Twenty Dirty Tricks to Train Software Engineers

Ray Dawson

Dept of Computer Science
Loughborough University
Loughborough, Leics. LE11 3TU, UK
+(44) 1509 222679
R.J.Dawson@Lboro.ac.UK

Abstract

Many employers find that graduates and sandwich students come to them poorly prepared for the every day problems encountered at the workplace. Although many university students undertake team projects at their institutions, an education environment has limitations that prevent the participants experiencing the full range of problems encountered in the real world. To overcome this, action was taken on courses at the Plessey Telecommunications company and Loughborough University to disrupt the students' software development progress. These actions appear mean and vindictive, and are labeled 'dirty tricks' in this paper, but their value has been appreciated by both the students and their employers. The experiences and learning provided by twenty 'dirty tricks' are described and their contribution towards teaching essential workplace skills is identified. The feedback from both students and employers has been mostly informal but the universally favourable comments received give strong indications that the courses achieved their aim of preparing the students for the workplace. The paper identifies some limitations on the number and types of 'dirty tricks' that can be employed at a university and concludes that companies would benefit if such dirty tricks were employed in company graduate induction programmes as well as in university courses.

Keywords: Education, Training, Project, Teamwork, Work experience

1. BACKGROUND

This paper is based on the experience of training projects to simulate the 'real world' given to graduates and undergraduates over the past 17 years. The history of these training projects can be traced back to when they were first introduced for new graduates by the author at the Training School at the Plessey Telecommunications company [4] (which later became GPT and is now part of the Siemens group). The author has since moved to Loughborough University, but the training courses at Plessey were continued after the author's departure and the trainers have kept the author informed of all developments in the course. The author has been involved in the development of similar projects for undergraduates at Loughborough where many of the features of the Plessey course have been adapted for the university environment [5].

2. SIMULATING THE REAL WORLD

In 1993 a two week, full time training course was set up at the Plessey company at the request of the company's software managers. Their experience of new computer science and software engineering graduates was that it would usually take up to six months before these graduates made a "useful contribution" to a software department. In discussions with these software managers it soon became clear that they believed that it was not just a lack of knowledge of the company tools and product line that reduced the graduates' effectiveness in the early months, but rather it was their lack of awareness of and preparation for the realities of the workplace such as changing objectives, problems with clients, or the pressures of imposed deadlines [2,4,6,7,12,13]. The course was set up to fill this gap in the graduates' education with up to twenty computer science graduates participating each year.

The course duration was two working weeks consisting of a half day of lectures at the beginning, a half day review session at the end, and the remainder devoted to a project that simulated a real working environment. The initial lectures were to prepare the students for the project. The aims were explained and the difficulties encountered in real world software engineering projects were described. This was then followed by the project itself. The students worked in teams of four or five members each and with two to four teams working on similar projects in competition with each other. The students were subjected to the typical company working procedures, methodologies and deadlines with role play by the course leader to simulate a customer, a manager and other personnel as required. When available, other training staff members were used to play some of the different roles. The review at the end of the course allowed

students to describe their experiences and the course leader to draw attention to the lessons that can be drawn.

At Loughborough University a teaching module for second year undergraduates was organized with similar aims to the Plessey course. The format of the course had to be altered, however, to fit in the modular framework of the university degree. The university real world projects were run over a period of eight to ten weeks on a part time basis, the students studying five other modules in parallel. Team sizes have varied from four to ten with many more groups working in parallel. The numbers of students involved (between 120 and 160 each year) and a general lack of staff time did mean that the module leader's contact with the teams was limited in comparison with the Plessey company course. However, there was sufficient time and resource to allow the role play to create the necessary personnel for a real world simulation.

3. WHAT ARE 'DIRTY TRICKS'?

Group project work is very common in university education, and often the groups work on realistic software developments [3,5,8,11], so what was new in the Plessey and Loughborough University real world projects? The difference lay in the lengths taken to make the students experience the most adverse aspects of software development encountered in the real world [4]. Every experienced worker knows the real world is full of surprises that can hinder a development [2,9]. A disruptive software upgrade, the departure of a key team member or political infighting between different stake holders are common events in real life but they are seldom experienced on a university course [6]. Indeed at a university events such as software upgrades are timed to occur in the holidays whenever possible precisely to avoid any disruption to students' work. The whole ethos of an education establishment is to produce the best possible environment for the students to learn, but this can create an artificial atmosphere well removed from the real world.

