UNIVERSITY OF LONDON | INTERNATIONAL PROGRAMMES

Software engineering project management

D. Murray and N. Sandford

CO3353

2013

Undergraduate study in
Computing and related programmes

This is an extract from a subject guide for an undergraduate course offered as part of the University of London International Programmes in Computing. Materials for these programmes are developed by academics at Goldsmiths.
For more information, see: www.londoninternational.ac.uk

Goldsmiths
UNIVERSITY OF LONDON

This guide was prepared for the University of London International Programmes by:

Dianne Murray

Neil Sandford

Putting People Before Computers, London, UK.

This is one of a series of subject guides published by the University. We regret that due to pressure of work the authors are unable to enter into any correspondence relating to, or arising from, the guide. If you have any comments on this subject guide, favourable or unfavourable, please use the form at the back of this guide.

University of London International Programmes
Publications Office
32 Russell Square
London WC1B 5DN
United Kingdom

www.londoninternational.ac.uk

Published by: University of London

Contents

Preface

About this course and subject guide

This is the subject guide for the Computing **CO3353 Software engineering project management** course. It introduces key concepts addressed by the Essential reading textbooks. For example, we give overviews of several testing techniques. However, the textbooks describe a larger number of techniques and explain them in more detail. The subject guide is intended to support and reinforce personal study, not to replace it. It will not be possible to pass this course by relying only on the subject guide. Do not cite this subject guide in your coursework or examination papers. Instead, go back to the original sources.

Software engineering project management focuses on techniques for managing software engineering (SE) projects, and builds on core software engineering concepts. A pass in Computing **CO2226 Software engineering, algorithm design and analysis**, is therefore a prerequisite, and you will also benefit from some programming experience, ideally as part of a team.

Course aims

This course provides an understanding of both theoretical and methodological issues involved in modern software engineering project management and focuses strongly on practical techniques.

The scale and complexity of the software systems now being developed demands that software engineers work in multi-functional teams and that they adopt scalable and robust methodologies and tools. As a student of business and creative computing, you need to develop the transferable skills in logical analysis, communication and project management necessary for working within team-based, professional environments.

You also need to extend your knowledge and understanding of the software development lifecycle and fundamental software engineering concepts (such as object-oriented programming, the software lifecycle, design for re-use and user-centred design). In addition to knowing and understanding the principles of traditional development methodologies, being able to understand and put to use contemporary approaches (such as '**Agile**' methods of software and project management) will help you to produce more robust processes and designs for delivering successful projects.

You should also learn to utilise that knowledge in a variety of contexts, ranging from embedded systems to the inherently parallel distributed environments of cloud computing.

Essential reading textbooks

Students will need to purchase **either** Ian Sommerville's book *Software engineering* (note: this includes advanced engineering topics beyond the scope of this course; such as reliability and security issues and the special demands of distributed and embedded systems) **or** Roger Pressman's book *Software engineering: a practitioner's approach* (with additional material for students wishing to reinforce their understanding of the fundamentals of software engineering and more than a quarter of the book dedicated to quality control techniques and testing). Both Essential reading textbooks have associated websites and additional online material which will be of benefit. These are the most recent editions of two long-standing texts.

Sommerville, I. *Software engineering*. (Harlow: Pearson Education, 2010) ninth edition [ISBN 9780137035151]; www.cs.st-andrews.ac.uk/~ifs/Books/SE9/, http://catalogue.pearsoned.co.uk/catalog/academic/product/0,1144,0137053460-NTE,00.html

Pressman, R.S. *Software engineering: a practitioner's approach*. (New York: McGraw Hill, 2010) seventh edition [ISBN 9780071267823 (pbk); 9780073375977 (hbk)]; www.rspa.com/sepa7e.html, http://highered.mcgraw-hill.com/sites/0073375977/information_center_view0; www.mhprofessional.com/product.php?isbn=0073523291

Recommended reading

This course is structured in accordance with the methodology known as the Rational Unified Process ('Rational' here being a reference to the company who developed it) and you are advised to consider acquiring Philippe Kruchten's book which extends the explanation of the Rational Unified Process (RUP), which is given in the Essential reading textbooks. There are descriptions of generic project management workflows (such as progress monitoring) as well as specific activities (such as supporting the software development environment) in Kruchten, which are missing from both Essential reading textbooks.

Kruchten, P. *The rational unified process – an introduction*. (Boston: Addison Wesley Professional, 2004) third edition [ISBN 9780321197702]; www.pearsonhighered.com/educator/product/Rational-Unified-Process-The-An-Introduction/9780321197702.page

Cornerstones of SE project management

To complete this course successfully, you will also need to understand and apply some key principles and address issues that are specifically relevant to project managers responsible for SE projects. Some management activities – and certain interactions and interdependencies between activities – are not addressed directly by the recommended textbooks. This subject guide highlights what we call the '**cornerstones**' of SE project management – the parts of a structure that hold everything else together. These cornerstones are listed in an appendix at the end of the guide.

CORNERSTONE 1

Project management is a creative activity. It requires a combination of knowledge, insights and skills in order to deliver the required outcomes for each unique project. If you do not take into account the specific challenges and rely instead on a single standard approach, you are not a project manager, you are a project administrator.

Further reading material

Other reading material is suggested in specific chapters, but the following are more broadly relevant. All textbooks are listed in the Bibliography in the Appendices section of this subject guide and in the *Computing Extended Booklist* for this course, which can be found on the virtual learning environment (VLE). Certain terminology used will be explained later in the subject guide or in the 'List of acronyms' at the end of the preface.

PMI, A guide to the Project Management Body of Knowledge. (*PMBOK® guide*) (Newtown Square, PA: Project Management Institute, 2013) fifth edition [ISBN 9781935589679]; www.pmi.org/en/PMBOK-Guide-and-Standards/Standards-Library-of-PMI-Global-Standards.aspx. This guidebook from a standards organisation contains comprehensive and well-structured descriptions of the relevant management concepts.

Highsmith, J. *Agile project management: creating innovative products*. (Boston: Addison Wesley Professional, 2010) second edition [ISBN 9780321658395]; www.pearsonhighered.com/educator/product/Agile-Project-Management-Creating-Innovative-Products/9780321658395.page This author analyses agility as an approach to project management and this is not just a guide to managing agile projects.

Cockburn, A. *Agile software development: the cooperative game*. (Boston: Addison Wesley Professional, 2006) second edition [ISBN 9780321482754]; www.pearsonhighered.com/educator/product/Agile-Software-Development-The-Cooperative-Game/9780321482754.page Highsmith's collaborator provides further insights into the philosophy and application of agile methods. Be warned that locating issues in Highsmith or Cockburn is sometimes made more difficult by a predilection for colloquial and polemical terminology such as 'argh-minutes' and '…agile teams don't need plans'.

Beyer, H. *User-centred agile methods. (Synthesis lectures on human-centred informatics)*. (Morgan & Claypool Publishers, 2010) [ISBN 9781608453726].

Lund, A. *User experience management: essential skills for leading effective UX teams*. (Morgan Kauffmann, 2011) [ISBN 9780123854964].

Additional resources

1. **Archive material** (papers, videos and adjunct proceedings) of SE-related conferences:

   The ACM SIGCHI User Interface and Software Technology (UIST) archive (http://uist.org/archive), contains material presented at the conferences from 1988–2013, as well as interviews with early UIST pioneers.

   The major SE conferences series have many online resources, along with the conference papers. Look for those such as: Requirements Engineering, International Conference on Software Engineering (ICSE), Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), IFIP – World Computer Congress.

2. **Case studies** (such as those presented by the Essential reading textbook authors):
   - Sommerville: www.cs.st-andrews.ac.uk/~ifs/Books/SE9/CaseStudies/index.html
   - Pressman and Lowe: www.SafeHomeAssured.com
3. **Video material** in repositories (of which there are many), such as:
   - Academic Earth: http://academicearth.org/subjects/computer-science
   - Google tech talks: http://research.google.com/video.html
   - VideoLectures.net: http://videolectures.net/Top/Computer%5FScience

Note: Unless otherwise stated, all websites in this subject guide were accessed in July 2013. We cannot guarantee, however, that they will stay current and you may need to perform an internet search to find the relevant pages.

Detailed reading references in this subject guide refer to the editions of the set textbooks listed above. New editions of one or more of these textbooks may have been published by the time you study this course. You can use a more recent edition of any of the books; use the detailed chapter and section headings and the index to identify relevant readings. Also check the VLE regularly for updated guidance on readings. Most authors provide links and 'Changes tables' or 'Correspondences' to help you migrate from earlier versions to the most recent updated versions of their textbooks. For the latest textbooks, web support is almost always available and, sometimes, there is an accompanying CD.

Diagrams

Note that colour versions of some of the diagrams in the subject guide are available in the electronic version; you may find them easier to read in this format.

Learning objectives

You should, at the end of the course, be able to draw upon appropriate techniques and have sufficient background knowledge to ensure good software engineering

management practices. You should gain the ability to select tools and methodologies that are fit for specific purposes, rather than encouraging a 'one size fits all' approach.

Learning outcomes for the subject guide

Having completed this subject guide, including the Essential reading and activities, you should be able to:

- demonstrate that you have acquired a structured view of the overall process of software development, with greater understanding of its main phases, specific notations and the specification and development tools employed
- show that you have developed a broader understanding of software engineering as a discipline, recognising its relationship and interactions with other computing and design disciplines
- demonstrate the ability to structure a problem into natural components and evaluate critically a piece of analysis and development work
- demonstrate your understanding of the implications of computer and network architectures for system-level design and development
- demonstrate that you understand how to identify, select and apply appropriate methods and tools for the development of solutions to specific real-world problems
- show that you can explain what it means to work effectively as part of a software development project team following a recognised development methodology, including the need to communicate effectively through written, oral and other forms of technical presentation.

Suggested study time

The Student Handbook states the following: 'To be able to gain the most benefit from the programme, and hence do well in the examinations, it is likely that you will have to spend at least 300 hours studying for each full course, though you are likely to benefit from spending up to twice this time'. Note that this subject is a half unit.

It is suggested that a chapter of the subject guide be read in detail, and immediate notes taken, each week. At the same time, the associated readings and web-based material should be reviewed. It is advisable to attempt the practice questions and to access relevant case study or video material when studying a particular topic. Some of these may be design exercises involving paper and pen prototyping, so sufficient time should be set aside, depending on how many are chosen. Revision should take place over a number of weeks before the examination, and practice examination questions undertaken on a timed basis.

Assessment

Important: The information and advice given are based on the examination structure used at the time this guide was written. Please note that subject guides may be used for several years. Because of this we strongly advise you to always check the current Regulations for relevant information about the examination. You should also carefully check the rubric/instructions on the paper you actually sit and follow those instructions.

The course is assessed by an unseen written examination consisting of three (out of a choice of five) questions. Guidance on how to prepare for examination questions is given in the next section.

There are also separate coursework assignment(s). Please note that the coursework assignments will **not** be the same as those that have previously been set for **CO3314 Software engineering management** and will change each year.

Coursework guidance

A detailed statement of what is required for your coursework assignments will be provided, with the marking scheme clearly indicated and suitable references given. Coursework assignments will be based on a prepared case study and will test the understanding and application of the techniques of software engineering project management covered in the syllabus. You should be able to structure a problem into natural components and to critically evaluate your own analysis and development work, presenting your material and arguments in an effective and informative manner.

The following guidance will help you to produce the appropriate standard of coursework:

- You should write in a report or essay format – not in note or bullet-point form – with a defined structure as detailed in the coursework assignment instructions. You do not need to restate the question asked or provide a table of contents, an index or a coversheet, or any extra information or appendices, and you should attempt to present your work both clearly and concisely. However, very short submissions are unacceptable.
- The structure, clarity and organisation of your work will be assessed and some marks may be awarded for it. Your submission must be well-presented in a coherent and logical fashion. It should be spell- and grammar-checked and you should structure it so that you have both a clear introduction and a conclusion. A concluding section is a required part of your coursework submission.
- You should include relevant diagrams, drawings or illustrations as graphics and screenshots. Note that these will have an impact on readability and presentation values.
- You must provide a References section at the end of your work, showing the books, articles and websites that you have referenced and consulted. All books cited, reports referred to and any material used (including all online resources) must be referenced. Students who do not provide references, or cite only the course textbook and subject guide, will be marked down.
- Websites should be referenced by the date of access and a complete and correct URL (generic site names are not acceptable). Other references should be in a standard format (namely, author names (correctly spelled, and with the correct last and first names or initials), year of publication, title, publisher, actual page numbers referenced). For a useful guide to referencing procedure, look, for example, at: http://education.exeter.ac.uk/dll/studyskills/harvard_referencing.htm There are also guidelines for referencing in the **CO3320 Project** guide. The submitted assignment must be your own work and must not duplicate another's work. Copying, plagiarism and unaccredited and wholesale reproduction of material from textbooks or from any online source are unacceptable. Any such work will be severely penalised or rejected. Any text that is not your own words and which is taken from any source must be placed in quotation marks and the source identified correctly in the References section. Failure to do so will incur serious penalties.
- Do try to be selective and critical in your choice of material and be extremely careful about the validity of information on internet sites and web sources. Citing and copying from Wikipedia and other encyclopaedic sources, for example, is not appropriate for a coursework submission. Note that personal or company-sponsored blogs, social networking material (such as Twitter or Facebook) or comments on any other type of posted transitory material cannot be used as references for academic work. Be aware that many information sites are really commercial advertising, or simply reproduce unvalidated and unchecked material from elsewhere.

- Do check the date of all material and do not use out of date sites, sites which list student work or projects, references from commercial publishers to journal paper abstracts only, or those which are simply personal opinions, blogs or comments.

Examination guidance

Failure to take account of the following points means that questions will not be answered as well as they should be.

Take a few minutes to make a plan of each question and to gather your thoughts instead of immediately starting to write. Organise your time in the examination to allow sufficient time to read over what you have written and to make any necessary corrections.

Ensure that you fully understand the topic area of the question (which will be related to the course outline). Check that you can answer every part and subsection of the question. It is generally better to answer three questions as fully as you can, rather than, for example, some in excessive detail and others very briefly.

Ensure that you understand what type of answer is expected. Read the question carefully and answer in the way that is requested: the wording will indicate the expected kind of answer and level of detail.

Ensure that the level of detail of the answer you give corresponds to the marks for that part. Try to achieve the balance reflected in the marks indicated. Do not spend excessive time and effort on a subsection of the question that is worth only a very small percentage of the overall marks. Similarly, do not write cryptic notes or single points for a part of the question that is worth a significant percentage of the available marks.

Ensure that you do provide diagrams or examples where requested since this is part of the marking scheme for that question. All figures and diagrams should be clearly labelled and described. Use tables and lists where appropriate – for example, in a question which asks you to contrast two approaches or itemise the differences between two techniques.

