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. Others have suggested that such an approach is viable for such processes as configuration management and testing. There has, however, been considerable sentiment that formalized process definitions are not likely to be useful in aiding in software design. This paper describes our experimentation with the hypothesis that formal definitions of design processes can facilitate the application of computing power to the design of software by novices. Specifically, we hypothesized that both design speed and design quality could be improved through the use of process definitions. Our experimentation supports this hypothesis.

1. INTRODUCTION AND STATEMENT OF THE PROBLEM

Software design is the difficult task of producing a model of a system that can provide assurance that the final system will satisfy its specified requirements. Design is therefore the task of producing a collection of model elements that are related to each other and to requirements elements such that the elements conspire together to satisfy the totality of the requirements.

It has been observed [12] 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 or may not be consistent – in fact the design will often need to go through states of inconsistency on the way to consistency. The designer must then decide both what task to perform next and on which part of the system to perform this task. The designer continues in this iterative fashion, fixing inconsistencies along the way, until the final design is complete.

At each point in this sequence of activities, there is a large number of design alternatives and choices of activities to undertake. These choices might be easily handled by an expert, but a novice is likely to be overwhelmed by the sheer number and variety of choices. Furthermore, an expert might be able to keep the overall design in mind, and know, in the midst of an iterative process, when progress on the overall goals is being made. A novice designer, on the other hand, cannot be expected to keep the overall goals clearly in mind.

The expert's iterative process can be viewed as a series of opportunistic responses to the inconsistency of the design – when inconsistencies are noticed, the expert reworks the violating parts of the design. We propose an approach to help the novice designer make progress in design by providing a partially-automated process that provides this kind of guidance.

Clearly, such a process should not be too rigid because design requires creativity, genius, ingenuity, and insight. However, design also requires a great deal of clerical work, documentation, and cross-checking. Because humans are good at the former, but computers are better at the latter, we seek to create a process in which each does what it is good at, and is relieved of needing to do what it is not good at. It is our belief that such a process, if correctly encoded, should be able to guide design progress effectively, while not being overly-restrictive.

Such a guiding process should be particularly helpful for novice designers working within the context of specific design styles. Such a process should be tailorable so that it directly supports specific kinds of activities needed to create designs in a particular, well-known design style or pattern. And within the context of such a process, a set of design consistency rules can be tailored for the design pattern of interest. In such a way, we believe that we should be able to give novice designers direct and focused guidance to make progress. We propose the dual hypotheses that by doing so, 1) we can guide novice designers to produce higher quality designs and 2) the novice designers will spend less time designing than with alternative approaches. In this paper, we present an experiment, the results of which support these hypotheses.

2. RELATED WORK

Design is well-understood to be a creative activity, requiring many mid-course changes in plan of attack. Visser [12] observed an engineering designer undertaking a design after having first documented a plan for how he would proceed with the design. In the end, the designer had not directly followed the plan, and instead

adopted an opportunistic approach. This opportunism seems to us to be inherent in the nature of the design process. Because of this opportunism, it has seemed difficult to capture the process of design in a formalized model. There are many efforts to better understand processes and improve upon them, and these activities have suggested to some that the process of design is too creative an activity to be amenable to formal modeling.

Because of this, tool-based approaches to helping designers tend not to focus on the process of design, instead focusing on the artifacts produced. For example, the Argo [9] design environment uses general-purpose design critics [10] to check design artifacts against consistency rules, giving feedback when the relations among these artifacts are inconsistent with respect to those rules. The Aesop [5] environment more strictly enforces design rules by restricting the creation of design artifacts that are incompatible with a design style being developed. Both of these systems, and others like them, provide guidance about how the artifacts should be related and structured in the end without giving direct guidance about the process to follow in order to achieve the desired structure and relations. This seems to us to risk overwhelming novice designers with too many choices.

Additionally, as noticed by Garlan, et al [5], there are times when rules should not be enforced. In particular, it is often the case that the only way to transform a design from one consistent design to another is to transition via an inconsistent intermediate state. If one is strictly enforcing all rules at all times, the rules would then have to be weaker to allow the intermediate states. Even if all the rules are not strictly enforced, but only used to provide warnings, the amount of warning feedback given would seem to us to be overwhelming at times, and therefore lacking in utility.

In previous work, we have argued that we can learn a lot by formalizing software processes as process programs [8], and partially automating their execution with appropriate infrastructure [2]. As part of this ongoing effort, we have developed a process-programming language called Little-JIL [14, 13] and an interpreter for it called Juliette [2] and have used both to encode and execute various complex processes, in software engineering as well as in such other domains as medicine and government. We have further argued that software design processes can indeed be defined using a suitably-flexible process program [3]. We have also used the flexibility of Little-JIL to support the formalization of the common activity of rework in design processes [4]. Based on all of this experience, we believe that design processes can indeed be encoded and executed while still providing the flexibility needed. In this paper, we take this work a step further by evaluating the approach with an experiment.

3. OUR APPROACH

Our approach is to augment the Aesop/Argo approach, emphasizing the evaluation of consistency rules among design artifacts, with process guidance provided by a Little-JIL process program, which is executed using Juliette. We start with the observation that the process one uses for one design style might very well differ from the process used for another, just as the consistency rules will differ (see [5]). Thus, for the experiments described in this paper, we chose to develop a process program and consistency rules for the Model-View-Controller (MVC) [6] architectural style. We made this choice because it seems that it provides many opportunities for defining important consistency rules and also because our own considerable experience in using the pattern has suggested an effective process for developing such designs.

