2010 International Symposium on Information Technology, 15-17 June 2010, Kuala Lumpur, Malaysia.

# A Taxonomy of Computer-supported Critics

Norhayati Mohd Ali, John Hosking
Department of Computer Science
University of Auckland
Private Bag 92019, Auckland, New Zealand
nmoh044@aucklanduni.ac.nz, john@cs.auckland.ac.nz

John Grundy
Faculty of Information and Communication Technologies
Swinburne University of Technology
PO Box 218, Hawthorn, Victoria, Australia
jgrundy@swin.edu.au

***Abstract*—Critics have emerged over the last several years as a specific tool feature to support users in computer-mediated tasks. These computer-supported critics provide proactive guidelines or suggestions for improvement to designs, code and other digital artifacts. The concept of a critic has been adopted in various domains, including: medical (ATTENDING, ONCONCIN), programming (Lisp-Critic, RevJava), software engineering (Argo/UML, ABCDE-Critic), design sketching (Design Evaluator) and others. Critics have proven to be an effective mechanism in providing feedback to users. In this paper we propose an initial critic taxonomy based on our review of the critic literature. We present the groups and elements of the critic taxonomy and explain the groups and critic elements together some examples. We believe our taxonomy will assist others in identifying, categorizing, developing and deploying computer-supported critics in a range of domains.**

***Keywords-critic; critic taxonomy; critiquing system;critic tool.***

## I. INTRODUCTION

The term "critic" was initially used by Miller to describe a software program that critiques human-generated solutions [21]. These types of program, also known as a critiquing system, have evolved in recent years to help users in computer-mediated tasks by providing feedback and suggestions for improvements [2, 6, 7, 9, 10, 16 -18, and 23]. The concept of such a critic is not new and it has been accepted in various domains such as medical applications (e.g., ATTENDING, ONCONCIN), programming (e.g., LISP-Critic, RevJava), software engineering (e.g., Argo/UML, ABCDE-Critic), and others. Furthermore, reports from various studies demonstrate that a computer-supported critic is an effective mechanism in providing feedback to users. For instance, the Design Evaluator supports designers with critical effective feedback and gives reasoning on the design sketches [23]. Likewise, the Java Critiquer detects statements in a student program code that can be improved for readability and best practice [18].

Various critic definitions can be found in the literature. Though each critic tool author often provides their own definition a common concept is that they provide knowledge support to users who lack specific pieces of knowledge about the problem or solution domains. Thus, a critic is primarily there to detect potential problems; offer advice and alternative solutions; and possibly provide automated or semi-automated design improvements to end users. Currently there is no accepted categorization of critics, definition of their different features and domains of discourse, nor a framework to compare and contrast different critics in any systematic way.

The aim of our paper is to present an initial critic taxonomy that we have developed from our reviews of the critic literature. Our critic taxonomy guides our own research work on the development of visual critic authoring templates for domain-specific visual language tools. We believe that it will be useful for critic developers in different domains to reason about their own critics. We first outline the creation and purpose of our critic taxonomy. We then explain the groups and elements that comprise the taxonomy. Finally we conclude with utilities of our critic taxonomy and suggest future work.

## II. PROPOSED CRITIC TAXONOMY

According to the Cambridge dictionary, a taxonomy is "a system for naming and organizing things …into groups which share similar qualities" [5]. The process of developing our critic taxonomy began with our review of the related literature concerning critics. Several articles and reports have been published to explain and discuss critics (or critiquing systems) [3, 4, 12, 13, 15, 18, and 24] as a supporting tool for a wide range of computer users. As we were designing our own critic authoring and realization system we decide to develop a critic taxonomy to assist us in reasoning about different kinds of critics we had come across. We then classified the information from the critic literature in the following groups, which were tailored to meet our specific needs.

- Critic domain – what domain(s) of discourse is the critic used in?
- Critiquing approach – does it compare or analyze target domain elements?
- Critic dimension – is the critic active, passive (invoked on user demand), reactive, proactive etc?
- Critic type – does the critic check for completeness, correctness, consistency, alternatives, a mixture?
- Modes of critic feedback – how does it provide end users with feedback?
- Types of critic feedback – suggestions, argumentation, explanation etc
- Critic implementation approach – how is the critic built or realized in the target tool(s)?
- Critic rules authoring – how are the rules embodied by the critic encoded?