Do not waste time by:

- providing unnecessary diagrams where this is not required
- restating the question in your own words or repeating the question text
- answering one question in great detail at the expense of others
- repeating details from one part of the question in another part.

List of acronyms

Below is a list of acronyms which are relevant to this subject. You may wish to add to this list as you study.

ATAM	Architectural Trade-off Analysis Method
AUP	Agile Unified Process
CASE	Computer-aided Software Engineering
CMMI	Capability Maturity Model Integration
COCOMO	COnstructive COst MOdel
COTS	Commercial Off The Shelf (software) components
CSR	Critical Software Reviews
CUPRIMDSO	Capability, Usability, Performance, Reliability, Install-ability, Maintainability, Documentation, Serviceability and Overall
DFD	Data Flow Diagram

DSDM®	Dynamic Systems Development Method
ERP	Enterprise Resource Planning
FPA	Function Point Analysis
FSAR	Final Software Acceptance Review
FURPS	Functionality, Usability, Reliability, Performance, Supportability
GQM	Goals, Questions, Metrics
IDE	Integrated Design Environment
IPM	Integrated Project Management (CMMI Level 3)
ISO/IEC	International Standards Organisation/International Electrotechnical Commission
ITIL	Information Technology Infrastructure Library
LASCAD	London Ambulance Service Computer-Aided Despatch system
LOC	Lines of Code
MVC	Model, View, Controller architecture
OGC	Office of Government Commerce (UK Government)
OO	Object Oriented
PDCA	'Plan Do Check Act'
PERT	Program Evaluation and Review Technique
PFD	Product Flow Diagram
PID	Project Initiation Documentation
PMBOK®	Project Management Body of Knowledge
PMC	Project Monitoring and Control (CMMI Level 3)
PMI	Project Management Institute
PP	Project Planning (CMMI Level 3)
PRINCE2	PRojects IN Controlled Environments
PSR	Preliminary Software Review
QA	Quality Assurance
QM	Quality Management
QMS	Quality Management Strategy
QPM	Quantitative Project Management (CMMI Level 3)
RBS	Risk Breakdown Structure
REQM	Requirements Management (CMMI Level 3)
RMMM	Risk Mitigation, Monitoring and Management
RSKM	Risk Management (CMMI Level 3)
RUP	Rational Unified Process
SAM	Supplier Agreement Management (CMMI Level 3)
SEI	Software Engineering Institute (Carnegie Mellon University, USA).
SOA	Service-Oriented Architecture
SRS	Software Requirements Specifications
SWOT	Strengths, Weaknesses, Opportunities and Threats

UML	Unified Modeling Language
UP	Unified Process
V & V	Verification and Validation
WBS	Work Breakdown Structure
XP	eXtreme Programming

Introduction: the need for software engineering

References cited testing

Cook, S. 'What the lessons learned from large, complex, technical projects tell us about the art of Systems Engineering', *Proceedings of INCOSE 2000 Annual Symposium, Minneapolis*, pp.723–30.

Fagan, M.E. 'Design and code inspections to reduce errors in program development', *IBM Systems Journal* 15(3) 1970, pp.182–211. (Available from IBM under Reprint Order No. G321-5433.)

Hosier, W.A. 'Pitfalls and safeguards in real-time digital systems with emphasis on programming', *IRE Transactions on Engineering Management* EM-8(2) 1961, pp.99–115.

Liskov, B. and S. Zilles 'Programming with abstract data types', *Proceedings of ACM SIGPLAN Symposium*, 1974, pp.50–59.

Naur, P. and B. Randell (eds) 'Software engineering. Report on a conference held in Garmisch, Oct. 1968, sponsored by NATO'.

Parnas, D.L. 'Abstract types defined as classes of variables', *ACM SIGPLAN Notices II* 2 1976, pp.149–54.

Royce, W.W. 'Managing the development of large software systems: concepts and techniques', *Proceedings of the IEEE* 1970.

Wirth, N. (2008) 'A brief history of software engineering'; www.inf.ethz.ch/personal/wirth/Articles/Miscellaneous/IEEE-Annals.pdf

> 'It is unfortunate that people dealing with computers often have little interest in the history of their subject. As a result, many concepts and ideas are propagated and advertised as being new, which existed decades ago, perhaps under a different terminology.' (Wirth, 2008)

Overview

The term 'Software Engineering' (SE) comes from recognition that formal methods and more sophisticated tools and techniques were needed to address the challenge of '…the construction of systems by large groups of people' (Naur and Randell, 1968), specifically real-time systems.

These 'frequently present unusually strict requirements for speed, capacity, reliability and compatibility, together with the need for a carefully designed stored program' (Hosier, 1961). These features, particularly the last, have implications that are not always foreseen by management.

Stephen Cook treats software engineering as a subset of the broader discipline of systems engineering, and sets the tone for this course when he advises the reader that:

> 'Systems engineering should not be considered merely a set of procedures to be followed. Rather, it is a … way of thinking that encompasses a set of competencies and a set of processes that can be invoked as needed, each of which can be achieved through a range of methods. An important aspect of systems engineering is the selection and tailoring of the processes to suit each project.' (Cook, 2000)

This reflects the need to balance between sometimes incompatible criteria such as functionality and reliability. Martin Rinard wrote:

> 'This combination is challenging because of the inherent tension between these two properties. Increasing the functionality increases the complexity, which in turn

> increases the difficulty of ensuring that the software executes reliably without errors.'
> (See: http://people.csail.mit.edu/rinard/paper/oopsla03.acceptability.pdf)

Software engineering covers the entire product lifecycle, from definition of requirements, through the design and development phase, all-important testing and delivery of long-term stakeholder benefits.

Two essential weaknesses of the early 'top-down' methodologies were the difficulty in obtaining complete, correct and appropriate specifications of what the user actually needs (and here we are talking both of 'business' users and the end-users who are typically employees or customers of the 'business') and the significant effort in development and testing that is inevitable if bespoke solutions are created as part of every project.

In most areas of engineering, a top-down approach reflects the predictable nature of the interactions between modules or sub-systems, whereas many of the interactions between software modules are largely unpredictable and only emerge during final integration. This has led to adoption of a range of iterative development processes that recognise the importance of user-centred design and prototyping as a way of configuring and deploying validated and robust components as part of an overall system.

In his commentary on standards for software lifecycle processes, Raghu Singh of the Federal Aviation Administration recognises that the *ISO/IEC 12207* development activities (see: www.iso.org/iso/catalogue_detail?csnumber=43447):

> '…may be iterated and overlapped, or may be recursed to offset any implied or default Waterfall sequence. All the tasks in an activity need not be completed in the first or any given iteration, but these tasks should have been completed as the final iteration comes to an end. These activities and tasks may be used to construct one or more developmental models (such as, the Waterfall, incremental, evolutionary, the Spiral, or other, or a combination of these) for a project or an organisation.'
> (See: www.abelia.com/docs/12207cpt.pdf)

The critical success factor is the way in which specialists performing the roles defined in the standard are integrated into a team operating within a common framework so that their activities are manageable. SE project management is a distinct specialism within the team and only the smallest of projects will not need a recognised team leader or project manager.

Unsurprisingly, a 2007 survey into project management practices (See: www.pwcprojects.co/Documentos/BRO Boosting Business Performance DC-07-1104-A v8[1].pdf) found that:

- Use of project management methodologies is widespread. Organisations that do not have a project management methodology reported lower-performing projects.
- Use of project management software positively impacts project performance.
- Project management certification and project performance are clearly connected.
- Projects with poor management are more likely to suffer from problems like bad estimates/missed deadlines, scope changes and insufficient resources – the top three reasons for project failure cited by 50 per cent of those interviewed.

CORNERSTONE 2

Project management is about controlling the strengths and weaknesses of the project team (and enterprise) and the opportunities and threats that can arise and lead to issues for the project, the 'internal' and 'external' factors of a SWOT analysis.

Software that is being produced today typically falls into one of three categories:

1. Software as a **component**, such as the fuel management system embedded in your car.

2. Software as a **tool**, such as an '**I**ntegrated **D**evelopment **E**nvironment' (**IDE**) in which other software is developed and maintained.
3. Software as a **process**, no longer simply deployed as a tool to help organisations do the things they do, but a result of rethinking what an organisation does and how it does it.

The significance of the last of these is that the organisation may be gambling its future in a game where it does not understand the rules, let alone have prior experience. This may be the case whether the project is seen as being technology-driven (and managed by someone with little or no experience of the human-resource issues involved) or organisation development-driven (and managed by someone without SE experience).

In his book *The change equation*, Peter Duschinsky argues that projects fail when '… the complexity of the project exceeds the capability of the organisation to cope with the changes needed' (See: http://wdn.ipublishcentral.net/management_books_2000/viewinside/207741229298377), where that capability necessarily includes the ability to manage both social/cultural factors and technical ones. In the subsequent paper with the same title (see: www.irma-international.org/viewtitle/43568) he delivers the message that only a quarter of these change projects succeed fully, while another quarter fail to deliver any benefit whatsoever.

The SE manager has to achieve a balance between the reliability of the end result and its general acceptability for the client (compliance with functional and non-functional requirements), its ease of maintenance and its efficient use of resources. There are three major sources of risk in software development which make the job more difficult:

- Projects are getting larger and more complex and demand more sophisticated tools and methodologies, especially in the context of business change programs.
- Development teams are also getting larger. Roles and responsibilities, especially those requiring people-oriented 'soft skills', need to be well-defined and understood.
- There are several software development lifecycle models and there is no 'one size fits all' solution.

The subject guide and the Rational Unified Process (RUP)

You will see that this subject guide does not simply follow the structure of the Essential reading textbooks. Rather, we have divided it into four parts, each corresponding to one of the phases of IBM's '**R**ational **U**nified **P**rocess' which is goal-driven, not activity-driven or plan-driven. It views each project phase as the journey from one milestone state to another. Each part consists of chapters addressing the key software engineering concepts that have to be applied in order to reach the required exit condition for that phase and each chapter includes discussion of associated project management issues.

Highsmith talks of a framework of project governance consisting of the **Concept phase, Expansion phase**, **Extension phase** and **Deployment phase**. Progression between phases is controlled by milestones called 'phase gates' which segment the phases and, importantly, segment the operational agile cycles of envision, explore and deploy. Cockburn says in his book that 'RUP is not incompatible with the principles' of agile methods. Indeed, there is an 'Agile Unified Process' (AUP) developed by Scott Ambler. (See: www.ambysoft.com/unifiedprocess/agileUP.html)

There is a clear relationship between Highsmith's governance framework and the RUP. RUP and AUP are sufficiently similar for us to be able to refer to '**U**nified **P**rocess' **(UP)** phases, UP milestones and UP disciplines compatible with agile and more traditional approaches.

Software engineering activities and workflows typically span several UP phases and are addressed in the subject guide at the most relevant time. For example, the 'mission' for testing is established in the inception phase but that mission is only achieved through

execution of component and system testing in later phases. Discussion of testing is therefore deferred until the chapter on the **Construction phase**.

Each chapter is grouped as part of the four '**phases**', except for the final chapter, which contains case studies.

Part 1: Inception phase – scoping and justifying a project

By the end of the first part, you should understand the software development process, project planning and the information needed to enable a project to be described in terms of scope and the approach to satisfying requirements.

- **Chapter 1** reviews the fundamental elements of a software development process and the way in which a project manager delivers project results. It also introduces the key features of the Unified Process.
- **Chapter 2** reviews the role of the business case and system requirements as the basis for the project plan.
- **Chapter 3** reviews techniques for project cost estimation and scheduling, with specific reference to planning within iterative and agile approaches.

Part 2: Elaboration phase – defining a response to stakeholder needs

By the end of this part, you should understand how to produce a robust implementation plan based on a stable architecture and appropriate approaches to risk and quality management.

- **Chapter 4** reviews approaches to risk identification and mitigation.
- **Chapter 5** reviews techniques for evaluating architectural options and design and modelling of system architecture.
- **Chapter 6** reviews the project manager's responsibilities for detailed planning, people and risk management and the project development environment, including configuration management.

Part 3: Construction phase – managing the provision of the functionality agreed

By the end of this part, you should understand what is involved in delivering a version of a stable system that can be deployed with real users.

- **Chapter 7** looks at the implementation workflow, including the use of design patterns, design for re-use and management of agile teams.
- **Chapter 8** reviews approaches to Quality Management (QM) and testing.

Part 4: Transition phase – producing a product release

By the end of this part, you should understand how to manage entry into the post-release maintenance cycle and authorise the start of a new development cycle if required.

- **Chapter 9** addresses deployment issues, including evolution, process improvement and management of expectations and change.

Part 5: Review and revision

The final chapter provides a look back on key lessons learned in the form of case studies (**Chapter 10**). There are various appendices to aid revision, in the form of revision topics, and the full list of cornerstones, as well as a Sample examination paper and an outline marking schema (Appendix 6).

Each chapter starts with a set of **Learning outcomes**, detailing the learning objectives for that topic and a brief Overview section. Also shown at the beginning of each chapter are details of the **Essential reading** required for full comprehension of the topic – normally a chapter in the designated textbooks – and any **Additional** or **Further reading**. These additional references have been chosen to enhance knowledge and expand coverage and

are usually books, journal articles or free-access websites. The **Bibliography** (Appendix 1) has the full citable details (and, where available, the online link to the publisher's website page for that book) of all the textbooks and print sources referred to in the subject guide.

There are also **References cited** sections where appropriate. These are usually citation details for quotations or for particular articles or websites referred to. These additional links and citations can be followed up, if wished, but it is not mandatory: they do not necessarily appear in the Bibliography.

Each chapter concludes with a summary of the main points as an *aide-memoir*. **Reminder of learning outcomes** and **Test your knowledge and understanding** sections are given at the end of each chapter to help with examination revision and to test understanding of the concepts covered in that chapter. Local tutors in institutions may also wish to use such suggested topics to lead discussion groups; therefore sample answers are not always provided.

