Introducing Software Engineers to the Real World

Ray Dawson, Loughborough University
Ron Newsham, University of Derby

Most software engineering graduates begin their careers lacking an appreciation of real-world conditions. Do universities have the resources to simulate this environment or must software companies provide such training themselves?

Although they may know much about software development theory, computer science graduates are often poorly prepared for the real-world practice of their craft.[1] In the case of GPT, management found that although graduates of computer science,[2] and even some other subjects,[3] had been taught software engineering principles, they still had difficulty handling a professional work environment. Specifically, they had trouble dealing with; frequent changes to specifications, priorities, equipment, and procedures; the expectations of customers, managers, and fellow team members to "do the impossible yesterday"; conflict—whether requirements conflict, disagreement among customers, or simply the differences between real-world practices and the theory they had been taught; and estimating and planning in a world of meetings, multiple tasks, constant distractions—and above all, the everyday disasters that inevitably occur with real people, equipment, and software. As one software manager said, graduates can be "very knowledgeable—but not a lot of use."[4]

TABLE 1
FEATURES OF THE COMPANY AND UNIVERSITY COURSES

Feature	GPT Company	Loughborough and Derby Universities
Students	12 maximum	40 to 100+
Duration	2 weeks full time, fixed 74 hours	About 80 hours over 10 weeks
Hours	Strictly limited to company hours	Unlimited
Introduction	Half day at start	Previous modules give background information
System experience	Familiar with both hardware and software	Often development hardware or software must be learned
Supervision	Full time, sometimes with two supervisors	One supervisor about one third of the time
Simulation	All real-world aspects simulated	Loughborough: real-world aspects in one module Derby: different real-world aspects in different modules
Team size	3 to 5	4 rising to 8 or 9 in recent years
Specification	Incomplete and ambiguous	Incomplete and ambiguous
Role play	Extensive	Limited by supervisor availability (some real customers used)
Forced changes	Specifications, working procedures, customer personalities, supporting software	Specifications only
Dirty tricks	Many	Few, other than specification changes
Assessment	No formal assessment	By tutor observation, deliverables, and student self-assessment
Review	Related to detailed observation of the graduates' experiences	Based only partly on observed student experiences

To address these deficiencies, GPT ran—over 14 years—project courses for new employees.[1] We were asked to set up and run a course based on practical work that simulated software development in the real world in every possible respect. Since then, we have both transferred to university teaching: Ray Dawson to Loughborough University in 1987 and Ron Newsham to the University of Derby in 1996. Our academic positions have given us experience with existing practices for teaching real-world awareness in computer science courses and the opportunity to put ideas from the company training course into practice in a university environment.

Many of GPT's ideas for simulating real projects can already be found in university courses. The universities of Loughborough and Derby are typical in this respect. Most university computer science courses involve students working on group projects in the manner of a small software house, sometimes referred to as the "software hut."[5,6] Nevertheless, GPT found that graduates still had much to learn from the company course.

THE GPT COURSE

Soon after joining GPT, graduates would attend the two-week training course we created, working in three- to five-member teams[7] that competed with one or two other teams. Table 1 shows the course's main features, as well as those at our current universities.

The course started with a half-day introductory session in which the students were told the aims of the course and warned where they would likely go wrong. They would then undertake a project chosen so it would be possible to produce some usable software in the time available, but with little margin for inefficiency and error. There was always scope for a team to take on more than it could handle. The course had at least one full-time staff member supervising and monitoring the participants, with another staff member assisting some of the time. The last afternoon was devoted to demonstrations and presentations by the teams and a review by the course leader.

After the introductory session, the participants were given the specification. This was not a true specification, but a brief, vague description that was in places

misleading and ambiguous. The specification always contained requirements that could not be met or that conflicted with each other. The first task of each group was to sort out the project's requirements and priorities, but these would then be deliberately and frequently changed by the course leader as the project progressed. The course leader also made changes to the working environment, in the directed working procedures, for example, or the hardware and software support tools.[8]

The staff members used role play to enact the part of customers, managers, technical advisors, and quality auditors. Course leaders kept each role distinct so that, for example, the "customer" would happily agree to product suggestions even though the "technical consultant," if asked, would have advised that they would be impossible to put into practice.

The availability of a second staff member allowed the simultaneous assumption of two different customer roles, to which each would be given a different personality and expectation set. No student was ever prepared for one customer to be enthusiastic, jumping at every suggestion no matter how impractical, while the other acted very conservatively and accepted any suggestion that differed from their original ideas reluctantly if at all.