The work was supported in part by a scholarship from the Ministry of Higher Education, Malaysia, a grant from the Software Process and Product Improvement and a Postgraduate Research Student Support account from the University of Auckland, New Zealand.

Fig.1 illustrates the groups and elements that make up our critic taxonomy. We briefly describe the groupings and their elements in the following section.

| Critic Groups and Elements | | | | | | |
|---|---|---|---|---|---|---|
| A. Critic Domain | | | | | | |
| **B. Critiquing Approach** | **C. Modes of Critic Feedback** | **D. Critic Rule Authoring** | **E. Critic Realization Approach** | **F. Critic Dimension** | **G. Types of Critic Feedback** | **H. Types of Critic** |
| Comparative | Textual | Insert new critic rule | Rule-based | Active | Explanation | Correctness |
| Analytical | Graphical (visual) | Modify critic rule | Knowledge-based | Passive | Argumentation | Completeness |
| | 3-Dimension (3D) Visualization | Delete critic rule | Predicates | Reactive | Suggestion | Consistency |
| | | Enable / disable critic rule | Pattern-matching | Proactive | Examples (or precedent) | Optimization |
| | | Critic rule authoring facility | Object constraint language (OCL) expression | Local | Simulation | Alternative |
| | | | Programming code | Global | Demonstration | Evolvability |
| | | | | | Interpretation | Presentation |
| | | | | | Positive feedback | Tool |
| | | | | | Negative feedback | Experiential |
| | | | | | Constructive feedback | Organization |
| | | | | | | Design Pattern |

Figure 1. Critic Taxonomy

## III. CRITIC TAXONOMY-GROUPS AND ELEMENTS

There are eight groups in our critic taxonomy and each group consists of several elements. Brief explanation of these groups and their elements are outlined below.

### *A. Critic Domain*

The first group in the critic taxonomy is the Critic Domain. There is no elements associate with this group. A domain is defined "as an area of interest …" [5]. Examples of domains are medical, business process, education, software engineering and architecture, among others. Critics are specified based on the domain knowledge of that particular area. In order to define and specify critics it is required that we understand the domain that we deal with. Only by understanding the domain knowledge will one be able to define and specify meaningful critics. The use and context of critics are varies from one domain to another. Reports from most research [3, 4, 12, 13, 15, and 16] provide either long or short descriptions of critics from different domains. This indicates that critics can be applied to various domains and problems. Furthermore, it provides one of the effective mechanisms in providing critic feedback to users.

### *B. Critiquing Approach*

The Critiquing Approach is the second group of our taxonomy. Elements in this group are comparative and analytical critiquing. Critiquing is a way to generate valid reasoning about a product or action [12]. Reports and articles from [3, 12, 13, 15, 18, and 24] have identified that critic tools commonly use comparative critiquing, analytical critiquing or both as their critiquing approaches.

In a comparative (also known as differential) critiquing, complete and extensive domain knowledge is essential to generate good solutions. When a user recognizes potential problems in a design, the critic system will then produce an optimal result from the predefined solutions in the system. The user-proposed design is then compared with the system's solution. The comparison will result in a report of the differences between the two solutions. Robbins stated a comparative approach can direct users to make their work like the one that the system proposed [15]. Hence, this approach guides the user *to* a known solution [15]. According to [18] the critics authoring in this approach is relatively intuitive and straightforward because it allows critic authors to write down problems and answers, and the system will take care of comparison and feedback generation. For example, TraumaTIQ [2] supports a physician's treatment planning. The TraumaTIQ interprets the physician's goal treatment plan, evaluates the inferred plan structure by comparing it to the system's recommended treatment plan, and finally generates a critique that addresses potential problem [2].