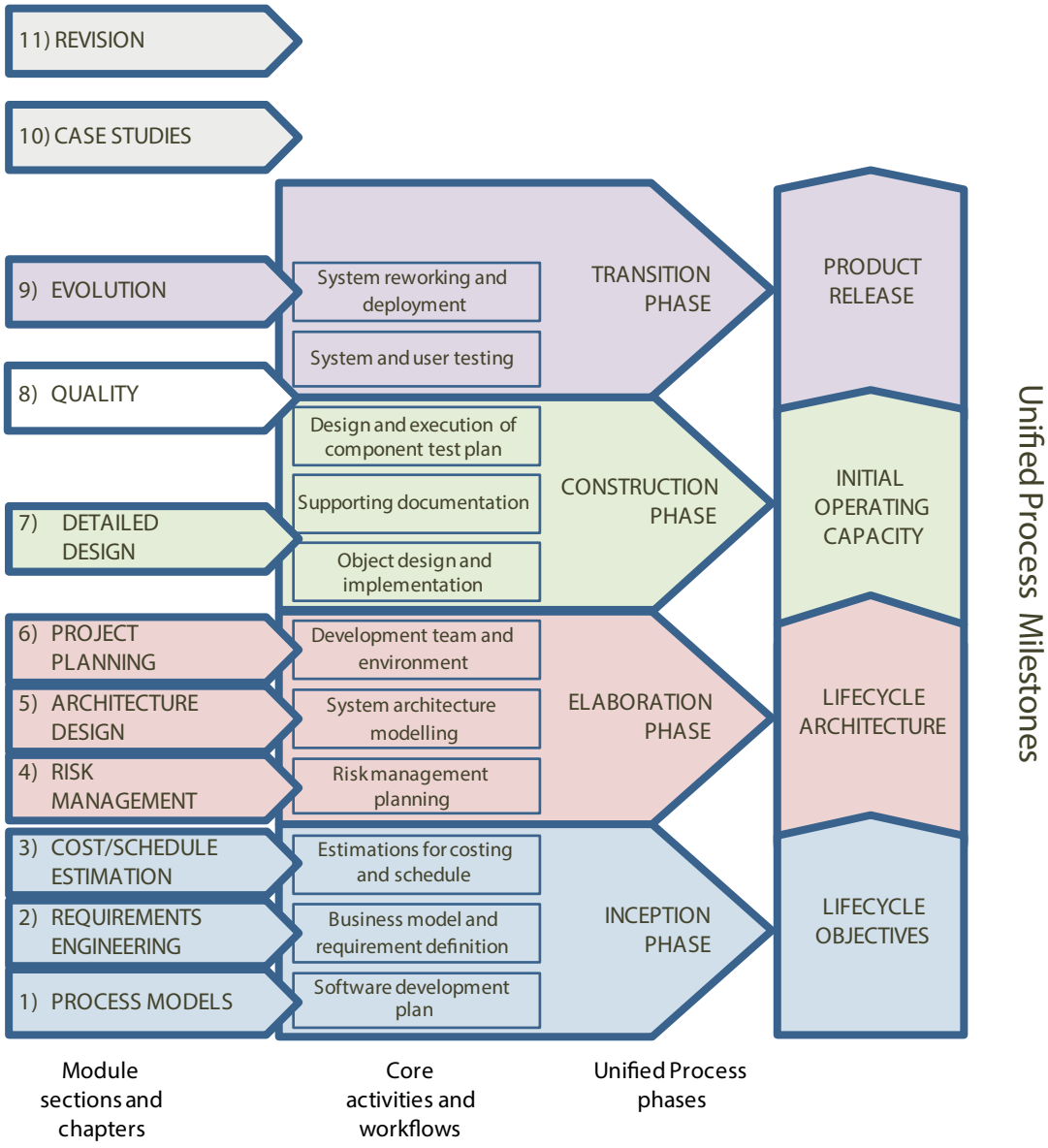


Figure 0.1: Architecture of the SE Project Management module and its relationship to the RUP.

A **Sample examination paper**, together with an outline marking scheme, can be found at the end of the subject guide.

A **List of acronyms**, expanding all the acronyms used in the subject guide, can be found in the Preface to the subject guide.

Summary of Introduction: the need for software engineering

- Software engineering is a profession, an engineering discipline.
- Software engineering covers the entire development process and applies to development of all kinds of software system although tools and techniques are rarely universal.

Test your knowledge and understanding: seven important references

Investigate the following milestones leading to today's software engineering environment.

1. The concept of abstraction developed in the mid-1970s, allowing the details of the implementation of a 'module' to be hidden behind a published 'interface' (Liskoy and Zilles, 1974; Parnas, 1976) leading to object-oriented system design with components (objects) that have unique properties and behaviours.
2. Recognition that the engineering lifecycle concept also applied to software engineering, which led in the mid-1970s to widespread use of the waterfall model (see: Royce, 1970), formal software inspection methods for each step (Fagan, 1976), comparable to those used to validate electrical circuit diagrams, architectural scale plans and the like. The lifecycle approach was formalised as a standard in 1995. (See: ISO/IEC 12207, Information Technology – Software life cycle processes (see www.iso.org/iso/catalogue_detail?csnumber=43447)
3. Identification in 1995 of the major factors that cause software projects to fail and key ingredients that can reduce project failure, in The Standish Group's first 'CHAOS report'. (See: www.spinroot.com/spin/Doc/course/Standish_Survey.htm)
4. The publication of the 'Manifesto for Agile Software Development' (see: http://agilemanifesto.org/principles.html) by the Agile Alliance in 2001.
5. The UK government's green paper on the development of 'e-government' services (HM Treasury, 2003) (see: www.publicnet.co.uk/abstracts/2003/10/20/measuring-the-expected-benefits-of-e-government), which differentiated between four types of 'service re-engineering' from a simple 'electronic customer interface' to 'end-to-end-process transformation'.
6. Growing recognition of strategic priorities that reach beyond 'on time and on budget' such as stakeholder satisfaction, delivery of benefits, quality of the result and return on investment (see: www.pwcprojects.co/Documentos/BRO Boosting Business Performance DC-07-1104-A v8[1].pdf), reported in 2007.
7. Widespread ignorance that information sent via cloud email could be stored by a service provider in another country with 38 per cent of those surveyed in 2011 sending valuable or confidential personal information via cloud email or messaging services (61 per cent of 18–24 year-olds, compared to 24 per cent of the 55+ age group). (See: www.rackspace.co.uk/uploads/involve/user_all/generation_cloud.pdf)

Part 1: Inception phase

Chapter 1: Software processes

Learning outcomes

By the end of this chapter, and having completed the Essential reading and activities, you should be able to:

- understand the concept of a process model as a way of describing a software development project and differentiate between common approaches such as waterfall, iterative and agile
- identify the key artefacts produced during an iteration of the software development process
- understand the role of the project manager in delivering those artefacts
- understand the ways in which the development process can affect quality.

Essential reading

Sommerville	Pressman
Chapter 2: Software processes	Chapter 2: Process models

Further reading

Ambler, S. 'A Manager's Introduction to the Rational Unified Process' (2005); www.ambysoft.com/unifiedprocess/rupIntroduction.html

Boehm, B.W. 'A spiral model of software development and enhancement', *IEEE Computer*, 21(5) 1988, pp.61–72.

Royce, W.W. 'Managing the development of large software systems: concepts and techniques', *Proceedings of IEEE WESTCON* (Los Angeles, 1970), pp.1–9; http://leadinganswers.typepad.com/leading_answers/files/original_waterfall_paper_winston_royce.pdf

Scacchi, W. 'Process models in software engineering', in J.J. Marciniak (ed.) *Encyclopedia of software engineering*. (New York: John Wiley and Sons, Inc., 2001) second edition Volume 2 [ISBN 9780471210078].

References cited

Booch, G., J. Rumbaugh and I. Jacobson *Unified modeling language user guide*. (New York: Addison Wesley Professional, 2005) second edition [ISBN 9780321267979].

Highsmith J. *Agile project management*. (Boston: Addison Wesley, 2009) second edition [ISBN 9780321658395].

PMI, *A guide to the project management body of knowledge. (PMBOK® guide)* (Newtown Square, PA: Project Management Institute, 2013) fifth edition [ISBN 9781935589679]; www.pmi.org/en/PMBOK-Guide-and-Standards/Standards-Library-of-PMI-Global-Standards.aspx

Sutherland, J. 'Agile development: lessons learned from the first Scrum' (2004–10); www.scrumalliance.org/resources/35

Overview

The development of a bespoke software system can be thought of as a process with three 'actors':

- A **system user**, whose requirements determine the intended scope and behaviour of the system.
- A **team of system engineers** who translate that specification into a design for the system.
- The **system** created from that design.

From the customer's perspective, the project team should be able to take the stated needs and implement and maintain the system through until the end of its useful life. In an ideal world, the process continues smoothly, perhaps through several iterations (versions) over time.

However, as all software developers know to their cost, customer needs change and the real world is rarely ideal. This leads to mismatches between what was intended and what actually occurred, which impacts on one or more of the three actors. Errors introduced and not trapped locally can have costly repercussions and steps need to be taken to limit the impact.

An error in the software engineer's understanding of requirements will affect the ability to address the system user's needs and result in reworking of the 'specify' activity.

A mistake in the creation of the system designed to meet those needs will result in reworking of the 'design' activity.

A failure on the part of the client to plan and support the deployment of the system will impact on the performance of the system and its users and reworking of the 'deploy' phase.

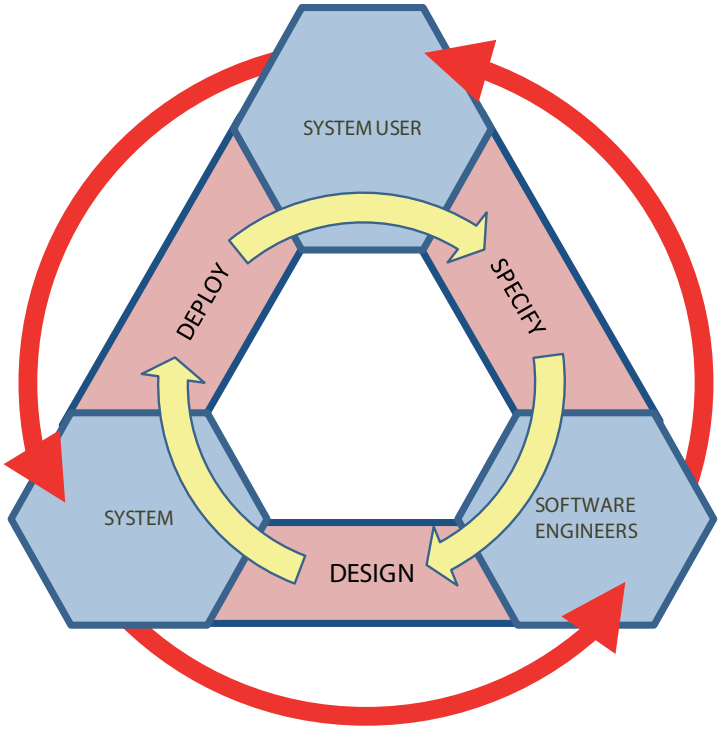


Figure 1.1: Failure in the development process.

Process modelling

Both recommended textbooks review and compare the most common types of software development process models in use today. There are useful process models that rely on iteration, such as incremental development, prototyping and incremental delivery. Process models can be activity-driven (such as the simple waterfall), plan-driven (the German 'V-Modell' and Boehm's 'Spiral' are both predicated on the existence of an overall project plan) or goal-driven (such as the UP's milestones, or the targets that drive the SCRUM technique (see Sutherland, 2004–10)).

CORNERSTONE 3

A process model is a high-level and incomplete description of the way a project is controlled and how it evolves. It introduces interfaces between actors and workflows and transitions between activities. Your choice of model does not fundamentally affect **what** you are going to do; customers still need to know that their requirements are the reference point for design and code still needs to be tested before and after integration. What the model allows you and your team to do is focus on the important transitions during the project.

Key concepts: phases and iteration, milestones and artefacts

Phases

The classical linear process model is the waterfall (Royce, 1970) shown in Figure 1.2. It begins with requirements which are the basis for specification. The expectation is that each activity in the process is visited once and its output is fit for purpose.

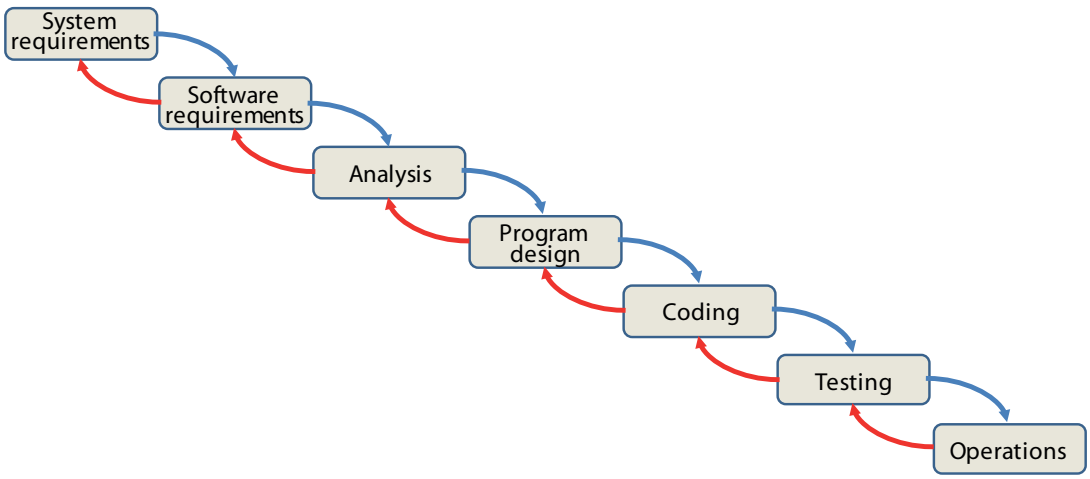


Figure 1.2: The classical waterfall model.

Figure 1.3 takes the waterfall testing phase and sub-divides it in accordance with the V-Modell approach to verification and validation. Localised testing is efficient but errors that remain undetected until acceptance testing still require reworking of everything that is a response to requirements. Quality issues are addressed in Chapter 8 of the subject guide.

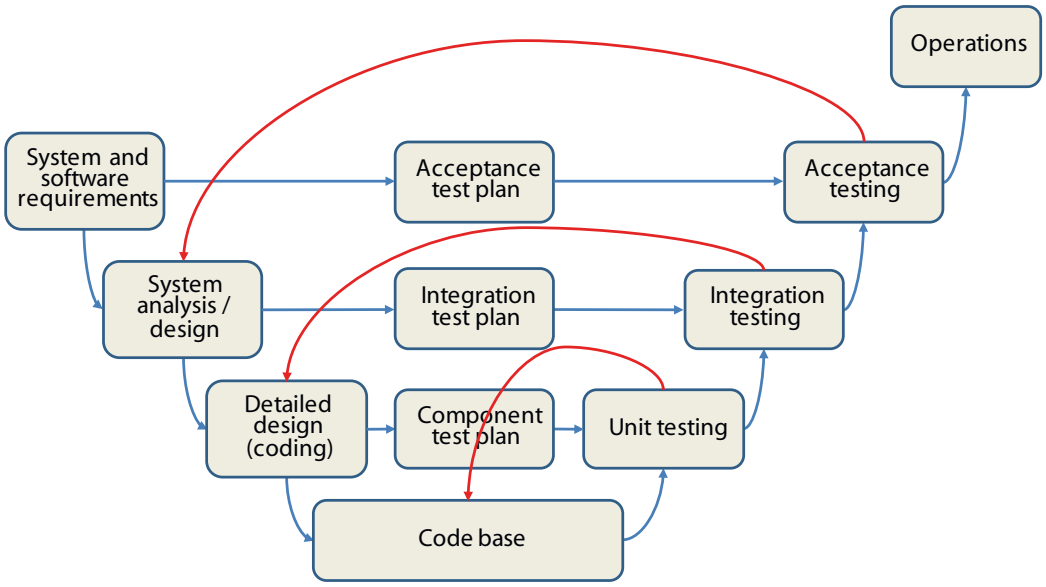


Figure 1.3: Waterfall model modified to V-Modell format.

