COMPUTER-AIDED CRITIQUING SYSTEMS: Lessons Learned and New Research Directions

YEONJOO OH, MARK D GROSS
CoDe Lab, School of Architecture, Carnegie Mellon University, USA
{yeonjoo, mdgross}@cmu.edu

AND

ELLEN YI-LUEN DO
College of Architecture and College of Computing
Georgia Institute of Technology, USA
ellendo@cc.gatech.edu

Abstract. A critiquing system helps designers improve their design artifacts by providing feedback. Computer-aided critiquing systems have been built in many fields and provide us with useful lessons. In this paper we analyze existing critiquing systems in terms of (1) critiquing process, (2) critiquing rules, and (3) intervention techniques. Based on this analysis, we suggest new research directions for critiquing systems in the domain of architectural design.

1. Introduction

A computer-aided critiquing system analyzes proposed design solutions and provides designers with feedback. This paper reviews existing systems in terms of three aspects: critiquing process, critiquing rules, and intervention techniques.

1.1. DEFINITIONS OF CRITIQUING SYSTEMS

A brief look at how one-on-one critiquing sessions function in design studio can help us form a better picture of how computer assisted critiquing might work in architectural design. A studio teacher observes students' progress by looking at their drawings and models and listening to their descriptions of their design. The teacher comments by interpreting and evaluating the design, offering alternatives and precedents or asking questions to raise new issues that students may not have thought about. The student reflects on and modifies the design based on these critiques .

Automated critiquing systems have been built to support design in different domains. Most critiquing systems focus on finding errors or problems in proposed designs. For instance, Silverman (1992) defines critiques as "*What the program thinks is wrong with the user-proposed solution*" In this view, a critiquing system corrects user's mistakes at hand.

Others researchers see critiquing systems a little differently. Fischer et al. (Fischer et al., 1991b) state that "*Critics operationalize Sch n's concept of a situation that talks back*" Robbins (1998) explains: "*A design critic is an intelligent user interface mechanism embedded in a design tool that analyzes a design in the context of decision-making and provides feedback to help the designer improve the design.*" In this view, critiquing systems not only offer negative critiques, but also help designers improve their solutions with "*constructive*"

feedback. In summary, a design critiquing system is a tool that analyzes a work-in-progress and provides feedback to help a designer improve the solution. It may ask relevant questions, point out errors, suggest alternatives, offer argumentation and rationale, or (in simple and obvious cases) automatically correct errors.

1.2. WHY REVIEW CRITIQUING SYSTEMS?

Although critiquing is essential in traditional architectural design, relatively few critiquing systems for architecture have been built. These architecture critiquing systems support only checking building codes. For instance, Singapore's CORENET system (2004) reads architectural drawings to check building codes and regulations such as fire safety requirements. The system provides graphic annotations (red circles) on CAD drawings to indicate problematic parts and generates a text document with the list of errors.

Existing critiquing systems for architectural design such as CORENET (2004), ICADS (Chun and Ming-Kit Lai, 1997), or Solibri Checker (Solibri.Inc., 2007) only point out errors. They are not tightly integrated with design process nor do they offer 'constructive' feedback to provide opportunities to improve designs. We believe that critiquing systems have potential to support architectural design beyond simply checking for errors. Much can be learned from existing critiquing systems in various design domains. Critiquing systems in civil engineering, medical treatment planning, and programming provide several implementation strategies such as timing, activation, modalities of feedback and types of feedback that would also be useful in the context of architectural design..

1.3. REVIEW CRITERIA

We chose critiquing systems that (1) support a human designer in making things, (2) store design knowledge to automatically detect parts or aspects of designs that can be improved, and (3) provide machine-generated feedback on design artifacts at hand.

Studio teachers often refer students to building precedents, which can be considerd a form of critiquing. Various Case-Based design systems have been built, such as Archie (Pearce et al., 1992) and DYNAMO (Neuckermans et al., 2007). We do not consider a Case-Based Design Aid as a critiquing system although a critiquing system could certainly present a designer with cases relevant to the task at hand.

This survey is organized into four sections. Section 2 identifies aspects of critiquing system research. Section 3 suggests new directions for critiquing systems in architectural design. We then conclude with a summary.

