Process Support to Help Novices Design Software Faster and Better

Aaron G. Cass
Department of Computer Science
Union College
Schenectady, NY 12308
+1 518 388-8051
cassa@union.edu

Leon J. Osterweil
Department of Computer Science
University of Massachusetts
Amherst, MA 01003-4610
+1 413 545 2013
ljo@cs.umass.edu

ABSTRACT

In earlier work we have argued that formal process definitions can be useful in improving our understanding and performance of software development processes. There has, however, been considerable sentiment that formalized processes cannot capture the creative process of software design. This paper describes our experimentation with the hypothesis that both design speed and design quality can be improved through the use of formalized process definitions. Our experimentation supports this hypothesis.

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: Object-oriented design methods; D.2.9 [**Management**]: Productivity

General Terms

Experimentation

Keywords

Consistency Rules, Process Programming, Software Design Process

1. INTRODUCTION

Others have observed [10] that software design, as practiced by experts, is a very iterative process. The designer undertakes a series of activities designed to arrive at a complete and consistent design model. Each activity will involve work on one or more parts of the model with the goal of adding to the model in such a way that it is still internally consistent and consistent with the requirements. After each activity, the design may be inconsistent. The designer must then decide both the task to perform next and which part of the system to perform it on. The designer continues this iteratively, until the design is complete.

The expert's iterative process can be viewed as a series of opportunistic responses to design inconsistency – when inconsistencies are noticed, the expert reworks their causes. We propose an approach to help novice designers progress in design by providing a partially-automated process to give this kind of guidance, helping the novice deal with the potentially overwhelming choices arising at each iteration.

Clearly, such a process should not be too rigid because design requires creativity and insight. However, some process rigidity seems necessary, as design also requires a great deal of (potentially automatable) clerical work. Further, the rigidity of a process tailored to help with a particular design pattern or architectural style should be useful because it can incorporate specialized pattern-specific steps and consistency rules to guide the novice.

We propose and evaluate the hypothesis that by doing so we can 1) guide novice designers to produce better designs and 2) reduce the time required to do so.

2. RELATED WORK

Design is considered to be a creative activity, requiring many mid-course changes in plan of attack. Visser [10] observed that an engineer did not follow his intended design plan, instead adopting an opportunistic approach. Because this opportunism seems inherent in the nature of design, this activity has been deemed difficult to capture with a rigid, formal process.

Thus, tool-based approaches to helping designers tend to focus on the artifacts produced instead of the process producing them. The Argo [8] environment uses general-purpose design critics [9] to check design artifacts against consistency rules, giving feedback when the relations among these artifacts are inconsistent. The Aesop [4] environment more strictly enforces rules by disallowing design artifacts incompatible with the architectural style being followed. Both systems provide guidance on artifact structure without direct guidance about the process to follow to achieve that structure. This seems to risk overwhelming novice designers with too many choices.

Additionally, as noticed by Garlan, et al [4], often the only way to transform a design from one consistent state to another is via an inconsistent intermediate state. If we enforce all rules at all times, the rules would then have to be weakened to allow these intermediate states. Even if the rules only provide warnings, the amount of warning feedback again risks overwhelming the novice designer.

In previous work, we advocated formalizing software processes as process programs [7]. To support this, we have developed a process-programming language called Little-JIL [12, 11] and an interpreter for it called Juliette [1] and have used both to encode and execute various complex processes. We have further argued that software design processes can be defined using a suitably-flexible

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ASE'05 November 7–11, 2005, Long Beach, California, USA.
Copyright 2005 ACM 1-58113-993-4/05/0011 ...$5.00.