In an analytical critiquing approach, as long as the domain knowledge is sufficient then solutions can be generated. Hence, this approach can be applied to domains where knowledge is incomplete. This approach uses rules to detect potential problems in the design and change it into *assistance opportunities* [15]. Thus, in a way it guides the user *away* from recognized problems [15]. Unlike comparative critiquing, this approach does not generate solutions on its own but instead analyses the user-proposed designs to identify any potential problems from a set of rules. It is not easy to author critics in analytical approach although it is applicable in a broad range of domains [18]. This is because, according to [18], the rules for all the problems in all situations need to be written. Argo [16] is a software design tool example that applies analytical critics. Argo uses analysis predicates to identify undesirable designs and then generates feedback items with more kinds of design context, such as contact information for relevant experts and stakeholders [16].

An example of a tool applying both comparative and analytical critiquing is UIDA (User Interface Design Assistant). UIDA is a system that critiques user interface window layouts [10]. UIDA performs analytical critiquing by applying 72 style rules written in an OPS5-like language and comparative critiquing via recording and comparing the particular set of rules satisfied by each layout [10].

In general, the choice of critiquing approach depends largely on application domain, the characteristics of the task it supports and the cognitive support needs of the user [13].

### C. Modes of Critic Feedback

The third group in our taxonomy is the Modes of Critic Feedback. Elements in this group consist of textual, graphical and 3D visualizations. Presenting critic feedbacks (also known as feedbacks or critiques) is another element to be considered in the design of a critic. Most critics provide critic feedbacks in textual messages. However, graphics can be used as well for presenting critic feedback. Oh et al., [23] recognize three modes used for presenting critiques in existing critic tools: text messages, graphic annotations and three dimensional (3D) visualizations. Text message refers to a critique that is presented in a written form. Graphic annotation refers to a critique that is presented in a graphical form. 3D visualizations involve critiques that are presented via images, diagrams, or animations in a three dimension format. In addition, animation and sound can also be used to provide feedback, in conjunction with one of these other mechanisms.

For instance, [23] developed Design Evaluator, a pen-based critic tool that generates critiques and displays them in textual and visual formats. The architectural floor plan in Design Evaluator display critiques in three ways: as text messages, annotated drawings and texture-mapped 3D models. When a designer selects a text message critique, the tool shows the critiques in two other forms, such as graphic annotation on a designer's floor plan diagram and generates a 3D texture-mapped VRML (Virtual Reality Model Language) model that shows the path via the floor plan [23]. As [4, 24] point out, communicating design information in a mixture of graphical critiques and text critiques is likely to be more effective than selecting one mode.

### D. Critic Rule Authoring

The fourth group in the taxonomy is the Critic Rule Authoring. Elements in this group are: insert new critic rule, modify critic rule, delete critic rule, enable and disable critic rule, and critic rule authoring facility. Critic rules are one of the important components in building critics. In general, critics are specified by a single rule or groups of rules (or procedures) to evaluate different aspects of a product or design in a domain. Thus, a critic development has to involve the writing of critic rules. As [18, 24] argue, critic rules are normally written in advance by the system designers to develop a critic system and it is hard for the user to modify the existing rules or add new critic rules after the critic system is deployed. However, as [13, 24] pointed out, critiquing capacity and issues may need to be adjusted from time to time in various situations. Furthermore, [12] emphasizes that users should not be required to have comprehensive programming knowledge in order to perform the modification of critic rules. For these reasons it is important to allow users to understand the critic rules and be able to modify and expand the rules by authoring new rules to incorporate in critic system. Qiu and Riesbeck [18] have explored the issue of authoring critic rules for educational critic system. They developed an educational critic tool for Java programming, called Java Critiquer. They explored the question of how users can author critic rules. Their Java Critiquer system provides the authoring capability, so that users (teacher) can check or modify the critiques in addition to the feedback that Java Critiquer generates [18]. The tool also allows teachers to gradually enter and update critic knowledge during real use of the system. In addition, some tools allow the user to enable and disable the critic rules via menu configuration. A disabled critic will never be executed until the user enables the critic rules (e.g., ABCDE-Critic, and Argo/UML). The capability for rule authoring is to enable end-user designers to construct and store their own critic rules. A rule authoring facility will allow critics to deal with various conditions and authorize end-user designers to add to the system's feedback process [24].

### E. Critic Realization Approach

The Critic Realization Approach is the fifth group in our taxonomy. This group is about implementing critics by using specific approaches. The elements in this group are: Rule-based, predicate, knowledge-based, pattern-matching, object constraint language (OCL) expressions, and programming code. In order to support critic development, several approaches have been applied to designing and realizing critics. Critics implementation in various domains uses a variety of approaches as outlined below.

