamma)$$

where *PMQR* is the product metric quality rating, which is between 0 and 5 value, *n* is the number of code metrics used in the calculation, *mv* is a code metric value and *f* is the function that transforms the metric value for the metric *i* to the product metric quality rating [1]. The quality rating transformation function is defined for each metric individually. The QI is composed of *n* product metrics. The number of metrics and its type should be defined according to the project and environment characteristics. Each product metric has its threshold values [1, 14]. For the original quality index QI(0), the following metrics have been chosen: the Depth of Inheritance Tree (DIT), Coupling between Objects (CBO), Lack of Cohesion in Methods (LCOM), and the Maintainability Index (MI). According to these statements regarding quality, the following hypothesis was declared:

H2: The Software Factories approach delivers code that has higher quality than code delivered with a traditional approach.

The corresponding null hypothesis states that there is no difference between the traditional and SF approach.

4.4. Experimental participants

We were aware that our ideal candidates for experimental subjects would be a group of people, who already had prior knowledge in the field of SF and approximately equal experience and familiarity with the SF approach. For practical reasons we searched for candidates among undergraduate students of the same course, which we previously trained to have the same amount of training with each approach available (SF and traditional). In this way, we minimized the effect that different prior knowledge or experience could have on the experimental results.

Because advanced development technologies were presented, the students needed to have experience with object-oriented technology, including an object-oriented programming language.

Participants were in their final year of their study, so they had a sufficient development and programming experience.

The reason for selecting final year students was, because research was based on a complex technology, such as SF approach is. We provide participants with extra help during courses, as mentioned before; training of each approach was available for them and a short questionnaire about their programming experiences was made. All that was provided for this, that relation between university students and subjects in other context (like more experienced developers) can be drawn.

4.5. Experimental objects

In this part, we will present the objects that the subjects are going to examine or work with. The objects are applications or projects, developed as web applications in different domains using the SF and traditional approach. In Table 3, the projects and their domains are presented. In the third cell, a percentage of the Core Asset Base (CAB) is presented when using the SF approach. With the creation of a project based on the SF approach, a common part for each project developed in such a way is created. Measured in the LOC metric, this CAB part contains 1,072 lines of code. Everything else represents the variabilities of each project. For the SF approach, a Microsoft Web Client Software Factory was used, which is a package for developing web applications using the SF approach in Microsoft Visual Studio Environment. As previously mentioned, this package creates a core project for developing web applications (ASP.NET) and uses known patterns for development, such as the Model-View-Presenter and Model-View-Controller. On the other hand, subjects who developed objects with a traditional approach, use the object-oriented development paradigm and the ASP.NET technology for developing web-based applications on the Microsoft platform. Both groups used the object–oriented programming language C#. Projects were developed in the Microsoft Visual Studio environment.

Table 3. Objects examined by subjects in study

Project	Domain	CAB (%)
Project1	Accounting (web application)	13.05
Project2	Warehousing (web application)	8.45
Project3	Banking (web application)	9.95
Project4	Warehousing (web application)	12.95
Project5	Accounting (web application)	13.46
Project6	Warehousing (web application)	10.68
Project7	Warehousing (web application)	26.08
Project8	Other services (cinema services – web application)	15.61
Project9	Other services (restaurant services – web application)	9.40
Project10	Other services (taxi services – web application)	15.54
Project11	Banking (web application)	18.35
Project12	Accounting (web application)	19.93
Project13	Accounting (web application)	10.74
Project14	Banking (web application)	10.17
Project15	Warehousing (web application)	11.91
Project16	Warehousing (web application)	10.16

4.6. Experimental environment and instruments

The experiment was conducted at a university setting, within the laboratory work of a subject related to SF. In this laboratory work we taught students how to develop multi-tier applications over a four-month long course. We used the process of this laboratory work for our experiment in such a way that the students had to build web applications using SF approach and traditional approach.

The experimental process started with a short questionnaire about the students' programming knowledge and experiences.

There were sixteen projects available and two different approaches used: the SF approach and the traditional approach. The first thing the students did was randomly choose their project and the approach they were going to use. Every project was developed in two different ways: one student developed it with a traditional approach and one with an SF approach.