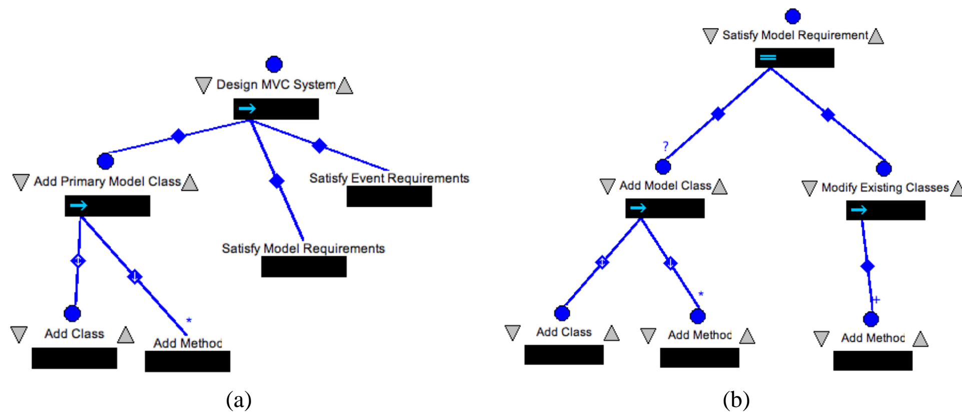

Figure 1: MVC Design Process

process program [2]. We have also used the flexibility achievable with Little-JIL to formalize the common activity of rework in design processes [3]. This experience convinces us that design processes can be encoded and executed while still providing the flexibility needed.

3. OUR APPROACH

Our approach is to augment the Aesop/Argo consistency rule approach with process guidance provided by a Little-JIL process program executed with Juliette. This paper describes experiments with a process program and consistency rules for the Model-View-Controller (MVC) [5] design pattern, chosen because there are many related elements whose relationships can be see as consistency rules.

3.1 Process Guidance

A local designer with extensive experience in developing MVC software helped us to design a process for designing MVC systems. Informally, the process begins by guiding users to satisfy those requirements addressing the storage of application data, and then goes on to guide users to address the event system that keeps views updated.

While the focus of this paper is not our process language Little-JIL, we present in Figures 1 and 3 the Little-JIL formalization of the process to clarify aspects of the process and show that the process is formally understood. As a Little-JIL process program, it is a hierarchy of steps defining the allowable orders of execution of the individual steps in the process. The symbol in a step's black bar indicates the allowable orders of its sub-steps. For example, the root step Design MVC System is a *sequential* step, meaning that its sub-steps must be executed left to right. Thus, the process mandates that the designer first add the primary model class (an activity which is decomposed into two steps to be executed sequentially) and then satisfy the model requirements. As indicated by the parallel lines symbol in Figure 1(b), satisfying a model requirement is a parallel activity[1]. Thus, while we add a new model class we can also modify existing ones. Notice also the *cardinality* symbol on the edge above Add Model Class. The question mark indicates that the step is optional. Add Method in the left side of Figure 1(b) can be executed zero or more times (shown with a *), while the other reference to Add Method can be executed one or more times (shown with a +).

[1]Note that we have left out a full elaboration of Satisfy Model Requirements from Figure 1(a). This is also a parallel step, in this case with children that are instances of Satisfy Model Requirement, one for each of the requirements for the system being developed. Due to a current limitation of Little-JIL, which we plan to address, these sub-steps are hard-coded into the process definition.

In addition to parallel steps, which allow flexible execution of the process, Little-JIL also defines semantics for *choice* steps like Add Registration Methods in Figure 3. This step is completed by *either* adding property change listeners or adding regular listeners. These language features allow the designer flexibility in activity ordering, enabling opportunism when the process programmer deems it appropriate, while always tracking overall design progress.

3.2 Artifact Guidance

We have developed a set of rules defining the structure of an MVC system, along with a system that automatically checks designs against those rules. One rule for example requires that model classes have registration methods used for adding and removing observers. Another rule requires that a parameter to those registration methods be a listener. To avoid overly-restricting the designer by disallowing inconsistent intermediate states, we do not strictly enforce all the rules, instead giving warnings when rules are violated. However, to avoid overwhelming the designer with feedback, we control the application of the rules based on the point the designer has reached in the process. We contend that if the design is going well, the design can be expected to satisfy *some* of the rules, but not necessarily *all* of them at any particular point. So, if we check only those rules that are applicable at those points, we get the benefit of a rule system and avoid overwhelming the designer with feedback.

