UDC 004.41
Preliminary Communication

Impact of Aspect-Oriented Programming on the Quality of Novices' Programs: A Comparative Study

Marija Katić *marija.katic@fer.hr*
University of Zagreb
Faculty of Electrical Engineering and Computing, Zagreb, Croatia

Ivica Botički *ivica.boticki@fer.hr*
University of Zagreb
Faculty of Electrical Engineering and Computing, Zagreb, Croatia

Krešimir Fertalj *kresimir.fertalj@fer.hr*
University of Zagreb
Faculty of Electrical Engineering and Computing, Zagreb, Croatia

Abstract

Aspect-oriented programming has been introduced in order to increase the modularity of object-oriented programs and is claimed to improve software quality. Although there are various researches on this claim, the question to what extent aspect-oriented programming improves the quality of programs depending on a developer's experience still remains. The purpose of this study is to investigate whether aspect-oriented programming used by novice programmers improves the quality of programs, in terms of software flexibility and readability (consequently reusability and maintainability as well). As a part of an undergraduate course in programming paradigms and languages, a systematic comparison between students' object-oriented and aspect-oriented solutions of the same problem was driven. In order to drive this comparison we have established the basis for the development of the new quality assessment model consisting of software metrics for an objective evaluation and student survey for subjective evaluation. The results show that the use of aspect-oriented programming lead to novices' programs that are easier to change and read (flexible and readable) compared to object-oriented programs. What is more, administered survey showed that students perceive their programs as more flexible and readable.

Keywords: Aspect-oriented programming; Software quality assessment; Novice programmer

1. Introduction

Due to the intensive development of software for big and complex systems, the necessity for improvement of internal program quality attributes such as coupling and cohesion so improving software quality characteristics such as flexibility (how easy is to introduce changes in program modules) or readability (how easy is to read source code) becomes essential. Although it has passed four decades since the idea of modularization as a mechanism of improving program flexibility [34], we are still facing issues of decreased software quality [9]. The switch to the object-oriented programming paradigm (OOP) has improved the idea of modularization thereby improving software quality. However, it has not provided mechanisms for resolving some ubiquitous concerns, often called cross-cutting concerns [14], such as synchronization, logging or transaction management. Such concerns should be isolated into separate modules since they are orthogonal to the problem domain and for the most part outside of it.

Aspect-oriented programming paradigm (AOP) [25] has been introduced as a complement to traditional programming paradigms such as procedural and object-oriented

paradigm in order to improve the overall quality of programs through the isolation of cross-cutting concerns into separate modules. Due to its isolation mechanisms, AOP has found its place in the contemporary world among both academics and practitioners. AOP programs are divided into two parts – base code which is more purpose-specific and encompasses concerns related to the core functionality of the program, and additional code which is represented by aspects and encompasses cross-cutting concerns. As a result, program code is less tangled and crosscutting concerns are not scattered in many modules any more, making the overall code more understandable and flexible.

However, there is no uniform approach to measuring quality of aspect-oriented programs neither uniform quality assessment model which introduces confusion into the field and results in the emergence of the two main strands of researchers: some claim it brings more complexity into programs [5], [36], while others advocate its benefits [17], [39], [41]. This is probably because of the phenomena that is best described by Steimann [38] in his essay discussing the paradoxical success of AOP. On one hand AOP improves modularity and the structure of code, while on the other hand it breaks their purposes (understandability and easier development).

Furthermore, although there is research [5], [24], [29], [39], [41] using human subjects in their investigations of the claimed benefits of AOP, the importance of the human factor in such research is not sufficiently emphasized. Moreover, its insufficient consideration is a general problem of computer science as pointed out by Hanenberg [21]. Participants of the studies that used human subjects in their investigations of the claimed benefits of AOP generally differed in their background. This prevented researchers from bringing conclusions of the benefits of AOP according to specific level of developers' experience. Additionally, most of the existing studies analyzed a limited number of aspect-oriented programs [1].

This paper presents a systematic case study in which novices' programs developed by using object-oriented and aspect-oriented programming paradigms are compared. The target group were students in their last year of an undergraduate software engineering university program. They, as novice programmers, were suitable candidates for our study.

In order to explore how aspect-oriented programming affects program quality expressed in terms of flexibility and readability quality characteristics, the following research questions have been explored:

(1) Does aspect-oriented programming affect the program quality of novice programmers' programs in terms of flexibility and readability (consequently reusability and maintainability)?
(2) If there is an impact of aspect-oriented programming on the program quality characteristics of flexibility and readability (consequently reusability and maintainability), what happens with them – are they deteriorated or improved?
(3) How aspect-oriented programming impacts the attitudes of novice programmers first time faced with it with respect to the quality of their programs? Are their attitudes in conformance with the answer on question (2)?

There are two main contributions of this study: (i) the basis for the development of the new quality assessment model for the evaluation of the impact of AOP on novices' programs and (ii) the quantification of the effects of applying both aspect-oriented and object-oriented programming paradigms on software quality characteristics of novices' programs.

The paper is organized as follows: Section 2 generally describes software quality measurement. Section 3 describes the design of the study, and the proposed quality assessment model is given in Section 4. Data collection and results are presented in Section 5. Section 6 discusses possible threats to validity of our research. Finally, related work and conclusions are given in Section 7 and Section 8, respectively.

2. BACKGROUND: Software Quality Measurement

Inadequate software quality is usually the cause of problems such as hard maintenance and difficult or impossible reusability of program components or, generally, program change.