3.1 Process Guidance

Consultation with a local designer who has much experience in developing MVC software led us to design a process for designing MVC systems. Informally, the process involves beginning with the model portion of the system, satisfying those requirements that deal with the storage of application data, and then working on the event system that keeps the views updated. The process is flexible, at times allowing the designer considerable choice in the order to execute process steps. The process also supports and encodes design rework by responding to problems by re-executing previously executed steps, thereby modeling for the novice the opportunistic rework practiced by experts (see [4]).

While the focus of this paper is not our process language Little-JIL, we present in Figures 1 and 2 the Little-JIL formalization of the process to clarify some of the aspects of the process and to demonstrate that the process is formally understood. As a Little-JIL process program, it is a hierarchy of steps that defines formally the allowable orders of execution of the individual steps in the process. The notation in the black bar for a step indicates the allowable orders of the sub-steps of that step. For example, the root step is Design MVC System and it is a *sequential* step, which means that its sub-steps are to be executed in left-to-right sequential order. So, this process mandates that the designer must first add the primary model class (an activity which is further decomposed into two other steps to be executed sequentially) and then must satisfy the model requirements. As indicated by the parallel lines in the step bar in Figure 1(b), satisfying a model requirement is a parallel activity[1]. This means that while we are adding a new model class we can also be modifying existing ones. Notice also the *cardinality* notation on the edge above Add Model Class. The question mark indicates that the step is optional. The cardinality of Add Method in the left side of Figure 1(b) is a Kleene star, indicating that we can add zero or more methods to the newly added model class. Similarly, the other reference to Add Method has a plus cardinality, meaning that we can add one or more methods to the existing classes that we are modifying.

In addition to the parallel step, which allows the designer quite a bit of flexibility in this design process, we also allow for a *choice* step like Add Registration Methods in Figure 2. This step can be achieved by *either* adding property change listeners or adding regular listeners. These language features allow the designer flexibility in the order in which to undertake activities, allowing the designer to act opportunistically when the process programmer deems it appropriate, while always keeping track of the overall design progress being made.

We have not yet shown Little-JIL's exception-handling mechanism, which we use to respond to problems discovered during execution of process steps. See section 3.3.

3.2 User Interface

The flexibility inherent in the process program is presented to the user though the user interface shown in Figure 3. The user inter-

[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 limitation of Little-JIL, these sub-steps are hard-coded into the process definition. See section 6 for more details.