In fact, we even respond to rule failures with process fragments that detail corrective action based on the process context. Little-JIL supports this with its exception handling mechanism. Detection of a violation causes an exception that can be caught and responded to with any Little-JIL step. So, instead of just giving a warning, we directly guide the user to fix the cause of the violation.

Consider the program snippet in Figure 2, which is executed (using Little-JIL's post-requisite mechanism) after an event class has been added. Check Consistency checks the design against a rule that requires all event classes to have constructors and that the constructors must have a model class as a parameter. If any violations of this rule are encountered, an exception is thrown. Each such exception is caught by Checker, causing Fix Class to be executed. Fix Class gives the user the choice of modifying a method or adding a method to fix the problem – because either the constructor exists and lacks the correct parameter or we need to add the constructor to solve the problem.

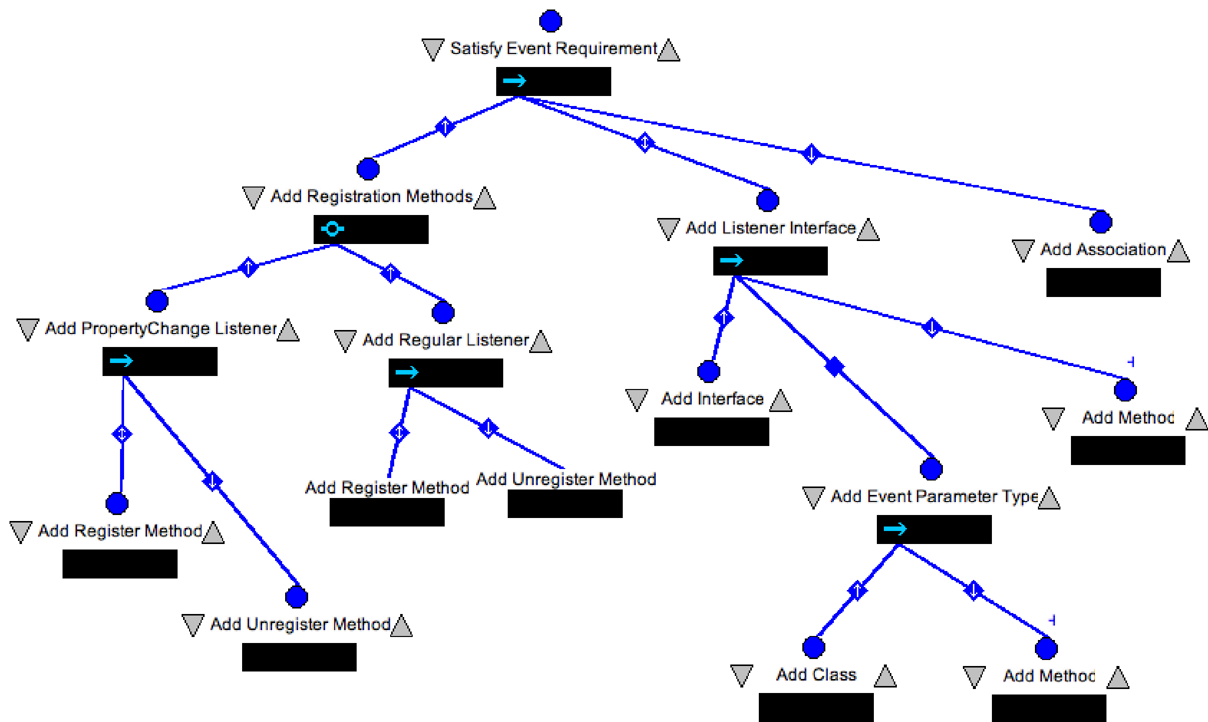

Figure 3: MVC Design Process, third diagram

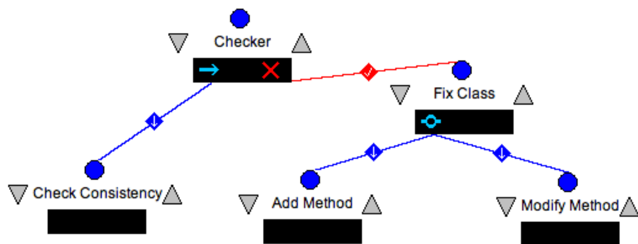

Figure 2: Consistency Exception

4. METHODS

In this section, we describe an experiment we performed to evaluate our approach by comparing it to an Argo-like approach that gives relatively more freedom but as a result must give less focused feedback. Our experimental approach was to develop or find design guidance tools using different approaches and have subjects use these tools to undertake a design task, measuring how well they do using these approaches.

Because we use a Little-JIL program to determine which activities to present to the user at which times, we can compare design guidance approaches by comparing Little-JIL process programs that codify them. Any differences users experience from using the two versions are thus attributable directly to the processes (as codified in the process programs) used in the two approaches. So, we created the following processes to study:

1. No Guidance (N): The user is allowed to perform any steps in any order. No consistency rules are checked.
2. Process Guidance (P): The process is as in our approach, but without consistency checks.
3. Artifact Guidance (A): This is the Argo-like process which is like process N but with consistency rules checked after each primitive action.
4. Combined Guidance (C): This is our approach, in which consistency checks are embedded in the process with responses to those checks also programmed as process fragments.

4.1 Hypotheses

We hypothesize that making use of process knowledge will help novice designers produce designs more quickly because they will spend less time making decisions, and make better decisions, because they will have fewer tasks from which to choose. The null hypothesis is thus:

> $H_0(1)$ **(duration using process guidance):** Design time without process guidance is equal to design time with process guidance.

We further hypothesize that novice designers will produce designs more slowly using artifact guidance because they will have to take time to respond to that guidance:

