ABSTRACT
In this paper we revisit previous research to elaborate on the question: “Can graduating students design software systems?” The work concluded that the answer was “not really”. We wished to determine if this was true currently at our institution and also to look at whether students were able to design software in groups, and evaluate others’ designs. In summary, it appears that our students, just as in the original experiment, struggle with doing design, even in a group situation. The representation of behavioural design was particularly lacking. That said, students were able to recognize weaknesses when evaluating other group designs. Based on our findings, we provide several pedagogic recommendations.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computers and Information Science Education - Computer Science Education.

General Terms
Measurement, Experimentation. Design

Keywords
Design, Software Engineering, Student Performance.

1. INTRODUCTION
In this paper we revisit research that started in 2003-2004 as part of the ‘Scaffolding’ experiment [11]. This was a multi-national, multi-institutional study that analysed software designs collected from different populations of students and educators.

One aspect of the analysis resulted in a paper published in 2006 [6], hereafter called the ‘Graduating Design paper’, that specifically asked the question “Can graduating students design software systems?” The authors of the paper looked at 149 designs produced by students nearing graduation, and concluded that “the majority of graduating students cannot design a software system.”

To reach this conclusion they placed each design into one of six categories, slightly abbreviated in Figure 1, which also shows the results. The conclusions, showing that over 60% contributed little progress towards a design, were clearly sobering for educators.

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, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCSE’11, March 9–12, 2011, Dallas, Texas, USA.
Copyright 2011 ACM 978-1-4503-0500-6/11/03...$10.00.
context: “Are students able to evaluate others’ designs, and recognise good and bad designs?”

Section 2 examines the relevant literature and looks further at the design categories in Figure 1. Section 3 describes the data collection and analysis. In Section 4 we look at the results, and Section 5 presents conclusions and pedagogical implications.

2. BACKGROUND

It is evident that we must enable our students to acquire the skills to undertake modelling using the appropriate notations and methodologies, and at the right level of abstraction so they can understand and communicate software problems and their solutions. We usually do this with fairly simple designs built from scratch. We note that most real-life problems involve maintaining existing large and complex systems, and it is even more difficult to teach students how to solve these “wicked” problems [5]. In this paper, as in the Graduating Design paper [6], we restrict ourselves to the simpler case (see Figure 2).

The categories of design achieved in the Graduating Design paper were correlated with the number of computer science courses completed. Those who had a ‘complete design’ had taken on average 17 CS courses. Moreover, students generally producing better designs took longer (students were allowed to spend as long as they liked). Complete and partial designs averaged 60-65 minutes; first step designs, 42 minutes; and the other categories less time [6].

The Graduating Design paper [6] followed Soloway [10] in advocating the following five phases in developing a design: understanding the problem specification, decomposing the problem into programmable goals and objects, selecting and composing plans to solve the problem, implementing plans in language constructs, and reflecting and evaluating the final artefact and process. The paper noted that the first phase can be mapped directly to the Restatement and Skumtomte categories. More information on the category relationships is found in [7].

Another approach is to consider what aspects of design are typically needed, and what artefacts those aspects lead to. All software development methodologies stipulate the analysis and design artefacts that should be produced and when. For example, the IBM Rational Unified Process (UP) [9] takes a use-case driven approach where UML use cases are initially expressed in their diagram and textual form, but individual use cases are then realized as activity diagrams, class diagrams and others. The UP captures analysis, behavioural and static structural information, and both for detailed and architectural views of a system.

Although agile methodologies place more emphasis on code and less on analysis and design artefacts, they still employ such artefacts [4]. For example, Agile Modeling [1] uses the full range of UML diagrams, as well as other kinds of diagram, e.g. GUI story boards. Much less emphasis is placed on software tools and more on using whiteboards and paper. Diagrams are only drawn to help one understand and solve a problem or to communicate with others, but not drawn for their own sake. Even Extreme Programming [2] advocates the use of CRC (Class, Responsibilities and Collaborations) [3] sessions involving brainstorming activities where class and communication diagrams are used, but also in a lightweight fashion.

What is clear from all these methodologies is that modelling using diagrams is required to represent the functional and non-functional analysis of a system (use cases or similar), the behaviour and structure of a system, and at both an architectural and detailed level. Furthermore, most methodologies explicitly encourage linkage between artefacts, and many are iterative and incrementally add to earlier models during the course of a project.