Milestones

Royce's 1970 paper is best known for its description of the linear process we call the classical waterfall. Royce also proposed formal review-points or milestones and test-plans which can be drawn up once the final software review has validated the program design. He suggested a '**P**reliminary **S**oftware **R**eview' (**PSR**) between supplier and customer before formally moving on to the analysis phase, based on a preliminary program design which acts as a form of prototype to guide design and coding. As work develops, '**C**ritical **S**oftware **R**eviews' (**CSR**s) can be scheduled with the customer, to further inform the coding process. Finally, the transition from testing to operational deployment is controlled by a '**F**inal **S**oftware **A**cceptance **R**eview' (**FSAR**) between the testing and operations phases.

The goal is to minimise the wasted work that results from re-iteration of previous phases. The PSR, CSR and FSAR are not simply reviews of what has happened already, they prevent problems being fed forward into the next phase. In this respect, they are akin to quality gates or milestones. The reviews have a significant effect on the way phases are re-iterated.

Since a CSR validates that the program design is a true expression of the output of the analysis phase, there is no point in looking at the program design for the source of an error found during the coding phase. It must have been present at the end of analysis – and perhaps even further back.

The complete architecture proposed by Royce in 1970 – with milestones, prototypes and code reviews – would look something like the iterated linear process in Figure 1.4.
The topic of project planning is covered in detail in Chapter 6 of the subject guide.

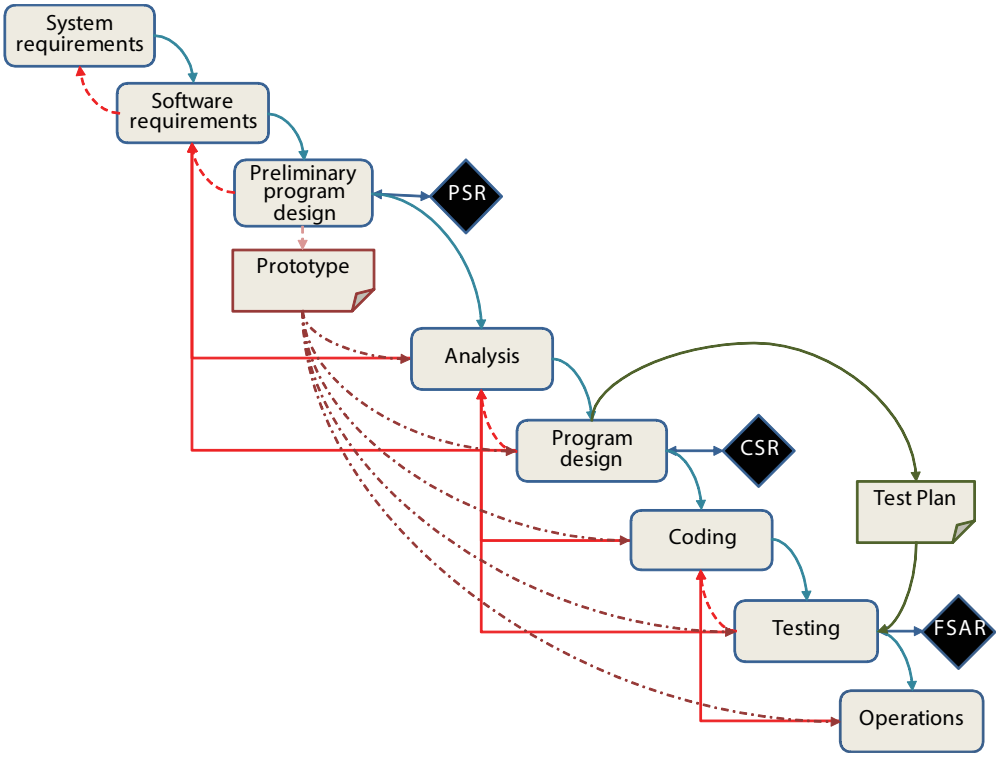


Figure 1.4: Controlling the waterfall, after Royce.

The UP approach

The three diagrams above are all expressions of the way a development project moves through discrete phases. Normally, these phases encapsulate specific workflows such as 'requirements engineering' and 'design'. The UP takes a different approach, separating the lifecycle phases from workflow disciplines. There are four reasons for this:

1. A workflow may extend across several phases. Testing, for example, begins during the elaboration phase, following the V-Modell approach of agreeing a test-plan before software is produced, and continues until the end of acceptance testing in the transition phase and possibly beyond that into planning for the next iteration of the UP lifecycle.
2. Several different workflows or disciplines may be required to fulfil all of the requirements for completing the phase. For example, coding and testing are concurrent or interleaved tasks within the construction phase but require different disciplines.
3. The same (or very similar) tasks may need to be performed several times within a single iteration of a phase. This occurs during incremental development and rapid prototyping.
4. Where the complete process is repeated (for a second release, for example), the planning and justification tasks may take advantage of the outputs from the previous iteration.

The term 'goal-driven' describes the philosophy of the UP very well. Each phase ends with an identifiable moment in time at which a major state change occurs in a project. The inception phase ends when the client is able to authorise the supplier to proceed with the project. That is, the inception phase produces the evidence (including a scoping

statement, budget and resource plan, schedule and analysis of risks) on which the client makes the decision to proceed.

This is a major milestone in the evolution of the project. The others are the stabilisation of project vision and architecture (the goals of the elaboration phase), readiness to roll-out the system developed in the construction phase and the final goal: product acceptance.

Figure 1.5 shows how the linear waterfall process can be broken down into the four UP phases, each with its goal expressed as a milestone. Inception leads to the ability to define system requirements as lifecycle objectives for this iteration. Elaboration results in a lifecycle architecture, or outline design. The end of the construction phase is marked by the availability of tested components, or an initial operating capacity. The cycle repeats if necessary once the product is released. Note how the milestones reduce the need for iteration within the lifecycle compared to Figure 1.4. Unresolved issues detected at those milestone points are either addressed before moving on to the next phase or deferred until the next UP iteration.

The use of the term 'milestone' refers back to the 18th century when roadside stones marked the distance to the destination. Milestones are not a measure of the distance already travelled; they tell you how much further you have to go. In a typical project where activities are planned backwards from an agreed endpoint, a milestone is not the date at which an agreed state is reached – it is the date at which the state was planned to be reached. There is a big difference.

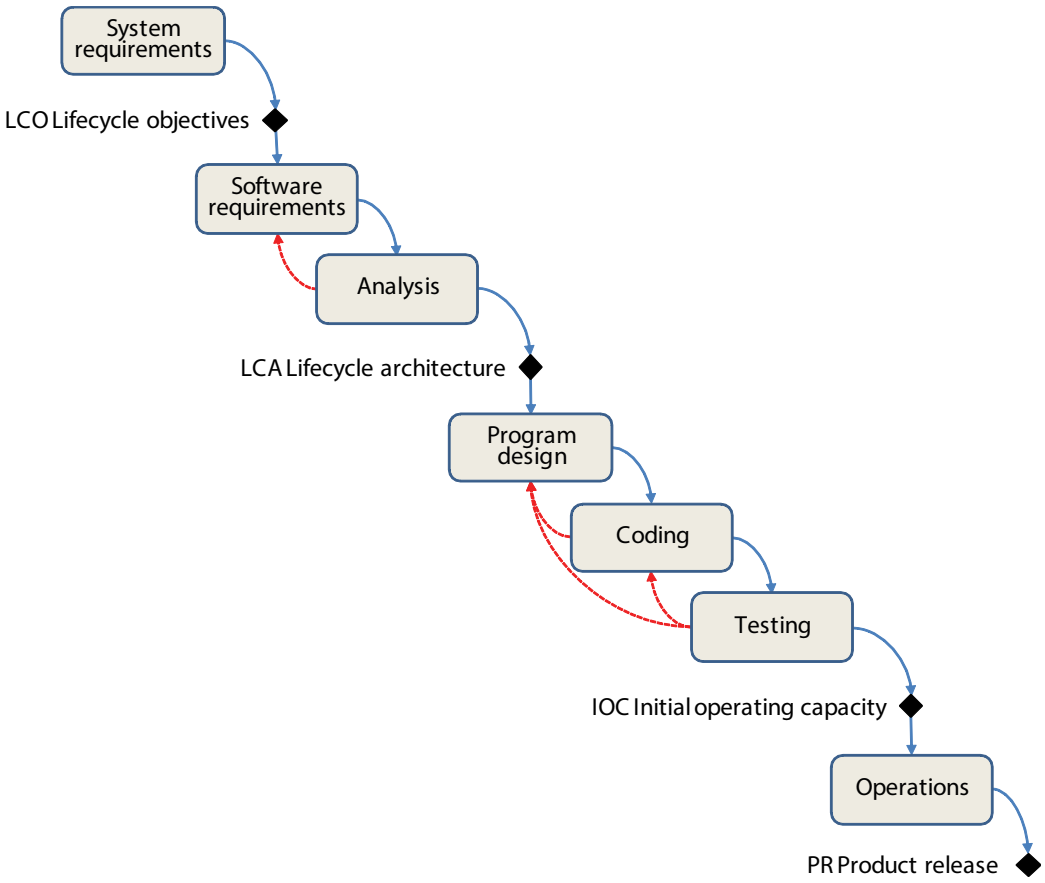


Figure 1.5: Waterfall controlled by UP milestones.

The other difference between a goal-driven and an activity-driven or plan-driven project is that the focus is on achieving results, not on doing the work that should deliver those results. Planning determines the way 'products' are produced, not the way work is organised.

CORNERSTONE 4

The purpose of a software project is to meet a client's need by delivering a software system. That system decomposes into sub-systems and components, and each component requires a number of products to be built, tested and integrated. There are two approaches that a project manager can adopt – focus on the product, or focus on the work required to deliver that product.

Artefacts

An output from the project, or '**product**', is known in the UP as an '**artefact**'. There are many definitions of artefact in different contexts, but the one we will focus on is that it is something that allows another, higher level, product to be constructed.

The artefacts that must be signed off to allow the 'Lifecycle objectives' milestone to be met at the end of the inception phase, for example, should be based on:

- project scope definition
- cost and schedule definitions
- risk identification
- project feasibility
- plans for the project (development) environment.

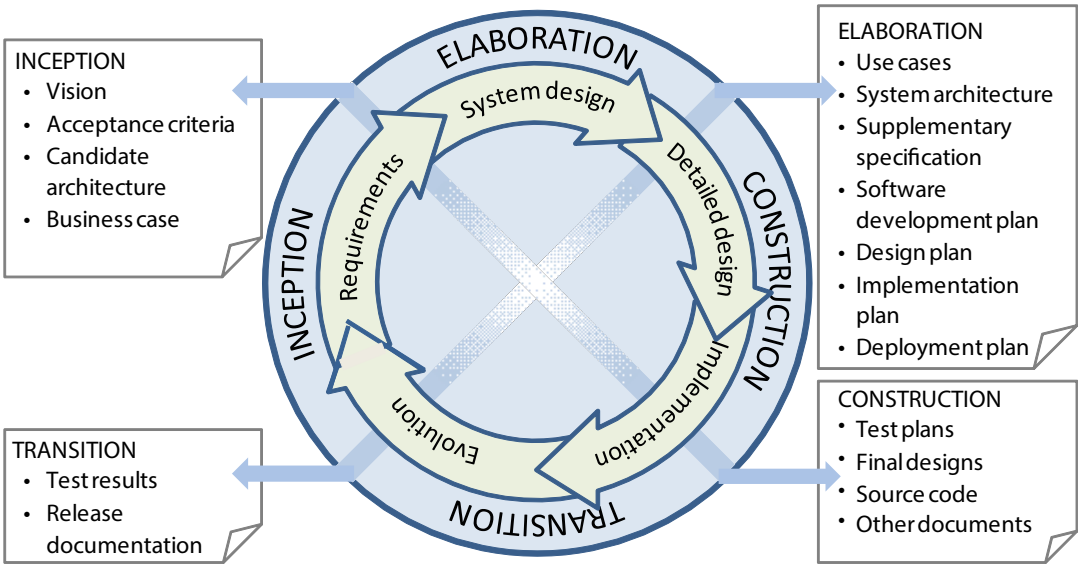


Figure 1.6: Relationships between phases, activities, milestones and artefacts.

Relationship between the UML and UP

There is a further difference between UP and the other methodologies described above and that is the adoption of a standard form of documentation – the 'Unified Modeling Language' (UML (™ Object Management Group, Inc.) (See: Booch et al., 2005).

The UML is the result of efforts to provide a standardised and easy-to-understand way of describing aspects of the structure, interactions and behaviour of a software system. It provides a diagrammatic way of modelling business processes, software requirements and architecture design in UP.

Table 1.1: Use of UML in the RUP.

Scope	UML	Purpose
Business modelling	Activity diagram	Definition of system boundaries and the flow of control between the system and its external context and environment.
	Class diagram	Identification of key business stakeholders or entities.
	Use case diagram	Definition of high-level business requirements.
	Data flow diagram	Description of business processes to give a high-level definition of system scope.
Software requirements	Use case diagram	A statement of a functional requirement (a specific task undertaken on behalf of a specific class of user).
Architectural design	Class diagram	Definition of the classes of object required by the architectural design and the relationships between them.
	Sequence diagram	A more detailed description of the interactions between the actors and the objects or components within a system.
	Component diagram	Definition of the interfaces (services) provided by a component of a system (especially a re-usable component) and the services that have to be provided by other components.
	Activity or sequence diagram	Description of the behaviour of objects in response to a stimulus.
	State diagram	Description of the way objects change state in response to a stimulus.
	Deployment diagram	Description of the deployment of software on hardware connected by middleware at run-time.

All of these models (and six other less-used diagrams) are described by Scott Ambler. (See: www.agilemodeling.com/essays/umlDiagrams.htm)

Agile methods

The principles of agile development are based in a large part on '**lean thinking**', an approach to manufacturing derived from the work of W. Edwards Deming in the 1950s. A 'lean' approach has five underlying principles, which advocates of the agile approach should recognise. We do not need to spend time here discussing the ramifications of the 2001 Agile Manifesto (see: http://agilemanifesto.org) except to characterise similarities and differences between agile and more traditional approaches based on these five principles.

