1

The choices facing the system developer

Given the task of developing a software system, how *does* one go about it? To start the building of a system of a thousand or maybe a million delivered lines of source code is a daunting (if exciting) prospect. No one should begin without a very clear idea about how the development is to be undertaken and how the quality of its output is to be assessed.

Turning this around we can say that anyone undertaking software development, on no matter what scale, must be strongly advised to establish a methodology for that development – one or more techniques that, by careful integration and control, will bring order and direction to the production process.

To paraphrase Freeman 1982: *every* software development organisation already has *some* methodology for building software systems. However, while some software is developed according to modern practices of software development, most of it is still built in an *ad hoc* way. So it is best to talk about software development techniques and methodologies in terms of *changing* current practices, replacing them with new techniques that improve the process of software development and the quality of the resulting products.

There is of course a dilemma here for us as system developers: technique X may offer us potential gains such as reduced development time, reduced costs, increased reliability, quality etc, but in adopting it we also incur the risks arising from the fact that we may be unfamiliar with technique X, it might not be suitable for the system we have to develop, and it might not suit our staff and their skills. And yet we must resolve this dilemma if the efficiency of development is to improve and the quality of our products is to increase.

The questions that need always to be asked are: what techniques are there, which are appropriate to our problem, and how can we monitor the quality of the products? Just as the builder of aircraft is constantly looking for new materials and new production processes so we as builders of software systems should be constantly searching, constantly choosing and constantly refining.

This is in fact largely a book about choice. One of its central messages is: look at the different techniques you can apply to the development process (and here are some examples), make your choice for your own situation and make a resolute decision to support that choice.

A second message is: establish a management framework within which you can apply the techniques you choose in a controllable fashion. This is essential. You must be able to define how you will mark your progress against tangible milestones.

The third message is that management of *technical* development requires the planning and monitoring of more than just the traditional *managerial* indicators – budget and timescale. Technical indicators – particularly the various symptoms of complexity – need to be managed at all stages of development to ensure that the right techniques are being used in the right way.

So, carefully selected and appropriate techniques, an appropriate development framework and monitorable technical indicators are necessary to improve software development.

The introduction of new techniques is however not solely a technical matter. It has a sociological impact in that it requires software engineers to change their working practices and there may be a temptation to feel that the newer, more formal and more mechanised techniques will devalue the software engineer's skills. On this topic Michael Jackson says that 'there is still misunderstanding about the impact of development methods on skill. The development method, certainly if it is any good, must enlarge your skill, it must make you able to do things that you could not do before and instead of reducing skills on the job it increases it.' We believe that even a cursory glance through the following chapters will convince doubting software engineers that the scope for increasing their skills and abilities is vast.

Once they have used this book as a primary input to their decision on what techniques to use, we urge readers to move to the authoritative texts for the detailed input. An important feature of the succeeding chapters is therefore the bibliographies. These concentrate on literature that is in the public domain either in book form, in international journals or in the proceedings of major conferences. In some cases, material published by software companies is included where it can be obtained easily.

Each technique that we describe is given its own bibliography of both central and supporting material. Furthermore, at the end of the book you will find 'Other bibliographies'. Each of these contains references to literature on a subject area that does not have a bibliography of its own earlier in the book.

All references whether appearing in the text or in bibliographies are given in an abbreviated form such as Pode 1982. Where an item has more than one author only the name of the first appears in references. You can find the corresponding full bibliographic references in the 'References' section just before the Index.

At various points we have included exercises for the reader. These are not tests on the comprehension or memory nor do they have simple answers such as '42'. They are more intended to provoke thought about some key idea in a technique, again with the aim of illuminating what sort of problems different techniques set out to solve, whether your problem is related and so on.

Finally, some words on the structure of the book.

In chapter 2 we set out the managerial framework necessary for the remaining chapters: the Work Breakdown Structure which gives a framework of activities and deliverable items resulting from them, the estimation of resources to be allocated to the activities, the use of a variety of managerial and technical indicators of progress and quality, and finally the control of the products of development – 'configuration control'.

