Extreme Software Engineering Project -- learning by doing

This chapter is a “how-to” guide for setting-up and running the course-long project. The project is the centerpiece of the course and everyone’s experience rises or falls with it. So it is very important to get good projects with willing, active clients. To keep those clients engaged, the instructor needs to provide them with productive teams of interested and equally active students.

**Soliciting projects**

As far as effort-to-payoff ratio, you can’t beat getting good projects with good clients. Expending a reasonable amount of effort before the course begins can make the entire semester much more successful. The opposite is also true. A poor project with a difficult or disinterested client can send ripples of complications and frustrations throughout all aspects of the course. No other single thing has a greater impact on the experience of the students. But even if the client proves problematic, the students will learn some valuable lessons, ones they are unfortunately sure to encounter again.

**What to look for in a good project**

Ideal projects are complex and involved and should address a real need. Projects should look like a lot of work. They should seem dauntingly difficult; after all, four or five students will be investing a course-worth of effort in the project. There should be lots of thorny issues evident at first glance, and other subtle ones emerging only as the development progresses. Real complexity is much more interesting and ornery than manufactured or contrived complexity. When thorniness stems from the needs of an actual client, there is added motivation of knowing that the difficulties are problem-dictated and not merely academic exercises.

Ideal projects are important, but not critical. The client should have a vested interest in the timely, successful completion of the system, but not absolutely depend on it. If the project isn’t of timely importance to the client, they may not give it sufficient priority or have a clear enough vision. Some client uncertainty is a fact of life for software development, but the client must understand enough about the needs of the project to answer specific questions and choose between options as the course unfolds. When the course is over, the project should not only be useful, but used. However, the project is foremost a learning experience not a professional software development effort. As such, only student pride and course grade can ride on the outcome. For example, it would be unwise to for an instructor to accept a project that if not fully implemented could negatively impact a major grant, detract from a tenure application or cause the university payroll system to crash.

Another important criterion is that the project should not be easily broken down into independent sub-projects. The team must work together in order to reach their goal of a single, large program, not a large collection of small programs. Systems that consist of
multiple programs only make reasonable projects if they are highly interrelated and developed as a cohesive system. An undesirable situation is one in which the team cleanly slices the work into individual chunks at the beginning of the course and can independently develop their pieces without any integration concerns.

The best projects are the ones that teams develop from scratch. When working with a blank canvas, students won’t have to struggle with leftover baggage from previous implementations. They can also start the XP process from the beginning instead of spending time learning the existing system and retrofitting unit tests to prior, non-XP projects. At the end of the course, students can have a fuller sense of accomplishment. Occasionally, a follow-on project can be acceptable. The best case of this is a complete re-implementation of the project in a new language or for a new platform. A partial re-implementation can possibly work if there are significant, self-contained enhancements to the system and the existing code that is kept can be treated as a black box.

A mixed re-implementation can work if done properly, one presenter at the 2001 XPUniverse conference uses a continuing project -- each semester the students take it a bit further. The benefits of this scenario include a more experienced customer and having the students gain realistic experience working with existing code. But there can be many pitfalls. Before accepting an enhancements-only project, make sure the existing code is solid and either well-documented or comes with an existing test suite from a previous XP development effort. The additions to the system are significant, and the interconnection between the existing system and the new features should be low and the cohesiveness of the collection of new features should be high. For example, if you have an existing, simple word processor and you want to add a drawing sub-system, that would be much better than wanting to add a collection of unrelated new features and upgrade existing ones. The real difficulties in this family of projects reside with the almost-complete, buggy systems that need a few additional features. The class is better off without a project like this. Students would spend inordinate amounts of time debugging code they didn’t write. And if you extreme program correctly, debugging is something you never need to do.

**Finding people who didn’t know they were looking for you**

Finding potential clients to provide projects for the development teams can be a win-win situation. The instructor has minimal-cost software developer team available for someone who is willing to make a modest time commitment. The potential client has a realistic problem in need of a software development solution (and most likely no budget with which to procure one). Together they can complement each other exactly – both parties get what they need.