1. Define the activities which add value to a product or service from the customer's perspective and aim to eliminate wasteful activities that do not add value. The agile practitioner would see the value in a brief daily staff meeting looking forward and the waste in a weekly written progress report looking back.
2. Understand how value is delivered to the customer, as an end-to-end process. Identify aspects of value that do not survive that end-to-end process and parts of the process that do not add value. The agile practitioner would see little or no

value in undertaking a comprehensive requirements analysis in areas where those requirements can change as the system evolves.

3. Work out how to remove waste from the process by concentrating on improving the flow of value and challenging ways of working that reduce the flow. The agile practitioner would see the correlation between the size of a task (such as an increment in functionality to be delivered) and the risk that it will contain errors and other forms of wasted work.
4. Learn how to respond to the customer's priorities rather than persisting with fixed ways of working. The agile practitioner would see the benefits of having a flexible and multi-skilled project team that can be configured to reduce the risk of bottlenecks when responding to the customer's requests.
5. The goal of the lean approach in manufacturing is creation of a perfect value chain through principles of continuous improvement and total quality management. Cockburn points out, in discussion of the differences between the agile approach and that of '**C**apability **M**aturity **M**odel **I**ntegration' (**CMMI**) (™ Carnegie Mellon University), that '…CMMI places optimisation of the process at the final level, Level 5. Agile teams start optimising the process immediately.'

One major difference between agile and more conventional methods is in the approach to governance. Governance is about information flows and the way they facilitate decision-making. Ambler (see: www.ambysoft.com/unifiedprocess/agileUP.html) describes delivery of incremental releases over time as 'serial in the large, iterative in the small' where senior management see a linear sequence of milestones and team members see a highly iterative set of envision, explore and deploy activities.

Highsmith (2009) discusses the need to address the concerns of senior management who are looking at a portfolio of projects from a perspective of investment and risk, on the one hand, as well as project and iteration managers at the other. His enterprise-level governance model for agile organisations is essentially based on the UP lifecycle.

The project management perspective

Focus on business case and managing risk

Royce was primarily concerned with managing the technical aspects of software development and most waterfall activities take place during the construction phase of the UP lifecycle. The waterfall starts after the project has been commissioned and ends when the system becomes operational. It does not explicitly include activities such as justification of the project business case or system deployment, evolution and retirement. As a methodology, it does not make risk planning or resource-planning explicit. There is no activity to address the identification or evaluation of options from a risk perspective or for optimising the use of resources by allowing teams to work in parallel on independent sub-goals.

All of these non-technical concerns are, however, within the scope of the UP inception, elaboration and transition phases so that UP provides an overall business context within which the project team can select an appropriate methodology for the construction phase – whether that follows a linear or iterated approach, prototyping, re-use of software objects or '**C**ommercial **O**ff **T**he **S**helf' (COTS) components, incremental development, incremental delivery or a partially automated process based on formal methods.

Some methodologies are more suited to large-scale, mission-critical, projects where risk management is a high priority. Some are appropriate when the client cannot articulate requirements for one reason or another or cannot visualise the end-product. Sometimes the most appropriate methodology depends on constraints associated with the context of the project, such as the need to migrate an operational system or a project that forms just part of an overall program of work.

One other common form of process model must also be considered, and that is Boehm's 'Spiral Model' (Boehm, 1988), described in both recommended textbooks. Here, we need only consider the essential characteristics of the Spiral.

- Since the elements in the model are addressed sequentially, one journey round the axis of the Spiral can represent one iteration of a process model (for example, the UP model based on business case acceptance, development plan acceptance, system acceptance and deployment).
- At a lower level, the journey could represent one phase of such a model (for example, the elaboration phase, which involves capturing and modelling requirements, identifying risks and ways of managing them, developing an architectural framework, identifying resource requirements and producing the development plan).
- Because the elements of the Spiral are organised concentrically round that axis, the journey can be divided into standard sub-goals, such as Boehm's objective-setting, risk assessment and reduction, development and validation followed by planning for the next iteration.
- Within a sub-goal, a standard set of activities may be required. Risk assessment and reduction typically requires analysis of the risks that were identified for this phase, prototyping or simulation to validate the analysis, benchmarking to provide a baseline for verification and so on.

Boehm's development and validation phase can be implemented, like the UP construction phase, using any valid process model. When constructing a web-app based on the '**M**odel, **V**iew, **C**ontroller' (**MVC**) architecture, for example; business rules for the model can be verified by prototyping and implemented incrementally; the view can be built by re-using existing objects and application-specific sub-systems within; the controller could be implemented and integrated using a waterfall approach.

The framework is robust enough to be used in non-prescriptive ways. For example, Boehm says 'prototype' but does not define the nature of the prototype.

Best practices

CORNERSTONE 5

A best practice in common use is the breakdown structure. A Work Breakdown Structure, for example, decomposes the project into phases, decomposes phases into activities and activities into tasks and sub-tasks. You can describe dependencies from one to another through a Work Flow Diagram (for example, a GANTT chart; named after the person who developed it, Henry Gantt). Similarly, you can represent the relationships between artefacts from a project in a Product Flow Diagram, which then defines the order in which they must be completed. You can also use breakdown structures to ensure that an analysis of factors such as risks or competences is comprehensive.

As described by Sommerville, UP embodies three views on the software engineering process.

- A dynamic model of the evolution of a project over time.
- A static description of the activities that form the components of that process model.
- A generalised description of the types of good practice which are appropriate for execution of those activities.

Best practices include suggestions for workflows and the document artefacts that will normally be needed, and their purpose/readership, content and structure.

Table 1.2: Best practice workflows and artefacts.

Workflow	Scope	UML artefacts
Business modelling	Business processes modelled in a form understood by both business and software analysts	Business use case descriptions organised into the business object model.
Requirements	Functional requirements	Use case descriptions, collectively the use case model.
	Non-functional constraints – trade-offs and decisions	Supplementary specifications.
Analysis and design (the '4+1' views, see Figure 5.1)	Static structure (logical view)	Object model (class diagram).
	Dynamic behaviour (process view)	Object model translated into sequence diagram. Objects with complex state-change behaviour may require a state diagram.
	Physical organisation of components (implementation view, optional physical views)	Object model translated into component diagram. Optional deployment diagram showing how components are deployed on distributed platform.
Implementation	Building components and sub-systems	The component model is implemented in code in agreed increments.
Testing	Component testing	Components are verified and validated against the relevant use case(s).
	System testing	The system produced by the implementation workflow is validated against the use case model and SRS.
Deployment	Acceptance testing Roll-out	A product release is created, distributed and installed in accordance with the deployment model.
Support	Configuration and change management	
	Project management	
	Development environment	

There are other sources of good practices, such as the AUP, PMI's '**P**roject **M**anagement **B**ody of **K**nowledge' (PMBOK®; PMI, 2013). The freely-available '**ProjectInABox**' provides document templates suited for use within a variety of project management approaches (see: www.projectinabox.org.uk).

Other practices include generic approaches to standard activities like engaging with users to elicit requirements and how to conduct milestone reviews.

Reminder of learning outcomes

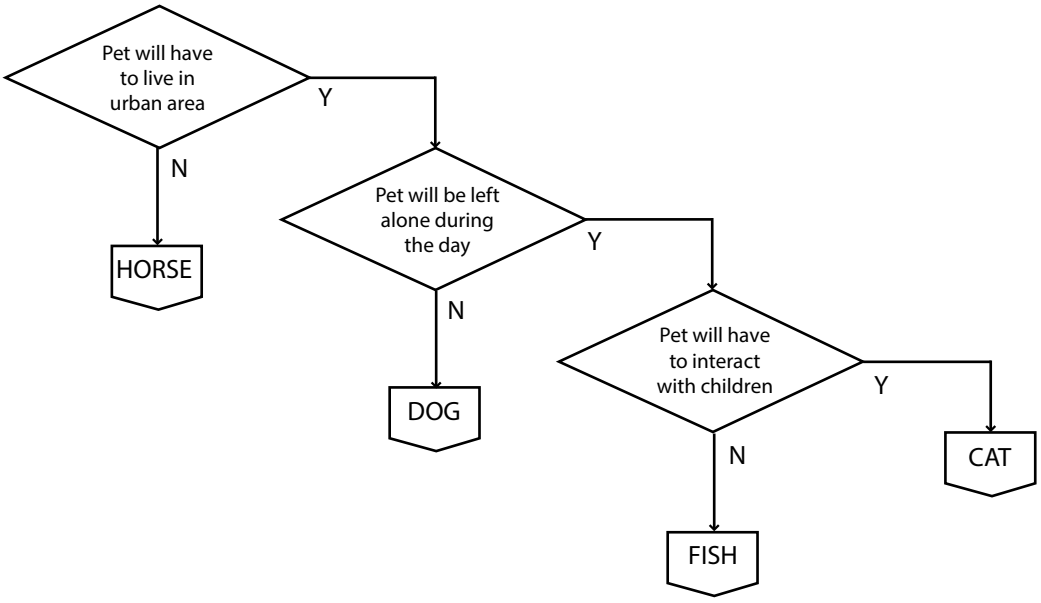
Having completed this chapter, and the Essential reading and activities, you should be able to:

- understand the concept of a process model as a way of describing a software development project and differentiate between common approaches such as waterfall, iterative and agile
- identify the key artefacts produced during an iteration of the software development process
- understand the role of the project manager in delivering those artefacts
- understand the ways in which the development process can affect quality.

Test your knowledge and understanding: defining appropriate models

We have discussed several key process models, but the Essential reading textbooks describe others. There is no 'one size fits all' process model and it is important for you to show that you can identify the strengths and weaknesses of the most-used models in a variety of contexts.

6. Consider the distinctive features of these key process models and the types of project that they are recommended for. For example, Pressman suggests that the waterfall approach is applicable when requirements are pre-defined and not going to change and the work is essentially linear. This may be the case with an embedded system or device driver.
7. Record your findings in a Decision Tree (a flowchart consisting of Yes/No branches) to select an appropriate process model based on the requirements of a project. For example, a decision tree for selecting a suitable pet might include the following:



8. You can also summarise your logic in a table with the key process models in the left hand column and the types of project on the right, building on the following:

Process model	Project type
Waterfall	Device driver (requirements are known)
	Embedded system (development is linear)

Chapter 2: Requirements engineering

Learning outcomes

By the end of this chapter, and having completed the Essential reading and activities, you should be able to:

- describe the differences between elicitation, specification, validation and documentation of requirements
- understand the differences between system and user requirements and between functional and non-functional requirements
- use natural language and graphical representations of requirements within a requirements document
- produce a system model based on requirements.

Essential reading

Sommerville	Pressman
Chapter 4: Requirements engineering	Chapter 5: Understanding requirements

References cited

Grady, R. and D. Caswell *Software metrics: establishing a company-wide program.* (Upper Saddle River, NJ: Prentice Hall, 1987) [ISBN 9780138218447].

Overview

The process of identifying and recording requirements is explained in depth in both recommended textbooks. The importance of the process is that it yields an understanding of the scope of the project (the business needs to be met), the outputs from the project (artefacts) and the way of assessing the outcomes (benefits) that are achieved. It normally forms the contractual agreement between supplier and customer.

CORNERSTONE 6

A specification document must comply with the 'four Cs':

Clear: No ambiguities or confusing cross references to other documents.

Concise: No unnecessary detail, repetition or confusing cross-references to other documents.

Correct: Attributed, with no conjecture, paraphrasing or unsubstantiated opinion.

Complete: Verified as part of the V & V process.

The first step in scoping many projects is a feasibility study and this is very often a technical rather than a commercial decision. Examples include:

1. A technical study to see if a legacy database contains accurate enough data to allow an enhancement.
2. A pilot to evaluate the feasibility of reverse-engineering an existing process.
3. A trial to see how many extra sales agents the current network will support.
4. An evaluation of three vendors' COTS software.
5. A mock-up interface to see whether staff would feel comfortable using the proposed new Human Resource system.
6. A field trial to measure a round-trip time using a cloud architecture.

However, stakeholders should also be concerned with non-technical issues that affect a project's feasibility. According to Figure 1.6 in the previous chapter, the purpose of the inception phase of a UP project is to produce a set of artefacts that allow the two prime project stakeholder groups, the customer(s) and their supplier(s) to determine the feasibility of a new project (or new iteration of an existing project). The result will normally be expressed in a business case. It is vital that the business case is kept up to date throughout the life of the project, as a way of evaluating the ongoing viability of the project in response to changes in the status of risks.

Figure 1.6 also contains a much longer list of artefacts generated within the elaboration phase. One of the consequences of splitting the 'front-end ' of the project into inception and elaboration phases is that it avoids the need to create a detailed software development plan and user requirements statement before finding that the project cannot be cost-justified commercially. In an iterative, incremental or agile environment, some requirements may remain undiscovered until the construction phase.

CORNERSTONE 7

The project team invariably focuses on the outputs it is to create, while the client is concerned with the outcomes of using those outputs. 'Outcome-based specifications' address the client's needs and do not prescribe **how** they are to be met.

The output-based specification for a new sales-support system might include 'produce a weekly printed report from the database enquiries table summarising the status of sales enquiries received in the last week grouped by sales territory'. The rationale for this kind of statement might be that managers need information in that format in preparation for the weekly sales-team video-conference. A more likely reason, of course, is 'we've always done it this way'. The outcome-based equivalent would simply be: 'sales managers require up-to-date information on the status of enquiries in each sales territory'. This would allow the innovative supplier to suggest the use of an on-screen 'dashboard'. By putting the dashboard on the intranet, where it can be accessed by the sales team as a whole, time can be saved each week in the sales-team video-conferences.

The decision to proceed with a project is likely to be based on the answers to the following questions:

Questions the customer should be asking
PROJECT OUTCOMES:
Does the supplier know what we are trying to achieve?
PROJECT OUTPUTS:
Can we agree what the key deliverables and their acceptance process(es) will be?
BUSINESS JUSTIFICATION:
Do the potential benefits outweigh the costs and risk of taking the project forward?
Short/Medium/Long-term projections: WILL THIS STILL BE THE RIGHT DECISION IN ONE/THREE/10 YEARS' TIME?

Questions the supplier should be asking

OUTCOMES:

Has the customer explained why the project is important to them?

PROJECT DOMAIN:

Are we analysing the right problem? Are we listening to the right stakeholders?

APPROACH:

Can we agree an overall project architecture? Can we do the job?

POTENTIAL THREATS:

Do we understand the development process well enough to be able to identify and evaluate the key risks?