In the courses at Plessey and Loughborough University steps were taken to re-introduce the problems experienced in the real world. This amounted to purposely hindering and disrupting the software development processes of the students involved. A series of actions were taken that at first sight appear to be mean and somewhat vindictive and which could even be interpreted as a sadistic desire to antagonize the innocent participants. This was not the case. Every 'dirty trick' action described in this paper has its purpose in simulating a common real life situation and each has an associated learning experience which will be of value to those on whom the dirty trick is inflicted!

In the next section twenty 'dirty trick' actions to simulate real world experience are described, and in each case the lessons that can be learned are detailed. In practice, the full collection of all twenty tricks has never been played on any one group of students, but each of the actions described has been tried either at the Plessey company or at Loughborough University at some point, and each has been found to give a valuable experience. Some of the tricks, such as banning overtime or purposely bringing down a computer, proved to be difficult or impossible in the university environment. Other tricks, such as changing the hardware platform, proved impractical in the company training school. Some tricks such as the inadequate specification and the changing requirements have been used every time the course has run. In general about half the tricks described may be used in any one course, with it being the responsibility of the course leader to decide which tricks are appropriate and possible at any one time.

4. THE TWENTY 'DIRTY TRICKS'

4.1 Give an Inadequate Specification

The students should be given a "specification" of typically no more than two or three pages long. Although it will at first sight appear to describe what is wanted, it should when examined in detail be far too vague. Statements should be ambiguous in places and some aspects can be omitted altogether. Although it should be labeled a specification it will in fact be no more than a concept document outlining the initial ideas of the project only.

Why? As any practicing software engineer will know, in the real world complete, unambiguous specifications are rare if they ever exist. This, however, is not the experience of students on undergraduate courses. Their set coursework is usually fully defined with teachers, in the interests of fairness, accepting any solution that could be considered to meet the requirements of the assignment as given. Students sometimes will seek clarification but this is not actively encouraged with students more likely to be told to read the question more carefully. This is an unfortunate training as the real world is not as forgiving.

Lessons Learned: An inadequate specification is needed to teach students that close liaison with the customer will be required to obtain the requirements, and the developers themselves may need to produce the specification to obtain detailed agreement with the customer.

4.2 Make Sure All Assumptions are Wrong

If the students have not sought clarification when the specification is ambiguous the 'customer' should ensure that the assumptions they have made are the wrong ones. This involves deliberately choosing variations on the project scenario to catch them out. This philosophy should be extended to any additional features included because the students assume that the customer will be pleased to receive a 'superior' product with added bells and whistles. In practice, in the role of customer it is not usually difficult to find some plausible reason to reject any unrequested addition. For example, an extra password feature could be rejected because "only authorized personnel will have access to the computer anyway".

Why? It is very common for students to make assumptions about what the customer would like. This is again part of their university experience where in the interests of fairness the teacher will accept any solution that fits the specification. Unfortunately, undergraduate students who deliver work with extra, unrequested features are often rewarded for their initiative with extra marks. This creates a dangerous ethos that any extras can be assumed acceptable without the need to ask the customer first.

Lesson Learned: This trick may seem to be particularly mean but there is an important lesson here that communication with the customer is essential at every stage of development. Students must learn that they do not have the whole picture of the problem and that different people with different viewpoints may have a quite different perspective on the needs and priorities. Asking first instead of making assumptions is an essential survival philosophy for software developers.

4.3 Present an Uncertain and Naive Customer

In the role play employed, the customer should be typical of many real world clients in not being sure of what he or she wants. As in the real world, knowing there is a problem should not be the same as knowing what the solution should be. This is particularly significant if the customers are personnel that have no experience of computers so that they cannot even imagine what form the solution should take.

Why? The usual student scenario is that the students expect their 'customer', the teacher who gives the assignment, to know far more than they do - they make the reasonable assumption that if the teacher is going to mark the work delivered then they must know what they want. Role play is needed to ensure the students can experience other types of customer. Representing a customer without computer literacy is surprisingly difficult for software engineering educators who are inevitably experts in the field, but it gives a valuable experience for the students who have not usually experienced such a customer before.

Lessons Learned: The lessons in dealing with uncertain and naive customers are that tact and a great deal of patience are required, and that customer education and training can be as significant a deliverable as the software itself. Undergraduate courses again fail to give students any experience of this.

4.4 Change the Requirements and Priorities

Once the project assignment has been set it should be changed on a regular basis. The aim here should be to change the specification in ways that seem perfectly reasonable so that the customer is not seen to be making arbitrary changes. For example a university class scheduling system may be given the additional requirement to ensure that wheelchair bound students are not scheduled in rooms with no wheelchair access.

Why? In a real world nothing stands still, the requirements and priorities of a project are changing all the time. In the artificial world of education and training, however, it is unusual for an exercise to be altered from the time it is set to the time the students hand in their work. To prepare students for the real world the requirements and priorities must change as frequently as possible. For example, at Loughborough University recent projects lasting eight weeks had changes to the requirements made every week.

