UNDERGRADUATE COMPUTING PROJECTS – AN INVESTIGATION INTO THE STUDENT EXPERIENCE

Mike Joy
Department of Computer Science
University of Warwick
Coventry
CV4 7AL

ABSTRACT

This paper reports the results of a survey of third year undergraduate computing students undertaking an individual project. Using data gathered from the students themselves, we offer new insights into how students approach such projects. We identify the amount and focus of effort that students experience as appropriate to such an activity, together with the role of the supervisor, and we contrast the methodological approaches which academics advise with those adopted by the students themselves.

Keywords
Undergraduate Student Experience, Software Projects

1. INTRODUCTION

Undergraduate computing degree programmes normally contain a substantial individual project which a student undertakes in their final year of study. In the UK, inclusion of a project is a prerequisite to programme accreditation by the British Computer Society ("BCS") [1].

A computing project will typically take the form of a "software engineering" exercise, where the student will work on an item of software, going through the various stages of the "software life cycle" (specification, design, implementation, and finally testing), writing a technical report describing the software and the process, and outlining their achievements. The ACM and IEEE-CS [2] refer to such a project as a "capstone project" [3], since the student is expected to apply the skills acquired during his/her degree, and demonstrate the holistic approach to project management expected of an IT professional. Such a project in the UK is usually an individual activity, rather than a group project as might be more common in the US.

The focus of this paper is the student expectation of such projects, and the educational experience that students gain from them. The motivation stems from conversations with both staff and students, which suggest that there are discrepancies between what students actually do during their project, and what staff believe they do.

For example, students are taught best practice for writing software, which amongst other things requires that a formal process ("methodology") be used [4]. Computing students usually enjoy writing program code, and tend not to be enthusiastic about what are sometimes perceived as bureaucratic tasks such as completing the documentation that a methodology requires [5, 6]. Although most project dissertations do in fact contain appropriate sections which document the methodology used, we suspect that these are often retrofitted after coding has been completed rather than actively used to support the software development process.

The purpose of the research reported here is to answer a number of such open questions related to undergraduate computing projects, in order to gain an understanding of the student experience.

What do computing students *really* do in their final year project?

1.1 Computing Degrees

The work reported here took place at the author's institution, a research-led university in the UK which supports a medium-sized Computer Science department offering a variety of undergraduate computing programmes. Most of the programmes offer a four year variant (a "Master of Engineering" degree), and all the degrees allow a student to spend a sandwich year in industry (or studying abroad). In all cases an individual project in the third year of study is a core component.

The Computer Science degree programme focuses on software and theory, and accounts for approximately 75% of the students. Several joint programmes are also supported, principally with engineering (where the emphasis is on computer systems and networks) and with business (the application of IT within a business context).

The detailed syllabi of these programmes are broadly similar to those of equivalent degrees in UK research-led universities, and each degree programme is partially or fully accredited by the BCS.

1.2 The Computing Project

The inclusion of a project in the undergraduate curriculum is generally regarded as educationally beneficial. Henry [7 p. 46] identifies several specific reasons, including:

- experience in applying knowledge and skills previously taught in the programme;
- preparation for subsequent employment;
- "self-direction" — development of the student's independent working skills;
- communication skills;
- assessment — "sorting the sheep from the goats".

Computing students must undertake an individual project during their third year of study, which is weighted 30 credits (one quarter of the year's work). The project is "unstructured" [4, p. 14] in the sense that the student is – within reason – free to choose both the topic and the methodology to be used for investigating the topic. Typically, such a project involves the design and implementation of an item of software, although students on joint programmes with engineering may build hardware, and students on joint programmes with business may be involved in activities whose primary focus is business rather than technical. Students are expected to apply good practice they have already learned during the programme, as well as learning any new technologies and other material which may be necessary to progress their work.

Each student is allocated a supervisor, who would normally be from the Computer Science Department, but supervisors from other departments are acceptable for students on interdisciplinary programmes. The normal expectation is that the supervisor will engage with each supervisee regularly, though at the time of this study no detailed guidance was provided as to the length or frequency of such meetings, and the pattern of meetings varies depending on the preferences of both the academic and the student.

There are four "deliverables" — an initial formal *specification* of their project, a short *progress report* at the end of the first term in December, a *presentation* at the end of the second term in March, and a final *dissertation* accounting for 80% of the marks. The presentation affords students the opportunity to demonstrate their software (where appropriate). The dissertation is a technical document of length between 12,000 and 18,000 words (recommended), handed it early in term 3 (late April) following the presentation.

The project is worth 30 credits, which should approximate to 300 hours of work for each student. This is likely not to be evenly spread throughout the year — many students concentrate their activities in the second and third terms (January through May), whilst a few prefer to finish it quickly in order to concentrate their efforts in the second term on other courses.

The students should already be familiar with basic project management skills, which they will have been taught during a core Software Engineering course in the second year of study.

Students tend to take great pride in the success of their projects. A good project is not only satisfying to the student, but it also provides concrete evidence of the student's skills which can be shown to potential employers. For such reasons, some students may devote more than the recommended amount of time and effort to their project, to the possible detriment of the other courses they are following.

1.3 Pedagogy of Computing Projects

Project work is highly valued by computing educators, since it supports students engaging in realistic activities which reinforce their understanding of the discipline, and draws on skills acquired in different modules throughout their degree [8]. For a degree programme to be accredited by the appropriate professional bodies (for example, the BCS in the UK or ABET in the US), the inclusion of such a project in the curriculum is mandatory [1, 2, 9], and accreditation is regarded by the UK Quality Assurance Agency ("QAA") Benchmark Standard for Computing [10] as an appropriate mechanism to safeguard standards.

There is much literature on the delivery of student projects (both individual and group) in a generic context [7, 11], although subject-specific texts are less common. Fincher, Petre and Clark's textbook [12] is a comprehensive guide to computing projects, and is a useful starting point for understanding the educational issues specific to the discipline. This is complemented by the straightforward practical advice to students offered by Dawson [13].

1.4 Computing Project Management

A software project (not necessarily in the student context) is very much oriented to the provision of a software deliverable – "writing a program which works" – and much of the subject matter taught to students relates directly to that process. Projects are "largely about meeting objectives" ([14]. A substantial component of a student computing project is the activity of managing the project, and comprises a variety of transferable skills, including:

- seminar delivery;
- report writing;
- time management;
- interpersonal skills.

1.5 Methodologies

The execution of a software project involves a process through which the project can be planned and progressed in a systematic way in order to ensure successful completion, the so-called "software life cycle". Such a process is referred to as a *methodology*, and the theory and practice of methodologies is central to the discipline known as software engineering.

Many methodologies exist. Perhaps the simplest is the *waterfall methodology* (or *waterfall model*), where the engineer performs a single iteration (known as the software life cycle) taking the project from the initial requirements, through the specification and design of the software, to the implementation (coding) and final testing phase [4].

Another approach is the *Personal Software Process* which focuses on quality assurance by requiring rigorous definition and examination of each software component at each stage of the software life cycle [15].

It is helpful at this stage to distinguish between "traditional" and "agile" methodologies. The former can be thought of as variants of the "waterfall" described above. Some newer methodologies are described as "agile", referring to their flexibility in reacting to changing customer needs. Martin [16] articulates a *Manifesto for Agile Software Development*:

> "Individuals and interactions over processes and tools,
>
> Working software over comprehensive documentation,
>
> Customer collaboration over contract negotiation,
>
> Responding to change over following a plan."