Therefore, software engineering research is partly concerned with providing mechanisms and techniques that improve software quality. In order to measure the extent to which a programming technique possibly enhances or deteriorates quality of a software product, quality in terms of measurable software characteristics needs to be defined. However, there are different ways of approaching software quality, which induced the development of different software quality models [3], [4], [7], [16], [23], [26], [30], and among them the standard for software-product quality measurements ISO 9126 which arose as a result of international efforts. In their endeavour to capture compound quality characteristics and their relationships, most of the models include a wide range of characteristics. While in early models there is no explicit differencing between internal and external quality characteristics [7], [23], [30], in newers [4] among which is Dromey's [16], the difference between structural form of programs that is associated with the low-level quality carrying properties (internal) such as modularity (coupling, cohesion) and the high-level quality attributes (external) such as maintainability, reusability, flexibility or functionality is emphasized. Moreover, Dormey established the link between the two views of quality [15].

However, object-oriented programming introduced some new concepts such as encapsulation and polymorphism which called for the definition of new models that account for these properties. The need for the extension of the Dromey's generic model was recognized by Bansiya et al. [4] who indicated vagueness of linking low and high-level quality attributes. They defined a hierarchical model for the assessment of the quality of program design before it is committed to an implementation.

Furthermore, the introduction of aspect-oriented concepts that actually extend object-oriented imposed new challenges while defining models for software quality measurement. There are quality models for AOP proposed as a part of the evaluation of the impact AOP has on reusability and maintainability quality characteristics [5], [37]. Recently aspect-oriented quality model [26] as an extension of the ISO 9126 standard has been proposed. It has introduced four new sub-characteristics: modularity, code-reusability, complexity, and reusability under the main characteristics of maintainability, efficiency, usability, and functionality respectively. However, it does not provide the way of calculation and the link between low and high levels of quality.

In engineering community it is believed that software quality is best estimated by experts, although such assessments are subjective. However, not all quality models include the human factor. Overall, the importance of automatic software quality assessment or prediction (depending on the stage in which is the software being developed) is obvious. This calls for appropriate quality models that establish the link between low and high level quality attributes and could be employed in the implementation of tools for automatic software quality assessment. Examples are the implementation of Bansiya et al. model [4] in [40] and [33] and the proposal and partial implementation of probabilistic software quality model by Bakota et al. [3].

3. Study Design

This research complements the existing empirical research regarding the impact of AOP on software quality. Study was conducted as a part of a course "Programming Paradigms and Languages", taught in the last year of software engineering university bachelor program. Course topics are divided into the three groups: imperative paradigm, object-oriented paradigm and the declarative paradigm. The students who enrolled this course had only basic knowledge of object-oriented principles and could be considered as novice programmers. The same teacher gave lectures to both groups eliminating possible differences in their teaching.

Although students were introduced with several design patterns (Singleton, Factory, Composite and Decorator) as part of the course, we were not interested in identifying design constructs, but only in the overall programs implementation.

This study only focuses on the middle part of the course, examining the outcomes of laboratory exercises that required students to solve specific engineering problems using AOP. Seventy five students (subjects in the following text) were divided into two subgroups: AOP

group (experimental group) and OOP group (control group). The separation was done by taking into account their prior academic success in the first part of the course (the imperative paradigm). There was a number of activities in that part of the course throughout which students collected points. According to the total amount of collected points, students were separated into the two groups containing equal proportions of students with similar academic ability. This allowed us to exclude the tertium quid effect of their prior academic success potentially biasing our study results. As a part of the experiment subjects were involved in three study activities: (i) lectures on AOP, (ii) implementation of programs in OOP or AOP and (iii) a web-based survey.

In the first activity only the AOP group participated. They were given a lecture and a tutorial demonstrating AOP's theoretical and technical topics. Seven days later, the second activity took place, with both groups taking part. Activity was implemented as a laboratory exercise. As a part of the second activity both groups were given the same assignment: the experimental group was required to use AOP, while the control group was supposed to use OOP only. The description of the assignment can be found in [8]. After additional seven days they were required to upload their solutions via an internal learning management system. Finally, in the third activity, which was conducted only few days after the second, both groups were required to answer the same set of items in a web-based survey.

4. Towards a Software Quality Assessment Model with Human Factor

Since, for our research, adopting an existing quality measurement model was a challenge due to their generalities or specifics, we decided to establish the basis for the much broader attempt to develop and validate the quality assessment model for aspect-oriented programs (QAM-AOP). There are several requirements the model has to satisfy in order to be adequate for our research:

(1) It should be possible to interpret high-levels of program quality (flexibility, readability) based on the measures obtained on the low level of program quality (coupling, cohesion).
(2) It should be possible to compare the quality of the object-oriented program to the one of the aspect-oriented program.
(3) It should be possible to incorporate human factor into the process of quality assessment in order to compare the subjective opinions of experts or programmers about the quality of program with the objective measures.
(4) It should be extensible and allow for the inclusion of a wider set of software metrics and quality properties, as well as the inclusion of different subjects relevant for giving program quality estimations.
(5) It should be possible to apply the model on the final program source code.

The models suggested by Bansiya et. al. [4] (completely developed and validated) as well as Bartsch et. al. [5] and Sant'Anna et. al. [37] (proposed as a part of similar research on the impact of AOP on program quality) only partially satisfy the stated requirements (Table 1).

Requirement	Bartsch [5]	Bansiya [4]	Sant'Anna [37]	QAM-AOP
1	✓	✓	✓	✓
2	✓		✓	✓
3	✓	✓		✓
4		✓		✓
5	✓		✓	✓

Table 1. Software Quality Assessment Models according to 5 requirements

Bansiya's model focuses only on object-oriented pre-implementation designs, while Bartsch's and Sant'Anna's are specifically constructed for their research. Bartsch's model is used to evaluate the impact of AOP only on maintenance tasks and also includes human

factor, while Sant'Anna's model is build and refined using Basili's GOM methodology [6] and evaluated in the context of two empirical studies.

