Chapter 4

Starting an XP project

Summary: Meeting the client or customer. The first attempt at defining the scope of the project. Some techniques for requirements elicitation. Basic business analysis. Functional and non-functional requirements. Identifying dependencies and constraints. The structure of a traditional requirements document. An example of a real requirements document from a project.

1. Project beginnings.

It is the first class for the project and you have now got a client and a project. Initially, it all seems very daunting and many students are pessimistic about being able to build something that looks very complicated with a technology or method that is unfamiliar. You will almost certainly succeed if the precautions that I have indicated in earlier chapters are taken.[1] If there is a failure in a team it is because of individual failures or a breakdown in communication it is rarely due to the teams being technically or intellectually unable to cope. It does depend, of course, on the project scope being appropriate, neither too hard or to easy and this requires some judgement and experience on the part of the tutor.

We will assume that you are starting with a requirements which includes a reasonably simple initial phase and you should confirm with your client and/or tutor whether there is a part of the system which is clearly within your capabilities and which can be addressed first in order to gain confidence. In fact, the XP approach of building things in stages and getting them to work properly will soon build up your confidence.

The initial project description may be nothing more than a paragraph and it might seem to be too vague to allow you to start. Remember, however, that this description is just a starting point to you exploring, with the client, the client's business, its needs and possible solutions and so there will be a lot of preliminary work to do to define the scope of the project and what a potential solution might look like.

This is not an easy stage of any project and it is impossible to learn exactly how to do it in books and lectures, there is no substitute for trying it out and reflecting, as you go, on how the process proceeds.

We will look at the initial descriptions of two real projects that were done by students in 2001 and see how one of them progressed to a complete solution using XP.

Project 1. Quizmaster.

Background.

The brief was given to design a questionnaire-generating programme for an organisation that provides training for lawyers. Its main purpose is to allow trainees to answer weekly exercises for specific topics using a computer as an aid. This is in contrast to the approach used previously where exercises are handed out on paper, the student answers the questions and this is marked manually.

Project description: The proposed system should automate this task by removing the need for the lecturer to do any marking or for the students to mark their own work. The student instead answers the exercise on the computer and a mark is returned immediately. The system also monitors certain statistics on the student's performance on

1. If it is any consolation I have run such projects for a dozen years or so and it always seems to work out.

these tests so that lecturers can see how individual students are progressing and if they have been doing the exercises that they should have been doing. The system is for use on the Legal Practice Centre course that typically has 120 students for the duration of a one year long course consisting of two semesters. Each student takes four compulsory topics during Semester One and three chosen topics during the second semester. The chosen topics for the second semester are not known at the start of the year.

The system must allow for different topics and for topics to be changed, so it has to be possible to add and remove topics. Also, since the students on the course will change it will also be necessary for the system to add and remove students, especially at the end of the course when all students will need to be removed. The system must maintain details of topics and their exercises, students and the topics they take and the required statistics for how students have performed on certain topics. This information will change often so it must be easy to change any of the information stored by the system.

Project 2. WASTETEC.

Background.

The client develops solutions for waste problems facing small and medium sized businesses in and around South Yorkshire. They act as a broker between two companies, one with unwanted waste and another requiring waste. The client does not charge for their service; they are part-financed by the European Community Regional Development Fund.

Current manual procedures: If a customer has waste to sell, he/she contacts the client and provides the necessary details. The client then adds this to the existing database. If another company wants to buy some waste, they approach the client with their requirements, and the client searches the database for any matches. If there is a match found, the client puts the two companies in contact with each other.

Project description: The client requires a WWW based search, enabling potential customers wanting to buy waste to get direct access to the relevant details of the database, without contacting the client, and releasing any company details. The client also requires the system to enable potential customers with waste to sell, to advertise their waste. If a customer finds a match for his/her requirements, they should then be able to make enquiries to the client, on-line. The client then follows this up separately, and is no longer a concern of the system.

As mentioned before these were actual projects carried out by 2nd year computer science and software engineering students at the University of Sheffield during the 2nd semester of 2000/ 2001.[1]

2. The first meetings with the client.

This might be a session involving all of the teams working on that client's problem and generally involves the client giving a presentation about their business and what they are trying to achieve, their objectives for the system. There will usually be an opportunity to ask general questions relating to the system but these should not be technical computing questions - apart from general things such as the sort of network available or to be purchased for the solution. Remember that the client may know very little about Computer Science or programming, that's why they have come to you, you are the experts. Their expertise is in their business.

