Chapter 4

Documentation in the software engineering process and the processes it involves

1. Introduction

Software engineering, like all other forms of problem solving, involves information processing to a great extent. Analysing, defining, internalising information to become knowledge, communicating, documenting and coding are all information processing activities that take place during the software engineering process. Whereas the previous chapter focused on the characteristics in general, this chapter and the next will focus on the functional aspects of software engineering in more depth. This chapter will emphasise documentation and the information processing behind it as a vital part of the software engineering process. The following aspects will be covered:

- Fundamentals of human information processing and communication;
- What must be done in the software engineering process;
- Why documentation is needed;
- Problems in the software engineering process;
- What is needed in the software engineering process.

2. Fundamentals of human information processing and communication

Documentation is a product of human information processing. To understand the impact and effectiveness of documentation in the software engineering process, one has to realise that software engineering involves information processing by humans and

machines. Again the human factor plays a pivotal and critical role (Pressman, 1997, p. 59). Because human involvement is so important and has such a great impact on software engineering, it is important to understand how humans process information in order to see how this process relates to the nature of documentation.

Humans constantly process information. Information is a product of observation. Information processing, for humans, takes place between a human brain or mind and an external environment when humans make observations about that environment. It also takes place between minds during the communication process. In the case of the software engineering process, the external environment or what is observed includes the people involved, the domain and the user requirements.

Information processing, however, has some complications to it. These complications are related to the fundamentals of the process itself and also because of human limitations (Loucopoulos, 1995, p. 66). Humans do not process information mechanically, but thinking and communicating involve ideas and emotions (Capra, 1997, p. 70). Our perceptions are not objective, predetermined representations of reality, but is dependent on the process of cognition. According to empiricist philosophers like George Berkeley (whose work has been reviewed in recent years) the mind and what is observed, is intimately connected (Flew, 1979, pp. 41 - 44). This view is to a great extent also supported by the quantum theory, which had a profound impact on our perceptions of reality in the last hundred years. Information processing, according to the quantum theory, is dependent on the influence that the act of observing has on what is observed. The quantum theory also has an established credibility with successful connections to the so-called hard sciences and philosophy, thereby connecting the empirical "objective world" with the subjective, "mind-orientated world". This connectedness

is also supported by some of the most respected theoretical physicists in the world. Roger Penrose in (Freedman, 1994) proposed the great possibility that there might be quantum activity in the brain. The implication of this proposal is that the reason for influencing a system while observing it, might be that observing or thinking involves the mind as part of the system being observed (Capra, 1997, pp. 263, 264). This has great implications for systems where people are extensively involved, like in the software engineering process. When people interact with a certain environment, they respond creatively and do not merely adapt to the surroundings (Matthews, 1999, p. 27). This is also supported by the chaos theory, which stipulates that everything is connected to and influences everything else. Information processing is therefore a fuzzy process.

This might seem trivial at first. However, it is well known that there are countless unpredictable uncertainties in every software system that cause changes with any possible influence on the system (Boehm, 2000, p. 27). In this, people also play an indispensable role in software systems. These uncertainties and the results thereof are therefore incorporated as part of the information being processed. Many of these inherent problems and difficulties, however, can be rectified through proper communication.

Information being processed ultimately becomes knowledge and is integrated into a knowledge base. Humans think by way of association. We store the knowledge we gather from our learning experiences in an associative network, thereby developing a frame of reference for every individual. While learning, people start to rely heavily on experience, connecting concrete actions in the physical world, thereby learning new concepts. These concepts are then abstracted and integrated into the learner's knowledge base or frame of reference. This inductive process allows a developer to get a much more holistic picture of the problem space, which

is very useful, because this is how the user experiences his or her environment (Neill, 1992, p. 31). By gaining knowledge in this way and the deductively process interactively, the functioning of the left and the right brain is integrated. This integration has a synergistic effect in that the whole is greater than the sum of its parts. Cooperation of the left and the right brain complements each other and also stimulates creativity. "The two hemispheres establish two different essential views of the same task" (Sodan, 1998, p. 105).

There is, however, no view or information that describes reality completely. Because of this inherent shortcoming, we have to interpret and complete the information individually to create a comprehensible model of reality (Burgoon, 1994, p. 103). This makes every individual's frame of reference unique. From this unique frame of reference ideas are formed. These ideas are then represented using symbols to describe our understanding of the interaction between our minds and our environment. In the process patterns are formed which result in the development of structures that form the basis for communication between different people's minds (Burgoon, 1994, p. 101). Every individual will create different pictures or ideas in his or her mind from the same external information because of that person's historical background. A person's historical background is influenced and even entirely made up of:

- The language(s) that person speaks;
- The culture and societal sub-cultures that person lives in;
- Political and socio-economical factors;
- Knowledge base and frame of reference;
- Experiences that are uniquely perceived by each person.

Apart from the influence of historical background and other factors previously mentioned, perceptions are sensitive to initial conditions. This phenomenon is described by the chaos

theory and suggests that everything can influence everything else. It also means that in every situation there is potential unknown important information or information not taken into account that might have a very important effect on the rest of the information at hand. This also has an effect on an individual's perception through time. "Common sense suggests and experiment confirms that a person does not always make the same choice when faced with the same options, even when the circumstances of choice seem in all relevant respects to be the same" (Neill, 1992, p. 15).

Information and knowledge are products of society. Society and the knowledge produced by it are essentially oral in nature. Knowledge also has a deep-experienced nature, something a person has to "feel". Knowledge is therefore a multi-dimensional phenomenon. Communicating this information and knowledge is thus a social event. According to the communications theory, a person only communicates successfully if the receiver of a message interprets the message in the way that it is supposed to be interpreted. Adding to that, communication only takes place successfully in relation to a common context (Neill, 1992, p. 11). This common context will be promoted if:

- Everyone were to form the same perception and observation from the same experience;
- Everyone were to describe their perceptions in exactly the same way.

Interpretations, however, differ because of background factors. In other words, the factors and processes behind what is communicated greatly influences what is communicated. The communication process is also not linear in structure, but typically follows the complex patterns of non-linear orientated systems (Van Schoor, 1986, pp. 4 - 10).

2.1 Language

Natural language is the most common communication structure and is used for general communication or is the vehicle by which messages are exchanged between people. However, even between people with the same cultural background and the same language, there can be and very often are many misunderstandings. This is because language has ambiguity as one of its attributes and people in their individuality and different backgrounds might interpret different meanings to the same message (Burgoon, 1994, p. 101). A message also becomes more abstract and ambiguous the further the communicator's knowledge is removed from the daily experience of what is communicated. These problems are even greater between people who speak different languages and have to adapt to speak a language that all parties involved can understand (Van Schoor, 1986, p. 142).

There are also other communication structures available, apart from natural language. Formalised languages like mathematics and computer languages are examples of such communication structures. An important difference between general language and formalised language, is that general language involves a lot of ambiguity and also includes non-verbal factors such as culture and emotion (Burgoon, 1994, pp. 103, 106). Information transfer through natural language therefore has a richer content that is based in the world of experience. Formalised languages on the other hand have a much more precise nature. Formalised languages have no ambiguities and are universal and independent of historical background and emotion, but are less flexible and have a narrower scope of use than general language (Pressman, 1997, pp. 45, 682). However, both language structures are of invaluable necessity in the software engineering process. Formalised languages, based on mathematics, play an important role as the fundamental technical base of the computer science field and thus in the design of

modeling systems and computer languages. Models and computer languages are the vehicles for communicating or translating human understandable messages into machine understandable format. As for natural language, communication between humans is a very important part of developing software systems. Requirements are translated into natural language, which is then translated into a formalised language or model and eventually into a programming language.

Although formalised communication structures are of great importance, it is not favourable to all communication-related aspects of the software engineering process. This is true especially for communication or transfer of knowledge. The problem with this specific process is that knowledge has such a rich unique nature. In other words, the knowledge to be communicated is interpreted and then transferred in this format to its recipients. This gives knowledge transfer a subjective nature and therefore makes effective formalisation difficult (Pressman, 1997, pp. 45, 682).

There are some problems in integrating these two different communication structures. The fundamental reason for this is that human thinking and communication has a non-deterministic nature, whereas computers are largely deterministic systems.

In considering all these background factors, sharing knowledge seems to be extremely difficult. But knowledge can be and is commonly understood and shared by people all the time. This is done by extensive communication where information flow has an iterative, non-linear structure involving feedback. Feedback continuously strengthens the common understanding between the communicators, thereby synchronising their knowledge (Communication, n.d.). Therefore, when someone is making observations or gaining knowledge about an environment, this knowledge has to be tested and corrected through proper

communication. This again emphasises the utmost importance of good communication in the software engineering process.

3. What must be done in the software engineering process

In addition to the fact that the software engineering process is acknowledged as a complex activity, a developer also needs to have expert knowledge of the problem domain (Glass, 1995, p. 190). This knowledge combined with the developer's technical knowledge, is a prerequisite for developing a successful system. A system is successful when it is technically sound and satisfies the user and business requirements, thereby enhancing the business processes in the domain environment.

The whole software engineering process must be coordinated by managing dependencies between goals, resource allocation, information availability, activities and actions to achieve these goals. A system to be developed must provide a solution to a problem in a specific domain. Every domain has a certain set of functions, problems, a specific terminology and a set of strategies from which solutions to problems must be derived (Zave, 1997, p. 2).

In the process of developing a software system, a massive amount of descriptive, qualitative, interrelated information on complex real world settings has to be captured, processed and presented. This information grows exponentially with the increase in project sise and therefore complexity (Roth, 1994, p. 164).

3.1. Information that must be captured, processed and documented

Developers collect development information from requirement specifications, documentation about the problem domain and from

the users themselves. This information includes operational information, management information, business rules and processes. All this information is then used for further analysis of the problem. When developing a software application to meet a user's requirements, the developer has to know exactly what the user needs. The developer needs to have the knowledge that the user gained through experience (Cucchiarelli, 1998, p. 53).

To gain this domain knowledge, it is important that the developer has to learn the rules - business or otherwise. It is, however, also important but very much neglected to have situational know-how that is gained by experience alone. As was mentioned before, the process of gaining knowledge first involves experience. By experimenting one gains the knowledge that is needed to establish a theory. This theory can then be tested, fine-tuned and can then be abstracted for further use (Zave, 1997, p. 2).

Throughout the software engineering process, information generated from the following activities and phases are captured, processed and documented:

- Brainstorming sessions;
- Problem exploration;
- Planning;
- Decision making;
- Analysing;
- Designing;
- Coding.

Although the software engineering process is a non-linear process, the information is presented here linearly in sequential phases because this is the way it is traditionally documented and managed.

Strategic phase

- Requirement specification;

- The feasibility of the project;
- Objectives;
- Priorities;
- Constraints;
- Critical success factors;
- Scope of the project;
- Proof of concept;
- Technology candidates that might be used;
- The methodology to be used;
- Organisational, technological or other issues;
- Resources needed;
- Delivery and acceptance plan;
- Training plan;
- Financial plan;
- Installation plan;
- Corporate entity model;
- Schedule to be followed.

Development phase

- Analysis
  - Process layout;
  - Prototypes;
  - Entity relationship diagram;
  - Function diagrams;
  - Function/Entity matrix or diagram;
  - Models for data-flow, function dependency, and state transition;
  - Initial transition strategy;
  - Audit and recovery procedures;
  - Outline of manual procedures;
  - User acceptance criteria;
  - Constraints;
  - Design approach.
- Design

- Logical design;
- Pseudo code, data flow and other diagrams;
- Database and file design;
- Specifications of manual procedures;
- Draft user manual;
- Test plan;
- Documentation of the application system architecture
  - Menu structures;
  - Batch procedures;
  - Manual procedures;
  - User interface and style for screens, reports and forms;
  - Function definition;
  - Error correction cycles, batch control;
  - Procedure exceptions.
- Code
  - Physical program design;
  - Unit, integration, system and acceptance testing
  - Code documentation.