An agile methodology eschews the conventional notion of a life cycle, and instead divides software development into a sequence of short iterations, at the end of which significant changes to the project requirements can be made.

There is currently much discussion of the pros and cons of the different types of methodology, and Stephens and Rosenberg [17] articulate the arguments in favour of the traditional approach. One argument often put forward against student use of an agile methodology is that it encourages programmers to avoid incorporating essential components of a system life cycle into their process — in other words, endorsing a "hacking" approach to their project management. In reality, this may be unfair, since adopting an agile methodology, and *following its process*, may be just as demanding as for a traditional methodology.

As an example of a major difference which would be apparent in a student project is the approach to documentation. A computer program requires documents to support it — the earlier components of a system life cycle, such as requirements specification and design, together with "comments" within the program code, are instances of documentation in a traditional methodology. In an agile methodology, documentation is also encouraged:

> "Software without documentation is a disaster ... the team needs to produce human-readable documents that describe the system and the rationale for their design decisions." [16, p. 5].

However, its place in the process is not a "skeleton" around which the software is built, rather one of many tools which support the software development process:

> "Huge software documents take a great deal of time to produce and even more time to keep in sync with the code ... produce no document unless its need is immediate and significant." [16, p. 5].

Although we, as computer scientists, view our work as grounded in best scientific and engineering practice, there is still a debate as to how we should apply that to our discipline. The conflict between 'traditional' and 'agile' methodologies is likely to remain unresolved for the foreseeable future. We might speculate that students may infer that, because the professionals are undecided, their own approaches to the issue of software development may be equally valid. This view may well be supportable. For example, Torvalds has expressed strong opinions on the software development process. The following quotation is included not just to indicate that there is genuine discussion about the type of methodology which should be used, but to emphasise that the debate is still at a very fundamental level .

> "A 'spec' is close to useless. I have *never* seen a spec that was both big enough to be useful *and* accurate. . . . Specs are a basis for talking *about* things. But they are *not* a basis for implementing software" [18].

As an holistic approach, a waterfall methodology is perhaps sufficient. Other methodologies may be viewed as iterative versions of the waterfall methodology — such as agile methodologies [16] — or waterfall-like but with added formalised process, documentation and supporting software [19, 20]. Furthermore, an individual component of the waterfall methodology may be expanded, for example the initial "requirements specification" may include material related to the business model supporting the project's possible deployment, or the "testing" phase may itself form the focus of the dissertation [21]. A theoretical or mathematical project may also fit into the model, since an abstract analysis of (for example) the speed of an algorithm can be considered as part of the design phase of an item of software.

The important point is not which methodology is used, rather that the student has chosen a methodological approach and applied it appropriately.

2. OPEN QUESTIONS

We have identified questions which will help us to paint a picture of how students experience their projects:

1. Anecdotal evidence suggests that many students devote far more time to their project than the 300 hours the credit weighting would suggest, and that many spend the previous summer working on it. How much time and effort do students actually put into their projects?
   How is this distributed throughout the year?
2. In year 2 of the programme, in the Software Engineering course, the concept of a software development *methodology* is introduced to students, and we strongly indicate that it is "good practice" to use one.
   Which methodologies do students use?
3. Most students present project reports which demonstrate that appropriate good practice has been followed. In particular, software is presented which has been designed using methodologies taught during year two, and fully tested. Anecdotally, it seems that many students code their software, and afterwards write the design of the software. Furthermore, whilst the testing phase of the software life cycle appears in the report, it is often questionable whether the activity actually took place. This is an issue which it is difficult for an academic to decide, since if a computer program appears to work during the demonstration, and there is evidence of detailed design and testing in the dissertation, it may not be possible to evidence at what stage in the process that documentation was generated.
   Do student reports accurately reflect the students' activities?
4. The supervision process is designed to support students throughout their project, however the approaches taken by different academics vary.
   What factors contribute to the success (or otherwise) of project supervision?
5. Students are under pressure to perform well, and this may conflict with their genuine interest in and enthusiasm for their studies.
   What are the principal motivating factors for students?

2.1 Research Methodology

We identified activities which would provide us with data to assist in answering the research questions identified above.

The focus of this work is the analysis of the student perspective on the project process, and central to our process must be the collection and analysis of data provided by students. In practice, this focuses our attention on two principal activities [22, 23]:

- anonymous student questionnaires;
- student interviews.

In order to assist us in establishing the validity of the data collected, two further activities were pursued:

- the process employed at our university is compared with the project management process at comparable Computer Science departments in the UK;
- data obtained from students is compared with data from other sources.

The questionnaire was anonymous, in order to encourage honest answers, and it was therefore not possible to compare individual students' data on their projects with an examination of the documentation and of the project reports themselves. However, some triangulation can be performed (for example, students pursuing a joint degree programme with business would be expected to perceive their project as interdisciplinary). Whilst the honesty of students providing anonymous cannot be guaranteed, nothing in the data we collected suggests that the information they provided us with is unreliable.

The questionnaire was delivered during a lecture late in the second term (in March), attended by the majority of project students, and the number of valid responses received (124) was almost exactly half of the student cohort (249). The Department does not normally take a register of attending students, and the impending presentations in the last two weeks of the term would have affected attendance, but the response rate was considered sufficient for a meaningful analysis.

Semi-structured interviews were conducted with twelve volunteers, and recorded and transcribed prior to analysis.

3. DEPARTMENTAL PERSPECTIVES

Prior to seeking student data, we felt that it was appropriate to compare our own project with the equivalent activity at comparable institutions, in order to establish whether the expectation of our third year project was typical. In particular, we sought to compare the following attributes of the project with other similar institutions:

- the *amount of time* expected for a student to complete the project, and
- the *size of dissertation* which a student is expected to produce.

Thirteen computing departments from comparable "pre 1992" institutions were chosen, each one satisfying the following criteria:

- the department offers a three year "single-honours" programme with a similar name and content to our own, and containing a third year individual project component; and
- the department publishes on its web site detailed information about its degree programme content, including the individual project.

The information on their (public) web sites was consulted at the start of the academic year. Nine of the thirteen included information about dissertation length, although the form of words varied, some suggesting recommended word counts, some recommending a number of pages. The responses – which we summarise here – suggest that our guidelines (12,000-18,000 words excluding appendices) is within the same range as most of the other computing departments:

- 10,000 words (maximum 12,000 *excluding* appendices)
- 40 sides single-spaced (maximum 60)
- 40-60 sides (guideline)
- 50 pages (maximum)
- 5,000-10,000 words (guideline)
- 10,000-14,000 words (guideline)
- 15,000-20,000 words (guideline)
- 6,000 words plus 25 pages appendices (maximum)
- 120 pages including appendices (maximum)

The reason for the variance in the case of the university which specified 5,000-10,000 words is explained by the design and specification components of the project being produced separately to the final report (and so the length of the final report is consequently reduced).

The *amount* of work required varied between 240 and 400 hours, reflecting differences in the size of the project. Finally, we asked what percentage of the project was assessed by the dissertation, and with one exception (the same university identified in the previous paragraph) this ranged between 75% and 100%.

These results suggest to us that there is a broad consensus about the amount of work involved in an individual project (between one quarter and one third of the year's workload), and that assessment of the final written dissertation counts for a large majority of the credit awarded for the project.