2. Aspects of Critiquing Systems

We review existing critiquing systems and identify three aspects of systems that would be useful for building critiquing systems for architectural design: (1) the process of critiquing, (2) the rules used by the system to trigger critiques, and (3) the techniques to decide when and how to intervene.

2.1. CRITIQUING PROCESS

All critiquing systems assume a simple cycle: the detection of problems and subsequent design improvement based on the offered criticism. In his survey of critiquing systems, Robbins (1998) identifies five phases: Activate – Detect – Advise – Improve – Record. Although the systems we review do not all cover all phases, his model can be a good starting point to look at critiquing systems. The Activate phase enables/ disables critiquing rules to support a user's current tasks. The Detect phase identifies problems by comparing a user's work with critiquing rules. The Advise phase informs users of the detected conflicts. The Improve phase provides

suggestions about how to fix the indicated problems. Finally, the Record phase records how designers resolve breakdowns based on the critiques offered.

We modify Robbins' model to situate critiquing systems in architectural design. We have added a Construct phase into Robbins's model because an environment where designers make things is essential. We have merged Robbins' two phases (Advise – Improve) because critiques include various types of feedback in one-on-one critiquing sessions: they point out mistakes, they demonstrate how to fix errors, or they make suggestions for subsequent design moves (Sch n, 1985). Our model is composed of Construct – Parse – Check – Critique – Maintain. In the Construct phase, designers make drawings to find solutions. For instance, Janus supports placing kitchen appliances in the working window (Fischer et al., 1989). Likewise, architects sketch a floor plan diagram in Design Evaluator (Oh et al., 2004). In the Parse phase a system converts a drawing into a symbolic representation of recognized elements and spatial relationships. In the Check phase, the system finds problematic parts by comparing the symbolic representation with previously stored rules. In the Critique phase, the system offers feedback to help users understand the status of their designs and indicate problems that may be improved. Finally, in the Maintain phase, the system records how designers revise their designs based on critiques and updates a user model or a task model.

2.2. CRITIQUING RULES

2.2.1. Forms of Rules

All the critiquing systems we reviewed are rule-based, where rules are defined in a '*predicate-action*' format. *Predicates* represent particular situations in design solutions such as <*dishwasher is placed in the left side of sink*>. When the situation is found in a user's design, the defined set of *actions* is invoked such as <*notifying a conflict has been detected*>. *Actions* include argumentations, suggestions, precedents, interpretation or priorities/ importance of problems.

2.2.2. Completeness of Knowledge

Existing systems commonly employ what Robbins (1998) calls 'comparative critiquing', 'analytic critiquing' or both. Comparative critiquing uses complete and extensive domain knowledge to generate presumably good solutions. In this approach, the system develops solutions by applying the stored rules and the specified problems by users. It then compares a user's work against generated solutions and reports the differences between them. TraumaTIQ (Gertner and Webber, 1998) supports a physician's treatment planning. It infers goals from treatment plan and generates its own plan. It then detects differences between physician's plan and the generated one. A comparative critiquing is more suitable for well-structured domains than ill-structured ones. Alternately, analytic critiquing requires only that the system has sufficient knowledge to detect possible problems. It offers critiques by evaluating a user's solution against the stored rules. For example, the Janus family evaluates users' kitchen layouts against constraints (Fischer et al., 1989). The analytic critiquing supports exploratory problem solving better than the comparative critiquing because design problems seldom have one right answer.

2.2.3. Management of Critiques (Critiquing Rules)

Some systems manage critiquing rules to offer relevant and timely feedback according to users, tasks and goals. They use specific representations to control how rules (critiques) are activated: (1) a model of the task the user is engaged in; (2) a model of the particular user who is doing the task; and (3) a model of the user's goals.

The purpose of a *task model* is to provide relevant and timely critiques to the task at hand. SEDAR (Fu et al., 1997) supports critiques of a roof design based on constructability standards. It models tasks (e.g. roof component layout, equipment layout, etc.) in a hierarchical structure

to infer which critiquing rules are relevant to the current situation and provides feedback appropriate to the task at hand.