Implementation phase

- Training and educational material;
- Completed systems documentation;
- User documentation;
- System help.

Maintenance phase

- Changes and enhancements;
- Software and document configurations (Barker, 1990; Pressman, 1997, pp. 24 - 48).

4. Why documentation is needed

Although the ultimate product of developing a software system is the coded product that processes information and thereby satisfies the user's requirement(s), there are also secondary products that are of equal importance. These secondary products can be grouped together as documentation. While working together and while developing a system together, people accumulate knowledge. Documentation communicates this knowledge to all parties involved in all phases of development. The documented information range from strategically related information through to program source code. It involves role-players like top-management, project managers, business analysts, system analysts, designers, coders, maintenance people and most important of all, the end-users whose requirements have to be met. "Productivity and quality of software development and maintenance, particularly in large and long-term projects, is related to software readability. The most important is documentation that provides the big picture and ties smaller pieces together" (Haneef, 1998, p. 75).

Apart from accommodating the communication of information to the role-players involved, documentation is also useful for tracing system requirements (Booch, 1994, p. 281). Documentation is also used for making formal and informal reviews and is also used to manage development projects. Documentation is generally the only view, except for the project plan, that management has on a project (Vescoukis, n.d.). "Despite the fact that documentation can't erase the "complexity", the "conformity" obligations and the changeability of software, it is an indispensable resource to master these elements" (Blanqui, 1997, p. 59). Documentation strings everyone and everything in the software engineering process together, thereby integrating the whole process.

5. Documentation problems in the software engineering process

5.1. Problems with communication

Although the ideal model for effective communication in software engineering is for everyone to be able to speak to everyone else whenever they need to and for as long as needed, it is simply not always feasible. Not only are people not always readily available to talk to because of work and also because of geographical limitations, but the number of communication paths also increase exponentially with the increase in the number of people involved. In practice, communication is routed through formal documentation, which streamlines the coordination of communication (McConnell 1997, p. 28). In doing so, most communication is done via text. The text that is compiled is the compiler's interpretation of whatever information is to be communicated to the receiver of such a text document.

The communication process between a user with a certain requirement, and the systems analyst forms the basis for the development of a system. It must of course be understood that because of their different views and knowledge references, the user and the analyst can be seen as speaking two different languages as far as the problem is concerned. The user speaks about the problem from the domain context. The analyst speaks from a technical background and unfortunately and erroneously often places the emphasis for the solution there (Loucopoulos, 1995, p. 66).

A huge problem in the software engineering process is that all to often the person that communicates requirements to the compiler of a text document is interpreting the real user's requirements. Usually the communicator of requirements is a manager and the compiler is a project manager or systems analyst that will be involved in the development of the system. The project manager or systems analyst then communicates, via the compiled text, the

requirements to the rest of the development team. The person that communicates the requirements to the project manager or systems analyst is already interpreting parts of the requirements (Burgoon, 1994, p. 101). The reason for this is that he or she is not just summarising the requirements of all those involved in the problem domain, but is likely to favour a management or operation-driven view, depending on what his or her involvement is. Putting this information in text format is two further levels of interpretation removed from the user's original requirements, because the project manager or systems analyst first has to interpret what the user is saying and then has to translate it into a text-based medium. The developer has to develop a system based on a document that contains information that is at least three levels of interpretation removed from the actual requirement information. With all this, it has to be taken into account that the real user often has difficulty in formulating his or her own requirements accurately. To make things even worse, development is often done solely according to the content of this document. This method of proceeding, more often than not, results in an application that the users feel does not meet their requirements.

Another important factor that gives rise to misunderstanding in the communication of requirements, is that there is a wealth of information in the problem domain that does not get communicated in the initial formulation of the requirements.

5.2. Problems with text based documentation

In using conventional methods of setting up development documentation, the information is arranged in a typically rigid, precompiled format that has a linear structure and is printed on paper. This arrangement has advantages in that the information is

packaged compactly, is easy to use and to measure the system by. However, these advantages are also its biggest drawback. The reason for this is that the information in these documents is already interpreted by at least one person. These interpretations are then formalised into a standard development document format. These one-dimensional representations, being formalised and standardised, omit much of the original information and concepts being communicated, especially as far as domain information, knowledge and requirement specifications are concerned. The format, in which representations of requirement information is made, is also not effective enough. "Abstract representations, such as written descriptions, flow charts, object class hierarchies etc. cannot provide a grounded understanding of the customer's requirements" (Winograd, 1995, p. 69).

In using text as a mode of communication, all non-verbal communication like body language is absent. Real-time feedback and making adjustments while communicating is also absent. The mode of communication changes because communication moves from a rich communicating environment to a much poorer one. In other words, the communication bandwidth is reduced considerably. It is also important to remember when working with knowledge and information in document format, that by documenting it, this multi-dimensional, experience-based and non-linear phenomenon is represented in a one-dimensional, linear structure (Neill, 1992, p. 152). The result is that knowledge in this format is weakly conceptualised.

To further complicate matters, as was mentioned before, people interact with an environment by responding creatively to it. This is in accordance with the work of Gadamer, the great philosopher of Hermeneutics. According to him, when a person reads a text, understanding that text means recreating the author's original intention of the information. This must be understood against the background that a software document, like a user requirement

specification, is already the compiler of the document's interpretation of the user's requirements. In reading the text, understanding it moves beyond its original psychological and historical context. This happens because throughout the process, the reader or interpreter transcends his or her own horizon, while pulling the information in the text beyond its original meaning. The meaning of a piece of text is therefore not fixed, but changes according to different people's interpretations. According to Gadamer, understanding then is to understand differently from what the author intended (Gadamer's philosophical hermeneutic. 1986).

Also, conventional documentation structure is entirely linear and therefore reflects a top-down structure. Doing development using a strictly top-down structure is not practical. A conventional software development life-cycle methodology, like the Waterfall model (which is a hardware engineering process model slightly adapted to software engineering), follows this structure and is therefore linear (Glass, 1995, p. 168). Developers on the other hand, follow a combination of top-down and bottom-up approaches in a non-linear way.

It must also be taken into account and remembered that a software system is dynamic. Changes are made to the system because of new insights or enhancements needed. In practice, after the system is implemented, these changes are very often not reflected in the documentation. Documentation, therefore, becomes outdated and worthless (Glass, 1995, p. 27).

5.3. Problems with managing software systems

Another problem with developing a software system is managing it in a conventional way. This implies sequential phases of development where the next phase only starts after completion of

the previous, thereby building on the results of the previous. Management wants to measure this sequential progress and the way it is measured, is with conventional linear text-based documents. This works fine for a conventional software development life-cycle like the waterfall model (Glass, 1995, p. 168). However, software project development very seldom, if ever, follows a linear, sequential development path. With the pace at which software development technology advances, managers also generally tend to lag behind in their technical knowledge. This makes them even more reliant on documentation, which, with documentation in its current state, results in escalating communication problems.

6. What is needed in the software engineering process

Because of the problems associated with processing and communicating information, a medium is needed that accommodates the most effective information processing and communication.

Information must make a strong but correct impression. The needed information must not be buried under mountains of information, but must be visible, readily available and useful. When using information for some task, it needs to be the right information, at the right time, to the right person and in the right format. "Information is power if and only if you have the knowledge to know what it means, the will to use it, the ability to apply it and access to a channel of communication" (Neill, 1992, p. 39). Since knowledge has quite a rich and experience-based nature, the developer cannot simply be distant to the user and his or her problem environment. The whole problem context has to be studied. This is a learning process for the developer.

What is needed is information about the problem to be solved that does not oversimplify the issues involved (Spiro, n.d.). Special efforts must be made to highlight complexities, exceptions and

contradictions. Imprecise, unformalisable data is important. Some even go so far as to say "requirements specifications are considered harmful because they tend to make rigid something that must remain inherently flexible" (Glass, 1995, p. 92).

6.1. Representing information

Such rich information should also not just be represented in mere conventional ways. Instead of having a single representation of the issues involved, the information should be represented in multiple ways and on different levels, illustrating the logic, analogies and relationships between them. There can potentially be many different approaches or views to a system or an application that will satisfy a certain set of requirements. A developer needs to be able to accommodate such flexibility efficiently (Feijs, 1998, p. 73). "The greater the number of fundamental ways of thinking that are superimposed on texts and interlocked with each other, the greater the fullness of coverage of material that inevitably is oversimplified in traditional approaches" (Jones, 1992, p. 147). Powerful solutions require the integration of all views and aspects involved. This conforms to the views that artificial intelligence expert from MIT, Marvin Minsky, calls for. That is, the "integration of logical and analogical methods in intelligent software" (Sodan, 1998, p. 111).

6.2. Managing information

In order to manage the software engineering process effectively, a tool is needed that enables the integration, organisation, coordination, maintenance, distribution and communication of the whole range of development components on all levels and from all

perspectives. Examples of these are ideas, specifications, models, designs and code (Marovac, 1997, p. 68, 70).

To manage this amount and diversity of information, tools and techniques are needed for examining and integrating not only information regarding the interaction with machines, but also information about the social systems that are involved in the domain environment. This rich data must be integrated into the design of the system. To integrate all of this information requires tools that support structures that range from being very informal like notes to very formal, for example the rigid automated structures of CASE tools (Norbert, 1995, p. 70).

6.3. Documentation

Because of its nature and volume, development information needs to be integrated as a whole. Although the knowledge that individuals generate when working together is valuable, a shared understanding of this knowledge is much more valuable for it creates a synergistic effect. The whole of this knowledge base is greater than the sum of its parts (Capra, 1997, p. 27). This knowledge is mutually defined and is constantly evolving. All persons involved in the process of development must have access in the form of in-depth drill-downs into development information or be able to access any piece of information from every possible angle. Persons involved must also be able to contribute to this information and reflect on it. Working through the knowledge base, an individual must be able to interpret the information presented. This means that relationships have to be created that link development objects to information at meta-level through the whole spectrum to information at operations level. From this, knowledge abstractions can be created, thereby reflecting the person's own understanding of the material and therefore the problem. This externalising process also communicates a person's

understanding of the problem to others (Marshall, 1995, p. 93). Relationships or links can also be made between specific people and particular sections of information. In this way, rich information can be shared and communicated by the multidisciplinary group of people involved.

Knowledge about the domain environment is much more important than is generally acknowledged. Domain knowledge supports the refinement of requirement specifications (Zave, 1997, p. 2). The software engineering process has actually much more to do with application domain problem solving than with just programming (Vescoukis, n.d.). Systems must be developed with the whole problem domain environment constantly in mind to see the big picture. To develop a system is to develop it as part of an environment and not as an isolated application. An application should not just be developed to satisfy a set of user requirements, but also to improve the overall process of which the software system will be a part. Domain knowledge is therefore very important.

This is more evident in the current working culture than before, because contracting developers generally know less about the business environment than developers, being full-time employees, do. Because of this situation, the role of accurate and good quality documentation becomes more and more important. "The domain knowledge should represent a communication bridge between the user and the analyst, allowing to define the objects of the real world and the processes that allow to transform objects." (Cucchiarelli, 1998, p. 53)

Conventional documentation is useful for certain aspects of the development process, like viewing and measuring a software project from a managerial point of view. Conventional documentation must therefore still be used, but must be complemented with non-conventional, non-linear documentation to

reflect information of the development process as a whole. In other words, formal and informal documentation are both necessary and must be integrated.