Clients from an academic setting are also often motivated by a desire to support the student’s education. So even projects that fall well short of functional expectations can still meet a measure of success in that context.
Soliciting the projects is really a two-step process. First the instructor must word must get out at least a few weeks before the course begins. E-mail and word-of-mouth has worked well. After a few offerings of the course, the word-of-mouth approach can be even more effective. A few happy former clients can be the best advertising. Often potential clients can’t believe their good fortune when they hear of the opportunity. Once a candidate client is identified, the instructor asks them to submit a project proposal. The proposal consists of a brief description of the nature of the project, describing its basic purpose and any major needs or restrictions. These might include a specific hardware configuration that the system must run on, a list of other packages the system must interact with, formats of existing data or graphical interaction requirements, and whether the project is an augmentation of an existing system or a from-scratch project.

After getting the drafts of suggested projects, the instructor eliminates those that are too large or small, too critical or otherwise unsuitable. The instructor then interviews the clients who proposed the remaining projects, making sure they understand nature of the course, the expected level of client time commitment, and the types of activities the client will participate in. A typical XP client will need to formally meet with the team at least once every two weeks for about thirty minutes to and hour. They also need to be reasonably available either in person of through e-mail to answer questions as they come up.

Depending on how the instructor matches students to projects, the “green-light” projects may not be known before the course starts. The advantage to knowing ahead of time is that client training can be done more effectively. A good reason to wait when there are more projects than teams is to allow the students to consider all the potential projects and select their top choices. Students will naturally be more interested and motivated when they self-select their projects. If knowing which projects that will “go” ahead of time is critical, the instructor can make a judgment call, trimming the list to match the number of teams and then let the students choose from the smaller list.

Preparating the Team

Another important preparation for this course is that students know well ahead of time, by reputation, that this course requires lots of work, but also that it can be one of the most rewarding. Obviously this reputation has to be built over time, but the way the course is initially presented will have a lot of influence its long-term success. The higher level of autonomy challenges students to prove their capabilities. Many rise quite admirably to the occasion. The disappointments, when they do come, tend to be from better students who didn’t realize the level of work required by the course and wait too long to adjust.

After a few offerings of the course, word will get around and students will come to the course expecting a significant undertaking. In this setting, attitude counts for a lot so the more mentally prepared the students are, the better. In addition to giving the students enough of an XP foundation to build on, the rest of the preparations involve getting the supporting players in place. Clients need to be identified and trained. If possible, enlisting a coach will make the course go much more smoothly. If the class is large, consisting of four or more teams, then the coach becomes essential.
Training the client

Client training is discussed at length in chapter *User Stories – exploring with the customer* and *The Planning Game – negotiating the future*, but briefly stated here, the client needs to know enough about the extreme programming process to be an active, decision-making member of the development team. Repeat customers make this step easier; they also make an excellent resource for assisting first-timers. So there’s some added incentive for keeping all the clients happy, they may help with training the next time around.

Realistically, most clients will need training. The instructor or the coach explains the extreme programming process to the client at the same time the student developers are working through their XP spike. The primary goals of the training are to outline the client’s working relationship with the development team, explain what the client can expect in terms of quick deliveries and feedback, describe what the team will expect from the client in terms of decisions and accessibility, and finally to demonstrate the steps of the planning game from the client’s perspective. At this initial session, the instructor makes it clear that the client makes all the decisions regarding the characteristics of the project.

The client and the development team will be interacting often, over a long period of time. Getting their relationship off on the right foot requires that the instructor prepare the client for the first meeting with the team. In addition to explaining what the client can and should expect throughout the semester, the instructor should do a dry run of that first meeting. With the instructor playing the role of the development team, he or she and the client will go through the first few stages of the planning game, generating user stories and acceptance tests. Depending on how the training goes, the instructor, coach or client may want to initiate additional sessions to make sure that the clients understands their role and responsibilities.

By the time that the students have their first meeting with the client, they will also have been through the planning game process as well. Thus, for the real thing, both the client and the students will have some experience to draw on and can help the process along. The instructor should also make the client comfortable with the idea of reaching out for assistance if he or she should need it. The first source of explanation and clarification should always be the students, but the instructor and coach are also an available resource to the client.