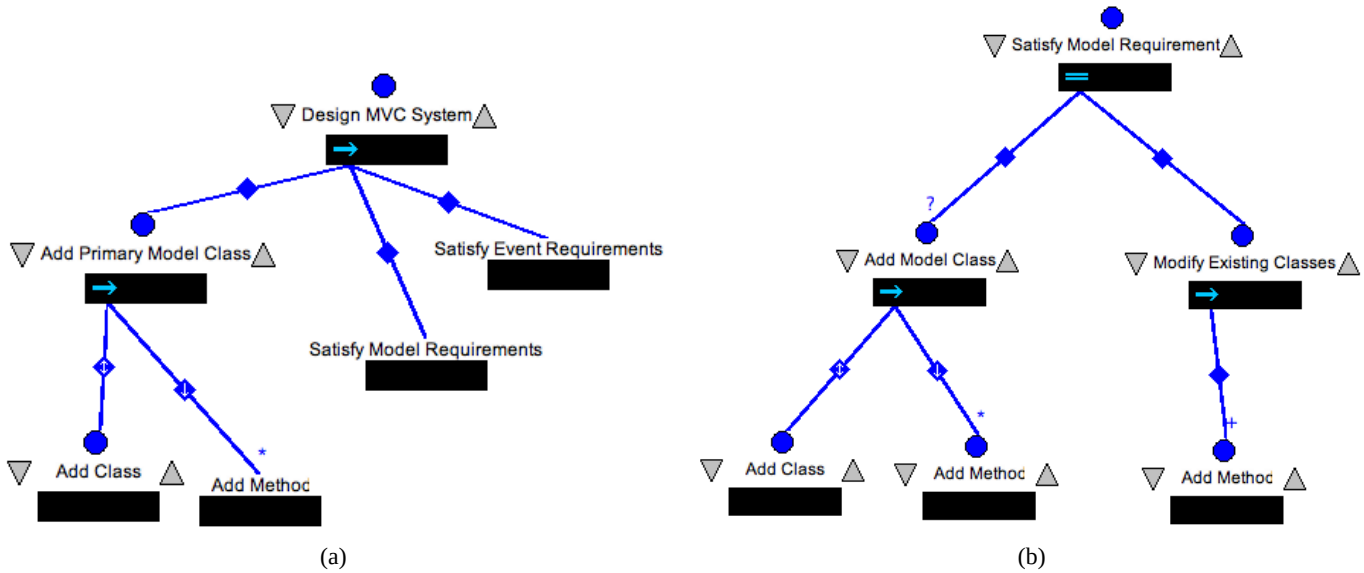

Figure 1: MVC Design Process

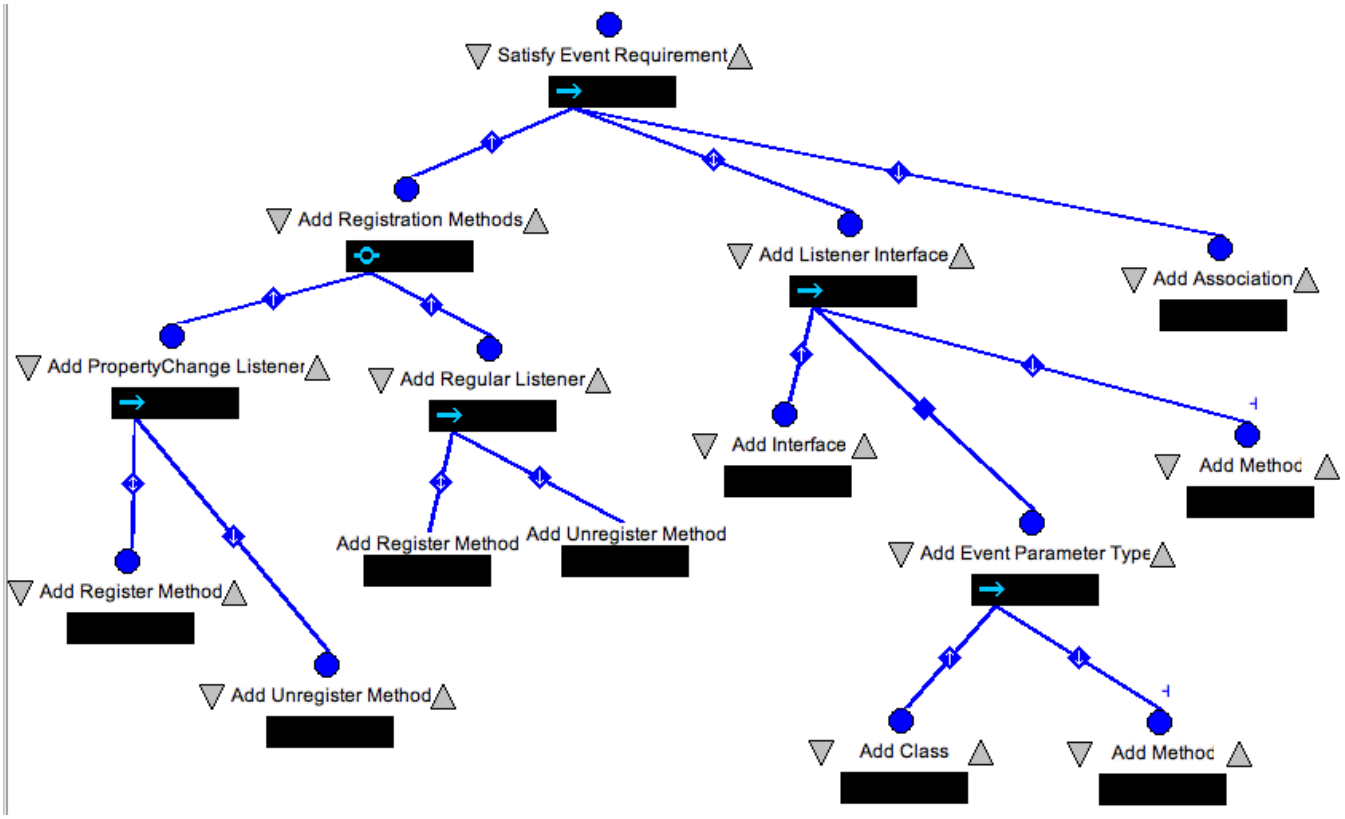

Figure 2: MVC Design Process, third diagram

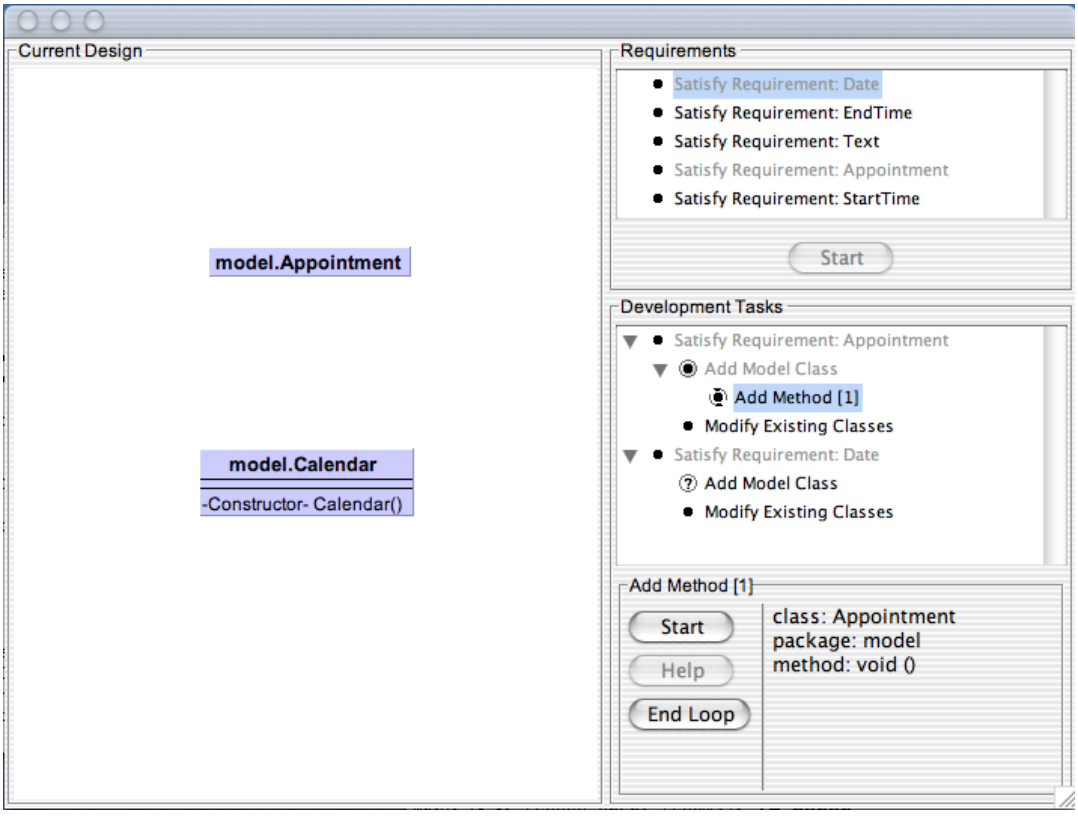

Figure 3: User Interface

face is broken into the design view and two task lists, the second of which has a more detailed view. The design view shows the current state of the design, in the form of a class diagram that is very much like those presented by other design tools. The task views present the various activities in which the user can engage at any one moment, while simultaneously showing the context in which those activities are taking place.

Consider for example the snapshot shown in Figure 3. In this scenario, the user is working on two instances of the Satisfy Model Requirement step, one for the Appointment requirement and one for the Date requirement. Since these two tasks are instances of a step that is a child of a parallel step, they may be carried out in any order, and thus the user is free to work on both at once – as is indeed shown in the task list[2]. Within the context of adding a class to satisfy the Appointment requirement, the user is about to add a method, and as such has selected the Add Method task. The bottom part of the task list then gives the user information about that task, such as parameter values and the iteration number. The icons next to the tasks in the task list indicate whether the step is optional or if it is part of a *loop* (i.e. if the associated Little-JIL step has a non-unit cardinality). For optional steps, the End Loop or Opt Out button is available to enable the user to specify an intent to complete the step without action and proceed forward.

When the user presses the start button on a primitive task, a dialog is presented to gather information from the user about the part of the design to be added or modified. Each piece of required information can be provided for the user, as specified by parameter passing specified as part of the Little-JIL program (not shown here), or it can be left to the user to fill in, thereby giving the user more or less control over the design being developed. When the user completes the dialog, the design model is updated and, depending on progress made in the Little-JIL program, other tasks may be presented to the user. In this way, the user has a range of freedom to work on different parts of the design, doing different activities, and yet the system can keep track of the progress made so far.

So, the process program provides flexibility and the user interface passes on this flexibility to the user. However, the process as described so far does not provide any artifact-based guidance. For that, we add consistency rules.

3.3 Artifact Guidance

We have developed a set of rules to describe the desired structure of an MVC system, along with a rule-checking system to automatically check the design against those rules. The inspiration for the rule-checking system is xlinkit [7], which we used in earlier versions of the system. xlinkit provides a facility for checking XML [1] documents against rules expressed as formulas over sets of elements and using their attributes. Our rules are described informally in much the same way, but we have implemented them in Java.

