Software Project Management: Methodologies & Techniques

SE Project 2003/2004 group E

17th September 2004

SE Project 2003/2004 group E
Software Engineering Project (2IP40)
Department of Mathematics & Computer Science
Technische Universiteit Eindhoven

Rico Huijbers	0499062
Funs Lemmens	0466028
Bram Senders	0511873
Sjoerd Simons	0459944
Bert Spaan	0494483
Paul van Tilburg	0459098
Koen Vossen	0559300

Abstract

This essay discusses a variety of commonly used software project management methodologies and techniques. Their application, advantages and disadvantages are discussed as well as their relation to each other.

The methodologies RUP, SSADM, PRINCE2, XP, Scrum and Crystal Clear are discussed, as well as the techniques PMBOK, COCOMO, MTA, EV and Critical path.

Contents

1 Introduction

There exist numerous project management methodologies and techniques in the world. This essay aims to make a selection and present an overview of the commonly used methodologies and techniques. Except for the usage also the availability of information and resources on the subject was a factor in selecting the methodologies and/or techniques.

This essay gives a global description of each of the selected methodologies, discusses their key points, advantages and disadvantages. Finally possible couplings with other discussed methodologies are considered. Each technique is discussed the same way, except that instead of the coupling, possible usages of the technique in other methodologies are considered.

The next section explains common terms, definitions and acronyms used throughout this document. Section 3 discusses a number of thin and thick project management methodologies. Section 4 will elaborate on some project management techniques and their usage in the discussed methodologies. This document will be concluded in the last section, Section 5.

2 Preliminaries

This section provides an overview over commonly used terms, definitions and acronyms used throughout the document.

2.1 List of definitions

Since there seems to be a lot of confusion, or at least different opinions, concerning the meaning of the following definitions, they are defined here as they are used in the remainder of this document.

- **Methodology:** A codified set of recommended practices, sometimes accompanied by training materials, formal educational programs, worksheets and diagramming tools.
- **Thick methodology:** A methodology that includes a large amount of formal process paperwork and documentation
- **Thin methodology:** A methodology that eschews formal process paperwork and documentation.
- **Project management:** The process of planning, organising, staffing, directing and controlling the production of software.
- **Technique:** A way of efficiently acquiring information of a software project in a manner that is not immediately obvious or straightforward.

The definitions are taken from [METH-WKP] but can also be found in [FOLDOC] and much more other sources.

2.2 List of acronyms

BSO	Business Systems Options
CASE	Computer Aided Software Engineering
CCTA	Central Computing and Telecommunications Agency
COCOMO	Constructive Cost Model
CPM	Critical Path Method
DFD	Data Flow Diagram
DFM	Data Flow Modelling
ELH	Entity Life History
EM	Entity/Event Modelling
EV	Earned Value
LDM	Logical Data Modelling
LDS	Logical Data Structure
MTA	Milestone Trend Analysis
OGC	Office of Government Commerce
PB	Project Board
PERT	Program Evaluation and Review Technique
PID	Project Initiation Document
PM	Project Management
PMBOK	Project Management Body of Knowledge
PPR	Post Project Review
PPRP	Post Project Review Plan
PRINCE	Projects in Controlled Environments
RUP	Rational Unified Process
SSADM	Structured Systems Analysis and Design Methodology
UML	Unified Modeling Language
XP	eXtreme Programming

3 Methodologies

In this section, we highlight a number of commonly used Software development methodologies. We consider both thick and thin (also called 'lightweight') methodologies. In this essay, the thick methodologies we consider are RUP, SSADM and PRINCE2. XP, SCRUM and Crystal Clear are considered as thin methodologies.

When discussing each methodology, we will focus on the management and business aspects of the methodology.

3.1 Rational Unified Process (RUP)

The Rational Unified Process (RUP) is a software design methodology created by the Rational Software Company. The Rational Software Company was acquired by IBM in 2003. RUP is a thick methodology; the whole software design process is described with high detail. RUP is hence particularly applicable on larger software projects. The RUP methodology is general enough to be used out of the box, but the modular nature of RUP—it is designed and documented using Unified Modeling Language (UML)—also makes it easy to adapt the methodology to the special needs of a single project or company.

One of the major differences between RUP and other methodologies like SSADM (see Section 3.2) is that RUP doesn't use a waterfall approach for software development. The phases of requirements, analysis, design, implementation, integration and testing are not done in strict sequence. In RUP, an iterative approach is used: a software product is designed and built in a succession of incremental iterations. Each iteration includes some, or most, of the development disciplines (requirements, analysis, design, implementation, testing, and so on). Figure 1 shows one iteration of a RUP project in a graphical way.

For more detailed descriptions of RUP, see [RUP-WEB] and [RUP-BOOK].

Application area

Due to the modular nature of RUP, it can be used for all sorts of software projects. It is even possible to use RUP for non-software projects. However, because of the complexity of the RUP methodology, it is used mostly for larger software projects.

Advantages

- The iterative approach leads to higher efficiency. Testing takes place in each iteration, not just at the end of the project life cycle. This way, problems are noticed earlier, and are therefore easier and cheaper to resolve. When using a waterfall approach, it can happen that, for example, software programmers have to wait for the completion of the design phase before starting implementing and integrate the design. Designing and

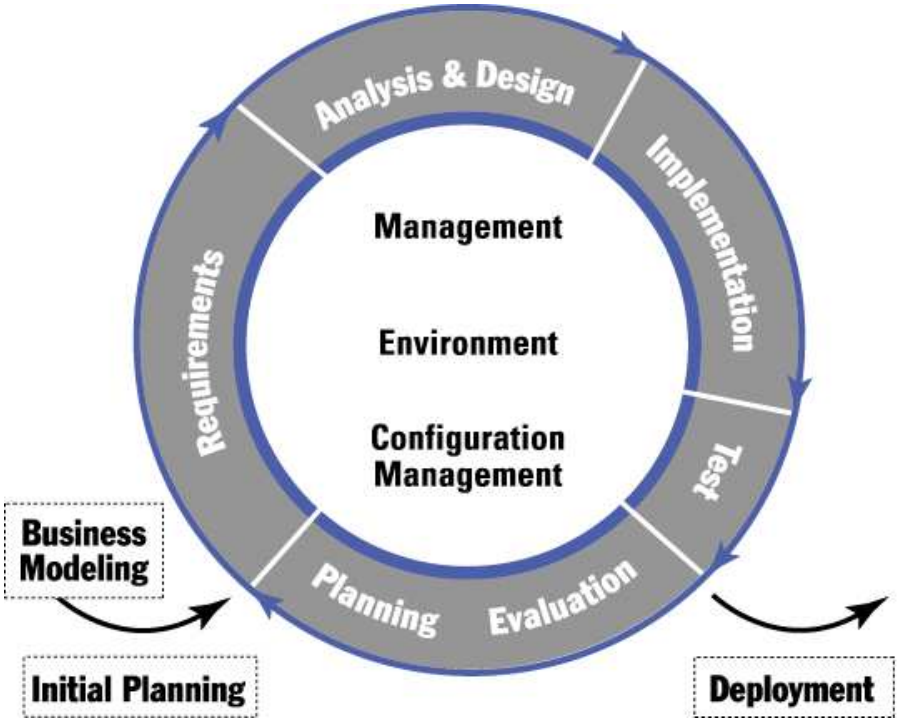

Figure 1: The iterative approach to software development [RUP-IMG]

building a software project with an iterative approach solves this problem. Integration and implementation will not only happen at the end of the project, but in every iteration. This saves time, since more team members can work more of the time.

- Managing changes in software requirements will be made easier by using RUP. Unless a software project is very small, it is nearly impossible to define all the software requirements at the beginning of a project. It will almost always take more than one step to know what the final software product will look like, for the customer as well as for the project members. Developing with iterations makes this process of changing, "creeping" requirements, that often leads to missed schedules and dissatisfied customers, less troublesome.

- RUP itself is software, too, and is distributed in an electronic and online form. Team members don't need to leave their computers for RUP related activities. No more searching in big, dusty books. All information about the software development methodology is available at the project members' fingertips. Also, the newest version of RUP is always present on the computer of each team member. And even more important, it makes sure that every team member is using the same version of RUP. RUP is designed and documented using UML, in an object oriented way. This makes it easy to adapt RUP to the special needs of a single project or organization.