Recruiting a coach

The coach serves as an extreme programming resource, a one-on-one XP tutor and sometimes cheering section. The coach pairs early with individual members of the teams, indoctrinating them to XP first hand. The coach should be available a lot as the teams do their XP spike and the early forays into their actual project. Initially, the coach helps the teams climb the XP learning curve through observation and participation. Typical activities include participating in the planning game, forcing the teams to test first, and guiding them through their initial stand up meetings. Later on in the semester the coach can usually provide additional support through a group e-mail account, a class
wiki or an occasional team meeting. Later in the semester the support generally comes in the form of technical advice, improving their estimation and bidding for tasks, and answering questions about some of the more advanced XP practices.

The obvious candidate for the position of XP coach is a previous instructor of the course. If no such person exists or that person is unavailable, the pool of potentials can be expanded in a number of ways. After the course has been taught at least once, former students who mastered the extreme programming techniques make potential candidates for coach. In some instances, technologically trained former clients might also serve as coaches. Ideally, the instructor should not also be the coach as they have different roles to play. Realistically, the course already requires a lot of outside manpower in the form of clients, so if necessary, the instructor can also coach. This situation is far preferable to having the instructor be both the boss and a client.

When coaching, the instructor needs to make it clear that during the coaching sessions, there is no evaluation of students going on. Another, more radical option for coach is an off-site, on-line coach. The first time the course is offered on a campus, there probably isn’t anyone with the requisite skills available. Find an XP practitioner interested in fostering a team of developing developers, after all one of the goals of extreme programming is to grow the XP community as quickly as possible. What better way than to act as an on-line coach for students being introduced to the methodology? If such an XP mentor is not locally available, consider a remote coach who could advise over the web.

**On being the “boss”**

In addition to teaching the course and setting up the simulated professional software development environment, the instructor also plays a role within the context – he or she is the “boss.” They are analogous to the owner of a small programming firm who has been contracted by the client to perform the work. In this role the boss can inquire of the team their project’s status, their ideas on approaching different problem -- computational or interpersonal, and their satisfaction level. The “boss” can also give the types of instructions to the students that a company owner can give the employees. Everything from pointing out ways to avoid potential problems, to pep talks, to reprimands for underachievement.

Likewise, the “employees” can expect things from their “boss”. They can look for help in resolving problems with a team member that cannot be handled by the team. They can also use the “boss” as a sounding board for technical issues, but not as the decision maker. Sometimes, if a project requires additional off-the-shelf software, the employees can make a “purchase request”. If circumstances warrant, the client may be asked to provide the necessary financial resources. The “boss” is also needed for resolving issues with the client. In both an academic and professional setting, the client (another professor or a paying customer) and the developers (students or employees) are not at the same organizational level. Within the XP project, they are supposed to be on the same team, and working towards the same goal, but if that’s not the case, the “boss” needs to step in. For example, if the client stops going to the delivery meetings, won’t make
project decisions or does not follow the planning game, the students have to go to the instructor for support.

Within the context of the course, the “boss”-“employee” relationship more or less mirrors an instructor-student relationship, so neither party requires a major adjustment. However, the autonomy afforded the students in this course does accentuate the differences enough to suit the simulated professional environment.

**Getting started**

The players are all in place for the academic simulation of the professional software development. Experience has shown that diving right into the semester long project results in the XP team being forced to do too many things at once. The practices that they can immediately understand and attempt (test-driven design and pair programming), are not the ones that should happen first (the ones that involve the client). Starting a long and interactive relationship with both parties unfamiliar with the XP process, very likely will lead to mutual frustration. Instead, the way to start the course is to give both parties some initial, independent training – the XP spike. When they do get together for their first meeting, they will both have a much better concept of XP and a little experience to rely on.

**How to run an XP spike**

The best way to learn extreme programming is by doing it. Feedback from XP students and clients strongly suggest that overlapping doing and learning result in a bumpy introduction. So this course is designed with a short “playpen” project at the beginning to give the students a quick feel for the process before entering into the actual project with the actual client. This Instructor’s resource includes a chapter *Presentation of the Course – Putting it All Together*, has suggested course outlines, the first several of which describe the content and activities for the XP spike.

