JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 25, 1067-1085 (2009)

Improving Object-Oriented Frameworks by Considering the Characteristics of Constituent Elements

GREGOR POLANČIČ, ROMANA VAJDE HORVAT AND IVAN ROZMAN
Department of Electrical Engineering and Computer Science
University of Maribor
Smetanova 17, 2000 Maribor, Slovenia

Advances in object-oriented frameworks (usually abbreviated as "OOF" or simply "frameworks") are currently regarded as one of the most promising areas in software development. However, many OOFs and related projects fail. To bypass known OOF related problems, a novel approach for the systematic improvement of OOFs will be introduced in this article. The proposed approach is based on the technology acceptance model (TAM) and the "divide and conquer" principle, which posits that a complex problem is easier to manage if it is broken down into simpler problems. The fundamental idea behind the research can be expressed with the assumption that elements which constitute OOF can influence the user's perceptions via the most important OOF characteristics – understandability, adaptability and confidence. The original outcome of our research is a conceptual OOF model and an OOF improvement process which can be used by framework developers as well as framework users. Several practical and theoretical implications of this work can be foreseen. Practitioners might use the outcomes of this research to develop more successful frameworks and for OOF evaluation purposes. From a theoretical viewpoint, this research can be used as a foundation for evaluating the implications of OOF-related guidelines and design approaches. While our research was mainly based on existing literature and common theories, we are aware of its limitations. Because of this, we plan to continue our research in several directions. Our current research is directed at empirically validating the conceptual OOF model that is presented in this article and at validating the proposed approach in an actual setting.

Keywords: object-oriented framework, TAM, design pattern, guideline, causal model, improvement process, divide and conquer, software quality

1. INTRODUCTION

Object-oriented frameworks (OOF) are a proven technology for reusing software designs and implementations in order to reduce costs and improve the quality of developed software [1, 2]. According to most cited definitions [3] the term OOF is described as "a set of classes that embodies an abstract design for a solution to a family of related problems." OOFs are incomplete systems that contain certain fixed aspects common to all applications in the problem domain, and certain variable aspects unique to each application made from it (also named framework instances) [4].

What makes OOFs considerably different from other reuse techniques, such as components, libraries or design patterns, is that OOFs aim to reuse larger-grained components and also higher-level designs. Additionally, OOFs define the flow of control and are therefore the main program of instantiated software [2].

Received September 10, 2007; revised March 4, 2008; accepted May 22, 2008.
Communicated by Jonathan Lee.

OOFs are becoming increasingly important to the software development community [5], especially for instantiating software within product lines and product families [6, 7]. Besides this, OOFs can also act as extension of generic programming languages, allowing developers to make gains from commonalities in the domain they perform (using domain frameworks), development practices they use (using support frameworks) or applications they develop (using application frameworks). Therefore, almost any object-oriented project uses at least one OOF [8] and, as Fontoura wrote: [9] "there are projections that OOFs will be at the core of the technology of the twenty-first century".

Besides the positive effects, OOF development and OOF instantiation continues to be a difficult endeavor [4, 10, 11]. Consequently, software developers may not decide to develop or use OOF despite their availability. Or they might develop or use OOF in an inappropriate way, which often leads to project failures. Because there are problems that make OOF development and instantiation difficult, practitioners and researchers have proposed several improvements to OOFs, ranging from documentation improvements [12], technical improvements [11] and general guidelines for OOF developments [13].

While these improvements stimulate new ideas in OOFs, most of them are based on personal experiences. They also do not include proven theoretical foundations and empirical confirmations, which is regarded as one of the main problems of software engineering [14]. Secondly, most framework improvements are only technologically oriented. However, while frameworks are intensively used by software developers, it is also reasonable to incorporate OOF-related stakeholders into OOF research. This supposition is consistent with the results of the Morisio *et al*. [15] study, which demonstrated the focal role of human factors in the success of software reuse. Finally, given the importance of OOFs and the extent of their impact on software development projects, there was no research identified that addressed fundamental issues such as *how* different OOF-related improvements influence application developers who use frameworks.

This article will attempt to address the limits of OOF by considering both the positive and negative aspects of existing OOF research. It builds upon proven theoretical foundations in IT acceptance research and concretizes underlying theoretical models with regard to OOF and its constituent parts. The original contributions of this article are: (1) an abstract view of OOF, (2) a conceptual model representing causal relationships between OOF constituent elements, OOF characteristics and OOF user perceptions, and (3) a high-level OOF improvement process, based on the conceptual model.

2. BACKGROUND AND THEORY

When identifying or applying improvements to OOF, the most important thing to consider is how these improvements will affect OOF users (application developers) [15]. A common way to measure the effects of improvements is to measure the success of an examined object (in our case OOF) before and after the improvements have been applied.

Several closely related socio-technical system success theories and models have been introduced in information systems (IS) and information technologies (IT). Especially in software engineering, software success measurement has traditionally been focused on delivering functional software within certain economical and temporal constraints. However, such measures of success have often proved to be complex and inac-

curate because of the diversity of software systems and the subjective nature of success [16].

On the other hand, there is empirical evidence that a more general and accurate measure of success lies within a system use that provides a more frugal and user-focused measure of system success [17, 18]. Therefore, investigations of IT use are generally characterized as the second stream of IS success research [19].

There is plenty of evidence that OOF success should be measured with system use and that traditional software success measures are inappropriate for OOF. First, OOFs are usually developed in an incremental and evolutionary way, which is formally presented in evolving frameworks pattern language [20]. Such a developmental approach makes temporal constraints for indicating success inappropriate. Secondly, the OOF evolutionary development approach is similar to open source development [21]. Therefore, many OOFs are developed within open source projects [22]. In the case of open source projects, the primary indicator of a project's success is the number of users [23], whereas the economic constraints of success indicators are unsuitable. Thirdly, the important distinction between OOF and applications is that OOF has to cover common concepts in the domain they operate in, whereas applications are only concerned with specific application requirements. While the desired functions are not the primary concern of OOF they cannot be treated as a relevant indicator of OOF success.