Table 4. Experimental study.

R	G_{SF}	O_1	X_{SF}	O_Q	O_P
	G_{TR}	O_1	X_{TR}	O_Q	O_P

Notes: R… randomization process, X_{SF} … **treatment** SF approach, X_{TR} … **treatment** Traditional approach, **G_{SF} … Group** SF approach, **G_{SF} … Group** Traditional approach**, O … observation, O_Q … observation** quality**, O_P … observation** productivity

The development process was divided into iterations.

In the first two iterations, the students had to be grouped together by project and had to complete the requirements. They also conducted a design phase for projects. For every project, quality design specifications were prepared and provided by the supervisor. It was necessary to check and analyze the specifications together with students per project, because both needed to have the same requirements and design. That was for the purpose of the SF benefits evaluation. Students filled out a document about their working status and productivity. After the design phase, the document was examined by a supervisor, who provided comments on the requirements.

After that, the education of developing web applications was turned over to the students. Also, tutorial implementations were added on the course site for them. Complete documentation was also provided. The supervisor was also available during courses to answer questions about the use of developing web applications and developing web applications using the SF approach.

Then the implementation phase began. The first step for students was to set up a project solution. Their task was to write down the time they spent learning the technology or use of the SF approach. The next step was the implementation of the project. For the implementation phase, a Microsoft

Web Client Software Factory was used for the SF approach while students who developed traditional approach used the ASP.NET technology for developing web-based applications on the Microsoft platform.

5. Experimental results

5.1. Descriptive statistics

All participants had some experience with object-oriented programming languages and relational databases, and therefore had the basic skills necessary for such a study (Table 5).

Table 5. Descriptive statistics.

Variable	Values	Freq.	Valid percent (%)
Gender	Female	3	9.4%
	Male	29	90.6%
Programming experience	Basic	26	81.3%
	Advanced	5	15.6%
	Expert	1	3.1%
Years of programming experiences (besides study)	0 years	16	50.0%
	< 1 year	11	34.4%
	1 – 2 years	3	9.4%
	>2 years	2	6.2%
Programming knowledge of .NET environment	Basic	26	81.3%
	Advanced	2	6.2%
	Expert	3	9.4%
	I don’t use .NET	1	3.1%
Knowledge of developing web applications in a .NET environment	Basic	20	62.5%
	Advanced	4	12.5%
	Expert	2	6.2%
	Don’t know	6	18.8%

Table 5 shows the descriptive statistics of the experiment's participants and their previous experience with programming in a .NET environment. This experience was self-reported. As noted in Table 5, we analyzed 32 out of a total 32 responses. As anticipated, the typical participant was a male who was introduced to the object-oriented development during the course of their college studies and had some programming experience.

5.2. Productivity hypothesis testing

In the GQM model (Table 2) [12], we set out to measure productivity with actual hours and the size-related metrics Lines of Code (LOC) and Number of Classes (NOC). The projects developed with the SF approach were measured together with part of the code, which represented the percentage of CAB code, as presented in Section 4.5.

Table 6. Actual hours spent on each project.

Actual spent hours				LOC		NOC	
	TR	SF	%	TR	SF	TR	SF
Project1	115	91	20.87	3289	8214	28	221
Project2	108	112	-3.70	2381	12691	33	276
Project3	124	104	16.13	5971	10771	46	140
Project4	106	89	16.04	5585	8281	53	190
Project5	112	107	4.46	3004	7965	37	239
Project6	124	99	20.16	6420	10038	36	219
Project7	108	91	15.74	2163	4111	28	129
Project8	116	94	18.97	1577	6867	13	158
Project9	122	96	21.31	7970	11409	84	297
Project10	121	104	14.05	2647	6899	34	151
Project11	107	96	10.28	1758	5842	26	154
Project12	116	97	16.38	3672	5378	48	118
Project13	116	98	15.52	2027	9983	25	217
Project14	108	92	14.81	4486	10542	36	249
Project15	104	88	15.38	2652	9002	18	227
Project16	123	103	16.26	5396	10552	48	291
Sum	1830	1561					
Mean	114.38	97.56	14.70	3812.38	8659.06	37.06	204.75
STDEV	6.97	6.89		1926.57	2390.23	16.69	58.08
p (H_0)	0.00			0.00		0.00	
df	30			30		30	
t	6.86			-6.31		-11.10	

The actual hours spent were compared in a pair. The null hypothesis H_{01} was tested, which stated that mean values in actual spent hours in both approaches are equal. The alternative hypothesis H_{A1} states that mean values in actual spent hours are not equal. Both tests were also made with LOC and NOC results.