The intent of the spike is not to build a system using XP, nor is it designed to present a complete picture of XP. Its purpose is to familiarize students with the practices and interactions of XP to the extent that they can begin a fuller and more comprehensive introduction. XP consists of a lot of very short cycles, in the course of one cycle students will have to perform the full complement of XP activities. The spike offers a quick, practical introduction to all of them.

The beginning of the course is dedicated to the spike. Everyone in the class works on the same mini-project, using it as a learning springboard. Completing the project is not important. The text includes full tutorials, including one for each of the first four lessons of the spike. The instructor can describe and demonstrate the practices in class, then send the students off to work the tutorials on their own, and finally have them form pairs to experiment with the practices.

For each practice covered, the instructor will have the inputs necessary for that practice prepared for use in the demonstration. By the end of the spike, the students will have used and developed all of the intermediate products used in an XP iteration. In some
cases, students will learn to use an intermediate product before they learn how to generate it. However, when the spike is over, all the practices will have been covered, forming a connected, end-to-end flow of an XP cycle.

Topics will not be introduced according to their “order of appearance” in a project timeline, but rather in an order that lets the practices build on one another. The first two are pair programming and test-driven design. These practices complement each other and students can learn to test first while programming with a partner. The next practices are simple design and refactoring. Watching the instructor refactor code that the students just wrote is a startling demonstration of the technique’s usefulness. Students are often amazed at how much their code can be improved.

The students need to understand their interactions with the client. It is the most important relationship in an XP effort. So, the next topics cover part of the planning game: decomposing user stories into tasks and estimating them. For these topics, the instructor or coach will portray the client in the classroom, developing the tasks that were used in first lesson on testing first and pairing. Outside the class, students will prioritize the tasks for each other, using the opportunity to see the XP process from the client’s perspective. Next covered are user stories to complete the planning game and a discussion of collective code ownership. The user stories feed directly into the task estimation, and collective code ownership gets the students thinking about how their group will manage this practice.

Continuous integration and acceptance testing flesh out the ideas of client interaction and code development. Acceptance testing closes the loop on the client’s participation in the process, and continuous integration makes collective code ownership both possible and necessary. The four remaining practices (small releases, metaphor, coding standards and sustainable pace) require less interactive participation and can be adequately covered in a lecture format. At the end of the first two weeks, the students will have had a whirlwind introduction to XP on a project that can now be abandoned. They are now ready to apply what they’ve learned to their actual project with their client.

**Setting up an XP development cycle template**

One of the most helpful tools that the instructor can provide the XP teams is a template for their iterations throughout the semester. In a semester course, a two-week iteration is the most appropriate. During the XP spike, the instructor needs to have all the teams plan their iteration templates. The template needs to include time slots allocated for the stand-up meeting, for client meetings, and for all pairing sessions. The template must cover a two-week iteration, but if possible, the weeks should look as much like each other as possible. That way, every Thursday right after lunch, Fiona and Keith always know that they are scheduled to pair program. The commitment to and consistency of a pairing session should be as important as a class. The amount of needless coordination and wasted planning time this avoids makes planning the template clearly worthwhile.

The first step in generating the template is having each member of each team create a free time schedule. If students are reluctant to provide enough times, remind them that they
need to spend a lot of time outside of class to justify the freedom and autonomy of the
course. If necessary, select a minimum number of hours that students must provide.
Have the students schedule the meeting with the client at a time when as many members
as possible can make it, reducing the chance that the meeting will be affected by
unexpected car problems or ineffectual alarm clocks. Schedule the group’s stand-up
meeting as soon after the client’s delivery as everyone can make it. Remember, this is a
short meeting, so it shouldn’t be too difficult to arrange. (Providing 15 minutes of class
time to assure the whole team is available has proved beneficial.) Once the beginning and
ending point of the team’s iteration has been defined, have the students schedule as many
pairing sessions as the constraints of the team members allow. Encourage them to
schedule three or more people for a pairing session. Multiple pairing sessions improve
the teams’ productivity, or, if there’s an odd man out, they can rotate or work in groups of
three. The other person can always do a non-pair activity such as running acceptance
tests or doing a spike. It also keeps the session alive if someone has to bail at the last
second.

