PGT Exam Performance Feedback 2017/2018 Semester 1

COMP61511 Software Engineering Concepts in Practice

Bijan Parsia
Christos Kotselidis

Comments Please see the attached report.

Exam Feedback

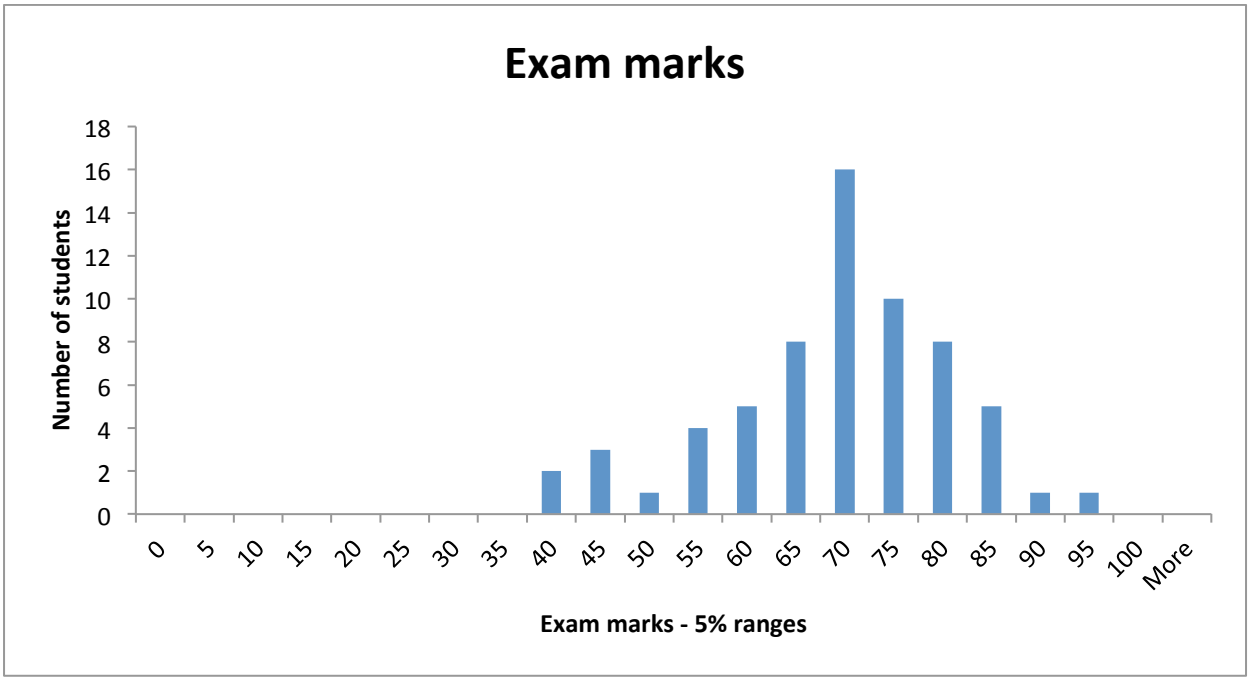
This feedback is based on the **COMP61511 exam** that took place in **2017/18**.

Overall Marks

Average Exam score: **26.88** (**67.19%) +/- SD 11.84%**

The average is atypically high which seems due to 9 easy multiple choice questions as well as the essays having a slightly higher average. This may be due to new instructional techniques for essay writing (videos, peer review, etc.)

Distribution of Marks



We had some of the highest marks ever for this course (the highest prior was 85). This suggests a strong cohort and a slightly too easy exam.

Average Exam Time: **1 hr 53 min** which is fairly typical.e

Question Difficulty

Below is how Blackboard defines question difficulty.

Difficulty: The percentage of students who answered the question correctly. The difficulty percentage is listed along with its category: Easy (greater than 80%), Medium (30% to 80%), and Hard (less than 30%). Difficulty values can range from 0% to 100%, with a high percentage indicating that the question was easy. A well constructed exam aims for most questions to be of medium difficulty with a few easy (to establish a floor) and a few hard (to help discriminate at the high end). Our exam needed a few fewer easy and a few more hard to be in balance.