According to the summarized data, there is a difference in the hours needed for the development phase in both approaches. Participants using the SF approach needed, on average, 14.70% less time to develop projects.

Table 6 shows the actual hours spent on each project. In the data, we can see that projects using the SF approach needed less time. This difference with other projects is also seen in the result of the size-related metrics, Lines of Code (LOC) and Number of Classes (NOC). This difference is especially visible in the traditional approach. The LOC value in projects using the traditional approach is high. The effort results are in favor of SF approach. Table 6 also shows statistics for the tested pairs. For all three factors - spent hours, LOC and NOC value - the difference is significant at $p<0.05$. The evaluation of results for the productivity criteria shows that the time and effort needed for the development of a SF project is less than the time and effort needed for the development of projects with a traditional approach. Therefore, the null hypothesis (H_{01}) is rejected in favor of the alternative hypothesis (H_{A1}).

5.3. Quality hypothesis testing

Our research was based on the quality of the code, which was measured with selected software product metrics. Also, the projects here developed with the SF approach were measured together with part of the code which represented the percentage of CAB code, as presented in Section 4.5. Based on [1, 14], the quality index was tested on data. The QI measure was compared in a pair. The null hypothesis H_{02} was tested, which stated that mean values in QI measure are equal. The alternative hypothesis H_{A2} states that mean values in QI measure are not equal.

An analysis of the results (Table 7) shows that the SF approach does deliver more quality code, as can be seen with the QI measurement. Normally, good code is expected to be QI > 3 [1, 14]. For the measurements, different metric sets were used. In all sets, the QI of the SF approach was better. The code developed with a traditional approach delivered less quality code, because it used more complex code and had a smaller set of classes. The SF approach brought a more controlled environment, which helped with the maintainability and complexity of projects. Table 7 also shows statistics for the tested pairs. For all QI measurement sets the difference is significant at $p<0.05$. Therefore, the null hypothesis (H_{02}) is rejected in favor of the alternative hypothesis (H_{A2}). All results point to better software quality code when using the SF approach.

Table 7. Quality index measurement

	QI(0)		QI(1)		QI(2)		QI(4)		QI(10)	
	TR	SF	TR	SF	TR	SF	TR	SF	TR	SF
Project1	2.75	4.00	2.25	3.00	2.50	3.75	2.25	2.75	3.00	3.75
Project2	3.00	3.75	2.25	2.75	2.75	3.50	2.25	2.75	3.25	3.50
Project3	3.50	4.00	2.50	3.00	2.75	3.50	1.75	2.75	2.75	3.50
Project4	2.50	3.75	2.00	3.00	2.50	3.50	2.00	2.75	3.00	3.50
Project5	3.00	4.00	2.25	3.00	3.00	3.75	2.25	3.00	3.25	3.75
Project6	3.50	4.00	2.50	3.00	2.75	3.75	1.75	2.75	2.75	3.50
Project7	3.00	4.00	2.25	3.00	3.00	3.75	2.25	2.75	3.00	3.50
Project8	3.00	3.75	2.75	3.00	2.00	3.25	1.75	2.75	2.75	3.00
Project9	3.00	4.00	2.25	3.00	3.00	3.75	2.25	2.75	3.00	3.75
Project10	3.00	3.75	2.50	3.00	2.75	3.50	2.25	2.75	3.00	3.50
Project11	3.00	4.00	2.50	3.00	3.00	3.50	2.25	3.00	3.00	3.50
Project12	3.00	3.75	2.50	3.00	3.00	3.75	2.25	2.75	3.00	3.50
Project13	3.25	3.75	2.50	3.00	3.00	3.50	2.25	2.75	3.00	3.50
Project14	2.75	4.00	2.25	3.00	2.75	3.50	2.25	3.00	3.00	3.50
Project15	2.75	4.00	2.50	3.00	2.75	3.75	2.25	2.75	3.00	3.75
Project16	3.00	4.00	2.25	3.00	2.75	3.75	2.25	3.00	3.00	3.75
Mean	3.00	3.91	2.38	2.98	2.77	3.61	2.14	2.81	2.98	3.55
STDEV	0.25	0.12	0.18	0.06	0.27	0.16	0.20	0.11	0.14	0.18
p (H_0)	0.00		0.00		0.00		0.00		0.00	
df	30		30		30		30		30	
t	-12.64		-12.63		-10.93		-11.57		-9.53	