If the team has any multiple pairing sessions, it will have to address version control
issues. If they are all pairing in the same room that will be easier than if two sets of pairs
are separated. In addition to setting up when and who will be pairing, it’s also a good
idea to consider where. If everyone knows this information, then unexpectedly free
teammates can drop in, and once they get used to their regular slots, accidental conflicts
are less likely. The other advantage of a fixed template schedule is that both the
instructor and the coach can also know when and where the pairings are and can also
drop-in occasionally.

Finally, on the actual piece of paper that contains the template schedule, have the students
put multiple contact methods for each member, whatever everyone’s comfortable with –
dorm room phone numbers, cell phone numbers, email, pagers, etc. If anyone can’t make
a scheduled pairing session, they can let everyone know so they can either try to salvage
it or at least not waste someone else’s time.

Accessing necessary software

This course can be taught with freely available software from the internet. For a basic
configuration, you’ll need to get the latest version of the Java language from Sun
Microsystems (java.sun.com) and a copy of JUnit from www.junit.org to provide
automated unit testing. Obviously, the class can be taught using different languages, and
there are many corresponding versions of the testing software in the xUnit family
available online. This text focuses on a Java implementation of the course, but
everything presented can be transliterated into other programming languages. Pointers to
these tools should be made available to the students as early as possible and the software
should be installed on the class hardware prior to the beginning of the semester.

Many IDE’s offer a free personal version or the option to purchase them at a discounted
educational price. (Jedit is an example open source version available at www.jedit.org,
Eclipse is another available at www.eclipse.org -- this is the my IDE of choice) While not
required for the course, a Java-centric IDE with built-in support for JUnit and more
refactoring tools as they become available, makes doing extreme programming that much easier. If a class IDE is used, and it is highly recommended, the students should also get it for their personal computing hardware. The instructor should make sure the software is installed prior to the semester and also provide links to the students to download or purchase this software.

In support of collective code ownership and continuous integration, every member of the group must have access to the most recent copy of the production code. The requirement for universal access means that everyone on the team is responsible for making the code available, especially if one person takes on the role of archivist or librarian. (In some groups, students have complained that one team member’s personal laptop has become the de facto archive site. This creates an effective barrier to collective ownership unless that student is religiously vigilant about uploading and downloading the system.) Additionally, the team needs an agreed upon protocol (or revision control software) to prevent ill-fated updates. Some simple solutions include: setting up a group e-mail account that allows uploading and downloading files, a group access hard drive on a local network, or identifying an ftp account that allows group access. This leaves open the possibility of clobbering code, but if the group is small and people are reasonable, a simple set of rules can avoid most problems. CVS is a typical revision control system that offers software support for maintaining consistent, yet dynamic code storage. Here too it is not impossible to overwrite code, it just takes more work to defeat the built-in safeguards.

The are other pieces of support software that can be put to use in this course and each instructor can decide for him or herself how much or little to use. Possible options include Ant (a system-building tool similar to the UNIX make tool (jakarta.apache.org)), and jFactor (a refactoring support tool that can plug-in to several different IDEs (www.instantiations.com/jfactor)). Ward Cunningham’s FIT acceptance test framework, (see the tutorial in chapter 14, Customer Written Tests – automating the acceptance process) is available at fit.c2.com. Currently there aren’t products that mechanically turn user stories into tasks, or unit tests into working code, and I dream of a hardware platform that allows multiple cursors controlled by separate mice for truly seamless paring, but there are always new things coming available to simplify the XP process – so keep checking.

**Providing a place to work**

XP developers have different equipment needs than typical programming classes. Simply using computers from those available in a general pool is not satisfactory. Because XP team members must program in pairs, they need more room and because they must talk during development, the computers cannot be where this would be prohibited or disturb others. Experience has shown that having multiple pairing locations is important because different combinations of pairs end up programming in different locations. Often it’s helpful if in addition to the provided equipment, students also make their own computers available. At a university this might mean scheduling a pairing session in the lounge of a team member’s dorm or bringing a laptop to an agreed upon location.
These equipment requirements are relatively easy to meet. But to get the full benefits of group or pairs working together, the team needs a place where at least two pairs can work together without interference from others. The benefit of group pairing is that when one pair is stuck on something, someone in the other pair may already know the solution. When they’re in the same room, progress continues without fanfare. Such a venue also allows for impromptu team meetings and shuffling of pairs. The work can continue as new team members arrive and depart.

