Comments Please see the attached report.

COMP61511: Software Engineering Concepts *In Practice*
Fall 2015 Exam General Feedback

Bijan Parsia, Course Leader and Marker

February 15, 2016

1 Exam Overview

The exam consisted of

- 23 objective questions (4 true/false (TF), 19 multiple choice questions (MCQs))
- 1 point each for a total of 23 points
- 3 essay questions
- Two questions were worth 5 points each and one worth 6 points for a total of 16 points

Total possible points: 39

2 Overall Performance

Overall, exam performance was rather good, with the mean and the median closely aligned at around 62% (which, for overall grade, would be a "merit". The range was large with a large gap between the min and max value with the min (38%) reaching a bit low adn the high (88%) reaching a bit high. This seems to align with the wide range of experience and background of this cohort. In future years, a challenge will be to narrow that spread a bit (corresponding to more support for the less prepared *while* giving more challenge to the best prepared).

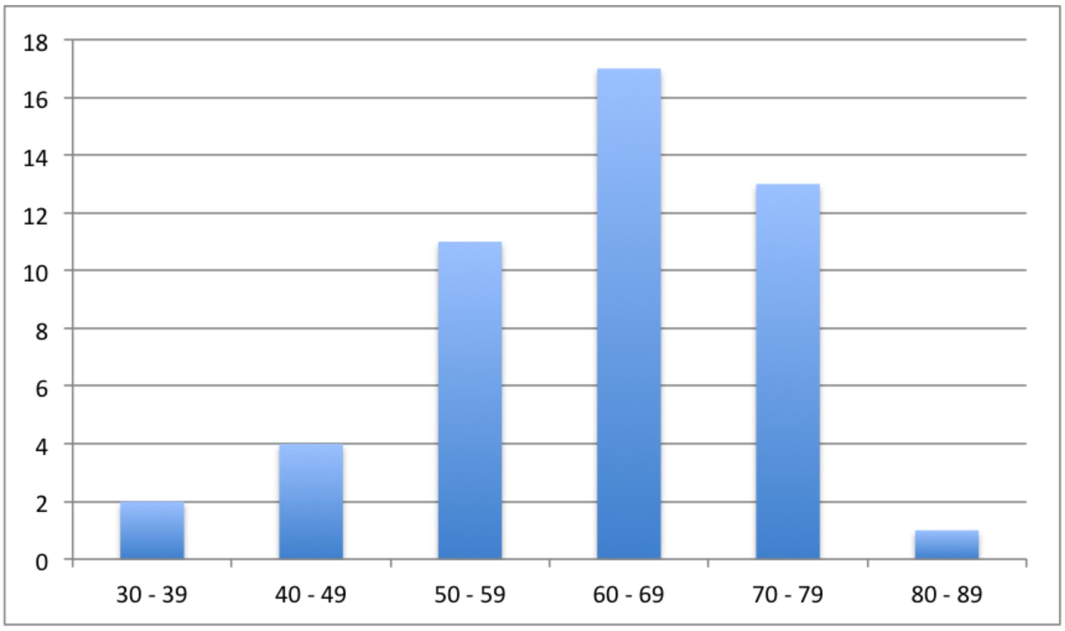
	Raw Score	Percentage
Minimum Value	15	38%
Maximum Value	34.5	88%
Range	19.5	50%
Average	24.34	62%
Median	24.75	63%
Standard Deviation	4.39	11%

As we can see from the graph, the distribution skewed a bit right. There were 13 exams in the distinction range (70-79).

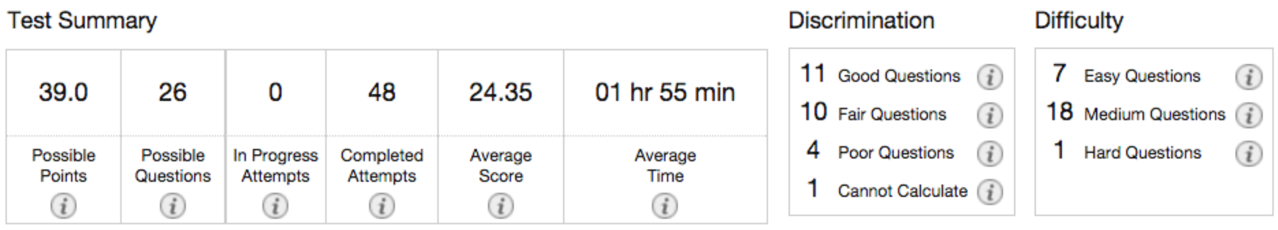
The average time taken was close to the full allotemnt (1 hr 55 mins out of 2 hrs). Unfortunately, finer details are not currently available from Blackboard, but some conversation with students suggests that the exam might be a bit long (esp. with third essay). This is also suggested by the fact that there were slightly more 0s or partially completed questions in the last essay.