Besides the above-presented benefits, the most important preference of system use success measure is that it has well-defined causal antecedents. Based on the logic that a system must first be accepted to be used [24], it might be presumed that ensuring acceptance increases the likelihood of a system's success. Therefore, understanding what influences users to use IT, continues to be a major issue in the IT success area [25].

Among theories that deal with the acceptance and use of IT, Davis's [26] Technology Acceptance Model (TAM), is one of the most cited, validated and oft-used theoretical models. As explained in Fig. 1, a key assumption of TAM is that external variables (EV) influence the decision for using particular IT only indirectly: through their impact on a user's beliefs, the perceived ease of use (PEOU) and perceived usefulness (PU). These two beliefs influence the users' attitude toward using IT (ATU). ATU sequentially influences the behavioral intention to use (BI), which is the key factor in determining actual IT use (U).

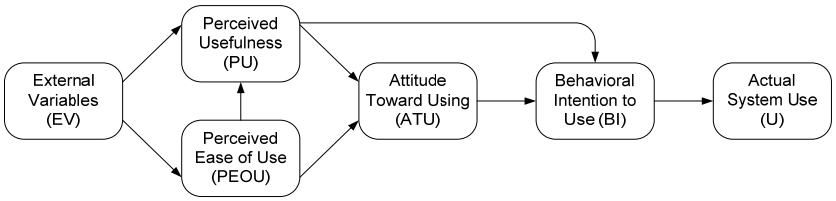

Fig. 1. Technology acceptance model – TAM [26].

The causal relationships between TAM constructs (see the arrows in Fig. 1) have been proven as valid for different types of IT and in different settings (voluntary, mandatory, individual and organizational) [25]. Moreover, end users and system developers were involved in TAM research [27]. Another important finding of TAM is its applica-

bility for post-adoption behavior, which has been proven by Hong *et al*. [28].

Several reasons can be given that justify using TAM as the central theoretical model of our research: (1) its specific focus on IT; (2) its proven validity, reliability and extensibility [27, 29]; (3) its extensive application [30]; (4) its accumulated research tradition [25]; and (5) its ability to be used in adoption (potential users) and post-adoption (actual users) behavior [28].

3. OOF EVALUATION AND IMPROVEMENT MODEL

Following the logic of TAM, we anticipate that if we try to improve OOF in a way that influence proper OOF-related EV in a "proper way," users will develop more positive user beliefs and attitudes toward OOF. This means that there will be a higher probability that users will be intent on starting to use, or continuing to use, OOF (Fig. 1).

However, while TAM does not define which EVs influence user perceptions and although EV depend on the type of IT [31], it remains unknown which EV to observe in a specific context. Different types of EVs have been investigated [25] where system characteristics have proven to have a focal impact on system use [32]. Besides, they showed the potential to directly affect both PEOU and PU [33].

Another evaluation problem appears when considering OOF as integrity. OOFs are generally regarded as the most complex reusable structures, made up of different design and implementation parts, where different parts might differently influence OOF success. Besides, OOFs differ among one another in several aspects [11, 34, 35], which makes it difficult to define general procedures and measures of OOF. However, a lot of research related to OOF constituting elements has already been conducted. Relying on the well proven "divide and conquer" principle, we anticipate that OOF can be easier to evaluate when evaluating constituent elements. In summary, the above research problems can be stated with two research questions:

– What OOF characteristics influence OOF acceptance? (1)
– What common elements constitute OOF? (2)

3.1 Identification of OOF Characteristics

The identification of initial concepts for the first research question was performed by surveying available literature. The search included books, journals, conferences, theses and technical reports found in electronic libraries (like Citeseer, Science Direct and IEEE). The search terms were constructed from general application areas (OOF, application frameworks, software frameworks), specific application areas (*e.g*. guidelines, success, quality, *etc*.) and specific terms (*e.g*. drivers, determinants, reasons, *etc*.). The search process was defined as saturated when combining these terms did not yield any new results.

The literature uncovered during the identification process was then analyzed for desirable OOF characteristics, where the factors in the literature were directly stated or embedded in textual descriptions. In addition to this, the factors were positively or negatively related to OOF success. The outcome resulted in raw factors which were additionally processed with cross-fertilization, duplicate removal and consistency formulation.

For each factor, the number of occurrences (multiplicity) was evaluated. Orthogonal factors on the same abstraction level were defined by the splitting and clustering of concepts (factors). In order to keep the number of factors low, factors that appeared only once were omitted from any further research.

In line with the above-defined research approach, the most common OOF characteristics were identified from the literature, where factors with the highest multiplicities were shown to be adaptability and understandability. While both factors correspond to REBOOT (Reuse Based on Object Oriented Techniques) reusability factors [36], concept clustering and concept splitting was performed according to REBOOT reusability model multi-level structure. Concept clustering was performed on maturity and fault tolerance factors, where the corresponding multiplicities were summarized (Fig. 2). Concept splitting was performed on reusability, where the multiplicities were equally divided between resulting factors. Finally, equal factors were merged and their multiplicities summarized. Trying to keep research within reasonable boundaries, efficiency and portability factors were omitted from this phase of research because they indicated the lowest multiplicities.

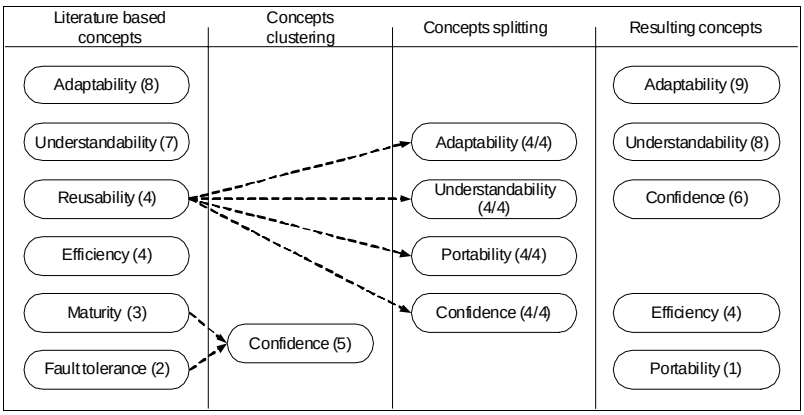

Fig. 2. Common OOF characteristics and their multiplicities.