Building upon these three models we propose the basis for the development of a generic model for the evaluation of the benefits or difficulties AOP introduces into the program quality. In further text we refer to this initial model proposal as QAM-AOP. Actually, QAM-AOP model is constructed by combining Bartsch, Bansiya and Sant'Anna model parts accompanied with the introduction of a survey for the comparison of programmers' subjective opinions about program quality with the objective quality measures.

The QAM-AOP (Fig 1) consists of quality attributes for the measurement of the low level of quality (program design and code) and quality characteristics for the evaluation of the higher level of quality. Two measurement techniques used are: (1) software metrics to measure program design and code and their results are aggregated to represent measures of quality attributes, and (2) survey to evaluate subjective opinions of subjects involved in programs development and their results are aggregated to represent measures of quality characteristics.

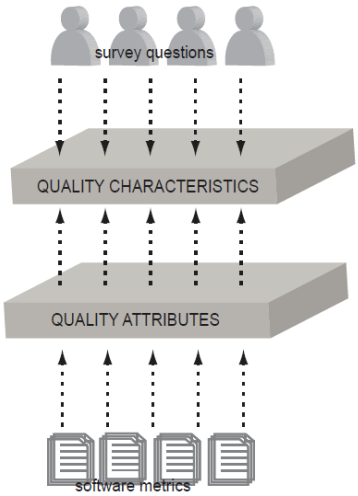

Figure 1. Quality Assessment Model (QAM-AOP)

The given structure of the model ensures that requirements imposed on the model definition can be satisfied. Moreover, QAM-AOP allows us to evaluate the source code quality of novices' programs as well as novices' beliefs regarding the benefits of AOP on program quality.

4.1. QAM-AOP Components

Before presenting model components descriptions (in the rest of this section), we would like to discuss their origins. Quality attributes (Section 4.2) used in this research are the subset of attributes given in Bansiya's and Sant'Anna's models. The similar set of quality characteristics (Section 4.3) the QAM-AOP examines is examined in the Sant'Anna's model as well (the only difference is that the QAM-AOP focuses on readability, while Sant'Anna emphasizes understandability). Bansiya's model apart from other characteristics also allows for the examination of the same set of characteristics as QAM-AOP, while the only characteristic Bartsch's model explores is maintainability.

In order to establish the relationship between attributes and characteristics and finally evaluate quality characteristics, *software metrics* (Section 4.4) and *survey* (Section 4.5) are used. QAM-AOP replicates relationships between software metrics and quality attributes given in the Sant'Anna's model and other similar research on quality attributes (Section 4.4 and Section 4.6.1). Additionally, QAM-AOP contains software metric to measure program complexity from the AOP point of view. This metric is called "aspect cross-cutting degree" (Table 2) and is discussed throughout the paper. QAM-AOP also replicates the relationship

between quality attributes and quality characteristics given in the Sant'Anna's model (Section 4.6.2).

Since none of existing quality models contain an adequate list of *survey* items for subjects' opinions evaluations, QAM-AOP introduces survey items employed in our other broader study [8]. By doing that, a new relationship between survey items and quality characteristics is introduced (Section 4.6.2).

Although QAM-AOP consists of a small set of components taken from several other studies, it represents the basis for the development of a generic model for measuring any set of quality characteristics (i.e. effectiveness, robustness ...) of aspect-oriented programs.

4.2. Identifying Quality Attributes

Quality attributes refer to the whole program implementation and not only the pre-implementation structures. There are no unique definitions for quality attributes so the definitions from the literature that seemed to be the most relevant for our research were chosen, keeping in mind the claimed benefits of AOP. Since it is claimed to improve the separation of crosscutting concerns increasing module cohesion, decreasing coupling between modules and consequently reducing the total amount of source code lines [27], [39], the fundamental modularity attributes of coupling and cohesion are examined. Additionally, code size is examined as well.

The first formal definition of coupling is given by Chidamber and Kemerer [13] who defined coupling between classes as "...any evidence of a method of one object using methods or instance variables of another object ...". It actually refers to the degree of interdependence between program parts potentially preventing modification of program parts independently of their coupled parts. Minimizing it is worthwhile not only during development of a program, but also in later stages of software development process in order to make maintenance tasks less tedious and less costly.

Cohesion is all about ensuring that each program component does exactly one thing. The original definition is given by Chidamber and Kemerer as well [13]. They proposed "Lack of Cohesion in Methods" metric which is calculated by counting the number of method pairs operating on disjoint sets of instance variables and subtracting it by the number of method pairs acting on at least one shared instance variable. Since in a highly cohesive system related functionalities get grouped together, the program should possess low coupling and high cohesion attributes.

According to [18] software size is described in terms of length, functionality and complexity. With respect to the length there are three development products whose size is calculated: specification, design and code. Since AOP is claimed to significantly affect the length of code by eliminating scattering and tangling in the code [25], this paper examines only the length of code as the physical size of the product expressed in terms of the number of lines of code (LOC). In the further text, when it is not stated differently, size actually means the length of code.

4.3. Identifying Quality Characteristics

Quality characteristics or external quality attributes are abstract concepts and therefore cannot be measured directly. However, we have tried to incorporate them into the model in such a way that promotion of measures from low level of quality to the higher level can provide a way for their automatic interpretation. QAM-AOP leaves a lot of freedom for the characteristics selection. Considering external quality characteristics evaluated in Bartsch's [5] and Sant'Anna's [37] models as well as selected quality attributes (Section 4.2) we limited our selection of characteristics only on flexibility and readability. Evaluations of flexibility and readability characteristics are used for promoting maintainability and reusability since they encompass common cognitive tasks [37]. In the further text we introduce definitions of quality characteristics.