One way to achieve this is to schedule a meeting room with the intent of having multiple laptops available. (Either provide them or ask participants to bring theirs.) This could also be accomplished by scheduling times when the XP group can have exclusive use of a small computer lab. A more permanent solution would be to create a separate XP lab with multiple machines. Such a room would have times scheduled exclusively for XP teams and only one team could use the room at a time. Even a set-up as simple as a room with just two computers either networked together or connected to the internet can provide enough infrastructure to increase the effectiveness of pairing.

**The Programming Project**

The project is the centerpiece of the hands-on software engineering course it is both the teaching experience and the product of the students’ effort. Unlike other computer science courses, there is no automatic reset button of a new assignment to give struggling students a chance to right themselves. Good, bad or ugly, the student’s have to work through whatever the project dishes out. If the upfront work of securing an appropriately complex project and a participatory client were done well, often the instructor can loosely hold the reigns for the rest of the course.

**Settling into a rhythm**

The XP team should strive for a short and steady development cycle, pumping out small releases to the client with the regularity of a heart monitor attached to a healthy patient. The pattern of the beat is spelled out in the iteration cycle template, but before the teams reach that goal state, there may be several trips through the ER. Even with the mini-project and the XP spike, the students will have a getting-acquainted period with XP. The instructor’s job, with much needed assistance from the coach, is to help get the team through that phase as quickly as possible. This can be accomplished through advanced preparation and small to moderate adjustments and corrections in real-time. Part II of this book, *The Iteration – Shaping the development process*, goes into detail of what is expected of the students during each iteration cycle. This section covers what the students and the client need from the instructor once the project begins.

The primary job of the instructor is that of teacher, preparing and presenting material in the classroom, and guiding and mentoring outside. The instructor also plays the important role of the “boss”, organizing and coordinating the projects and interfacing with the client. In addition, at the beginning of the semester, the instructor serves, in a lesser capacity, a similar purpose as the coach. They both need to pair with as many students as possible, giving a one-on-one test-driven design tutorial. They both also need to pay particular attention during this time for early signs of struggling. Stepping in
quickly can prevent more costly corrections after frustration has set in. Finally, in the classroom, during office hours and online, the instructor needs to be available as a resource to answer questions and offer advice on any subject that the teams encounter.

The client also needs to get into the cyclic rhythm of an XP development, and the instructor can assist again by answering questions and offering advice. The expectation is that the need here will be less, because the students will be able to fill in most of the gaps for the client. If the client is feeling uncomfortable with the process after the initial training, the instructor can attend and facilitate the initial meetings between the client and the team. After that, on an as needed basis, the instructor can continue to attend the meetings.

**Reporting project status**

The instructor can keep up with project progress in a number of different ways: listening to the “public” (in class) status reports, attending the team’s regular stand-up meetings where tasks are assigned and estimated, and reading the group e-mail accounts.

The instructor should schedule class time for groups to present a “state-of-the-project” address. This is an informal presentation, but more structured than impromptu conversations or e-mail discussions. The team gives a general overview of the current project status, perhaps including a demonstration of the current software. They should include what they are currently working on and briefly describe any concerns. The team also fields any questions from the boss or other members of the class. A side benefit of a public report is that all teams can gain from one team’s missteps as well as their successes. Although every group will face and resolve many problems during the semester, each team’s experiences will be different. Sharing them can only enhance the benefits gained in the course. The benefits take two forms: practical and theoretical. If another team encounters a problem before your team does, you can apply whatever solution they used, or avoid it if it turned out to be unhelpful. On the other hand, if a group has to deal with something that your team never comes across, then that will add to the sum of experiences that they take away from the course. As with everything else in XP, no specific individual is responsible for reporting the team’s status. The presenter or presenters must be determined by the group and may be different for each report.