Critics implemented with a rule-based approach consist of a condition and an action. If the condition is true, then the action is performed. Actions can include suggestions, explanations, argumentations, messages or precedents of

problems. For instance, ABCDE-Critic [6] uses rule-based expression to specify critics that comment on UML class diagram-based designs. The critic tool invokes critics when a condition clause is found to be true in the current design parts warning a user that the design possibly have error [6].

A knowledge-based approach can be used to specify critics. The knowledge base represents the most important component of a knowledge-based system. The format of the knowledge refers to how this knowledge is represented internally within the knowledge-base system so that it can be used in problem-solving. Several knowledge representation schemes that are commonly used: predicate, rules, frames, associative networks and object. For instance, the IDEA (**I**nteractive **DE**sign **A**ssistant) tool [9] produces design pattern critics implemented with Prolog rules that are directly integrated with a knowledge base. Bergenti and Poggi stated the knowledge base of IDEA is comprised with a set of design rules, corresponding critics, and a set of consolidation rules [9]. The rules for creating the pattern-specific critics are not easy as it requires a high-level of understanding of a design patterns and detailed knowledge of the Prolog and knowledge base structures.

Another approach is to use the pattern-matching. As stated by [22], "a pattern is any arrangement of objects or entities". Basically, a pattern matching process involves an attempt to relate two patterns where one is a theoretical pattern and the other is an operational one [22] or it can consists of left-hand side and right-hand side rules. For instance, the Java Critiquer tool performs automatic critiquing using a pattern matching approach [18]. The left-hand side of a rule is a LMX pattern (Language for Mapping XML). The right-hand side of a rule is a critique. In the Java Critiquer, the pattern is a JavaML pattern for matching JavaML code generated from the Java parser. When a pattern is matched, its matching critique is added right below the problematic Java source code [18].

The predicate logic approach can be used to implement critics. According to [8], predicate logic is based on the idea that sentences can express relationships between objects as well as qualities and attributes of such objects. The argument or terms of the predicate is represented by the objects. The use of terms lets a predicate to express a relationship about various objects rather than just a simple object [8]. Furthermore, predicates can be applied to represent an action or an action relationship between two objects [8]. An example of a tool that applies the predicates approach is the Design Evaluator [23]. The *evaluation* layer in Design Evaluator evaluates sketches with predicates that embody design rules. The tool produces critiques when the rules identify a pattern in the design. The tool checks recognized spatial information with each rule. A design critique is shown in the visualization layer when a rule violation is found [23]. The rules are coded as Lisp predicates that apply to the design objects.

The OCL expression approach is another way to specify critics. Kleppe and Warmer claimed that the Object Constraint Language (OCL) is a language that offers ways to specify the semantics of an object-oriented model in a very accurate style [1]. The semantics are expressed in invariants and pre-and-post conditions, which are all types of constraints [1]. A research of model checking by [14], demonstrated the use of OCL to express constraints via a simple domain-specific language (DSL) called Class Diagrams (CD). As [14] argue, OCL needs extensions to support additional elements such as the severity of a constraint attached to constraints. They classified the severity of a constraint as an *error*, a *warning* or a *critic* [14]. Thus, in their CD example, they show how a critic is specified using an OCL expression.

Apart from the approaches stated above, critics can be realized through the use of programming code. For instance, critics in Argo/UML [17] are coded as Java classes. Class Critic identifies several methods that can be overridden to define and alter a new critic. Each critic's constructor identifies the headline, problem description, and related decision categories [17].

### *F. Critic Dimension*

The sixth group in our critic taxonomy is the Critic Dimension. The elements within this group are based on Fischer's suggestion [11]. Report and articles from [13, 15, 18 and 24] support Fischer's suggestion on critic classification dimensions. The critic dimensions are shown in Table 1.