In terms of difficulty breakdown, the strong majority of the questions are in the (desired) "medium" level, which means that between 30-80 of exam takers got it correct. Generally, instead of having the entire exam at the "medium" level, I prefer to have a bit of progression from some easy, to mostly medium, to some hard, which this exam reflects.

Discrimiation (i.e., whether student performance on a particular question "tracks" their performance on the rest of the exam – a question where students who did overall poorly on the exam did better than students



who did overall well on the exam exhibits poor discrimination) was also quite reasonable, though there's a wide band of "fair" quality questions. This may be due to the large cluster of grades in the middle and relatively few questions overall.



Test Summary

39.0	26	0	48	24.35	01 hr 55 min
Possible Points	Possible Questions	In Progress Attempts	Completed Attempts	Average Score	Average Time

Discrimination

11	Good Questions
10	Fair Questions
4	Poor Questions
1	Cannot Calculate

Difficulty

7	Easy Questions
18	Medium Questions
1	Hard Questions

3 MCQs

Question 4 was about the relation between product and project success and the correct answer got the *lowest* selection. (12.5%) The other options were fairly evenly distributed which suggests that none were inherently misleading. However, this question is *linguistically* complicated and will be revisted.

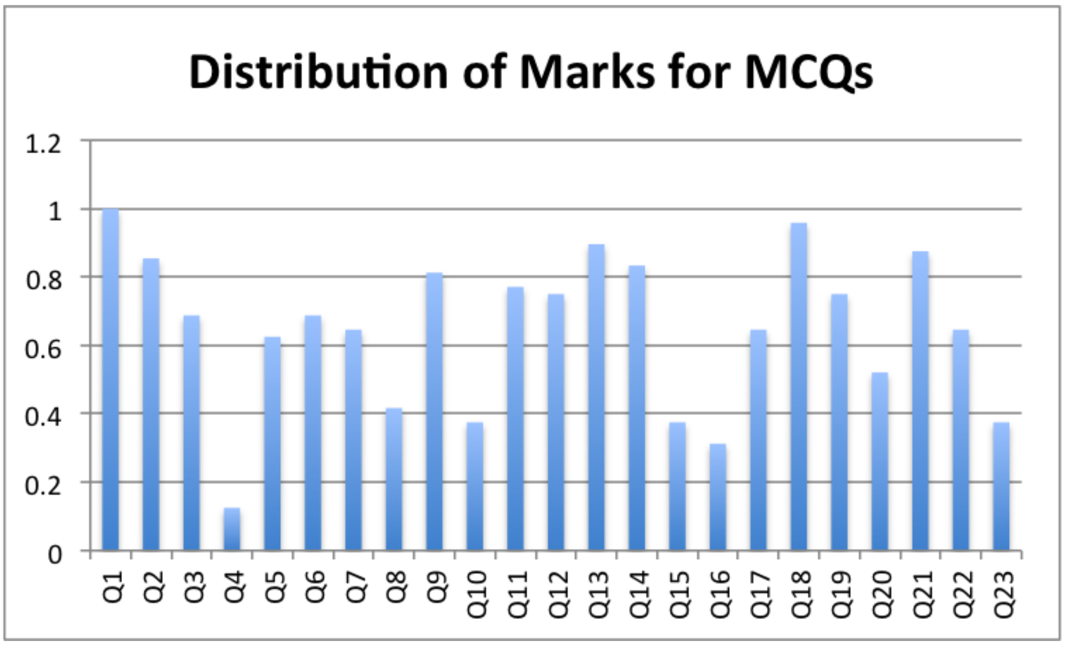
For Question 10 many people thought that *component test* was a *kind* of integration test. But integration tests test *muliple* components.

Question 15 was a straight recall question. Many people decided to infer that @Before and @After indicated test sequencing. Of course, the desirability of test *independence* would have blocked that inference, so analysis could rescue lack of specific recall.