Lessons Learned: The students learn that events in a real world make change inevitable so a software engineer must, therefore, plan for change with open architectures, adaptable designs and flexible planning.

4.5 Have Conflicting Requirements and Pressures

Introduce the students to a no win situation where it becomes impossible to satisfy two conflicting requirements. For example, a requirement to make certain output graphical may be incompatible with a maintenance requirement to use a particular software package. Often the conflict can be between satisfying the 'customer' and pleasing the 'manager'. A manager may want to save costs by reusing existing software, whereas the customer may have quite different ideas based on a desire for compatibility with other software products.

Why? The nature of education is that students are used to "right" and "wrong" answers to a problem. They can therefore have considerable difficulty when encountering a problem where there is no perfect answer.

Lessons Learned: The lesson here is that in real life compromise is often necessary and that negotiation is required to enable such a compromise to be accepted by the customer. This is a particularly important lesson for certain students who take pride in being perfectionist; they learn that perfect solutions will not always be possible in a less than perfect world.

4.6 Present Customers with Conflicting Ideas

A particular example of conflict can be created by introducing more than one customer with more than one solution idea. Most real systems are used by more than one person and this inevitably means that there will be different solution ideas. In a training project the role play should include different customer personalities each with different priorities and objectives. A 'manager' may want a system that has facilities for monitoring and reporting usage statistics whereas an 'operator' may find this totally unacceptable. In this case the students would be given quite different messages from the two different roles, a situation that can be made more difficult by ensuring that the two personalities are never available at the same time.

Why? Staff resource restrictions mean that very few students have encountered a problem with more than one customer at their university. This type of conflict shows that more than one viewpoint in a problem is possible, and that

developers need to communicate with all users and stakeholders to build a complete picture of requirements.

Lessons Learned: Students learn that, unlike in the university environment, satisfying one person does not guarantee that the solution offered will be acceptable. Students also learn that the politics of dealing with people makes requirements analysis a far more complex task than simply finding out the facts.

4.7 Present Customers with Different Personalities

It is instructive to provide customer roles with different personalities as well as viewpoints. In the Plessey training projects, for example, one customer would be very enthusiastic readily accepting any suggestion put forward while another customer would be very reluctant to deviate from his original ideas. Of the two, experience has shown it can be the enthusiastic customer that gives the most problems, leading the students into commitments they could never achieve with statements such as "Oh yes, that is a good idea and we could do the same thing with X, Y and Z too couldn't we?". The students can be drawn into commitments well beyond their abilities to deliver.

Why? In real life each person is different and so negotiating styles also need to differ when handling different personalities. This is not usually experienced by students as they have rarely encountered more than one customer on a project.

Lessons Learned: This re-enforces the lesson that it is the people that complicate the analysis of requirements, and again it shows that different view points are possible. The students learn that negotiation is a necessary skill where care and caution are just as important as the students natural willingness and enthusiasm to please. They learn to think before they act and consider the consequences before they commit themselves, and that even the principle that "the customer is always right" may require compromise.

4.8 Ban Overtime

The students should be restricted to a strict number of hours for the project development. For example, at the Plessey company work on the training projects was limited to set company hours with no extra work allowed overnight or even in the lunch hour. In a university environment this restriction is difficult if not impossible to impose as most project work is mixed with other work and spread over a longer period. Often, therefore, it is only when students comes to their first place of employment that they experience restrictions on their work time.

Why? Students regularly take extra time to finish their assignments. Most students will work long hours in the last few days before a deadline. It is not uncommon to work through the night and to abandon other work and lectures if the assignment is significant. This last minute rush becomes such a way of life for most students that they do not realize how much extra time is put in and consequently how poor their original estimates were.

Lessons Learned: Even if no changes are imposed to disrupt the project there are lessons to be learned in having a set number of hours to complete a task. For university students this is an experience that few have encountered when developing software. It can be a real shock for students to find the final deadline approaching with no way of putting in the extra hours of work required. Most are genuinely surprised to find how bad their time estimation and planning had been.

4.9 Give Additional Tasks to Disrupt the Schedule

In addition to the project they are working on the students should be required to undertake further activities which are not known about in advance. The additional activity could affect the whole team, for example, a "manager" created by the role play could call a short notice progress meeting which requires the preparation of reports as well as the time to attend the meeting itself. Alternatively an errand could be found for a key individual to, say, deliver an item to another site which will take him or her away from the team for half a day.

Why? In the workplace meetings, training, administration and even coffee breaks all take time and yet there is rarely adequate planning for such activities.