The remainder of the book is then built on this framework. First of all each of the development techniques that we describe is summarised in chapter 3. This is to allow you to scan the spectrum relatively quickly. Chapters 4 to 8 then take each phase of development in turn. Chapter 4 on Project Inception and chapter 8 on System Acceptance and Post-Acceptance Development bracket the three major chapters that are principally concerned with system development: System Definition, System Design and System Production. In those three chapters the framework of activities and deliverables is expanded to further detail, the techniques of use in that phase are given more detailed treatments, the indicators relevant to the products of the phase are described, some general words of wisdom are collected under the heading of memorabilia, and checklists for various topics appear.

Finally, chapter 9 ('Project Debriefing') returns us to the idea that the introduction of new techniques needs itself to be monitored if we are to find and make the best use of those that are the best for us. We need to record and learn from our own mistakes and successes.

In a similar sense we hope that our readers will treat this as an open-ended book, one to which they can add their own experiences with techniques, their own indicator data, memorabilia, checklists and so on.

2

The software development process

The transformation of your client's ideas about his proposed system into the physical, delivered system is a long one. If this transformation is to be at all manageable, it needs to be tackled as a sequence of stages, each bringing you closer to your destination.

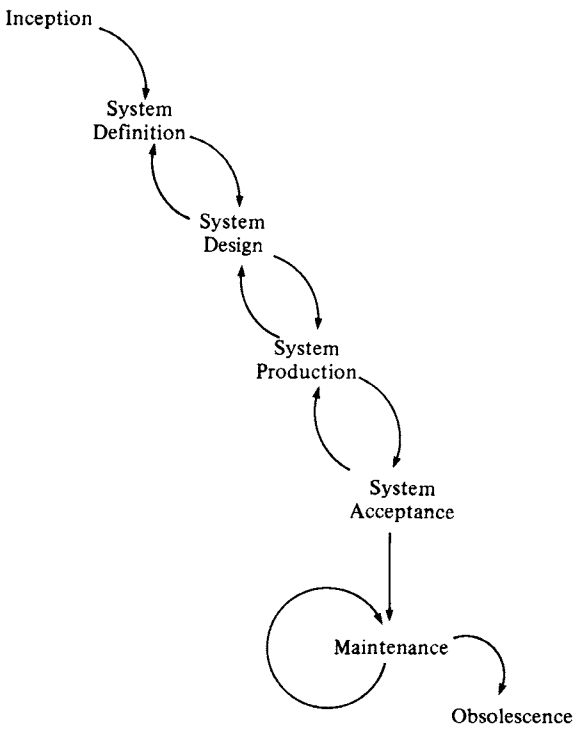
The simplest model of the software development process is a purely sequential one with each succeeding stage following on in serial fashion from the previous one. Such a model could serve for a small one-man project but a better approximation to reality is contained in the so-called 'cascade' or 'waterfall' model suggested in figure 2.1. This recognises that it is generally necessary – even desirable – for the stages to overlap and even repeat.

It is normally customary at this point to talk of the software development 'lifecycle' and to draw a diagram along the lines of figure 2.2. This custom is based on the premise that the original development of a system from scratch has the same underlying sequence of events as the enhancement or correction of a system: inception, definition, design, implementation and acceptance. Superficially this is true but the similarity is slight and the generalisation at times unhelpful.

In this book we prefer to talk about the 'development path' and 'maintenance cycles' and to draw the 'b-diagram' along the lines of figure 2.3. This reflects reality more closely. The development path, starting from nothing more than a desire for a system, stands alone. At its termination, the maintenance cycle, and possibly several simultaneously, take up the resulting system together with ideas for enhancements and in their turn produce new systems, but always as developments from the initial system

Since the evidence points strongly to the fact that maintenance consumes the greater part of software budgets we can deduce that we should expend the greater part of our effort on improving the efficiency of the maintenance activity. Our contention here is that efficiency is largely predetermined by the quality (in many senses) of the

Fig. 2.1. The cascade model of software development.



system being maintained, i.e. by the quality of the output from the development path. Given also that the initial development has to be successful in the first place for there even to be further maintenance we have placed the emphasis of this book on the development path alone so we remain with the simple cascade model of the development path.