Documentation should not be seen as being static, isolated and created at one point in time, but as a network of dynamic collections of interacting information modules composed on demand (Huser, 1995, p. 49). Documentation must also be viewed as being part of the product being developed and must therefore evolve with the development of the project.

To get an accurate picture of what the desired solution must be like, the different views of users and developers must be integrated (Cucchiarelli, 1998, p. 53). The emphasis must be placed on knowledge rather than just information.

7. Conclusion

In this chapter it is evident that the software engineering process is dependent on the processing of information and the communication thereof. Because people are involved, these processes are imprecise, uncertain, subjective and complex. The characteristics of information processing present problems in the transfer of information. It is also evident that documentation plays an indispensable role throughout the software engineering process. Conventional documentation is necessary for some purposes, but is inadequate for others. Software engineering documentation must be able to adapt to and accommodate these factors.

This chapter aimed at showing that creating documentation is not a trivial process, but is very important. Creating documentation is in fact as much a part of software engineering as the software system or application itself. Whereas this chapter focussed on

the creation of documentation and the processes involved with it, the next chapter will focus on creating the application and the tools and techniques used to do it.

Chapter 5

Methods, techniques and tools in the software engineering process

1. Introduction

Software engineering operates in both the problem and solution domains. Whereas understanding the problem and documentation related issues usually centers on the problem domain, development centers on the solution domain with its technical and methodological aspects. The problem is that traditionally the emphasis in software engineering was almost entirely placed on the solution domain. These two working areas must, however, be closely integrated. This is so because on the one side software problems are constantly demanding solutions that are more complex and cover a greater diversity. On the other side, software engineering involves increasingly more than just programming. Other problem solving aspects like creativity, communication, understanding, idea generation, intuition and thinking in terms of analogies are becoming increasingly important (Jarzabek, 1998, p. 95). This calls for development strategies and tools that can accommodate complexity and diversity and can adapt to changes. Whereas chapter four focused on documentation, information processing, understanding and communication, this chapter will focus mainly on development issues.

2. Methods

For development to be adaptable to complexity and change, development strategies need to include methods that focus on the system as a whole as well as on system components and relationships between components.

The parts of a system can differ considerably from each other. For most effectiveness, each different part must be approached in a way that is best suited to it. This implies that different parts of a system might be approached in different ways. "Multiple strategy systems and holistic approaches are important for dealing with variations in application behaviour" (Sodan, 1998, p. 110). The alternative is to force a single development approach on the system as a whole. Although this approach will certainly enforce uniformity across the whole system, using a development approach where it is not suitable might result in a less efficient system. It can also lead to the introduction of unnecessary complexity to an already complex system. It is important to use methods that are practical. If an approach does not work well enough, a different approach should be used (McConnell, 1993, p. 163).

2.1. Methodologies

In developing systems or applications, each system is largely unique. To begin with, the requirement specifications are unique in terms of what is required, and in how it is formulated (Boehm, 2000, p. 32). This situation is influenced by how well the people involved know the problem, what is required and the environment where the system will function. Secondly, the type of system that is to be developed is also important. The system is characterised by its size and complexity (ref. chapters 1, 3), as well as by the possible existence of a standardised way or well known pattern of developing such a system.

The two areas of software engineering are known as the problem and the solution spaces. Knowledge about these two areas will determine the nature of the chosen methodology. A methodology is a formalised way of handling complexity in the software engineering process. Methodologies are fundamentally grouped,

with regards to the processes that are followed, into scientific methods (involving formulating hypotheses and testing them) and mathematically driven engineering methods (Glass, 1995, p. 79). These two approaches can also be defined as bottom-up or inductive and top-down or deductive approaches respectively.

A system may have parts that have to be approached in different ways (Sodan, 1998, p. 110). In the case of either the areas of a system or part of a system being largely a grey area, the methodology must preferably have a research and development or bottom-up nature with the creation of knowledge and a product in mind. In case of both areas being well known, an engineering or top-down approach with mainly the product in mind should be sufficient. However in a system of significant size and complexity both these approaches are normally used interchangeably. Theory and practice are used intertwined (Glass, 1995, p. 138)

Methodologies can further be broken down according to where the emphasis is placed on the components of a system. A software system consists of data components and functional components that act on those data components. The emphasis in a methodology is placed either on the data or the functional components or both. There are mainly three different paradigms with regards to software engineering methodologies.

The first is the functional approach where the system is primarily analysed and understood in terms of the functional aspects and only secondary with regards to data. A system according to this approach is a top-down hierarchical breakdown from high-level abstract functionality down to progressively lower-level functions (Mazza, 1996, p. 52).

With the second approach (which is called the information engineering approach) the emphasis is primarily on the data

components and only secondary with regards to the functional aspects (Pressman, 1997, p. 237; Sebesta, 1993, p. 21). A system according to this approach is a bottom-up decomposition of data-entities and the relationships between them. All entities are on the same level.

The third approach is the object-oriented approach, which falls between the other two and is a combination of both. Data components and the functions that act on them are encapsulated in modules or objects. The emphasis here is on both data and functional components and an integration of the two (Vessey, 1998, p. 100). A system according to this approach primarily has bottom-up decomposition of entities called objects and the relationships between them. Relationships between objects also include inheritance. This gives object relationship also a hierarchical nature and top-down decomposition structure. The functionality of an object is decomposed according to the functional approach or a top-down hierarchical breakdown of functions.

Software engineering has evolved into a systems approach (with its synergistic qualities, where the emphasis is on the relationships between components). The evolution is essentially from a mechanistic approach to problem solving to systems thinking (Capra, 1997, p. 27). "System's thinking involves a shift from objective to 'epistemic' science; to a framework in which epistemology - 'the method of questioning' - becomes an integral part of scientific theories" (Capra, 1997, p. 40).

This is combined with any of the three methodological approaches or a combination of them depending on the application or part thereof. However, the object-oriented approach is currently the method of choice. Analysis and design, which are recurring activities for knowing and manipulating the system as a whole and its components, is a combination of top-down and bottom-up

activities (Ghezzi, 1991, p. 115). The result is a very flexible software engineering process that can be applied to a rigid development flow as well as to a flexible evolving system. "There is growing recognition that software, like all complex systems, evolves over a period of time" (Pressman, 1997, p. 37).

The system as a whole is visible and accessible as well as each individual component with its detail. The whole approach is holistic, making the whole greater than the sum of its parts.

2.2. Systems development life-cycle

A systems development life-cycle (SDLC) is in essence an implementation of a specific methodology or combination of methodologies. The systems development life-cycle (SDLC) is a succession of steps to be followed when developing a system. The SDLC or "development system model, is the collection of people, processes and tools that implements the development sequence. If systems development is a series of transformations from goals to requirements to design to code, the development system makes the transformation happen. The development system is an information system that manipulates different descriptions of the system being built" (Bullock, 1999, p. 119). This pattern or framework of transformation dictates the life-cycle of an application through the various stages:

- Initial goals and requirements;
- Feasibility studies;
- Strategic planning;
- Initial development;
- Testing and verification;
- Implementation;
- Extending or evolving the system according to new and changing requirements;

- Maintenance and user support;
- Phaseout and closedown (Rajlich, 2000, p. 66).

Although all models of the software development life-cycle generally have the same phases or stages, the process structure can vary between the following:

- Formal mathematical models (Pressman, 1997, pp. 45, 46, 681 - 694);
- Linear sequential (like the Waterfall model where phases follow sequentially) (Glass, 1995, p. 168);
- Incremental models like the evolutionary model (where the system is further developed from and initial core) (Pressman, 1997, pp. 37 - 39);
- Prototyping models (where a system gets developed further from a prototype that was initially developed as a system specification or used for analysis) (Pressman, 1997, pp. 32 - 34);
- Iterative models (like the spiral model where risks are managed in each iteration involving extensive customer feedback) (Pressman, 1997, pp. 39 - 42);
- Models based on fourth generation development technology (where prototypes can be generated and overall development is done on a higher level than normal) (Dawson, 1995, pp. 80, 81).

To make the development process adaptable to varying project needs, development models can be combined into a non-deterministic meta-model for developing in a different way in the same project when necessary (Dawson, 1995, pp. 80, 81).

Backstage to actual development being done, the whole process must also be managed by setting objectives and coordinating people and work (Mazza, 1996, pp. 298, 299). This ranges from a laid back approach to projects being highly managed (Glass, 1995,

pp. 8 - 10). Strongly managed projects go hand in hand with a lot of formalisation and standardisation.

2.3. Techniques

Apart from following a methodology or methodologies during development, there are also techniques that are used to assist in developing software. These techniques give the developer a better understanding of the system and help reduce complexity.

2.3.1. Modeling

When designing a model of a system, the developer first identifies the system components and their relationships to one another. A model is then constructed using this information. The components can range from basic components like data and functional components to systems within the bigger system, consisting of data and functional components themselves (Pressman, 1997, pp. 300 - 312). Models can range from having very high level abstract information to low implementation level models with detailed information. "Modeling is a central part of all of the activities that lead up to the deployment of good software. We build models to communicate the desired structure and behaviour of our system. We build models to visualise and control its architecture. We build models to better understand the system we are building, often exposing opportunities for simplification and re-use. We build models to manage risk" (Booch, 1998).

2.3.2. Modularisation

An important concept of breaking up a system into smaller manageable components is called modularisation (Pressman, 1997, p. 349). As was mentioned above, system components can be systems themselves. Although a system functions as an integrated whole, there are sub-systems of components that function together as units within the whole system. There can even be sub-systems within sub-systems. Every one of these sub-systems are modules and modularisation can even be applied down to pieces of code. These modules must preferably be loosely coupled and largely independent of other modules. This will reduce the complexity of the system as a whole (McConnell, 1993, p. 774).

The idea behind identifying and creating modules is to encapsulate a single alone standing concept into each module (Mazza, 1996, p. 110). This organises the components or elements of a system or an abstract model thereof. "Concept clustering benefits abstraction to help users not forming deviant concept models" (Nielsen, 1995, p. 318). Modularisation, if implemented properly, helps to accommodate complexity in a system because the modules are largely independent of each other, therefore allowing the developer to concentrate on specialised portions one at a time. This has the advantage that the user can view the system or parts of the system from a high abstract level or zoom into the lowest detail. In both cases the rest of the system can largely be ignored, which shelters the developer from taking unnecessary information into account.

2.3.3. Abstraction

Abstractions are level-specific constructions of a system from high-level broadly defined components to low-level detailed components (Pressman, 1997, p. 347). This enables the developer to concentrate on a specific level at a time, ignoring the details and complexities of the system on other levels at this

stage (McConnell, 1993, p. 775). A hierarchical breakdown, as in the functional approach, is a form of abstraction. High-level functionality gives an abstract view of the combined functions at a lower level. Inheritance in the object-oriented approach is also a form of abstraction. Inheritance is implemented via the use of templates called classes. Classes can inherit from, or incorporate other classes. Objects are created from these classes (Gil, 1998, p. 118). To decompose a system from high-level abstract components to low-level components is called step-wise refinement.

2.3.4. Patterns, frameworks and architectures

A pattern is a "configuration of relationships" (Capra, 1997, pp. 27, 81). When working in or experiencing our environment, we develop patterns and frameworks to help us understand it better, to deal with it easier and to reduce complexity (Olson, 1993, p. 45). Although every system is largely unique, there are similarities between systems of the same kind. These similarities can be grouped together into a framework or pattern. Such a framework or pattern, formalised in computer terms, is known as a software architecture (Gil, 1998, 119). For instance, point-of-sale systems have the same general architecture. The same is true for payroll systems, client/server systems and many others. Knowledge about such an architecture helps a developer to work within a known general structure. Where fitting and relevant, frameworks or patterns in certain areas can also be used in other related or even non-related areas as metaphors for better understanding or communication (McConnell, 1993, p. 9). "Software architecture encompasses the set of significant decisions about the organization of a software system: the selection of the structural elements and their interfaces by which a system is composed, together with their behavior as specified in the collaborations among those elements, the

composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization" (Booch, 1998).