Finally, if this were not enough to disrupt the participants' performance, we played deliberate tricks on the course participants. The central computer in the network could be made to crash unexpectedly, or we found urgent errands for key team members. One overnight trick we played involved replacing all the computer file space with a copy made half a day earlier. While this may seem particularly mean, it helped us determine the ability of each group to find and recover from the problem and provided them with a useful lesson in configuration management.

Formal assessment of the individuals or groups at GPT was unnecessary. Software managers indicated their satisfaction by continuing to send their new recruits to the course. Motivating attendees to take the course project seriously was not a problem because the new recruits tended to be enthusiastic and keen to make a good impression, especially when managers attended the final presentations and review.

Project progress. Progress on the GPT course project would start well. After deriving what they considered to be a well-thought-out specification and a realistic plan, which even allowed a little leeway for things to go wrong, the groups would usually manage to keep within if not ahead of schedule during the first week.

By half-way through the second week, each team would attempt to integrate the work of its members. The inevitable problems would lead to concern, then alarm, that the approaching deadline might not be reached. The end of the course became a scramble to get *something* ready in time. During this second week, significant disasters occurred that held up progress: an unknown fault would be found in the compiler, a team member would be out sick for a day, or the computer response time would fall to an unacceptable level. If all appeared to be going too well, the course leaders would introduce a few dirty tricks, although in over half the cases the groups produced so many problems for themselves that no further intervention was necessary! The final demonstrations were usually accompanied by a series of excuses as the products were delivered incomplete and full of errors.

Course value. The lessons learned on the GPT course were many, with working in groups being perhaps the most important for the few who had not experienced this before,[9,10] though it is now common in university undergraduate courses.

Although most participants knew various development philosophies,[11] no group ever tried to put these into practice, relying always on the conventional waterfall[11] development model. They then experienced the need to spend time in the design phase to get it right from the beginning, and the difficulties encountered when the design is changed in the coding phase. Most groups discovered that their designs were inadequate and that they would have benefited from using alternative development strategies.

The participants learned the difficulties of determining the requirements and getting the initial design correct when dealing with "real" customers who start with differing and vague ideas that they then clarify and develop in unexpected directions.[8,12]

The projects' integration problems taught participants the importance of defining interfaces, unit testing, planning, and allowing enough time for the integration process itself. In particular, participants learned the difference between testing a module to their own satisfaction and testing it to the satisfaction of other team members.

One overnight trick we played involved replacing all the computer file space with a copy made half a day earlier.

The participants learned their own limitations, particularly in their estimation and planning of tasks. They became more realistic, learning that they cannot expect to live in an ideal world, and that disasters are not exceptional. They also learned it was no use complaining to the "customer" about the lack of time or inadequate resources when they had negotiated their own targets with full knowledge of the development environment.

UNIVERSITY COURSES

We are aware, through contacts and

published papers, that many university computer science courses have some element devoted to teaching real-world issues[2,5,6] along lines similar to our GPT course. In this respect, Loughborough and Derby universities are typical.

Increasing team size is an advantage because larger teams force students to plan and organize their work more carefully.

Although we used our GPT experience to influence the courses at these universities, we know that other universities employ similar ideas and practices. It is useful, therefore, to identify the successes and failures in these courses.

Loughborough. For the last 15 years, Loughborough University's Department of Computer Studies has run group project courses in the second year of the undergraduate program. The students have taken group programming projects of either five or 10 weeks' duration, with students reporting that they spend around a quarter of their time on these projects. The projects let students learn to work effectively as a team. In earlier years, each team consisted of four students, growing more recently to groups of eight or nine each. The increasing team size is an advantage because larger teams force students to plan and organize their work more carefully.

Increasingly, elements of the GPT course have been introduced into the Loughborough University projects. Specifications are left deliberately vague and ambiguous, the requirements and priorities change throughout the project, and the module lecturer plays the roles of customers, managers, and quality auditors. The teams are encouraged to seek "customer" feedback during interviews and demonstrations, while the "manager" and "quality auditor" dictate the internal planning and reporting procedures. The main difference from the GPT course is in the extent of these real-world aspects. University resource restrictions mean that fewer changes are imposed and that customers, managers, and quality auditors are less demanding.

