Design Studies in Software Engineering Courses

Spencer Rugaber
Georgia Institute of Technology
spencer@cc.gatech.edu

Categories and Subject Descriptors

D.2.10 [**Design**]; K.3.2 [**Computer science education**]

General Terms

Design

Keywords

Software engineering education, design studies

1. BACKSTORY: CÉSAR PELLI

Several years ago, I was visiting the National Building Museum in Washington D.C. At the time of my visit, there was an exhibit spotlighting the architect César Pelli, designer of, among other things, the Ronald Reagan National Airport in Washington, the World Financial Center in New York, and the Petronas Towers in Kuala Lumpur, Malaysia. But on this day, much of the exhibit was devoted to a more modest building, a small hutlike abode in a natural setting. There were perhaps a dozen variations, built from wood and standing several inches tall. This was labeled a "design study", and it raised in my mind the question of whether software engineering students could undertake such exercises as part of their training. Since that time, as part of an advanced software engineering course, Software Architecture and Design[1], I have incorporated design studies as a learn-by-doing project element. By *design study* I mean a systematic exploration of a theme, including the construction of more than one solution to a problem and a comparison of the results.

2. COURSE BACKGROUND AND STRUCTURE

The content covered by the course comprises the design aspects of software development including specification, design notations, software architecture and architectural styles, middleware, components, design patterns, and design reviews. The course normally includes between 30-40 students, all with advanced software engineering backgrounds. There are several skill-based assignments and three interrelated projects, all organized as design studies.

For the projects, the students are divided into teams of approximately four persons to solve a design problem. Teams are shuffled for each project so that the students are forced to work with a variety of teammates. Teams are balanced with respect to programming skills and graduate/undergraduate participants.

The projects build on each other in the sense that the solution begins with the selection of appropriate pieces from preceding projects. This has several advantages. First it means that any given project can be somewhat more ambitious because some of the software for it already exists. This, in turn, means that the students can focus more of their attention on design and less on coding. Second, the students must evaluate each other's work, a skill important in actual industrial practice. Third, they realize that what they produce will be reused rather than thrown away and that they must, therefore, work harder at improving its quality, its generality, and its documentation. Finally, it means that the amount of domain knowledge the students must acquire to solve the problem is amortized across the three projects.

3. SOFTWARE DESIGN STUDIES

For all three projects, the teams must construct multiple solutions, compare the results and prepare a report. In the reports, the students must describe their designs, not only with UML diagrams and accompanying text, but also in terms of rationale. That is, they must explicitly list their major design decisions, the alternatives they considered and the reasons for their choices. After the design description, the reports must contain the results of their design study. This includes the following material:

- *Experimental conditions*: machines; operating system, language version and APIs; database technology
- *Procedure*: number of runs made, run configurations, initial conditions
- *Variables*: independent and dependent variables
- *Measurements*: how measured, metrics, statistical analysis, anomalies detected, presentation of data
- *Code*: source and byte code sizes, classes, interfaces, and methods used

[1] `http://www.cc.gatech.edu/classes/AY2008/cs4330_spring`

SOD'07 Science of Design Symposium 2007, Humboldt State University, Arcata, California, USA

- *Reuse*: extent to which code was reused and how much it had to be adapted
- *Analysis*: how the results relate to the design decisions made: precise statement of tradeoff in nonfunctional properties, discussion of any anomalous measurements

After turning in their reports, students present their results in class to each other, both as a demonstration of the executing solution and with a description of their particular design choices. Teams can see what the other teams have done. Audience questioning enables students to determine how other teams solved difficult parts of the design problem.