The rules are relatively straightforward and derived from the structural requirements placed on MVC systems. For example, there is

[2]In this example, we have configured the system to display instances of Satisfy Model Requirement with a task name different from the step name, to indicate what requirement is being worked on. Each other step is displayed to the user using the step's name.

a rule that says that model classes must have registration methods used for adding and removing observers, and another rule that requires that the parameter to those registration methods must be a listener. In this experiment we chose to use style-specific rules instead of general-purpose rules because we thought that their more focused guidance would make for a more effective process. Since we have a process model for a particular design style, it seems advisable to have style-specific rules to match. In future work we hope to explore the viability of more general-purpose rules.

To avoid overly-restricting the designer by disallowing inconsistent intermediate states, we avoid strict enforcement of the rules. If we did not, we would have to weaken the rules considerably in such a way that a design that satisfied all the rules for a particular style might not clearly have been an instance of that style. Instead, we choose to give warnings when rules are violated. However, it seems clear that this will result in many temporary violations and designers will not necessarily know which warnings to heed and which can be safely ignored. This seems particularly likely to be true for novice designers. The risk is that users of such a design tool will learn to ignore *all* the rules, again weakening the rule system.

Therefore, our approach is to use a rich rule set but control the application of the rules based on what point the designer has reached in the process. Our contention is that at certain points in the design process, if the design is going well, the design can be expected to satisfy *some* of the rules, but not necessarily *all* of them. So, if at various points in the process we check only those rules that should be obeyed at those points, we should be able to get the benefit of a rule system while avoiding, as much as possible, the drawback of overwhelming the designer with useless feedback.

In fact, we can take this idea one step further to include programmed repairs of the design. At certain points in the process, we should know something specific about the state of the design because of the process steps that have preceded. Then if we see a violation of one of the rules, we have a context in which to understand how to fix the design to bring it in line with the rules. In Little-JIL, we can accomplish this with the exception handling mechanism. When a violation is noticed, we throw an exception, which can be caught and to which we can respond with any Little-JIL step, which can, in turn, have sub-steps. So, instead of just giving a warning to the user of the design tool, we can directly give alternative steps to undertake to fix the problem.