3.1.1 Understandability

Understandability is defined as "the capability of a software product to enable the user to understand whether the software is suitable and how it can be used for particular tasks and conditions of use" [37]. The REBOOT model breaks up understandability into four criteria: self-descriptiveness, documentation level, structural complexity and inheritance complexity. Understandability is often substituted with "ease of use." However, ease of use represents a user's reaction to using OOF. Thus, it can be hypothesized (H1) that understandability positively influences user's attitudes toward OOF, mediated by the perceived ease of use.

3.1.2 Adaptability

Adaptability (*i.e.* flexibility) is the ability to use OOF in more than one context. RE-

BOOT defines adaptability as the "ease with which a component can be adapted to fulfill a requirement that differs from that for which it was originally constructed." Adaptability is the most common OOF characteristic (see Fig. 2), therefore it can be hypothesized (H2) that OOF adaptability positively influences the perceived usefulness of OOF. However, adaptability usually comes with increased complexity which reduces the ease of using OOF [38]. So, the third hypothesis (H3) can be defined as: adaptability negatively influences perceived ease of using OOF.

3.1.3 Confidence

Confidence is defined as the "probability that a module, program or system performs its defined purpose satisfactorily over a period of time in another environment than for which it was originally constructed" [36]. Confidence is important for OOF for two major reasons. Firstly, while the decision to develop or use OOF is a long-term one, the potential OOF user needs to have confidence that OOF will perform satisfactorily over a period of time. Secondly, while OOF constitutes parts of end applications (framework instances) the potential OOF defectiveness usually affects all its instances. It can be hypothesized (H4) that confidence increases the perceived usefulness of OOF.

3.1.4 Anticipated causal relationships

Fig. 3 shows the conceptual causal model, which includes important OOF characteristics, TAM user beliefs (PEOU and PU) and anticipated causal relationships. No research has been identified which could empirically prove the causal model. However, for this stage of our research, we believe that there are enough indices within the surveyed literature.

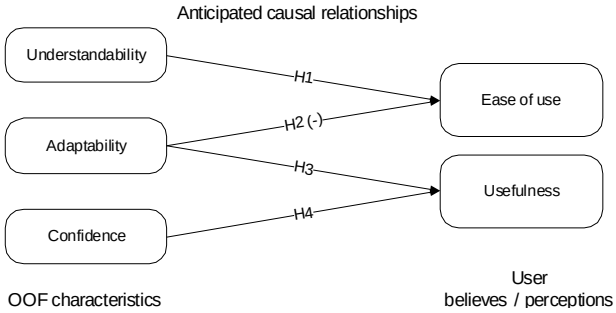

Fig. 3. Important OOF characteristics and their anticipated impact on user beliefs.

3.2 Identification of OOF Constituent Elements

While OOF consists of part code and part design reuse, OOF constituting elements can be primary divided into code elements and design elements [39, 40]. Besides this, OOF design is dependent on OOF development guidelines. Surveying the literature, we find that OOF constituent elements are consistently defined across the literature.

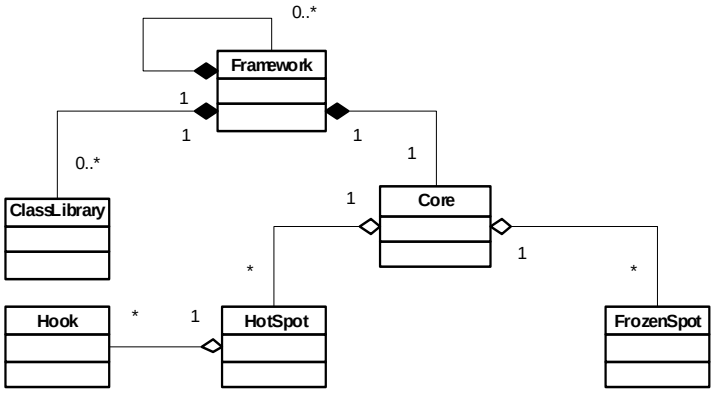

Fig. 4. The common OOF code elements.

3.2.1 Code elements

Several authors have already analyzed the structure of OOF [38, 39]. The results of their research are summarized in the UML class model in Fig. 4.

Fig. 4 shows the most common OOF code (class) elements and their relationships. At the top level, OOF consists of interfaces, abstract and concrete classes that constitute the framework core and framework library. The framework core defines the main functions and behavior of the OOF, whereas the framework library extends the framework core with concrete components that can be used by developers with little or no modifications. In addition to this, OOFs might include other OOFs. Framework core elements can be further divided into two major groups: hot spots and frozen spots.

Hot spots (or hinges) are the general areas of variability within a framework. Hot spots are framework parts where programmers can add their own code to implement the functionalities specific to their own project. A hot spot can contain several hooks, which represent actual places (methods) in the framework that can be adapted or extended in order to provide application-specific functionality.

In contrast, frozen spots within the framework capture the commonalities across framework instances. These remain unchanged in any instantiation of the framework.

3.2.2 Design elements

Different authors have investigated or defined different OOF design elements [38, 40, 41] where the most common design elements have been shown to be dynamic binding and design patterns. Dynamic binding (run-time binding, late binding) is a technique by which the exact piece of code to be executed is determined at run-time (as opposed to compile-time). Dynamic binding is identified as one of the key issues of OOF and represents the core concept that distinguishes OOF from other reuse techniques (components, libraries). Dynamic binding allows OOF to call instances and is therefore known as "inversion of control" or "Hollywood principle"[1] [22]. Inversion of control can be implemented with subclassing, dependency injections, template methods and closures.

[1] Don't call us-we'll call you.