If you are not sharing your client with other teams then the first meeting will be a more informal one. It is important to prepare for it.

1.In both cases the students involved in the projects, there were 6 teams of about 5 for each project, competed to build the best solution for the client. For each client 3 teams used XP and the others used the "traditional" UML design-led approach described in most software engineering text books. It was an interesting experiment to see which approach did best, if there was a clear difference. As it happened the best systems, as decided by the clients, were the XP systems. Furthermore the students who used XP found that it was much less stressful and enjoyable than those who used the traditional method. Some of the reasons for this have been discussed before and we will return to a reflection of the XP process in the last chapter. Be prepared to engage with this reflection process, yourself.

Starting with the initial project description there are a number of things you should do:

Research into your client's business:

have they got a web page?
can you find any other published information about their business?
what do they sell - products or services or what?
what sort of clients do they typically have?
who are their competitors, what can you find out about them?

Preparing carefully for the first meeting will impress the client that you are professionals and will give them the confidence to proceed, don't forget that they are giving up some of their time and this is a cost to their business.

Turn up looking smart, on time and at the right place. Do not chew gum or do any other things that would distract the client's into doubting whether your are worth working with. These first impressions are important, you may think that they are trivial or superficial issues but it is part of the business expectation that the clients will have. In later life you will have to recognise these things, anyway, so it might as well be now!

3. Initial stages of building a requirements document.

Although we will be using stories as the basis for the software development it is important to have a clearly structured list of requirements, both functional and non-functional, so that an overall description of the complete target system is available.

This is developed in discussions with the client. At a suitable point we will develop a system metaphor and extract stories from this requirements and start developing the system.

The process of carrying out a full *requirements analysis* in a traditional software engineering context is often done at the start of the project and only occasionally revisited in any fundamental way later. In XP we will have a more continuous dialogue with the client and so the full requirements document will be a rather fluid document. We emphasise, here the construction of an initial one. It will contain, not only the functional requirements but also details of important *quality attributes* of the system.

Requirements analysis and specification is deceptively difficult since many clients don't know what they really want and they don't know what it costs or how long it will take to deliver. They often fail to recognise how hard it is to create a reliable system and how long it takes. Some might expect it to be done by next week!

Clients express problems naturally in their own words, words that might be unfamiliar to us or used in different ways, don't assume that your understanding of a particular word or term is the same as theirs. We need to identify what the terminology means and to agree on it. The construction of a *glossary* of business and technical terms should be an outcome of this dialogue.

When talking to clients realise that there may be hidden factors at stake: political, historical, geographical. You may need to understand these features of a business organisation in order to understand the reasons for particular requirements. Remember that there are probably more per-

sonnel involved in the business, who may have different requirements and different priorities, it is an important but delicate task to ascertain these.

The initial software requirements analysis can be divided into a number of activities:

Problem recognition;
Evaluation and synthesis;
Modelling and metaphor building;
Specification of user stories;
Review and discussion.

In the first week or two of the project, you should be evaluating and synthesising the problem and requirements information from the client. Always write down your thoughts, refer to these at your formal group meetings and put a date on them. Later, you may need to revisit some issue when you have forgotten the details. Although we wish to keep the paperwork to a minimum records of this stage should be saved, carefully.

You should already be modelling aspects of the client's business processes, in an attempt to clarify and make more specific your understanding of these processes. We will suggest a suitable way to help collect your thoughts together in the next Chapter.

By the next week, you should be refining some of your models so that they can be incorporated into the requirements document.

Problem evaluation involves:

defining all external observable relevant business objects;
evaluating the flow and content of relevant information in the business;
defining and elaborating all relevant software functions;
understanding relevant business behaviour (events);
understanding user behaviour (tasks);
establishing systems interface characteristics;
uncovering additional constraints.

All of these activities are difficult.