POTENTIAL BENEFITS:

Can we evaluate the potential business benefits (profit, avoiding disbanding the team, acquiring a new customer, positioning ourselves in a new market or developing new skills)?

Short/Medium/Long-term projections: WILL THIS STILL BE THE RIGHT DECISION IN ONE/THREE/10 YEARS' TIME?

Depending on the scale of the project, some or all of the following are likely to be used to help evaluate the answers:

Customer business information sources

- **Business vision** – Corporate goals.
- **Business rules** – Policies and constraints.
- **Business case** – Source of information about corporate values (intangible).

Project definition sources

- definition of scope
- user requirements, possibly supported by user profiles and key high-level use case definitions
- supplementary, non-functional, requirements
- interfaces: users, hardware, software, communications
- an inventory of risks with strategies for their avoidance or reduction
- project approach
- project architecture
- high-level project plan
- high-level resource plan
- project evolution.

Validation

Validating requirements locally with users – through mock-ups and scenario-based models – will reduce the risk of starting development on the basis of an incomplete or inaccurate specification of the static elements of the system. For example, one of the first things a web-designer is likely to ask a new client for is a list of examples of existing websites that embody the expected visual styling, tone of language, user functionality and overall business purpose.

Partially-functional prototypes provide another tool for eliciting feedback from users as the detailed design evolves. This has become increasingly valuable as developers gain access to re-usable library objects, especially user-interface objects, and acquire skills in using techniques from the field of usability.

Software Requirements Specifications

Descriptions of 'Software Requirements Specification' (SRS) documents can be found in Sommerville, Karl Viegers (see: www.processimpact.com/goodies.shtml) (the RUP-based origin of the outline referenced in Pressman) and, of course, the UP itself. These are very similar in structure, if not identical.

The content of each section should be clear from the following outline:

Introduction	Document purpose and project scope
Project overview	Business context for the proposed system
System features (from user requirements)	Functional requirements
External interfaces	Operational context
Supplementary requirements	Non-functional requirements

The lean thinking of agile methodologies such as Ambler's AUP emphasises reusing relevant source documents rather than rewriting or editing them. By the time the construction phase has started, the agile SRS could include UP artefacts containing documentation of business rules, domain model, organisation model, project glossary, analysis of automation opportunities, a business process model, technical requirements, use case model, a user interface model and definitions of acceptance tests. The exact mix will vary from project to project.

The SRS is not complete until the end of the high-level analysis phase, by which time the use cases or user requirements statements have been transformed into concrete system features.

CORNERSTONE 8

Functional requirements describe the way the system behaves.

Non-functional requirements describe the context within which that system behaviour is delivered.

User requirements are the functional and non-functional requirements that describe the business goals for the system.

System requirements are the functional and non-functional requirements that have to be met in order to achieve the business goals for the system.

Key concepts: gathering, analysing and formalising requirements

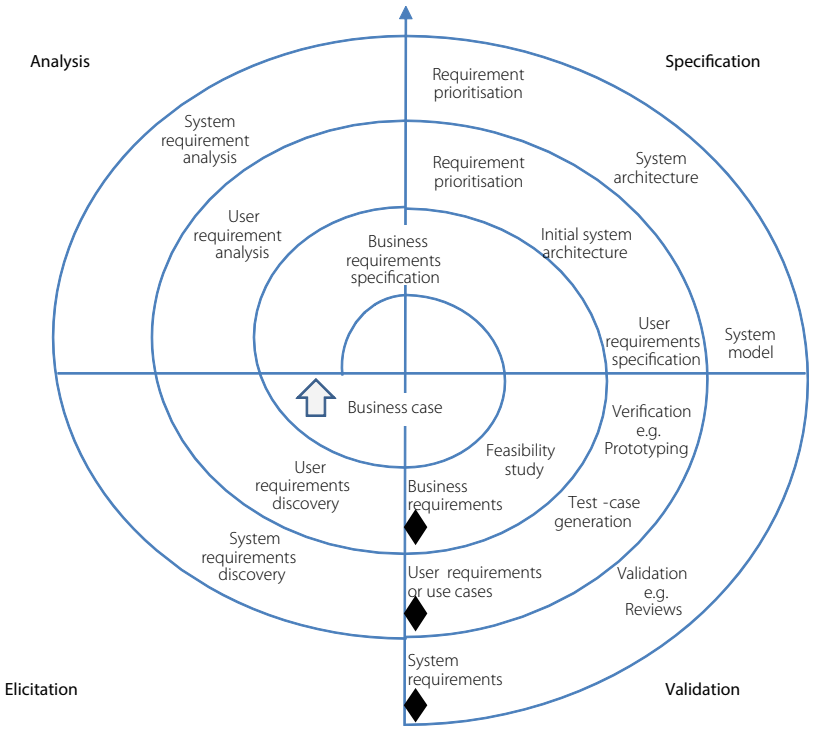


Figure 2.1: Requirements engineering, after Sommerville.

Figure 2.1 captures the logic of two diagrams from Sommerville, integrating details of elicitation and analysis activities within the process of defining an SRS and business, user and system requirements. The Spiral approach highlights the order in which these different specifications are normally produced:

- Business requirements define the scope of the project.
- User requirements define the system to be produced in sufficient detail to be able to plan the project.
- System requirements define the system in sufficient detail to allow it to be implemented.

The Sommerville and Pressman textbooks both describe the process of requirements engineering, including the various forms that a completed set of requirements can take – unstructured or structured narrative (natural language), high-level mathematical or logical expressions or graphical representations such as the UML. The process begins with the identification of the customer's requirements and there are various ways in which those requirements can be captured (elicited or discovered) and analysed.

The primary aims of analysis are to ensure the accuracy and completeness of the understanding of the customer's needs, which may influence the scope of the project. Prioritisation of the specific requirements allows functionality to be allocated to the first or subsequent iterations of the project.

The project management perspective

Modelling the business requirement

PMBOK® identifies seven reasons why a development project might be initiated. Since PMBOK® is not specifically concerned with software engineering, we have modified the list to reflect the types of project that you are likely to encounter:

- responding to market demand for a feature, product or service
- addressing internal needs or opportunities within the organisation
- servicing a customer request for a bespoke system
- technology-driven need to update an existing system
- constraints such as legal changes
- improving the way a system operates, such as reducing environmental impact
- infrastructure development (originally categorised as 'social need', such as providing drinking water).

Some projects will have clear monetary benefit (such as a successful upgrade to an existing system) while others may have less tangible outcomes such as improving customer satisfaction. In all cases, the decision to proceed is based on acceptance of the business case which gathers the evidence that delivering the project will result in the expected benefits. It will have to be revisited at various times during the project.

The starting point is normally 'the cost of doing nothing'. In other words, the factors in the decision are relative to that baseline, but are not absolute. This has implications when considering the acceptability of the system (discussed below), the trade-off between incompatible requirements, such as the time it will take to deliver something that 'works' and something that is robust.

It is the project manager's responsibility to provide accurate and complete evidence for the business case. As the project evolves, the business case will evolve to include cost and schedule estimates, which we will address in the next chapter of the subject guide. It will also include documentation of the requirements that have been identified and the response in terms of specific features, since these are the basis on which the costs are calculated. The project manager's main concerns are, therefore, whether the requirement is:

Understandable?	Complete within a well-defined context? (For example, if there are multiple user-types, the requirement may only apply to one user-type.) Stated in a clear, unambiguous way for the intended reader? Defined in a form that is consistent with the body of requirements? A description of a required outcome rather than a way of achieving that outcome? Attributable? Current (relevant to the current situation, not something learned from an out-of-date customer document)?
Implementable?	Necessary ('need to have' or 'nice to have')? Realistic? Measurable (non-functional requirements)? Testable? A prerequisite for another requirement? Dependent on the implementation of other requirement(s)?

This checklist is an example of a breakdown structure and every project manager should have such a framework to ensure that their analysis is complete.

FURPS – Functionality, Usability, Reliability, Performance, Supportability

'**F**unctionality, **U**sability, **R**eliability, **P**erformance, **S**upportability' (**FURPS**) (Grady and Caswell, 1987), developed at Hewlett Packard, is based in part on the thinking of Joseph M. Juran (see: www.juran.com). It defines the relationship between the functional and non-functional properties of a system in terms of its '-**ilities**'. 'An "**ility**" is a characteristic or quality of a system that applies across a set of functional or system requirements' (see: www.softwarearchitecturenotes.com/architectureRequirements.html). In other words, a given area of functionality must be implemented in accordance with a given set of non-functional requirements. Where there is a tension between two areas, it is the project manager's responsibility to establish what the customer's priorities are and what constitutes an acceptable balance between functionality, usability, reliability, performance and supportability. The following list of ilities may look like a simple checklist based on those categories, but it provides the customer with a mechanism for quantifying the importance of the various non-functional requirements, using a simple weighting function (a scale of 1 to 5).

FURPS+ (the '+' signifies that extra attributes have been added since 1987) is similar to ISO 9126 which divides **Supportability** into **Maintainability** and **Portability**. In either form, it is an important part of the discussion of measuring progress and also the related issue of the quality of that progress.

Functionality	Feature set, capabilities, generality, security.
Usability	Human factors, aesthetics, consistency, documentation.
Reliability	Frequency/severity of errors, recoverability, predictability, accuracy, mean time between failures.
Performance	Speed, efficiency, resource consumption, throughput, response time.
Supportability	Testability, extensibility, adaptability, maintainability, compatibility, serviceability, installability, localisability.

The relationship between requirements and system acceptability

A successful software engineering project is critically dependent on the accuracy and thoroughness of the requirements engineering process in order to:

- Determine the customer's functional and non-functional requirements.
- Balance those requirements to deliver the functionality, reliability, performance and supportability requested to an acceptable extent.
- Identify constraints that affect the project's duration, cost, choice of methodologies and standards.

Change

Evolution of a system is the subject of Chapter 9 of the subject guide. Here, we only need to say that the process of specifying and implementing a new version of a system should follow the process described above for the initial system. That is:

1. Changing business requirements drives the process through the Spiral model shown above – namely, high-level user requirements followed by revised system specification.
2. The decision to proceed is based on a business case.
3. The system is achievable in terms of balancing requirements.

Implications for project planning

Project planning, at a detailed level, is discussed in Chapter 6 of the subject guide. However, the requirements capture process described in the previous section is critical to success in planning what the development team will do as well as for providing evidence as to whether the project should proceed or not. The decisions that need to be made are:

1. Who will do the work?
2. What are the implications of the chosen project architecture?
3. How can the major risks be avoided or mitigated?
4. How can the work be broken down to make the project more controllable?
5. What resources need to be acquired?

These decisions are typically taken in parallel with and throughout the process of translating user requirements into system requirements.

Reminder of learning outcomes

Having completed this chapter, and the Essential reading and activities, you should be able to:

- describe the differences between elicitation, specification, validation and documentation of requirements
- understand the differences between system and user requirements and between functional and non-functional requirements
- use natural language and graphical representations of requirements within a requirements document
- produce a system model based on requirements.

Test your knowledge and understanding: gathering evidence

Sommerville proposes a template for an SRS and it is important that you understand the likely sources of the evidence that you use to develop those requirements and how the project documentation evolves.

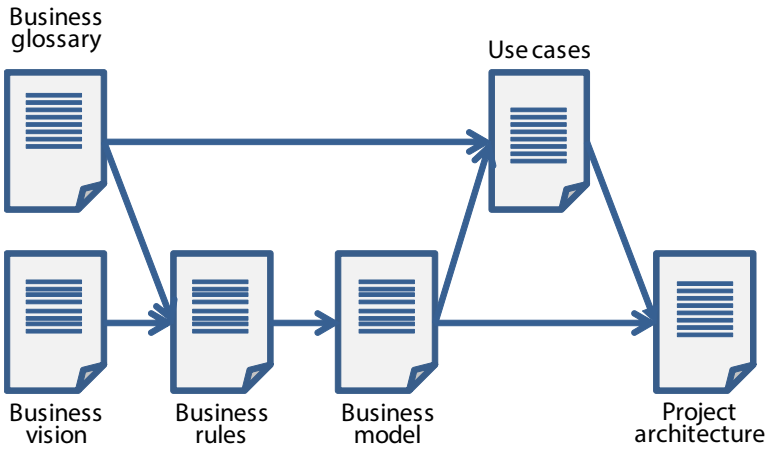


Figure 2.2: Extract from a Product Flow Diagram.

1. Review the content of this chapter, and in particular the outline SRS.
2. Now, construct a **'P**roduct **F**low **D**iagram**' (PFD)**, following the example in Figure 2.2 to explain the artefacts needed to produce an SRS.
3. Finally, divide the artefacts into those produced before the end of the inception phase and those produced during the elaboration phase.

Chapter 3: Planning, cost and schedule estimation

Learning outcomes

By the end of this chapter, and having completed the Essential reading and activities, you should be able to:

- describe the structure of a project schedule
- apply resources to execute a project plan
- understand the constraints on the accuracy of resourcing and budgeting.

Essential reading

Sommerville	Pressman
Chapter 23: Project planning	Chapter 26: Estimation for software projects Chapter 27: Project scheduling

Further reading

Fenton, N. and S.L. Phleeger *Software metrics: a rigorous and practical approach*. (Boston: PWS Publishing, 1998) second edition [ISBN 9780534954253].

References cited

Hsu, C.C. and B.A. Sandford 'The Delphi technique: making sense of consensus', *Practical Assessment, Research & Evaluation* 12(10) 2007, pp.1–7; http://pareonline.net/pdf/v12n10.pdf

Overview

Successful execution of a software project is dependent on three planning activities:

1. Budget/resource-planning.
2. Scheduling.
3. Monitoring and progress management.

In 1995, the Standish Group estimated that software projects were costing US companies and government agencies $250bn/yr (see: www.spinroot.com/spin/Doc/course/Standish_Survey.htm). A third of this money ($81bn) would be on projects that were cancelled before they saw the light of day. A third of the balance ($59bn) would go on finishing software projects that would be completed, but exceed their original time estimates.

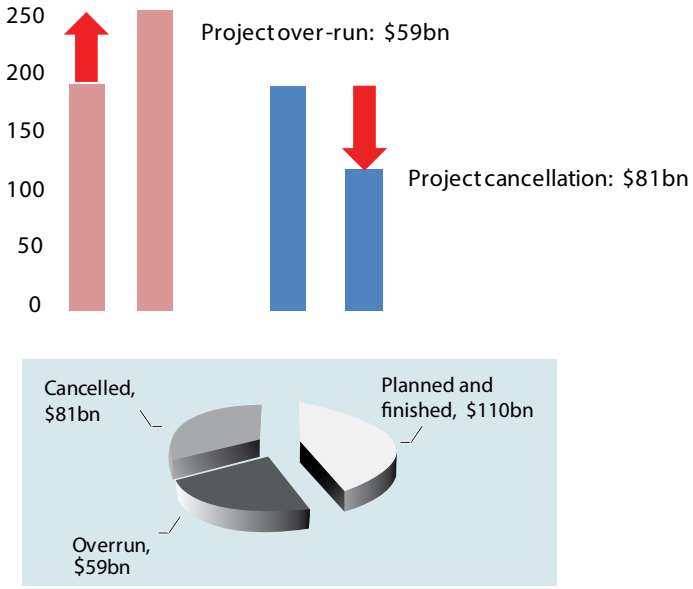


Figure 3.1: Project failures, after Standish.