Disadvantages

- RUP is a commercial product, no open or free standard. Before RUP can be used, the RUP has to be bought from IBM, as an electronic software and documentation package (a trial version can be downloaded from the IBM website, however). The RUP only exists in an electronic form, which can sometimes limit its use.
- RUP, as said before, describes the whole software design process with high detail; it is a very complex methodology, difficult to comprehend for both project managers and project members. Therefore, it is not the most appropriate software design methodology for most small projects.
- Starting to use RUP as software development methodology is difficult. Everyone participating in the project will have to learn working with RUP. For details about applying RUP on projects, see [RUP-TRANS].

Coupling with other methodologies

RUP is a thick methodology, and coupling with other methodologies is therefore not always possible. There is a way to couple the Crystal Clear methodology with RUP. See Section 3.6 for more information.

3.2 SSADM

Structured Systems Analysis and Design Methodology (SSADM) is a widely used computer application development method in the UK. Just like PRINCE (see Section 3.3), its use is often specified as a requirement for government computing projects. Today it is increasingly being adopted by the public sector in Europe.

> "SSADM has been used by the government in computing since its launch in 1981. It was commissioned by the Central Computing and Telecommunications Agency (CCTA) in a bid to standardize the many and varied IT projects being developed across government departments. The CCTA investigated a number of approaches before accepting a tender from Learmonth & Burchett Management Systems to develop a method." [SSADM-GUI]

Since 1981 SSADM has been further developed and refined and in 1990 version 4 of it was launched. SSADM is an open standard, which means that it is freely available for use in industry and many companies offer support, training and Computer Aided Software Engineering (CASE) tools for it.

In detail, SSADM sets out a cascade or waterfall view of systems development, in which there are a series of steps, each of which leads to the next step (see the model in Figure 2). SSADM's steps are [SSADM-INTR]:

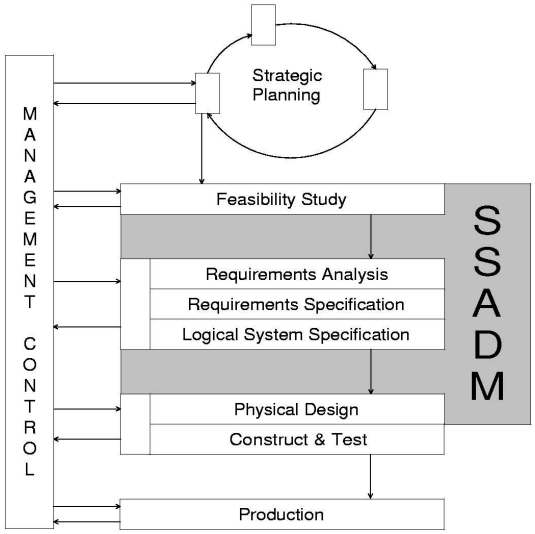

Figure 2: SSADM—Process Model [SSADM-WEB]

1. **Feasibility Study:** The feasibility study consists of one single stage, which involves conducting a high level analysis of a business area to determine whether a system can cost effectively support the business requirements. In the Feasibility Study an overview Data Flow Diagram (DFD) is produced together with a high level Logical Data Structure (LDS). At this stage the DFD will represent the existing system and the LDS may be incomplete and contain unresolved many-to-many relationships.

2. **Requirements Analysis:**

   - **Investigation of the current environment**—During this stage the systems requirements are identified and the current business environment is modelled in terms of the processes carried out and the data structures involved. In this DFDs and LDSs are used to produce detailed logical models of the current system.
   - **Business Systems Options (BSO)**—During this stage up to six business system options are produced and presented. As a result one of these options is adopted and refined. DFDs and LDSs are produced to support each business system option and the final chosen option. The transition from the former stage to this stage is a key part of SSADM: this is where we move from a logical model of the current system to a logical model of the required system. This means that here the DFDs and LDSs have to be refined to cater to new or changed requirements.

3. **Requirements Specification:** The Requirements Specification consists of a single stage which involves further developing the work carried out in the Requirements Analysis: detailed functional and non-functional requirements are identified and new techniques are introduced to define the required processing and data structures.

4. **Logical System Specification:**
   - **Technical system options**—In this stage up to six technical options (specifying the development and implementation environments) are produced, one being selected.
   - **Logical design**—In this stage the logical design of update and enquiry processing and system dialogues (menus etc.) is carried out.
5. **Physical Design:** The Physical Design consists of a single stage in which the logical system specification and technical system specification are used to create a physical database design and a set of program specifications.

SSADM revolves around the use of three key techniques:

1. **Logical Data Modelling (LDM):** This is the process of identifying, modelling and documenting the data requirements of a business information system. A LDM consists of a LDS and the associated documentation. LDSs represent Entities (things about which a business needs to record information) and Relationships (necessary associations between entities).
2. **Data Flow Modelling (DFM):** This is the process of identifying, modelling and documenting how data flows around a business information system. A Data Flow Model consists of a set of integrated DFDs supported by appropriate documentation. DFDs represent processes (activities which transform data from one form to another), data stores (holding areas for data), external entities (things which send data into a system or receive data from a system and finally data flows (routes by which data can flow).
3. **Entity/Event Modelling (EM):** This is the process of identifying, modelling and documenting the business events which affect each entity and the sequence in which these events occur. An EM consists of a set of Entity Life Historys (ELHs) (one for each entity) and appropriate supporting documentation.

Application area

SSADM was originally developed to standardize the many and varied IT projects being developed across government departments. Today, SSADM Version 4, can be used in all kinds of analysis and design stages of system development.

SSADM can be used for practically any size of project: small (1–2 persons, less than one man year), medium (4–10 persons, 1–20 man years) and large projects. Furthermore SSADM can be used to develop new projects, but it can also be used to maintain existing systems. [SSADM-WEB]

Advantages

- As mentioned before SSADM is an open standard, which means that it is freely available for use in industry and many companies offer support, training and CASE tools for it.

- SSADM divides an application development project into modules, stages, steps, and tasks, and provides a framework for describing projects in a fashion suited to managing the project.
- SSADM's objectives are to:
  - Improve project management and control
  - Make more effective use of experienced and inexperienced development staff
  - Develop better quality systems
  - Make projects resilient to the loss of staff
  - Enable projects to be supported by computer based tools such as computer aided software engineering systems
  - Establish a framework for good communications between participants in a project
- SSADM can reduce the chances of initial requirements being misunderstood and of the systems functionality straying from the requirements through the use of inadequate analysis and design techniques.

Disadvantages

- SSADM is a typical example of a structured methodology, which means that the purpose of it is to:
  - Formalize the requirements elicitation process to reduce the chances of misunderstanding the requirements.
  - Introduce best practice techniques to the analysis and design process.
- As mentioned before SSADM can reduce the chances of initial requirements being misunderstood and of the systems functionality straying from the requirements through the use of inadequate analysis and design techniques. However, SSADM assumes that the requirements (in the form of an agreed requirements specification) will not change during the development of a project. Following each step of SSADM rigorously can be time consuming and there may be a considerable delay between inception and delivery (which is typically the first time the users see a working system). The longer the development time the more chance of the system meeting the requirements specification but not satisfying the business requirements at the time of delivery.

Coupling with other methodologies

SSADM covers those aspects of the life-cycle of a system from the feasibility study stage to the production of a physical design; it is generally used in conjunction with other project management methods or techniques, which are concerned with broader aspects of project management. So almost every method or project technique discussed in this document can be used in combination with SSADM, but because both methods are developed in the UK, very often PRINCE is used (see Section 3.3). [SSADM-WI]

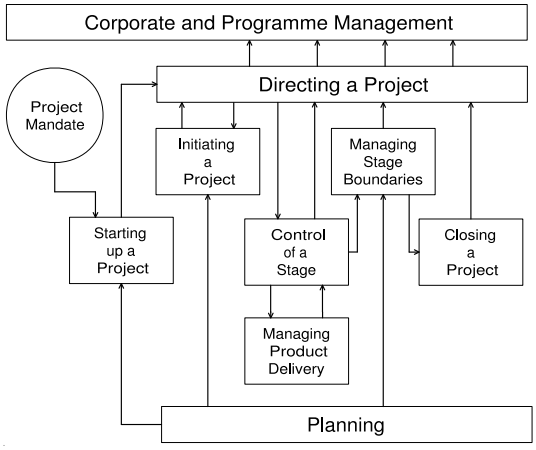

Figure 3: PRINCE2—Process Model [PRIN2-OFF]

3.3 PRINCE2

Projects in Controlled Environments (PRINCE) is a project management methodology covering the organization, management and control of projects. A project has a clear beginning, middle and end, a clear organizational structure and defined objectives. You can use a managing methodology like PRINCE to ensure that a project is successful, which means that it finishes on time, within budget and provides the customer with what they have asked for.