4. EXAMPLE: HEAT DIFFUSION

As an example of the sort of projects that can be used for design studies, this section describes a connected series of three projects built on the domain of heat diffusion through an orthogonally gridded medium. The spread of heat is modeled using a discrete version of the Fick's Law diffusion equations, which are provided to the students in pseudocode form. The three projects respectively model the warming of a rectangular metal plate, a rotating sphere, and a tilted and orbiting sphere. For each of the studies, the students are also confronted with design tradeoffs among nonfunctional requirements. Respectively for the three studies they are 1) performance, precision, and flexibility; 2) performance and modularity; and 3) persistence, accuracy, storage, and performance. That is, the student teams must provide multiple solutions in each of which a nonfunctional factor is dominant. They then perform an analysis that makes explicit the tradeoffs involved.

4.1 Study 1: Heated Plate

For Study 1, the students must provide five solutions to the Heated Plate problem. Topically, the first project emphasizes design notation (UML), modularity, and adaptability. The design study itself consists of solving the diffusion problem five different ways. One dimension involves the choice of whether or not the topology of the grid is encoded in an array or as connected objects. Presumably, arrays are more efficient, but objects are more adaptable to future changes. Another dimension is whether the information about the temperature is saved in doubles or in floats. The natural assumptions are that doubles are more precise, but operations on them are more costly.

The first four solutions are tightly constrained. Because their outputs are produced as text, automated testing can be performed to determine their correctness. But tightly constrained solutions limit another aspect of design—creativity. Therefore, the fifth solution asks them to provide a graphical user interface and a visualization. They are free to structure these as they see fit as long as they can argue for the usability of their approach.

4.2 Study 2: Rotating Sphere

The second project builds upon the first in the following ways. First, the rectangular plate, becomes a sphere. This means that the grid cells are no longer uniformly sized. Now, instead of heat transferring between adjacent cells uniformly in all directions, the amount of heat spread is proportional to the length of the shared boundary. With the sphere, there are no longer any edges to act as a source of heat. Consequently an external heat source must be provided to radiate heat to the surface of the sphere. Moreover, the sphere rotates so that only part of its surface faces the heat source at any given time.

At this time during the course, the lecture topic is software architecture. Consequently, the teams must produce solutions that vary architecturally. In particular, they must produce singlethreaded and multithreaded versions, as well as versions that run in a distributed fashion using multiple virtual machines.

4.3 Study 3: Tilted, Orbiting Sphere

The third design study also adds both functional and nonfunctional enhancements to the previous design studies. Functionally, the sphere is now tilted toward the heat source, engendering seasonal variations. Moreover the sphere orbits the heat source in an elliptical path. Both the degree of tilt of the sphere's axis and the eccentricity of the orbit can be set by the end user.

Nonfunctionally the students must deal with the issue of persistence. That is, the computed data values that are to be visualized may be produced in real-time or they may come from a database. Choice of mechanism for persistence (flat files, XML, relational database) is up to the teams, but they must justify their choices. The visualization should be independent of whether the data is persistent or not. This means that interpolations must be provided. Interpolations are of two sorts. They may be temporal; that is, the visualization may need temperature values for time intervals that were not computed. They may also be geographic; that is, the display grid on the visualization side may be finer grained than that produced by the original simulation. Persistence requires storage space, and the amount of data to be stored can grow quite large. So the students must make a decision about how much data to store, both in terms of sampling rate and precision.

The lecture topic during this period of the course is Design Patterns. Hence, in addition to persistence, teams are encouraged via extra credit to incorporate and document any design patterns they used on the third project.

5. CONCLUSIONS

Design studies have succeeded in encouraging students to focus on design issues. Nevertheless, there are further improvements that can be made.

- Provide even more pseudo code to reduce further the amount of coding required of the students.
- Have students individually prepare a postmortem analysis. The intent is to encourage more of the students to become engaged in the design process rather than relying on a single team member to be chief architect.
- Perform in-class design reviews of student solutions. Design is largely a process of synthesizing a solution to a problem. Reviews complement this activity by stressing analysis. Moreover, public reviews would encourage teams to write higher quality code.
- More strongly formalize the reusability aspects of the project. Require justification for the selection of reused code. Specifically track the amount of reuse and the amount of adaptation required.