Lessons Learned: Planners in the real world will be aware that there is a tendency, whether in software development or otherwise, to forget the time overhead of activities not directly related to the principle task. The students learn that these other activities are commonplace so they must be more realistic and flexible in their planning.

4.10 Change the Deadlines

The students should be told part way through the project that the customer requires the product at an earlier date than originally specified. There should be room for negotiation with scope for only some deliverables to be delivered at the earlier date, the rest being delivered later or possibly being dropped altogether. However, the students should not be offered any compromise in the first instance, all flexibility only coming through negotiation.

Why? Changing the delivery date for a project is a particularly hard hitting exercise for students who are used to being given a project with a delivery schedule fixed from the outset. In real life deadlines do change but it is a matter of negotiation of what the change will be and what the cost will be to the customer. In an education or training environment, however, there is not usually any scope to negotiate in terms of cost, but negotiation may be possible if the project requires a number of deliverables to be produced at different stages.

Lessons Learned: The lessons in this are that deadlines are subject to a number of influences. Students are used to

negotiation over deadlines when trying to persuade their supervisors to allow late delivery of their work because they had been ill, their computer had crashed, or whatever. They are quite unprepared, however, for the tables to be turned with the project "customer" asking for an earlier delivery. They learn that simply complaining about their position is less likely to yield results than well argued negotiation, but also that this is a two sided affair with both parties looking for maximum advantage. Once again, there are also lessons that change is inevitable and that flexibility is a major asset in project planning.

4.11 Introduce Quality Inspections

Role play in project work in a real world simulation needs to include characters other than the management and customers. The role of a quality auditor requiring inspections at very short notice can be usefully introduced. This is particularly effective when the team is feeling most vulnerable, such as when the project is about two thirds complete and the first sign of problems of integration of team members work start to appear.

Why? Many students pride themselves in being able to produce "high quality" software. In reality code comments and documentation are often produced at the end of the project despite what they may claim. This can be a problem in the workplace when outside influences can mean a team member can be lost at short notice leaving others to continue their work.

Lessons Learned: Students learn that the real world requires standards that must be maintained throughout a project and not handled as an afterthought.

4.12 Present a 'Different Truth'

The customer should say one thing one day and something else the next and then deny that anything different has been said. Put more bluntly this means telling bare faced lies! However, the different statements should not be obviously contradictory but should be disguised in a different emphasis or in a different context. For example it may be stated one day that "only the manager will use the software" but the next day this becomes "the management team will use the software". This change could mean that in addition to the manager, the section leaders and even the manager's secretary are involved with all the potential multi-user access this implies. However, while the change may have major significance there is only a slight change in the words. This leaves the students with some degree of self doubt about what the were told in the first instance which in turn undermines any attempt to protest.

Why? Students need to appreciate that in the real world mistakes are made but not everyone will own up to them or even realize they have made a mistake at all.

Lessons Learned: The lesson here is not simply that there are dishonest people about but that genuine mistakes can be made. The students learn the value of having agreements on paper and not just in spoken form. They learn to protect themselves by taking notes during interviews and meetings and to double check they have accurately recorded the information. They learn the hard way that they cannot rely on the word of others or even their own memories without written evidence.

4.13 Change the Team

The team membership should be changed mid project if at all possible. Where there are a number of team projects working in parallel this can be achieved by swapping team members around. Ideally only small changes affecting just one team member should be made but this should be done more than once during the life of the project. If the team has a dominant character that monopolizes the planning that person should be the prime target for any change.

Why? Another unreal aspect of group project work in educational and training environments is that project team membership usually remains stable. In the real world this is less likely to be the case. The stability of a team on a project of more than a few months can easily be disturbed by members joining or leaving the team.

Lessons Learned: The lesson in this is that communication is a necessary part of team work. If the removal of key team members means the team can no longer function then it means that they had kept too much information to themselves. The students learn that for effective teamwork each member needs to know their own role and how it fits with others. If the team communicate well and the team structure is known then a flexible approach means that it can be adapted to meet unexpected personnel problems.

4.14 Change the Working Procedures

In the role of manager, the course leader should lay down the working practices expected of the student teams. For example, regular progress meetings and interim internal reports could be required. However, this like the project specification should not be stable. The project manager could change his or her mind, could suffer from other work pressures that prevent his attendance at meetings, or could even be replaced himself. In terms of the role play this would simply involve the course leader announcing that the previous manager had been promoted to a job elsewhere and that now a new manager had taken over, though to make the change effective the new manager would need a new personality who requires significant changes in procedure and the product produced.