The inclusion (or otherwise) of the credit for other assessed components (such as presentations and specifications) in the mark awarded for the final dissertation would appear to be a reflection of the different styles of delivery, and of the administrative processes used.

We conclude that the expectation we have for Computer Science project students at our university is broadly similar to other similar institutions in the UK.

4. STUDENT ACTIVITIES

In this section we consider the activities which students have engaged in during the project, and discuss the individual research questions we have posed.

4.1 Time and Effort Spent on the Project

Henry reminds us that students take their projects very seriously:

> "On most projects the amount of time allowed to do the work is a gross underestimate of that needed. Most students take much longer to complete the project than the number of hours specified. Substantial numbers get so involved with the project they neglect their other course work" [7, p95].

Henry's research centres around projects which are significantly smaller than those on modern computing programmes. She recommends that 5,000 word unstructured projects should be allocated at least 70 hours, which suggests that a computer science project with a median length of 15,000 words should take 210 hours. In practice, students' projects tend to be at the upper end of the 12,000-18,000 range advised, suggesting that 250 hours is more realistic. Our allocation of 300 hours (30 credits at 10 hours per credit unit) is in line with Henry's findings.

In order to ascertain the amount of time spent on the project, we included a question in the questionnaire, and also asked the interviewed students. In the former case, we asked the students to identify the time spent in each term and vacation, so that not only might they be able to offer a reasonably accurate estimate, but also so that the distribution of time spent could be analysed.

The number of hours spent, using the data obtained from the questionnaire, is displayed in Figure 1, and the data from the interviewees in Figure 2.

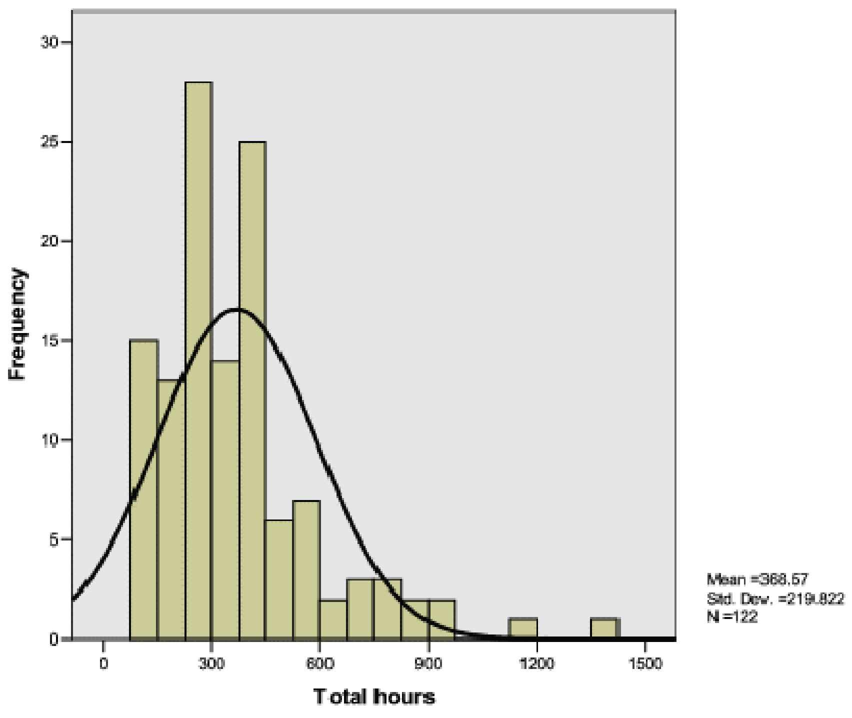

Figure 1: Time spent on project (from questionnaires) vs. numbers of students

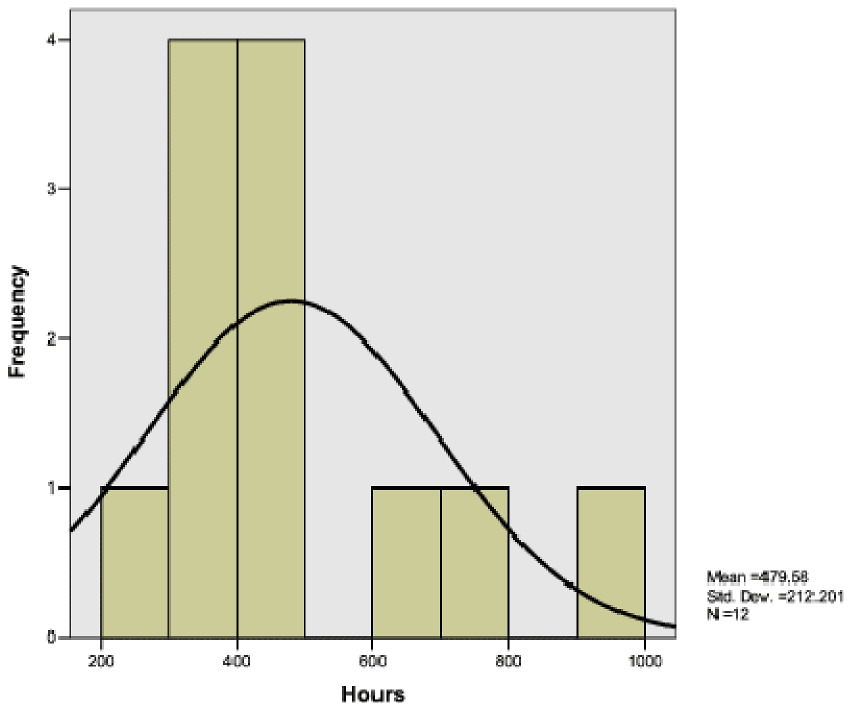

Figure 2: Time spent on project (from interviews) vs. numbers of students

Our hypothesis was that students would spend roughly 300 hours on their project, and both sets of data suggest that students actually tend to spend rather more time (a mean of 369 hours). However, it is interesting to note that 43 students claimed to have spent 400 hours or more on the project (over one third of the respondents), and 14 to have spent 600 or over. If the “outliers” are removed, and the mean time spent is re-calculated over those students who claim to have spent less than 600 hours on the project, then the mean time becomes 307, which is almost exactly what was hypothesised.

The validity of these data initially appeared questionable, especially since several students were claiming they had spent around 1000 hours, the equivalent of 20 weeks full time working a 50 hour week. However, one of the interviewees claimed 1000 hours, and a detailed analysis during the interview of the time he spent did indeed confirm that this figure was realistic. Although this student achieved a high mark, he admitted the time was not well spent:

"It was an area I was interested in, but I didn't know a lot about it, so a lot of it was wasted."

For the 88.5% of students who spent no more than double the recommended time on the project, the distribution of time spent on the project is (approximately) normal, as can be seen in Figure 1, with a mean of (approximately) 300 hours.

Of the twelve interviewees, nine received marks which were first class or high 2:1 (that is, students in the top 30% of their cohort), suggesting that this sample consisted of relatively highly motivated students, and this may explain the high time spent on the project compared to the questionnaire average. The interviewees also had realistic estimates as to how their effort compared with that of their peers, as their comments in Table 1 support.

