GPT's Preparation of Students for Programming in the Real World

R.J.Dawson B.Sc., M.Phil., C.Eng., MBCS
Lecturer, Dept of Computer Studies, Loughborough University
(Formerly Software Lecturer, Plessey Networks and Office Systems)

R.W.Newsham B.Sc., MBCS
Software Lecturer, GEC-Plessey Telecommunications Ltd., Beeston, Nottingham

R.S.Kerridge B.Eng., B.A.
Software Training Manager, GEC-Plessey Telecommunications Ltd., Beeston, Nottingham

Summary

For some years GEC-Plessey Telecommunications has run courses for new software engineering graduates to introduce them to the real life problems of programming in the real world. For two weeks the graduates work in groups on software project with each group in competition with the others. The course leader undertakes role play to give the groups experience of the customer, manager, technical consultant and quality auditor personnel. A few "dirty tricks" help to simulate the real world environment. A "wash-up" session at the end reviews the lessons learned. The course is found to have advantages over corresponding undergraduate courses as the restricted hours and full supervision tend to highlight the problems that occur and ensure the lessons of the course are more effectively learned. It is concluded that benefits of realism and awareness gained by the graduates will mean the GPT course will continue to be a worthwhile course valued by company managers, and that other companies may find it useful to incorporate such a course into their own training programmes.

Introduction - The Need For The GPT Course

The GEC-Plessey Telecommunications company at Nottingham (formerly Plessey Networks and Office Systems) has for the past eight years been running two week, full time project courses for new software engineering graduate entrants to the company. These courses were devised at the request of managers to 'knock the new graduates into shape' by giving them an experience of industrial working methods and practices. The courses are based on practical work with each graduate spending 90% of the time working on a programming project. This project work is undertaken in groups as an important part of the course requirement is to give experience of working effectively within a team.

It has been found by the software managers that although graduates of computer science (Ref 1), and even some graduates of electronics (Ref 2) and other subjects (Ref 3,4), had been given some education of software engineering principles, they still seemed to have difficulty in putting the theory into practice in the 'real' work environment. In later years there has been a marked improvement in graduates in this respect as more and more degree courses have introduced group projects, some working in the manner of a small software house sometimes referred to as the 'Software Hut' (Ref 5,6,7), some from as early as the first year (Ref 8,9). Nevertheless, GPT have still found that graduates will learn a lot from the GPT course even though some undergraduates will have already experienced group project work at their university or polytechnic. This paper examines the GPT course and suggests why it is more successful than undergraduate courses in giving an experience of software engineering principles and practice in the 'real' world.

The Course Format

Each of the GPT software engineering training courses is two weeks in length and is given to graduates working in software departments within the first couple of months of joining the company. In recent years a need has also been found for hardware engineers to have an appreciation of the principles of software engineering, and so many hardware department managers have sent their graduates on the software engineering course.

During the course the participants are divided into groups with each team working on the same project in competition with each other. The size of the groups has varied between a minimum of 3 people (Ref 10) and a maximum of 5. Whenever possible a group size of 4 has been used, this size being found to be the optimum to ensure that all members 'pulled their weight' without any one member becoming too dominant. Two or three such groups make up the total numbers for a single course.

The project work involves programming in a high level language. Pascal, Coral and C have all been used, depending on the abilities of the participants. Often the course follows an earlier GPT course in a specific language so the graduates are familiar with both the language and the equipment provided at the GPT training centre. The training centre staff also get to know the graduates before the course which enables the teams to be picked with each group having a range of abilities and with every group having an equal chance of producing the best project.

Choice of Project

A number of different projects have been used for this course. In each case the project is chosen such that it would be possible to produce some usable software in the time available. However, there is always scope for any team to attempt to take on more than they can handle, and there is little margin for inefficiency and error in the development schedule.

Where possible a 'real' project is used (Refs 3,11), such as development of an administration system for the GPT Training Centre's own needs. There is no doubt that the thought that the best project would be actually used gives a little bit of extra motivation. Motivation has never been a problem, however, each participant has to be present for the contractual company hours, and graduates are normally very keen to get on and do something positive within their new company environment. Successful courses have been run using simple computer games as the target product. Indeed, any project seems to be successful providing the participants can find some interest in the product they are developing.

Project Supervision

Supervision of the course is full time with at least one member of staff overseeing the two or three project groups involved. During the course the course leader will take on specific roles relating to the project. At any one time the course leader can play the part of:

- The lecturer/technical consultant - giving technical advice regarding the computer hardware and software, and any software engineering problems.