Flexibility, as an expected benefit of modular programming, is the ease of making changes to one module without the need of changing other modules [30], [34]. Because of better support for separation of concerns, it is expected from AOP programs to be more modular compared to OOP programs.

Readability refers to the ease with which source code of a system can be read and understood especially at the detailed-statement level [31]. It can be considered as a sub-characteristic of understandability characteristic. Understandability, also mentioned in Bartsch 's and Sant'Anna quality models [5], [23], [37], generally, is about how easy it is to comprehend a system at the both system-organizational and detailed-statement levels [31]. We are interested only in information about students' perception of how easy it is for them to read and understand their source code which emphasises only the relevance of readability characteristic for this research.

Reusability is the ease of reusing software in different contexts or even using parts of a system (including requirements, designs, documentation and code) in other systems [30], [31]. It actually means how flexible and readable the program is [37].

Maintainability includes most difficult and costly tasks of software engineering [32]. It means how easy it is to change or add software features or to correct defects. It again means how flexible and readable the program is [37].

4.4. Identifying Software Metrics

Software metrics are a way of measuring the degree to which quality attributes are presented in source code [18]. However, there is no unique set of metrics that can be applied to all programming paradigms. Our main goal was to find reasonably small set of metrics that is sufficient for the determination of impacts AOP has on novices' programs through corresponding quality attributes. Metric set proposed by Chidamber and Kemerer [13], which has made the greatest impact on the OOP program measurement seemed the most appropriate for our research because it is applicable on the final programs (implementation is finished).

QA	Metric name	Metric description
COUPLING	Aspect cross-cutting degree (ACD)	Applicable only to AOP programs. It is the number of all modules affected by an aspect or the number of aspects that affect measured module. This metric describes the overall impact of an aspect on other modules. High value is desirable because it means high reusability of an aspect.
	Coupling between modules (CBM) ↓	Number of other modules declaring methods or properties used in a measured module.
	Number of children (NOC) ↑	Number of derived modules.
	Response for a module (RFM) ↓	Number of all its methods and all methods invoked from them
	Depth of inheritance tree (DIT) ↓	Number of its base modules in its inheritance hierarchy.
COHESION	Lack of cohesion in operations (LCO) ↑	Degree of similarity between methods of the same module.
SIZE	Lines of code (LOC) ↓	The logical number of code lines.

Table 2. Software metrics and corresponding quality attributes [11], [12], [19], [37]

However, in order to measure the degree of coupling this set was not sufficient. The issue of inappropriateness of existing metrics to measure AOP programs due to specifics of AOP mechanisms was recognized by Ceccato et al. [12] who proposed a set of aspect-oriented metrics. We have chosen eight metrics that best reflect the presence of selected quality attributes (Table 2). These metrics have already been used in several previous studies [11], [19], [37], and showed as successful indicators of quality attributes.

Table 2 presents the set of software metrics used in previous studies [11], [12], [19], [37], so we omitted their detailed descriptions.

4.5. Identifying Applicable Human Factors

In order to satisfy the human factor requirement in the QAM-AOP model, a set of questions (survey) for subjects was included. The main goal of the survey is to examine subjects' perception of AOP effects on software quality characteristics. Survey items are created with respect to the definitions of quality characteristics to be rated subjectively on an ordinal scale (1-5) [28], from "disagreeing strongly" to "agreeing strongly". The list of relevant survey items is given in the Table 3.

Item	Item text
I1	The use of AOP increases the time needed to complete the second laboratory exercise.
I2	Without AOP it takes more time to complete the second laboratory exercise.
I3	The use of AOP eases the completion of the second laboratory exercise.
I4	The use of AOP makes my program from the second laboratory exercises easier to extend.
I5	Without the use of AOP in the second laboratory exercise, it is more difficult to extend the application according to the specification for the third laboratory exercise.
I6	The use of AOP makes my program from the second laboratory exercise more reusable.
I7	The use of AOP makes my program from the second laboratory exercise easier to read.
I8	The use of AOP reduces the number of lines in program from the second laboratory exercise significantly.
I9	The use of AOP reduces the interdependence of program components (classes, aspects) and the implementation details of other components.
I10	The use of AOP in the second laboratory exercise makes grouping relevant functions into separate modules (classes, aspects) easier.

Table 3. Survey items [8]

4.6. Towards Relationship between QAM-AOP Levels

Quality attributes are measured with appropriate software metrics (Table 2). However, since quality characteristics cannot be measured directly, the model provides mappings between measurement techniques at both levels of quality, low and high, thus allowing the promotion of measurement results of software metrics to the higher quality characteristic level (Table 4). Additionally, the model includes survey items exploring participants' perception of the new technology effects on quality characteristics (Table 4).

4.6.1. Software Metrics and Quality Attributes

Table 2 illustrates the impact of software metrics on quality attributes. An up arrow symbol (↑) denotes that a higher metric value has a positive impact on a quality attribute, while a down arrow symbol (↓) denotes that a smaller metric value has a positive impact on a quality attribute. There are no arrow symbols near the ACD metric because this metric is valid only for AOP programs and is used for an additional coupling calculation. It is needed because existing research calculates coupling mostly without taking into account the number of aspects used by the measured module [36]. Having both coupling measures calculated, with (CBM+ACD) and without (only CBM) aspects allows us to come up with additional comparisons in examining AOP programs (Section 5.2 and Section 5.3). Since coupling metrics are most often used when measuring maintainability [10], according to the literature review we expect that coupling in AOP programs decreases compared to OOP programs.