Hours spent	Comment
250	"Overall about average"
345	"I think I'm between middle and high"
350	"I'm probably about average"
350	"You'd get some students, probably less, but most of the students I'd say equals or more"
380	"A little above the middle"
400	"More than a lot of people"
400	"I probably put in a bit more"
450	"I've put in at least as much if not more hours than most of the people I know, and substantially more than many people"
630	"It just comes across that I've been spending more time on it than the others"
750	For many people ... I've probably done twice as much as they've done in hours"
1000	"Probably more, I'd say, than the majority"

Table 1: Interviewees' views on the time taken Comment

4.1.1 Activity Distribution

In common with most UK universities, the academic year runs from October through to June, and the final year project is submitted towards the end of the academic year. We expected that for most students, their activities would be concentrated during the latter part of the year. We calculated the number of hours spent by each student up to and including the Christmas vacation, as a percentage of the total hours spent on the project, which gave a distribution with a mean of 37.62% and an SD of 15.93% (n=122), suggesting that this was indeed correct.

4.1.2 Activity Profiles

Our expectation was that students would spend the majority of their time working on the "main part" of their project (including activities such as software development), with perhaps 10% each spent researching the topic, 20% conducting other background reading, and another 20% taken by document preparation. The questionnaire data broadly supports this hypothesis, as illustrated in Figures 3, 4 and 5.

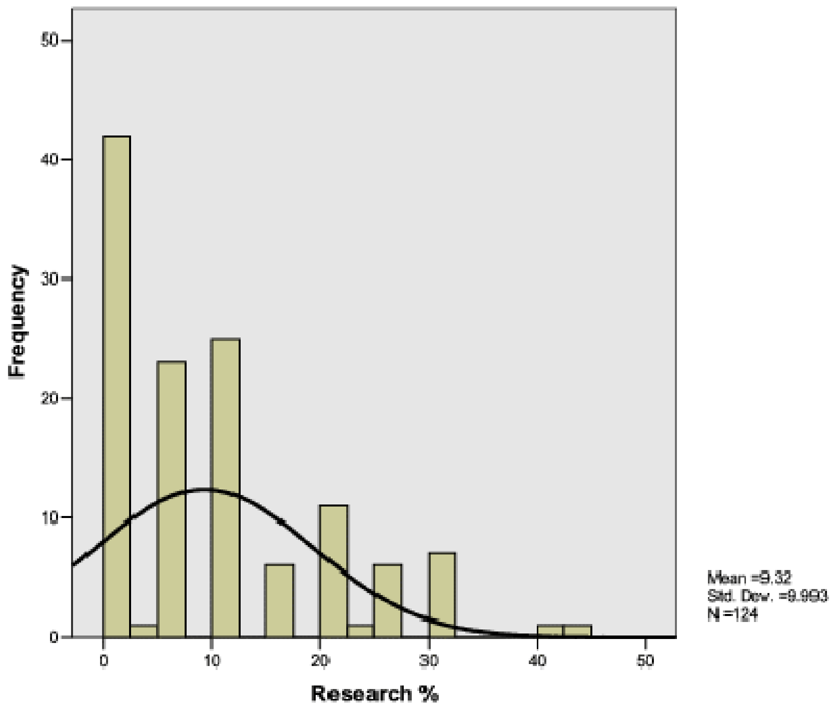

Figure 3: Percent of project spent on research vs. numbers of students

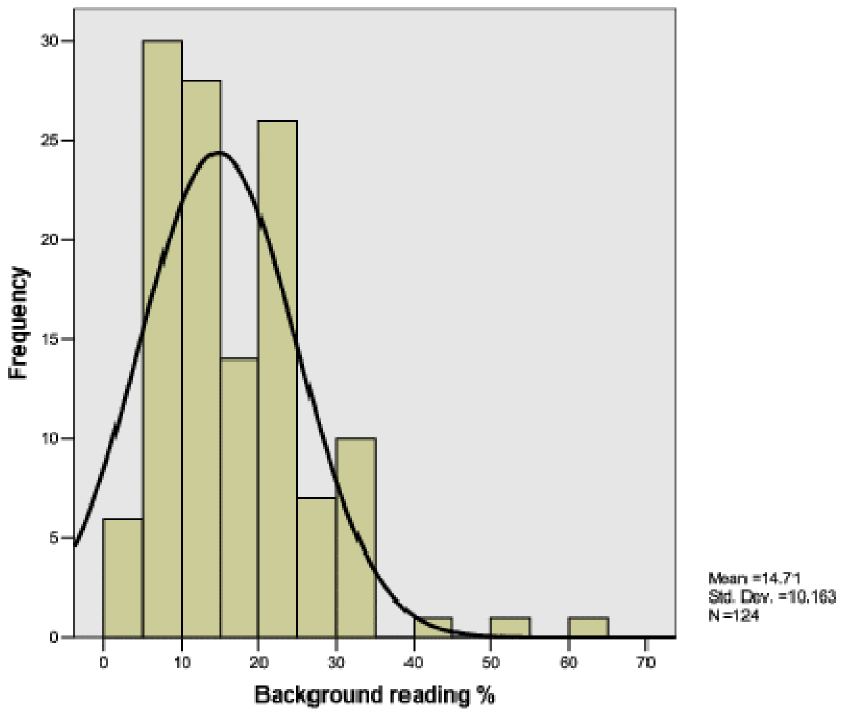

Figure 4: Percent of project spent on background reading vs. numbers of students

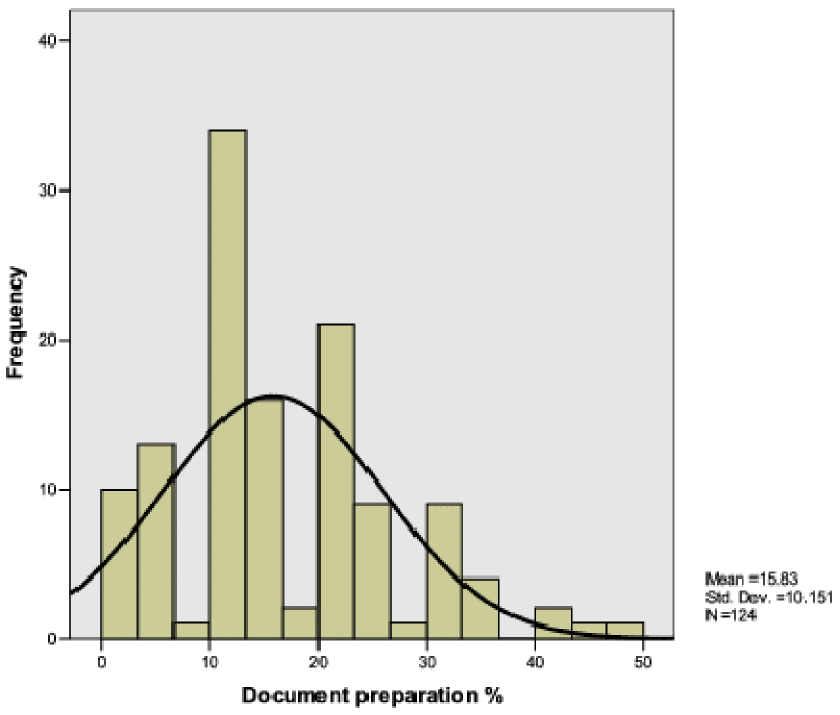

Figure 5: Percent of project spent on documents vs. numbers of students

Our assumption that 300 hours should be spent on the project and that 1/2 hour per week should be spent meeting the student's supervisor, suggests that about 3% of the project time will be dedicated to meetings, and Figure 6 is consistent with this, although a significant number of students were claiming 10% or more of their time on this activity.