TABLE I. CRITIC DIMENSION (ADOPTED FROM [11,15])

| **Critic Dimension** | **Brief Description** |
|---|---|
| Active critics | Continuously critique a user's design or task. |
| Passive critics | Wait until a user asks for a critique |
| Reactive critics | Critique on the design or task that the user has done. |
| Proactive critics | Guide the user by presenting guidelines before the user make a decision. |
| Local critics | Critics that evaluate individual design elements. |
| Global critics | Critics that consider interactions between most or all of the elements in a design. |

In a critic development, a critic designer has to consider the use of active critics and passive critics. An active critic usually continuously monitors the user tasks and warns the user as soon as a critic is violated and then offers a critic feedback. An active critic makes the user aware of their unsatisfactory design when the potential problem is easy to correct [11]. However, as [11] argues some users may find it a disturbance to have something continuously criticise them without giving them an opportunity to develop their own design or task.

In contrast, a passive critic only works when a user asks for a check of critic violation. Normally, after the user completes preliminary design, the user then asks for evaluation of the design. Passive critics are less intrusive compared to active critics because they allow the user to control when to activate the critics. The problem with passive critics is that most of the time the user does not activate them early enough to prevent potential problems [18]. Argo/UML

provides active critics when a user attempts to draw a design diagram. For example, when a user selects a new class to place in the class diagram design, several critics trigger to indicate that part of the design has been started, but still not completed [17]. Whereas, the Java Critiquer uses passive critics to allows students to concentrate on their programming tasks without interruption [18].

There are critic tools that employ either reactive or proactive critics. A reactive critic provides critiques on the user's accomplished design, whereas a proactive critic attempts to lead the user before the user makes any specific action decision. Similar to these two approaches is the critic dimensions suggested by Silverman [3] i.e. *before*, *during* and *after*. Silverman's *before critic* is similar to Fischer's proactive critic. *During* and *after critics* can be viewed as Fischer's reactive critics. However, *during* and *after* critic is different in terms of whether a user's work is completed or not. The SEDAR [19] tool adopts Silverman's three dimensions: *before* (error prevention), *during* (design review critic, design decision) and *after* (error detection). The Heuristic Requirements Assistant (HeRA) [7] tool provides proactive support because while a user is typing the requirements, it analyzes the input and warns the user of any ambiguities or incomplete specification detected [7].

Finally, critics can be classified as either local or global critics. Local critics are critics that evaluate individual design elements and global critics involve the interactions between most or all of the elements in a design [15]. For instance, the HeRA [7] tool provides users with local and global critics. The local critics of the tool is concerned with the current focus of the requirements editor (i.e. requirements, use cases, and a glossary), while the global critics allow users to analyze a global perspective in terms of list of all critiques and inference of global process diagrams (i.e. UML Use Case Diagram, Event-driven Process Chain models, and Use Case Point View) [7].

### G. Types of Critic Feedback

The next group in our taxonomy is the Critic Feedback. There are ten elements in this group: explanation, argumentation, suggestion, example (or precedent), interpretation, simulation, demonstration, positive feedback, negative feedback, and constructive feedback. There are many ways to present critic feedback (also known as feedback or critique) to users. Critic tools can offer critic feedback to users by choosing the appropriate techniques from the ten elements. However, the most widely used techniques are explanation, suggestion, and argumentation.

Explanations technique is widely used in most critic tools. Explanation as defined in a Cambridge dictionary is "details or reasons that someone gives to make something clear or easy to understand" [5]. Thus, critics must produce explanations so that the user has the chance to assess the details and reasons before making a decision as whether to accept the critique generated by the tool. The explanations can be focused on the violations of general guidelines or the differences between the user's design solution and system's solution [12]. Furthermore, it is essential to validate a critique via explanation because without details or reasons, a user will not accept the critique. Explanations can be simple or in-depth.

Argumentation is another option for offering critic feedback. It is also another mechanism for explanation where it can contains issues, answers, and arguments about a product or design domain. A user, who may not understand critiques offered by a critic tool, may wants to know more information about the critiques. Thus, via an argumentation component, the user can obtain the required information to justify the critique. An example of argumentation style is shown in the ABCDE-Critic tool. The ABCDE-Critic incorporates an *argumentative hypermedia system* to provide in-depth explanation for user that does not understand or wants more information about critics [6]. The argumentation component contains issues, answers and arguments about the design domain [6].