- The customer - who may have some overall technical knowledge but cannot follow in detail how any proposed system will work. Like all 'real' customers, he can be vague about his requirements, he can change his mind, and can say things he didn't really mean!

- The manager - who may ask for project plans, progress reports, and technical details from time to time. The manager can apply additional requirements or constraints on the project teams working methods. He may also apply pressure when the project gets behind schedule.

- The quality auditor - who reenforces the manager role with quality checks and inspections of documentation and code throughout the project.

These roles are for the most part kept quite distinct. The course leader will make it clear which role he is taking at any one time and will refuse to answer questions unsuitable for his current role with

comments such as "Only the customer would know that". Advice is also limited to the current role, so that, for example, as 'the customer' the course leader will happily agree to product suggestions even though he knows as 'technical consultant' the suggestions would be difficult, or impossible to put into practice.

If another member of staff is available, both will take the role of customer, acting as different directors of a client company. In these circumstances the customers are deliberately given different personalities, and different ideas about what they expect from the product. For example, one customer may be enthusiastic jumping at every suggestion a team may make no matter how impractical, the other may be very conservative, putting a dampener on any suggestion that differs from his original ideas, regardless of how good the suggestion is. Although many students have been told customers may differ in this way, few are prepared for the difficulties and contradictions encountered. For most graduates, it is the first time they have met this type of problem.

Course Introduction - The First Day

The course starts with a half day of lectures giving an introduction to the industry working methods and practices and explaining some of the problems of working in groups. Although the amount the graduates take in at this stage is limited, and many graduates of computer science will have covered the same theory in their degree course, it is useful to refer back to these lectures as problems arise and at the end of the course. Indeed, it is possible to make quite accurate predictions of where problems will arise in the project knowing full well that most of the problems will still occur despite the warnings. This element of "I told you so" helps to drive home the message as the remainder of the course unfolds.

By lunch time of the first day the course participants are given the problem they are to undertake. The specification is deliberately vague, incomplete, and in places ambiguous. Working as individuals in the first instance, each participant is asked to write down what he or she knows from the specification and what are the areas that need clarification. The 'customer' will not answer questions at this stage. The course participants are then assigned to their groups where the first task is to pool their ideas and prepare a list of questions for the customer.

During the afternoon of the first day each group has the opportunity to interview the 'customer', or 'customers' if there is more than one member of staff participating. During the interview the customers are rather naive about their own requirements, but a considerable amount of additional information is given depending on the skill of the interviewing groups in choosing the right questions. Some of the extra information may actually contradict the initial specification, it is up to the groups to determine what the customers really want.

Following the interview the 'manager' will ask for an initial project schedule for the remainder of the time available. The groups are given until the end of the first day to produce their schedule. Without exception every group has so far produced a plan to develop their product in a single step referred to as the 'waterfall' approach by Boehm (Ref 12). Although warnings of the dangers of this method are given during the morning's session, no attempt is made to suggest any alternative once the actual problem is revealed. It would appear that while some graduates have learned of alternatives on their undergraduate course, none have so far had any ability to put these ideas into practice. The groups are allowed to follow the 'waterfall' route in order that they experience the usual problems, and can appreciate the lessons learned when discussed at the wind up session on the last afternoon of the course.

Project Documentation Activities

As the project progresses the groups are asked to produce several documents. The documents required and the format of the documents has varied over the years as the course has been developed. In general, a customer requirements specification, a system design specification, a detailed design specification and a project development plan are usually required in the first week of the project, and a user manual, including any installation instructions, and maintenance manual are required by the project completion time. The scope and detail of each document depends on the project undertaken and the discretion of the course leader acting as 'manager'.

Attempts are made to ensure that the groups do not get bogged down in the documentation. The customer requirements may be required in the form of a short presentation to the customer and the rest of the class. Prompt sheets are usually supplied for the system and detailed design specifications and it is stressed that hand written documents with as many crossings out as required are perfectly acceptable providing they can still be read. Sometimes a group may, despite discouragement, make persistent demands for word processing facilities. In this case, to make a point, the course leader may provide what they desire. The result is always the same, the group spends too much time making their documents look pretty while their competitors are getting on with the project.

To save time the groups may be asked to present some documents verbally to the 'manager' who will take notes during the presentation. The notes are sufficient for future reference when, for example, cross checking the code against design. Questions by the 'manager' can make the verbal presentation interactive - a useful way of showing the group concerned how inadequately thought out their designs and plans may be.