6. Discussion

6.1. Threats to validity and research limitations

By threats, we are referring to threats to internal and external validity [27]. The first threat to validity of this study is the participants' background. It was the first time that many participants developed web applications, especially in the chosen technology. The concept of the SF approach was also new to them. Second, the experimental setting alone is a threat to validity, because it was the participants' random choice to select what approach and which project they would work on. Third, when dealing with university students, it was also difficult to validate the accuracy of the provided effort data and have confidence in them to actually fill out the data for the actual spent hours correctly.

External validity refers to the approximate truth of conclusions involving generalizations within different contexts [26]. External validity threats are always present when experiments are performed with students. However, last year students have a sufficient development and programming experience, thus we can consider them as a less experienced software engineers. Consequently, we regard them as representatives of the context where we would like to generalize the achieved results. Another possible threat to external validity is that our projects were small, suggesting that their complexity and functionality may be limited when compared to large software projects. Nevertheless, replications should be performed with different subjects in different contexts to confirm or contradict the results.

Conclusion validity concerns the issues that affect the ability of drawing a correct conclusion. A definition of conclusion validity could be the degree to which conclusions we reach about relationships in our data are reasonable [26]. The conclusion validity threats were mitigated by the experiment design and by the properly selection of the population. Regarding the recruited subjects, we drew a fair sample from that population and conducted our experiment with subjects belonging to this sample. Moreover, proper tests were performed to statistically reject null hypotheses.

Readers should also interpret our results while considering the following limitations. The metrics thresholds values for quality indexes were used from [14] and these values are programming-language and design-approach dependent. For development, the Microsoft development environment and technology was used.

6.2. Theoretical and practical implications

Several theoretical and practical implications of our work can be foreseen. First, we defined a model based on the GQM approach for the evaluation of an SF approach within the context of quality and productivity. For researchers and practitioners, the evaluation model can also be applied to other research areas, like software frameworks, software product lines and the adoption of design patterns. Second, some researchers and practitioners have already proposed some of the benefits gained when using the SF approach. But only the productivity factor was evaluated and researched. In this research, we added the quality of code factor and investigated its impact. Product development managers can, through our research, gain an idea on the economic benefits of software development, because less effort is needed and there is improved quality for the products developed with the SF approach.

For the SF approach, it was known that its implementation provides benefits, such as quality and productivity. Papers in related work have shown benefits being achieved, albeit using different metrics and variables. Our empirical study on the other hand contributed to the collective knowledge of the SF approach with regard to the quality of the developed code.

6.3. Future work and conclusions

In our study we followed an empirical approach in evaluating engineering techniques to gain transferrable insights about them. As previously mentioned, this is to rarely done in our field, but is still necessary. We presented an empirical study to assess the impact of the Software Factories (SF) approach on a set of product and code quality indicators. The study was conducted in a university setting on 16 projects developed using “traditional” and SF approaches during an experiment. We specifically studied the impact of an SF approach on quality of code and productivity. The results showed that the use of the SF approach in software development has a statistically significant impact on the quality of code and productivity. In the experiment, small projects were used and a difference in quality and productivity could already be seen. One can conclude that this will also hold true for larger projects, which are greater and more complex. With the SF approach, developed projects are more maintainable and have better quality.

The achieved results can be considered relevant as we tried to minimize the gap between the university setting and industry environment. Indeed, the selected participants are not far from actual stakeholders since they were familiar with object-oriented programming and development.