PRINCE was first developed by the CCTA, which is now part of the Office of Government Commerce (OGC), in 1989 as a UK Government standard for IT project management.

Since its introduction, PRINCE has become widely used in both the public and private sectors and is now the de facto standard for project management in the UK. Although PRINCE was originally developed for the needs of IT projects, the methodology has also been used on many non-IT projects. The latest version of the methodology, PRINCE2, is designed to incorporate the requirements of existing users and to enhance the methodology towards a generic, best practice approach for the management of all types of projects.

PRINCE2 is a process-based approach for project management providing an easily tailored and scalable methodology for the management of all types of projects. Each process is defined with its key inputs and outputs together with the specific objectives to be achieved and activities to be carried out. The methodology describes how a project is divided into manageable stages enabling efficient control of resources and regular progress monitoring throughout the project. The various roles and responsibilities for managing a project are fully described and are adaptable to suit the size and complexity of the project, and the skills of the organization.[PRIN2-OFF]

PRINCE2 outlines eight processes that are required to successfully carry out a project (also see the process model in Figure 3). These are [PRIN2-WEB]:

1. **Starting up a Project:** To make sure that the project has a very clear beginning, this process occurs even before the project has actually started. All decision making persons have to come together and will appoint a Project Manager. Together they will discuss the project and outline reasons for it and how decide how the project is to be carried out. All this information will be put together in a 'Project Brief'.

2. **Initiating a Project:** Before a project can be approved during the 'Directing a Project' process it must be carefully planned to ensure that it meets its objectives. Detailed estimations of costs, needed time and other resources have to be made and these are put together by the Project Manager into a so called Project Initiation Document (PID) for approval by the Project Board (PB).

3. **Directing a Project:** After the Project Brief and the PID have been put together, the project has to be approved by a group of senior managers, called the Project Board (PB). During the rest of the project this PB has the overall responsibility for the success of the project whereas the Project Manager has the day to day responsibility. He will inform the PB about the project's progress with the help of regular reports.

4. **Controlling a Stage:** One of the advantages of PRINCE2 is that projects are divided into manageable stages to ensure the project remains manageable and controlled. How many stages are used, will depend on the size of the project and the level of risk. In PRINCE2 each project stage must be completed before the next stage can be started and each new stage is planned in the stage proceeding it. Also the Stage Plans will be approved by the PB to help ensure that the project remains within budget and delivers its objectives.

5. **Managing Stage Boundaries:** This process involves preparing for the next stage and reviewing the current stage. The Project Manager makes suggestions to the PB about the likelihood of the project achieving its business objectives and any changes in the business case, project plan, risks and issues. When a project has clear stage boundaries it can be easily controlled and managed by permitting the project to continue only once the PB is satisfied with the current stage end and next stage plan.

6. **Planning:** Each project plan, stage plan and team plan must consider key planning aspects. These include what products to produce, the activities required to produce these products, estimated resources (including costs and time), scheduling the activities and analyzing risks. By following the PRINCE2 planning process all these points are conducted in a sensible, logical sequence. Ensuring consistency enables plans to be compared and streamlines the planning process.

7. **Managing Product Delivery:** The goal of a PRINCE2 project is to deliver products. A product can be a physical thing such as a poster or it could be an intangible deliverable such as a service or sales agreement. In fact everything produced in PRINCE2 (even a document) is called a product. Often a Project Manager does not create the product. A third party supplier and/or their colleagues may do some or all of the work. It is

the Project Manager's responsibility to ensure that the supplier produces the correct products at the right time by providing a description of the work to be done.

8. **Closing a Project:** At the end of the project, after its products have been delivered, the project is closed down with approval of the PB. The Project Manager plans what will be done to evaluate the project's outcome, which is called the Post Project Review (PPR). A controlled close down is in effect the last demonstrable PRINCE2 project action. Any lessons learned are recorded, resources are released and the Post Project Review Plan (PPRP) is created.

The key concepts that are fundamental to PRINCE2 are:

- **Control:** Being able to control your project is key to its success. For this reason PRINCE2 breaks down projects into easily managed stages, essentially breaking a large project into 'bite size chunks'.
- **Quality:** To ensure that a product (or service) meets the customer's quality expectations these must be defined and agreed when a project is being planned.
- **Planning:** In PRINCE2 planning does not end once the project has started. Of the eight PRINCE2 processes all but one involve planning, even the final process.
- **Lesson Learned:** Every time we carry out a project we learn something. All lessons, mistakes, ideas or successes are captured in the Lessons Learned Log. At the end of the project these are collated into a Lessons Learned Report, allowing others in the organization to benefit from them.

Application area

PRINCE2 is a project management methodology owned and maintained by the OGC in the UK. It summarizes best practice from a variety of industries and backgrounds. PRINCE2 has been adopted by the National Health Service as its preferred methodology and a number of governments world wide are looking at adopting it as their standard project management methodology. PRINCE2 is one of the few government standards that has grown organically to be adopted by both private and public organizations.

Some organizations that use PRINCE2 as a project management methodology are the UK Police Forces, Rolls Royce, the British Medical Association, Norwich Union, the UK Department of Justice and London Underground (see also [PRIN2-WEB]).

Advantages

Besides the key concepts of PRINCE2 mentioned in Section 3.3, there are some other advantages of the use of PRINCE2:

- PRINCE2 is a structured methodology providing organizations with a standard approach to the management of projects. The methodology embodies proven and established best-practice in project management. It is widely recognized and understood, and so provides a common language for all participants in the project, also PRINCE2 is very useful for educative use.
- PRINCE2 enables projects to have:
  - A controlled and organized start, middle and end
  - Regular reviews of progress against plan and against the Business Case
  - Flexible decision points
  - Automatic management control of any deviations from the plan
  - The involvement of management and stakeholders at the right time and place during the project
  - Good communication channels between the project, project management, and the rest of the organization
- Because there are no chapters on test methods in the PRINCE2 handbook, the choice which test method you want to use, is absolutely without restrictions. PRINCE2 gives you a free choice of test method, but does ask attention for it when putting together the project quality plan, which is part of the PID.

Disadvantages

- Every person who works on a PRINCE2 project should be quite familiar with every aspect of PRINCE2 to know how to play the game. It often happens that this is not the case, because it is very expensive to give everyone involved a course to study PRINCE2.
- Using PRINCE2 means that a lot of documents and lists have to be written, and because Project Managers only have to inform the PB about the status of the project, when something goes wrong they can easily blame others. It is also very easy to blame other project groups when something goes wrong. This leads to the so called “cover your ass behavior”.
- Splitting up a PRINCE2 project often results in a lack of knowledge of the project by responsible persons like the Project Manager. Also it’s not useful to make use of expensive Project Managers when the only work they have to do is to administrate and inform the PB.

Coupling with other methodologies

PRINCE2 is a very complete project management methodology and it covers a project from the very beginning till the delivery of the final product. So PRINCE2 does not need to be coupled to other methodologies or project techniques to complete PRINCE2 itself, though often it is used as an addition to meet with the weaknesses and incompleteness of SSADM.

3.4 eXtreme Programming (XP)

XP is a software engineering methodology that has been formulated in 1996 by Kent Beck. It is explained in detail in [XP-BOOK]. XP has received fair media attention, and is most renowned for its practices that are sometimes regarded as controversial, such as pair programming and test-driven development. In this document, we will not concern ourselves with these aspects of eXtreme Programming, but instead we will focus on the management part.

Principles of XP

XP aims to reduce the risk involved in software development.