Note that, because exception handling steps are simply Little-JIL steps, they are presented to the user in the same manner as the other steps. This seems particularly important for novice designers because they lack experience and are therefore likely to run into inconsistencies. The inconsistencies are what drive the progress forward, and thus treating them differently from the nominal flow does not seem desirable.

Consider the program snippet in Figure 4, which is executed (by way of Little-JIL's post-requisite mechanism) after an event class has been added. Check Consistency checks the design with respect to the single rule that states that any event class must have a constructor and that constructor must have a model class as a parameter (so that the model class can indicate the object that generated the event). If any violations of this rule are encountered, an exception is thrown. Each violation 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.

Note that at other points in the process, we can check different rules. For example, when we are creating an event class, we add methods to it. One of the rules states that none of these methods can be setters. So, instead of waiting until we've added all such methods, we can check the design with respect to this rule after each method is added. In this way, we place consistency checking for the rule at the lowest level at which we can expect it to be satisfied, and we can respond to violations with knowledge of what the designer is attempting at that moment. Note also that we can check multiple rules at any point in the process and we could then have multiple exception handlers, one to handle each kind of violation.

With this mechanism, we believe that we can leverage our knowledge of the progress made in the process to give directed guidance to novice designers so that they are not overwhelmed by feedback, nor overwhelmed by choices of what to do next. We hypothesize that this will have two main effects:

- The designs produced will be of higher quality than without this directed guidance.
- The novice designer's progress will be ensured and the design will therefore be completed more quickly than without this directed guidance.

4. METHODS

In this section, we describe an experiment we undertook to evaluate our approach. The experiment was designed to compare our approach to an Argo-like approach that gives relatively more freedom but as a result must give less focused, and therefore possibly more overwhelming, feedback. Therefore, our experiment design is essentially 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. Our general hypothesis is that our approach will produce better designs, more quickly, than other comparison approaches.

4.1 Infrastructure Support For a Comparison Approach

Because the infrastructure we have developed uses a Little-JIL program to know which activities to present at which times, creating two different user experiences is accomplished simply by creating two Little-JIL process programs. So, to compare design guidance approaches, we compare Little-JIL process programs that codify them. Any differences users experience from using the two versions are attributable directly to the processes (as codified in the process programs) used in the two approaches. Every other aspect of the user experience is the same for the two approaches – users interact with the tools in the same way, the tools use the same underlying technology, and the tools perform the low-level tasks with the same level of performance.

So, to compare the Argo-like approach to our approach, we need only create a Little-JIL process program for the Argo-like approach. Since Argo does not encode process knowledge, this second Little-JIL program must not restrict the actions of the user. This is accomplished with a very simple process program consisting only of a single parallel step with all of the primitive design actions as substeps, each with Kleene star cardinality, so that the user can perform any steps in any order, and as many times as the user may choose.

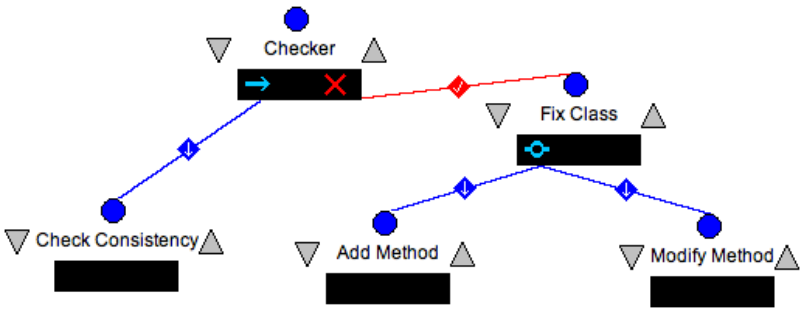

Figure 4: Consistency Exception

After each primitive design action, we check the design rules and present warnings in a separate warning box.

The Argo-like process lacks process-based guidance, but retains artifact-based guidance. In an effort to understand which of these two factors makes the most difference in the design, we also implemented a process with neither process guidance nor artifact guidance, as well as one that only had process guidance. We therefore were able to carry out a factored experiment with two factors, each with two levels (on and off), for a total of four treatments:

1. No Guidance (N): The process is a single parallel step with all primitive design steps as sub-steps, with Kleene star cardinality. No consistency rules are checked, so there is no artifact guidance nor process guidance.
2. Process Guidance (P): The process is as in our approach, but without consistency checks, therefore not giving artifact guidance but giving process guidance.
3. Artifact Guidance (A): This is the Argo-like process described above.
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.

4.2 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, as they will have a less overwhelming set of 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 slower 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.

4.3 Sampling and Blocking

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 these students knew at least something of MVC so that we did not have to also teach them about design. We note that this level of experience with the MVC pattern also seems representative of the experience level that novice designers in industry might have.

To populate the four treatment options, we established groups of four subjects based on their performance on a pretest. The pretest tested their design analysis skills, on the assumption that design analysis relates directly to design performance. This seems reasonable given that one must analyze one's own design to determine when the design is complete. We did not use an actual design task as the pretest because we wanted to avoid a training effect.

We ended up with four groups of four subjects. Two subjects dropped out in the course of the experiment because they misunderstood or did not follow directions. So, we therefore had fourteen subjects, broken into four groups based on their performance on the pretest.