However, these particular data should be treated with caution. As a triangulation exercise, we compared the percent time spent on meetings with the number of hours students claimed to have spent in face-to-face meetings with their supervisor, and we expected reasonable consistency. There were substantial discrepancies. For example, of the 13 students who claimed to have spent 0% of their time in meetings, 6 also claimed to have spent at least 5 hours face-to-face with their supervisor. Of the 5 students who claimed to have spent 20% of their time in meetings, two spent 8 hours or less in face-to-face meeting.

The simplest explanation may be that to calculate a percentage the students were using mental arithmetic — they were not observed to be performing calculations on the question sheets or elsewhere, nor to be using calculators — and not all students are skilled at performing accurate mental calculations.

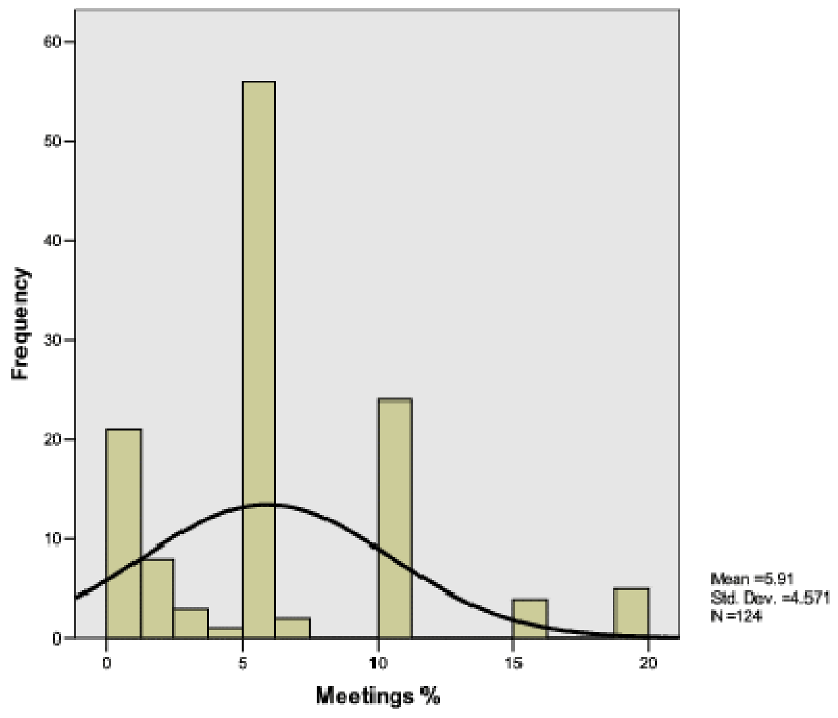

Figure 6: Percent of project spent on meeting vs. numbers of students

Perhaps the most interesting finding was the amount of online searching being performed (Figure 7), suggesting that it has now become a major source of information for students.

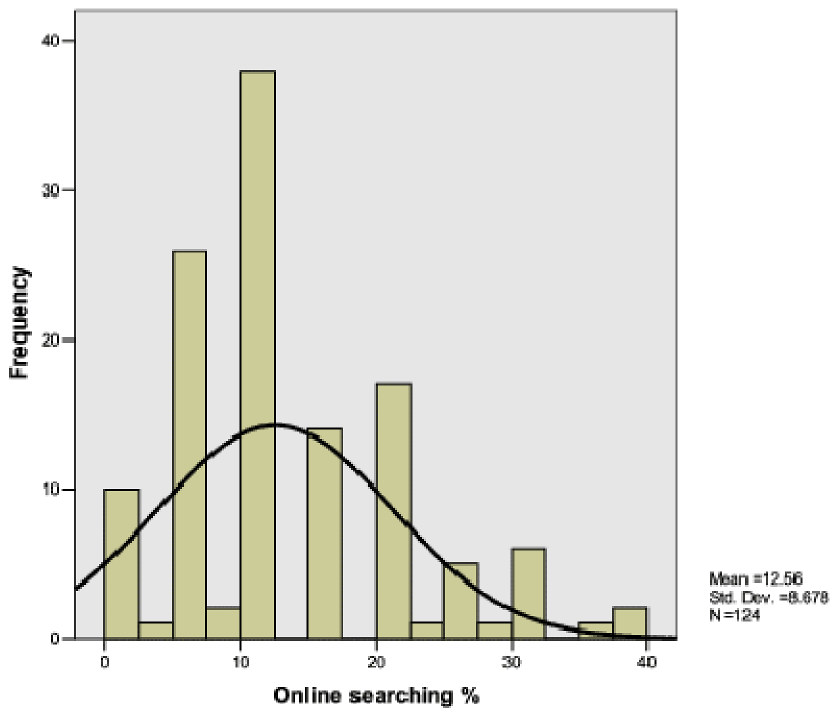

Figure 7: Percent of project spent on online searching vs. numbers of students

4.1.3 Other Data

A search through Staff-Student Liaison Committee records for the last 5 years revealed no substantive discussion of the project, and in feedback forms (n=107) for the last two academic years only 9.3% of students explicitly commented on the workload being high, in particular identifying the "large volume of research" as an issue, and the credits allocated to the course — as one student commented, "A bit too much work for the number of credits".

4.2 Software Methodologies

As we have discussed above, we expect that during the process of software development, the developer will employ a "methodology" to ensure that the software produced meets the requirements of the exercise, and has been properly tested so that errors have been eliminated.

We would expect students to be aware of a number of methodologies in common use, which have been discussed in other core courses. For example, Fusion [19] and its successor the Rational Process [20] are mentioned in our software engineering module, but not taught in detail.

In the second year of study, all computing students are introduced to software methodologies through the Introduction to Software Engineering course, and are required to put their newly-acquired skills into practice by undertaking a group-based project.

Our students are also made aware of agile methodologies including *Extreme Programming* (or *XP*) [16]. We expected that a significant number of students would claim to be using XP whilst not, in fact, subscribing to all fourteen "agile practices" which form the methodology. Indeed, it is impossible for a single programmer working alone to use all of them, since one of them is *pair programming*, which requires programmers always to code in pairs. Furthermore, there is evidence [24] that from both pragmatic and educational viewpoints, a "pick and mix" approach to selecting some of the fourteen agile practices, tailored for different development tasks, is appropriate. Our choice of XP as one of the named methodologies is one which we interpret loosely, and fully expect the respondents to do so also.

The *Empirical Modelling* methodology is one which is essentially only used locally, as part of the research group led by Beynon and Russ [25]. Roughly 20 students were supervised by staff in that research group, so we expected about 10 students would have used Empirical Modelling as their methodology.

The use of the waterfall methodology was expected to be popular, since it is relatively simple. The methodology is not "formal", and variants of it exist, such as the *evolutionary*, *transformation* and *spiral* models [26, pp. 402–419]. We chose the phrase *informal waterfall* in our questionnaire to denote a methodology which might be classified as "waterfall", without being prescriptive about which variant was adopted.

We wished to discover to what extent students had used a software methodology during their project, and to identify which ones were popular. Table 2 identifies the numbers of students who had used a particular methodology "a lot", "some" or "not at all". The final row, "Overall", represents the number who used one or more of the named methodologies (some students used several in combination).