Initially the instructor, the coach or both will attend the stand-up meetings to help make them happen. Early on the goal length of 15 minutes for these meetings will be a difficult to acheive. Completing the task allocation and planning for the upcoming cycle will require a concerted effort from all parties concerned. Once the meetings start to go more quickly, the instructor can and should take on a much more passive role, using the stand-up meeting as another form of status reporting. Finding time outside class when all team members and the instructor (and possibly the coach) can meet on a regular basis is a challenge. So, one solution is to schedule class time for the stand-up meetings. The instructor meets with each team in turn, while the other groups take advantage of their proximity. They might bring laptops to the class and spend the time pairing, or use the time to discuss and resolve existing obstacles.
The instructor should keep up-to-date on each team’s group e-mail account. Even if it is only a passive reading, there is no better way to maintain currency with the projects. (Note that it is not necessary to have the client on e-mail group – but he or she is a member of the team, so there’s nothing to hide from them. If they care to be included, add them to the distribution list.) If the team knows that the instructor is participating, they can post questions, and resolve issues without meeting face-to-face. Likewise, when the instructor sees something on the group that needs attention – an alternative option, a resolution to a disagreement or clarification of a misunderstood point – he or she can join right in. When a direct interaction with the team is necessary, since the instructor has been following the e-discussions, the conversation can start mid-stream. Obviously, if the course has an XP coach, the group e-mail is one of the main ways that he or she interacts with the teams.

Problems arise unpredictably

One unavoidable characteristic of a course-long project is that you cannot predict which software development problems will come up or when they will occur. With certainty, enough genuinely interesting issues will manifest each time the course is taught. However, there is no order in which to cover the material such that no problems arises before it has been discussed in class. There are some problems that are associated with specific stages of the course (resistance to testing first at the beginning, and project burnout towards the end), and those can be addressed in a time-appropriate manner. However, you can’t predict when a grossly incorrect estimation will throw a project off track or whether a turf war over a specific piece of code will erupt. To counteract this fact of software life, and turn a weakness into a strength, instructors can use these unexpected problems to drive some of the course content. By setting aside a portion of class time to address pressing issues, instructors can keep lectures pertinent and help navigate development obstacles. Students should inform their instructors when they are aware that the potential problems exist, but instructors must also be cognizant of the ones that students may not be aware of. By keeping up with the group e-mails or with frequent visits to the group wikis, instructors can tell whether or not the process is working. Often, either the coach or the instructor can immediately suggest a solution online and then all the groups can discuss the problem and proposed solution in the next class.

Delivery, installation and training

In offerings of the hands-on software engineering class using traditional software development methodologies, delivery of the final software occurred during the last or second-to-last week of classes. This left little time for training when there were unexpected problems with the installation. In addition, this would often be the first chance that the client had to really use the software, so while they were training, they were also evaluating the software and were sometimes frustrated that it was too late for modifications. Even when the installation went smoothly, the cracks of free time during the last weeks of a semester are not sufficient for client training. The delivery does include a user’s guide to help fill in the missing pieces, but it can’t provide the interactivity of a training session.
One of the nice aspects of XP is that the client doesn’t have to wait until the end of the project to get the software and begin to learn how to use it. The nature of XP is such that delivery, installation and training are ongoing, incremental activities and not just three more things to worry about squeezing in before the final deadline. In each iteration of the XP cycle, the team delivers new software to the client. In order to evaluate the new code and decide on the goals for the next iteration, the client must try out the software. Part of the evaluation is running the acceptance tests. As described in chapter 8 Acceptance Tests – Determining That a Story is Completed, each user story should have an acceptance test, sometimes written on the back of the user story index card. When the team makes a delivery and successful installation, the client should try out the acceptance tests for the current user stories as well as all previously completed user stories.

In a wonderfully effective way, this serves several purposes. It provides feedback to the software team, letting them know the status of their development. It gives the client an opportunity to reflect on the system and consider re-ordering his or her priorities. And, it serves as a partial training session for the client. The client who gets many small lessons spread out over time will have a much better understanding and ability than one who gets a few, intense sessions over a very short period of time.