4.6.2. Quality Characteristics, Survey Items and Quality Attributes

Table 4 presents the relationship between quality attributes and quality characteristics. When there is a relationship, the appropriate cell is marked with an "X". Furthermore, the table presents survey items examining quality characteristics. If an item is intended to examine a subject's perception and belief regarding the impact of AOP on a quality characteristic, the appropriate cell is marked with an "X". All quality characteristics should be presented in a program: an up arrow symbol (↑) denotes that higher value of source code quality attribute or a survey item answer has a positive impact on a quality characteristic, while a down arrow symbol (↓) denotes that smaller value of a quality attribute has a positive impact on a quality characteristic. Maintainability and reusability are excluded from Table 4 since both flexibility and readability are used for their evaluation as it is stated in Section 4.3.

Qual. ch.	Survey items ↑											Quality attributes		
	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10		↓Coupling	↑Cohesion	↓Size
Flexibility	X	X	X	X	X	X			X	X		X	X	
Readability	X	X	X				X	X				X	X	X

Table 4. The relationship between quality characteristics, survey items and quality attributes

5. Data Collection and Results

Since there was no tool for code analysis and metric calculation for programming languages used in our research (C# and its AOP extension PostSharp [35]), we decided to developed it from scratch [8]. Furthermore, in order to collect information about students' perception of the impact of AOP on their programs, a web-based survey was administered with the use of Google Docs system [20]. Google Docs Likert Scale items were labeled with 1 for "strongly disagree" and 5 for "strongly agree". Students from both the experimental and control groups participated in the survey.

5.1. Analysis of Collected Software Metrics Data

Table 5 presents results of the independent sample t-test comparing the OOP and AOP groups according to the calculated metrics results. Results of Levene's test for the homogeneity of variance and the independent samples t-test show there is a significant difference between groups according to three metrics. For LOC there is a mean difference of MD=160.86 which is significant $t(73)=-2.61$, $p<0.05$ 2-tailed, effect size $r=.29$ [8]. Both CBM and RFM pass the 2-tailed significance with the mean differences between groups of MD= 6.953, $t(73)=-0.68$, $p<0.05$, $r=.19$ and MD=24.740, $t(73)=-0.90$, $p<0.05$, $r=.10$ respectively. Therefore, there is a medium-sized effect for LOC and CBM and a small-sized effect for RFM. Other metrics do have notable mean differences but do not produce significant results when group differences

are concerned. An interesting thing to note is that the experimental group achieves better results according to all metrics.

Metrics	Sig. (2-tailed) *p<0.05	Mean Difference
NOC	0,736	-0,188
CBM	0,091*	6,953
RFM	0,097*	24,740
DIT	0,564	-0,336
LOC	0,011*	-160,864
LCO	0,508	33,087
WOM	0,846	-2,124

Table 5. Metric results for experimental and test group [8]

5.2. Analysis of Interdependence between Software Metrics and Quality Attributes

Table 6 shows the results of the quality attributes comparison for the two groups: the experimental and the control group. In order to compare the quality attributes we summed up normalized software metrics results and came up with quantitative measures that can be compared. We compare mean values of these calculated values by giving percentage differences between the means of each quality attribute. Generally, lower values of coupling and size, and higher values of cohesion imply a better result. Therefore a positive percentage difference indicates that OOP implementation exhibits better results, while a negative value is in favour of AOP. It is the opposite in the case of the cohesion attribute.

In our analysis we approach the calculation of the coupling quality attribute from two different angles: the first one is with aspects taken into account (ACD is added to CBM) which results in coupling increased in AOP programs by about 28%. In second calculation, without the ACD metric, coupling in AOP programs decreases by about 11%. These conclusions are in accordance with those from the literature [36]. Furthermore, size decrease in favour of AOP by around 5% and 15% respectively. And finally, the cohesion increases by about 17% in favour of AOP.

Quality attributes	Group	N	Mean	Std. Error Mean	% Diff.
Coupling (1st calc.) (NOC+ACD+CBM+RFM+DIT)	AOP	43	1.117	0.117	27.73
	OOP	32	0.845	0.126	
Coupling (2nd calc.) (NOC+CBM+RFM+DIT)	AOP	43	0.754	0.1	-11.38
	OOP	32	0.845	0.126	
Cohesion (LCO)	AOP	43	0.151	0.027	17.27
	OOP	32	0.127	0.021	
Size (LOC)	AOP	43	0.692	0.051	15.34
	OOP	32	0.807	0.054	

Table 6. Results of quality attributes comparison for the experimental and control group

5.3. Analysis of Relationship between Quality Attributes and Quality Characteristics

Quality attributes are aggregated into the quality characteristics and the comparison between AOP and OOP group is given. However, it is worth noting that these two quality characteristics are not independent. When observing how set of quality attributes affect the certain quality characteristic (Table 4), there are two different kinds of comparisons that can

be done (Table 7). The main reason for doing the two comparisons is coupling which, as a quality attribute, can be calculated in two different ways (Table 6). Generally, lower mean values of all quality characteristics imply a better result. Therefore a positive percentage value indicates that OOP implementation exhibits better results, while a negative value is in favour of AOP.

As expected according to the literature review, in the first comparison (when ACD for coupling is counted), OOP programs give better results for all quality characteristics. In the second comparison (when ACD is not counted) AOP programs give better result. Readability has about 10% better values in favour of OOP in the first comparison and about 11% in favour of AOP in the second comparison. Finally, flexibility has better result for about 26% in favour of OOP in the first comparison and only for about 7% in favour of AOP in the second comparison.