Table 1. Design patterns, commonly used in OOF.

Design pattern	Description
Adapter	Adapts a class interface into one that a client expects.
Composite	Defines a composite object designed as a composition of one or more similar objects, all exhibiting similar functionality. The composite object then exposes the properties and methods for child object manipulation as if it was a simple object.
Façade	Provides a simplified interface to a larger body of code, such as a class library.
Observer	Objects register to observe an event which may be raised by another object.
Singleton	Restricts instantiation of a class to a single object.
Strategy	The strategy pattern is useful for situations where it is necessary to dynamically swap algorithms used in an application.
Template method	Defines the skeleton of an algorithm in an operation, deferring some steps to subclasses. The Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

Table 2. Experience based OOF related guidelines.

Author(s)	Guideline
[45]	Reduce the number of classes and methods users have to override.
	Simplify the interaction between OOF and application extensions.
	Isolate platform dependent code.
	Do as much as possible within the framework.
	Factor code so that users can override limiting assumptions.
	Provide notification hooks so that users can react to important state changes.
[46]	Consolidate similar functionality into a single abstraction.
	Break down larger abstractions into smaller ones with greater flexibility.
	Implement each key variation of an abstraction class.
	Use composition rather than inheritance.
[11]	The interface of a component should be separated from its implementation.
	Interfaces should be role oriented.
	Role inheritance should be used to combine different roles.
	Prefer louse coupling over delegation.
	Prefer delegation over inheritance.
	Use standard technology.
	Automate configuration.
	Automate documentation.
[5]	Justify framework development.
	Prevent the framework of becoming unmanageable.
	Skilled team – develop framework in a clear and consistent way.
	Develop pilot applications.
	Use small objects to increase flexibility and restrict complexity.
	Perform pilot-based tests.
	Involve framework users in framework development.
	Accept change requests if several teams will use the additional functionality.
[20]	Develop OOF according to the evolving frameworks pattern language.

Design patterns have been associated with OOF from their very beginnings [38]. They represent descriptions of communicating objects and classes that are customized to solve a general design problem within a particular context [42]. Design patterns can be classified in terms of the underlying problem they solve: fundamental, creational, structural, behavioral and concurrency patterns. In OOF, design patterns are used for several purposes (see Table 1); however they are especially suitable for designing hot spots to improve OOF flexibility and extensibility. Several authors have already examined design patterns in OOF [4, 6, 12, 43, 44].

3.2.3 Development guidelines

OOFs are, besides their code elements and design elements, highly dependent on development philosophy or guidelines. Therefore, we treat OOF guidelines as a constituent part of OOF. The major concern of OOF development guidelines is to improve the interaction between the OOF user and OOF [40], which is similar to our research objectives. A set of 71 OOF-related guidelines has been defined by Landin *et al*. [13], where the most common guidelines are summarized in Table 2.

In addition to this, some guidelines are favored by a specific OOF. For example, "use convention over configuration" and "don't repeat yourself (DRY)", are major development guidelines in the "Ruby on Rails" (RoR) framework. Despite that these guidelines are strongly related to RoR, they are language and OOF independent. On the other side, the developers of the "Keel framework" have identified common anti-patterns (classes of commonly reinvented bad solutions) which they try to avoid in Keel development.

3.3 Conceptual OOF Model

The three groups of OOF constituent elements are interdependent. First, design elements (dynamic binding and design patterns) influence OOF interfaces and classes. For example, a simplified use of an OOF class library can be achieved using a Façade pattern (see Tables 1 and 3). Second, the guidelines as presented in Table 2 might influence design elements. For example: "break down larger abstractions into smaller ones with greater flexibility" implies the use of aggregate or composite design patterns (Table 1). Third, guidelines influence code elements directly (for example "reduce the number of classes and methods users have to override") or indirectly (through design elements).

When considering TAM, causal relationships (solid arrows) between OOF elements and causal relationships between OOF characteristics and user beliefs, a causal OOF model can be defined as presented in Fig. 5.

It can be anticipated that the model, shown in Fig. 5, is capable of explaining the major antecedents of OOF success in the following way: elements, which constitute OOF, influence important OOF characteristics. These characteristics further, directly or indirectly (mediated by user perceptions), impact users' intentions to use or further use a specific OOF. Finally, "continuous use" can be treated as a valid indicator of OOF success.

The relationship between OOF constituent elements and OOF characteristics can be defined in two ways. First, code elements can be directly analyzed for their adaptability,

Table 3. An example of the connectivity matrix.

[*i*]	Design pattern/guideline	Impacts	[*j*]		
			1. Adaptability	2. Understand	3. Confidence
1	Template method design pattern.	Hot spots	1	– 1	0
2	Implement each key variation of an abstract class.	Hot spots	0	1	0
3	Façade design pattern.	Class library	0	1	0
4	Composite design pattern.	Hot spots	0	1	0
5	Break down larger abstractions into smaller ones.	Hot spots	1	– 1	0
6	Reduce the number of classes and methods, users have to override.	Hot spots	– 1	1	0
7	Involve users into framework development.	All elements	0	1	1

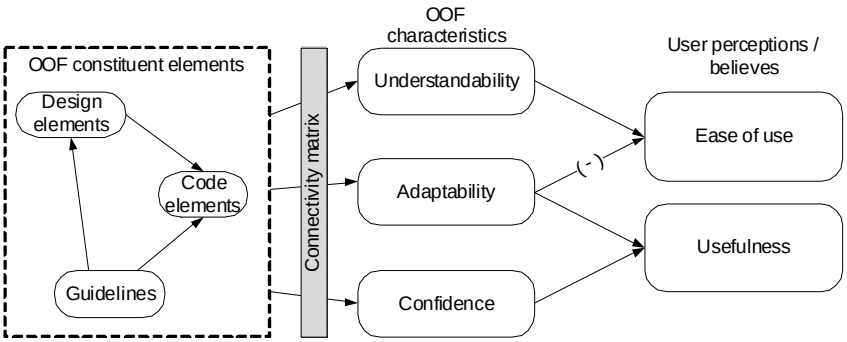

Fig. 5. Conceptual OOF evaluation model with anticipated causal relationships.