In particular, it aims to reduce the cost of delaying design decisions. In [XP-BOOK, p. 11–14], Beck gives a treatment of the cost and revenues of design decisions and feature implements (which he calls 'options'), and he concludes that it is more beneficial to delay options of which it is uncertain whether they will generate revenue (i.e. there is a certain amount of risk involved in implementing the option).

Traditionally, the cost of making decisions about (and therefore changes to) a software project would rise exponentially during the course of development. It would therefore be costly to defer options, because implementing them later on might be too costly, and possibly even cost more than the value of the option would be.

XP reduces the cost of making modifications later on during development, and thereby allows decisions that entail high risk to be deferred until a sound judgement can be made on them. All practices of XP work together to achieve this goal.

Planning

An XP project is made up of *releases*. The first release aims to produce an initial, working version of the product. The subsequent releases add functionality to the project, change behaviour and fix bugs. An XP project typically lasts the entire lifetime of the application: the software is constantly tweaked and updated to be as useful as possible. Of course, this approach is not required: the project can be ended when the customer decides the product is 'finished'. There is typically one to three months time between releases. Each release is divided up *iterations*. Usually, they are one to three weeks in length.

In an XP project, requirements are not fixed in advance. At the start of the project, or whenever he can think of one, the customer writes down a desired feature in a so–called *user story*, which clarifies the feature by means of a typical 'use case'.

At the start of the project, a *release plan* is drawn up. First, all stories are written by the customer. The development team then assigns a cost to each story. This cost should be one, two or three 'ideal programming weeks' for a single developer. If the cost is greater, the story should be split up. If the

cost is less, multiple stories should be merged together. The stories are then divided over a number of releases. Release dates can then be calculated from the stories assigned to each release, or the stories can be divided such that fixed release dates will be met. For this, you will need to estimate how to convert ideal programming weeks to calendar weeks. The first time, this will be hard (and estimates might be off), but as the project progresses the estimates will become better.

At the start of each iteration, the customer selects the stories from that release that are of the greatest value to him, which will be implemented during the iteration. The stories are then broken up into smaller units called *tasks*. Each developer has the opportunity to assume responsibility for a number of tasks. The tasks are then estimated, in ideal programming days, by the developer that chose them (making sure no–one has too much or too little to do), and implemented during the course of the iteration.

Experience from an iteration or a release, such as the ideal programming time realized, can be taken into account to estimate better next release or iteration planning.

Application area

From [XP-BOOK, p. XV]:

> "XP is a lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements."

XP is a good choice when requirements are unclear (which might occur when because himself does not know exactly what he wants), or prone to change (because of changing business situations, or as a result of external conditions). Because in XP the development of a product is divided into many small cycles, and each cycle is planned separately, changes to the planning can be made constantly, quickly and easily.

Team size is an issue when implementing XP. XP is meant for small-to- medium-sized teams. In practice, this means that teams should be maximum ten people. A few more is probably okay, but twenty is too many ([XP-BOOK, p. 157]).

Advantages

- An XP project is very malleable. A usable product can be released very quickly, at which point the business can already take advantage of the product, and the product can and will be improved continually thereafter, with feedback that stems from live use. Especially when the project is exploratory for the customer as well, having feedback from live use and adapting to changing minds, wishes and circumstances can be invaluable.
- Additionally, the process is very transparent. Progress, position and direction of the project are very transparent, which will make management happy as well.

Disadvantages

- The biggest roadblock to implementing XP in any given environment, will usually be the customer. The 'customer', or a person that plays the role of the customer, has to be an integral part of the development team. This means that the customer will have to be available on-site at all times. Sometimes, this is just not feasible, or the customer will refuse to assign an employee to the development team full-time. In such cases, XP will not be able to work properly, and should be abandoned.

Coupling with other methodologies

Beck is very liberal about adapting XP. He suggests experimenting with the practices prescribed by XP, then keeping the ones that work, and adjusting or even ditching the ones that don't.

However, he goes on theorizing that the maximum benefit of XP can only be achieved by putting all practices into effect. He calls this the 20–80 rule: if you do 80% of the work, you will only see 20% of the benefit. You will need to use all the practices to get all of the benefit.

This suggests that coupling XP with other methodologies will be detrimental to its usefulness, or the coupling should be limited to *adding* practices to XP, without removing any.

3.5 Scrum

From [SCRUM-WKP]:

> "Scrum is an agile method for project management, in use since at least 1990. It has been called a "hyper-productivity tool", and has been documented to drastically improve productivity in teams previously paralyzed by heavier methodologies—quickly producing results where there had been little or none."

Scrum uses the following concepts:

- **Sprint:** A period of 30 days or less where a set of work will be performed to create a deliverable
- **Backlog:** All work to be performed in the foreseeable future, both well defined and requiring further definition.
- **Sprint backlog:** The work that should be done during the current sprint.
- **Product backlog:** The work that should be done for the whole product as desired by the customer.
- **Scrum:** A daily meeting at which progress and impediments to progress are reviewed.

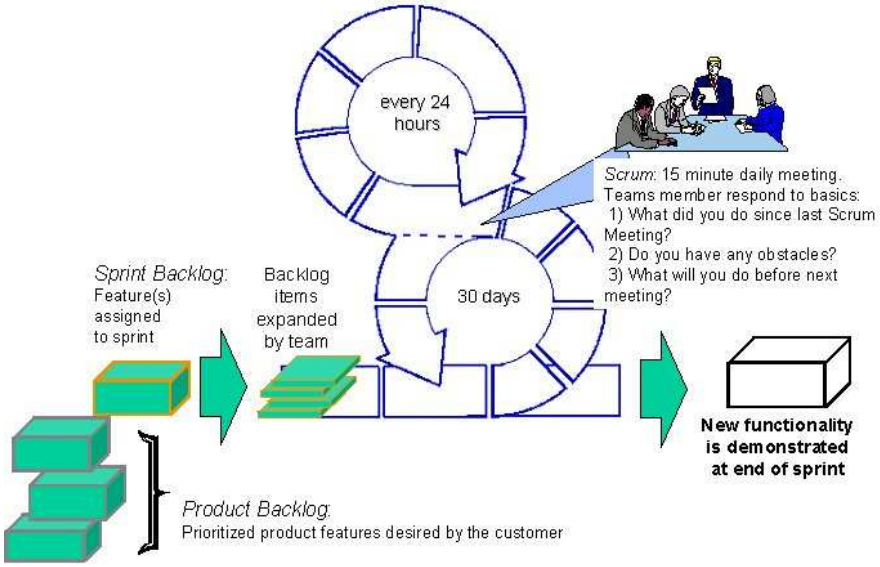

Figure 4: The Scrum workflow [SCRUM-CC]

Scrum is an iterative, incremental process for developing a product. The different iterations are shown in Figure 4.

Scrum works by first defining a backlog of things that need to be done; this list is usually maintained by one person. Other interested parties can request things to be put on a backlog. For each sprint a subset of the backlog is chosen to be done. During the sprint the team will only work on the things that are in the sprint's backlog to keep people focused and creative.

Each day of a sprint there is a Scrum in which the team members respond to the following questions:

- What did you do since the last Scrum meeting?
- Do you have any obstacles?
- What will you do before the next meeting?

The sprint is lead by the so called Scrum Master. It's his job to remove all the obstacles that the team has encountered as soon as possible. This ensures that the team itself can stay focused on the task itself. A Scrum ensures that the team as a whole stays in touch with all parts of the sprint. During a Scrum, management and the customer may be present but only the Scrum Master and the team members are allowed to talk. This is to be sure that the Scrum is short and focus on the task at hand.

After each sprint there is usually a demonstration of what has been done. Before the start of a new sprint a discussion with the team and management is held to establish a new sprint backlog based on the result of the last sprint and changes to the environment/requirements.

Scrum's main focus points are on team empowerment and adaptability. During a sprint a team itself is responsible for doing the given work. The only interaction with management is to tell them what's getting in their way and what needs to be removed to improve productivity. Because after each sprint, the next increment can be changed according to the accomplishments and the changes in the environment, the project is very flexible and adaptable.

Application area

Theoratically Scrum can always be applied when a group of people should work together to archive a common goal. It has even been used as a project management approach, in a so called "Scrum of Scrums". Of course to work properly the teams should be small, but this can be solved by dividing projects into subteams. In the ideal situation all team members should be at the same location for optimal communications among the members. But when this is not the case the Scrum meetings an be held as a teleconference.