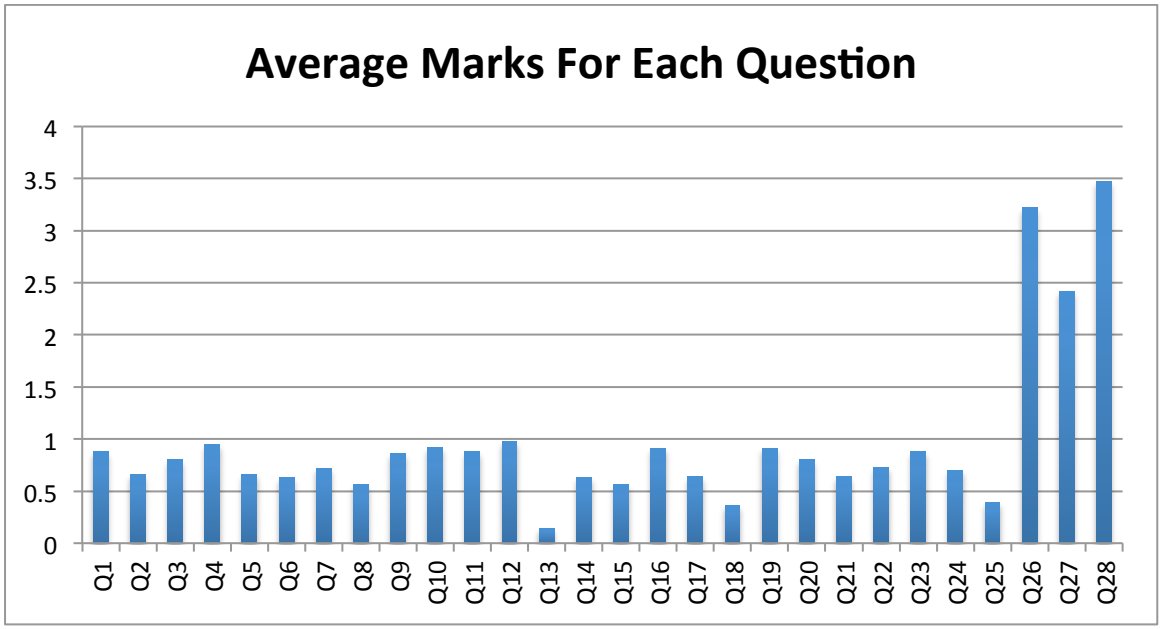
9	Easy Questions
18	Medium Questions
1	Hard Questions

Discrimination

Discrimination: Indicates how well a question differentiates between students who know the subject matter and those who don't. A question is a good discriminator when students who answer the question correctly also do well on the test. Most questions exhibiting good or fair discrimination means that, by and large, the exam doesn't have any weird or trick questions.

15	Good questions
10	Fair Questions
3	Poor Questions
0	Cannot Calculate

Distribution of Marks by Quesiton



Multiple Choice Feedback

All MCQs had a maximum mark of 1 while the three essays had a maximum mark of 5.

Q13 was the hard question and it seems a bit off in that it depends on how one reads the significance of the "change in functionality" (i.e., as enough to break the refactoring or as not enough to do so).

Q4 was an intended easy question.

Q9 (which of the following is an internal quality) was hoped to be easy, given the emphasis on the quality chart in class, but wasn't in prior years.

Q10 (about when patents come into being) was speculative. In prior years, students didn't pay sufficient attention to the IP lecture. That wasn't the case this year.

Q11 (definition of defect) was also speculative. Prior years had trouble with this, but we shifted emphasis on the definitions which seems to have helped.

Q12 (about what a program crash indicates) was also heavily targeted in class and performance was high.

Q16 was intended to be easy (insuring everyone knew the general principle).

Q19, Q20, Q23 were all basic recall. Given the overall improvement on recall, we'll swap these out.

Essay General Feedback

Q26 (Average: 3.22; Min: 0, Max: 5):

One critical argument in support of the General Principle is the exponential increase in cost of late bugs. Many folks twigged to this, but it needs to be coupled with *how* and *when* a focus on quality helps.

Time is a cost. Some people contrasted time with cost as if you could trade them off. But you can't typically lower costs by adding time (maybe by replacing more expensive time with less expensive but that's a cost-cost trade off).

10-50 lines of software a developer produces on average on a day does *not* take mere minutes. This is a confusion about how debugged LOC are produced.

You need to discuss the paradox, not just explain the principle.

Q27 (Average: 2.43; Min: 0.5; Max: 5):

It is key that the order of activities is (potentially) scrambled or at least iterated. The clearest way to see this is to think of prototypes or walking skeletons. In those, after an initial guess at the problem, you *build a whole solution* which you may throw away or radically revise. This means you do (some) architecture *and* construction before completing problem def.

Many people didn't recall what a wicked problem is.

A wicked problem doesn't have to be solved "twice" or "again". If you fully solved the problem in the first round, then you're done (even if you didn't know what problem you were solving). But since you don't know what the problem is before attempting to solve it, it's unlikely that you'll hit on a correct solution. But you could get lucky!

Q28 (Average: 3.47; Min: 2; Max: 5):

In general students shown a good understanding regarding the principle of "why" we do testing. The majority of the answers covered verification and design while only a few mentioned comprehension.
Testing plays an important role in understanding our system by exercising different scenarios in order to assess how our program behaves.