Derby. The School of Mathematics and Computing at the University of Derby also has modules that simulate the real world, but as at Loughborough, the modules are not as in-depth as the GPT course. At Derby, different modules highlight different aspects. One second-year undergraduate module concentrates on the customer, taking the relatively unusual step of using real customers for these projects. These customers may be a different staff member within the department, staff within other university departments, or even people from outside the university environment. In each case, the customer must have a suitably sized project that is needed but is not so critical that the software must be delivered in perfect condition. In these projects, student teams are directed to produce a delivery in at least two stages. This is enforced first to show the advantages of customer interaction and second to increase the chance that the projects will produce something of value within the available time.

Another module at Derby concentrates more on management of the group's project work. Students must present detailed plans at the start of the project and then regularly report on their progress in relation to their plans. The students experience frequent and strict quality audits of both their software and its associated documentation, and at certain stages must justify their ideas in a management presentation. Yet another module emphasizes teamwork and group dynamics in the project undertaken.

University limitations. The university modules do not attempt to give the students as wide an experience as did our GPT course. This is not due to a lack of time: students are expected to spend a total of around 80 hours on a module, including the time they are expected to work on their own. The GPT course totaled 74 hours. The Loughborough courses attempt the full simulation of the real world but with less role play, fewer changes, and fewer real-world procedures imposed. At Derby, each module may cover one aspect of the GPT course to the same depth, but other aspects receive less attention. Other than changes to customer requirements, neither university tries to simulate the day-to-day problems of the real world by playing tricks on the participating groups.

Closer inspection reveals the GPT course had several advantages over the university modules.

♦ Since their early school days, most students have been completing their work in a last-minute rush. As a deadline approaches, university students will abandon work for other modules and will often miss lectures. They put in long hours and it is not unusual to find them working through the night to meet a coursework hand-in date. For many students, this last-minute rush is normal and subconsciously planned from the start. Students do not appreciate the extra time they spend as a result of their planning failures. In contrast, we kept the GPT course strictly to company hours, with no extra work allowed even during the lunch period. Without the opportunity to compensate for delays and mistakes with additional time, participants taking the GPT course were surprised and alarmed to find how far wrong their planning could go.

♦ The level of supervision possible in the GPT course would be the envy of every university lecturer. The course leader was always present, with a second staff member also available sometimes. In a university course, students are supervised for only about one third of their project work time. Limited staff resources would make any further supervi-

sion difficult and disproportionate to other modules. The extra course supervision at GPT let the role play be more extensive, as "customers" and "managers" could be on hand more frequently to influence development. In university courses, extending the role play to present more than one customer becomes much more difficult; neither Loughborough nor Derby attempts to do so. At GPT, the availability of the course leader also made monitoring of teams more effective, enabling immediate action if a group was thought to need assistance or, in some cases, if the group was progressing too well and needed the distraction of an imposed disaster.

♦ The GPT course numbers gave a significantly better staff-student ratio. At GPT, a course was considered large if it contained a dozen students, whereas at universities a class size of less than 40 is rare, with numbers over 100 possible. Thus, time spent with each university project group is a fraction of what the GPT participants enjoyed.

♦ The lower supervision levels at the universities reduces the time a supervisor can observe the different groups' experiences with a view to emphasizing points in the review session. Also, unlike the participants in the GPT course, the student groups are not necessarily working side by side and so do not have the same opportunity to learn from other groups' experiences. In university courses, many of the significant events that illustrate important lessons will, unfortunately, be missed altogether.

♦ Because they were not being assessed or graded, there was no need to treat GPT course participants equally. For example, sending a key group member on an "urgent" errand for half a day could provide excellent lessons in team organization and product documentation. This event would be noted by all other groups, who would then be able to discuss and learn from its effects in the course review. The assessment of students in undergraduate modules implies that all participants must be treated fairly. This means that dirty tricks can only be played when they affect all students equally. The limitations on the tricks that can be played leaves students with too much of an expectation of an ideal environment and severely restricts their education on how to cope with the inevitable occurrences of real life they will later experience.

♦ Without the GPT course's level of supervision it is difficult for universities to assess each member of a group. Usually, the universities rely on an element of self-assessment in which students must evaluate each of their group members' contributions and attribute to each the work completed. While such analysis has some merit in forcing the students to review how they functioned as a team, the demarcation of tasks to individuals also reduces collective responsibility for the work. It becomes too easy for students to blame others for their failures.