4. Techniques for requirements elicitation.

There are a number of useful approaches that can be used to elicit user requirements and to gain user involvement. Here are 6 approaches that can be useful:

Interviews;
Structured questionnaires;
Observation - again only successful, if you can do it unobtrusively;
Concurrent protocols - where a user describes his/her tasks whilst performing them;
Card sorting - useful if you want to understand the user's classification of his/her knowledge domain;
Carrying out a user role yourself;

Interviews have to be prepared carefully. In the first meeting, when you know little about the problem, then it is important to ask the client to describe all the key aspects of the system, try to guide them away from the desire to get to intricate detail about what they want when you simply do not understand what they are talking about. As you get immersed in their business context it is important to manage the meetings carefully. Identify what you want to know beforehand and prepare a set of questions that will help you to find out what you need. Once these questions are answered then you can explore further areas. It will often be the case that a question will stimulate the client into telling you some other piece of information, carefully record this. It is best to go to the meeting with all the team but make sure that there is a principal speaker and someone to record what is said. There is nothing more *off putting* for a client than to be faced with people asking questions from all angles on all sorts of disconnected topics. Plan your meeting carefully and try to stick to it. The same advice applies to any other stakeholder you meet, such as a user of the proposed system.

If you are not able to meet the client or the user then leaving a structured written *questionnaire* is another technique. Try to group related questions together. Also try to make your questions clear, unambiguous and relevant. Leave a contact number or email in case the person filling in the form has a query. Make sure that people know where to send the finished questionnaire and try to impress upon them, with tact, of course, that you need it by a specific date if the project is not to be held up.

Sometimes it is possible to visit the client and *observe* the business in action. Here you may be able to observe users in their current work. This is helpful in providing you with a context and a better idea of what the users are like, what they expect or are comfortable with and what sort of system you might be trying to emulate. Pay particular attention to the sort of user interfaces that seem popular. Take care not to disrupt their work too much. Some users are happy to talk their way through their tasks while you are there.

5. Putting your knowledge together.

Gathering all this information is one thing but putting it all together into a coherent model of the business is quite another. There are no simple solutions to this problem. Common sense is the best approach!

Defining all external observable relevant business objects.
We need to look at the sorts of things that are *coherent entities* in the part of the business we are considering. These could include: products, contracts, orders, invoices and such like. Make a proper list of them and try to distinguish between those that are involved with the external activities of the business, for example objects that are apparent to the customers, agents and suppliers of the business, and to those involved in the monitoring of the company such as taxation and other government authorities and the objects that are defined for the convenience of the internal management of the company, these might be: internal orders, memos and planning material, records and archives of company activity etc.

Evaluating the flow and content of relevant information in the business.
Each business process will involve a number of individual processes which take place in an organised way. What is the order in which this information is processed, what type of information is it? Try to get a general picture of what happens and when during typical scenarios of business activity. You will refer to the business objects described above, if you come across one that has

not been identified, previously, then it needs to be added to the list. Equally, if you found an object that doesn't seem to feature in any process that you are analysing, eliminate it. It may be that you find some difficulty in modelling things at the right level, there is always the temptation to try to describe things in too much detail. Try to avoid this at this stage. We are looking for a rather "broad brush" description of what is going on.

Defining and elaborating all relevant software functions.
Now we can start imagining what our software is going to do. It might be replacing some existing function, either a manual operation or in some obsolete software, or it might be a new feature that has not been implemented with software before. We will come back to this process in Chapter 6, at the moment it suffices to write it down as clearly and concisely as possible.

Understanding relevant business behaviour (events).
Now we have to try to figure out how these things actually relate to each other. We should try to define some common scenarios which explain the overall operation of the business processes through the medium of identifying the events that cause the scenario to operate. These could be the placing of an order by a customer, here we might need to identify what sort of customer is involved, a new or existing one, trade or retail. The business process involved for each of these may be different and so the system will be expected to behave differently as well. This leads to us identifying the different conditions that must apply for the different cases. Again we need to check that our business objects and processes described above are consistent with this.

Understanding user behaviour with task analysis.
There are many techniques for task analysis which can be used to elicit user requirements relatively easily. Task analysis tends to concentrate on the way users conduct business processes now. It may include user actions which do not involve interaction with a computer. Nevertheless, a task analysis model can form a useful representation for discussion with your users, helping to identify aspects of the task with which users are comfortable and familiar with and which could be incorporated into the structure of interaction with the required system. Alternatively, it can help identify aspects of the task which are currently problematic and could be improved in the required system.

As if requirements capture and analysis were not sufficiently complicated, we must often obtain the views of different users, who are likely to have different stakes in the outcome of the new system. Hence, you may need to identify and resolve stakeholder views. You should ask yourselves, who are your users? They are not necessarily a single, homogeneous group of people with the same tasks, the same goals or the same view of the world who are the clients?

In Checkland's Soft Systems Methodology, [Check90] a distinction is made between clients, who usually commission the system and stand to benefit from its outcomes, and actors.