As the project continues, the functionality of the new deliveries continue to increase, the current testing sessions reinforce the earlier ones. At some point, enough system functionality might exist that the client can actually apply the software in a limited way. This supplies more accurate feedback into story prioritization. At the end of the project, the developers make a final delivery that is incrementally about the same size as all the previous deliveries. Most of the software it contains has already been seen, tested and used by the client. The developers show the client how to use the new portions, just like every other delivery. Incremental releases of the software give both the client and the developers increased confidence levels in the system.

Finally, just as with traditional projects, XP teams include a user’s guide with the delivery of the system. The client, with many weeks of incremental training completed, can use the document as a reference manual instead of a tutorial.

The user’s guide
Doing a side-by-side comparison of past user’s guides produced for this course reveals an important lesson in planning ahead. For groups that used the traditional methodology, writing the user’s guide often conflicted with the final development push, one or both of these activities sometimes got short-changed. Some groups merely printed out their project’s help pages and bound them together. Other groups, worked on the user’s guide early in the project, worked on it as much as possible as the project progressed and towards the end of the course, dedicated one person to the task of completing the document. That user’s guide was well put-together. In addition to walking the user through the steps of using the software, the team tried to anticipate user issues and questions and answer them. They also included instructions on how to install or remove the software and a section on troubleshooting. In short, they tried to look at the system from the user’s point of view and write the user’s guide with that perspective in mind.
Writing the user’s guide for an XP group presents a different set of considerations. Obviously, creating the user’s guide is going to take resources away from the software development, but the client isn’t going to have a story about the user’s guide – it’s not code. To arrange for resources to write the document, the group could pick one member, select a sub-group or require everyone to dedicate some time to the project. For any development cycles that include writing, the team reduces the total number of units the client can use to select user stories. On the other hand, one could argue that the client should have the right to decide whether they want a user’s guide or additional system functionality. This suggests that there should be a user story about the document and it should be prioritized and bid on like any other one. In the software engineering course, a user’s guide is required, so the team needs to choose a strategy for allocating resources for writing it.

The purpose of the user’s guide is as a reference for installing and running the software. Good ones show how, starting with the installation CD, to make the system perform the operations in is intended to. They also include: a sequence of screen shots demonstrating the capabilities and options of the system, examples of all supporting files, and a list of system error messages and their causes or solutions. Exceptional ones have a tutorial on using the software, a section on troubleshooting the system, entry points for both novices and experienced users, and development notes for future maintenance programmers. In the academic environment where all the developers graduate in a year or two, the user’s guide becomes the client’s only resource on the system. If the client only uses the system infrequently, or needs to train another user, that resource is invaluable.

**Presenting the project**

The students, who have developed their projects in a professional manner, will have an opportunity at the end of the course to present them in an equally professional manner. In an undergraduate setting, students will dress in appropriate (interview quality) attire and have a chance to practice their public speaking. The instructor/boss, the coach and the clients should all be there, as well as any other interested faculty and students. One of the goals of the course is to give students a large responsibility and the associated authority. A public demonstration of the fruits of their labors is an appropriate finale. The next time they have to present to a client, it may not be the case that everyone in the room is pulling for them. For instructors who have followed the no-exams policy, the project presentations make a suitable substitute for the normally scheduled final exam. The teams should present their product as if it were a professional software release to a paying client.

The presentation is aimed at the clients, so it should focus on the larger issues of the original problem to be solved, the system’s functionality and interface, and possibly the algorithms used. Students should clearly state what portion of the system was completed and what was not. The presentation should definitely include a demonstration of the working code, formal delivery of the user’s guide (if not already delivered) and a question and answer session in which the client can address any of their remaining concerns. The instructor/boss, the coach and any other attendees may also ask questions.
or make comments. The presentation should *not* include any display of actual code, internal issues that the team dealt with or a description of the timeline of the project. The client is concerned with what they are being delivered and generally not the details of the possibly circuitous route it took to get there. The presentation is also not a forum for discussion of the XP methodology – for or against. Nor is it a place to air any grievances. The byword for the final presentation is “professionalism”; everyone should be gracious and polite.