> $H_0(2)$ **(duration using artifact guidance):** Design time without artifact guidance is equal to design time with artifact guidance.

We also expect that more guidance, of either kind, will produce higher quality designs:

> $H_0(3)$ **(quality using process guidance):** Design quality without process guidance is equal to design quality with process guidance.

$H_0(4)$ **(quality using artifact guidance):** Design quality without artifact guidance is equal to design quality with artifact guidance.

We intend to test all of these hypotheses with one-sided hypothesis tests.

4.2 Experiment Design

Our sample of subjects was drawn from a population of students that had recently passed a course at Union College in which, among other things, they developed MVC designs. We chose this group of subjects because college students can be expected to be novice designers, and yet we did not have to also teach them about design.

We established groups of four subjects, one for each treatment, based on their performance on a pretest, which tested their design analysis skills, on the assumption that design analysis relates directly to design performance. We did not use an actual design task because we wanted to avoid a training effect. There were four groups of four subjects. Two subjects dropped out in the course of the experiment because they misunderstood or did not follow directions.

Each subject was asked to design an MVC system for managing a date book using one of the four processes. We captured the designs they produced and measured the time they took to perform the task. We allowed the subjects to work on their designs until either they deemed their designs complete or there were no more tasks to carry out.

The independent variables are the two factors (process guidance and artifact guidance), giving us four treatments. We measured design speed by instrumenting our tool-set to log the beginning time and the time for every user action. We measured design quality by having three design experts rate the designs produced. We supplied these experts with the designs, in random order, which they graded on a scale from 1 to 5 (best). To ensure that we got sufficient spread of grades from which to derive rank-order data, we asked the experts to assign each of the five different grades between two and four times (inclusive). We summed the scores given by each expert to establish a quality score for each design.

4.3 Threats to Validity

Internal Validity. We avoid training effects by having each subject perform only a single design task. We avoid maturation threats by having all subjects perform the design task within a week of the pretest. We avoid a history threat by grouping subjects by pretest score so that subjects with similar skills are compared with each other.

External Validity. Our measures may only be valid for MVC designs. We have attempted to deal with this threat by being very careful not to do anything that we did not think would generalize to other design patterns. Another possible threat to external validity is that our sample of subjects might not be representative of the novice designers in the population. This seems unlikely to us because our subjects are students with little design experience – the level of expertise we expect novice designers in industry will have.