Some critics offer alternatives or suggestions to the user's solution. The suggestion style is also known as *solution-generating critics* [12] which are capable of suggesting alternatives to the user's solution. An example is the JANUS system where a problem detecting critic shows that there is a stove close to a door [12]. Another option is to provide examples (precedents) to support critics. Examples are a way of helping users to understand something by showing them how it is used. For example, the Design Evaluator [23] provides an exemplar Web page for the designer to look at when a critique is selected.

Other ways for presenting critic feedback are either to provide positive or negative feedback. A positive feedback provides a critique in a praising way when a user produces a good design. A negative feedback is a complaint when a user produces a poor design. Positive and negative feedback is related to human's evaluation that normally based on advantages and disadvantages, good and bad. Apart from the styles stated above, critic feedback can be presented through the use of a simulation component to allow users to carry out "what-if" analysis (e.g., JANUS, HeRA), interpretation from a certain perspective, and constructive feedback (e.g., Argo/UML). A combination of styles in presenting critic feedback certainly facilitates users to clarify their understandings, as well as improve their knowledge.

### H. Types of Critic

Finally, the last group in our taxonomy is the Types of Critic. Critics can be classified according to the type of domain knowledge that they present [15, 16]. Thus, the Critic Domain group and the Types of Critic group complement to each other. Table II shows the list of critic types. According to Robbins, those critics are descriptive rather than definitive [15]. In fact, new categories can be defined based on the application domain. For instance, IDEA [9] offers pattern-specific critics to assist the architects in finding and improving the realizations of design patterns in UML designs.

TABLE II. CRITIC TYPES (ADOPTED FROM [15])

| Critic Types | Brief Description |
|---|---|
| Correctness critics | identify syntactic and semantics flaws |
| Completeness critics | remind the designer to finalize design works |
| Consistency critics | show contradictions within the design |
| Optimization critics | advice better values for design parameters |
| Alternative critics | prompt the architect to consider options to a specified design decision |
| Evolvability critics | deal with issues such as modularization, that affect the effort needed to modify the design over time |
| Presentation critics | search for awkward use of notation that reduces readability |
| Tool critics | notify the designer of other accessible design tools at the times when those tools are useful |
| Experiential critics | offer reminders of previous experiences with similar designs or design elements |
| Organization critics | express the importance of other stakeholders in the development organization |

## IV. CONCLUSIONS

We proposed and illustrated a new critic taxonomy based on several aspects that characterize critics (or critiquing systems). These aspects are gathered widely from the critic literature. Our critic taxonomy identifies eight groups: critic domain, critiquing approach, modes of critic feedback, critic rule authoring, critic realization approach, critic dimension, types of critic feedback, and types of critic.

The utility of our critic taxonomy is manifold: 1) to provide an overview of critic research, 2) to identify and distinguish key critic elements, and 3) to recognize techniques or methods applied in critics. We hope that this taxonomy will provide meaningful way of describing and reasoning about critics. We also believe that our critic taxonomy will be useful in guiding the critic developer towards realizing robust critic capabilities by comparing and contrasting different critic dimensions.

We have applied our taxonomy to ten tools that have critic support. The mapping of the tools to our critic taxonomy shows that the practice of critics is supported by the critic taxonomy. Furthermore, this critic taxonomy development has assisted us in identifying the needs of our own visual critic authoring research and tools [20]. Hence, we have introduced a new approach to implement critics which we call a template-based approach [20].

In the future, we are planning to improve our critic authoring tool by considering the elements defined in our taxonomy. We also plan a larger evaluation with end-users experimenting and applying our critic tool.

## REFERENCES


[1] A. Kleppe and J. Warmer, "The Semantics of the OCL Action Clause", in C. Tony & J.Warmer (Eds.), Object Modeling with the OCL, LNCS, vol.2263, Springer-Verlag Berlin Heidelberg, 2002, pp. 213-227.

[2] A. S. Gertner and B. L. Webber, TraumaTIQ:online decision support for trauma management, IEEE Intelligent Systems, pp. 32-39, 1998.

[3] B. G. Silverman, Survey of expert critiquing systems:practical and theoretical frontiers, Communications of the ACM, vol.35(4), pp. 106-127, April 1992.