Quality Audits

Quality checks are made whenever the course leader is shown any document or computer listing. Regardless of which hat he may be wearing at the time the course leader will refuse to examine any document or code unless it is properly identified with name, version number and time of issue. As well as stressing the need for proper release control, this frequently saves the course leaders time when, for example, he is asked as a technical consultant to help debug code from out of date listings!

At any time a group may be asked to produce documents and code listings for inspection by the 'quality auditor'. General checks are made for structure and style and also that the code corresponds to the design. It is not difficult to catch a group out by finding differences between the code and the design, usually as a result of inadequate initial designs leading to participants performing retrospective design with much 'hacking' at the terminal.

The Course Conclusion - The Last Afternoon

The deadline for completing the project work is the lunchtime of the final day. During the last afternoon the groups each give a demonstration of their product and a short presentation where each group describes not only what they have done, but also how well they have performed as a team. As well as the 'customer' in the person of the course leader, other members of the training centre staff and often the participants own managers will attend the final demonstrations and presentations. All those attending, including the competitor groups, may ask questions and pass comments on each team's performance. The knowledge that their managers may attend the final presentations gives a useful added incentive for all participants to produce a respectable performance during the course.

Following the presentations the course leader holds a 'wash up' session reviewing the progress made and emphasising the lessons to be learned. It is usually possible to make useful comparisons at this stage, each group having encountered different problems through their projects. It is often possible to show the penalty paid by a group for a mistake by drawing on the experience of a competitor group who has not done things in quite the same way. The groups are encouraged to make their own comments and observations in this session, as a point is more forcibly made if it comes from the course participants rather than a member of the training centre staff.

Project Progress

The normal progress pattern for the project courses is for all to appear to go well in the first week. Half way through the week the groups will have what they consider to be a well thought out specification, and a realistic plan to produce the goods in the time available, usually they even allow a little leeway for things to go wrong. Neither the specification nor the plan will be flawless with an experienced course leader having little difficulty in finding a few holes in their proposals. Nevertheless after a bit of patching the participants are able to get under way, and indeed, during the remainder of the first week, and perhaps the first day of the second, the groups usually manage to keep within, if not ahead of their schedule.

The first signs of trouble usually come in the early part of the second week when the 'quality auditor' points out differences between code and design. The initial design is usually inadequate or flawed and corrections have invariably been made at the terminal, directly to the code. The dangers of this action are often not fully appreciated at this stage as graduates appear to regard this method of working as normal even if they admit it is undesirable.

By about the second day of the second week the groups begin to see the first problems appear as they attempt significant integrations of different members' work. By half way through the second week there are signs of concern, if not alarm, that the deadline of the end of the week may not be met. The remainder of the week becomes a mad scramble to get something ready in time. The pressures of the second week often begin to show on the course participants who have been known to fall out with each other as recriminations fly over why a delay has occurred. The demonstrations at the end of the week are more often than not accompanied by a series of excuses as the products fail to live up to the specifications and are found to be full of errors.

The demands of the course leader are varied and follow no set pattern. Advantage can be taken of the quieter moments to observe the working methods and progress of the group. A few strong hints can prevent or reduce the more undesirable working practices such as a whole group sitting round a single terminal taking turns to press the keys, and a quiet word here and there may reduce the pressures and bring a group back onto speaking terms! There are, however, occasions when the course leader is redundant, especially if there are only two groups participating. These periods are not long and tend to be prone to interruption so it useful to have one or two small administration tasks that can be used to fill in the odd spare moment.

During the second week significant disasters seem to occur to hold up progress, an unknown fault will be found in the compiler, a team member will be off sick for a day, or the computer response will fall to an unacceptable level. Despite warnings, few participants make adequate backups of their work and configuration control is often haphazard. It is often just assumed that backups will automatically be made of the system, though few will ask whether, and how often this is done. The 'manager' or 'quality auditor' usually has to draw attention to the problem in order that the consequence of a system crash or even just an accidental deletion on their own part will not be so serious as to destroy the project.

To test the configuration control and to make a point, occasionally, if progress seems to be going too well, the computer may have to develop a fault with a little help from the course leader, but in fact, more often than not this type of disaster occurs quite naturally! Another of the dirty tricks that can be made by the course leader is to take a copy of one or more groups work over a lunch time, and then, half a day later, to replace their current systems by the out of date copy. The time taken to recover from such a mishap can be very revealing of each groups organisation and control, and can be the subject of much discussion in the course wash up session.