Methodology	**A lot**	**Some**	**Not at all**
Fusion	4	2	122
Rational Process	3	9	116
eXtreme Programming	17	23	88
Empirical Modelling	10	13	105
Informal Waterfall	38	35	55
Other	4	2	122
Overall	**73**	**24**	**31**

Table 2: Numbers of students using each methodology

The numbers of students claiming to use each individual methodology "a lot" is roughly in line with expectations. The number employing Empirical Modelling substantially is half the number known to be doing projects related to Empirical Modelling, and since the response sample is half the number of students, this is consistent. The "informal waterfall" methodology is the most popular, and very few have used named methodologies such as Fusion or the Rational Process.

We had also asked respondents to classify their project on a scale from 1 (practical) to 5 (investigative), and we would expect that the more practical a project, the more likely a student would be to use a methodology. This is illustrated in Table 3 and supported by students' comments in Table 4 and the actual correlation (Spearman) is significant at the 0.05 level (n=126).

Project type	A lot	Some	Not at all
1 (practical)	22	7	7
2	22	8	7
3	16	4	3
4	10	3	5
5 (investigative)	0	3	9

Table 3: Project type vs. overall use of methodologies

Project type	Comment
1	"I used UML; it was kind of because we'd used it in the second year"
1	"Theoretically I would have done; I suppose it's waterfall"
3	"I didn't intentionally use one"
3	"I didn't really use any"
3	"This one is quite unique, I think ... it's not like the thing we were taught during the second year"
4	"Not really ... it was kind of bottom-up"
5	"It doesn't really apply"
5	"Tailor-made"
5	"Very limited - iterative"

Table 4: Student comments on methodologies used for the five project types

These numbers, however, may not be accurate. There appears to be confusion in the minds of some students as to what is meant be the word "methodology". The following quotation suggests the student has not really understood the fundamental difference between an agile process and the waterfall methodology.

> "Agile. Using bits from XP ... there was an initial design phase ... then I'd look at it again and see whether I could add extra design features; so that's a kind of waterfall methodology."

One student discussed how he felt that the use of tools such as UML [27] would have been "overkill", and made the following observation:

> "... when I was at school I lost marks on my project because I wrote the pseudocode after the C++, and I told the teachers that I could think better in C++ than in pseudocode."

One student with an investigative project commented "What do you mean by methodology?". This was an unexpected response, since the concept of methodology is extensively covered in the second year of study. Another, a student, studying on an interdisciplinary programme with business, whose investigative project had a strong focus on business models for Internet-based enterprises, noted that his method "... was very research-based first, but there's no formal methodology I used". These responses suggest that the concept of a "methodology" outside the context of software development is not understood, and indeed research methodologies do not form part of the programme syllabus.

These results suggest that students are generally using software methodologies where their project has a high practical software development component, and it is appropriate to do so, but that for projects with a significant investigative content this is not so much the case.

4.3 Report Accuracy

Whilst students are expected to follow an acceptable methodology for developing their software, many students appear to "retrofit" the methodology after they have completed their software development. It is beyond the scope of this dissertation to examine the reasons for this phenomenon in detail, but we can hope to gain some idea as to the extent it may be happening, and this can be accomplished by examining the "design documentation" produced by the students, since fundamental to the software development process is the understanding that the developer must specify and design their product prior to writing program code.

Only 13.9% of the respondents (n=108) claimed "I have undertaken a detailed design of my software before I started to write code" (response 1), and 23.1% claim to have "undertaken a detailed design of my software, but I interleave the design with the coding using several distinct design and coding stages" (response 2). 15.5% explicitly state to have used an "agile methodology for which the software life cycle is not appropriate" (response 5).

Of the remaining responses, the majority (41.7%) offered response 3 — "I undertake a detailed design of my software, but I mix the design with the coding", and only 5.6% openly admitted that they "document the design of my software after I have completed the coding" (response 4).

The small number of students explicitly retrofitting their design documentation would seem to indicate that our initial concern may have been unjustified. However, the large number of responses (3) was unexpected.

In order to interpret the statement "I undertake a detailed design of my software, but I mix the design with the coding", we must first of all exclude the possibility that the student has used a standard methodology with multiple iteration (response 2) or has used a novel (agile) methodology (response 5). The student is therefore not following established practice, designing and coding ad hoc, and ignoring the "good practice" they have been taught.

There was no correlation between the responses to this question, students' use of particular methodologies, or the type of project being undertaken.

The reluctance of students to adopt more than a "light touch" when applying methodologies to their software development suggests that the "good practice" we teach is not fully appreciated. A solution might be to formally require explicit use of a methodology requiring formal milestones, such as PSP [15], and to ensure that checks are made throughout the process to ensure that students are complying — however, this might be demotivating for the students. Alternatively, the methodologies we present to students may not be completely appropriate for such an individual project, and we might consider introducing a "lightweight" methodology which the students would engage with more readily.

4.4 Student and Supervisor

Contact between student and supervisor is important, and regular meetings help to structure the process and to allow for routine monitoring. Such meetings may take the form of regular appointments, office hours, or student-initiated appointments, and may be individual or collective [12 pp. 88–93].

4.4.1 Length and Frequency of Supervision Sessions

Each student project is supervised by one academic, and we wished to discover how the relationship between student and supervisor works in practice. We assume that the supervisor will be competent to give appropriate administrative and technical advice throughout the project, and we were interested in two aspects of the supervision process — the amount of time spent on the activity, and the content of supervision meetings. We asked students to tell us:

- the time spent with their supervisor (both face to face, and using communication tools);
- how many meetings were arranged each term;
- how each meeting was instigated (by the student, the academic, or as a regular appointment);
- whether the student considered the amount of supervision sufficient;
- how much of the contact related to technical matters as opposed to general guidance.

Many supervisors in the Department suggest that a weekly meeting of 30 minutes duration during term time is appropriate amount for supervision to be effective, but recognize that a variation on this may be appropriate depending on both parties. We wished to discover to what extent this is a guideline which is being followed, whether it is effective, and perhaps more importantly whether it is considered appropriate by the students themselves.

	Low (n=32)		OK (n=81)	
	Mean	**SD**	**Mean**	**SD**
Time spent face-to-face (hrs)	3.69	2.64	9.96	6.85
Time spent online (hrs)	1.66	2.29	1.90	2.43
Total time with supervisor (hrs)	5.35	3.56	11.86	7.66
No of meetings per term	3.90	2.56	6.67	3.17

Table 5: Time spent with supervisor vs. satisfied and dissatisfied student groups

Only two students considered that they had spent too much time with their supervisor, but both of these students' answers may be unreliable. One of the two students claimed to have met their supervisor only once per term, and spent only 2 hours total with them. The other student (who had indeed spent 20 hours total) stated that it was she who instigated the meetings, which is perhaps not consistent with her statement that she had spent too much time. An alternative hypothesis would be that she saw a need to seek extra assistance, but considered that need to be a weakness.

The responses (n=113) for the remaining students are summarised in Table 5 and figure 8, and are presented under the following categories:

- "Low": students who considered they had spent *too little* time with their supervisor, and
- "OK": students who considered they had spent the *right amount* of time with their supervisor.

This data suggests that the guideline of 30 minutes per week is appropriate. Certainly, those students who felt they had insufficient time were mostly well below the guideline. The relatively large number of "OK" students who spent very little time with their supervisor can be explained by noting that many of the students are very able and highly self-motivated, and may not require much supervision.