[4] B. G. Silverman and T.M. Mehzer, Expert critics in engineering design: lessons learned and research needs, AI Magazine, vol.13(1), (1992) (AAAI).

[5] Cambridge Dictionary, http://dictionary.cambridge.org/

[6] C. R. B. Souza, J.S. Ferreira Jr, K.M. Goncalves, and J. Wainer, A group critic system for object-oriented analysis and design, In Proceedings of the 15th IEEE Conference on Automated Software Engineering, IEEE Press, 2000, pp. 313-316.

[7] E. Knauss, D. Lubke, and S. Meyer, Feedback-driven requirements engineering: the heuristic requirements assistant, In Proceedings of IEEE 31st International Conference on Software Engineering, 16-24 May 2009, pp. 587-590.

[8] E. Tyugu, Algorithms and architectures of artificial intelligence, Frontiers in AI and Applications, vol. 159, IOS Press, 2007.

[9] F. Bergenti and A. Poggi, Improving UML designs using automatic design pattern detection, In Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering (SEKE), 2000, pp. 336-343.

[10] G. A. Bolcer, User interface design assistance for alrge-scale software development, Automated Software Engineering, vol.2(3), pp. 203-218, September 1995.

[11] G. Fischer, Human-computer interaction software: lessons learned, challenges ahead, IEEE Software, vol.6, pp.44 – 52, January 1989.

[12] G. Fischer, A. C. Lemke, and T. Mastaglio, Critics: an emerging apporach to knowledge-based human computer interaction, International Journal o Man-Machine Studies, 35, pp. 695-721, 1991.

[13] H. Irandoust, 2006. Critiquing systems for decision support. DRDC Valcartier TR 2003-321. http://pubs.drdc.gc.ca/PDFS/unc44/p524782.pdf.

[14] J. Bézivin and F. Jouault, Using ATL for checking models. Electronic Notes in Theoretical, Computer Science, 152, Elsevier, 2006, pp. 69–81.

[15] J. E. Robbins, 1998. Design critiquing systems, Technical Report UCI-98-41. http://www.ics.uci.edu/~jrobbins/papers/CritiquingSurvey.pdf

[16] J. E Robbins, and D. F. Redmiles, Software architecture critics in the Argo design environment. Knowledge-Based Systems 11(1), 1998, pp. 47-60.

[17] J. E Robbins, and D. F. Redmiles, Cognitive support, UML adherence, and XMI interchange in Argo/UML, Information and Software Technology, vol.42(2), pp. 79-89, January 2000.

[18] L. Qiu, and C. K. Riesbeck, "An incremental model for developing educational critiquing systems: experiences with the Java Critiquer", Journal of Interactive Learning Research, 2008(19), pp.119-145.

[19] M. C. Fu, C. C. Hayes, and E. W. East, SEDAR: Expert critiquing system for flat and low-slope roof design and review, Journal of Computing in Civil Engineering, vol.11(1), pp. 60 – 68, January 1997.

[20] N. M. Ali, J. Hosking, J. Huh, and J. Grundy, Template-based critic authoring for domain-specific visual language tools, In Proceedings of the 2009 IEEE Symposium on Visual Languages and Human-Centric Computing, Corvallis, Oregon, USA, pp. 111-118.

[21] P. L. Miller, Expert critiquing systems: practice-based medical consultation by computer. Springer Verlag, New York, 1986.

[22] W. M. K. Trochim, Outcome Pattern Matching and Program Theory, Journal of Evaluation and Program Planning, vol.12(4), pp. 355-366, January 1989.

[23] Y. Oh, E.Y.-L. Do, and M.D. Gross, "Intellligent critiquing of design sketches", in JL Randall Davis, T Stahovich, R Miller and E Saund (Eds), Making Pen-based Interaction Intelligent and Natural, The AAAI Press, Arlington, Virginia, 2004, pp 127-133.

[24] Y. Oh, M.D. Gross, and E.Y.-L. Do, Computer-aided critiquing systems, lessons learned and new research directions, In Proceedings of the 13th International Conference on Computer Aided Architectural Design Research in Asia, Chiang Mai (Thailand) 9-12 April 2008, pp.161-167.