♦ The students are in only their second year when they take the university software engineering modules. Because the GPT course was created for graduates, the participants inevitably had more computing experience and maturity, enabling them to gain a far better understanding of real-world issues. This suggests it would be better if the university software engineering modules were given in the final year. However, as assessment of individuals is difficult within group projects, the universities would be reluctant to introduce significant group work in the last year of a degree course.

University modules' value. Many of the lessons taught in the university modules are similar to those in the GPT course. The problems bring out the importance of teamwork, communication, planning, testing, and allowing enough time for the integration process, while the realistic conditions teach students that disasters are not exceptional.

When the modules are compared with the GPT course, however, there is a significant difference in how well these lessons are learned. The fixed hours of the GPT course would highlight the participants' inability to plan and deliver work on time in a way they had never experienced at their university. The close observation of the groups by the course leader allowed him to emphasize points in the review session by relating to events they experienced. We found that using the participants' own work as examples of good or bad practice was particularly effective at ensuring the message would be remembered.

Although the university modules give a useful insight into software development in the real world, they cannot match the industrial atmosphere created in the GPT course. The extensive role play, the standard company hours, and the ability to create everyday disasters gave the GPT course a realistic feel that would be difficult to simulate in a university environment.

The assessment of students in undergraduate modules implies that all participants must be treated fairly.

IMPROVING UNIVERSITY COURSES

No doubt most university lecturers will feel that, given the level of staff time and resources available to GPT, they could increase the effectiveness of any similar university course. However, the GPT course had a few advantages that could transfer effectively to the university environment without additional resources.

One of the GPT course's principle advantages was the set company hours. To emulate this, Loughborough restricted student access to the software to set hours during the week. However, the choice of

software used in the project is critical in this context because most students have access to the same software on their own PCs. For example, the Loughborough project involved accessing an Oracle server from a PC client. Although there was no practical way to restrict the groups' development of the client software, the module teacher presumed that students would not have access to their own Oracle database software. The experiment failed because the structured query language needed to access Oracle was similar to the widely available, PC-based Access database's SQL interface. The students developed their programs using their own computers, then devoted the restricted course hours to converting those programs to Oracle.

The need to be fair restricts the number and type of dirty tricks that can be played, but not some manufactured disasters.

Thus, if students are to be restricted in their project hours, care must be taken to use software that is not similar to or compatible with generally available PC software. Unfortunately, this restriction severely limits the potential candidates for the project's language and environment, making it quite likely that any programming languages that participants learned earlier in the university course would not be usable.

The problem of the relatively lower level of supervision in a university course is more difficult to overcome. Both Loughborough and Derby have had some success drawing on the students' own observations by making each group present their ideas and experiences to other students. In such a presentation, the students must justify their plans and actions, which shows they can be professional and learn from their mistakes. This supplements the lecturer's own comments in the review at the end of the module. Although reasonably successful, this process cannot compare with the informed comments of the GPT course leader who, by knowing what to look for, could pick out course examples of every type of good and bad practice to discuss in the review.

One suggestion to help increase university module supervision is to use more senior students, perhaps graduate students, to assist in the supervision. However, the requirement to be fair makes this difficult. The second-year students' learning experience would be clearly affected by the supervising student's abilities. For this reason, neither Loughborough nor Derby have any plans to implement this suggestion.

The need to be fair also restricts the number and type of dirty tricks that can be played on the groups. This does not stop certain manufactured disasters from being imposed on all groups, however. Changing the requirements as the project progresses is an easy trick to play and one that gives experience of an all-too-common real-world problem—yet many participants had not encountered this phenomenon before joining GPT.

Other types of change are rarely experienced by university students, though in many cases they would not be difficult to introduce. For example, on one occasion at Loughborough the database management software was upgraded to the next version in the middle of the project. Despite the new version being so-called "backward compatible," a whole week passed before the students could run any of their project software—a useful and very realistic foretaste of the real world. Nevertheless, the GPT course leader's ability to tailor the disasters to the circumstances of each group inevitably made these lessons more appropriate, more timely, and consequently, more effective than the universities could possibly achieve.

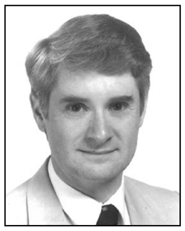
That veteran GPT software managers continued to send newly recruited computer science graduates to the course we developed indicates they believe it to be valuable. That universities introduced similar modules shows that educators also recognize the need for providing real-world software development experience. We believe that the Loughborough and Derby modules are typical of modules in many, if not most, university computer science courses. Although these modules impart valuable experience, we found the GPT course to be far more effective in delivering the necessary lessons.