Construct Validity. We wish to compare approaches, which are typically embodied in tools. However, in order to reduce internal validity threats, we have embodied the different approaches in instances of a single tool. Even if one approach has better tools, we are not measuring using those better tools. However, our expectation is that process and constraints can be applied in several tool contexts, not just using our infrastructure.

Our measure of quality is another source of possible threats to construct validity. We are not using an entirely objective measure of design quality. However, the collected wisdom of three expert designers with a variety of professional and academic experience seems to us to be a good measure of design quality.

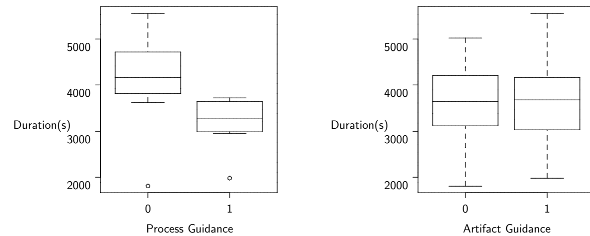

Figure 4: Duration vs. Process Guidance and Artifact Guidance

5. RESULTS AND DISCUSSION

We had intended to use an analysis of variance (ANOVA) test to determine whether there is an effect for each of the factors and whether the factors interact, factoring out the variance due to group (i.e. factoring out the starting skill level as measured by the pretest). Unfortunately, we found that the pretest score explained little of the variance in design duration (p=0.372). This suggests that the pretest is not very good at predicting design performance.

Instead, we focus on the main effects of each of the factors, disregarding the grouping. Figure 4 shows box plots of the duration data, which show a large improvement for process guidance but no effect for artifact guidance. We can reject hypothesis $H_0(1)$ (p=0.027 for a one-sided Wilcoxon rank-sum test), but we are unable to reject $H_0(2)$ (p=0.426).

Figure 5 shows box plots that summarize quality scores versus process-guidance and artifact-guidance, which again show an improvement for process guidance but no improvement for artifact guidance. We can reject hypothesis $H_0(3)$ with a one-sided t-test (p=0.0437), but are unable to reject null hypothesis $H_0(4)$ (p=0.274 with a t-test).

While our experiments are inconclusive with respect to the effects of artifact guidance, we do see statistically significant positive effects of process guidance on design speed and design quality.

6. FUTURE WORK

In addition to infrastructure and language improvements, we plan to extend the work to validate the approach for design approaches and disciplines beyond MVC. While we expect that the results apply beyond MVC, we need further experimentation to verify this. We are working on rule sets and corresponding processes for different design patterns, trying to find commonality so that we can develop a general-purpose tool-set that can offer guidance on specific design patterns.

Because we believe that novices both need more guidance and more tolerant of it, we are comfortable developing relatively prescriptive processes for them. We plan to also evaluate our approach as a device for supporting experts as well, but the challenges will be great. Experts are likely to insist on fewer restrictions and therefore less guidance – finding the fine line between not enough guidance and too much guidance is likely to be problematic.

7. CONCLUSION

Software design is a complex activity of creating a model of a system that is internally consistent and consistent with require-

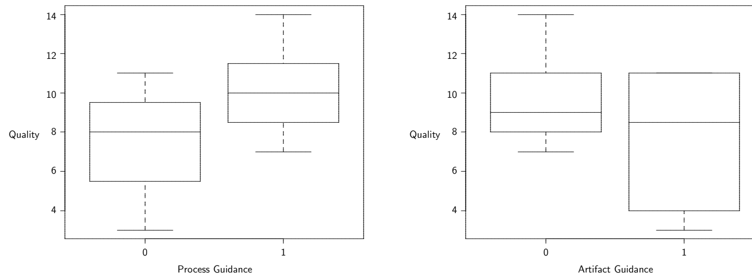

Figure 5: Quality vs. Process Guidance and Artifact Guidance

ments. These consistency requirements drive experts in their design activities, and can be used to help novices as well. By combining consistency rules with process guidance, we help the novice designer produce better software designs, and produce them faster. On the other hand, consistency rules alone do not have a significant effect on design duration or quality. So, while others have indicated that the process of design cannot be formalized or automated, we find instead that doing so has great value for novice designers.