Once we have established a simple model of the software development process we will have a basis for planning and managing. This will be necessary for two critical activities:

- estimating the task in hand – how it will be split up amongst staff, how long will be allowed for each activity and how the activities will be co-ordinated;
- monitoring the task, as it proceeds, against the original plans and estimates.

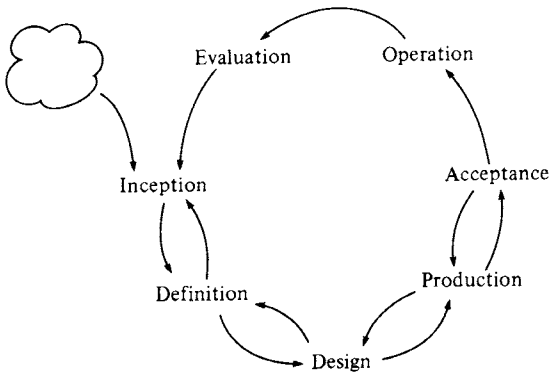
In this chapter we

- develop the concept of the phases and stages into which a software development project can be broken,
- look at the techniques for allocating effort, and
- describe the 'indicators' that can be monitored during a project to check its progress against plan.

These three items constitute the model of the software development process that underlies the remainder of the book. As you start off the development of your own system it is important to decide on the subset of the complete model that matches your situation. Establishing that subset is one of the most important management decisions you will make as it determines all of your planning and monitoring.

The model of the software development process is also the ideal background against which to see the great variety of development methodologies that are available. In chapter 3 we describe some of these individually in sufficient detail that you can decide which might be appropriate to your own project. Chapters 4 to 9 deal individually with the successive project phases discussing in further detail what each methodology means for that phase and for the stages within the phase. First then, let us investigate those phases and stages.

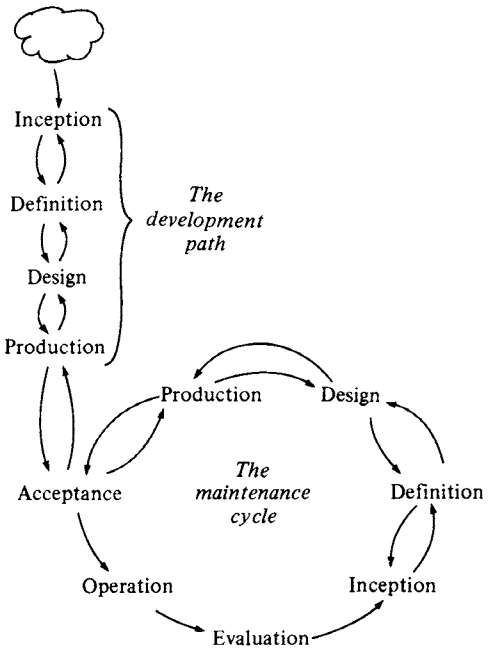
Fig. 2.2. The traditional software development lifecycle diagram.



2.1 Phases and stages

There is no single breakdown of the development path. The one presented here works in practice but you may wish to change it to fit your own purposes or preconceptions. The essential property of any good breakdown is, however, *completeness*. The aim is not to forget anything. The model must describe all the steps required in producing

Fig. 2.3. The b-model of software development and maintenance.



the deliverable items, typically software, hardware, manuals, training and support.

We must, therefore, be able to identify in our model (and in any real life plan based on it) the point at which we can deliver each of the items that we have contracted with our client to supply and to see a network of activities culminating in those deliverables.

The first level of breakdown that we have adopted is into these phases:

- project inception,
- system definition,
- system design,
- system production,
- system acceptance,
- post-acceptance development, and
- project debriefing.

These phases have a very natural serial progression:

- decide to do something for some reason,
- agree on what is to be done,
- work out how to do it,
- do it,
- have it accepted,
- look after it following delivery,
- look back at how it all went.