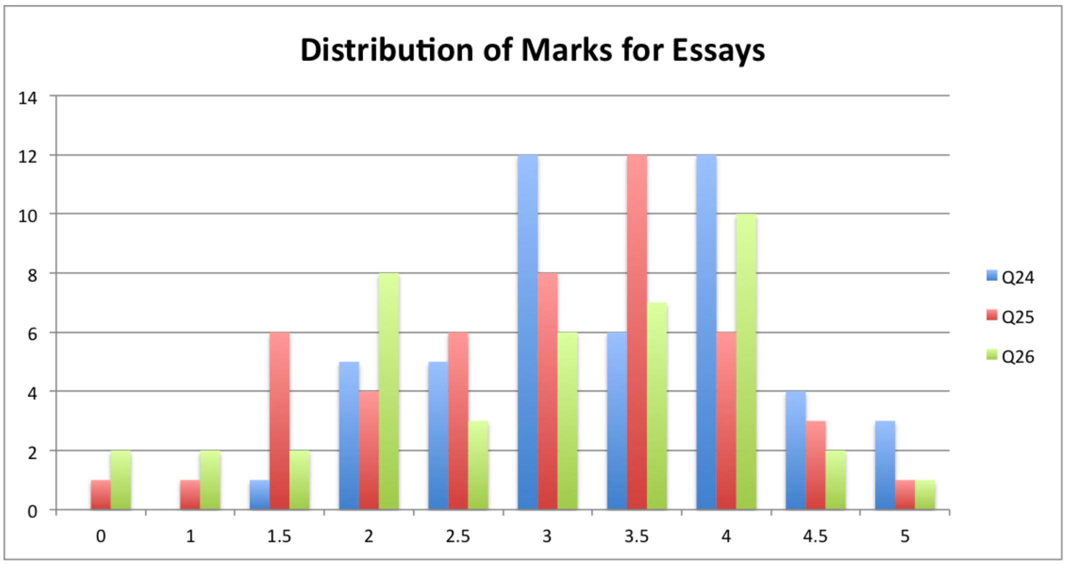
Question 16 was intended as a fun history question. Surprisingly, some people answered "played Glinda the Good Witch in the Wizard of Oz." which indicated some guessing going on.

Question 20 was *extremely* disappointing in that it claimed that inheritence is a form of composition. We spent serious time in class and had a coursework essay on it. This was intended as a 100% pass question. This performance suggests that a new approach is needed for this topic, which is super important.

Question 23 perhaps was a bit misleading. The top two answers both included the idea of rapid change (but so did the third most popular answer) which is the basic right idea. But "jargon" isn't the key issue (that's just changes in terminology).



4 Essays



Q24:
People did best on this question with an average of 3.38.

A common problem was people taking the canonical generica example from class of needing a quick bug fix which entails a violation of the SRP. Note that this isn't a specific example, it's generic! You need to dig in a bit more to get clear on what might be fixed by making one class violate SRP (think about it: what so of bug is fixed by adding responsibilities to a class?) If you just do the generic, it's hard to get above a 3 or 3.5.

Following up, in general, it's rare for an essay question to target recall knowledge. So repeating a generic scenario from class is very unlikely to be sufficient. In general, the point is to flesh out the scenario, not merely repeat it.

Even if it is nominally your own scenario, if it's generic it is weak. E.g., "For my project, I created a class with 2 responsibilities violating SRP because I didn't have much time" doesn't actually add concrete detail! You need to say what the responsibilities were.

Furthermore, we'd really need specific reasons why it's faster to violate the SRP. If there's a new responsibility isn't it just as easy to create a new class? It's not very slow to do so! So there has to be some "nearness" of functionality which makes this plausible.

Q25: The average here was 2.95, making it the most difficult essay.

If you don't know what a diseconomy of scale is, it will be challenging to do well on the question. This happened a number of times and I tried to give points where I could. But if you didn't put anything about unit costs it would be very challenging to get above 2.

It's not evidence of diseconomy of scale if the total price of the software (or a class) goes up as the size goes up. Even with economies of scale, unit cost doesn't go negative. That is, the marginal cost of each additional unit is positive. So if I make 100 widgets that costs less than 1000. However, making a lot of 1000 (i.e., in one manufacturing run) will cost less than 10 runs of 100 if we achieve economies of scale. Similarly for diseconomies. Unit price needs to go up!

Note that there are lots of good software engineering practices but not all of them address diseconomies of scale! Also, just mentioning them doesn't explain how. E.g., most of the SOLID principles are promoting modularity which is what mitigates diseconomy of scale.

One solution proposed a lot is to have clear and complete and unchanging problem definition and requirements upfront. The biggest problem with this, of course, is the wicked problem aspect of software construction.

Q26:

People definitely had time budgeting issues for this question, though the average was a bit better than Q25 at 3.08.

It's important to link the wickedness to the staging. Remember that the stages don't have to happen linearly in any situation!

People focused on changes in requirements. Problem definition changes, by themselves, don't make for a wicked problem. For example, if today a customer requested a Flash implementation of a game and after you finished it they changed their mind and wanted all HTML5 tech instead (because they read that it was the new cool), this isn't an example of being wicked because the solving of the problem didn't contribute its definition. Contrariwise, if the experience of using the Flash game was what made them realise that they didn't want a lot of animation or the "flash feel", that would be wickedness, because the clarification came through the exploration of the solution.

Saying "wicked" a lot doesn't help. E.g., (made up example) "Software is very difficult and expensive so that makes it wicked to design". You need the actual meaning of "wicked problem"... "wicked" doesn't have separate usage.