Why? Changes in management personnel and procedures are not experienced at university where changes in the teaching staff or teaching conditions would not normally occur during the course of a student project.

Lessons Learned: There are three lessons here. Firstly the lesson that teams need to be flexible and adaptable is re-enforced, secondly, that time must be planned for the inevitable disruption caused, and thirdly, the importance of

quality in both product and development process is emphasized. On this last point it is usually possible to show that teams who are well organized and have actively promoted quality in their work are more able to adapt to externally imposed changes.

4.15 Upgrade the Software

If possible, the software used for the project development should be 'upgraded' to a later version during the project life. The upgrade should be claimed to be fully backwards compatible and the students should be told that the upgrade should not affect them. It will of course, as any experienced software engineer will know. One course at Loughborough University experienced an upgrade of database software which completely eliminated any further work before the end of the project! This effect was more severe than even the module leader had anticipated.

Why? The purpose of this trick is to dampen students enthusiasm and desire for all the latest software and gadgetry. Students tend to believe that being up to date with the latest fashion is the only real measure of quality.

Lessons Learned: Students need to learn that the whole picture must be taken into account and that every new acquisition has a price in terms of time and effort as well as in monetary terms, and this must be balanced against the gains obtained. It also teaches students to be realistic, to have a healthy skepticism of manufacturers' claims, and to always allow contingency time for the inevitable unexpected problems every project encounters.

4.16 Change the Hardware

Towards the end of the project the customer should announce that they have just decided to standardize on a particular hardware platform which is different from the system under development in the project. Obviously the customer will then want the product to be changed to work on this new platform. The trick is to make this change possible, though at a price. If the product is being developed in, say, C or Java then the change is at least theoretically possible, but in practice the different environments will inevitably mean some quite extensive changes are needed in the software being produced.

Why? Handled correctly this could be the students first introduction to legacy systems. The students will need to decide whether to take time to change the existing system with the risk loosing some product functionality through the time lost, or to continue with on the current, legacy platform to enable the full product to be developed. The answer will depend on the circumstances of each project, but either way it is instructive for students to have to consider the problem.

Lessons Learned: This teaches the value of keeping to standard features of a programming language and the problems and issues of porting software. Even if the students opt not to change platforms (and usually that is the only practical option) by pressing the students to consider the problem the lessons can still be effective.

4.17 Crash the Hardware

This may be a trick held in reserve in case any project team appears to be doing too well. If a mainframe is being used then bringing down the computer is relatively straight forward unless there are users not involved in the training who are sharing the computer. If PCs or other single user computers are being used then there is often the opportunity to take down a server to give the same effect. In the 'customer' role, however, the course leader should be unsympathetic about the delays incurred.

Why? A hardware crash is typical of the every day disasters that occurs in most software projects. Projects are completed late far too frequently in the software industry and yet whenever this happens, the developers always seem to be able to put the delay down to some excuse or other.

Lessons Learned: The students need to be shown that to *always* have an excuse is unacceptable, disasters are not exceptional in the real world and to assume, as students usually do, that everything will go smoothly is not reasonable. The universities, in their attempts to be fair do, unfortunately, encourage an excuse culture by always trying to take into account any unforeseen adverse circumstances a student may encounter. Students need to learn there are limits for excuses in the real world.

A useful by-product of taking the hardware out of action for a while is that students can find themselves with time to stop and think. Such reflection will often cause the teams to reassess and adjust their approach. This is a useful lesson in the value of reviewing and updating plans throughout a project, activities which are often forgotten in the rush to meet an approaching deadline.

4.18 Slow the Software

When a project is in its later stages of development the development hardware and software tend to be under the greatest demand. This will often mean that performance starts to suffer with compilations, builds and test runs taking annoying lengths of time that leave the students frustrated, looking at an hourglass on the computer screen. If this does not happen of its own accord it should be made to happen by loading the computers or network with unrelated activity such that everything slows down.

Why? The slowing down of a system under load is a common event and so it cannot be acceptable as an excuse.

Lessons Learned: Like the hardware crashes, the main lesson is to show students they must be more realistic in their planning. Some of the more organized students manage to avoid time wasting by ensuring they have other tasks they can undertake while waiting for the computer to respond. This acts as a useful lesson to other students that action can often be taken to minimize the effect of these every day disasters if a little thought is given beforehand.

4.19 Disrupt the File Store

If possible, disrupt the file store the students are using. At Plessey this was done over a lunch period by replacing the students file area with a copy made the previous evening. This sort of disruption is relatively straight forward if the file store area is kept on a central machine. However it becomes much more difficult in distributed systems and indeed, to the best of the author's knowledge, it has not been successfully carried out in such an environment

Why? This sort of restoration of the last backup dump is not uncommon in a main frame style environment. Whether, the students are then told that computer problems have meant the file store "has reverted to the state at the last backup" or whether the students are left to work this out for themselves, the effect will vary enormously from one team to another.