We shall see that the individual techniques available to improve your product and its production rarely span more than a couple of these phases and need preparation in earlier ones. So you will need to put together a coherent set of techniques which carry you through from inception to debriefing. It is the purpose of this book to help you choose that set and also to keep you reminded at each phase of its inherent dangers and pitfalls and how to avoid them by preparation.

We can hope that, in time, more comprehensive tools will become available that integrate the development process from definition to acceptance. Steps have been taken in this direction and some are described later in the book, but in the meantime we must rely on a synthesis of lesser tools.

2.1.1 Delineating the phases – deliverables

We said above that the phases have a 'natural serial progression'. This is true, but it is not a truth that we should feel bound to. For instance, it is a brave person who defines a system without being certain they can build it, or, in other words, without doing sufficient (albeit high-level) design to satisfy themselves that they are setting a soluble problem. There will be design done during definition; equally there will be some software production during design, and so on. In fact, taking a realistic viewpoint, we can consider this iteration between phases as being symptomatic of that very same foresight which makes projects flow smoothly, that ability to see early on in the project the groundwork that should be done now for the activities that come later.

There is, of course, a danger here, namely that of iterating so much between phases that the boundaries between them become blurred and indistinguishable. If this happens, two results are unavoidable. Firstly, progress becomes impossible to monitor and hence management becomes impossible. Secondly, development takes place on shifting foundations, and this is a situation that leads to errors, frustration and finally failure.

The solution to this is simple. Each phase (and – we shall see – each stage) must have a clear start and a clear end, and to give the start and end precise and material meaning we define them in terms of *deliverable items*. In general, these deliverable items will be documents. Often those documents will be agreed and signed by the parties to whom they will be important, as an indication of everyone's commitment to them as definitive statements. In the coming chapters we shall circumscribe each phase by defining the deliverable items that are its *inputs* – i.e. without which the phase cannot start – and those that are its *outputs* – i.e. those without whose completion the phase cannot be said to have finished.

These measures will allow you to tell when you have finished a stage or phase. When drawing up your plans for the project these deliverable items must be identified for each phase and stage. They are the subgoals on the way to producing the final goal – successful delivery of a working system. With this scheme of deliverables it also becomes possible for you to iterate between phases in a manageable way – an important ability if you are working in a new area with new problems and innovative techniques.

2.1.2 The phase profile

Figure 2.1 showed some phases overlapping in time. Provided we maintain the safeguards introduced with the notion of deliverables, this overlapping can be successfully handled. Indeed, it is often essential that this overlap or iteration take place for technical reasons.

With this in mind we go on to examine the profile in a little more detail – to take each phase apart and identify the stages that compose it. The stages in a phase form a mini-network of their own, sometimes overlapping and interdependent but finally coming together at the deliverable item(s) of the phase.

To some extent the different techniques that you can adopt will determine which stages are relevant. You should therefore regard the network that is built up in this chapter as a basic matrix on which you can plug the detailed sub-networks appropriate to the techniques you choose (where these exist). These sub-networks are given with the techniques in chapters 5–8.

2.1.3 The Project Inception Phase

This is the starting phase in the development of a system; the phase in which you conceive of the need for the system, justify that need and decide on system requirements. The deliverable item from this phase is a Requirement Specification for the system upon which a decision to proceed with system development has been based.

In reaching the point where a decision to proceed can sensibly be made you may need to carry out a great deal of analysis such as market research, cost–benefit analysis, operations research and tender analysis. All such activities are aimed at producing a Requirement Specification which defines a system that will meet your needs and which can be developed with the resources available.

2.1.4 The System Definition Phase

This phase has two purposes: to define *what* is to be produced and to define *how* it is to be produced. There are four deliverable items each associated with one of the following stages:

- Produce Functional Specification;
- Produce Project Plan;
- Produce Quality Management Plan;
- Produce Acceptance Test Specification.