Quality characteristics	Group	N	Mean	Std. Error Mean	Diff. in %
Readability* (Coupling+Cohesion+Size) (Coupling =NOC+ACD+CBM+RFM+DIT)	AOP	43	1,961	0,173	9,68
	OOP	32	1,780	0,180	
Readability ** (Coupling+Cohesion+Size) (Coupling =NOC+CBM+RFM+DIT)	AOP	43	1,598	0,156	-10,78
	OOP	32	1,780	0,180	
Flexibility * (Coupling+Cohesion) (Coupling =NOC+ACD+CBM+RFM+DIT)	AOP	43	1,268	0,130	26,43
	OOP	32	0,972	0,136	
Flexibility ** (Coupling+Cohesion) (Coupling =NOC+CBM+RFM+DIT)	AOP	43	0,906	0,114	-7,03
	OOP	32	0,972	0,136	

Table 7. Quality attributes aggregated into the quality characteristics and compared across the groups

5.4. Analysis of Relationship between Quality Characteristics and Survey Results

Relationship between quality characteristics and survey items (Table 4) is important for the interpretation of students' opinion about the effects of AOP on the quality characteristics of the software they produced.

Item	Overall mean	Sig.(2-tailed)	Mean Difference
I1	1,73	0,014*	-0,613
I2	4,12	0,056	0,514
I3	4,29	0,023*	0,552
I4	4,19	0,097*	0,378
I5	3,55	0,416	-0,200
I6	4,23	0,419	0,183
I7	4,38	0,694	0,080
I8	4,41	0,867	-0,033
I9	3,90	0,039*	0,409
I10	3,87	0,011*	0,524

Table 8. The analysis results of the administered survey [8]

Table 8 presents results of the independent sample t-test comparing the OOP and AOP groups according to student responses. Their responses are mostly in favour of AOP. They think it helped them in solving their laboratory exercises and that it saved time needed to develop the programs. What is more, the significance test shows there is a significant difference in opinion regarding the produced programs between aspect-oriented and object-oriented groups in the terms of flexibility and readability, and additionally the time to complete the laboratory exercise (I1, I3, I4, I9 and I10). When the interdependence of components and modularity are concerned (coupling, cohesion), students agree less, but nevertheless are in favour of AOP [8].

6. Validity Evaluation

Based on the classification scheme for different types of threats to validity of an experiment [42] this section discusses the possible threats to validity. Although we did our best to minimize threats to internal, external, conclusion and construct validity of the study there are some threats identified.

Possible threats to internal validity are the size and the complexity of analyzed programs and the tasks students were given to recognize and implement the separation of concerns. However, more complex tasks could also be threat to internal validity because the effects on beginner's programs are investigated.

A possible threat to external validity is the time when a control group was presented with an AOP lecture. That occurred three days before the deadline for the task upload. However, analysis revealed that a few students circumvented the rules and used AOP concepts in their implementations. We treated them as a part of the experimental group.

Threats to conclusion validity are generalizations of findings from a specific technology (PostSharp) to AOP. Another threat to conclusion validity is reliability of the metric suite we have used. We minimized that risk because the same metric suite has successfully been used in other research [2], [37]. Furthermore, the QAM-AOP model proposal is not evaluated. However, this model relies on the three existing models [4], [5], [37] and consists of several components evaluated as a part of other models. The number of participants is also a threat. The sample size of 75 students means that the results, although significant, come with a medium effect size. With an alpha-error of 0.05, the required sample to reproduce these small to medium sized effect with power of 80% is 132. Therefore, the reproduction of the study (with power of 80%) comes with a relatively high price coming from a need for a big sample.

7. Related Work

There are studies that also used human subjects in their comparisons of AOP and OOP programs [5], [29], [39], [41]. Tonella and Ceccato [39] compared OOP and their refactored counterpart AOP implementations of Java Standard Library and three Java applications with a focus on external quality attributes (maintainability and understandability), and internal quality attributes (size and modularity). Twelve academics and programmers were involved in the experiment. They found a significant increase in class cohesion and significant decrease of the class coupling in the refactored implementations, denoting improved modularity. Overall, their study indicated that the migration to AOP produced code which is easier to understand and maintain. However, no significant size reduction was recorded.

Walker et al. [41] conducted two experiments in order to investigate the effects of AOP on program modularization. As a parameter they took programers' performance and experience when working on two programming tasks. While in the first experiment they studied whether AOP enhances a developer's ability to find and fix faults of a multithreaded program, the main concern of the second experiment was the ease of changing an existing distributed system. The sessions during which both AOP and OOP groups worked on the tasks were videotaped. Based on session analysis they concluded that in the first experiment AOP group finished the tasks faster than OOP group, while in the second experiment they

needed more time. The main conclusion is that AOP programs are easier to understand when the scope of aspect-code is well-defined.

Madeyski and Szala [29] conducted an empirical study of a web-based manuscript submission and review system to examine differences in software development efficiency and design quality between AOP and OOP approaches. However, only three experienced programmers participated in the study. While two of them developed the system using only OOP, the third one with a year of AOP experience used AspectJ for the development of the system. Due to the limited number of subjects they could not generalize the impact of AOP on software development efficiency and software design quality. Statistical tests showed that AOP approach did not significantly affect class-level software quality metrics such as WOM, CBM, RFM and LCO.

Bartsch and Harrison [5] conducted an experiment to investigate the effect of AOP on understandability and modifiability. Eleven software professionals with no prior knowledge of AOP were confronted with five AOP tutorials. Afterwards they were asked to implement given maintenance tasks on either OOP or AOP program and then to answer a questionnaire about their experience on given tasks. Although the results did not show a statistically significant impact of aspect-oriented programming for 2-tailed significance levels of 5% or 10%, they indicated a slight advantage for the subjects using the OOP system. The main reason is the time that subjects needed to answer the questions, which was slightly shorter in case of OOP. However, they did not find a difference in understandability rating of the two systems as well as any evidence of an effect of AOP on the modifiability of the software system.