Lessons Learned: This gives an excellent lesson in configuration management, particularly when contrast can be made between a number of teams working in competition to each other. A well organized team with good configuration management can recover remarkably quickly, other teams may never recover. It also shows the students the value of quality in the development process as a team that knows and keeps to systematic processes, even though the files have reverted to their state half a day previously, can loose far less than the half day overall.

4.20 Say "I Told You So!"

This is probably the most infuriating trick of all. At the start of the project the students can be told what sort of experiences they will encounter, and they can be told when they will feel they are on target (up to about three quarters of the way through), when they will start to go wrong (in the integration phase) and how badly they will fare overall - yet the project will still follow the same pattern. The course leader can then give a very smug expression and declare "I told you so!".

Why? The expression "I told you so" is so annoying for those at the receiving end who then feel frustrated, rather guilty and angry with themselves, but nothing works better than these few words to drive a point home!

Lessons Learned: The students learn much about themselves and in particular their own limitations through the activities described in this paper. They learn to become much more realistic about their own and others' capabilities and about the environment they work in. Above all they learn that they still have much to learn.

5. STUDENT REACTIONS

Whenever the courses at either the company or the university have been described the first reaction of academics and industrialists is surprise that the author ever lived to tell the tale! Concern is expressed that the students would get angry, frustrated and disillusioned on these courses and this could prevent any value being gained. This in fact is quite the opposite of the real case. Providing the students have the aims and objectives explained to them at the beginning and have the lessons reviewed at the end they have all been very appreciative of the value of the course. Indeed many comment that they found they have enjoyed the challenge and that the course had proved to give one of the their most interesting and rewarding learning experiences.

In the sixteen years of running these courses there have only been two cases, both at the company, where the student has taken exception to the 'dirty tricks' used. In both cases they declared the actions were completely unfair and they declined to participate further in the course. Significantly, in each case the graduate later proved to be unable to fit in to the company's software development environment and both left the company after only a short period. It must be stressed that the Plessey Telecommunications software development environment was and still is typical of any company with a large software component to its products. The vast majority of graduates settle in well and employees recruited from other companies have no difficulty adapting to the environment at the company. It must be concluded, therefore, that the two graduates concerned were exceptional cases that are likely to have had difficulties in any software development organisation.

	Problem understanding	People handling skills	Negotiation skills	Compromise skills	Planning skills	Adaptability	Quality understanding	Organizational skills	Design skills	Possible at a university?
Inadequate specification	✓	✓	✓					✓		✓
Make assumptions wrong	✓		✓				✓			✓
Uncertain customer	✓	✓	✓							✓
Change requirements	✓		✓		✓	✓			✓	✓
Conflicting requirements	✓		✓	✓						✓

Conflicting customer ideas	✓	✓	✓	✓		✓				✓
Different personalities	✓	✓	✓			✓				✓
Ban overtime					✓			✓		X
Additional tasks					✓	✓		✓		**?**
Change deadlines			✓	✓	✓	✓				**?**
Quality inspections					✓		✓	✓		✓
Different truths		✓				✓				✓
Change the teams		✓			✓	✓	✓	✓		X
Change working procedures		✓				✓		✓		✓
Upgrade the software					✓	✓				**?**
Change the hardware					✓	✓				✓
Crash the hardware					✓	✓		✓		X
Slow the software					✓	✓		✓		X
Disrupt the file store					✓	✓	✓	✓		X
Say “I told you so”	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: The skills and understanding gained from the twenty dirty tricks.

6. THE OVERALL LEARNING EXPERIENCE

Each individual ‘dirty trick’ gives its own real world experience and learning outcomes. Overall the course gives a good preparation for the workplace. Several essential skills for software engineering are developed. The students gain a greater understanding of what requirements analysis really means, that there is not necessarily right or wrong answers to everything and that the people in the system make the analysis considerably more complicated [6]. They learn that people are human and make mistakes so they must be realistic about how correct and complete information they are given may be. They learn the value of people handling skills and that negotiation and compromise are also essential skills for dealing with some conflicts. They learn that change is inevitable, that disasters are not exceptional and that project planning must allow time for the many unexpected problems that occur each day. They also learn they must be flexible and adaptable in their designs, organisation, methods and planning to cope with the unexpected. Above all they learn to be more realistic in their expectations and to know some of their own limitations.