Assessment

There is no formal assessment of the individuals or groups on the GPT project course. Only if a course participant performs either exceptionally well or exceptionally badly will a manager be notified of an individual's performance. As managers continue to be keen to send their newly recruited graduates (and sometimes other employees) on the course it can be deduced that they are pleased with the knowledge and experience the course gives. Only on a couple of occasions has a participant performed badly and refused to learn from the course. The two individuals concerned have both since left the company having found their outlook incompatible with the company's environment.

The attitude of the participants are surveyed both informally at the final wash up session, and more formally by question sheets at the end of the course. These attitudes have on the whole given a very positive assessment of the course with participants feeling they have gained a useful insight into software development in the 'real world'. The course is always subject to further development, however, so constructive criticisms are welcomed and have often lead to minor changes in the way the course is run.

Lessons Learned

The lessons learned on the course are many:

- For those that have never worked within a group before the experience of sharing and coordinating the work load is invaluable. This is perhaps the most important lesson of all for those who have never worked in a team before.

- The necessity of getting the design right from beginning if the 'waterfall' development model is followed is emphasised, along with the difficulties encountered if the design is changed in the coding phase. Most groups realise that the time they spent on design was inadequate when seen in retrospect. The course leader will also point out where advantage could have been gained by alternative development strategies.

- The difficulty of getting the requirements and design correct from the beginning when dealing with 'real' customers is also illustrated. To simulate 'real' customers the training centre staff deliberately start with differing and vague ideas that clarify and change as the course progresses. The customers ideas do not, in general, evolve in the same direction as the developments by the groups.

- The problems of integration are emphasised, underlining the importance of defining interfaces, unit testing, and of planning and allowing enough time for the integration process itself.

- The participants become more realistic in their expectations when they learn that disasters are not exceptional. Often they will complain about the inadequacies of the computer and its software, or that too much was expected from them in the time available. In each case it is pointed out that they were responsible for negotiating their own targets with the customer with a full knowledge of the development equipment before the project was started. They learn that they cannot expect to live in an ideal world.

In each case some or all of these lessons could be said to have been gained from the participants' university or polytechnic courses. The difference is in how well the lessons have been learned. The graduates will often know the theory of the points made, but they do not fully appreciate the implications when it comes to their own work. There is no doubt that software managers appreciate their new graduate entrants being brought 'down to earth'! This extra realism and awareness allows the graduates to slot into an existing team more easily and increases their effectiveness by reducing mistakes made through inexperience.

The Advantages of the GPT Course Over Similar Undergraduate Courses.

Many universities and polytechnics have course modules in software engineering and the principles are well covered in standard text books (Ref 13,14). Of these courses, many contain components that appear similar to the GPT course. Although usually spread over a longer period, in parallel to their other studies, most computer science students and those on some other courses undertake group software development projects as part of a study of software engineering. The project sizes are often similar, with similar sized teams, and with similar documentation requirements, quality audits or role play of the supervisor (Refs.15-19). Why is it, therefore, that the GPT course seems more successful in getting the points across? Examination of the course in detail reveals a number of differences:

- The course participants are kept strictly to company hours while on the GPT course. No extra work is allowed even over the lunch period. Students, on the other hand, are well used to doing their work at the last minute. Many have been experiencing a last minute rush to hand in work since their early school days. It is not unusual to find students working through the night to meet a coursework hand in date. It comes as no great surprise, therefore, to find themselves in a last minute rush to finish an undergraduate project though the problem may be more severe than usual. The extra hours required at the end of a project are subconsciously planned from the start. This is not the case on the GPT course where the graduates are aware of exactly how much time they have to complete a project. Consequently they are very surprised and quite alarmed to find out how far their planning can go wrong. This even applies to those that have worked in group projects in their university courses

beforehand, though they are more aware and better prepared than their colleagues. The strict company hours of the GPT courses play a major part in 'bringing home' the lessons of scheduling and timing.

- The role play by the course leader and other members of the GPT training centre staff is significant. Although some role play is enacted by some undergraduate project supervisors the roles are not usually as formally defined and the contact hours are often limited. This role play, particularly when playing the deliberately vague nature of a customer is a valuable insight for the students who are used to supervisors that at least pretend they know what they are talking about. Very few graduates had met more than one customer presenting different ideas and opinions in their undergraduate projects.

- The close contact and observation of the groups by the course leader allows him to emphasise points in the 'wash up' session at the end of the course by relating to events experienced by the groups during the course. Furthermore, as the groups have all been working side by side for the same set hours, each group has been able to observe the experiences of others, and, as they are all working to the same project specification, each group can see where they have been outperformed by other groups. The course leader can draw attention to these differences during the wash up session to stress the conclusions that are made.