Figure 2: Scaffolding Design brief, abbreviated on [6]

3. DATA COLLECTION AND ANALYSIS

Data was collected from students who were taking a compulsory final year, semester one course entitled ‘agile methodologies’. All of the students had taken a curriculum of at least a year and a half of full-time computing including courses on Java programming, software design, database and data structures. Some students had also had a year-long internship experience. In their first year, the students had been introduced to UML through use cases, class and instance diagrams in their first course, and sequence diagrams in their second course. In addition, they had looked at object-oriented design patterns in their second year and were being introduced to refactoring and anti-patterns in the Agile methodologies course.

3.1 Group Design – part 1

In the first part of the experiment, in week nine of the semester, students were organised into six groups, each of which contained about 10 final year students. Each group then had about 50 minutes to produce a solution to the same exercise that was used in the Graduating Design paper, outlined in Figure 2. The students were told (as in the original experiment):

Getting People to Sleep

Your brief is to design a "super alarm clock" for University students to help them manage their own sleep patterns, and also to provide data to support a research project into the extent of sleep problems in this community. You may assume that, for the prototype, each student will have a Pocket PC (or similar device) which is permanently connected to a network.

Your system will need to:

- Allow a student to set an alarm to wake themselves up.
- Allow a student to set an alarm to remind themselves to sleep.
- Record when a student tells the system they are about to sleep.
- Record when a student tells the system that they have woken up, and whether it is due to an alarm or not.
- Make recommendations as to when a student needs to go to sleep. This should include "yellow alerts" when the student will need sleep soon, and "red alerts" when they need to sleep now.
- Store the collected data in a server or database for later analysis.
- The server/database system (which will also trigger the yellow/red alerts) will be designed and implemented by another team. You should, however, indicate in your design the behaviour you expect from the back-end system.
- Report students who are becoming dangerously sleep-deprived to someone who cares about them (their mother?).
“In doing this you should (1) produce an initial solution that someone (not necessarily you) could work from (2) divide your solution into not less than two and not more than ten parts, giving each a name and adding a short description of what it is and what it does – in short, why it is a part. If important to your design, you may indicate an order to the parts, or add some additional detail as to how the parts fit together.”

The designs were produced on flip chart paper, collected and photographed.

In the original experiment the time was not fixed and complete designs took on average 60 minutes, but this was not achievable within the constraints of our course. We note that in our experiment the groups all finished before the time was up. We also note that not all members of each group contributed to the same degree and that credit was given just for attending the session, although on the whole participation was quite active.

3.2 Evaluation of Designs – part 2

As an individual exercise, the students were then presented with the designs from the ‘other’ groups. They were asked to evaluate each design and determine which was the best design and why. As one of these ‘other’ designs, the authors included a design that we produced ourselves, based on a solution shared by the authors of the Graduating Design paper. The students were not aware of which group produced which design, or of the extra design. Evaluations of the designs were obtained from 47 students.

3.3 Analysis

The authors looked at the seven group designs and categorised them using a modified version of the criteria in [7] and rooted in the discussion in Section 2. The categorisation shown in Table 1 is based on an attempt to answer the question: if a system architect was presented with a design for a system that the architect was about to maintain or build, then what information would s/he hope to find in the perfect design document?

Table 1 summarises the artefacts as marking criteria used to assess a design, providing relative weights for each design area. The artefacts shown are possibilities rather than absolutes. We considered other artefacts so long as they conveyed useful information for a given design area.

Although not shown in Table 1, each design area has a discrete set of nominal marks (0, 0.25, 0.5, 0.75, 1) used when evaluating a design. The ‘real mark’ awarded for a design area is the nominal mark multiplied by the weighting. The total real mark awarded for a design will range from 0 to 10. The six categories of Figure 1, from the original Graduating Design paper, are mapped to real marks in Table 2.

Our motivation for using the marking criteria of Table 1 and then mapping to categories was to find an objective way to evaluate designs, and to find a scheme that could be applied to very different kinds of design. By using weightings, we are able to tune marking criteria based on context. For example, a small system should place less emphasis on architectural design than a large system comprised of many subsystems. The emphasis on behavioural versus structural design also depends on the nature of the system being constructed.

The Getting People to Sleep problem places particular emphasis on message passing behaviour, whereas its static structure is relatively simple. The relative importance persistence or user interface plays in the design will also depend on the nature of the problem. Likewise, the security design area will vary in importance (here, we gave it a weighting of zero as it was not mentioned in the problem statement). These weightings also recognise that if software engineers are provided with limited time to undertake a design exercise then they should focus on the key design areas for the given problem.