Advantages

- During a Scrum sprint there are no disturbances from the outside; this keeps the team focused and creative. Which is very good for the productivity.
- At the end of each sprint, what has been done and what should be done in the next sprint can be evaluated. This keeps the process very flexible.

Disadvantages

Because a Scrum team should be responsible for itself during a sprint, it's important that management doesn't interfere with how the work is being done. For this management needs to fully trust the team to do the right thing, which could potentially be problematic.

Coupling with other methodologies

Scrum itself doesn't tell how a product should be engineered; this is up to the team. This allows Scrum to be coupled with various other methodologies. Adapting a technique as eXtreme Programming (Section 3.4) for the engineering part has been reported to be very successful. Because of this Scrum is sometimes referred to as being a "candy wrapper and not the candy bar".

Also combining Scrum with Crystal Clear (Section 3.6) can be interesting.

3.6 Crystal Clear

From [CC-BOOK, p. 307]:

> "Crystal Clear is a highly optimized way to use a small, colocated team, prioritizing for *safety*[1] in delivering a satisfactory outcome, *efficiency* in development, and *habitability* of the working conventions."

The Crystal Clear methodology[2] is part of the Crystal family of methodologies, where every methodology is characterized by a color (Clear, Yellow, Orange, Red, Maroon, Blue, Violet). That color represents the number of people for which the methodology is suited; Crystal Clear is the lightest color and is meant for the smallest project groups, of two to eight people. Darker colors are for larger groups—these will not be discussed here.

Crystal Clear has at its core seven properties that should be established for every project that wishes to adhere to the methodology. While all of these are desired, only the first three are mandatory; the other four will get the project further into the safety zone. The seven properties are:

1. **Frequent Delivery:** When delivering working, tested code to the actual software users once every few months (or more often, if possible), users will be able to deliver feedback on implemented requirements, sponsors will see progress and developers will get a morale boost.
2. **Reflective Improvement:** Taking time to let the team reflect on what works and what doesn't work for the project, and improving the things that don't work.
3. **Osmotic Communication:** Having the entire team so close together (if possible in the same room, otherwise in adjacent rooms) that people don't have to go to a lot of trouble to raise or answer questions, but can do so instantly, will make people work together naturally, inspect each others' work and pick up relevant information as if by osmosis.
4. **Personal Safety:** If people feel safe to speak up without fear of reprisal, they can give constructive criticism on other people's work and admit their own mistakes, leading to honesty and ultimately to trust.
5. **Focus:** If everybody has time to focus on their primary objectives for two hours a day, for two consecutive days every week, without any distractions that can make them lose their train of thought (like meetings or other work), people will be more focused and work will be finished quicker.[3]
6. **Easy Access to Expert Users:** If expert users are available to the team, they can answer questions and deliver feedback on quality and design decisions.

[1] Although Cockburn in [CC-BOOK] often talks about the concept of *project safety* and frequently states the importance of getting the project further into the *safety zone*, he never explicitly defines this "safety zone" concept. Reading the book, it seems to be a bounded space, where the further a project resides inside the bounds of this space, the more likely it is to succeed.

[2] The principal creator of Crystal Clear is Alistair Cockburn; the methodology is most fully described in his book *Crystal Clear* [CC-BOOK].

[3] Four hours of totally undistracted work a week might not seem much, but Cockburn insists that this is more than we usually get.

7. **Technical Environment with Automated Tests, Configuration Management & Frequent Integration:** A proper technical environment where testing and configuration management/version control tasks (like making backups and merging changes) do not have to be done by hand will make life easier for developers.

Crystal Clear offers several concrete procedures/techniques that can help establish these critical properties (see [CC-BOOK, Chapter 3]), but these are optional: If the team knows of other ways to satisfy the properties, there is nothing that stands in their way. In general, it can be said that Crystal Clear values *properties over techniques*. This also makes Crystal Clear a low-threshold methodology: project groups can carry over their established methods and techniques—which the group has either grown into or were developed to fit their specific situation—to Crystal Clear, and thus will not have to learn a set of new ones before coming up to speed.

Application area

As explained above, Crystal Clear is meant for project groups consisting of two to eight people working at the same physical location, with one or more expert users available. In general, this means any setting where the first three (but ideally, all seven) of the properties can be fulfilled are applicable.

However, the above does not have to be strictly adhered to. All methodologies in the Crystal family support the *stretch to fit* principle, which states that when a potential project does not fit within the target methodology, the principles and practices to be carried out by the methodology can be stretched to fit the particular case. For example, teams that are significantly larger than eight people have carried out Crystal Clear successfully by stretching it to fit their needs.

Advantages

- Because the seven properties are based upon behavior that has been observed in successful project groups, those practising Crystal Clear might well be on the right track to bringing the project to an end successfully. While this is of course no guarantee of success—there are always other factors that contribute to or detract from a project's success—it is likely that these properties contain at least *some* quality that does indeed make the difference between a successful project and an unsuccessful one. In fact, [CC-BOOK] cites numerous stories that demonstrate the success of the properties and techniques provided as well as Crystal Clear in general.
- Unlike traditional, thick methodologies like SSADM or PRINCE, Crystal Clear is flexible as to what project teams are supposed to do and how to do it. This is expressed in the *properties over techniques* and *stretch to fit* principles. In fact, Crystal Clear was explicitly designed to be usable by as many project groups as possible, with the least number of new techniques to learn. It differs in this respect even from a fellow agile methodology

like XP, which explicitly states what practices (techniques) to use, and so narrows its potential users to those who can fulfill adhering to all of those practices.

Disadvantages

- One of Crystal Clear's major strengths is also its principal disadvantage: It tries to be a methodology that is applicable in as many cases as possible. This clearly prevents it from ever being a "best" methodology (like XP strives to be) in any *specific* case, as Cockburn earnestly admits in [CC-PPP]:

  > "Go right ahead and stick with XP. [...] Clear is for the majority. Anyone who can step up to XP can benefit from doing so."

- Another disadvantage might be that Crystal Clear is still relatively new: the only book written on the subject is not yet published, so it may not have a lot of real-world usage yet. On the other hand, the principles behind the methodology are all based on real experiences drawn from real projects, so perhaps wider exposure will reveal that Crystal Clear indeed works "as advertised".

Coupling with other methodologies

Because Crystal Clear is such a lightweight methodology, it can be coupled with several other methodologies to reap the benefits of both. This can be done in one of two ways: either by adding one or more techniques from the other methodology to Crystal Clear, or by *merging* both methodologies to practice them at the same time. As might be expected, the first is easier to attain than the second.

The coupling with some other methodologies described in this essay is illustrated below.

- Crystal Clear can be coupled with XP (Section 3.4). As explained in [CC-BOOK, p. 251], any of XP's practices can be added to Crystal Clear; adding regular delivery and reminding and informing documentation to XP's *planning game* will realize a full merger of the two. See [CC-PPP] and [CC-WIKI1] for a further comparison of Crystal Clear and XP.
- Crystal Clear can also be coupled with Scrum (Section 3.5). According to [CC-BOOK, p. 251], this produces a *No-Process process*, where you can "start anywhere, work in short cycles with high communication and reflective feedback, and eventually you will end up with what you need."
- Crystal Clear could possibly even be coupled with a thick methodology like RUP (Section 3.1). It would be difficult, because Crystal Clear is (as explained earlier) *stretch to fit*, while RUP could be described as *shrink to fit*; even if it would be possible to make them cooperate, it would still be of questionable usefulness. See [CC-BOOK, p. 254] and an interesting discussion on [CC-WIKI2].

4 Project techniques

This section discusses a number of project management techniques. These techniques can be used as an aid to estimate, track and evaluate different aspects of the project.

We start with a discussion of PMBOK, which is actually not a technique in itself, but rather a collection of industry-standard techniques. After that, we discuss COCOMO, MTA, EV and Critical Path.