A design is a pattern of organisation (Capra, 1997, p. 155). Patterns are very important and powerful - "design patterns are effectively greater than the sum of their parts" (Gil, 1998, p. 119).

2.3.5. Re-use

Re-use means making use of generic components. This results in smaller systems with more stability because tested, working components can be used instead of new, untested components. Generic does not just pertain to code, but also to other development components like knowledge, development information, designs, models, modules, data, interfaces, abstractions, frameworks and architectures (Pressman, 1997, pp. 728, 732, 733; Shaw, 1996, p. 153). In object-oriented development, classes are used to make many object instances of the same class. Classes can also be inherited from other classes, therefore incorporating further re-use. However, although re-use of generic components has many advantages, it also requires good documentation for effective use.

Platform-independent generic code components, like binary components or components that run on virtual machines, are re-usable code components that evolved from the object-oriented concept (Meyer, 1999, p. 144). These components are pieces of code that are independent of their development environments and the programming languages it was coded in. Such components can be used and developed in various development environments and programming languages. They can also be incorporated into various application architectures. The same component, for example, can

be used in conventional client/server, web and mainframe applications. A binary component can also be deployed to work on different operating systems. Apart from being independent of, and integrating environments, generic platform independent components can also be scalable as far as number of users is concerned. Like object-oriented components, these components are self-containing, encapsulated modules that can only be accessed through an interface. These components are therefore highly re-usable and for this reason good documentation on the working of a component is essential. Binary components can also be combined in development with code. Therefore certain functionality may be incorporated and do not have to be developed (Maurer, 2000, p. 29).

The above-mentioned techniques can be used in all software engineering approaches and methodologies.

3. Tools

Development tools are used to assist in the development and implementation of a system that was identified as the solution to a problem. Development tools assist developers in analysing and designing systems. These tools can assist the developer on all levels of the software engineering process - from the strategic phase through the development of the working system and even with testing and maintenance. In software engineering, and in this case using software engineering tools, there is always a trade-off between control, structure and standardisation on the one hand and flexibility, creativity and uniqueness on the other. The choice of tool to be used must be in accordance with the type of system to be developed, the knowledge of the problem and the choice of methodology to be used.

There are tools to accommodate every facet and phase of the software engineering process. Tools used in the strategic phase are mainly aimed at planning and other high-level coordinating activities. The components of this phase are all the available resources and the task at hand is to allocate and balance these resources efficiently in terms of what is to be achieved through the whole process. Tools used in the development phase are geared towards analysing, designing, testing and implementing a software system. Development tools can broadly be grouped into upper and lower range development tools. With upper range tools the emphasis is placed on logical design or modeling in problem or business specific terms. Lower range tools are used for physical design in technical terms and for the writing or generating of program code. There are also tools used for testing and implementation. Tools used in the maintenance phase are utilised for diagnosing problem areas, correcting errors and making changes.

3.1. Computer Aided Software Engineering (CASE)

Tools are also used where the software engineering process is integrated. These tools are categorised as Computer Aided Software Engineering (CASE) tools (Pressman, 1997, p. 808). The philosophy behind CASE technology is to standardise the software engineering process. Development is done according to a pre-defined methodology and working practices. The advantage of integration is that there are standardised models, documentation, code and that there is an integrated process flow between software engineering phases. This presents an ideal, seamless development framework that can also accommodate development information. All these factors should facilitate better coordination and communication by improving readability and keeping documentation up to date (Glass, 1995, p. 124). The disadvantage, however, is that in a non-linear, unpredictable

process, which characterises the software engineering process, a linear, rigid way of working can present problems. Although CASE technology has advanced much in terms of accommodating both flexibility and integration, a problem still exists with integrating tools and data from multiple vendors. This is needed to ensure technological flexibility as well as an integrated process (Blanqui, 1997, p. 60). CASE can therefore be defined as an implementation of the software engineering process.

There are also other integrated development environments. These environments are not integrated so much by specific methodologies, but by architectures that link all developed components or systems. These developed components can be used while developing other components or systems, thereby using the functionality of pre-developed systems in others. Development in such an environment is methodology independent. The integrating platform is not so much on a development platform, but underneath on the operating platform. An example of this is client server development using Microsoft's COM architecture (Microsoft component services, 1998).

3.2. Modeling tools

A model of a design is usually associated with a graphical representation in the form of entity and flow diagrams, simulations or prototypes (Loucopoulos, 1995, pp. 131, 135). These models allow the developer to visualise the component relationships, workflow and processes involved in a system before it is developed (Pressman, 1997, p. 810). Apart from using visual and graphical modeling techniques and technology, modeling can also be done using design or modeling languages. One such a language is the "Program Design Language (PDL), also called a pidgin language in that it uses the vocabulary of one language (i.e. English) and the overall syntax of another (i.e. a

structured programming language)" (Pressman, 1997, p. 411). The Unified Modeling Language (UML) is an object oriented modeling language that is a combined effort of several design and modeling strategies. The general goals with UML are:
"To represent complete systems (instead of only the software portion) using object-oriented concepts; To establish an explicit coupling between concepts and executable code; To take into account the scaling factors that are inherent to complex and critical systems; To create a modelling language usable by both humans and machines" (Unification of methods, 1997).

Another such modeling language is the Specification and Description Language (SDL). This language is used to model event-driven, distributed systems. UML and SDL is used for graphical programming or visual software engineering (Bjorkander, 2000, p. 30). The advantage of using modeling tools like UML and SDL is that analysts and programmers can understand each other better, because they are using the same tool (Bjorkander, 2000, p. 35).

Mathematics can also be used as a modeling language for producing highly formalised designs or models (Mazza, 1996, p. 15).

3.3. Databases

A database is a software product that is used to manage and retrieve data. Initially, with the emphasis on functionality according to the functional approach, databases had essentially flat structures. With the shift in emphasis from functional to data components, databases became more structured and able to handle more complex data structures. As database technology got more sophisticated, the processing emphasis shifted from the front-end part of the application to the back-end.

Database technology has evolved over the years into four basic models.

3.3.1. Flat file database

These databases include indexed databases (such as ISAM, 2001) and provide indexed sequential access to data. This type of access is fast and can be sequential or random using an index key. Indexed databases can only be used for storing and retrieving data. The application using the database must handle the referential integrity and data validation (ISAM databases, n.d.).

3.3.2. Hierarchical database

A hierarchical database organises data in a tree structure. Data structures are directly linked which results in very fast data access and built-in referential integrity - "No child is allowed to exist without its parent" (Date, 1990, p. 758). However, this type of database model does not handle complex relationships very well.

3.3.3. Network database

The structure of a network database is essentially an extension of the hierarchical data structure. "In a hierarchic structure, a child record has exactly one parent; in a network structure, a child record can have any number of parents" (Date, 1990, p. 792).

3.3.4. Relational database

Relational database technology is currently the de facto standard for database management technology. The relational model is very flexible and can handle complex relationships. The components in

the database structure are all on the same level. The low-level structure of pointers between records is transparent to the user. Relational database technology also has a very flexible and powerful interface in the form of SQL (Structured Query Language) to define, retrieve, maintain and manipulate data, relationships and integrity between data components (Date, 1990, p. 249). In addition to this powerful interface, most relational database products also have a procedural programming language that works together with SQL to add further power, flexibility and ease of use to database programming in the form of stored procedures (Leavitt, 2000, p. 16).

3.3.5. Object-oriented databases

Whereas relational databases are structured in terms of relationships, object-oriented databases (OODB) are structured in terms of objects. OODB is therefore a natural match for object-oriented design and programming. The application and the database use the same object model. This is very useful for managing complex relationships among objects (Leavitt, 2000, p. 17).

Databases can also be integrated to combine and integrate data in different data structures. For example, different databases can logically be treated as objects and integrated through object interfaces to form a uniform universal database, although the physical implementations of the data remain fundamentally different. By having integrated access to various kinds of information in an organisation, a repository of the organisation's information can be created. Repositories can be seen as "database applications that contain meta-data, or data about data and are also central to the power of CASE technology" (The repository renaissance, 1999).

There are also interface technologies that can facilitate seamless access to different database technologies. Two examples

of these are Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC). There are even technologies that give seamless access to both structured and unstructured information. An example of this is Microsoft's OLEDB that is a gateway to various types of data sources (Understanding ODBC and OLE, 1997).

3.4. Programming languages

Programming languages have evolved over the years, in accordance with the changes in development methodologies, from predominantly functionally orientated languages to languages where emphasis is placed on data components. Programming languages have evolved further into object-oriented languages. Programming languages and their specific strengths and characteristics are related to the types of applications that they are used for (Mazza, 1996, p. 114). It is very important to choose the right language for a specific type of application (Vaughn, 1997). Apart from the influence of changes in methodologies and the type of application being developed, the changes in operating environments also determine the choice of programming language. The mainframe, the personal computer (especially the Windows environment) and the Internet are examples of this.

3.4.1. Procedural languages

With procedural languages the programmer specifies the order of execution using sequential statements, selection statements (IF, CASE), iteration (Loops), modules, functions and procedures (Mazza, 1996, p. 178). Procedural languages are further categorised in terms of use. For example: Fortran is traditionally used for scientific applications, COBOL for business applications and C for systems programming (Sebesta,

1993, pp. 6, 7). There are also general-purpose languages like Pascal and Basic.

3.4.2. Object-oriented languages

Object-oriented languages have all the structures of procedural languages. However, it also includes object-oriented constructs like inheritance and polymorphism (Mazza, 1996, p. 178). Object-oriented programming languages are divided into:

- Pure object-oriented languages like Smalltalk and Java in which the use of object-oriented programming constructs is compulsory;
- Hybrid object-oriented languages like C++ where object-oriented language constructs are part of the language, but the use of it is optional.

Object-oriented programming is also done in the Windows and Internet environments with languages like Visual Basic, Delphi, Visual C++ and Visual Java.

3.4.3. Internet development languages

With the development of applications on the Internet came a new set of programming languages. The Standard Generalized Markup Language (SGML) is such a language. It is accepted as an Internet standard by the World Wide Web Consortium - ISO-8879. The well-known Hypertext Markup Language (HTML) and the Extended Markup Language (XML) are sub-sets of SGML. These languages are used to define the appearances of Web pages (HTML) and data (XML) and is interpreted by an Internet browser. There are also scripting languages that make web pages more dynamic and able to be manipulated. Scripting languages are object-oriented or procedural language constructs and are sometimes even sub-sets of

a another language, like VBScript is from Visual Basic. Script is also interpreted by a browser (McGee, 1996).

3.4.4. Conventional programming languages versus query and declarative languages

Procedural and object-oriented languages are mainstream programming languages. By far most development is done in these languages. However, there are also other types of languages that are used for specialised types of applications. The structure of such a language revolves around the characteristics of the application being developed. Examples of these languages are functional languages like LISP and logic programming languages like Prolog, which are used for artificial intelligence programming (Sebesta, 1993, p. 6). Although the internal implementation of these two types of languages differs from each other, they share an essential characteristic in that they support declarative structuring. This means that the programmer only has to specify what is to be done and not how it is to be done, because the underlying structure facilitates this (Mazza, 1996, pp. 179, 180). Procedural and object-oriented languages differ essentially from functional languages like LISP, logic programming languages like Prolog and query languages like SQL in that "the program logic is embedded in the sequence of operations, instead of in a data-model (eg. the trees of Prolog, the lists of LISP and the tables of relational database management systems)" (Mazza, 1996, p. 178).