The Argo system supports software design with user constructed UML (Unified Modeling Language) diagrams that represent software components and interactions among them. When Argo detects conflicts against previously defined rules, it offers feedback. There, the user must explicitly choose the current tasks (e.g. system typology, component selection, etc.) from the listed tasks (Robbins and Redmiles, 1998). The system activates only critiquing rules that match those tasks that the user selected. For instance, when a user indicates that she is making a rough organization, critiques related to details should not be active.

A *user model* enables systems to adapt to a particular designer's preferences, knowledge and past actions, tailoring explanations according to an individual user's level of expertise. Or a user model may control rules by considering the user's preferences. For instance, Lisp-Critic (Mastaglio, 1990) utilizes this user model representing a programmer's understanding of Lisp, and usage preferences. For instance, when a programmer prefers to use the function named 'first' over the more traditional form 'car', Lisp-Critic turns off the rule that triggers the critique: 'car-to-first transformation'.

A user may create a *goal model* by specifying his/her task goals besides developing design solutions. This tells the system what the user is trying to accomplish. In Janus a user fills out a form to enter goals (Fischer et al., 1989, Fischer et al., 1991a), so the system would activate only rules relevant to the specified design goals. For example, if a user defines the goal model to be a kitchen for one person, Janus deactivates critiquing rules associated with family kitchen design.

2.2.4. End User Critiquing Rule Authoring

Rules in most critiquing systems are written by system designers in advance. Once written, there is no easy way for the user to adjust the established rules or to incorporate new rules. However, critiquing scope and contents may need to be changed from time to time in various situations. This insight has led several researchers to explore rule authoring. For instance, Qiu and Riesbeck (2004) explored the question of how users can create critiquing rules. An interesting feature of their Java Critiquer, a system to teach programming, is to integrate authoring with usage of the critiquing system, so that a teacher can review or modify the critiques that are generated in a feedback process. The teacher can insert critiques in addition to the feedback that Java Critiquer generates. Over time, teachers gradually extend knowledge-base by documenting predicates and the associated critiques they trigger. Rule authoring improves the accuracy, relevance and scope of critiquing. It enables users to store their own rules. It is an important feature that enables systems to deal with diverse situations. Rule authoring empowers designers to participate in the system's feedback process.

2.3. INTERVENTION TECHNIQUES

2.3.1. Timing

An important aspect of intervention is timing—when the software offers critiques. Fischer (1989) identifies a classification dimension: reactive and proactive. Reactive critiquing offers feedback on the work that the designer has done. Proactive critiquing guides the designer by presenting guidelines, before s/he makes a design move. Silverman (1992) identifies another dimension: before, during and after. Before critiquing corresponds to Fischer's proactive critiquing. During and after critiquing can be considered as reactive. The difference between during and after critiquing is whether a designer's task is completed. SEDAR adopts Silverman's dimensions and takes all three strategies: before (error prevention), during (design review critic, design suggestion) and after (error detection) (Fu et al., 1997). Most building code checking systems in architecture provide *reactive* critiques *after* a designer has finished

his/her work. For instance, ICADS (Chun and Ming-Kit Lai, 1997) checks building codes (e.g. fire exit) and rules-of-thumb of interior design, *after* all design decisions have been made.

2.3.2. Activation

Fischer (1989) has identified two activation strategies: active and passive. Active critiquing continuously monitors design moves and offers feedback. Passive critiquing provides feedback when a designer requests it. Anderson et al. (1995) studied how differently users respond in these two activation settings. While using the passive setting, users do not ask for evaluation until they have completed a preliminary solution. In the active setting most users fixed the errors immediately 80% of the time. Their experiments show us that active critiquing can be a better activation method. However, active critiquing may distract users in their designing tasks.

2.3.3. Modalities of Critiques

We identify three modalities used in existing systems: text messages, graphic annotations and 3D visualizations. Most systems provide feedback in a written form. Several critiquing systems provide visual critiques along with text messages. For instance, Design Evaluator (Oh et al., 2004) makes graphic annotation on a designer's floor plan diagram and generates a 3D texture-mapped VRML (Virtual Reality Model Language) model. It annotates paths through 3D floor plan. The DAISY system provides graphic annotation on a UML diagram (Souza et al., 2003). Some building code-checking systems illustrate results in 3D space by circling problematic parts in red (Han et al., 2002, Xu et al., 2004). Several researchers emphasize the strength of graphic annotations on drawings and 3D models. Based on his usability tests, Fu (1997) argues that graphic annotations help designer understand the offered critiques better because: designers are working with drawings; and it is harder for designers to relate text critiques in a separate window than seeing graphic elements appear in their drawings.