understandability and confidence, using metrics for reusable components. REBOOT [36] defines 35 metrics that are used to define criteria for reusable software, where criteria can be further broken down into factors like portability, adaptability and confidence. Each metric is defined and explained in detail. For example, the understandability factor consists of the following criteria: (1) self-descriptiveness, (2) documentation level, (3) structural complexity and (4) inheritance complexity. Each of the criteria is measured by several metrics. For example, self-descriptiveness is measured by comment density (*M*3)[2] and comment descriptiveness (*M*4), as presented below[3]:

$$M3 = \frac{Linse\ of\ comments}{Lines\ of\ code}, \tag{1}$$

$$M4 = \frac{\frac{Qc4(\sum Qm1_i + Qm2_i)}{\#\ member\ functions}}{3}. \tag{2}$$

[2] *M*3 and *M*4 are standardized REBOOT [36] Metrics IDs.
[3] *Qc*4, *Qm*1 and *Qm*2 represent "check-list" metrics. See REBOOT [36] for details.

Second, design elements and guidelines can be analyzed directly or indirectly, via their impact on code elements. Direct impacts can be obtained by surveying existing OOF literature and other relevant literature for identifying OOF guidelines and design elements and the potential impact they might have on OOF characteristics. In a recent research, Guéhéneuc *et al*. [43] proposed to threat design patterns and corresponding design motifs as "laws of software quality". Further, design patterns are commonly united in catalogues, like GoF (Gang of Four) [42]. These catalogues describe patterns in a structured way, including their implications [44]. In a similar way, Landin in his thesis [13] catalogued a set of 71 guidelines which can be applied to OOF.

In addition to existing research, empirical research can be performed to identify causal relationships between them and OOF characteristics. In this case, the impacts are commonly represented with standardized regression weights (factor loadings).

As a result of catalogues and empirical research, we can form a connectivity matrix[4] for OOF constituent elements and OOF characteristics (Table 3), where rows are OOF design elements or guidelines, and columns show OOF characteristics. Each cell in the matrix contains the number "x_{ij}", which represents the impact between OOF design element or guideline "i" and the OOF characteristic "j". The values x_{ij} can be defined as the following: (1) $x_{ij} < 0$ represents the negative impact of a design element or the guideline "i" on OOF characteristic "j"; (2) $x_{ij} > 0$ represents the positive impact of a design element or guideline "i" on OOF characteristic "j"; and (3) $x_{ij} = 0$ represents no impact between a design element or the guideline "i" and the OOF characteristic "j".

4. OOF IMPROVEMENT PROCESS

Based on the conceptual OOF evaluation model (Fig. 5), several scenarios for OOF improvements can be anticipated[5]. A high-level OOF improvement process is presented in Fig. 6. It is based on Booch's finding, which stated that "developers will use a specific OOF only if its benefits (suits to PU) outweigh its drawbacks (suits to PEOU)" [48]. We decided to use Business Process Modeling Notation (BPMN) to model the process because it is easily understandable, supported by OMG (Object Management Group) and highly expressive [49].

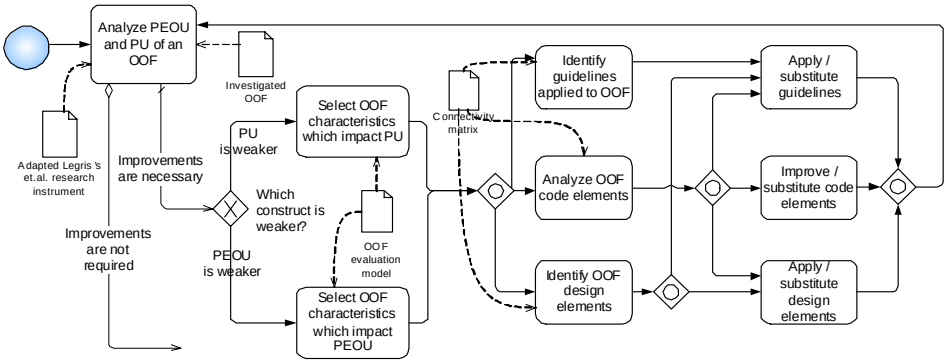

Fig. 6. High-level OOF improvement process.

[4] A similar solution was proposed by Guéhéneuc *et al*. [43] for design patterns quality characteristics.

[5] This part of the research has been partially influenced by the research of Vavpotič *et al*. [47], who examined scenarios for improving software development methodologies.

The main idea behind the proposed OOF improvement process lies in an iterative and evolutionary process, which improves OOF by particularly improving its ease of use and usefulness. Iterative and evolutionary improvement process suits to OOF development model as proposed by Roberts [20]. The major process activities with practical examples are presented below.

4.1 Analyzing User Perceptions Regarding OOF

To analyze how users perceive OOF, they need to be surveyed for their perceptions of the investigated OOF. The measurement of psychometric user responses is commonly based on the Likert scale. The Likert scale is a multi-item scale, where typical items are statements. In Likert scale, the respondent is asked to indicate his or her degree of agreement with the statement, most commonly on a seven-point scale with the endpoints "Strongly disagree" and "Strongly agree." Legris *et al*. gave great attention to investigating the use of the PEOU and PU measures [27]. Based on their findings, we propose a five-item Likert-based measurement tool for measuring PEOU, and four items for measuring PU of OOF:

– PEOU1: The OOF is rigid and inflexible to interact with.
– PEOU2: I find it is easy to get the OOF to do what I want it to do.
– PEOU3: Overall, I believe that the OOF is easy to use.
– PEOU4: Learning to operate the OOF is easy for me.
– PEOU5: I find it takes a lot of effort to become skillful at using the OOF.
– PU1: I believe that using the OOF will further increase my productivity.
– PU2: I believe that using the OOF will further increase my job performance.
– PU3: I believe that using the OOF will further enhance my job effectiveness.
– PU4: Overall, I believe the OOF will be further useful in my job.