4. Applications

The methodology, operating platform, tools and techniques to be used must be chosen to match an application's characteristics. The way in which this is done, as mentioned already, depends on

where the emphasis is placed with regards to the system components, which are data and functions. This choice is also strongly influenced by changes in technology (Redmond-Pyle, 1996, p. 99)

The functional approach applies mainly to function-orientated applications and much emphasis is placed on processes or algorithms. Most scientific and engineering applications fall within this category. The information engineering approach finds application in data-driven systems where data is more important than functions and also more stable, like record keeping of business data. The object-oriented approach applies mainly to systems that relate to real world objects where data and functions are closely related (McConnell, 1993, p. 160). Object-oriented techniques also seem to be effective in accommodating complex, evolving systems (Meyer, 1999, p. 144).

Although these approaches differ fundamentally, they can be combined. Currently relational models mapped to relational databases, combined with object-oriented and component driven programming, is dominant in business applications. Also, using object-orientation, applications developed according to different development approaches can be integrated. Maps to different types of databases and multiple databases can also be handled by treating all of these different portions as objects with certain relationships between them (Shaw, 1996, p. 82).

There are many types of applications that are developed according to different architectures. Business applications currently revolve around the client/server architecture. People are very much involved in these applications, which means that there is a lot of interaction taking place and therefore the interface is important. Client/server applications need to be flexible, to adapt to change. These applications also need to be scalable to accommodate varying numbers of users. Client/server applications

are mainly data-driven and therefore the database is very important (Vaughn, 1997). Client/server applications have a specific architecture according to which applications are developed. Basically, a client/server application consists of a client component, requesting services from a server component. The server component then executes some services or queries according to the request and sends a reply or result back to the client. The server component consists further of business and data components. The business component contains the business rules of the application and therefore manages the application. The data component stores and manipulates the data of the application. It does this on command of the business component. The business component also communicates with the client component and regulates information to and from the client component. The client component is the interface to the user (Pressman, 1997, pp. 784 - 787).

With client/server applications, there is always a balanced emphasis between the client and server components. This can have the following configurations:

- All the processing intelligence is done on the client component. The server component is just a file server.
- All the processing is done on the server component.
  - The client component is just a screen emulation of the client processing on the server, much like the mainframe model (Dedo, 1997) or
  - The client component executes script generated on the server component like it is done on the Internet.
- The client and server components share processing responsibilities. Client and server validation and processing is done at the respective components.

All these configurations can be implemented in a conventional client server environment as well as on the Internet (Vaughn, 1997).

Apart from the general architecture, there are certain technological frameworks that accommodate client server applications.

The first is Microsoft's COM (Common Object Model). The COM model is implemented as ActiveX technology. The ActiveX technology framework integrates client, business and data components in both conventional client server and the Internet. An application can be designed to be distributed to many users or be used as a standalone application. Components can also be made to communicate while they are in different processes and even across a network (Maurer, 2000, p. 32). The second framework is Sun corporation's Java technology. In many aspects the Java model has the same architecture as Microsoft's ActiveX technology, except for the fact that the underlying implementations differ and that Java's components are more platform independent (Dragan, 1997, p. 38)

5. Developers

As in all other aspects of the software engineering process, people are also central to the actual development of a system or a solution. "For starters it's more about people working together than it is about defined processes" (Bach, 1999, p. 148). Because of this integrated process that involves people so much, a broad range of solid problem solving skills is necessary which involves much learning. "It's more about skill than it is about methods" (Bach, 1999, p. 149). When developers are too specialised, they might lack many of the broad range of engineering skills (Redmond-Pyle, 1996, p. 102). Because of the complexity of the

broad range of variables involved in the development of software and their interdependence, a whole range of problem solving skills is needed. In fact, a software engineer needs to have the skills of both a systems engineer and a programmer (Holmes, 2000, p. 159). The combination of skills has the effect that the whole is greater than the sum of its parts. Also, because technology evolves and changes so fast, specific skills can become outdated quickly. Broad base training will help a developer to adapt to changes quicker because a developer can use his or her knowledge of the whole process to incorporate new techniques and technology (Clark, 2000, p. 12). This broad base training will also amount to better communication and mutual understanding, which is crucial in software engineering.

6. What is needed

With the increase in system complexity, information needs to be structured and represented in a way that is closer to reality. This is needed to conquer complexity problems that otherwise result because of artificial design. To do this, what is needed is an environment that accommodates the development of flexible, adaptable applications. Representations that suit the information best need to be used. Development methods, techniques and tools will increasingly need to be able to deal with imperfect and incomplete data. Therefore data representations must change from being artificial, simple and rigid to a world of rich, flexible information that represents the real world of human information more closely (Korth, 1997, p. 141). To facilitate flexibility and power, a combination of the best-fit methods, techniques and tools to develop an application must be used. The advantages of doing this, however, can vanish if there is no proper coordination and communication. What is needed is a close integration of the processes, activities and information in the problem space with that of the solution space. What is further needed is a medium to facilitate and manage this integration.

This medium must be able to mimic and adapt to different development methodologies or paradigms and the variety of development tools, techniques and applications (Jetly, 1999). This medium must also accommodate top-down and bottom-up working approaches as well as the interaction between analysis and design activities. The medium must also facilitate the visibility, communication and understanding of knowledge concerning all these aspects (McConnell, 1993, p. 394). This is necessary, because people are pivotal to the development process.

7. Conclusion

This chapter examined the development of software in terms of the methods, techniques and tools that are involved. The application or product developed must satisfy the initial user requirements of either having a problem solved or enhancing an existing process. However, the borders of application sophistication and complexity are constantly expanding (Longstaff, 2000, p. 43). Equally, development sophistication has to adapt and keep up. Developers, who are pivotal to the whole process, need to develop the necessary skills. The methods, techniques and tools to their disposal need to assist and accommodate developers in improving their skills so that the whole of developers, methods, techniques, tools and application becomes greater than the sum of the individual parts.

This chapter concentrated more on the technical aspects of software engineering and the information to be communicated is based on standard terminology. Communication, however, can go astray because people are still involved. They have a variety of skills and knowledge and communicate from their technical points of reference. The problem with communication in this case is less

severe because knowledge is more formalised, but it is still very real.

Chapter 6

Hypermedia technology as a proposed solution

1. Introduction

In chapters three to five, the most important problems in the software engineering process were discussed. In these chapters the communication of development information and the use and limits of the documentation used are emphasised. This chapter proposes hypermedia technology as an extension of conventional documentation media and as a solution to its shortcomings. The proposed solution will be discussed in the context of the above-mentioned chapters.

In chapter three, the characteristics of software and software engineering, the problems surrounding it and what is needed to accommodate these problems were discussed. Software engineering is characterised as a complex, uncertain, non-linear, multi-disciplinary, human-orientated, communication-driven process.

In chapter four, information processing and the documentation in the software engineering process, the problems surrounding it and what is needed to accommodate these problems were discussed.

In chapter five, the methods and tools in developing software applications and what is needed to integrate it with the rest of the software engineering resources were discussed.

2. What is needed in general

From the chapters mentioned above, the following was identified in terms of what is needed in general.

To accommodate the problematic issues identified in these chapters, a form of representation is necessary that is flexible and adaptable enough to enable a person to view a system as a whole, while working in detail on parts of the system. People involved in software engineering must be able to cope with large quantities of information and variables and the relationships between them. Needed information must be visible, readily available and useful. The proposed form of representation or medium must also accommodate change and integrate the variety of different disciplines involved in the software engineering process. Taking the human factor in software engineering into account, combined with peoples' different personalities and perspectives, this medium needs to facilitate effective communication, coordination, organisation, processing, integration and maintenance of structured, unstructured, complex, dynamic, incomplete and evolving information (ref. Chapter 3).

Formal and informal information must be integrated. This must be done to enable those involved to reach a common understanding of the problem and its solution. The medium must also accommodate these people in the collaboration in problem solving activities and sharing of information. The information needs to be more visible in real world or problem specific terms. Information in the problem and solution spaces also needs to be integrated more closely (ref. Chapter 4).

Apart from accommodating and adapting to different types of information, people and environments, the medium also needs to be adaptable to various problem solving methodologies and activities (ref. Chapter 5).

3. Hypermedia in general

Hypermedia is an integrating technology with its potential to unify diverse media, tasks, information structures, applications, software, hardware, users, technologies and geographic barriers (Jetly, 1999). This unifying quality provides for a seamless multi-faceted environment with functionality that persists between the above- mentioned components rather than being dependent on any of them. This model provides the means to increase the quality of heterogeneous information and to increase the ease with which it can be used (Woodhead, 1991, p.10; Andersen, 1999).

These characteristics make hypermedia systems powerful tools for education, communication and cooperation. This will play an increasingly prominent role in the manipulation of, and access to information. Hypermedia technology is flexible and dynamic enough to accommodate changes in the structure and content of a body of information (ref. chapter 2).

Hypermedia technology accommodates the problem solving process because it integrates activities like analysing, interpreting and forming conclusions. Hypermedia technology also accommodates creative problem solving. It is well suited for creative activities like brainstorming, lateral thinking, idea processing and the use of analogies to trigger new ideas (Nielsen, 1995, p. 105). Information can be internalised in a constructive rather than a receptive way. Hypermedia technology also facilitates learning activities like remembering and conceptualisation because it "complements visual memory, which is a commonplace principle of human learning" (Chun, 1995, p. 97). Learning is a very important part of problem solving and therefore also of software engineering.

4. Hypermedia technology in support of software engineering characteristics

4.1. Structure

Hypermedia technology relates to software engineering structurally. Although hypermedia systems can be designed to accommodate any structure, hypermedia technology, like software engineering, inherently has a non-linear structure. This is in accordance with the fact that problem solving also has a non-linear structure. "Software engineering thus inherits its non-linear characteristic from human nature. This complexity and the apparent chaotic organization of human activity is natural and relates to the richness of the creative process" (Nanard, 1995, p. 51). Hypermedia technology is also suited for this area of problem solving because of its potential to integrate information in both the problem and solution domains.

Hypermedia systems are interactive systems. "Interactive systems are more powerful problem-solving engines than algorithms" (Wegner, 1997, p. 81). The reason for this is that algorithms cannot adapt interactively. Linear development models, like the Waterfall-model, are, in a sense, algorithmic in nature. This is contrary to non-linear, feedback models of development, which can be defined as being interactive and grounded in external reality with its incomplete information. Through this model physics and cognition can be modeled empirically (Wegner, 1997, p. 91). Knowledge acquisition is a process of design and is gained through the mechanism of evolutionary epistemology (Spiro, n.d.). Programming is re-defined as learning and experimenting based on software design. "Interactive models provide a unifying framework for understanding the evolution of computing technology, as well as interdisciplinary connections to physics and philosophy" (Wegner, 1997, p. 91).

4.2. Complexity

Software engineering is complex mainly because of size and because people are involved.

Hypermedia can accommodate problems associated with complexity to a great extent - "open hypermedia systems have been used to address the complexity and heterogeneity of large-scale software development" (Andersen, 1999). With hypermedia technology it is possible to manipulate the structure as well as the content of a system of information. The developer has a high level and simultaneously a detailed perspective of the system, therefore bridging the gap between different abstraction levels (Jetly, 1999). Hypermedia technology also assists developers in coping with massive amounts of information by increasing the connection density of information items to accommodate the mental capacity of the developer (Roth, 1994, p. 164). In cases of extreme complexity, people resort to heuristics and intuition in solving problems (Glass, 1995, pp. 46, 80). Hypermedia technology accommodates the use of intuition and heuristics. "With hypertext we connect things at the speed of a flash of intuition. Hypertext reading and writing supports the intuitive leap over the traditional step-by-step logical chain" (Heim, 1993, pp. 31, 96).

