Verifying the Preskriptor Process with Students

Introduction

Traditionally, the most difficult transition of the development process for a non-trivial software system has been the one from the requirements for the system (i.e. what the system is supposed to do) to its design (i.e. the high level components and structure of the system). This step involves going from the problem's domain to the domain of its solution. One of the factors that make the design of software systems so challenging is that they have to achieve many different requirements (problems) at the same time, and there is often not a single solution to a problem, but many possible ones.

In our previous work we introduced a new methodology that makes it possible to derive a less constraining architectural specification (architectural prescription) from a goal oriented requirements specification. An architecture prescription specifies the components and their interactions considering only the problem domain goals they have to achieve.

We claim that our methodology guarantees satisfaction of all requirements, it provides means for better separation of concerns, and, most importantly, it makes it easier to perform the high level design of the system starting from the system's requirements than traditional methodologies.

The purpose of this experiment is to achieve an initial validation of our methodology in a laboratory setting. A laboratory setting gives us careful control over some of the issues while also providing an inexpensive way to do preliminary experimental work.

Apart from providing an initial validation of the Preskriptor process, this experiment will give us additional insight on the design of future experiments that will be in the industrial environment.

Sample selection and ruling out threats to validity

Our research sample will be constituted by junior and senior computer engineering and computer science students. The sample size shall be of at least 65[1], number that provides a good statistical power in the data analysis phase. We'll take our sample from four classes: a programming class and a software engineering class in the computer engineering department, and two similar classes in the computer science department. The classes we selected are: Software Engineering Processes and Data Structures in C++, in the computer engineering department; Software Engineering and Object Oriented Programming, in the computer science department.

This sample will not be a homogeneous one, as the students' knowledge of programming and their knowledge of the software engineering methodologies may vary significantly among them. Also, the fact that computer engineering and computer science

[1] Given by the Cohen table considering a statistic t, a medium effect, α=0.05 and a power equal to 0.8

students are exposed to some classes that are different in nature may affect their performance in the project we will assign them. For example, computer engineers generally have taken more hardware classes, while computer scientists have taken more programming and theory classes.
To rule out the biases deriving from assigning all the students with more or a different kinds of knowledge to one treatment or to another, we shall split the sample according to two independent variables: *knowledge of software engineering* and *student's major*. The *knowledge of software engineering* variable can be: *none*, for students never exposed to the field; *some*, for students that took or are taking classes in the field. *Student's major* can either be *computer engineering* or *computer science*.
Students in the software engineering classes have taken for sure programming classes because of the departments prerequisites regulations for those classes. Students in the programming classes may have taken a software engineering class before and we ask them if it is so before assigning them to the right sub-sample. As a result of the split we'll get four sub-samples of students.

The treatments shall be the following. One treatment shall be the Preskriptor methodology, which takes as input goal oriented requirements specifications and produces an architectural prescription. Another treatment shall be going from informal requirements to an architectural design in an ADL: Acme. A third treatment shall be methodology from informal requirements to an informal architecture design.

To achieve better internal validity and external validity, we'll assign the same number of representatives of each sub-sample to each treatment. In fact if, for example, we assigned some students from sub-sample S1 to all treatments but treatment T1, the cause of better, equal or worse results of treatment T1 could have been due to the fact that no representatives from group S1 performed the project using T1. If we decided to use only a homogeneous sub-sample of students, we would have lost in external validity, which we intend as a generalization of our results to the computer engineering/science majoring student population.

To not confuse the students, they shall be randomly assigned to only one methodology, and shall have to use only that one for the system's design. Students won't be told anything about the origin of the methodology they are taught. This approach rules out the possibility that the group to which our new methodology is given feels more/less motivated. The different methodologies are taught by instructors who support them, in about one hour lectures. This approach rules out biases in the results deriving from teaching alternative methodologies in a less enthusiastic way.

The system's design is an individual project. It's important that the students don't cooperate with each other. It's also important that the students are committed to the project in terms of effort and time. To motivate the students, the instructors shall emphasize the importance of the experiment. To discourage student cooperation or communication of results on the individual projects, the instructors shall tell them how important it is for the success of the experiment that they don't communicate with each other. Even grading the project may be possible in those classes for which the topics of requirements specifications and software design are taught anyway. Being graded, or being offered extra credit may, will provide additional motivation to the students.

The experiment's duration will be short enough to rule out significant changes in the independent variables used to split the sample.

Criteria and measurements for the dependent variables

The Preskriptor methodology is designed to guarantee the achievement of all the requirements by the architecture designed according to it. This claim can be proven by the following property of the Preskriptor methodology P_1: the Preskriptor methodology is iterated till all the requirements are achieved by designed system.

Property P_1 proves that we will achieve all the requirements given that we solve all the conflicts among requirements at the requirements specification level and that we iterate the Preskriptor process long enough.

However, we do have to take measures to have an idea of till what extent this property of the designed architecture is achieved by the traditional methodologies. Also, students may apply our methodology not correctly and therefore not achieve this fundamental product quality. Therefore, to measure the achievement of the property thanks to the different methodologies, we'll ask the students to identify, in the architecture they designed, for each requirement which the components that contribute to achieve it are, if any. A variable, called requirements satisfaction variable will contain the number of requirements that have at least a component taking them into account.