4.2 Selecting OOF Characteristics

Corresponding the results of the previous step, OOF characteristics, which have an impact on the "weaker" user beliefs construct (PEOU or PU), should be selected. The "weaker construct" can be defined as those with lower average values of Likert measures for PEOU and PU of OOF. According to the OOF causal model (Fig. 5), two major scenarios for improving PEOU and PU of OOF exist:

– Improving PU of OOF. This can be improved by improving the adaptability and confidence of OOF.
– Improving PEOU of OOF. This can be improved by improving the understandability and simultaneously lessening OOF overgeneralization. For example, if OOF users better evaluate PU than PEOU, understandability should be primarily improved, where the cause might be an over-generalized OOF (see Fig. 5).

4.3 Analyzing and Improving OOF Constituent Elements

The selected OOF characteristics can be improved by analyzing, identifying and resolving problems in elements which constitute OOF. According to Fig. 5, these ele-

ments are: code elements, design elements and guidelines. The focus should be on those elements which have an impact on the "problematic"[6] OOF characteristics. "Problematic" code elements can be analyzed with measurements, whereas problematic design elements and guidelines can be identified from existing catalogues or the connectivity matrix (Table 3).

Finally, the identified problems within OOF constituent elements should be resolved by modifying or replacing existing elements with those elements which positively influence "problematic" OOF characteristics. This can be done by applying design patterns or guidelines from the connectivity matrix (Table 3). Besides this, we should also be aware that OOF constituent elements are interdependent (Fig. 5) which means that there are several ways to improve some elements. For example, if OOF is perceived as difficult to use, understandability should be improved where the adaptability might be reduced. In this case, the root of the problem might be in design patterns applied to hot spots. These often result in higher adaptability, but simultaneously complicate OOF. A guideline such as "implement each key variation of an abstraction class" (see Table 2) could be used to resolve this problem.

4.4 Continual Improvement

The effects of OOF improvement can be evaluated with the same instrument used in the first step of the process – by analyzing OOF user perceptions. The following scenarios might occur when measuring user beliefs after applied OOF improvements:

1. Applied improvements have positively influenced the weaker user belief construct. In this case, we have achieved the expected results with positively influencing factors which influence OOF acceptance and consequently OOF success. The OOF improvement process can finish.
2. Applied improvements have no impact on the weaker user belief construct. In this case, we might apply wrong improvements to OOF constituent elements. There might be a problem within the OOF evaluation model, connectivity matrix or some factors outside the scope of the model influence investigated construct.
3. Applied improvements negatively influence the opposite user belief construct. The most reasonable explanation for this case is the contrariwise impact of adaptability on PEOU and PU. So, OOF should be modified in a way that will assure a reasonable extent of adaptability while not being too difficult to use.

5. CONCLUSION

A novel approach for the systematic evaluation and improvement of object-oriented frameworks (OOF) was introduced in this article. The proposed approach is based on the well-known "divide and conquer" principle which posts that a complex problem is easier to manage if it is broken down into simpler parts. We divided OOF into its most general constituent elements. Moreover, the technology acceptance model (TAM) was used as a focal theory for user-based OOF improvement. Combining both parts of the research resulted in the presumption that elements which constitute OOF, subsequently influence

[6] Those, which negatively influences the "weaker" user beliefs construct.

OOF user perceptions via the most important OOF characteristics: understandability, adaptability and confidence. According to this presumption, a causal OOF evaluation model was defined. And finally, the proposed OOF model was used as a basic one in the OOF improvement process.

Several other theoretical and practical implications of this research can be foreseen. The proposed OOF model can offer the basis for developers' OOF design directions. Contrarily, software users can use the model to evaluate the appropriateness of OOF.

From a theoretical standpoint, this research can be seen as an umbrella or complementary research to existing OOF research, which is mainly concerned with partial (code, design patterns, guidelines or documentation) OOF improvements. Finally, the ideas behind this research might stimulate researchers in other IT areas to examine and improve existing and coming technologies in a more systematic and engineering manner.

5.1 Validity Threats and Research Limits

We are aware that our research is not without limitations. Several theoretical and practical sources for constructing the OOF evaluation model (Fig. 5) have been used. However, we are fully aware of the research limits and the fact that a model can represent only an approximation of real world phenomenon.

Some threats to the structural validity of the conceptual OOF model exist. First, the causal relationships, as defined in this model, were derived from a generally applicable theory (TAM) and OOF-related literature, although none of these relationships was explicitly proven in our research. Our current activities are aiming in confirming or rejecting hypotheses, stated in this research.

Secondly, while trading simplicity against completeness we decided to investigate only the most relevant user perceptions and OOF characteristics. We used TAM's user perceptions; however, TAM and its successor TAM2 [50, 51] have evolved into a more complex model, known as the Unified Theory of Acceptance and Use of Technology (UTAUT) [52]. Compared with TAM, UTAUT defines two extra user perceptions: social influences and facilitating conditions. However, we decided for TAM, because only one study exists that confirms UTAUT validity and robustness. In addition, we decided to include only the most common OOF characteristics in the model. There is no guarantee that the investigated body of literature was wide or diverse enough to identify all OOF characteristics. In addition, the clustering and splitting of concepts may greatly affect the number of occurrences (multiplicity).

Thirdly, to apply the model to a broad range of OOF, an abstract OOF structure was defined, where the structure of an actual OOF might differ from the abstract OOF structure. This might lessen the practical value of our research.

5.2 Future Research Activities

We are aware that the conceptual OOF evaluation model and the OOF improvement process presented in this article suffer from incomplete research. Thus, this research has to continue in several directions.

First, the anticipated causal relationships (hypotheses) of the conceptual OOF model should be validated by surveying actual OOF users. This research requires the operation-

alization of the proposed model, the identification of a proper research sample and the statistical analysis of results. These are our current research activities.