The research has subsequently been repeated on several occasions and, while the numbers may vary year-on-year, there is no evidence of any real improvement in the number of projects that are completed on-time and on-budget, with all features and functions as initially specified or a reduction in the number of 'impaired' projects that are terminated prematurely.

The remainder, of course, are the projects that are completed and operational but over budget, over the time estimate, and offer fewer features and functions than originally specified.

Plan-driven development and agile development

The essence of a plan-driven approach to a project is that the customer and developer commit to a detailed project plan (specifying and scheduling release of a series of products) before entering the construction phase. The factor that remains (relatively) constant is the functionality and performance of the product. Deviation from plan results in a product going out of tolerance (time, cost or quality).

An agile development team does not expect to be constrained by a detailed plan (although high-level planning may be required by the principle of '**serial in the large, iterative in the small**') since the 'constant' is not the product but the timing of the iteration cycle. Functionality is dynamically prioritised and allocated to a planned release.

The two approaches are simply different ways of achieving the same goal: fulfilment of the customer's strategic criteria identified by Price Waterhouse Cooper (PWC). Not only time and budget but stakeholder satisfaction, benefit realisation, quality and return on investment. Sommerville makes the point that many projects will incorporate elements of both approaches and lists a series of questions that affect the way the project is managed. The implications of these technical, organisational and human issues can be represented as follows:

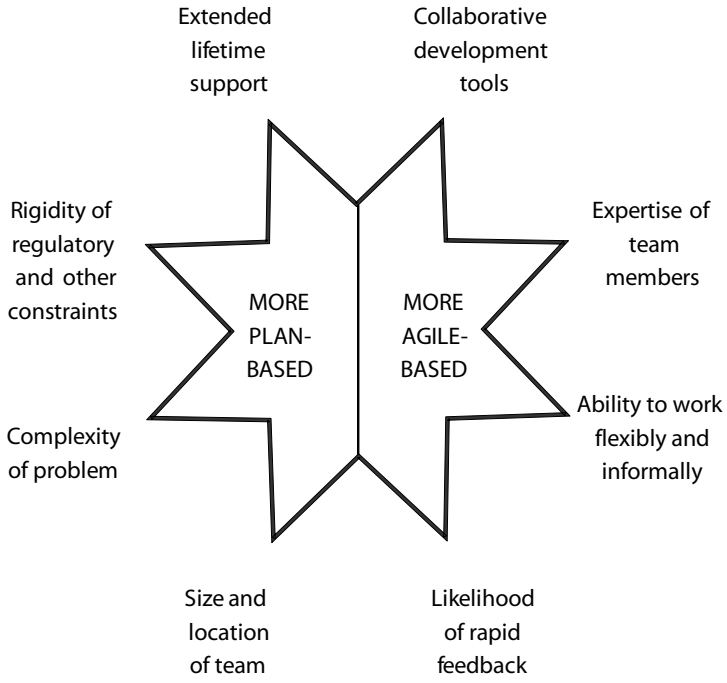


Figure 3.2: Constraints on development strategy.

Key concepts: time, money and quality

Scheduling: time

Scheduling is essentially the translation of the process model into a network of activities. Four factors determine how the network is constructed:

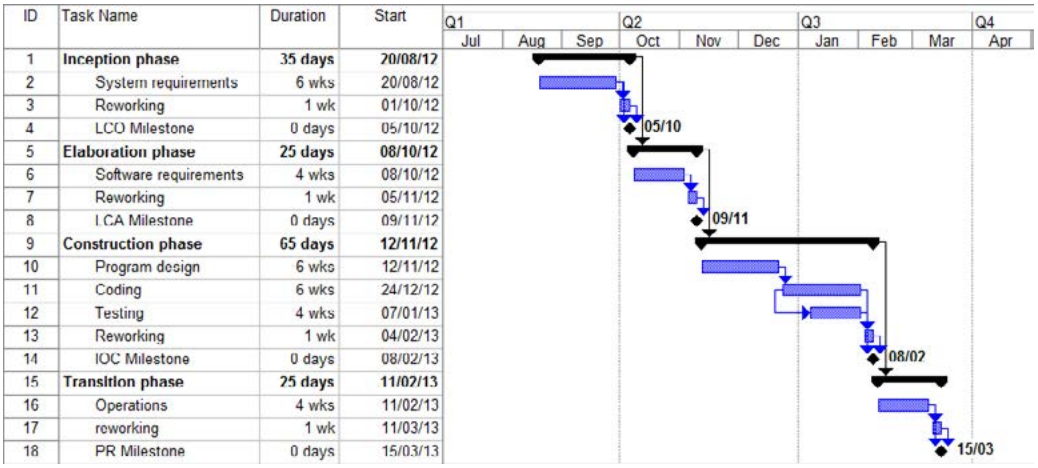
1. The way the project decomposes into **tasks** (work breakdown structure).
2. The availability of the **resources** needed to execute the tasks.
3. The estimated **time** to completion for each task.
4. The **interdependencies** that dictate prerequisites for starting a task.

Three additional factors affect the overall duration of the project.

5. **Reworking** caused by design/software errors.
6. **Reworking** caused by changes in customer requirements.
7. **Delays** caused by risks that materialise as issues.

The most common expression of the network is the bar chart or GANTT chart. This consists of rows reflecting the hierarchical organisation of tasks in the '**W**ork **B**reakdown **S**tructure' (**WBS**) with time as the horizontal axis. Each bar (representing the expected start and finish dates) has an implied or explicit relationship with others – its predecessors and successors. Overlap of parts of two bars means they are concurrent. Most project planning software (such as Microsoft Project) maps 'work to task' as well as 'task to time'. Resources can be allocated to tasks and as the project unfolds, the tool can track the rate of progress and consumption of resource.

The project plan for the process depicted in Figure 1.5 in Chapter 1 of the subject guide could be depicted as follows:



ID	Task Name	Duration	Start
1	**Inception phase**	**35 days**	**20/08/12**
2	System requirements	6 wks	20/08/12
3	Reworking	1 wk	01/10/12
4	LCO Milestone	0 days	05/10/12
5	**Elaboration phase**	**25 days**	**08/10/12**
6	Software requirements	4 wks	08/10/12
7	Reworking	1 wk	05/11/12
8	LCA Milestone	0 days	09/11/12
9	**Construction phase**	**65 days**	**12/11/12**
10	Program design	6 wks	12/11/12
11	Coding	6 wks	24/12/12
12	Testing	4 wks	07/01/13
13	Reworking	1 wk	04/02/13
14	IOC Milestone	0 days	08/02/13
15	**Transition phase**	**25 days**	**11/02/13**
16	Operations	4 wks	11/02/13
17	reworking	1 wk	11/03/13
18	PR Milestone	0 days	15/03/13

Figure 3.3: 'Waterfall' GANTT chart. Also available on the VLE.

This GANTT chart assigns a bar (and therefore resource) to reworking that arises from unsuccessful milestone reviews. This means that a known level of resources have been scheduled to reworking. The bar appears to the left (that is, before, in time) of the milestone, not after it as might be assumed. The reason is that the milestone has not been passed if reworking is taking place.

The WBS does not in itself carry any timing or inter-dependency information. One good way of informing the design of the GANTT chart is to design a Product Flow Diagram (PFD) linking the outputs of each task. It is actually easier to develop an accurate WBS from the PFD than it is to do the design the other way round.

Estimating: cost

There are five components of a costing calculation for software development:

1. **Scale** of problem
2. **Efficiency** of staff
3. **Unit cost** of staff
4. Project-specific **equipment costs** (for example, servers, licenses)
5. **Overheads** associated with staff time (travel, accommodation, utility equipment such as phones, laptops).

Estimating the work required to deliver the end result is the one difficult part of this equation, especially in the early days of the inception phase. Whether the unit of work is **'L**ines **O**f **C**ode**' (LOC)** or methods of a class, efficiency is an expression of the number of those units that a member of staff can produce, error-free, in a working day. This is relatively straightforward in the case of a stable, effective development team working on a familiar task and likely to be much more accurate than the estimate of the number of units of work required.

There are three basic methods for estimating the scale of the problem:

1. Guesses (estimates made by experienced practitioners)

   One way of improving the accuracy of expert estimates is to elicit smallest (s), largest (l) and most likely (m) values, where the expected value (E) is given by E = (s+l+4m)/6 assuming a beta distribution.

2. Revisions of previous guesses (based on the actual effort required to deliver similar projects)

   Where a pool of expertise is available, a 'Delphic consensus' (also known as the 'Delphi technique') can be used. Even an average is more robust than a single opinion.

3. Statistical manipulation of guesses, where the guesses become parameters of a calculation

   There are three significant approaches to statistical estimation of effort described in the recommended textbooks. These are:

   1. Constructive Cost Model (COCOMO II)
   2. Function Point Analysis (FPA)
   3. Object Oriented (OO) approach.

   Sommerville identifies several reasons why the result may not be used as the final price:

   Exchanging profit for opportunity such as:

   - entry into new market or working with new client
   - retention of rights to use code
   - gaining experience, skills or specialist staff.

   Exchanging profit later for cash flow now in order to pay the bills (mostly salaries). Remember that this is itself a risky option, profit or loss being '…**the very small difference between two very large numbers**'.

Performance: quality

Management of the delivery of quality is addressed in Chapter 8. However, the project plan itself must reflect the approach to quality. It is generally assumed that the goal of a project sponsor and, therefore, his or her designated project manager is to deliver 'on time' and 'on budget', but that is only part of the story. A Price Waterhouse Cooper survey (2007) (see: www.pwcprojects.co/Documentos/BRO Boosting Business Performance DC-07-1104-A v8[1].pdf) found that only 19 per cent of managers prioritised on-time delivery and a further 18 per cent budget as their primary criterion for success.

Nearly two-thirds, therefore, emphasised more strategic criteria such as satisfaction of their stakeholders (20 per cent), delivery of benefits (17 per cent), quality of the result (15 per cent) and return on investment (9 per cent). Part of the role of the project manager is to establish methods of monitoring progress towards such strategic objectives and this will affect the selection of the most appropriate SE methodologies for a particular project. The solution (and the way of managing its implementation) depends on the nature of the problem.

CORNERSTONE 9

ISO/IEC 12207 defines validation as confirmation that the final, as-built, system fulfils its specific intended use, that it is 'fit for purpose'. Verification confirms that the output of each activity (such as requirements capture, design, code, integration or documentation) fulfils the requirements or conditions imposed by the preceding activities.

This is often expressed in terms of validating that the 'right thing' is being built and verifying that the thing is being 'built right'.

The project management perspective

There are several project management methodologies that can be applied to software projects including '**PR**ojects **IN C**ontrolled **E**nvironments' (PRINCE2) (™ HM Government), the Project Management Institute's PMBOK® mentioned earlier and ITIL ('Information Technology Infrastructure Library') (™ HM Government).

We discuss the principles that underpin these methodologies throughout this subject guide. For example, PRINCE2 takes a product-centric view of a project, contains mechanisms for controlling the transition from phase to phase and divides work into work packages, each typically responsible for delivery of one major project artefact.

We will not examine your knowledge of the terminology or processes of a specific methodology, but students are encouraged to investigate one or more of the major ones. For example, there is a comprehensive web-based PRINCE2 reference resource available (see: www.crazycolour.co.uk/prince2/?title=Main_Page). A freestanding tool has been produced called 'Project In a Box' that includes a free downloadable 'community edition' (see: www.projectinabox.org.uk). This also contains support for ITIL and other methodologies.

Product-based and activity-based planning

Product-based planning is one of our cornerstones. It allows resources to be scheduled according to the role each resource plays in constructing the product and it allows progress to be monitored in terms of delivery of products rather than completion of tasks that contribute to those products. For example, three teams working in parallel are planned to produce components (user interface, the database transactions and the implementation of business rules) that will result in achievement of a major sub-system.

The modified PFD (Figure 3.4) shows the status of the sub-system or component. Design is complete, work has started on the business rules module and the other two modules are finished but not tested. Knowing that the user interface design team has completed their task tells you nothing about the status of the sub-system. You can only make a meaningful statement about the status of the higher-level product when all three components are ready for integration, which is when it reaches its first milestone (prerequisites are in place).

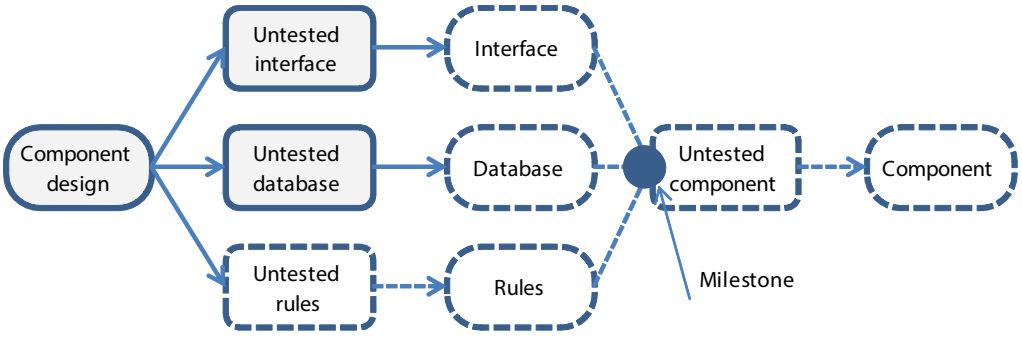


Figure 3.4: PFD showing product status.

Timing

The tracking GANTT (Figure 3.5) shows the activity-based view of the project on 26 September when the database work finishes and is ready for testing, three days late. The user-interface task is on schedule and 60 per cent through the unit testing sub-task.

But the third activity, the business rules work, is only 80 per cent complete. It has a scheduled nine days of work left before the tested component can be released. That would imply that the first milestone on 28 September, already revised to 3 October due to delay in the database task, will probably slip to 8 October. The finished artefact would be more than a week late.