4.3. Multi-disciplinary nature

Software engineering is multi-disciplinary in nature. It is both a science and an art form. "These two sides of the software engineering process are not independent but part of the same development activity" (Nanard, 1995, p. 50).

Hypermedia is a multi-disciplinary technology. Hypermedia systems are used for research and application in diverse areas that range from philosophical to technical. "Whereas the bulk of hypertext

literature continues to deal with the technical problems of design and implementation that were the focus of most of the early literature, authors are turning with increasing frequency to the epistemological, philosophical and sociological consequences of hypertext, and borrowing methods and terminology from disciplines far removed from computer science. It may be that hypertext studies have reached a kind of intellectual crossroads, where the technical problems have become sufficiently familiar that it is now possible to address the consequences of this new form of literature as a new literary form" (Harpold, 1991).

Hypermedia technology is also technology independent and is strongly related to a whole range of computer-based technologies like:

- Knowledge-based systems;
- Frame-based systems;
- Rule-based systems;
- Project management;
- Systems development tools like CASE (Computer Aided Software Engineering);
- Natural language systems;
- Database technology, particularly relational and object oriented databases;
- Artificial intelligence technology like:
  - Neural networks,
  - Expert systems;
- Text retrieval systems;
- Computer graphics;
- Interactive media technology;
- Computer-based training;
- Distributed client/server network systems like the Internet;
- Component-based embedded software.

These components can be embedded like the different media in a hypermedia system is embedded in the structure, thereby extending a hypermedia system (Woodhead, 1991, p. 10; Jetly, 1999). Embedded components and hypermedia technology are also incorporated into operating systems like Windows and Macintosh, as well as in development tools for these operating systems.

A hypermedia system can integrate diverse systems that are otherwise not easily integrated. It has the potential to do this by accommodating different information structures and media in one medium. Different systems can be tightly integrated or a meta-level of integration can be provided that ties these systems together. In the context of software engineering, the information of a system can therefore be integrated as an integral part of a developed system. But not only can different components of a system be integrated. Through a hypermedia system, all aspects of the software engineering process, including people and problem-solving perspectives, can be integrated and accommodated (Jetly, 1999). Even seemingly unrelated aspects like rigid, linear, formalised engineering procedures and creative, unstructured processes can be integrated. "The hypermedia paradigm is used also to smoothly integrate the formal (used by the machine) and informal (used by the human being) knowledge representations" (Schwabe, 2001). In the same manner, logical and analogical thinking processes can also be integrated. All this integration is done through integrating the information that results from these aspects. This accommodates flexible communication (Roth, 1994, p. 151). Hypermedia technology also links human and machine created information and offers closer man-machine coupling (Bell, 1997, p. 32).

4.4. Human orientated

Software engineering is a human orientated process. Hypermedia technology is a human orientated technology. "Hypermedia technology permits access to information in a manner similar in structure to human thought processes" (Vatcharaporn, 1994, p. 105). "Hypermedia technology, is from a functional perspective similar to functional level models of neurology and the higher level cognitive models of human associative memory, which are also used in artificial intelligence technology" (Woodhead, 1991, p. 136).

Developers are also curious people by nature (McConnell, 1993, p. 757). In hypermedia technology, the emphasis in regard to information processing, is on discovering and forming new ideas. Therefore, hypermedia systems can be used to create documentation that enable people to learn and explore.

A hypermedia system enables relatively easy use of and access to a body of information. "Hypermedia enables the author to create links and relationships between a large number of documents and the reader to locate and follow the links" (Drori, 1997, p. 35). For information to make an impression on someone in this sea of information, it must stand out and be colourful to make an impression on someone. To do this effectively, it should engage as many senses as possible. According to self-help expert Anthony Robbins, information must touch us emotionally to catch our attention. Hypermedia technology, with its structural diversity and range of media, can accommodate this to a great extent. Different users' perspectives are also taken into account (Brun-Cottan, 1995, p. 62).

4.5. Communication

Communication is an extremely important aspect of software engineering. "Communication coupled systems represent the most

flexible and ultimately the most powerful strategy for the coordination of multiple possibly heterogeneous, distributed sources of knowledge" (Cucchiarelli, 1998, p. 54).

Hypermedia technology can assist people in communicating in spite of their differences in thinking, interpreting and problem-solving. Like hypermedia technology, communication is also non-linear in structure (Van Schoor, 1986, pp. 4 - 10). Not only can information be presented in different ways as to accommodate these differences, but information can also be presented uninterpreted, so that misinterpretations and individual perspectives do not hamper communication. "One person's experience may not correspond to another's, and subjective judgement comes into play as to whose opinion is correct. Usually the person with greater authority wins. With the ability to quantify the effects through simulation, a much greater degree of insight and understanding can be brought to bear on the decision-making process. Thus simulation can be a significant influence in communication and consensus building" (Christie, 1999). Hypermedia information also has an analogical link with the objects the information is about. This brings information as close to reality as possible. These representations can provide a valuable link between the problem, domain and possible solutions.

Apart from structural flexibility, hypermedia technology further accommodates communication by extending the communication bandwidth through integrating a variety of media (text, sound, graphics, video, and animation) (Narayanan, 1997). Hypermedia technology facilitates communication-driven information.

5. Hypermedia technology in support of information processing and documentation

5.1. Information processing

Hypermedia technology makes it possible to record, document, manipulate and organise development information. Technical and abstract information can be manipulated to form meaningful patterns of richly structured, interconnected data. Information can be structured to convey and externalise the organisational structure of a domain or a subject. The information is further complemented because it is closer to reality and richer in detail than processed information with a conventional structure. With the multiple media involved, the developer can experience, observe and obtain feedback on what he or she is trying to understand (Brun-Cottan, 1995, p. 70).

The knowledge structure that a person's frame of reference consists of, being an associative network, can therefore be seen as a hypermedia system (Vatcharaporn, 1994, p. 105). The hyperstructure is a natural way in which information is processed and stored. Apart from processing and storing information very effectively, hypermedia technology is also regarded by professional communicators as a breakthrough in communication technology for the transfer of large amounts of knowledge. This is true especially for task-orientated and technical information that must be in a format that allows efficient access to it.

Knowledge gained from processing information in this way, is designed and created rather than interpreted. The emphasis is on knowledge, rather than on information. Hypermedia technology accommodates the management of the interrelationships of the knowledge of an application.

5.2. Collaboration and sharing

Hypermedia technology supports the collaboration process (Jetly, 1999). While gathering information, the reader discovers and

socially constructs a knowledge base that reflects his or her understanding of what is investigated. During the collaboration process, all individual interpretations are negotiated with the group (Roth, 1994, pp. 154, 155). This interactive process results in a better collective knowledge. The whole of the process then becomes greater than the sum of its parts.

Hypermedia technology enables the sharing of information irrespective of media, format or structure. A hypermedia system can connect and share information:

- With different configurations;
- Between different databases;
- On different locations (Drori, 1997, p. 35).

Hypermedia technology enables people to effectively share information, because individual preferences and differing perspectives can be accommodated. "People differ in how they approach learning of new ideas and concepts while solving problems" (Vatcharaporn, 1994, p. 101). These differences are important for finding good solutions, but can be detrimental to communication and therefore to understanding, if they are not all part of an integrated whole. Hypermedia technology facilitates the management of these issues by accommodating the individual and social aspects of problem-solving. "Hypermedia technology makes the cognitive process common ground, which transforms collaborative work" (Jonassen, 1989, p. 16). "Hypermedia technology adapts to different cognitive styles facilitating social interaction, problem-solving and concept formation" (Chun, 1995, p. 111). By accommodating and facilitating the integration of these differences into one medium; brainstorming, generating ideas, argumentation, problem negotiation and co-operative design are stimulated (Jetly, 1999).

Hypermedia technology enables people involved in the development process to report their interpretations of the shared content throughout the process. This information can be made available to people who are geographically removed from each other through networks like the World Wide Web in the Internet, Intranet or Extranet environment (Jetly, 1999). The structure of the information does not have to change because the World Wide Web is a hypermedia environment.

5.3. Presentation of information

The effective presentation of information to a variety of audiences is very important, because there are so many parties involved in developing a software system. With hypermedia technology, information can be presented in virtually any format. This has the implication that information can be manipulated to fit the particular structure or purpose of a presentation. (Schwabe, 2001). Information in a hypermedia system can also simulate the structure of the system being developed. This way the information can be presented as it is. Apart from presentation advantages, reporting can also be very versatile.

Hypermedia information can also be very visible as far as understanding what is being portrayed is concerned. "Experience with the graphical presentation of data has shown that it enables certain types of complex information to be assimilated much faster and more easily and that user environments become more convenient and enjoyable with the addition of graphics" (Sodan, 1998, p. 105).

Hypermedia technology accommodates the representation of information on a number of dimensional levels. When presenting the model for a potential solution on a two-dimensional plane,

some information is lost because multi-dimensional objects from the problem domain are reduced to a plane with less dimensionality. With a three-dimensional representation, the objects in the model are closer related to the real domain-objects than with a two-dimensional representation. A three-dimensional representation can also be more functional, for example, objects in the model can be rotated, zoomed into, walked through and viewed from different angles (Feijs, 1998, pp. 74, 75).

5.4. Documentation

Hypermedia technology is very useful for packaging reference information because individual divergent pieces of information can be linked in context. Different levels and types of documentation can be seamlessly integrated as a whole (Jetly, 1999).

Reference materials have the following general characteristics:

- Information is organised into fragments;
- The fragments relate to each other;
- They are organised into discrete sections and contexts;
- The user needs only a small fraction at a time;
- They are organic in that they expand gradually during their life-cycles;
- They are not used in a linear fashion;
- They are difficult to manage.

The above-mentioned characteristics make reference documents very good candidates for hypermedia technology (Woodhead, 1991, p. 66).

Hypermedia technology is ideal for gaining access to development documentation. The information in projects that have a very definite, fixed structure, as well as projects with an emerging, changing structure, can be accommodated by hypermedia technology with its structural flexibility, which can range from being highly structured to having no structure at all. Links and relationships can be created between documents and used to navigate through the knowledge base (Drori, 1997, p. 35). Users and developers can also add to this knowledge base. This structural and functional diversity amounts to a documentation flexibility that cannot be met by conventional means. When a problem is not explicitly stated, hypermedia technology is also much more effective than linear text formats (Jones, 1992 p. 146). This is very important because the real problem in software engineering is very often not stated explicitly. This means that developers have to spend a lot of time constructing knowledge, which makes hypermedia documents ideal for software engineering documentation.

"Following the Hypermedia philosophy of maximum access, allows developers to analyze system information to identify structural relationships that are not possible with conventional linear documentation media" (Bieber, 1995, p. 103). This can result in volumes of documentation that are difficult to handle. However, hypermedia technology is very useful in coping with large amounts of information (Roth, 1994, p. 164). Hypermedia technology follows a systems approach to information integration. "The systems approach to handling the information explosion phenomenon is to enable a copy of the original document to be saved and permit access to it in different ways, as opposed to the alternative of distributing many copies of the same document" (Drori, 1997, p. 35). The result is a reduction in the total volume of documentation and better control and management of updated information.

The use of hypermedia technology, however, does not implicate that conventional documentation should not be used, but rather that it should be integrated into the hyperstructure of documents.

Apart from information that is specific to the development of an application, hyperlinks can also be created to point to literature that can assist development in general (Jetly, 1999).