Secondly, the proposed improvement process should be validated in a case study, which is an ideal methodology when a holistic, in-depth investigation is needed [53]. We are planning to perform an explanatory case study in an actual organization which is basing its product line on an in-house-developed OOF. We anticipate that the results of the case study will help us identify both the major practical benefits and limits of our research.

The third direction of future activities would be to improve the practical value of the proposed OOF improvement process in light of case study results, where some improvement ideas are: (1) a partial or full guidance through the OOF improvement process, and (2) the establishment of a knowledge base of elements and their impacts of concrete OOFs. A partial automation of the proposed model could be realized with software for the evaluation of OOF code elements and with software for identifying design elements and guidelines in a concrete OOF. This research could be based on research, performed by Guéhéneuc *et al*. [43]. A useful knowledge base could contain the evaluation results of existing OOFs and the impact of applied OOF guidelines and design patterns on OOF characteristics (the connectivity matrix).

REFERENCES

1. S. A. Mamrak and S. Sinha, "A case study: Productivity and quality gains using an object-oriented framework," *Software-Practice and Experience*, Vol. 29, 1999, pp. 501-518.
2. M. Morisio, D. Romano, and I. Stamelos, "Quality, productivity, and learning in framework-based development: An exploratory case study," *IEEE Transactions on Software Engineering*, Vol. 28, 2002, pp. 876-888.
3. R. E. Johnson and B. Foote, "Designing reusable classes," *Journal of Object-Oriented Programming*, Vol. 1, 1988, pp. 22-35.
4. S. Srinivasan, "Design patterns in object-oriented frameworks," *Computer*, Vol. 32, 1999, pp. 24-32.
5. D. Manolescu, J. Noble, and M. Voelter, "Patterns for successful object-oriented framework development," *Pattern Languages of Program Design 5*, Addison Wesley Professional, 2006, pp. 401-431.
6. H. C. Cunningham, Y. Liu, and C. H. Zhang, "Using classic problems to teach Java framework design," *Science of Computer Programming*, Vol. 59, 2006, pp. 147-169.
7. D. Batory, R. Cardone, and Y. Smaragdakis, "Object-oriented frameworks and product lines," in *Proceedings of the 1st Software Product Line Conference*, 2000, pp. 227-247.
8. S. Sparks, K. Benner, and C. Faris, "Managing object-oriented framework reuse," *Computer*, Vol. 29, 1996, pp. 52-61.
9. M. F. Fontoura, "Object-oriented application frameworks: The untold story," in *Proceedings of the 14th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 1999, pp. 1-2.
10. J. Bosch, P. Molin, M. Mattsson, and P. Bengtsson, "Object-oriented frameworks – Problems and experiences," *Building Application Frameworks: Object Oriented*

Foundations of Framework Design, Vol. 1, 1997, pp. 1-2.
11. J. van Gurp and J. Bosch, "Design, implementation and evolution of object oriented frameworks: Concepts and guidelines," *Software-Practice and Experience*, Vol. 31, 2001, pp. 277-300.
12. R. E. Johnson, "Documenting frameworks using patterns," in *Proceedings of Conference on Object-Oriented Programming Systems*, *Languages*, *and Applications*, *ACM SIGPLAN Notices*, Vol. 27, 1992, pp. 63-76.
13. N. Landin and A. Niklasson, "Development of object-oriented frameworks," Master Thesis, Department of Communication Systems, Lund Institute of Technology, Lund University, 1995, pp. 1-154.
14. M. Shaw, "Prospects for an engineering discipline of software," *IEEE Software*, Vol. 7, 1990, pp. 15-24.
15. M. Morisio, M. Ezran, and C. Tully, "Success and failure factors in software reuse," *IEEE Transactions on Software Engineering*, Vol. 28, 2002, pp. 340-357.
16. S. Behrens, K. Jamieson, D. Jones, and M. Cranston, "Predicting system success using the technology acceptance model: A case study," in *Proceedings of the 16th Australasian Conference on Information Systems*, 2005, pp. 1-10.
17. W. H. DeLone and E. R. Mclean, "Information systems success: The quest for the dependent variable," *Information Systems Research*, Vol. 3, 1992, pp. 60-96.
18. I. Juhani, "An empirical test of the DeLone-McLean model of information system success," *ACM SIGMIS Database*, Vol. 36, 2005, pp. 8-27.
19. E. J. Garrity, B. Glassberg, Y. J. Kim, G. L. Sanders, and S. K. Shin, "An experimental investigation of web-based information systems success in the context of electronic commerce," *Decision Support Systems*, Vol. 39, 2005, pp. 485-503.
20. D. Roberts and R. Johnson, "Evolving frameworks: A pattern language for developing object-oriented frameworks," in *Proceedings of the 3rd Conference on Pattern Languages and Programming*, 1997, pp. 1-13.
21. V. Potdar and E. Chang, "Open source and closed source development methodologies," in *Proceedings of the 4th Workshop on Open Source Software Engineering,* 2004, pp. 105-109.
22. R. E. Johnson, "J2EE development frameworks," *Computer*, Vol. 38, 2005, pp. 107-110.
23. K. Crowston, H. Annabi, and J. Howison, "Defining open source software project success," in *Proceedings of International Conference on Information Systems*, 2003, pp. 1-14.
24. M. K. Chang and W. Cheung, "Determinants of the intention to use Internet/WWW at work: A confirmatory study," *Information and Management*, Vol. 39, 2001, pp. 1-14.
25. J. H. Sharp, "Development, extension, and application: A review of the technology acceptance model," in *Proceedings of the Information Systems Educators Conference*, Vol. 23, 2006, pp. 1-9.
26. F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS Quarterly*, Vol. 13, 1989, pp. 318-331.
27. P. Legris, J. Ingham, and P. Collerette, "Why do people use information technology? A critical review of the technology acceptance model," *Information and Management*, Vol. 40, 2003, pp. 191-204.