However, despite the significance of the achieved results, we are going to replicate the experiment in different contexts. In particular, we plan to perform the replication with industry subjects (more experienced software developers).

Our future direction aims to investigate the impact of the SF approach on documentation, reusability, maintainability and modularity in software development.

References

1. M. Heričko et al., A method for calculating acknowledged project effort using a quality index. Informatica (Ljublj.), (2007), vol. 31, no. 4, pp. 431-436.
2. J. Greenfield, K. Short, Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, John Wiley & Sons, Indianapolis, 2004.
3. G. Lenz, C. Wienands, Practical Software Factories in .NET, Apress, 2006
4. J.S.Her et al., A framework for evaluating reusability of core asset in product line engineering, Information and Software Technology 49 (2007) 740-760.
5. D. Zubrow, G. Chastek, Measures for software product lines, Technical Notes CMU/SEI-2003-TN-031, 2003
6. P. Clements, L.M. Northrop, Software Product Lines - Practices and Patterns, Addison-Wesley, Boston, 2001.
7. H. Gomaa, Designing Software Product Lines with UML, George Mason University, Addison-Wesley, Boston, 2004.
8. Etzkorn et al., Automated reusability quality analysis of OO legacy software, Information and Software Technology 43 (2001) 295-308.

9. Chatzigeorgiou et al., An empirical study on students' ability to comprehend design patterns, Computers & Education 51 (2008) 1007–1016.
10. N. Fenton, S. L. Pfleeger, "Software Metrics: A Rigorous and Practical Approach", second edition, International Thomson Computer Press, London, UK, 1997.
11. P. Knauber et al., Quantifying Product Line Benefits, PFE-4 2001, LCNS 2290, pp. 155-163, 2002.
12. V. Basili, (1994). "The Goal Question Metric Approach" (PDF). ftp://ftp.cs.umd.edu/pub/sel/papers/gqm.pdf, Retrieved on 22.01.2011.
13. S.Ajila, R.Dumitrescu, Experimental use of code delta, code churn, and rate of change to understand software product line evolution, The Journal of Systems and Software 80 (2007) 74-91.
14. A. Živkovič, U. Goljat, M .Heričko, Improving the usability of the source code quality index with interchangeable metrics sets. *Inf. process. lett.*, Feb. 2010, vol. 110, iss. 6, p. 236-240
15. V. R. Basili , L. C. Briand , W. L. Melo, How reuse influences productivity in object-oriented systems, Communications of the ACM, v.39 n.10, p.104-116, Oct. 1996
16. S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design, IEEE Transactions on Software Engineering 20 (6) (Jun. 1994) 476–493.
17. J. Menendez, The Software Factory: Integrating CASE technologies to improve productivity, Report – Lean 96-02, July 1996, MIT
18. K. Pohl, G. Böckle, and F. van der Linden, Software Product Line Engineering: Foundations, Principles, and Techniques, 1st ed.: Springer-Verlag, 2005
19. M. Aoyama, Beyond Software Factories, concurrent-development process and an evolution of sofware process technology in Japan, Information and Software Technology 38 (1996) 133-143
20. Y. Chen, G. Gannod, J. Collofello, A software product line process simulator, Soft. Process Improve. Pract. 11(4), 385-409 (2006)
21. G. Polančič, M. Heričko, I. Rozman, An empirical examination of application frameworks success based on technology acceptance model, The Journal of Systems and Software 83 (2010) 574-584.
22. M. Morisio, D. Romano, I. Stamelos, Quality, Productivity and Learning in Framework-Based development: An Exploratory Case Study, IEEE Transactions on Software Engineering, vol.28,no.9, 2002
23. A. Mamrak, S. Sinha, A Case Study: Productivity and Quality Gains Using an Object-Oriented Framework, Soft. Pract. Exper. 29(6), 501-518 (1999)
24. V.R. Basili, L.C. Briand, W.L. Melo, A validation of object-oriented design metrics as quality indicators, IEEE Transactions on Software Engineering 22 (10) (Oct. 1996) 751–761.
25. P. Tomaszewski, L. Lundberg, The increase of productivity over time – an industrial case study, Information and Software Technology 48 (2006) 915-927
26. Shaddish,W., Cook T. & Campbell, D., Experimental and Quasi-Experimental Design for Generalized Causal Inference, Houghton Mifflin Co., 2002
27. Jedlitschka, A., Pfahl, D., Reporting guidelines for controlled experiments in software engineering, in Proc. ACM/IEEE International Symposium on Empirical Software Engineering (ISESE) 2005, IEEE Computer Society Press, pp. 95-195
28. Yoshihiro Matsumoto. "Toshiba Fuchu Software Factory," *Modern Software Engineering*, pp. 479-501, Van Nostrand Reinhold, New York (1990).