The Functional Specification is the document that will be your bible for the remainder of the project. It describes what you are contracting to produce. The Project Plan and Quality Management (QM) Plan describe how you will produce it.

The Project Plan is your statement of what managerial steps you will take to ensure that the product is delivered on time and within budget. The Quality Management Plan is your statement of what technical and managerial steps you will take to ensure the quality of your product. We will not be telling you in this book exactly what should appear in these two plans as this is largely an organisational matter. You will find in this book useful references in the bibliographies plus checklists of items that should appear in the Plans and that are essential if you want to exploit the ideas and techniques in this book.

Your Quality Management Plan will give clear instructions to staff of what they are required to produce whilst the Project Plan tells them the resources they have been given to do the job – time, computers, budget etc.

As we shall see later on, a prime quality in the Functional Specification of a system is that it should be testable. Now is the time, therefore, to ensure this quality by writing down and agreeing with your client the strategy that he will use to test the system before accepting it. This Acceptance Test Specification and the Functional Specification will together be the inputs to a document – the Acceptance Test Description – that will be written during the Design and Production Phases and that will describe those Acceptance Tests in full detail.

We can add another stage to the phase:

- Perform System Modelling.

Now is the time during the project to carry out what we will call 'modelling' of the system. 'System model' is a generic term for any object that allows you to assess some quality of the system being built. As examples, you may wish to model the proposed user interface so that the future users of the system can assess its friendliness; stringent performance requirements may require you to simulate the system in some fashion to see whether

the proposed hardware and software will have the throughput/speed/reliability/resilience that is being written into the Functional Specification; you may need to do some preliminary high-level design to check the very feasibility of the system you are soon to agree to produce by signing the Functional Specification. In summary, you can see that the System Definition Phase also has System Models as deliverable items.

2.1.5 The System Design Phase

The central task of this phase is clear: to take the agreed Functional Specification and derive from it a design that will satisfy it. Precisely how this transformation is made will depend on the methodology you choose from the ones available. The process may be wholly manual or it may be automated to some extent, but the deliverable item remains the same – a System Design Specification. This is the seminal document that is the basis for your Production Phase. It may take any one of a number of forms – each methodology has its own bias and its own output documents. The essential points are that your Quality Management Plan should have defined what the System Design Specification should contain and what it should look like, and that the System Design Phase will not be complete until it is produced.

Whichever technique you adopt to generate the System Design Specification, its production will give you a far more detailed breakdown of the work to be done in subsequent phases than you have had so far. You should therefore take the opportunity to refine your Project Plans and possibly to review your Quality Management Plan in the same way. In addition to the Specify System Design stage in this phase we can, therefore, also identify two other stages:

- Refine Project Plan, and
- Refine QM Plan.

It may be appropriate to consider either re-running the System Models produced during System Definition or building new ones testing out features of the design. Typical examples are load and throughput models using mock-ups of the proposed program structure or overlaying mechanism, or benchmarks for the timing of critical computations. To cover these we introduce the stage Refine System Models.

2.1.6 The System Production Phase

It is a sad fact that this phase is often mistaken for the entire development project. It is certainly the largest phase – something like half of pre-acceptance development in terms of effort – but without adequately resourced System Definition and Design Phases preceding it, it is very likely to fail. It has one major deliverable item – a System Ready for Trials – and a number of minor but nevertheless important ones:

- User Documentation;
- User Training Schedules;
- Acceptance Test Description;
- System Conversion Schedule.

Your choice of software development procedures will determine the stages within the phase. Those stages are typically

- Design in Detail (Modules),
- Code Modules,
- Test Modules,
- Integrate Subsystems,
- Integrate System.

The Test Modules activity may then be broken down further, each subordinate activity relating to the modules for a particular subsystem, and may possibly contain activities covering the development of test software.

The last stage will deliver the System Ready for Trials. The other deliverable items will come from

- Prepare User Documentation,
- Prepare Training Schedules,
- Prepare Acceptance Test Description,
- Prepare System Conversion Schedule.