Studies that compared real-world systems of different sizes from different domains [19], [36] are also worth mentioning. Przybylek [36] compared AOP and OOP programs with regard to coupling, cohesion and size. Ten programs originally implemented in Java and afterwards refactored with AspectJ were compared. They found that the average coupling between modules is significantly higher in most of the refactored versions. The main reason for that is that not only direct references between modules such as inheritance and composition are taken into account, but also AOP-specific dependencies that are dependencies when aspects are used by a given module. With regard to cohesion, they concluded that for the LCO metric the impact of AOP remained unclear. Furthermore, the refactored versions were larger in half of the cases.

Garcia et al. [19] investigated whether aspect-oriented approaches support improved modularization of crosscutting concerns relative to design patterns. They replicated an existing software engineering experiment of Hannemann and Kiczales [22] and used the same two Java and AspectJ implementations of 23 Gang-of-Four design patterns for a comparison in their quantitative study. However, they performed additional analysis after the introduction of changes to the existing implementations. The results showed that although aspect-oriented implementations provided better separation of concerns, some patterns showed increased coupling, complexity and size in their AOP solutions. They indicated that general conclusions could not be drawn due to the limited scope of their experience.

8. Conclusions and Future Work

In this comparative study novices' programs developed by using AOP and OOP techniques have been compared according to the proposed QAM-AOP model. Seventy five subjects with the same background participated in the study.

There are two main contributions of the research reported in this paper. Firstly, QAM-AOP model as a basis for the measurement of quality characteristics of flexibility and readability (reusability and maintainability) is proposed. The model reflects the effects of AOP on novices' programs by examining only software quality characteristics that are affected by AOP through the low-level quality attributes of coupling, cohesion and size. This model can be used in similar research where subjects do not necessarily have to be from academia. However, the most important contributions of this study are the analysis and

quantifications of the effects of AOP and OOP programming paradigms on software quality characteristics.

We have made an attempt to answer three research questions. Based upon tested lower-level quality attributes we came to the following conclusions. AOP positively affects cohesion and size attributes which is proved by the analysis of the relationship between software metrics and quality attributes. Coupling decreases slightly when aspects get excluded from metrics calculation. This brings us to the conclusion that AOP improves coupling only when aspects are excluded from metrics calculation. With respect to size and coupling, the most important metrics for such conclusions are LOC and CBM respectively, because statistical tests showed medium-sized effect for them. The analysis of the relationship between quality attributes and quality characteristics showed there are differences between experimental and control group for studied quality characteristics that are in favour of AOP when the ACD metric is excluded from the analysis. The analysis of the relationship between quality characteristics and survey results showed significant difference in favour of AOP for studied quality characteristics.

Therefore we can conclude that AOP does affect flexibility and readability (consequently reusability and maintainability) of novices' programs so that AOP programs are easier to change, reuse, read and maintain only when ACD metric is excluded from the analysis (answers to research questions (1) and (2)). Furthermore, since experimental group's responses are mostly positively inclined towards AOP, it shows us students attitudes are mostly in conformance with the software metric results when ACD metric is excluded from the analysis which means they mostly advocate AOP benefits (answer to the research question (3)). However, as it is discussed in validity evaluation chapter this might be the case because their tasks are not complex enough to notice the other AOP side (additional complexity introduction).

This research eliminates the main limitation of the existing research which is a limited number of subjects with the same background as well as the number of analyzed AOP and OOP programs per study [19], [29]. It also provides evidence which could help in resolving the confusion coming from the two strands of aspect-oriented researchers. This is because it confirms that AOP programs are easier to change and understand and have reduced coupling and increased cohesion [39], [41]. Furthermore, by performing an additional analysis (inclusion or exclusion of ACD metric into the analysis) it brings conclusions that are in favour of OOP [5], [36] .

Eventually, it is important to note that this research offers limited opportunities for generalizations. The scope of this research is limited to the specific technology (PostSharp), specific complexity of the task and the number of subjects involved. Nevertheless, the goal was to introduce some evidence for a more general discussion and we see it as an initial step towards formalizing the discourse of the impact of AOP on the quality of novices' programs.

In our future work we intend to extend the size and the complexity of analyzed programs and to replicate research in industry environment with the more experienced programmers. After the final establishment and validation of the QAM-AOP model, we would like to explore the effects of human factor when experts in the field are included in the quality estimation.

Acknowledgements

This research is founded by the Ministry of Science, Education and Sport, Republic of Croatia, under the project grants 036-0361983-2019 and 036-0361983-2022. The authors would like to thank Sharpcrafters, the company producing post-compilation AOP framework PostSharp, for providing a free academic license of the library to be used by the students and the authors.

References

[1] Ali, M.S; et al. A systematic review of comparative evidence of aspect-oriented programming. *Information and Software Technology*, 52(9):871-887, 2010.

[2] Alshayeb, M. Empirical investigation of refactoring effect on software quality. *Information and Software Technology*, 51(9):1319-1326, 2009.

[3] Bakota, T; et al. A probabilistic software quality model. In *Proceedings of the Twenty-Seventh International Conference on Software Maintenance*, pages 243-252, Williamsburg, VA, USA, 2011.

[4] Bansiya, J; Davis, C.G. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*, 28(1):4-17, 2002.

[5] Bartsch, M; Harrison, R. An exploratory study of the effect of aspect-oriented programming on maintainability. *Software Quality Journal*, 16(1):23-44, 2008.

[6] Basili, V; et al. The goal question metric approach. *Encyclopedia of Software Engineering*, 2:528-532, 1994.

[7] Boehm, B.W; et al. *Characteristics of Software Quality*. American Elsevier, New York, 1978.

[8] Botički, I; et al. Exploring educational benefits of introducing aspect-oriented programming into a programming course. *IEEE Transactions on Education*, forthcoming.