Acknowledgments

We would like to thank Alexander Wise, Vandana Bajaj, Heather Conboy, David Fisher, and Timothy Sliski for their help in developing infrastructure or undertaking the experiments. We also thank David D. Jensen, for his considerable help in experiment design and analysis, as well as Lori A. Clarke, W. Richards Adrion, and Janis Terpenny for their advice throughout the project. We would also like to thank Anthony Finkelstein and Christian Nentwich for early discussions of their xlinkit [6] rule engine, which helped us in our development of the rule checker used in our experiments.

This research was partially supported by the Air Force Research Laboratory/IFTD and the Defense Advanced Research Projects Agency under Contract F30602-97-2-0032, by the U.S. Department of Defense/Army and the Defense Advance Research Projects Agency under Contract DAAH01-00-C-R231, and by the National Science Foundation under Award No. CCR-0204321 and Award No. CCR-0205575. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied of the Defense Advanced Research Projects Agency, the Air Force Research Laboratory/IFTD, the U.S. Dept. of Defense, the U. S. Army, The National Science Foundation, or the U.S. Government.

8. REFERENCES

[1] A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, S. M. Sutton, Jr., and A. Wise. Little-JIL/Juliette: A process definition language and interpreter. In *Proc. of the* 22nd *Int. Conf. on Soft. Eng.*, June 2000. Limerick, Ireland.

[2] A. G. Cass and L. J. Osterweil. Design guidance through the controlled application of constraints. In *Proc. of the Tenth Int. Workshop on Soft. Specification and Design*, Nov. 5–7, 2000. San Diego, CA.

[3] A. G. Cass, S. M. Sutton, Jr., and L. J. Osterweil. Formalizing rework in software processes. In *Proc. of the Ninth European Workshop on Soft. Process Technology*, Sept. 1–2, 2003. Helsinki, Finland.

[4] D. Garlan, R. Allen, and J. Ockerbloom. Exploiting style in architectural design environments. In *Proc. of the Second ACM SIGSOFT Symp. on the Foundations of Soft. Eng.* Assoc. of Computing Machinery Press, Dec. 1994. New Orleans, LA.

[5] G. E. Krasner and S. T. Pope. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. *J. of Object-Oriented Prog.*, 1(3):26 – 49, Aug./Sept. 1988.

[6] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: A consistency checking and smart link generation service. *ACM Trans. on Internet Tech.*, 2002.

[7] L. J. Osterweil. Software processes are software, too. In *Proc. of the Ninth Int. Conf. on Soft. Eng.*, Mar. 1987. Monterey, CA.

[8] J. E. Robbins, D. M. Hilbert, and D. F. Redmiles. Argo: A design environment for evolving software architectures. In *Proc. of the Nineteenth Int. Conf. on Soft. Eng.*, pages 600–601. Assoc. of Computing Machinery Press, May 1997.

[9] J. E. Robbins, D. M. Hilbert, and D. F. Redmiles. Software architecture critics in Argo. In *Proc. of the Third Int. Conf. on Intelligent User Interfaces*, pages 141–144. Assoc. of Computing Machinery Press, Jan. 1997. San Francisco, CA.

[10] W. Visser. More or less following a plan during design: Opportunistic deviations in specification. *Int. J. of Man-Machine Stud.*, 33(3):247–278, Sept. 1990.

[11] A. Wise. Little-JIL 1.0 Language Report. Technical Report 98-24, U. of Massachusetts, Dept. of Comp. Sci., Apr. 1998.

[12] A. Wise, A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, and S. M. Sutton, Jr. Using Little-JIL to coordinate agents in software engineering. In *Proc. of the Automated Software Engineering Conf.*, Sept. 2000. Grenoble, France.