4.1 Project Management Body of Knowledge (PMBOK)

The Project Management Body of Knowledge is an inclusive term that describes the sum of knowledge within the profession of Project Management (PM). As with other professions such as law, medicine, and accounting, the body of knowledge rests with the practitioners and academics that apply and advance it. The full Project Management Body of Knowledge (PMBOK) includes knowledge of proven traditional practices that are widely applied, as well as knowledge of innovative and advanced practices that have seen more limited use, and includes both published and unpublished material [PMBOK-PMI].

The PMBOK framework splits the project processes into five distinct process groups: initiating, planning, executing, controlling and closing. Note that these groups do not imply that the project has to go through each one in this order; they are only provided in order to be able to structure and categorize the different project processes.

PMBOK also identifies several project knowledge areas: integration management, scope management, time management, cost management, quality management, human resource management, communications management, risk management and procurement management.

By using this twin categorization in process groups and knowledge areas, we can classify project processes, obtaining the table in Figure 4.1.

The PMBOK Guide includes summaries of generally accepted techniques and methodologies that can be used to implement these project processes. Note that these techniques and methodologies need not be, and mainly are not courtesy of PMBOK. *Generally accepted* means being applicable to most projects most of the time and having widespread consensus about their value of usefulness.

Please refer to the guide itself for more in-depth information about the generally accepted techniques and methodologies [PMBOK-PMI].

Application area

PMBOK tries to reflect the growth of knowledge and practices in the field of project management by capturing those practices, tools, techniques and other relevant items that have become generally accepted.

Generally accepted does not mean that the knowledge and practices described in the PMBOK framework are or should be applied uniformly on all projects;

Knowledge Areas / Process Groups	**Initiating**	**Planning**	**Executing**	**Controlling**	**Closing**
Project Integration Management		Project Plan Development	Project Plan Execution	Integrated Change Control	
Project Scope Mangament	Initiation Scope Definition	Scope Planning	Scope Change Control	Scope Verification	
Project Time Management		Activity Definition Activity Sequencing Activity Duration Estimating Schedule Development		Schedule Control	
Project Cost Management		Resource Planning Cost Estimating Cost Budgeting		Cost Control	
Project Quality Management		Quality Planning	Quality Assurance	Quality Control	
Project Human Resource Management		Organization Planning Staff Acquisition	Team Development		
Project Communications Management		Communications Planning	Information Distribution	Performance Reporting	Administrative Closure
Risk Project Management		Risk Management Planning Risk Identification Qualitative Risk Analysis Quantitative Risk Analysis Risk Response Planning		Risk Monitoring and Control	
Project Procurement Management		Procurement Planning Solicitation Planning	Solicitation Source Selection Contract Administration		Contract Closeout

Table 1: Mapping of Project Management Processes to Process Groups and Knowledge Areas. [PMBOK-PMI].

the project management team is always responsible for determining what is appropriate for any given project [PMBOK-PMI].

A few well-known techniques included in the PMBOK framework are Earned Value (EV) management (see also Section 4.4), Program Evaluation and Review Technique (PERT) [PMBOK-PERT] and Critical Path Method (CPM) (Section 4.5).

Advantages

- PMBOK provides a general project management framework in the form of process groups and knowledge areas.
- PMBOK gives a concise summary of and reference to generally accepted project management principles.
- PMBOK proposes a unified project management terminology.

Disadvantages

- PMBOK is only a framework; the actual needs of the project in question should be determined by a knowledgeable managerial team.
- PMBOK provides minimal coverage of various project management methodologies and techniques. One definitely needs to consult specialized texts on these subjects in order to learn the ins and outs.
- PMBOK only covers those aspects of the project management process that are profession independent.

Usage in methodologies

Since PMBOK really is a collection of generally accepted project management techniques, these techniques can easily be integrated in other methodologies when applicable.

4.2 Constructive Cost Model (COCOMO)

COCOMO is an empirical, algorithmic model for estimating the effort, schedule and costs of a software project. It was derived by collecting relevant data from a large number of software projects, then analyzing the data to discover the formulae that were the best-fit to the observations [CCM-SWENG, p. 522].

The first version of the COCOMO model (now known as COCOMO 81) was a three-level model where the levels reflected the detail of the analysis of the cost estimate. The first level (basic) provided an initial, rough estimate; the second level modified this using a number of project and process multipliers and the most detailed level produced estimates for different phases of the project [CCM-SWENG, p. 523].

COCOMO 81 makes various assumptions about the software development process in order to produce its estimates. The latter will only be somewhat accurate when the project uses the waterfall process model and every line of code is produced from scratch. It also fails to take into account that nowadays higher-level programming languages are employed, supported by various automated tools. We will not elaborate on this version, since it has been obsoleted by COCOMO 2.

COCOMO 2 includes support for various development methodologies such as component-based development and prototyping, fourth generation programming languages and CASE support tools. COCOMO 2 still consists of three levels, but these have been given slightly different interpretations:

- **The early prototyping level:** Size estimates are based on object points. These object points are a simple way of quantifying the perceived complexity of requirements that need to be implemented. The required effort is then computed by applying a simple extrapolation from the object points and programmer productivity. Object points are based on the number of screens, reports and modules in third generation programming languages, and can be weighed by the perceived complexity of the screen, report or module in question.
- **The early design level:** This level corresponds to the completion of the system requirements with (perhaps) some initial design. Estimates are based on function points, which are obtained by working out the object points in detail. More specifically, the total number of points is computed by measuring or estimating the following program features: external inputs and outputs, user interactions, external interfaces and files used by the system. The function points are then converted to number of lines of source code using the tables provided by the COCOMO model.
- **The post-architecture level:** Once the system architecture has been designed a reasonably accurate estimate of the software size can be made. The estimate as this level uses a more extensive set of multipliers reflecting personnel capabilities, product and project characteristics [CCM-SWENG, p. 523–524].

For more specific information about the COCOMO 2 model, please refer to [CCM-BOEHM].

Application area

COCOMO is a well-known empirical algorithmic cost estimation technique. It is well-documented, in the public domain and is supported by public domain and commercial tools. It has been widely used and has a long pedigree from its first instantiation in 1981 [CCM-SWENG, p. 522–523].

The application of the first instantiation of the model was limited due to the rather large constraints on the development process. This issue has been mitigated by continued improvements on, and extensions of the model, resulting in COCOMO 2. A refinement of the model for the Ada programming language is available as well.

Advantages

- Although it's hard to pinpoint the exact cost of any given project, one can still obtain usable data by calculating optimistic and pessimistic estimates.
- Implementation and execution of the model is very simple and efficient. As a result, it is supported by public as well as commercial tools.
- COCOMO is a well-known and well-documented technique.

Disadvantages

- It is quite difficult to come up with satisfactory estimates for the size of a project when the latter still in an early stage of development.
- The use of the number of lines of source code as a measure of complexity is highly disputable. Even though COCOMO tries to take this into account by providing different tables for all major programming languages, there are still lots of inconsistencies such as: expressivity differences between programmers, usage of subroutines, general code reuse, etcetera.
- Several input parameters in the model cannot by determined quantitatively; they need be estimated as well. A few examples: experience and productivity of the programmers, maturity and capability of CASE tools. The accuracy of the ultimate estimates of the COCOMO model depends considerably on the exactness of the initial ones.
- The COCOMO model has not been revised since 1995. Therefore, it is likely the model fails to take into account new theories and practices in the Software Engineering field, resulting in worse estimates.

Usage in methodologies

As stated above, the COCOMO model can only be applied when the project in question satisfies a given number of criteria. Additionally, it is advisable to try out other estimation techniques, as to get a feeling of the accuracy of the estimates that have been obtained. Other possible techniques include:

- Expert judgement
- Estimation by analogy
- Other algorithmic cost estimation models

Each of these techniques has advantages and disadvantages, and none is appropriate in all circumstances, since cost estimation of software engineering projects is a very complex task due to the highly dynamic character of the profession.

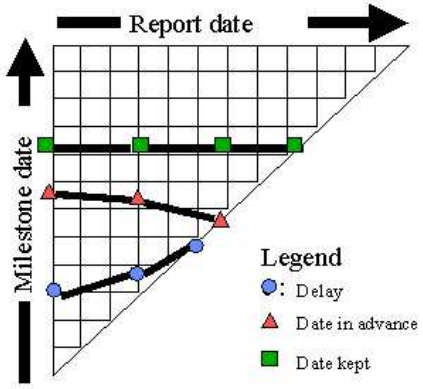

Figure 5: MTA Chart [MTA-SAP]

Tools

Implementing the COCOMO model comes down to evaluating some simple mathematical formulae wherein the variables should be chosen so as to match the characteristics of the project under scrutiny as closely as possible.