6. Hypermedia technology in support of the development process

6.1. Development approaches and methodologies

Throughout the software engineering process, analysis and design activities continuously take place. As already mentioned, these two activities are interactive and are connected by a feedback loop. However, analysis and design must be understood in a software engineering context. In this context, the emphasis is on experimentation, discovery, exploration and synthesis, rather than on analysis and absorption of standard versions (Woodhead, 1991, p. 68, Norman, 1994, p. 35). People do the same activities when using a hypermedia system.

Hypermedia technology can accommodate development extensively, because development information can be organised to mimic the structure of analysis and design activities. Information can be structured in a hierarchy to accommodate a deductive, top-down functional approach. Information can also be structured in a connectionist model of associative links and nodes to accommodate an inductive, bottom-up object-oriented or data approach. A combination of both information structures can also be integrated to accommodate the use of top-down and bottom-up approaches interactively (Mazza, 1996, p. 209). This results in an iterative model, which is how the software engineering process is

structured. This interactive, iterative approach also accommodates user-centered and participatory development because it is similar to human interaction (Nanard, 1995, pp. 50, 51; Brun-Cottan, 1995, pp. 61, 62).

Hypermedia technology also supports and complements any development methodology, paradigm and technique, but is especially close to the object-oriented approach and techniques. A hypermedia system of development information maps as naturally to the object oriented application as the application maps to the data in an object-oriented database. In light of this, hypermedia technology must be integrated into the design of an application and augment both interface and analytical activities (Bieber, 1995, pp. 99, 100).

As was mentioned in chapter five, within the framework of a methodology, the developer aims to work with smaller, more manageable portions of the system. This can be an isolated module or a higher level of abstraction.

A module is a self-contained unit that encapsulates a unique piece of information or design or code. Modules are equal to the nodes in hypermedia technology (Nielsen, 1995, pp. 50, 309). Hypermedia technology adds some useful functionality to development modules or nodes. Nodes can be linked to other nodes, providing continuity between them. Linking or referencing gives structure to an otherwise fragmented group of nodes. The advantage of being able to link nodes is that modules or nodes can be molded into different structures or frameworks. Another advantage is that when the content of a node is designed to represent a single idea, nodes will be modular, avoiding duplication and promoting re-use (Andersen, 1999). Therefore, using hypermedia technology, a group of modules can be moulded into an integrated system.

Like modular development, the use of abstraction is also invaluable in the development of software. Abstraction is as much an integral part of the hyper-structure as it is of software engineering. Abstraction allows users and developers to view the system in terms of its separate but related components on different levels of detail. The advantages of this are consistency and modularity (Nanard, 1995, p. 52). Hypermedia technology facilitates a seamless jumping or moving between different levels of abstraction.

A common and very effective way of making the design of a system visible is to model it. In terms of modeling, hypermedia technology enables information to be structured and manipulated to create a model that is a better match of model and reality, than is the case with conventional models. A system's components can be modeled in terms of their relationships to each other and integrated with the process or workflow model of the same system (Bieber, 1995, p. 101). These models can be static or dynamic. From the models, there can be hyperlinks to information or other system and development components. Models are therefore integrated with other aspects of development. Using various media with this flexible structure enables the creation of powerful information-rich models.

Using Virtual Reality and three-dimensional animation in coherence with hypermedia technology, this powerful modeling functionality can be extended even further (ref. Chapter 6 - 5.3). Models or representations can be enriched by building realistic, interactive user-involved simulations of a problem domain, as well as different solutions and even solutions integrated into the problem domain (Feijs, 1998, p. 75). Interactive systems are powerful modeling tools. They are grounded in reality, are rich in behaviour, embraces incompleteness and are empirically driven models. "Interactive models provide a unifying framework for interdisciplinary

connections to physics and philosophy" (Wegner, 1997, p. 91). Hypermedia technology also accommodates any type of modeling whether it is language-based or graphically orientated, because of its flexibility and variety of structures and media.

6.2. Systems development life-cycle

6.2.1. Strategic phase

The development of a software system starts with an initial requirement from the user for such a system. What is very important, is that this knowledge of the user's requirement must be viewed in the context of the user's working environment with its processes and variables, as well as the development environment with its variables, processes and resources. Proper planning must be done in this regard. Planning is done according to the availability of resources. All the variables and resources have to be integrated and synchronised (Boehm, 2000, pp. 114 - 116).

Hypermedia technology accommodates the integration, coordination and management of all the information of a system on different levels. Hypermedia is also described as the ideal application manager because it can unify diverse systems seamlessly. Hypertext-based tools can also be used to manage requirements (Jetly, 1999).

6.2.2. Development phase

Analysis

In the development phase, the developer has to refine his or her understanding of the requirements. In order for a system to be developed successfully, developers need to have a good solid

understanding of the user's problem in the context of the working environment or domain, as well as of the organisation and processes that define the domain. Although it is ideal, it is, however, not always possible for a developer to be in the user's domain whenever needed. Hypermedia technology can be of much value in such a situation as well. Apart from describing the domain in text and graphics, animation and video can also be used to communicate the environment clearly (Brun-Cottan, 1995, pp. 61, 62). Using video is especially valuable because the information is not interpreted. "Using video-based hypermedia allows for studying real world practices while still being able to access and explore associated information" (Nonnecke, 1995, pp. 185, 186). Hypermedia technology can accommodate the feeling experience that is needed to properly understand a user's requirements. "Video personalizes the communication" (Pressman, 1997, p. 830). A hypermedia system can also be used as a prototyping tool for analysing and clarifying user requirements (Roth, 1994, pp. 158, 161, 162).

Design

According to these analysis, there is an iterative cycle of further refined and detailed designs. These designs progress from being problem-specific to being solution-specific and from being high-level logical designs to being low-level physical designs. Hypermedia technology can be used to unite all the designs with all the analysis and other development information and artifacts. Hypermedia technology also accommodates development aspects such as:

- Interface development (Roth, 1994, p. 156);
- Co-development (Brun-Cottan, 1995, p. 65);
- Program or process simulation (Mazza, 1996, p. 209).

Code

Although system specifications are supposed to reflect the working of a system, it sadly is an ideal that does not usually realise in practice. This often leaves program code as the only view to the insides of a system. Viewing a system at code level, however, is not an easy task because the system is visible only on a detail-level and therefore the "big picture" is lost. Using code visualisation software and techniques, one can zoom in and out of different levels of code from a global overview to the lowest detail-level (Ball, 1996, pp. 36, 37). This is done by using textual and graphical representations of code interactively. Links can also be created to re-usable code components (Andersen, 1999). Hypermedia technology can also be used to link external information to code in order to integrate all development aspects (Jetly, 1999). This will make system information an integral part of a system and code an integral part of system documentation.

Testing

When doing testing and implementation, using hypermedia technology, errors, limitations and other issues that surface through testing can be documented and linked to other relevant development aspects (Jetly, 1999).

6.2.3. Implementation phase

To install a system, documentation is needed to guide the installation. Hypermedia technology can be very useful in reducing the complexity of an installation by guiding the installer through a simulation of the real installation (Narayanan, 1997).

Because applications or systems are developed for people to use, these people need to know how to use it. Hypermedia technology accommodates learning and is very often used for computer-based training (CBT). Tutorials and other training methods and

materials can be designed to train the users in getting to know the working of the system and how to use it. Context sensitive systems can also be designed by using hypermedia technology and therefore it makes technical information more accessible (Woodhead, 1991, p. 66; Chun, 1995, p. 107).

6.2.4. Maintenance phase

Because someone other than the person or persons that developed the system very often does maintenance on a system, the person that does the maintenance is extremely reliant on system documentation. If all the documentation is integrated in a hypermedia system, the maintainer can get a view or perspective of the system on all levels through links between documents and between documents and code (Jetly, 1999).

6.3. Tools

6.3.1. Computer Aided Software Engineering (CASE)

CASE (Computer Aided Software Engineering) tools are valuable tools in creating an integrated, process-driven software engineering environment. Hypermedia technology complements development technology such as CASE because the existing integrated body of information, constructs and programs within a CASE system can be extended to incorporate other aspects that is part of development but not incorporated in a CASE system. A CASE system is a very controlled environment and hypermedia technology complements it by incorporating hard and soft software engineering techniques, thereby making it more flexible (Jonassen, 1989, p. 35). Hypermedia technology can therefore be used to manage a CASE environment by drawing together different working environments and managing them (Drori, 1997, p. 35).

6.3.2. Databases

Hypermedia technology accommodates a variety of database structures that range from text documents to highly structured databases, but is particularly close to relational and object-oriented databases. Either of these types of databases are frequently used as the back-end of a hypermedia system where the information is stored (Nielsen, 1995, p. 131). A hypermedia system of information can therefore use the same database as the application being developed, resulting in a closer integration between the application and the information surrounding it. Hypermedia technology also bridges the compatibility gap between different types of databases and manages the transfer of information between them.

There is also an increasing need for databases to represent more real data (data that is closer to reality) to conquer complexities and problems that are associated with artificial, simplified representations. Hypermedia technology, with its ability to integrate databases and represent information in a way that is close to reality, can accommodate this need.

6.3.3. Programming languages

Programming languages like C++, Java, Smalltalk and others are used to extend hypermedia systems. This has the implication that hypermedia technology as a documentation and presentation medium can be extended indefinitely in terms of its flexibility and functionality. This has the potential effect that programming used in presentation and documentation can be used in the application, thereby facilitating to some degree a seamless

progression from development information to programming language code for an application or system solution.

6.4. Applications

With problems to be solved becoming more demanding and complex, applications need to represent reality more closely to avoid problems associated with artificial, simplified solutions to complex real world problems.

In terms of structure and content, hypermedia technology can adapt to applications and technologies of vastly different natures. Apart from accommodating different user preferences and differences in subject matter, hypermedia technology can also accommodate various types of different structures as far as applications are concerned. This includes well-structured applications with the following structures:

- Conceptual structures with pre-determined relations;
- Task related structures resembling the processes and activities of a task;
- Knowledge related structures that are based on an expert's knowledge;
- Problem and solution related structures that simulate problem-solving or decision making;
- Chronological, sequential structures;
- Parts and whole structures;
- Cause and effect structures;
- Antecedent and consequent structures (Jonassen, 1989, pp. 48, 53).

Hypermedia technology is also well suited to less or open structured applications. "Hypermedia technology is invaluable in

applications that explore alternative structures in which the domain structure is not well understood at the outset or changes during the course of a task. Many of these applications involve the collection, comprehension and interpretation of diverse materials. These activities are information-intensive like analysis, design or evaluation and are collaborative efforts" (Marshall, 1995, p. 88). These types of applications mentioned above are identical to activities in the initial phases of the software engineering life-cycle, like requirement analysis and prototyping. Hypermedia technology also has a strong resemblance to so-called "soft computing" applications, like neural network and fuzzy logic applications.

In terms of development, hypermedia technology is strongly related to distributed systems like Internet and Intranet applications. This is evident with http (hypertext transfer protocol) that forms the backbone of the WWW (World Wide Web) and the use of HTML (Hypertext Markup Language) in Web applications.

Of utmost importance, however, is the close relationship that hypermedia technology and hypermedia systems have with client/server technology. The architecture of hypermedia systems is especially close to the client/server architecture, and hypermedia technology is also very similar to object-oriented and relational technologies. In general, hypermedia systems consist of three architectural levels or layers:

- A presentation layer which is also the client-interface and which is usually a Graphical User Interface (GUI).
- The second layer is called a Hypertext Abstract Machine (HAM) layer, which manages the nodes and links and therefore the structure of the information. The heart of a hypermedia system is the Hypertext Abstract Machine (HAM) (Andersen, 1999).

- The last layer or database layer is where the information is physically stored. The database layer can range from simple, flat text files to sophisticated structures like relational databases, for example SQL server or Oracle (Nielsen, 1995, p. 131).