4.4 Variables and Measures

The independent variables are the two factors mentioned above, giving us four treatments. We wish to measure as dependent variables both design speed and design quality. Design speed is easy to measure by measuring the time subjects take to perform a design task using a particular approach. We have therefore instrumented our tool-set to log the beginning time and the time for every user action. The duration is then measured by subtracting the start time from the time of the last action. This is measured in milliseconds and recorded in our analysis in seconds.

In order to measure design speed, we allow the subjects to work until they are complete with their design, either because they deem them complete or because there are no more tasks to carry out.

Measuring design quality is more problematic. The quality of the

performance of the task is how well it meets the objectives and in this case the objectives are to produce a good design according to a particular design style. Because this is the goal, we considered using our consistency rules as a measure of design quality – a design that fits the pattern well will violate few of the rules. However, we have rejected using this measure because it would seem unfair – subjects that directly use the rules in their design task would certainly have an advantage on this measure when compared with subjects that do not have access to the rules beforehand. We also considered using other standard design quality metrics like coupling and cohesion, but they do not seem to measure the real variable of fitness to the particular desired pattern.

Since purely objective measures did not seem appropriate here, we decided to use expert opinion to rate the designs produced by subjects in our experiment. To reduce the subjectiveness of this measure, we decided to get input from three design experts. Our design experts are:

- A technical staff member of a research laboratory and current instructor of software engineering.
- A technical staff member of a software engineering research laboratory, with extensive experience reviewing designs created by others.
- An industrial software engineer with recent past experience teaching software engineering at the university-level.

We supplied these experts with the designs produced by our subjects, in random order, and asked them to give a grade on a scale from 1 to 5, with 5 being best. To ensure that we got a good amount of rank-order information, we asked the experts to assign each grade at most 4 times and at least 2 times. To derive a design quality score from these individual grades, we have simply added the scores received by each expert.

Note that our experts had no prior knowledge of the processes involved in our experiments, they did not have any way to determine which group the designs were from, and they were not coached on how to grade the designs – we relied on their design expertise alone to assess the quality of designs produced in our experiment.

4.5 Experiment Setup

We gave each of the subjects a design task to perform and one of the four tools to use – each member of a group using a different one of the four tools. The design task was to design an MVC system for managing a date book. The subjects were given a set of requirements for the system and documentation describing how to use the tools. The documentation was different for the different treatments only in that some of the tool behavior was not available for some treatments. For example, for treatments without process guidance, there was only one task list for the low-level tasks while those with process guidance added a high-level task list for higher-level tasks. Also, the Argo-like approach had a warning list.

We captured the designs they produced and measured the time they took to perform the task. We allowed the subjects to work with the tools until their designs were complete, either because they deemed them complete or because there were no more tasks to carry out.

4.6 Threats to Validity

In this section, we outline the primary risks that our results might become meaningless, and discuss what we have done to ameliorate these risks. See [11] for general explanation of validity threats.

4.6.1 Internal Validity

Our experimental design is aimed primarily at reducing risks of internal invalidity. We avoid training effects by only having each subject perform 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. Note also that the pretest scores are in a relatively small range.

4.6.2 External Validity

In our experiments, we attempt to measure the effects of process guidance and artifact guidance. However, we measure these only in the context of processes and constraints for MVC designs. So, our measures may only be valid for such 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 styles.

Another possible threat to external validity is that our sample of subjects might not be representative of the novice designers in the population. While this might be true, it 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.

4.6.3 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. It might even be the case that one approach might enable better tools that are not possible at all with the other approaches. 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.

5. RESULTS AND DISCUSSION

We start with the design duration variable. 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, due to the distributions of the measure data, this did not provide any useful information. We found that the grouping of subjects by 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 determining the main effect of each of the two factors, disregarding the grouping. Figure 5 shows box plots of the duration data. There seems to be a large improvement in duration for process guidance versus no process guidance. In fact, we can reject hypothesis $H_0(1)$ (p=0.027 for a one-sided Wilcoxon rank-sum test).

The second box plot in Figure 5 shows the effect of artifact guidance. There appears to be no effect, and in fact we are not able to reject $H_0(2)$ (p=0.426).