Much of the reason for our course's superiority can be attributed to full-time company hours and greater staff and equipment resources. Although, as we've suggested, universities can overcome some of their time and resource limitations, their modules will never match the learning experience of a company course.

Perhaps, then, the ideal situation is to mix university education and company training. The computer science students can and should receive some real-world experience while still at university, through some form of software engineering project modules. This should then become a foundation for one or more further training courses that will be provided when the graduates enter a company. None of this training will prevent totally the all-too-common mistakes that can give software development such a bad name,[12] but by teaching computer science graduates to be more aware of real-world issues, universities and software companies can increase professionalism in the software industry. ♦

REFERENCES

1. R.J. Dawson, R.W. Newsham, and R.S. Kerridge, "Introducing New Software Engineering Graduates To The 'Real World' at the GPT Company," *Software Eng. J.*, Vol. 7, No. 3, 1992, pp. 171-176.
2. L.M. Leventhal and B.T. Mynatt, "Components of Typical Undergraduate Software Engineering Courses: Results from a Survey," *IEEE Trans. Software Eng.*, Vol. 13, No. 11, 1987, pp. 1193-1198.
3. C. Alder, "Software Engineering in an Electronic Engineering Degree," *Software Eng. J.*, Vol. 4, No. 4, 1989, pp.191-199.
4. R.J. Dawson and R.W. Newsham, *The Responsible Software Engineer*, C. Mayers, T. Hall, and D. Pitt, eds., Springer-Verlag, London, 1996, pp. 320-331.
5. J.J. Horning and D.B. Wortman, "Software Hut: a Computer Program Engineering Project in the Form of a Game," *IEEE Trans. Software Eng.*, Vol. 3, No. 4, 1977, pp. 325-330.
6. B.M. Roper, "Training First Year Undergraduates to Produce Quality Software," *University Computing*, Vol.10, 1988, pp. 9-12.
7. D.B. Wortman, "Software Projects in an Academic Environment," *IEEE Trans. Software Eng.*, Vol. 13, No. 11, 1987, pp. 1176-1181.
8. M. Jarke and K. Pohl, "Requirements Engineering in 2001: (Virtually) Managing a Changing Reality," *Software Eng. J.*, Vol. 9, No. 6, 1994, pp. 257-266.
9. F. Milsom, *The Responsible Software Engineer*, C. Mayers, T. Hall, and D. Pitt, eds., Springer-Verlag, London, 1996, pp. 306-319.
10. N. Chapman et al., "'Slick Systems' and 'Happy Hackers': Experience with Group Projects at UCL," *Software Eng. J.*, Vol. 8, No. 3, 1993, pp. 132-136.
11. R.S. Pressman, *Software Engineering—A Practitioner's Approach*, European ed., McGraw-Hill, New York, 1994.
12. M. Van Genuchten, "Why is Software Late? An Empirical Study of Reasons for Delay in Software Development," *IEEE Trans. Software Eng.*, Vol. 17, No. 6, 1991, pp. 582-590.



Ray Dawson is a lecturer in computer studies at Loughborough University. His interests include software development practices, software quality and cost, software estimation and planning, systems analysis, databases, and object-oriented methods. His first industrial position was with Plessey Telecommunications (which later became GPT) developing software for digital telephone exchanges. In 1983, he became a software lecturer with Plessey before moving to Loughborough University in 1987.

Dawson received a BSc in mathematics with engineering and an MPhil from the University of Nottingham. He is a member of the British Computer Society, through which he received Chartered Engineer status.

Ron Newsham is a senior lecturer in software engineering at the University of Derby. He is researching the maintenance problems of software design notations. He was a lecturer in the GPT Beeston Software Training Department between 1988 and 1995, becoming the software training manager in 1995. Prior to that he was a software tools specialist supporting development on System-X telephone exchange software.

Newsham received a BSc in music and electronics from Keele University and an MSc in Software Engineering from Nottingham Trent University. He is a member of the British Computer Society, through which he received Chartered Engineer status.

Address questions about this article to Dawson at Dept. of Computer Studies, Loughborough University, Loughborough, Leicestershire, UK; R.J.Dawson@Lboro.ac.UK; or to Newsham at School of Mathematics and Computing, University of Derby, Kedleston Road, Derby, UK; R.W.Newsham@Derby.ac.UK.