Table 1. Marking criteria for the group designs with weightings that reflect the Getting People to Sleep problem

<table>
<thead>
<tr>
<th>Design area</th>
<th>Possible artefacts</th>
<th>Weight (total = 10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: Problem analysis</td>
<td>Use case diagrams with actors linked. Use case descriptions. Stories with stakeholders and business value identified. Interaction scenarios. Functional and non-functional requirements.</td>
<td>2</td>
</tr>
<tr>
<td>B1 Architecture</td>
<td>Sequence, communication, activity diagrams or similar showing major flows of control through system.</td>
<td>1.5</td>
</tr>
<tr>
<td>B1b: Structural</td>
<td>Component diagrams with associations or similar showing major parts of system.</td>
<td>1</td>
</tr>
<tr>
<td>B2: Detailed</td>
<td>Key algorithms (code, pseudo-code, data flow or activity diagrams...). More detailed sequence diagrams or communication diagrams for more significant object interactions. State diagrams.</td>
<td>1</td>
</tr>
<tr>
<td>B2b: Structural</td>
<td>Detailed class diagrams (classes and associations) exploding components from architecture. Identification of boundary, control, entity classes.</td>
<td>0.5</td>
</tr>
<tr>
<td>B3: Database</td>
<td>Kind of database: relational, XML, non-SQL. Design of structures: tables, ER diagrams.</td>
<td>1.5</td>
</tr>
<tr>
<td>B4: GUI</td>
<td>Mock-ups. Storyboards</td>
<td>0.5</td>
</tr>
<tr>
<td>B5: Security</td>
<td>Relevant discussion of authentication, authorisation, confidentiality, data integrity, security auditing; XSS, XSRF, SQL injection.</td>
<td>0</td>
</tr>
<tr>
<td>C: Linkage of artefacts</td>
<td>Linkage by placement, words or lines between: analysis and design artefacts; architectural and detailed design artefacts; GUI and other design artefacts; database and other design artefacts; structural and behavioural design artefacts.</td>
<td>2</td>
</tr>
</tbody>
</table>
Table 2. Mapping of real marks to original categories

<table>
<thead>
<tr>
<th>Real mark range</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Nothing</td>
</tr>
<tr>
<td>&gt;0, &lt;2</td>
<td>Restatement</td>
</tr>
<tr>
<td>≥2, &lt;4</td>
<td>Skumtomte</td>
</tr>
<tr>
<td>≥4, &lt;6</td>
<td>First step</td>
</tr>
<tr>
<td>≥6, &lt;8</td>
<td>Partial design</td>
</tr>
<tr>
<td>≥8, ≤10</td>
<td>Complete design</td>
</tr>
</tbody>
</table>

Development methodology may also affect weightings. An agile methodology will discourage overly detailed requirements analysis or big design up-front [2], but rather take an iterative and evolutionary approach to finding both requirements and design. Many methodologies also place less emphasis on documenting the design [1, 4]. These methodologies the customer is expected to remain an integral part of the development process providing revised requirements as understanding improves. An architect joining such a project would expect to see fewer detailed UML diagrams, partial capture of requirements reflecting the current iteration cycle, and more emphasis on customer-level and developer-level tests. In contrast, a traditional, phased-based waterfall-style methodology will tend to produce large amounts of documentation of all types.

The above factors should therefore be taken into account when applying the marking criteria to any design weightings. For the Getting People to Sleep problem we assume a fairly traditional methodology, but one where time is limited.

4. RESULTS

In this section we look at our analysis of the student groups’ designs and also their analysis of each other’s (and our) designs.

4.1 Group Designs

The categorisation of the group designs, based on the marking criteria in Table 1 is shown in Table 3 (marks of zero omitted).

Two of the authors independently analysed and categorised the six groups and the authors’ design. The marks awarded were compared and found to be similar. One author missed some static detailed design information for Group E. After correction, the authors’ categories for each group were the same.

As possible with just six designs to compare, our groups did as badly as those in the original experiment. That said, the way groups were categorised was different and the time allocated to our students was fixed at 50 minutes (it took 60 minutes on average for complete designs in the original study.)

The worst design (Group C) had two sheets that largely restated the problem and one (Figure 3) with an unconventional use case diagram for part of the problem. The mark correctly suggests that the focus is restatement. While some designs included vague class diagrams at best (Figure 4), most included none.