2.3.4. Types of Feedback

Critiquing systems offer negative evaluation as well as positive critiques, explanations, argumentations, suggestions, examples (precedents), and interpretations. The Janus system praises the good aspects of a kitchen layout so that designers might be more likely to retain them in further revision (Fischer et al., 1991a). Some systems offer critiques with detailed explanations (Souza et al., 2003) and argumentation about why particular parts are desirable or problematic (Fischer and Morch, 1991b). Some offer suggestions to help design revisions. KID (Knowing-in-Design) provides precedents as potential solutions that could be graphic cues for further design moves (Nakakoji et al., 1998). Some also interpret design solutions from a specific viewpoint; for instance, Stanford's building code checker provides 3D VRML models to predict building user's movements and the performance of a disabled user with wheelchairs (Han et al., 2002).

3. New Research Directions of Critiquing systems

OUR DISCUSSION ABOVE SUGGESTS SEVERAL PROMISING RESEARCH DIRECTIONS.

Integration with design task: Critiquing systems should be tightly integrated with design tasks while users are situated in their tasks. Currently, most critiquing systems in architecture support the solutions only after design tasks have been done, such as code checking. We believe that critiquing is more powerful when situated in the designing process.

Intervention techniques: Intervention should be based on observations of the way architects and studio instructors critique in practice. To date, critiquing systems provide evaluations, suggestions, precedents, and interpretations. They do not change critiquing types and modalities according to user's individual differences, whereas human critics do. Furthermore, people ask

questions about design solutions. Such questions encourage designers to elaborate on their reasoning and decisions and help them discover design problems by themselves. More sophisticated intervention techniques can mitigate the negative connotation of critiquing systems. For example, rather than tell the designer "that doorway is too small", a system might point to the doorway and ask, "could you get a wheelchair through that door?" Intervention techniques and user model that have been explored in the research community of human-computer interaction can be adapted to critiquing systems. A critiquing system can provide feedback in a different form (using visual aids, asking questions, offering negative and positive evaluation, or offering a detailed description) that reflects the differing needs of each user.

System engineering opportunities: Another opportunity involves system engineering issues such as critique authoring and development of a software toolkit for critiquing systems. Our proposed process model may be a useful foundation. In this toolkit, one would design a system by entering design knowledge, designing interfaces and interactions (critiques).

4. Summary

We have reviewed several aspects of critiquing systems (critiquing process model, critiquing rules, and intervention techniques) and presented a critiquing process model, Construct – Parse – Check – Critique – Maintain. We have discussed four topics of critiquing rules by describing forms of rules, completeness of knowledge, management of critiquing rules and end user rule authoring. Also we have described intervention techniques such as timing, activation, modalities of critiques, and types of feedback. These aspects can be used constructively in a design of a critiquing system. Furthermore, we have suggested new research directions. First, critiquing systems need to tightly integrate with in-progress design tasks. Second, more sophisticated intervention techniques need to be developed to provide more relevant and useful feedback to users. Third, a software toolkit for critiquing systems can be developed based on our process model.

Acknowledgements

This research was supported in part by the National Science Foundation Grant ITR-0326054.

References

Anderson, J. R., Corbett, A. T., Koedinger, K. R. & Pelletier, R. (1995) Cognitive Tutors: Lessons Learned. *J. Learning Sciences,* 4(2)**,** 167 - 207.

Chun, H. W. & Ming-Kit Lai, E. (1997) Intelligent Critic System for Architectural Design. *IEEE Transactions on Knowledge and Data Engineering,* 9(4)**,** 625 - 639.

Corenet (2004) Corenet e-Plan Check System. Retrieved 5th. Dec. 2007, http://www.corenet.gov.sg/corenet/

Fischer, G. (1989) Human-Computer Interaction Software: Lessons Learned, Challenges Ahead. *IEEE Software,* 6(1)**,** 44 - 52.