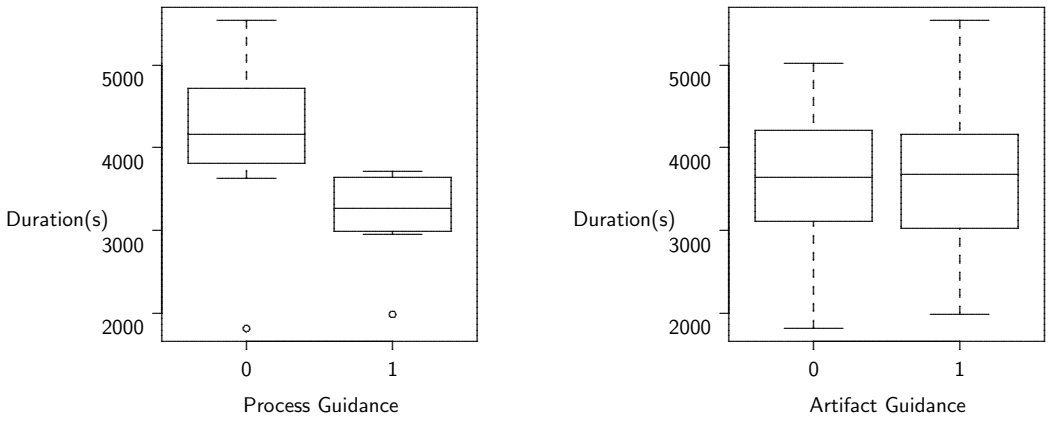

Figure 5: Duration vs. Process Guidance and Artifact Guidance

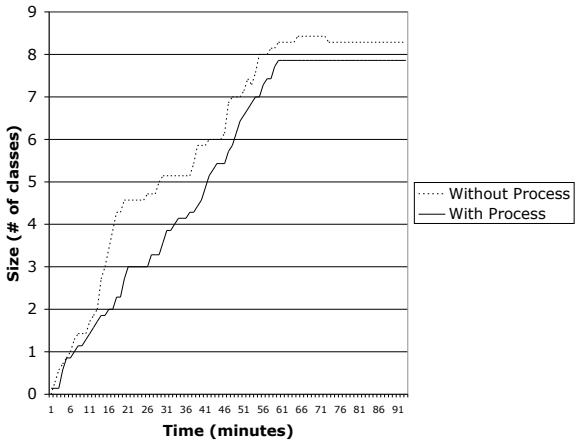

Figure 6: Number of classes versus time

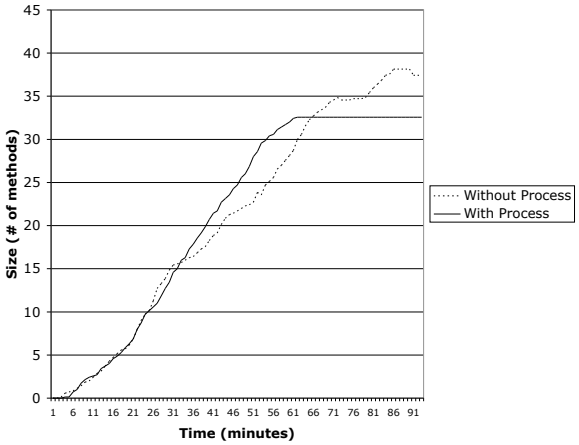

Figure 7: Number of methods versus time

The statistical analysis indicates that there is less than a 3 percent chance of observing a difference of this magnitude given that the null hypothesis $H_0(1)$ is true. Note also that the Wilcoxon rank-sum test makes no assumptions about the distributions of the samples. So, while we will definitely try to get larger sample sizes in the future, for this experiment, the sample size seems large enough to support the effect we are measuring here.

So, subjects without process guidance spend more time designing that those with it. What are these subjects doing with the extra time? One possibility is that they are spending more time because they are producing more design. To determine whether this is the case, we have analyzed the differences in design size, as measured by the number of classes and the number of methods, between the process-guided and the non-process-guided designs. Figures 6 and 7 show, respectively, the number of classes and the number of methods, at one minute intervals, for subjects with and without process guidance (averaged over all such subjects). In both cases, we see that process-guided subjects produced, on average, slightly smaller designs. However, the differences in final number of classes and final number of methods are not statistically significant (p=0.818 and p=0.484 using two-sample t-tests).

Even though we do not find a significant difference in the final design sizes, Figures 6 and 7 seem to show differences in how the different groups of subjects arrived at these design sizes. In fact, a Kolmogorov-Smirnov two-sample test shows that it is very unlikely that the temporal distributions are the same for the process-guided and non-process-guided groups for either number of classes (p= 1.77e-06) or number of methods (p=0.000436).

Figures 6 and 7 show that those with process guidance build fewer classes early, and then rapidly create classes near the end of their design tasks. On the other hand, the number of methods does not differ much early in the tasks, but later the non-process-guided subjects seem to slow the rate of adding methods. Additionally, the average rate of method creation for process-guided subjects is very steady throughout the design task duration.

The differences in the rates of class creation seem best to be ex-

plained by the structure imposed by the process providing the guidance. The process guides the user to work on the model classes before working on the event classes, so the early part of the design task will be focused on a few classes. On the other hand, those without process guidance can work on the event classes at any time. Without guidance that it makes sense to work on the model classes first, the subject might choose to spread their effort over the entire design, thus resulting in more classes created early than the process-guided subjects.

Method and class creation rates slow near the end of the design task duration for non-process-guided subjects, while process-guided subjects do not show this effect. Further experimentation will certainly be needed to ascertain what is occurring here, but one reasonable hypothesis is that after the design takes a preliminary shape, individual design decisions are more difficult to make because there are many possible new design elements to add and those new design elements must work with existing ones in potentially intricate ways. The hypothesis is that the process-guided subjects give a more focused effort because the process tells them what to focus on, reducing the decision-making burden placed on the subject.

The above results suggest that subjects without process guidance are not doing more design, but perhaps they are producing better design? Figure 8 shows box plots that summarize quality scores, derived from experts' rankings of the designs, versus process-guidance and versus artifact-guidance. We see what looks like a distinct increase in design quality with process guidance as compared to quality without process guidance. In fact, we can reject the null hypothesis $H_0(3)$ listed above, with a one-sided t-test (p=0.0437).

Artifact guidance, on the other hand, shows a negative effect on design quality, though this difference is not statistically significant. We are unable to reject null hypothesis $H_0(4)$ above (p=0.274 with a t-test).