The two designs that had the highest marks (Group F and ours) were complete in more areas, and were the only designs with any linkage between parts (Figure 5).
Overall, the groups were strongest at analyzing the problem and providing use case diagrams. Despite the fact that class diagrams are an important part of the design curriculum starting in their first year, two groups (C and D) had nothing resembling a class diagram and other groups had only sketchy ones – mainly names but no detail or relationships.

Structural design was stronger than behavioural design, the latter being mainly absent in all but group F’s and the authors’ submissions. Having identified the main use cases, groups did not pursue them into answering the question ‘how would our system do that?’

### 4.2 Evaluation of Designs

Table 4 shows the results of the evaluation exercise – students who said that either of 2 (or 3) designs was ‘the best’ were counted as .5 (or .33). No favourite was chosen by 6 students.

<table>
<thead>
<tr>
<th>Best design to implement from</th>
<th>Number of students</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2+1*.33</td>
<td>2.33</td>
</tr>
<tr>
<td>B</td>
<td>0+2*.5</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1+1*.33</td>
<td>1.33</td>
</tr>
<tr>
<td>E</td>
<td>2+1*.5</td>
<td>2.5</td>
</tr>
<tr>
<td>F</td>
<td>16 +3*.5+2*.33</td>
<td>18.16</td>
</tr>
<tr>
<td>G (authors)</td>
<td>13 +4*.5+2*.33</td>
<td>15.66</td>
</tr>
<tr>
<td>None identified</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

We note that the design which we evaluated as ‘restatement’ received no support and that the two designs we considered to be reasonable designs were indeed rated best by the students. Design F was seen as the best by the most students (it was also the longest and had the most arrows which may have had an effect). A disproportionate number of the students who chose design G as best were in group F so not able to choose their own.

Students’ discussions of the designs varied between very superficial:

S10 on design F: “This design was the most in-depth and had nice pictures.”

and quite thoughtful and perceptive:

S12 on design G: “…the group has expanded to designing the database schema and including sequence diagrams to explain the relationship between system functionality and database design.”

Some students implied a use case diagram fully describes problem behaviour:

S15 on design D: “A good use case diagram that covers the major behaviour in the brief”

Students recognized the weakness of design C, even using similar terminology to ours:

S21 on design C: “There is a very simple use-case diagram and another image that just re-iterates some of the points given to us in the brief.”

On the other hand, some students seemed to be confused about what a design was for:

S31 on design C: “Design C sets out clearly what the critical levels of alert are … [but] more design artifacts should be used to tell the programmer or system architect how to build such a system.”
5. DISCUSSION
What did we learn or contribute from this exercise? The first thing is that our experiment confirmed the results of the Graduating Design paper. It appears that many graduating students cannot design software systems. This was very depressing for the authors. We learned that our current crop of students did not do as well as we expected in the design exercise, even when working in groups. Only one student group produced what we considered to be a reasonable design. We note that the original experiment took volunteer students from each institution who may be of a higher caliber on the whole, whereas we used a complete cross section of student abilities. Another threat to the validity of this experiment is that no consideration was taken of group structure or dynamics. We had hoped that working in a group would lead to better solutions, but it may have hindered the design process as opposed to helped it.

Second, our marking criteria elaborates the simpler categorisation of the Graduating Design paper and is useful for assessing designs developed under different software engineering methodologies.

Finally, our analysis has implications for teaching and learning. The main thing that was lacking in the student designs was any attempt to describe behaviour of the system under consideration, leading to our first teaching recommendation (TR):

TR1 More emphasis on behaviour when teaching design.

While the student groups did seem to be able to construct use-case diagrams, they did not relate them to further artefacts in the design. There is a danger when using use-case diagrams that they become simply restatements of the problem, and that is particularly likely if they are not linked to something else in the design – probably a sequence diagram:

TR2: Stress that use-case diagrams need to be linked with other artefacts.

On a positive note, students did seem, on the whole to be able to recognize better designs. We now feel that this exercise should be introduced much earlier into the curriculum. Did the students in group C think what they did was a good design at the time, and what would they think once they had seen other designs? This fits in with the literature on peer-marking: that students contribute to the learning of other students [8]:

TR3: Give students examples of other students’ designs so that they can see how they are doing and ‘up their game’.

6. ACKNOWLEDGMENTS
Thanks to our students for allowing us to use their work and to the authors of the Graduating Design paper for advice and support.

7. REFERENCES