- As undergraduate courses are normally assessed as part of the degree, university and polytechnic project supervisors usually to try and be totally fair to the participating students. This tends to reduce the inclination to try any 'dirty tricks' such as deliberately disrupting the project work. This unfortunately leaves the students with too much of an expectation of an ideal environment. The dirty tricks, although seeming a little mean at times, educate the course participants on how to cope with the inevitable occurrences of the real life environment they will later experience.

Collectively, these points mean the GPT course gives a better insight into the working methods and practices of the industrial or commercial environment. The the set company hours the role play by the course leader and even the dirty tricks help to produce the right atmosphere to facilitate typical industrial working practices, and at the same time, an experience of the 'real world' is given in relatively informal way. There is no doubt that the GPT managers appreciate their new graduates having an idea of what to expect when they start work.

Conclusion

The GPT software engineering group project course provides a valuable education for new graduates entering the company. The extra resources of staff time and equipment that the company can supply help to give an insight into the industrial working practices and experiences of the 'real world'. While there is an obvious expense for such a course the enthusiasm of GPT software managers to continue sending their new graduates is a strong indication that the benefits outweigh the costs. Many universities and polytechnics have group projects within the software engineering components of their courses and these have improved the standard of awareness of the software engineering principles over recent years. However, the company environment, and the extra resources of equipment and available lecturer time, all help to effectively simulate 'real life' working conditions on the GPT course and to bring home the lessons learned to an extent that is not possible within a higher education establishment.

There is every indication that company managers will continue to want to send their new graduates on the GPT course in the future. It must be concluded, therefore, that other companies should find the GPT course a useful model to include in their own training programmes for software engineering graduate recruits.

Bibliography

1. LEVENTHAL,LM & MYNATT,BT: 'Components of typical undergraduate software engineering courses: Results from a survey', IEEE Trans.Softw.Eng., 1987, SE-13,(11), pp.1193-1198

2. ALDER,C: 'Software engineering in an Electronic Engineering degree', Softw.Eng.Journal, 1989,4 (4),pp.191-199
3. BUSENBURG,SN & TAM, WC: 'An academic program providing realistic training in software engineering',Comm. ACM, 1979, 22(6), pp.341-345
4. GURNEY,W, MAAS,P & MAY,G: 'An experiment in software engineering education', Softw.Eng.Journal, 1987, 2 (4), pp.127-132
5. HORNING,JJ and WORTMAN,DB: 'Software Hut: a computer program engineering project in the form of a game',IEEE Trans.Softw.Eng.,1977,SE-3, (4), pp.325-330
6. WASSERMAN,AI and FREEMAN,P (Eds.):'Software Engineering Education: Needs and Objectives' (Springer-Verlag, 1976)
7. MacCALLUM STEWART,L: 'The Software Hut',Computer Bulletin,1985, 1 (1),pp.8-9
8. MACE,BM: 'Software engineering within the context of a Computer Science degree programme', Softw.Eng.Journal,1987, 2 (4), pp.200-202
9. ROPER,BM: 'Training first year undergraduates to produce quality software', University Computing, 1988, 10, pp.9-12
10. WORTMAN,DB: 'Software projects in an academic environment', IEEE Trans.Softw.Eng., 1987,SE-13,(11),pp.1176-1181
11. BURNS,JE and ROBERTSON,EL, 'Two complimentary course sequences on the design and implementation of software products', IEEE Trans.Softw.Eng., 1987,SE-13,(11),pp.1170-1175
12. BOEHM,B: Software Engineering Economics' (Prentice Hall,1981)
13. SUMMERVILLE,I: 'Software engineering, 3rd. Ed.' (Addison Wesley, 1988)
14. PRESSMAN,RS: 'Software engineering - a practitioner's approach' (McGraw-Hill, 1987)
15. WINDER,R, COLE,R and EASTEAL,C: 'Software engineering in a first degree', Softw.Eng.Journal,1987,2 (4),pp.133-139
16. KING,PJB: 'Experiences with group projects in software engineering', Softw.Eng.Journal, 1987,2 (4),pp.133-139
17. JONES,A and BIRTLE,M: 'An individual assessment technique for group projects in software engineering' Softw.Eng.Journal, 1987,2 (4),pp.133-139
18. McKEEMAN,WM: 'Experience with a software engineering project course', IEEE Trans.Softw.Eng., 1987, SE-13,(11), pp.1182-1192
19. WEISS,DM: 'Teaching a software design methodology', IEEE Trans.Softw.Eng., 1987, SE-13,(11), pp.1156-1163