As such, one can easily find implementations on the Internet by using a decent search engine such as Google. An implementation by the CSE Center of Software Engineering at USC can be found at [CCM-JAVA]. Other implementations are [CCM-SCE+], [CCM-STAR] and [CCM-NASA].

4.3 Milestone Trend Analysis (MTA)

MTA is a software engineering technique for evaluating the actual progress of a project in relation to its planning.

This relatively simple technique consists of recording the dates of the milestone deadlines at the times they are changed, i.e. when they are postponed or advanced. This way one gets a matrix of data: the columns of the matrix delimit the project milestones, the rows the dates on which the deadlines were reevaluated, while an actual cell contains the new deadline estimate for the milestone in question.

Of course, one can greatly enhance insight in these data by using some simple visualization techniques. This can be done by plotting the estimated deadlines against the dates on which they were evaluated. The latter are usually placed on the X-axis, the former on the Y-axis. The evolution of a project milestone deadline is thus visible as a curve on the graph: downward movement of the curve signifies that the deadline in question was advanced, while upward movement means postponement. One can also easily spot milestone completion: this is the case when the curve intersects the line $y = x$. The general shape of the graph is often roughly triangular: this is the result of the fact that we stop plotting a curve when the milestone in question has reached completion, i.e. when it intersects with the angular bisector of the first quadrant.

An example of a typical MTA chart can be seen in figure 5.

Application area

MTA can be applied to every project that uses milestones as the major indicators of progress. It is in essence a very simple and elegant technique that can easily be applied to assess progress.

Of course, MTA is an evaluation technique that is to be employed during the execution of a project. Its major uses are preventing and correcting schedule slippage, and post-mortem schedule evaluation.

Advantages

- MTA is a simple, elegant and effective technique.
- MTA is widely used and supported.
- MTA has a large application area.

Disadvantages

- MTA in itself does not keep track of inter-package dependencies. Therefore, when a certain milestone completion date is altered, one needs to make sure its dependencies are altered as well. This does not prove to be much of a problem in practice however, since MTA is available as a plugin for more comprehensive project management tools that can keep track of dependencies.
- The inputs of the MTA technique are of course estimates of milestone completion deadlines. As such, it is imperative these estimates are made by knowledgeable and experienced engineers. MTA will not be of much use if these estimates are not reasonably accurate.

Usage in methodologies

As stated above, the only prerequisite is that the project under scrutiny uses milestones. MTA does not impose any further restrictions on the process model and can help to clarify progress assessment in almost any project.

Tools

An MTA plugin for the well-known Microsoft Project management tool is available at [MTA-PROIT].

Dr. Dipp, a distributed tool supporting time-registration and schedule evaluation of software engineering projects, provides an MTA feature as well [DRDIPP].

4.4 Earned Value (EV) management

In earned value management the progress of a project is estimated by comparing what already has been done with the estimates that were made at the beginning of a project. By extrapolating these measurements, a project manager can judge how much resources will be used at the end of a project.

Some common acronyms that are used use in the EV management:

BCWS Budgeted Cost for Work Scheduled

BCWP Budgeted Cost for Work Performed

ACWP Actual Cost of Work Performed

BAC Budget At Completion

EAC Estimate At Completion

Budgeted Cost for Work Performed (BCWP) is also known as the earned value. This value shows what a project really has earned at a certain point in time. The cost of an amount of work can be expressed in different ways. For example in dollars or in hours.

Furthermore one has to choose when something has been earned. It can be chosen to only set something to be earned when the full task is done. Or say that the part of the task that already had been done has been earned. In the last case the problem is that estimating how far a task has progressed is difficult. In the first case the problem is that work on a task will skew the figures a little until the task has been done. For example when 95% of the work has been done the earned value of that task is still zero, while their is a significant amount of spent value on it.

EV indicators

- **Cost Variance:** $CV = BCWP - ACWP$

  This shows the difference between the budgeted cost for a certain amount of work (BCWP) and the real cost of an amount of work (ACWP). A negative number indicates that the cost has been underestimated, while a positive number indicates that the cost has been overestimated

- **Schedule Variance:** $SV = BCWP - BCWS$

  This shows the difference between what has been earned at a certain and what should have been earned.

- **Budget Remaining:** $BR = BAC - ACWP_{Cumulative}$

  This shows the amount of budget that is still available to complete the project.

- **Work Remaining:** $BCWR = BAC - BCWP_{Cumulative}$

  This shows the amount that still has to be earned in this project, thus the work that remains to be done.

- **Variance at Completion:** $VAC = BAC - EAC$

  This shows the difference between the planned cost at the end and the estimated cost at the end. A negative value indicates that the project is costing more than planned and a positive value indicates that it's costing less.

- **Cost Performance Index (Efficiency):** $CPI_e = BCWP/ACWP$

  This shows how efficient the project is being done in terms of cost. For example a value of 2 shows that the project is currently costing half of the amount planned. Or in other words it's being done twice as efficiently as estimated.

- **Schedule Performance Index (Efficiency):** $SPI_e = BCWP/BCWS$

  This shows how efficient the project is being done in terms of time. For example a value of 2 shows that the project is going twice as fast as estimated.

- **Estimate At Completion:** $EAC = BAC/CPI_e$

  This gives a prediction what the cost will at the end of the project. The equivalent but longer version $EAC = ACWP + (BAC - BCWP)/CPI_e$ is often used in the literature. It's important to note that CPI_e is a moving target and changes during the course of the project. Instead of using the CPI_e of the whole project, the CPI_e of for example the last three months can be used. This takes the current performance of the project better into account. Of course choosing shorter timespans for the CPI_e will increase the influence of short periods of peak performance.

Advantages

During a project a manager can judge if the projects is on schedule/budget. If that's not the case an estimate can be made how far the project is over budget.

Disadvantages

It's very difficult to estimate the real Earned Value at a certain point in time. Wrong estimates of this value can make a project look like it's doing a lot better then it really is (or the other way around).

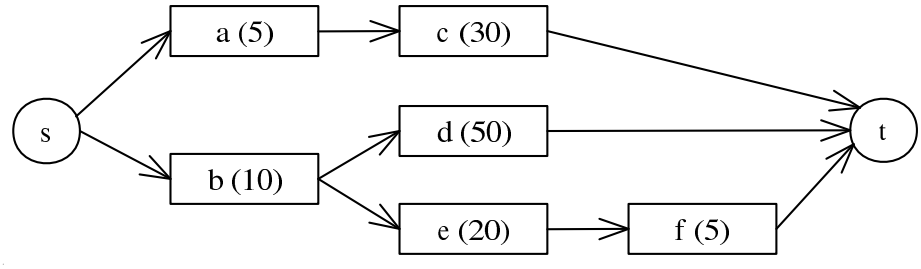

Figure 6: An example of a project network

Usage in methodologies

EV management can be used to monitor projects where there is planning beforehand when certain goals should be reached. This encompasses most thick methods. Examples include PRINCE2 (section 3.3 and SSADM (section 3.2).

Tools

wInsight is a tool for earned value management that intergrates with a wide range of project management tools. For more information see [EV-WINSIGHT].

Just like for MTA Dr. Dipp[DRDIPP] also provides a possiblity to display the projects status in an EV chart.

4.5 Critical path

The critical path technique operates on a directed acyclic graph that sequentially orders all tasks that need to be completed in the project. We term this graph the *project network*. An example of a project network can be seen in Figure 6. The tasks connected in a project network are typically the terminal elements of a Work Breakdown Structure.

The graph specifies the order in which the different tasks need to be completed, and the dependencies between them. Each task has an associated cost in time. The critical path is the longest path from the start of the project to the finish, and its cost is the shortest period in which the project can be completed. Any delay on tasks on the critical path will delay the entire project. In our example, the critical path is (s, b, d, t), with a cost of 60 days.

A related concept is *slack*; this is the time that a single activity can be delayed, without delaying the project. By definition, the slack of all activities on the critical path is 0.