28. S. J. Hong, J. Y. L. Thong, and K. Y. Tam, "Understanding continued information technology usage behavior: A comparison of three models in the context of mobile internet," *Decision Support Systems*, Vol. 42, 2006, pp. 1819-1834.
29. W. R. King and J. He, "A meta-analysis of the technology acceptance model," *Information and Management*, Vol. 43, 2006, pp. 740-755.
30. A. L. Lederer, D. J. Maupin, M. P. Sena, and Y. Zhuang, "The technology acceptance model and the world wide web," *Decision Support Systems*, Vol. 29, 2000, pp. 269-282.
31. P. B. Seddon, S. Staples, R. Patnayakuni, and M. Bowtell, "Dimensions of information systems success," *Communications of Association of Information Systems*, Vol. 2, 1999, pp. 1-61.
32. S. S. Al-Gahtani, "System characteristics, user perceptions and attitudes in the prediction of information technology acceptance: A structural equation model," in *Proceedings of the Diffusion Interest Group in Information Technology Conference*, 1998, pp. 1-23.
33. W. Y. Hong, J. Y. L. Thong, W. M. Wong, and K. Y. Tam, "Determinants of user acceptance of digital libraries: An empirical examination of individual differences and system characteristics," *Journal of Management Information Systems*, Vol. 18, 2001, pp. 97-124.
34. M. Mattsson, "Object-oriented frameworks, a survey of methodological issues," Department of Computer Science, Lund University, 1996.
35. A. Krajnc and M. Hericko, "Classification of object-oriented frameworks," *EUROCON*, Vol. 2, 2003, pp. 57-61.
36. G. Sindre, R. Conradi, and E. A. Karlsson, "The reboot approach to software reuse," *Journal of Systems and Software*, Vol. 30, 1995, pp. 201-212.
37. ISO 9126. ISO/IEC TR 9126-Software engineering, product quality, quality model, International Organization for Standardization, 2001.
38. G. Froehlich, J. Hoover, L. Liu, and P. Sorenson, "Designing object-oriented frameworks," *CRC Handbook of Object Technology*, 1998.
39. L. Sangdon, C. Hansuk, Y. Youngjong, and L. Sangduck, "Storage and management of object-oriented frameworks," in *Proceedings of International Conference on Systems*, *Man*, *and Cybernetics*, 1999, pp. 762-767.
40. M. Mattsson, "Evolution and composition of object-oriented frameworks," Ph.D. Thesis, Department of Software Engineering and Computer Science, University of Karlskrona, Ronneby, 2000.
41. D. Parsons, A. Rashid, A. Telea, and A. Speck, "An architectural pattern for designing component-based application frameworks," *Software-Practice and Experience*, Vol. 36, 2006, pp. 157-190.
42. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, Canada, 1995.
43. Y. G. Guéhéneuc, J. Y. Guyomarc'h, K. Khosravi, and H. Sahraoui, "Design patterns as laws of quality," *Object-Oriented Design Knowledge: Principles*, *Heuristics*, *and Best Practices*, Vol. 1, 2006, pp. 105-142.
44. R. Lajoie and R. K. Keller, "Design and reuse in object-oriented frameworks: Patterns, contracts, and motifs in concert," in *Proceedings of the 62nd Congress of the Association Canadienne Francaise pour l'Avancement des Sciences*, 1994, pp. 1-12.

45. Taligent Inc., The Power of Frameworks: For Windows and Os/2, Taligent Press, 1995.
46. A. Birrer and T. Eggenschwiler, Frameworks in the Financial Engineering Domain: An Experience Report, 1993, p. 21.
47. D. Vavpotič and M. Krisper, "Measuring and improving technical and social suitability of software development methodology," *Advances in Theory*, *Practice and Education*, 2004, pp. 1-5.
48. G. Booch, "Design an application framework," *Dr. Dobbs Journal*, Vol. 19, 1994, pp. 24-32.
49. T. Rozman, R. H. Vajde, and I. Rozman, "Experiences with business process modeling notation in educational process," in *Proceedings of International Business Information Management Conference*, 2003, pp. 310-315.
50. V. Venkatesh and F. D. Davis, "A theoretical extension of the technology acceptance model: Four longitudinal field studies," *Management Science*, Vol. 46, 2000, pp. 186-204.
51. V. Cechticky, P. Chevalley, A. Pasetti, and W. Schaufelberger, "A generative approach to framework instantiation," in *Proceedings of the 2nd International Conference on Generative Programming and Component Engineering*, 2003, pp. 267-286.
52. V. Venkatesh, M. G. Morris, G. B. Davis, and F. D. Davis, "User acceptance of information technology: Toward a unified view," *MIS Quarterly*, Vol. 27, 2003, pp. 425-478.
53. W. Tellis, "Introduction to case study," The Qualitative Report, Vol. 3, 1997, http://www.nova.edu/ssss/QR/QR3-2/tellis1.html.

Gregor Polančič Ph.D., researcher and teaching assistant at Faculty of Electrical Engineering and Computer Science at University of Maribor. His work and research interests include business process modeling, web technologies, object-oriented frameworks, open source concepts, empirical research methods and user acceptance of IT. During more than ten years of work in IT area he has been involved in numerous software engineering projects. He is author of BPMN Poster and Research Methods Poster, which are freely available at www.itposter.net.

Romana Vajde Horvat Ph.D., teaches and works in topics related to software process improvement, quality management and standardization in IT. Her research is related to Complexity of Software Process Models based on Petri nets. During eighteen years of work in IT area she has managed and cooperated in numerous projects relates to process improvement in organizations. She cooperated in ISO workgroups for development of guidelines for process management in quality management systems. She is certified PRINCE2 project manager and is actively involved

in the European Certification Association for several job roles and training schemas related to information technology.

Ivan Rozman received the Ph.D. degree from University of Maribor, Maribor, Slovenia in 1983. His interests include quality assessment, project management, software metrics and software design. He has been involved in numerous commercial and government projects in many of them as a project manager. Prof. Rozman is author and co-author of numerous articles published in different scientific journals and a member of several program committees at domestic and international conferences.