Our second claim is that the Preskriptor methodology provides products with better separation of concerns. By separation of concerns we mean that only a clearly identifiable subset of the system's components and eventually the system's topology (in the case of non-functional requirements) are used to achieve a particular goal. By clearly a identifiable subset we mean a subset of the system in strict sense and that we know exactly which are the components belonging to it. By achieving separation of concerns, we don't have to worry about a particular goal in any of the components not belonging to its subset, and that when we have to modify a component contributing to the achievement of a goal we are aware of his contribution and we can make sure keep that we keep it after modifying it.

To measure the second property, for each requirement R_i and for each component C_i that is not in the set of components identified by the student as achieving requirement R_i, we will ask the student what the degree of confidence with which he/she thinks that component C doesn't contribute by any mean to the achievement of the requirement R. The level of confidence shall assume one of the following values: completely sure, almost sure, not sure. Even if a student clearly knows for sure which components contribute to a particular requirement, this is not a guarantee that her/his architecture has achieved separation of concerns; for she/he may have just assigned all the requirements to all the components! For each student project we will compute the average number of components per requirement; in general, the smaller this number is the better the separation of concern. This measure could not yield a correct result in case the student refined a particular component but didn't refine others. In this case, the requirement assigned to the refined component may then be assigned to all its sub-components. This would raise the average number of components per requirement, yet the obtained architecture would provide the same degree of separation of concern than the older. To find this out, we shall analyze the architectures designed by the students, to see if there was any rationale in assigning a requirement to some components so that the selected

components constitute a subsystem that can be identified by a property. These questions will be asked only after they have turned in their completed project. We have to take these measures a posteriori, or else we would be implicitly modifying the methodology that the student is using or conditioning their behavior during the experiment. For example, to answer the second question, a student would use traceability even if the methodology does not require him to do so.

Our third hypothesis concerns a very important process quality, easiness, and states that the Preskriptor methodology makes it easier to design an architecture for the system given its requirements. One of the criterions we shall use for the *easiness* is the actual time spent by the participant students to learn the methodology and the time they took to design the architecture. To measure this criterion we'll ask the participants to time themselves when they are working on our project. Since most of them will be working with a new methodology, we'll ask them to take track of how much time they spent to learn the methodology. Given that the sample contains students that have different background knowledge of software engineering, the average time they spent is a pretty good indicator of the difficulty to learn the different methodologies. We'll also ask them how much time they spent on the design the system.

Another criterion we'll consider is the easiness as perceived by the students. To measure it, once the project is completed, we will ask the students to answer the following questionnaire:

Please rate your level of agreement with each of the following statements.
The distance between any two consecutive levels of agreement (such as strongly agree and agree, agree and undecided, etc.) are the same.

- I understood the rationale behind the process
Strongly agree, agree, undecided, disagree, strongly disagree

- It's not clear why the process is structured like that
Strongly agree, agree, undecided, disagree, strongly disagree

- There were some points difficult to understand
Strongly agree, agree, undecided, disagree, strongly disagree

- The different points of the methodology were clear
Strongly agree, agree, undecided, disagree, strongly disagree

- I knew what to do to start the project
Strongly agree, agree, undecided, disagree, strongly disagree

- I wasn't sure how to begin working on the project
Strongly agree, agree, undecided, disagree, strongly disagree

- When working on the project, at the end of each step I knew what I had to do next
Strongly agree, agree, undecided, disagree, strongly disagree

- There were times, working on the project, when I didn't know what to do after performing a step

Strongly agree, agree, undecided, disagree, strongly disagree

- It was straightforward to perform what each step required me to do
Strongly agree, agree, undecided, disagree, strongly disagree

- Some steps were hard to perform
Strongly agree, agree, undecided, disagree, strongly disagree

- I clearly knew when the project was completed
Strongly agree, agree, undecided, disagree, strongly disagree

- When I finished working on the project I wasn't sure if there was still something left to do
Strongly agree, agree, undecided, disagree, strongly disagree

We'll use a five points scale to rate the statements. We can characterize the easiness of a process by the easiness of understanding it, and by its usefulness in developing an architecture.

The first four statements aim at measuring the first of these two characteristics. There are redundancies, i.e. statements that provide the same measure, such as *I understood the rationale of the process* and *It is not clear to me why such a process should work*. One statement is positive and the other one is negative. This redundancy is useful to rule out biases that may arise from the subjects' ratings. Some students may not be motivated and just select the same rating for each statement; other students may always agree with a statement. In a similar way all the other positive statements have a negative counterpart. Above we wrote the statements that provide the same measure one after the other one. In the actual questionnaire they shall not be next to each other so that it won't be evident they are a different wording of the same sentence.

The latter eight statements are used to measure how useful the methodology was, how easy it was for the student to apply the methodology. The student may find it difficult to derive in practice an architecture from its requirements, even when he learned well the methodology.