29. Yoshihiro Matsumoto. "A Software Factory, An Overall Approach to Software Production," *Software Reusability* ed. by P. Freeman, IEEE Computer Society, March 1987.
30. ISO/IEC 9126. Software Product Evaluation – Quality Characteristics and Guidelines for the User, International Organization for Standardization, Geneva, 2001.
31. J. McGregor, Qualitative SIMPLE, Journal of Object technology, vol. 7, no. 7, September-October 2008.
32. K. Schmid , M. Verlage, The Economic Impact of Product Line Adoption and Evolution, IEEE Software, v.19 n.4, p.50-57, July 2002
33. K. Schmid, An Economic Perspective on Product Line Software Development, First Workshop on Economics-Driven Software Engineering Research, 1999
34. J. Poulin, Measuring Software Reuse, Addison Wesley, 1996
35. J. Poulin, The Economics of Product Line Development, International Journal of Applied Software Technology, vol. 3, 20-34, March 1997
36. E.A Karlsson, Software Reuse: A Holistic approach, John Wiley, 1995
37. F. van der Linden, K. Schmid, E. Rommes, Software Product Lines in Action, The best industrial Practice in Product Line Engineering, Springer, 2007
38. C. Ebert, R. Dumke, Software Measurement, Springer, 2007
39. D. Galorath, M. Evans, Software sizing, estimation and risk management, Auerbach Publications, 2006
40. Lord Kelvin Quotations, http://zapatopi.net/kelvin/quotes/, Retrieved on 28.5.2012.
41. Frakes, W. B., & Kang, K. (2005). Software reuses research: Status and future. IEEE Transactions on Software Engineering, 31(7), 529–536.

Andrej Krajnc is a researcher and PhD student at the University of Maribor, Faculty of EE&CS, Institute of Informatics. He received his B.Sc. in computer science from the University of Maribor in 2006. His research work covers different aspect of Software Product Lines, Software Factories, .NET platform, object technology and software metrics. He worked in several industry projects.

Marjan Hericko is a Full Professor at the University of Maribor, Faculty of EE&CS, Institute of Informatics. He received his M.Sc. (1993) and Ph.D. (1998) in computer science from the University of Maribor. His research interests include all aspects of IS development with emphasis on metrics, software patterns, process models and modeling.

Uros Goljat is a teaching assistant at the University of Maribor. His research work covers agile software development methodologies (Scrum, XP, etc.), different aspects of object-oriented technology, internal software quality, and .NET platform. He gained his practical experiences in cooperation with industry on several projects. Uros received his Computer Science B.Sc. in 2007 from University of Maribor.

Črt Gerlec is a researcher and PhD student associated with the Faculty of Electrical Engineering and Computer Science, Institute of Informatics at the University of Maribor. His research interests are mining software repositories, software evolution, software quality, software metrics, information systems and more. He is experienced software developer on Microsoft.NET platform and expert for software architecture, design patterns and best practices.

Dr. Gregor Polančič is an assistant professor at the Institute of Informatics. He received his PhD in Computer Science from the University of Maribor in 2008. His main research interests are: (1) analysis and design of software, (2) web applications, communities, architectures and patterns, (3) FLOSS software, projects and development models, (4) business process modeling, informatization and re-engineering and (5) computer mediated communication and collaboration. Dr. Polančič has been a work co-ordinator/member of several applied projects and work co-ordinator/member in several international research projects. Dr. Polančič has appeared as an author or co-author in more than 10 peer-reviewed scientific journals. In all, his bibliography contains more than 130 records.

Received: July 20, 2011; Accepted: June 04, 2012.