[9] Broy, M; Denert E, editors. *Software Pioneers: Contributions to Software Engineering*. Springer Berlin Heidelberg, 2002.

[10] Burrows, R; et al. Coupling metrics for aspect-oriented programming: a systematic review of maintainability studies. *Communications in Computer and Information Science*, 69(2):277-290, 2010.

[11] Cacho, N; et al. Composing design patterns: a scalability study of aspect-oriented programming. In *Proceedings of the Fifth International Conference on Aspect-Oriented Software Development*, pages 109-121, Bonn, Germany, 2006.

[12] Ceccato, M; Tonella, P. Measuring the effects of software aspectization. In *Proceedings of the First Workshop on Aspect Reverse Engineering*, Delft, The Netherlands, 2004.

[13] Chidamber, S.R; Kemerer, C.F. A metric suite for object-oriented design. *IEEE Transactions on Software Engineering*, 20(6):476-493, 1994.

[14] Dijkstra, E.W. *Selected Writings on Computing: A Personal Perspective (Monographs in Computer Science)*. Springer New York, 1982.

[15] Dromey, R.G. Cornering the chimera. *IEEE Software*, 13(1):33-43, 1996.

[16] Dromey, R.G. A model for software product quality. *IEEE Transactions on Software Engineering*, 21(2):146-162, 1995.

[17] Endrikat, S; Hanenberg, S. Is aspect-oriented programming a rewarding investment into future code changes? A socio-technical study on development and maintenance time. In *Proceedings of the Nineteenth IEEE International Conference on Program Comprehension*, pages 51-60, Kingston, ON, Canada, 2011.

[18] Fenton, N.E; Pfleeger, S.L. *Software Metrics: A Rigorous and Practical Approach*. Course Technology, Boston, MA, USA, 1998.

[19] Garcia, A; et al. Modularizing design patterns with aspects: a quantitative study. In *Proceedings of the Fourth International Conference on Aspect-Oriented Software Development*, pages 3-14, Chicago, Illinois, 2005.

[20] Google Docs. https://docs.google.com/support/bin/answer.py?hl=en&answer=139706&topic=1360904, downloaded: October, 1st 2011.

[21] Hanenberg, S. Faith, hope, and love: an essay on software science's neglect of human factors. *Acm Sigplan Notices*, 45(10):933-946, 2010.

[22] Hannemann, J; Kiczales, G. Design pattern implementation in Java and AspectJ. In *Proceedings of the Seventeenth ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 161-173, Seattle, WA, USA, 2002.

[23] International Organization for Standardization. Software engineering - product quality - part 1: quality model. Technical Report ISO/IEC 9126-1, International Organization for Standardization, Geneva, Switzerland, 2001.

[24] Josupeit-Walter, M; et al. Preliminary results of an experiment repetition for measuring the impact of aspect-oriented programming on development time. In *Proceedings of Empirical Evaluation of Software Composition Techniques Workshop*, Saint-Malo, France, 2010.

[25] Kiczales, G; et al. Aspect-oriented programming. In *Proceedings of the Eleventh European Conference on Object-Oriented Programming*, pages 220-242, Jyväskylä, Finland, 1997.

[26] Kumar, A; et al. A quantitative evaluation of aspect-oriented software quality model. *ACM SIGSOFT Software Engineering Notes*, 34(5):1-9, 2009.

[27] Laddad, R. Aspect-oriented programming will improve quality. *IEEE Software*, 20(6):90-91, 2003.

[28] Likert, R. A technique for the measurement of attitude. *Archives of Psychology*, 22(140):1-55, 1932.

[29] Madeyski, L; Szala, L. Impact of aspe ct-oriented programming on s oftware development efficiency and design quality: an empirical study. *Iet Software*, 1(5):180-187, 2007.

[30] McCall, J.A; et al. Factors in software quality. Technical Report RADC-TR-77-369, Volume 1, Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Base, New York, The National Technical Information Service, 1977.

[31] McConnell, S. *Code Complete*. Microsoft Press, Redmond, WA, USA 2004.

[32] Mens, T; Demeyer S, editors. *Software Evolution*. Springer Berlin Heidelberg, 2008.

[33] Osbeck, J; et al. Investigation of automatic prediction of software quality. In *Proceedings of Annual Meeting of the North American Fuzzy Information Processing Society*, pages 1-6, El Paso, Texas, 2011.

[34] Parnas, D.L. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053-1058, 1972.

[35] PostSharp. http://www.sharpcrafters.com/, downloaded: October, 1st 2011.

[36] Przybyłek, A. Where the truth lies: AOP and its impact on software modularity. In *Proceedings of the Fourteenth International Conference on F undamental Approaches to Software Engineering*, pages 447-461, Strarbrüken, Germany, 2011.

[37] Sant'Anna, C; et al. On the reuse and maintenance of aspect-oriented software: an assessment framework. In *Proceedings of the Seventeenth Brazilian Symposium on Software Engineering*, pages 19-34, Natal - RN, Brazil, 2003.

[38] Steimann, F. The paradoxical success of a spect-oriented programming. *ACM SIGPLAN Notices*, 41(10):481-497, 2006.

[39] Tonella, P; Ceccato, M. Refactoring the aspectizable interfaces: an empirical assessment. *IEEE Transactions on Software Engineering*, 31(10):819-832, 2005.

[40] Virani, S.S; et al. Software quality management tool for engineering managers. In *Proceedings of Industrial Engineering Research Conference*, pages 1401-1406, Vancouver, Canada, 2008.

[41] Walker, R.J; et al. An initial assessment of aspect-oriented programming. In *Proceedings of the Twenty-First International Conference on Software Engineering*, pages 120-130, Los Angeles, CA, USA, 1999.

[42] Wohlin, C; et al. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.