2.1.7 The System Acceptance Phase

This short phase is the last opportunity for detecting and removing errors in your system before it goes into full use or parallel running prior to cutover. Just like the earlier phases of a project, System Acceptance is receiving increasing attention with regard to thoroughness and rigour in the approaches made to it. Formalised and exhaustive

tests are becoming the norm, and it is for this reason that we have stressed the need for the early production of the Acceptance Test Specification (the strategic document) and the Acceptance Test Description (the detailed tactical document). These two are the principal inputs to the first stage of this phase, together with a System Ready for Trials out of the System Production Phase. The output deliverable item is the Accepted System.

The second stage is that of installing the system. This takes as its inputs

- the Accepted System,
- the System Conversion Schedule,
- the User Documentation,
- the Training Schedules,

and produces as its deliverable item the Running System. Exactly where the boundaries lie between the System Ready for Trials, the Accepted System and the Running System cannot be defined for the general case but it is important that you should identify the actual progression for your own system. The Install System activity can range from immediate full use to a prolonged process of parallel running, gradual cutover, pilot testing and so on – almost an extension of the acceptance process.

2.1.8 The Post-Acceptance Development Phase

It is now common knowledge that delivery and installation of the first version of a system often only mark the beginning of a long process of enhancement, bug-fixing and reworking that continues until the system becomes obsolete. We must therefore place emphasis on two areas:

- firstly, building longevity into the system right from the start through good design and implementation practices;
- secondly, managing the post-acceptance development in a way that preserves the integrity of the system, thereby delaying the decay that would otherwise set in.

The inputs to the phase arising internally from the project are:

- the Accepted System;
- the Functional Specification;
- the System Design Specification;
- all detailed design documentation;
- all source code;
- all test records;
- the full archive of working papers, discussion documents, and early versions of specifications, in other words, not just the system itself and all the final specifications but also all the background documents that record implicitly what design decisions were made and why. Ideally, these 'second-order facts' should appear in the primary documents, but, like polished mathematical theorems, the final specifications all too rarely show the route taken to them. This is unfortunate as the maintenance team's task of preserving the system's overall integrity will often need the second-order facts to appreciate just what form that integrity takes. Hence the importance of a definition and design archive.

Once underway, the phase also takes inputs from external sources and we can group them generically under three headings:

- enhancement requests,
- environment changes,
- error reports.

All of these will probably cause changes to the internal inputs in some way so that the outputs of the phase are clearly new versions of those internal inputs.

2.1.9 The Project Debriefing Phase

This is possibly the most neglected phase in the whole development process, and yet it has a great deal to offer. One of the main themes of this handbook – the use of indicators – is designed, amongst other things, to maximise the value of the debriefing exercise by ensuring that throughout development you are recording your experience both quantitatively and qualitatively. The aim is to crystallise that experience in a form that will make it of use both to yourself and to others in your organisation in subsequent software development projects. To this end, the phase has a single deliverable item – the Project Debriefing Report (PDR).

In addition to collecting indicator statistics throughout the project, you should aim to be

continuously recording all the successful *and* unsuccessful decisions and actions of the project. Once the system has been accepted you will have all the raw material necessary for a PDR.

A good PDR will act as a beacon for future projects and a collection of PDRs within an organisation will allow it to capitalise on its experience in software projects far more rigorously than by simply relying on the memories of those who took part in them. The PDRs can be salutary reminders for the experienced, handbooks for the novice and, if properly handled, the basis for a corporate policy of planning, estimating, quality management and technical practices. Do not pass up this opportunity – include a Project Debriefing Phase in your plan.