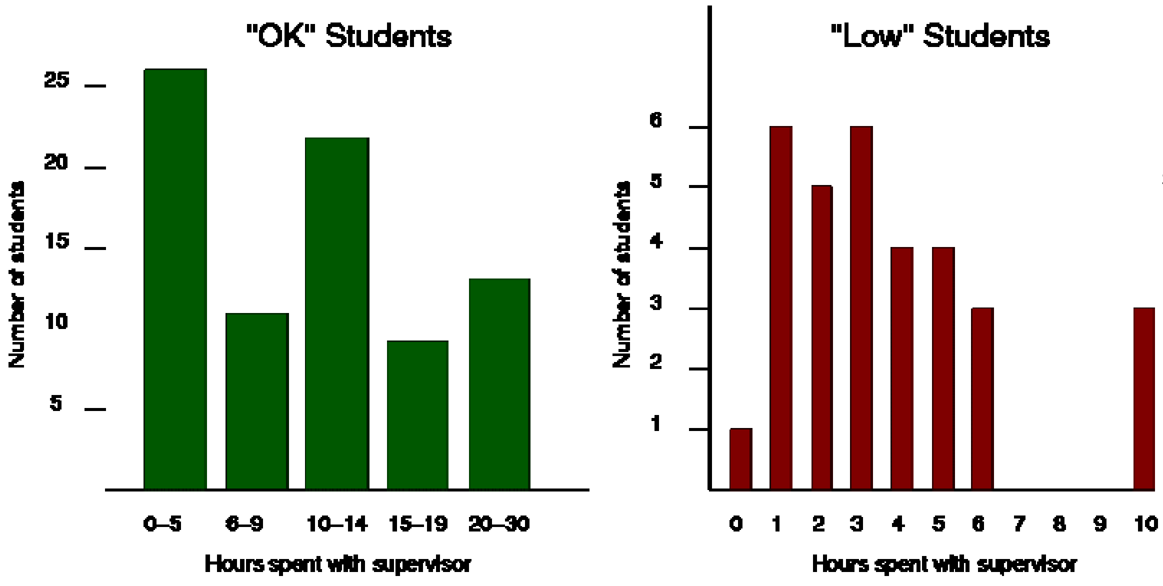

Figure 8: Time spent with supervisor for satisfied and dissatisfied student groups

Of the 132 students who responded to the questionnaire, 28 were female (21%), however of the 15 students who spent 20 hours or more with their supervisor, 7 were female (50%). Whilst this is a small sample, it does suggest that there is a gender difference.

Only 17 students spent as much time engaged in email (or similar) communication as face to face. Of these, 8 are "Low" students, suggesting that face to face communication may be important in supporting the supervisor-student relationship.

4.4.2 Content of supervision

The role of the supervisor inevitably changes depending on the individuals involved and the type and focus of the project. Fincher *et al.* identify four principal roles:

- observer/commentator;
- line manager;
- project manager;
- master/mentor [12].

Of these, the first three relate mainly to administration of the project, and the final one to technical issues. Our hypothesis was that most of the supervision process relates to administrative and management issues. The motivation for this is the knowledge that some of our colleagues supervise students who have chosen topics where the supervisor is not able to offer detailed technical advice, and we wished to discover how widespread this phenomenon is. It was felt unreasonable to ask students to distinguish in detail between the four roles, and that simply to ask them to quantify the amount of time on the administration and on the technical discussion was appropriate.

Only 12 students (n=128) considered that more than half of their supervision time related to technical issues, and 29 saw the split between administration and technical issues as 50/50, so most of the students (68%) spent more time addressing general issues. Technical issues were not discussed at all by 23 students, of whom 11 were “Low” students.

This clearly indicates that supervision is generally regarded as an activity which supports the process of managing the project, rather than as an opportunity to get detailed technical advice.

4.5 Motivation

The third year project exists partly because it must (for reasons of accreditation), and partly because it is educationally right and proper that students take part in such an exercise. We offer students a large amount of flexibility in selecting their project (and their supervisor), and our hope is that the project is seen as exciting and stimulating. For the majority of students we believe this to be true, but we wished to explore students’ motivation in more detail, and this formed part of our interviews.

General issues relating to student motivation are well-documented, [12 pp. 96–99, 28], and work has been done on the motivation of students engaging with the activity of programming [29]. Material which specifically addresses motivation in the context of a large individual computing project is more scarce, although a survey by Capon suggests that, at that time, the principal motivating factor is “a personal skill that would be useful in the future, either technical or managerial” [30].

Henry [7] suggests that students who pursue “unstructured” projects are more positive about their work than those undertaking more structured exercises, and reports the results of a survey of projects undertaken at the Open University in which 90% of respondents undertaking a large unstructured project found it to be “very interesting”. Henry also notes that “the degree of perceived value relates very closely to the amount of freedom given the student and size of the enterprise — more freedom breeds greater satisfaction”. Although Henry’s research did not involve computing students, we might expect that our unstructured approach to computer science projects would encourage students to be motivated by intrinsic interest in their projects.

We initially classified the twelve responses as either intrinsic (the student was motivated mainly by interest in the subject), or extrinsic (the student was driven by some external factor, such as the simple desire to achieve a high mark).

Eight of the twelve students clearly expressed a view that they were intrinsically motivated, with comments such as “exploring new areas, and seeing how far I could possibly go” and “I really liked working on something new; when I saw myself doing something and achieving something, this was the best motivation”. One student also identified the difficulty of the task as a major factor, and that real-world utility was important for him: “I worked for [name of company] for a while; things out there are extremely expensive and complicated”. Interestingly, one of these students also identified a secondary external factor: “And to live up to expectations – my brother had done a dissertation, and looking at his, his was very impressive”.

Of the four students for whom the motivation was extrinsic, one (who is an overseas student) explained that he had been working for a company who had instructed him as to his project content, and that he would be continuing the work for that company when he returned. The other three indicated that they were not enjoying the experience since their choice of project title was poor. As one student remarked, “I didn’t really choose a project I was interested in, which was a bit stupid really.”

In all four instances, the project was perceived as a job to be done, a more structured task with a defined end result.

The end of course feedback forms (n=107) reinforce this, with 19 students explicitly remarking that the freedom to choose their own project was a positive aspect of the project.

Only one student identified a positive extrinsic factor, namely “Good practice for commercial project development.”

Although the interviews represent only a very small sample of students, our results support our hypothesis that the freedom and flexibility which our projects allow most of our students to be intrinsically motivated. Furthermore, we have evidence, both from the interviews and from the course feedback forms, that the choice of project title is critical in motivating students.

5. CONCLUSIONS AND FURTHER WORK

The purpose of this research has been to gain a deeper understanding of how individual undergraduate computer science projects are undertaken. We began with several questions, some pedagogic, some very practical, and we have collected and analysed data which provides evidence to help answer most of those questions.

5.1 Summary of Results

These results raise issues about how computing projects should be delivered, not all of which make comfortable reading. Our guidance about workload and about supervision time seems appropriate, but there are other practical and pedagogic concerns.