Application area

Critical Path can be used for task scheduling in just about any project management scheme. However, the grade of dependencies between the tasks must

be high enough to make critical path calculation useful. Calculating the critical path for all the deliverables in a (linear) waterfall methodology just won't be all that surprising.

Advantages

Critical Path analysis is very clear and unambiguous. It can be used to identify the most important activities, and make sure extra care is given to them. Furthermore, for activities that are not on the critical path, the slack can be calculated and taken into account.

Disadvantages

Critical path was designed for routine activities, which can be estimated easily and correctly. Uncertainty about the duration of a task cannot be expressed in the critical path model, and reality can therefore sometimes deviate from the model's predictions.

Usage in methodologies

For Critical Path scheduling to be effective, tasks must be known early in advance, and for analysis to be useful, the tasks must have visible dependencies. This makes it unsuitable for methodologies like XP, where activities are small, scheduled only shortly in advance, and tasks have few to no depedencies upon each other.

Tools

Critical Path analysis is a basic project management technique that is widely supported by a variety of project management applications. A very well–known one is Microsoft Project [CP-MSPROJ]. Another tool that can do Critical Path analysis is PlanBee [CP-PLANBEE]. Another one is Open Plan [CP-OPENPL]. These are all commercial applications. A Free Software application that supports Critical Path Analysis is for example Manage-XPS [CP-MAN-XPS]. There are undoubtedly many more.

5 Conclusion

In this document we have concisely reviewed a selection of well-known project management methodologies and techniques. Perhaps the most telling observation we can make in retrospect is that software project management remains a highly unpredictable discipline despite the considerable number of management tools that are available at the moment.

A plausible cause of this unpredictable behaviour is the fact that software development is a relatively young field when compared to other established engineering disciplines. Even though a lot of valuable knowledge about software engineering has already been collected, we're really experiencing the baby steps of an emerging field.

One can ask oneself the question whether software engineering will ever become a highly predictable activity due to the unusual complexity of its core activities, which are highly non-tangible, contrary to those of other engineering disciplines.

We believe that, even though highly structured management methodologies such as RUP can still be valuable for large, static software projects, the future of software engineering lies in the use of highly agile and interactive development methods such as Extreme Programming. Current developments such as open-source development seem to assert themselves as powerful tools in the battle against complexity. However, always keep in mind there is No Silver Bullet.

References

[CC-BOOK] Alistair Cockburn — Crystal Clear
(due out September 2004, June 2004 draft available from
`http://alistair.cockburn.us/crystal/books/alistairsbooks.html`)

[CC-PPP] The Portland Pattern Repository Wiki — Crystal Clear Methodology
`http://c2.com/cgi-bin/wiki?CrystalClearMethodology`

[CC-WIKI1] Crystal Wiki — Crystal versus XP
`http://alistair.cockburn.us/crystal/wiki/FaqCrystalVsXp`

[CC-WIKI2] Crystal Wiki — Crystal versus RUP
`http://alistair.cockburn.us/crystal/wiki/FaqCrystalVsRup`

[CCM-BOEHM] B. Boehm, B. Clark, et al. — Cost models for future life cycle processes: COCOMO
Annals of Software engineering, 1995

[CCM-JAVA] E. Horowitz, et al. — COCOMO 2 Implentation in Java
`http://sunset.usc.edu/research/COCOMOII/`

[CCM-SWENG] I. Somerville — Software Engineering
Addison Wesley, 2001

[CCM-SCE+] SchemeQuest — SCEPlus Excel plugin
`http://www.schemequest.com/`

[CCM-STAR] SoftStar — CoStar COCOMO II Implementation
`http://www.softstarsystems.com/overview.htm`

[CCM-NASA] NASA — Online COCOMO implementation
`http://www.jsc.nasa.gov/bu2/COCOMO.html`

[CP-PLANBEE] PlanBee Critical Path Analysis
`http://www.guysoftware.com/planbee.htm`

[CP-MAN-XPS] Manage-XPS — Critical Path Analysis
`http://industrialonline.net/manage_xps.htm`

[CP-MSPROJ] Microsoft Project — Critical Path Analysis
`http://office.microsoft.com/en-us/assistance/HP010404341033.aspx`

[CP-OPENPL] Open Plan — Critical Path Analysis
`http://www.welcom.com/content.cfm?page=22`

[DRDIPP] Software Engineering Project 2003/2004 group E — Dr. Dipp
Technische Universiteit Eindhoven, 2004

[EV-DOD] DoD — Earned Value Management Web site
`http://www.acq.osd.mil/pm/`

[EV-NASA] NASA — Earned Value Management
`http://evm.nasa.gov/`

[EV-WKP] Wikipedia — Earned value management
`http://en.wikipedia.org/wiki/Earned_value_management`

[EV-WINSIGHT] wInsight earned value management tools
`http://www.winsight.com/products/?Product=wInsight`

[FOLDOC] Imperial College London — The Free On-Line Dictionary of Computing
`http://foldoc.doc.ic.ac.uk/foldoc/`

[MTA-PROIT] Project IT GmbH — Project-MTA: Milestone Trend Analysis plugin for MS Project
`http://www.project-report.com/`

[MTA-SAP] SAP — Milestone Trend Analysis Chart
`http://help.sap.com/`

[METH-WKP] Wikipedia — Methodology (Software engineering)
`http://en.wikipedia.org/wiki/Methodology_(software_engineering)`

[PMBOK-PERT] NetMBA Business Knowledge Center — PERT
`http://www.netmba.com/operations/project/pert/`

[PMBOK-PMI] Project Management Institute — A Guide to the Project Management Body of Knowledge
Project Management Institute, 2000

[PM-WKP] Wikipedia — List of project management topics
`http://en.wikipedia.org/wiki/List_of_project_management_topics`

[PRIN2-COMP1] J.H.E. Roos — Computable Discussie: 14/05/04 - Basis Prince II aangetast
`http://www.computable.nl/artikels/archief4/d20rr4bo.htm`

[PRIN2-COMP2] Jos Keetels — Computable Discussie: 24/10/03 - 'Prince II is totaal ongeschikt'
`http://www.computable.nl/artikels/archief3/d43rr3ep.htm`

[PRIN2-OFF] The Official PRINCE2 website
`http://www.ogc.gov.uk/prince/`

[PRIN2-WEB] Fiona Spence — What is PRINCE2?
`http://www.crazycolour.com/p2/0009.shtml`

[RUP-BOOK] Philippe Kruchten — The Rational Unified Process
Third Edition, 2004

[RUP-PDF] Philippe Kruchten — What Is the Rational Unified Process?
`http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/jan01/WhatIstheRationalUnifiedProcessJan01.pdf`

[RUP-TRANS] Per Kroll — Transitioning from waterfall to iterative development
`http://www-106.ibm.com/developerworks/rational/library/4243.html`

[RUP-WEB] The IBM home page of the Rational Unified Process
`http://www-306.ibm.com/software/awdtools/rup/`

[RUP-IMG] Karl Scharbert — Business Analysis als Bestandteil eines Prozess-Modells bei IT-Projekten
`http://www.karlscharbert.de/ba/baBestandteilVorgehensmodell.html`

[RUP-WKP] Wikipedia — The Rational Unified Process
`http://en.wikipedia.org/wiki/RUP`

[SCRUM-WKP] Wikipedia — Scrum (in management)
`http://en.wikipedia.org/wiki/Scrum_(in_management)`

[SCRUM-CC] Control Chaos, home of Scrum
`http://www.controlchaos.com/`

[SSADM-GUI] Malcolm Eva — SSADM Version 4 - A Users Guide
McGraw Hill

[SSADM-INTR] Introduction to Methodologies and SSADM
`http://www.comp.glam.ac.uk/pages/staff/tdhutchings/chapter4.html`

[SSADM-MS] Model Systems: SSADM
`http://www.modelsys.com/msssadm.htm`

[SSADM-WEB] Mike Goodland / Karel Riha — SSADM - An Introduction
`http://www.dcs.bbk.ac.uk/~steve/1/`

[SSADM-WI] Structured Systems Analysis & Design Method - A whatis definition
`http://whatis.techtarget.com/definition/0,289893,sid9_gci213458,00.html`

[XP-BOOK] Kent Beck — Extreme Programming Explained: embrace change
Addison Wesley, 1999