2.1.10 The Work Breakdown Structure

We have now dissected a project and broken it down into a network of co-operating activities. In doing so we have supplied ourselves with a structured breakdown of all the work that needs to be done during the project. This Work Breakdown Structure (WBS) can be written down in words with the indentation that software engineers rely on to show a hierarchy:

1000 Project Inception
  1100 Project Inception Management
2000 System Definition
  2100 System Definition Management
  2200 Produce Functional Specification
  2300 Produce Project Plan
  2400 Produce Quality Management Plan
  2500 Produce Acceptance Test Specification
  2600 Perform System Modelling
3000 System Design
  3100 System Design Management
  3200 Specify System Design
  3300 Refine Project Plan
  3400 Refine Quality Management Plan
  3500 Refine System Models
  3600 Prepare Acceptance Test Description
4000 System Production
  4100 System Production Management
  4200 Produce System for Trial
  4300 Prepare User Documentation
  4400 Prepare Training Schedules
  4500 Prepare Acceptance Test Description
  4600 Prepare System Conversion Schedule
  4700 Refine Project Plan
  4800 Refine Quality Management Plan
5000 System Acceptance
  5100 System Acceptance Management
  5200 Perform System Acceptance
  5300 Install System
6000 Post-Acceptance Development
  6100 PAD Management
  6200 Correct Bugs in System
  6300 Enhance System
  6400 Adjust System to Environmental Changes
7000 Project Debriefing
  7100 Write Project Debriefing Report
8000 Overall Project Management
  8100 General Management
  8200 Liaise with Client and Suppliers
  8300 Control Quality
  8400 Train New Staff

Figure 2.4 shows the basic dependencies between these activities and their relationship to the deliverable items defined earlier.

We have adopted a style of numbering scheme most commonly used for a WBS. It forms a convenient shorthand when planning and reporting – particularly if automated tools are used. Some of the estimating models described in section 2.2 contain algorithms for deciding what proportion of the total effort budget should be allocated to each phase and stage – in other words, how that budget should be mapped onto the WBS.

Once your project is under way you will want your staff to report how much time they are spending on the activities assigned to them and you will naturally want to have them do this against the WBS for comparison with the original targets.

Each activity in your WBS can be broken down to as low a level as you need. We call an activity at the lowest level a *work package*. It corresponds to a single area of work that can be allocated to one or more individuals, so you should expect to decompose the WBS until you end up with appropriately sized lumps.

You will see from the WBS that Project Management and Quality Management activities appear at phase level and at project level. It is up to you whether you have one or both of these. You should consider phase-level management on a large project with team leaders and group leaders reporting to a project manager. On a small project, the single manager may not be able to sensibly separate his management activities between

phases and hence a project-level management work package would suffice.

Whichever work packages you leave in or leave out, you should bear in mind that completeness is the most important property of a WBS. Projects frequently overrun their budgets and timescales for two reasons:

(i) they take longer to do the things that they expect to do;
(ii) they completely forget to plan for other things.

Careful compilation of the WBS is your way of avoiding the second trap. For instance, if you expect system integration to be a major activity requiring its own full-time integration leader then you will need to budget them into your WBS at the planning stage – probably as a management work package during System Production Phase.

One of the central recommendations of this book is that during the course of development you should collect data about certain 'indicators' of status and progress. In time you will begin to collect the statistics and indicator behaviour data from a number of projects and you will want to be able to contrast and compare them. This is now the time, therefore – before you start the next project – to decide on a *standard* WBS, one that is comprehensive, whose activities are well-defined by their deliverable items, in other words, a WBS that you can use consistently (either in whole or in part) on all projects in your organisation or even just on your own. The usefulness of the statistics you collect will be greatly enhanced and your understanding of the whole development process will deepen.

2.1.11 Prototyping

Prototyping of new products has been commonplace in the hardware engineering community for years. Hardware prototypes are produced to demonstrate quickly and at relatively low expense the viability and utility of a product and to find and solve problems in the product before it is cast into its final form.

Although the same arguments can be applied to software prototyping as for hardware, the practice of producing software prototypes is far less common. The reason for this is twofold. Firstly, there tends to be a belief that software prototyping is not cost-effective. Secondly, the tools for software prototyping are not as readily available or as obvious as for hardware prototyping.

The belief that software prototyping is not cost-effective is almost certainly misguided (e.g. Boehm 1975). Large numbers of projects that have run into very expensive difficulties could have saved themselves a lot of trouble by a relatively

Fig. 2.4. The network of activities and deliverable items.