Who are the actors? Actors are system users, who have to play a part in the system, but who may not directly benefit from it. How are you going to gain these stakeholders' involvement in and commitment to the development process?

At the end of this process you should have identified the dependencies that the solution needs to relate to within the business context as well as any basic assumptions that pertain. You may also have started to think about the constraints that will affect your solution, the available resources you have at your disposal, time, technology and so on. This needs to be clarified, it is no use trying to specify a system that you are not able to build!

The *functional* requirements should be stated, eventually, in a tabular form using simple English statements. These will be derived from the user stories as we will see in the next chapter. In fact, the functional requirements document is really just a summary of the story cards as they exist at the time. Where it is necessary to break a complex functional requirement down into a set of simpler ones, do so but try to preserve the connection between related requirements by grouping and numbering them together.

Looking at the Quizmaster system it is clear that one actor or user is the lecturer and they wish to be able to set tests. Now a test will consist of a number of questions and so the task of setting a test will involve the subtasks of setting a single question a number of times.

The requirement:

n. the lecturer can set a test

could then be restated as:

n.1 the lecturer can set a question
and
n.2 the lecturer can create a test from a number of questions.

If some of the requirements are poorly defined or subject to change identify them and put some measure on their risk of change, even if it is just a number 1...5 (low risk...high risk) which you allocate on the basis of your best guess given the knowledge you have available. We will find this useful, later.

We also classify each requirement as being:
mandatory - it must be present in the final solution;
desirable - it should be present if at all possible;
optional - only implemented if all others are done and there is still time.

Naturally the client will specify these levels and they may change during the course of the project.

We will use user story cards as the main mechanism for determing functional requirements.

6. Specifying and measuring the quality attributes of the system.

We talk about two main types of requirements: *functional* and *non-functional*. Put simply the functional requirement describes *what* the system has to do and the non-functional describes *how well* it is supposed to do it. This is a little simplistic but it will do for a start.

We often put most emphasis on the functional requirements and neglect the non-functional or assume that they are easily dealt with. In fact, identifying the non-functional requirements can be difficult. We need to define them carefully and what is more we need to set some sort of acceptability levels for them and a means of demonstrating compliance with these levels. It is possible to refine the notion of non-functional requirements into two categories: *quality attributes*, which determine how well the system should perform and *resource attributes*, which constrain

or limit the possible solutions to your business problem. Unless these are addressed a system may not be successful, even if all the functional requirements are met.

For most software systems some of these attributes will be **critical**, that is, unless each of those attributes achieves some required level then the system will probably not be successful, no matter how well it may meet its functional requirements or meet the goals for its other attributes. Thus it is essential to identify all the attributes, and to identify which ones are critical, and then to ensure that they are all met.

Identifying Attributes

The International Standards Organisation provides a taxonomy of quality attributes in its draft standard for software systems, (ISO 9126). [***REF***] As you read through the following list, based on that standard, make a note of those attributes which you feel could be critical to your project. The list is not exhaustive: you may see other classifications of qualities elsewhere and you may identify critical qualities for your system that do not appear here. Some of the issues that are discussed here may not be relevant to your own project. Think about them and focus on those that seem to be the most critical. Discuss this with your client.

***Functionality*:**
Suitability - the presence of an appropriate set of functions for specified tasks
Accuracy - the presence of correct and predictable results from specified input
Interoperability - the ability to interact with other specified systems
Compliance - the adherence to specified standards, laws and regulations
Security - the ability to prevent unauthorised access to programs and data

***Reliability*:**
Maturity - the frequency of faults/ rate of software failure
Fault tolerance - the continuity of software execution in the presence of faults
Recoverability - the ease with which performance and data can be recovered in the case of system failure

***Usability*:**
Understandability - the effort required by users to recognise application concepts and their applicability to user tasks
Learnability - the ease with which an application's functions can be learned
Operability - the effort required by users to operate and control the application

From the ISO 9241 standard for usability in software and hardware design a number of other issues are identified such as:

System efficiency:
Time behaviour - the adequacy of system response and performance times
Resource behaviour - the acceptability of amount (and duration) of resources consumed in performing system functions

***Maintainability*:**
Analysability - the ability to identify and diagnose deficiencies in the system
Changeability - the ability to modify the system, to add new functions, remove faults or adapt to environmental change

Stability - the risk of unexpected effects arising from modifications to the system
Testability - the ease with which correct system functioning can be verified

***Portability*:**
Adaptability - the opportunity afforded to adapt the system to different specified environments
Installability - the effort required to install the software in a specified environment
Replaceability - the effort required to use the software in place of other software in a specified environment

This list of attributes is much larger than you will require. Select the most appropriate and concentrate on these.