We initially established that the projects at our university are typical of those offered by comparable universities in the UK, and we have taken evidence from students (by questionnaires, interviews, and other sources) to answer several research questions. Our analysis suggests the following:

1. Most students spend approximately the 300 hours expected in order to complete the project — a few exceed this substantially, but are aware that they are putting in more effort than their peers. The time students spend on their projects is not evenly distributed throughout the year, with on average less that 40% done in the half year preceding Christmas.
2. Most students used a methodology to a greater or lesser extent, with the majority of them opting for a “waterfall methodology”, with very few using a named methodology. A significant proportion are claiming to use an agile methodology.
3. A substantial number of students are failing to follow “good practice” (following the systems life cycle) whilst developing their projects, in spite of writing their dissertations to make it appear to the contrary.
4. Students who spend on average half an hour a week discussing their project with their supervisor consider this sufficient, and this suggests that this is an appropriate guideline for project management. The majority of supervision time is focused on project management issues rather than technical discussion, and project supervision is seen as a very important component of the project process.
5. Students who have chosen their project themselves tend to be highly intrinsically motivated, whereas students whose title has been chosen for them (for whatever reason) appear more extrinsically motivated. The freedom of choice which we allow for students to decide their own topic is seen as a strong motivating factor.

5.2 Limitations and Further Work

The research presented here suffers from limitations, including the restriction of the study to one cohort of students at a single institution. A future study might compare data from several institutions to establish whether factors such as the profile of the student cohort (ethnicity, age, gender, and so on) affect the results. This research has focused on individual software development projects, and further work is indicated to identify those factors which transfer to the settings either of a group project or of an individual project of a different genre.

Although the questionnaire was completed by half of the cohort, there is evidence from studies elsewhere that there is a correlation between attendance at teaching sessions (including lectures and seminars) and subsequent attainment [32, 32]. In other words, the sample who completed the questionnaire may represent the stronger students in the cohort. This might have been addressed if a register had been taken, however the action of taking a register might itself have compromised the perceived anonymity of the exercise.

The content of a project in a rapidly changing discipline such as computer science will vary depending on when the project was delivered. The observation of a project course over a period of time would allow changing trends in the delivery of projects to be identified.

There are three types of stakeholder in undergraduate projects — the students, the academic staff, and the institutional process. In this study, we have concentrated on the “student perspective”, A complementary study might concentrate on the “staff perspective”, to identify the differing pedagogic and pragmatic approaches taken in the supervision of project students within a single institution. We have also avoided the added complexity of projects with external stakeholders, such as commercial sponsors.

The project requirements of other computing departments has of necessity only been analysed at a sufficient depth to place the results presented here in context. The detailed requirements for computing projects at other institutions will inevitably differ, and a study of the pedagogies and processes used at a variety of institutions would serve to highlight the “institution perspective”.

A further study might consider the details of the individual reports to ascertain whether (and how) individual students perceived the content of their submissions differently to the academics marking them.

6. REFERENCES

[1] BCS, *Guidelines on Course Accreditation* (2007), http://www.bcs.org/upload/pdf/heaguidelines.pdf. [Accessed 21 November 2008].

[2] ACM/IEEE-CS, *Computing Curricula 2001* (2001), www.acm.org/sigcse/cc2001/ [Accessed 21 November 2008].

[3] Clear T., Goldweber M., Young F.H., Leidig P.M., and Scott K., Resources for Instructors of Capstone Courses in Computing, *Working Group report, Annual Joint Conference Integrating Technology into Computer Science Education*, Canterbury, UK, 93-113 (2001).

[4] Sommerville, I., *Software Engineering (8th edition,* Addison-Wesley (2006*)*.

[5] Umphress D.A., Hendrix T.D., and Cross J.H., Software process in the classroom: the Capstone project experience, *IEEE Software* **19**(5), 78-81 (2002).

[6] Clear T., Documentation and agile methods: striking a balance, *ACM SIGCSE Bulletin* 35(2), 12-13 (2003).

[7] Henry J., *Teaching through Projects*, Kogan Page (1994).

[8] Clark M.A.C., and Boyle R.D, A Personal Theory of Teaching Computing Through Final Year Projects, *Computer Science Education* **9**(3), 200-214 (1999).

[9] Engineering Council, *UK-SPEC*, ECUK (2004).

[10] QAA, *Academic Standards – Computing*, Quality Assurance Agency for Higher Education (2007). http://www.qaa.ac.uk/academicinfrastructure/benchmark/statements/Computing07.pdf, [Accessed 21 November 2008].

[11] Jaques D., *Learning in Groups: A Handbook for Improving Group Working*, Kogan Page (2000).

[12] Fincher S., Petre M. and Clark M., (Eds), *Computer Science Project Work: Principles and Pragmatics*, Springer (2001).

[13] Dawson C.W., Projects in Computing and Information Systems: A Student’s Guide, Addison-Wesley (2005).

[14] Hughes B. and Cotterell M., *Software Project Management (4th edition),* McGraw-Hill (2005).

[15] Humphrey W.S., *PSP: A Self-Improvement Process for Software Engineers*, Addison-Wesley (2005).

[16] Martin R.C., *Agile Software Development,* Pearson (2003).

[17] Stephens M., and Rosenberg D., *Extreme Programming Refactored: The Case Against XP*, Apress (2003).

[18] Torvalds L., Re: I request inclusion of SAS Transport Layer and AIC-94xx into the kernel, in *Linux Kernel Mailing List* (2005). http://www.lkml.org/lkml/2005/9/29/233, [Accessed 21 November 2008].

[19] Coleman D., Arnold P., Bodoff S., Dollin C., Gilchrist H., Hayes F. and Jeremaes P., *Object-Oriented Development: The Fusion Method*. Prentice-Hall (1993).

[20] Kruchten, P., *The Rational Process: An Introduction (3rd edition),* AddisonWesley (2004).

[21] Kaner C., Falk J. and Nguyen H.Q., *Testing Computer Software (2nd Edition),* Wiley (1999).

[22] Oppenheim A.N., *Questionnaire Design, Interviewing and Attitude Measurement,* Continuum (1992).

[23] Cohen L., Mannion L. and Morrison K., *Research Methods in Education (6th edition),* Routledge (2007).

[24] Chow A. and Joy M., Shifting the Focus from Methodologies to Techniques, *Proceedings of the 6th Annual HEA-ICS Conference,* York, 25–29, University of Ulster (2005).

[25] Empirical Modelling Group Empirical Modelling (2008), http://www.empiricalmodelling.com [Accessed 121 November 2008].

[26] Ghezzi C., Jazayeri M. and Mandrioli D., *Fundamentals of Software Engineering,* Pearson Education (2003).

[27] Booch G., Rumbaugh J. and Jacobson I., *Modeling Language User Guide (2nd edition),* Addison-Wesley (2005).

[28] Armstrong S. S. and Thompson G. (Eds.), Motivating Students, Routledge (1998).

[29] Jenkins T., *The Motivation of Students of Programming*. MSc thesis, University of Kent (2001). www.cs.kent.ac.uk/pubs/2001/1401/ [Accessed 21 November 2008].

[30] Capon P., Maximizing Learning Outcomes of Computer Science Projects, *Computer Science Education* **9**(3), 184-199 (1999).

[31] Colby J., Attendance and Attainment, *Proceedings of the 5th Annual LTSN-ICS Conference*, University of Ulster (2004).

[32] Burd E. and Hodgson B., Attendance and Attainment Revisited, *Proceedings of the 6th Annual HEA-ICS Conference,* York, University of Ulster (2005).