Table 1 shows where some of the essential skills come from in terms of the dirty tricks inflicted on the students. The table should not be taken as being a complete and exact coverage of the relationship between the skills and tricks described. For example, problem understanding could to some extent be gained from all the dirty tricks employed. The table, therefore, concentrates on the most significant outcomes and emphasis of each action.

7. DOES IT WORK?

It is difficult to quote anything other than informal feedback to justify the claim that these courses are very successful. Formal feedback forms at both the company and the university showed the students felt the courses to be enjoyable and valuable, but as this tends to be the case for nearly all hands on practical work it is difficult to attribute any significance to the effect of the ‘dirty trick’ actions. However, the informal feedback is plentiful and consistently enthusiastic so there is some reason to feel the courses are successfully achieving their goal of preparing the students for the realities of the working environment.

At the Plessey company the informal feedback came from the software managers and other experienced employees who worked with the graduates after attending the course. On some occasions representatives from the students eventual destination departments attended the review session at the end of the course to see how their new recruits were getting on. Always the feedback was that the experiences the students were getting gave an excellent preparation for what they would encounter when they started work in their development teams. The ultimate indication of the company course’s success was that the managers, without exception, continued to send new graduate employees on the course year after year.

At the university the project module specification is subject to peer scrutiny and the students project work is reviewed by two internal and one external examiner. All have expressed satisfaction that the course has value in academic terms. However, it is the University’s informal feedback from industrial contacts that arguably gives the greatest indication of success. The Department of Computer Science has an Industrial Advisory Committee to give advice and guidence in its course development. The real world project module was presented to this committee as recently as June 1999 when it received an enthusiastic response. The real evidence, however, is in employers’ reaction to graduates and industrial sandwich year students from the University. Employers have reported back to the university that they

found the students from Loughborough to be particularly well prepared for the workplace. The most common comment being that they seem more able to think and find out for themselves, whereas many students from elsewhere have to be "spoon-fed" information and told what to do all the time. As a result many companies have contacted the Computer Science Department looking for ways to encourage more students to apply for positions at their organisation.

8. COULD THE UNIVERSITIES DO MORE?

The value employers have clearly put on the courses described prompts the question "Should the universities do more 'real world' preparation?" [6,10]. This, however, is a controversial subject as many academics would argue that this is really 'training' rather than 'education' and the rightful place for such preparation is at the companies' own workplaces. The counter argument is that as the lessons learned seem to be so significant any course that does not include some element of real world teaching is failing to give the complete picture. A balance is needed which Loughborough University believes it has achieved. The real world project module is given as one of twelve modules in the second year, which makes it one thirty-sixth of the degree. This, it is believed, is neither neglecting the subject or overdoing it. The picture is not quite as simple as this, however, as elements of real world teaching can be found in many of the other modules, but on the other hand, there are some teaching and learning aspects found in the real world project module that are not directly connected with the real world issues discussed in this paper. The 'dirty tricks' experience is only given in the one second year module which it is believed is sufficient.

If the universities do provide a module of real world preparation, do the companies need to provide any such training themselves? The answer must be 'yes' if the complete real world experience is to be given. The problem is that there are some real world experiences that are difficult or impossible to provide at a university. One problem lies in the shared nature of the university computers which means students taking a particular module are unlikely to have exclusive use of equipment. This makes it impossible to bring down a computer or network, slow the software performance or disrupt the file store. It also means that the timing of activities such as upgrading the software is limited and may not be possible every time the real world module is run.

Restricting the development time to set hours also proved to be surprisingly difficult in the university environment. The difficulty is that staffing resources means that the students cannot be supervised at all times during a project - they have to be left to do some of the work on their own. A further problem is that the licensing costs of development software means that if large classes take the module the software tools used for project work tend to be inexpensive. This means the students often have their own copies on their own computers which allows them to by-pass any restrictions on the use of university owned software and hardware. Even if the software is not generally available for student ownership there is often something similar that the students can use instead. For example, an attempt to restrict the students access to an Oracle® database server was defeated as students developed and tested their SQL using Microsoft Access™ on their own computers.

The universities also have a duty to be fair when assessing students performance. This means a dirty trick cannot be used on an individual person or team without doing the same to all students involved. This severely limits the scope to provide additional tasks to disrupt the schedule. It also means that it is not feasible to change the teams mid project - even if all team members were equally affected it would make the assessment of individuals very difficult. Fairness is also associated with openness which is being increasingly practiced in the universities. For example, at Loughborough, all coursework hand in dates must be specified before the module starts which makes changing deadlines impossible. Other actions become more difficult but are still possible. Staffing limitations means that more than one member of staff may not be available so the course leader may have to play all the roles him or herself. This and the limited time available means that students get less interaction with each role though there is still sufficient contact to ensure valuable experience is gained.