Specifying the acceptable level of an attribute.
Having identified critical quality attributes, you need to specify what level or measure of each attribute is acceptable in your system.You should identify at least:

the *worst* acceptable level
the *planned* level - be ambitious, but remain realistic!
the *best* level - just to provide a marker for what might be technically possible but infeasible for you.

It might be useful to identify the present level (if there is an existing system to evaluate). These levels should be specified and measurable in quantitative terms or metrics. It is not good enough to specify that your system will be "very" efficient, “easy to use” or "extremely" adaptable. You must attempt to define *operational*, *measurable* criteria against which your system can be judged. This will lead on to defining a set of tests that will establish whether the attribute has been delivered to the required level. We will look at testing in a later chapter but it will often be important to identify, at least in general terms, what the testing approach will be.

For example, if one of your usability criteria for your system is its suitability for the task,

a measure of its *effectiveness* is the *percentage of user goals achieved in a given time*;
a measure of its *efficiency* is the *time for a type of user to complete a set of tasks*;

A series of experiments (tests) could be organised in which users are asked to carry out some important tasks using the system, we would then be measuring how well these were carried out, how long it took, how many mistakes were made etc. These experiments should be repeated with as many people as possible in order to get a useful result. Alternatively, a measure of *satisfaction*, for example, can be gained on a *rating* scale, e.g. a scale of 1-5 by using suitable questionnaires distributed to a selection of users during a trial period of evaluation. If access to real users is not possible in the timescale you could use some of your friends, preferably those with a similar knowledge of computing as the intended users of the system.

For each numbered attribute we will specify a quality level and eventually a test for determining whether it is met in the final, delivered, software.

User characteristics and user interface characteristics
It is worth writing down a description of who the intended users of the systems are expected to be.

Some of the basic principles behind your design of the user interface should also be documented. This does not mean that you should design some specific interface options but some simple diagrams can help the client to visualise how the system might look.

7. The formal requirements document.

You must include the following information on any document you produce:

document type - e.g. requirements document
author(s)
version
date

The main body of the requirements document should have the following components:

Introduction and background;
Elementary business model;
User characteristics;
Functional requirements;
Non-functional requirements;
Dependencies and assumptions;
Constraints;
User interface characteristics;
Plan - a schedule of work with milestones, meetings and deliverables.
Glossary of terms (with an index)

The requirements will change as we progress, one of the key points about XP is that it attempts to address this. We will therefore regard the requirements document produced during this phase as an *initial* one which will change as the project unfurls but it is important that the client, as well as the team, have some idea where they are going.

The document should have clearly defined sections and paragraphs, referenced by number and listed on a contents page. This is vitally important for future cross-referencing between other system deliverables and the requirements specification. As Tom Gilb makes clear, thorough cross-checking is necessary for software reliability. It is also vital when we come to testing that each requirement has a suitable test or set of tests associated with it. This then demonstrates that we have met that particular requirement.

In the future, you will only be able to determine whether your designs, your test plan and test cases and your coding are complete and correct, by reference to the sections of your requirements document.

You should discuss with the client whether they could sign off the document as a gesture of good faith. It should be made clear, however, that, in the spirit of XP, the requirements are not totally fixed. Explain to the client how XP works, so that they will expect to work with you on the stories, will help to identify the priorities while you try to estimate the resource implications. Tell them that major changes, without good business reasons, will threaten the project. Some changes will be feasible, some not and it is important to remember that there will be a fixed time

limit to the project. It's better to have a good basic and useful system than an incomplete and useless one.

8. Exercise.

1. Read the requirements document in Appendix A. Criticise it, in particular:
   Are all the terms clear?
   Are the functional requirements consistent, unambiguous, repetitive?
   Are the non-functional requirements clearly defined, do they have suitable acceptance levels and procedures identified?
   How would you deal with any significant change in the requirements introduced by the client?
   Would it be easy to maintain?

References.

[Check90] Checkland, P. and Scholes, J. *Soft systems methodology in action*, Wiley, 1990.
[GILB88] GilbT. *Software project management*,
[ISO]
[White88] Whiteside et al, 1988.