Fischer, G., Lemke, A. C., Mastaglio, T. & Morch, A. I. (1991a) The Role of Critiquing in Cooperative Problem Solving. *ACM Trans Inf Syst,* 9(2)**,** 123 - 151.

Fischer, G., Lemke, A. C., Mccall, R. & Morch, A. I. (1991b) Making Argumentation Serve Design. *Human Computer Interactions,* 6(3-4)**,** 393 - 419.

Fischer, G., Mccall, R. & Morch, A. (1989) JANUS: Integrating Hypertext with a Knowledge-Based Design Environment. *ACM Conference on Hypertext and Hypermedia.* Pittsburgh, PA, ACM Press. 105 - 117.

Fu, M. C., Hayes, C. C. & East, E. W. (1997) SEDAR: Expert Critiquing System for Flat and Low-slope Roof Design and Review. *J. Computing in Civil Eng* 11(1)**,** 60 - 68.

Gertner, A. S. & Webber, B. L. (1998) TraumaTIQ: online decision support for trauma management. *IEEE Intelligent Systems,* 13(1)**,** 32 - 39.

Han, C. S., Law, K. H., Latombe, J.-C. & Kunz, J. C. (2002) A Performance-Based Approach to Wheelchair Accessible Route Analysis. *Advanced Engineering Informatics,* 16 (1)**,** 53-71.

Mastaglio, T. (1990) User Modeling in Computer-based Critics. *IEEE The 23rd Annual Hawaii International Conference on System Sciences.,* IEEE Press. 403 - 412.

Nakakoji, K., Yamamoto, Y., Suzuki, T., Takada, S. & Gross, M. D. (1998) From Critiquing to Representational Talkback: computer support for revealing features in design. *Knowledge-Based Systems,* 11(7-8)**,** 457 - 468.

Neuckermans, H., Wolpers, M., Casaer, M., & Heylighen, A. (2007) Data and Metadata in Architectural Repositories. *The 12th Intern'l Conf on Computer Aided Architectural Design Research in Asia (CAADRIA),* Nanjing, China, 489 - 497

Oh, Y., Do, E. Y.-L. & Gross, M. D. (2004) Intelligent Critiquing of Design Sketches. In DAVIS, R. (Ed.) *American Association for Artificial Intelligence Fall Symposium - Making Pen-based Interaction Intelligent and Natural.* The AAAI Press. 127 - 133.

Pearce, M., Goel, A. K., Kolodner, I. L., Zimring, C., Sentosa, L., Billington, R. (1992) Case-based Design Support: A Case Study in Architectural Design, *IEEE Expert,* 7(5), 14-20.

Qiu, L. & Riesbeck, C. K. (2004) Incremental Authoring of Computer-based Interactive Learning Environments for Problem-based Learning. *IEEE Intern'l Conf. on Advanced Learning Technologies (ICALT).* Finland, IEEE Press. 171 - 175.

Robbins, J. E. (1998) Design Critiquing Systems. *Tech Report UCI-98-41.* Department of Information and Computer Science, University of California, Irvine.

Robbins, J. E. & Redmiles, D. F. (1998) Software Architecture Critics in the Argo Design Environment. *Knowledge-Based Systems,* 11(1)**,** 47 - 60.

Sch n, D. A. (1985) *The Design Studio,* London, RIBA.

Silverman, B. G. (1992) *Critiquing Human Error: A Knowledge Based Human-Computer Collaboration Approach,* Academic Press.

Solibri.Inc. (2007) The World Leader In Design Spell Checking. http://www.Solibri.Com/

Souza, C. R. B. D., Oliveira, H. L. R., Rocha, C. R. P. D., Goncalves, K. M. & Redmiles, D. F. (2003) Using Critiquing Systems for Inconsistency Detection in Software Engineering Models. *The Fifteenth Intern'l Conf on Software Engineering and Knowledge Engineering (SEKE 2003).* San Francisco, CA, USA. 196 - 203.

Xu, R., Solihin, W. & Huang, Z. (2004) Code Checking and Visualization of an Architecture Design. *IEEE Visualization* IEEE Computer Society. 10 - 11.