An extra column in Table 1 indicates which of the dirty tricks given in this paper can be employed on a university course. Those tricks marked with a '**?**' indicate it might be possible but it depends on the circumstances at that university. Those tricks marked with a 'X' will prove difficult, if not impossible, at any university.

9. CONCLUSIONS

This paper has described courses given at the Plessey Telecommunications company and Loughborough University to provide new graduates and undergraduate students with a better preparation for the real world. A simulation of the real world in a project in an education environment has many limitations that prevent the participants experiencing the full range of problems likely to be encountered in the workplace, however. To overcome this, action was taken on the Plessey and Loughborough courses to disrupt the students' software development progress. These actions may appear mean and vindictive, and indeed have been labeled 'dirty tricks' in this paper, but their value has been appreciated by both the students and their employers. The feedback from the courses described has been mostly informal in nature but the universally favourable comments received give a strong indication that the courses achieved their aim of preparing the students for the workplace.

The experiences and learning provided by each of the twenty 'dirty tricks' has been listed along with their contribution towards teaching essential skills such as people

handling and planning. It must be concluded that other employers would benefit from their graduate and sandwich student recruits attending courses utilizing dirty tricks similar to those described in this paper. Whether such courses should be provided by the employers or beforehand by the universities is an arguable point. This paper has shown that such a course can be given at a university as part of a degree programme and that employers appreciate the value of such courses. However, this paper has also shown that the limitations at a university will restrict the number and nature of the 'dirty tricks' employed which reduces the experiences that students can be given. Therefore, even if all universities implement modules similar to that provided at Loughborough, companies would continue to benefit from organising their own equivalent courses [1].

If, as many employers believe, computer science and software engineering graduates are "very knowledgeable, but not a lot of use" then there is much to be gained by playing a few 'dirty tricks' both at university and during company induction programmes to introduce them to the realities of a real software engineering development.

Acknowledgements
The author would like to thank Ron Newsham and Roger Kerridge who kept the author informed of developments in the graduate training course at the Plessey company. The author would also like to thank Dr. Ian Newman and Mrs. Lesley Parks and other colleagues at Loughborough University who contributed ideas and assistance in the development of an equivalent course at the university.

References

1. Bach, J., SE education: we're on our own, IEEE Software, vol.14,6 (1997), 26-28
2. Burgess, W.P., Can quality survive amidst adversity? in Proc. of Software Quality Management VII (Amsterdam, April 1998), Springer, 204-208
3. Chapman, N., Fox, M., Keravnou, E., Lee, M., Levene, M., Long, D., Rounce, P., Samet, P. and Winder,R., 'Slick systems' and 'happy hackers': experience with group projects at UCL, Softw.Eng.J., vol. 8,3 (1993) 132-136
4. Dawson, R.J., Newsham, R.W. and Kerridge, R.S., Introducing New Software Engineering Graduates to the 'Real World' at the GPT Company, Softw.Eng.J., vol. 7,3 (1992) 171-176
5. Dawson, R.J., Newsham, R.W. and Fernley, B.W., 'Bringing the "Real World' of Software Engineering to University Undergraduate Courses, IEE Proc. in Software Engineering , 144,5-6 (1997) 287-290
6. Hilburn, T.B., Software Engineering Education: A Modest Proposal, IEEE Softw., vol.14,6 (1997),44-48
7. Jarke, M. and Pohl,K., Requirements engineering in 2001: (virtually) managing a changing reality, Softw.Eng.J., vol. 9,6 (1994), 257-266,
8. Leventhal, L.M., and Mynatt, B.T., Components of typical undergraduate software engineering courses: Results from a survey, IEEE Trans.Softw.Eng., vol. SE-13,11 (1987) 1193-1198
9. Mahmood, Z., Software engineering in the UK - Result of a pilot survey, in Proc. of INSPIRE IV, Training and Teaching for the Understanding of Software Quality (Heraklion, Crete, Sept. 1999), British Computer Society, 111-122
10. McCracken, W.M., SE education: what academia can do, IEEE Software, vol.14,6 (1997), 27-29
11. Milsom, F., Student Projects and Professionalism, in The Responsible Software Engineer, Chapter 32, 1996, Springer, London, 306-319
12. Saunders, B. and Georgiadou, E., Awareness and practice of information systems development methodologies in business today, in Proc. of Software Quality Management VIII (Southampton, UK, April 1999), British Computer Society, 133-144
13. Van Genuchten, M., Why is software late? An empirical study of reasons for delay in software development, IEEE Trans.Softw.Eng., vol. SE-17,6 (1991), 582-590