So, our experiments are inconclusive with respect to the effects of artifact guidance on design speed or design quality. We do, however, see statistically significant and positive effects on design speed and design quality of process guidance. To emphasize the results here, note that we have a formalized process for software design. It might seem that to formalize a process is to make it overly-restrictive and that this overly-restrictive process cannot support such a creative activity as design. However, we have shown that we can produce a flexible, yet formal, process for software design. Through experimentation, we have shown that this formal process can support the creative activity of software design and help novice designers produce high quality designs. Additionally, we have shown that the process does not slow designers down – in fact, they produce their designs more quickly than without it. So, while some would argue that formalizing processes is both impossible and undesirable, we find that it does indeed lead to significant benefits for novice designers.

6. FUTURE WORK

One of the goals of this work has been to evaluate the usefulness of Little-JIL for developing processes that software engineers might use. We see this work as validation that the language is on the right track. However, we do plan to address some short-comings clarified by this work. As we mentioned before, in our formalization of the process we had to hard-code the requirements for the system into the Little-JIL program. Of course, this makes the process non-portable to other design problems. Our plan is to introduce a new cardinality feature to the language that will allow a process programmer to indicate that a step should be instantiated once for each of a collection of artifacts. If we then model the requirements as such a collection, we can write a process program that does not have specific knowledge of the requirements of the system under design.

In addition to infrastructure and language improvements, we plan to extend the work to validate the approach for different domains. The existing process is designed specifically for MVC designs. While we expect that the results are transferable to other design styles, we will need further experimentation to verify this. We are currently working on developing rule sets and corresponding processes for different design styles. Our intention is to try to find commonality so that we can develop a general-purpose tool-set that can still offer guidance on specific design styles.

Also, the processes we have developed to date are aimed at helping novices. Because we work with novices, we feel more free to make somewhat restrictive processes, the theory being that novices both need more guidance and appreciate it more. We plan to further develop the approach for 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.

So, while this experiment has taught us something very valuable about how to help novice designers with MVC designs, we need much more experimentation in the future to help non-novices with non-MVC designs as well.

7. CONCLUSION

Software design is a complex activity of creating a model of a system that is internally consistent and consistent with the requirements. These consistency requirements drive experts in their design activities, and can be effectively used to help novices as well. By combining consistency rules with high-quality process guidance, we help the novice designer to make design progress and, perhaps more importantly, we help the novice designer know when to stop. Therefore, process guidance helps the novice designer produce better software designs, and produce them faster, than they would otherwise. On the other hand, consistency rules 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 a great advantage for novice designers.

Acknowledgments

We would like to thank Alexander Wise, for his help in developing the Little-JIL processes used in this experiment, and Vandana Bajaj for her help in carrying out the experimental runs. We would also like to thank Heather Conboy, David Fisher, and Timothy Sliski for their help in evaluating design quality.

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, in particular for their suggestions that led to the quality metric used and to a better analysis of how the subjects' designs evolved over time.

We would also like to thank Anthony Finkelstein and Christian

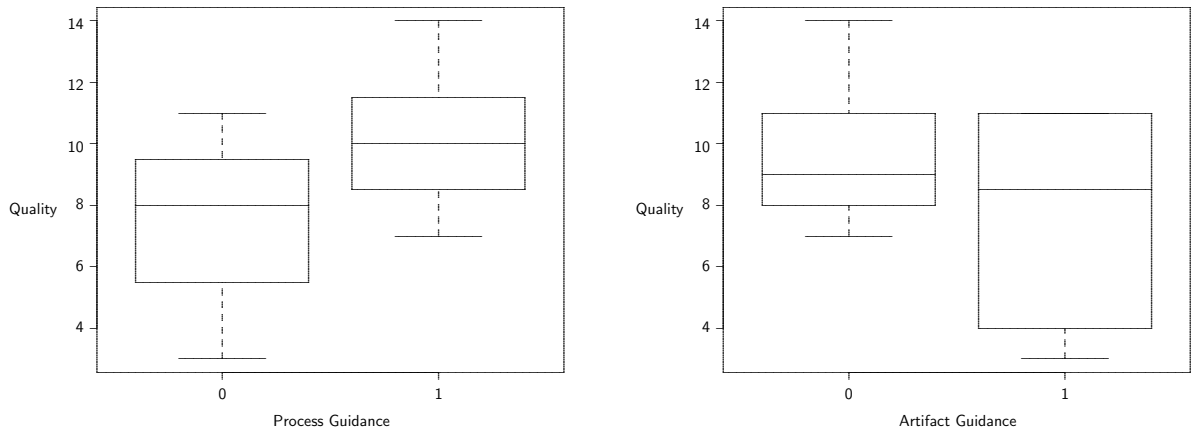

Figure 8: Quality vs. Process Guidance and Artifact Guidance

Nentwich for early discussions of their xlinkit 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] T. Bray, J. Paoli, M. Sperberg-McQueen, and E. Maler, editors. *The Extensible Markup Language (XML) 1.0*. World Wide Web Consortium, second edition, Oct. 2000.

[2] 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* 22^{nd} *Int. Conf. on Soft. Eng.*, June 2000. Limerick, Ireland.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: A consistency checking and smart link generation service. *ACM Trans. on Internet Tech.*, 2002. To appear. Available at www.cs.ucl.ac.uk/staff/A.Finkelstein/papers.

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

[9] 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.

[10] 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.

[11] W. M. Trochim. The research methods knowledge base, 2nd edition. Internet WWW page at URL: http://www.socialresearchmethods.net/kb/index.htm. version current as of November 7, 2004.

[12] 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.

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

[14] 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.