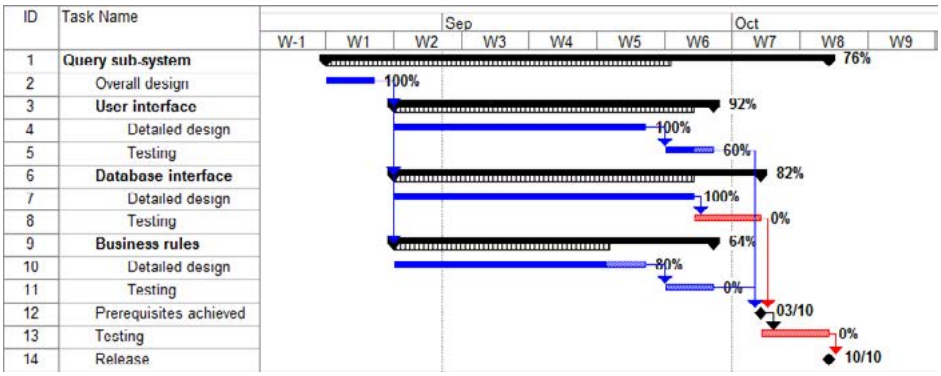


Figure 3.5: Tracking GANTT. Also available on the VLE.

Monitoring

Monitoring the plan requires an objective assessment of the extent to which a product has been achieved (often, internal milestones or 'checkpoints' are used for this) rather than a statement of the effort that has been used en route to that checkpoint. Consider the following:

'The GUI prototype has been evaluated by end-users and our business rules have been accepted by the client, so we are ready to begin the final iteration of the construction on time.'
This means that relevant checkpoints have been identified and the purpose of the milestone is understood.
'We were late producing the specification but have caught up by leaving some of the lower priority things till the next release.'
This means that the specification (for the current release) has not been approved. Therefore the milestone to start development has not been achieved. An exception plan (see Chapter 6 of the subject guide) is needed.
'We're three months into a twelve month project with a fixed team, so we must be 25 per cent finished by now.'
This is frequently an indication that there are no milestones, no resource plan and therefore no way of managing the transitions between activities.

Delegated responsibility and exception-based reporting

The example above does not consider how an 'early-warning' system can be put in place. The PRINCE2 approach is to delegate responsibility for monitoring progress on a product to the team actually doing the work. During planning, agreement is reached on the resources to be made available, the time available for doing the work and the quality of the end results – within specified tolerances. From that point on, the project manager only needs to know two things:

The **status** of the work (hopefully complete) when the milestone date is reached.

Early warning that the product is going out of tolerance for time, cost or quality.

The database task is currently three days, or 15 per cent, late. This is probably within tolerance. The business rules team, however, should have been aware by the end of, say, the third week of their design task, that they were at risk of over-running (and probably running out of resources). This should have raised an exception condition for the sub-system.

That is the limit of delegated responsibility. The project manager is then able to take action to rectify or normalise the situation, which would depend on the circumstances of the rest of the project but could be one of the following:

- Revise the delivery date for the sub-system.
- Try to honour the original schedule by assigning extra resource to the business rules team.
- Redefine the specification for the sub-system to reduce the work needed.

Once again, the project manager is in place to consider and strike a balance between time, cost and quality.

Accuracy of projections

Boehm (1995) observed a large variation (a factor of 4) in the initial estimated cost of a wide range of projects. One of the goals of a project manager is to reduce that variation, if not immediately then at least before the end of the elaboration phase of the project. A series of quality gates can be implemented, at which the uncertainty should be reduced significantly. For example, the completion of the initial requirements capture could be the first gate and aim to halve the uncertainty. This is illustrated in Figure 3.6 below, and forms a major part of risk management.

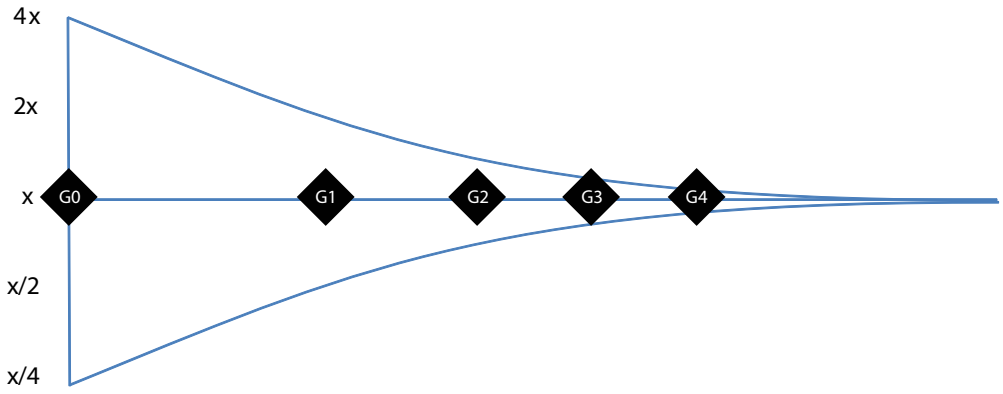


Figure 3.6: Controlling estimation error.

Reminder of learning outcomes

Having completed this chapter, and the Essential reading and activities, you should be able to:

- describe the structure of a project schedule
- apply resources to execute a project plan
- understand the constraints on the accuracy of resourcing and budgeting.

Test your knowledge and understanding: The PRINCE2 approach

Several formal project management methodologies have been introduced in this chapter. These are mostly intended for use in large projects but we have focused on PRINCE2 because it is scalable to large and small projects and because of the availability of resources such as CC Training and ProjectInABox. It is important that you can demonstrate an understanding of this scalability.

PRINCE2 is not covered in either Pressman or Sommerville, but free web-based resources are available. We have identified several in this chapter (under the heading 'Project management perspective') and introduce exception-based reporting (see section 'Delegated responsibility and exception-based reporting'). A more detailed explanation of the methodology is presented in Chapter 6 (Figure 6.2).

There are several other key concepts such as dividing a project into stages with detailed Stage Plans and delegated responsibility through the use of Work Packages (discrete well-defined products).

Read about the principles of the methodology from a source such as CC Training (www.crazycolour.com/prince2/?title=Main_Page) and consider how it corresponds to the process models you have been studying. As you explore this, draw up two lists from the material you read – one of concepts that are familiar and one of concepts that are less familiar. This second list is likely to consist of some of the formal methods used in larger projects, such as managing stage boundaries and closing a project.

Summary of the inception phase

- SE processes include addressing capture and specification of the system needs (requirements engineering), translation of those needs into an executable software system (design and implementation) and checking that the result addresses the specification and meets the real needs of the users (validation).
- The way these processes interact can be represented in the form of process models and the RUP is a generic modelling tool.
- Requirements for systems evolve and the SE processes must be able to cope with change.
- Agile methods identify and respond to requirements incrementally, using short development cycles. XP uses test driven development and involves the customer as part of the team. Scrum provides a priority-driven project management framework.
- Agile is a valid alternative to a conventional plan-driven approach as long as it is understood by the development team, acceptable to the organisation and scaleable to address the communication needs of large development teams.
- A software requirements document is an agreed and normally contractual statement of functional requirements (what the system should do) and non-functional requirements (constraints to be imposed on the system).
- The requirements engineering process includes an iterative and validated process of discovery, analysis and documentation of changing customer requirements.

By now, you should be confident that you would be able to:

1. Define the scope of a project in terms of core requirements, key features and the main constraints and risks.
2. Produce a plan describing the development process that you would follow, supported by estimates of cost and schedule in line with the client's business case.

Appendix 2: Revision support

Collected together below in Appendix 2 are what you should have gained from each main chapter, in terms of your abilities to carry out these tasks and activities, and to understand the processes underlying them in terms of the four main phases.

This is followed in Appendix 3 by the collected set of all cornerstone points made in the subject guide to aid revision. Appendix 4 contains 'Revision guidance notes', while Appendix 5 contains a Sample examination paper. Finally, Appendix 6 contains an outline marking scheme, but no sample answers.

Summary of main processes in the four phases

Inception phase	Scoping and justifying a project.
Elaboration phase	Defining a response to stakeholder needs.
Construction phase	Managing the provision of the functionality agreed.
Transition phase	Producing a product release.

1. Inception phase

- Define the scope of a project in terms of core requirements, key features and the main constraints and risks.
- Produce a plan describing the development process that you would follow, supported by estimates of cost and schedule in line with the client's business case.

Software processes

- Understand the concept of a process model as a way of describing a software development project and differentiate between common approaches such as waterfall, iterative and agile.
- Identify the key artefacts produced during an iteration of the software development process.
- Understand the role of the project manager in delivering those artefacts.
- Understand the ways in which the development process can affect quality.

Requirements engineering

- Describe the differences between elicitation, specification, validation and documentation of requirements.
- Understand the differences between system and user requirements and between functional and non-functional requirements.
- Use natural language and graphical representations of requirements within a requirements document.
- Produce a system model based on requirements.

Planning, cost and schedule estimation

- Describe the structure of a project schedule.
- Apply resources to execute a project plan.
- Understand the constraints on the accuracy of resourcing and budgeting.

2. Elaboration phase

- Validate that the current project plan and architecture design are complete and able to deliver the project vision and requirements.
- Demonstrate that major risks have been identified and that each is being managed appropriately.

Risk management

- Describe the steps required in order to understand how a risk should be mitigated, monitored and managed.
- Give examples of risks created by the use of technical and human resources, the way the client and developer organisations work and performance-related risks.
- Identify probable causes of risks in those five areas and ways in which pro-active risk management can be used to reduce the probability and/or impact.

Architecture, modelling and design

- Use UML graphical models to represent the context (external environment) in which the system will operate, the interactions between the system and that environment, the static and dynamic structures of the system and its responses to stimuli.
- Make informed decisions during the architectural design process.
- Base a system architecture design on a generic architectural pattern where appropriate.

Managing the execution of the project plan

- Describe the main functions of a project manager.
- Understand the dynamics of a development team.
- Identify the tools and infrastructure required for the development environment.
- Describe the roles and responsibilities for configuration management.

3. Construction phase

- Validate that the system being developed is fit for purpose.
- Be able to deploy it for beta-testing with the target user community.
- Provide those users with the necessary documentation and support, including training.

Detailed design

- Apply object-oriented principles in the detailed design phase.
- Describe the use of UML documentation for decomposing the architectural design into usable object-oriented software.
- Explain the purpose of design patterns and give examples of design patterns for data-processing, process-control and e-commerce applications.

Quality management

- Define quality as it applies to software engineering and identify the key quality-related activities that take place at each phase of the project lifecycle.
- Explain the advantages and limitations of qualitative and quantitative quality measures and give examples of each.

- Explain the difference between verification and validation and the purpose of testing at the sub-function, component, system and user levels.

4. Transition phase

- Validate that the objectives for this iteration of the system have been met and that all artefacts are complete.
- Initiate the planning of another iteration of system development.
- Apply lessons learned.

Evolution

- Describe the key decisions that have to be taken when scheduling the next iteration of a system.
- Explain how process effectiveness can be measured and improved with explicit reference to GQM and CMMI.

Notes

Appendix 3: Cornerstones

CORNERSTONE 1

Project management is a creative activity. It requires a combination of knowledge, insights and skills in order to deliver the required outcomes for each unique project. If you do not take into account the specific challenges and rely instead on a single standard approach, you are not a project manager, you are a project administrator.

CORNERSTONE 2

Project management is about controlling the strengths and weaknesses of the project team (and enterprise) and the opportunities and threats that can arise and lead to issues for the project, the 'internal' and 'external' factors of a SWOT analysis.

CORNERSTONE 3

A process model is a high-level and incomplete description of the way a project is controlled and how it evolves. It introduces interfaces between actors and workflows and transitions between activities. Your choice of model does not fundamentally affect WHAT you are going to do; customers still need to know that their requirements are the reference point for design and code still needs to be tested before and after integration. What the model allows you and your team to do is focus on the important transitions during the project.

CORNERSTONE 4

The purpose of a software project is to meet a client's need by delivering a software system. That system decomposes into sub-systems and components, and each component requires a number of products to be built, tested and integrated. There are two approaches that a project manager can adopt – focus on the product, or focus on the work required to deliver that product.

CORNERSTONE 5

A best practice in common use is the breakdown structure. A Work Breakdown Structure, for example, decomposes the project into phases, decomposes phases into activities and activities into tasks and sub-tasks. You can describe dependencies from one to another through a Work Flow Diagram (e.g. a GANTT chart). Similarly, you can represent the relationships between artefacts from a project in a Product Flow Diagram, which then defines the order in which they must be completed. You can also use breakdown structures to ensure that an analysis of factors such as risks or competences is comprehensive.

CORNERSTONE 6

A specification document must comply with the 'four C's':

Clear: No ambiguities or confusing cross references to other documents.

Concise: No unnecessary detail, repetition or confusing cross-references to other documents.

Correct: Attributed, with no conjecture, paraphrasing or unsubstantiated opinion.

Complete: Verified as part of the V & V process.

CORNERSTONE 7

The project team invariably focuses on the outputs it is to create, while the client is concerned with the outcomes of using those outputs. 'Outcome-based specifications' address the client's needs and do not prescribe **how** they are to be met.

CORNERSTONE 8

Functional requirements describe the way the system behaves.

Non-functional requirements describe the context within which that system behaviour is delivered.

User requirements are the functional and non-functional requirements that describe the business goals for the system.

System requirements are the functional and non-functional requirements that have to be met in order to achieve the business goals for the system.

CORNERSTONE 9

ISO/IEC 12207 defines validation as confirmation that the final, as-built, system fulfils its specific intended use, that it is 'fit for purpose'. Verification confirms that the output of each activity (such as requirements capture, design, code, integration or documentation) fulfils the requirements or conditions imposed by preceding activities.

This is often expressed in terms of validating that the 'right thing' is being built and verifying that the thing is being 'built right'.

CORNERSTONE 10

The outcome of a risk analysis should reflect the status of the project AFTER the mitigation has been brought into the project plan, in other words the risk register reflects the residual risk only. This implies that risk assessment is an iterated or continuous process in order to re-evaluate mitigated risks.

CORNERSTONE 11

When deciding whether to use the UML during architectural design, you need to be able to answer yes to the following five questions.

1. Do we have good tools for doing things this way?
2. Do we understand how to use them?
3. Do we know what we are trying to achieve?
4. Will we document the decisions taken clearly?
5. Will the client understand the result?

CORNERSTONE 12

Formal documents are critical. They tend to be kept (if not actually read) and they tend to change over time. Do not simply flag the changes by giving a document a different date in its filename or by changing the name from V0.2 to V0.3. Give formal documents a history section and keep them (without ever changing the filename) in a document archive such as Sharepoint. The LEAN philosophy acts as a good guide: if you are not going to keep it, don't bother updating it. Conversely, if you do not maintain it, there is little point in keeping it!

CORNERSTONE 13

'A team is not a bunch of people with job titles, but a congregation of individuals, each of whom has a role which is understood by other members. Members of a team seek out certain roles and they perform most effectively in the ones that are most natural to them.' (Dr R.M. Belbin)