The operations of each layer are largely transparent to the other layers which makes layers modular and weakly coupled. The operations of the system as a whole are also transparent to the user. The result is a set of seamless interfaces between layers and between the system and the user.

The architecture of hypermedia systems resembles the 3-tier client/server model to a great extent. The 3-tier client/server model also consists of three layers or levels or tiers. They are the presentation, the business and the database layers. The presentation layer interfaces with the user much like the presentation layer of a hypertext system. The business layer contains the business rules and the operations necessary for enforcing those rules and correlates with the Hypertext Abstract Machine (HAM). The database layer also does very much the same for both client/server and hypermedia systems in storing and manipulating information.

This has the implication that documentation, development and application can be integrated naturally. With the integration and management capabilities of hypermedia technologies, a hypermedia system can be the ideal application manager in a client/server development and application environment. The hypermedia structure also enables seamless integration between applications by treating these applications as nodes within a larger context (Andersen, 1999).

7. Conclusion

Hypermedia technology is a human, as well as a system orientated technology with capabilities to accommodate human thinking, learning, communication, collaboration, problem-solving and the development of software. Hypermedia technology can accommodate the problems associated with the broad spectrum of characteristics and activities of software and the software engineering process. Because of its capabilities hypermedia technology can add value to software engineering as a problem-solving process.

Chapter 7

Conclusion

1. Introduction

This is the concluding chapter. The hypothesis, as stated in chapter one, was evaluated in terms of the research results in chapters two to six.

2. The problem

The problem that was stated in chapter one is as follow:
The problems surrounding the software engineering process can largely be attributed to the lack of proper coordination and integration of information used for development.

3. The hypothesis

The hypothesis that was stated in chapter one is as follow:

The characteristics of hypermedia technology, as far as the coordination and integration of information is concerned, seems to provide a solution to the problem of coordinating and integrating the information used for development as encountered in the software engineering process.

The coordination and integration of development information involve the transfer of information. Because people are so vitally involved, it can be defined as a communication problem. This strongly relates to the characteristics of software and software engineering and the processes and activities involved.

It is hypothesised that hypermedia technology can help to solve the communication problem mentioned above.

This problem, viewed from a communications perspective, can be broken down into the following:

- Communication problems between the user and the developers;
- Communication problems between people in the development process;
- Communication problems between developers and development information;
- Communication problems between user and user documentation;

4. The hypothesis as researched

The communication problems mentioned in the hypothesis is further elaborated upon and placed in context of the software engineering process and activities in chapters three, four and five. From these chapters, the vital importance of effective communication is evident and the problems associated with it are emphasised.

In chapter three, communication problems were described in terms of the inherent characteristics of software and software engineering. From this chapter the importance of effective communication in the software engineering process is evident. Communication is viewed as one of the core characteristics of the software engineering process. The communication process is, however, to a great extent hampered by problems that occur as a result of the other characteristics of software engineering that was mentioned in this chapter.

Communication revolves around the involvement of people. To complement this, software engineering is, amongst others, also characterised as a human-orientated process (Wood, 1998).

Size, as an attribute of complexity, is the major cause of communication problems because with an increase in the size of a project, more people are involved. With more people involved, the number of communication paths increases exponentially (McConnell, 1996, p. 28). People also have different perspectives and the greater the number of people involved, the greater the communication risk. In light of this, communication problems are further increased because software engineering is a non-linear, unpredictable, interdependent, creative process involving various different disciplines, levels, phases, people and activities (Olson, 1993, pp. 35, 55). These aspects result in large quantities of complex information that has to be communicated between the people involved.

In addition to the communication problems already mentioned, information is very often communicated through formally structured text-based documentation which is a poor communication medium compared to direct communication.

Apart from the negative effects that the characteristics of software engineering can have on communication, ineffective communication can also have adverse effects on software systems being developed. Ineffective communication, to begin with, can therefore result in a vicious cycle of ineffective software engineering.

In chapter four, communication problems were viewed from the perspective of processing information and documenting it. From this chapter it is evident that the problems in communicating information begins before the actual communication takes place. This has to do with how people internalise information.

Perceptions are not objective, predetermined representations of reality, but is dependent on cognitive processes. When people interact with a certain environment, they respond creatively and

do not just merely adapt to their surroundings (Matthews, 1999, p. 27). When a person processes information, it becomes knowledge and is integrated into his or her frame of reference. This frame of reference or knowledge base is unique to everyone. Apart from this, a person's perception is also influenced by that person's historical and cultural background and personal experiences.

All these factors give knowledge and perception a very subjective nature, with communication being tied to perception (Burgoon, 1995, p. 109). The solution to this problem is effective communication. However, language, as the vehicle for communication, includes ambiguity as one of its characteristics. Therefore, the medium for transferring these subjective perceptions allows for different interpretations of the same message. Effective communication therefore requires to be a process that involves a lot of feedback, correcting and fine-tuning to synchronise the perceptions of the communicators (Neill, 1992, p. 11).

In light of these problems concerning communication, information is usually processed and transferred through several levels of interpretation, because problem and development information are communicated through different software engineering phases by different people.

As was mentioned before, communication of information is mostly done through text-based documentation. This results in a further reduction of the richness of information or communication bandwidth (Neill, 1992, p. 152).

Chapter five centered on the technical aspects of software engineering and the information to be communicated, is more formalised in nature. However, people are still involved. These people usually have a variety of highly specialised skills and

knowledge. This high specialisation, combined with the fact that designs cannot contain all available information, can pose challenges to effective communication (Winograd, 1995, p. 69).

Chapters three, four and five highlighted what is needed to accommodate these problems associated with the communication, coordination and integration of information. The characteristics of hypermedia technology (chapter two) are of such a nature that hypermedia technology can make a powerful contribution to solving these problems. Hypermedia technology, by virtue of these characteristics facilitates information in regard to:

- visibility and comprehensibility (Sodan, 1998, p. 105);
- integration and management (Roth, 1994, p. 151; Mazza, 1996, p. 209);
- integration of and adaptability to differing perspectives (Brun-Cottan, 1995, p. 62; Chun, 1995, p. 111);
- structural flexibility and adaptability (Bieber, 1995, p. 103);
- different levels of abstraction (Ball, 1996, pp. 36, 37);
- collaboration of processing activities (Roth, 1994, pp. 154, 155, Jetly, 1999).

Hypermedia technology provides a richer documentation medium with a greater communication bandwidth than conventional documentation media like text-based documents (Jones, 1992, p. 146).

Hypermedia technology as a solution to the communication problems in the software engineering process, was discussed in chapter six.

The research of the problem and the proposed solution will now be consolidated into a table.

Tabular representation of hypermedia technology in relation to solving the problems as a result of software engineering characteristics and related aspects

Table 1 is a list of hypermedia characteristics.
Table 2 contains:

- the characteristics and related aspects of software engineering and software;
- problems as a result of these characteristics and aspects;
- what is needed to solve these problems;
- proposed solutions as listed in Table 1

Key to the tables
H = Hypermedia
SE = Software engineering

Table 1

Characteristics of hypermedia	
1	H information can be structured to adapt to different perspectives and subjects
2	Different information types and structures can be integrated in H
3	H information can be reused
4	H information has a higher bandwidth (associative structure and multi-media) than other media
5	H information is intuitive
6	H accommodates different levels of abstraction seamlessly
7	H accommodates free exploration
8	H accommodates manipulation and construction of information
9	H accommodates integration of media, heterogeneous information, applications, people, technology, geography
10	H support computing, communication, teaching, interaction, learning, thinking
11	H handles large volumes of information and reduces the cognitive overhead
12	H stimulates argumentation
13	H accommodates different views of same material
14	H accommodates simulation
15	H accommodates and stimulates analogy
16	H information is close to real world
17	H is technology independent
18	H integrates diverse systems
19	H stimulates creativity
20	H is human orientated
21	H supports collaboration, sharing
22	H is ideal for reference materials, development documentation
23	H accommodates modularisation, encapsulation, reuse and other problem-solving techniques and methods
24	H bridges the gap between different types of databases
25	H accommodates all applications especially C/S
26	H is the ideal application manager
27	H is dynamic and accommodates change
28	H has an associative structure
29	H is non-linear
30	H supports concept formation and understanding

Table 2

SE characteristics and issues	Problems involved	What is needed	Hypermedia solution as per Table 1
SE involves technical, social, organisational,cultural aspects	Information is diverse and not easily integrated	Need medium to integrate and coordinate information of all software engineering aspects	1,2,18,26
SE is a multi-disciplinary process	Information is diverse and not easily integrated	All the aspects involved need to be integrated as a whole	2,9,18
SE is based on intellectual content	Information is subjective and complex	Effective conceptualisation	10,12,15,19, 20
Software systems are flexible and dynamic	Conventional documentation is not flexible	Need adaptable, flexible documentation that accommodates changing, evolving systems	1,27
SE is complex	Simplified models and information do not represent the complexities involved	Need to structure and represent information in a format that is closer to reality to conquer complexity	1,16
	Software is largely invisible	Need a documentation medium to make information more visible	14
	Formal techniques are not always viable to deal with highly complex situations	SE need intuition to conquer complexity	5
SE has an element of uncertainty	Highly structured conventional documentation do not accommodate uncertainty effectively	Need integration of rich, imprecise, uncertain, subjective, complex and no oversimplified information	16
Size is a huge factor in SE	Huge numbers of variables and relations	Need to cope with huge amounts of information and relationships	11
	Different levels of abstraction	Need to understand whole while working on detail	6
SE is a non-linear process	Conventional documentation is linear	Need a non-linear documentation medium	28,29
SE components are interdependent	Manipulation of components may influence other components	Need medium to manage integration of components	26,29
SE is a interactive, iterative process	Conventional documentation is not interactive	Needs documentation medium that accommodates experimenting and exploration	7,8
SE involves heterogeneous	Conventional documentation	Need a medium to accommodate structured/	1,9,24

information	contains mostly homogeneously structured information	unstructured and formal/informal and logical/analogical and human/machine information	
SE is a human orientated process			20
SE involves human information processing	Is subjective and leads to miscommunication	Need to accommodate information processing and differences and integration thereof	1,2,9,12,13, 15,19
SE involves communication process	Conventional documentation does not communicate information well enough	Need a medium to extend communication bandwidth	4,10,21
		Need a medium to accommodate collaboration and sharing of information	21
Different people are involved	Conflicting perspectives, interpretations	Need to coordinate and integrate differences and communication	1,4,10,21
		Need multiple representations	1,13
SE involves learning	Conventional documents accommodates absorption of interpreted knowledge	Need a medium that accommodate construction of knowledge, understanding	10
SE involves different problem solving activities and methodologies	A conventional document is rigidly structured according to a specific methodology	Integrate and accommodate various different activities and methodologies	23
		Needs documentation medium to accommodate modeling, modularisation, abstraction, reuse	23

5. Conclusion

It is important to take note that software engineering is not a conventional engineering discipline, but has some interesting characteristics involving humans to a great extent. This human involvement includes human nature, background, the mind, information processing and communication and the problems associated with these aspects.

The characteristics of hypermedia technology accommodate and complement software engineering characteristics and integrate

well with software engineering.

In light of the problem at hand and the research being done, the following conclusion is made: Hypermedia is capable of making a large contribution to solving many of the problems related to coordinating, communicating and integrating software engineering information. It must however be noticed that hypermedia technology cannot be a substitute for effective, direct communication.

6. Future research

The following related areas for research is suggested:

- Further research on the representation of software engineering information using hypermedia technology;
- Further research on how hypermedia technology can practically be used as a documentation medium for software engineering information;
- Research on human and communication aspects in software engineering;
- Research on the influences of the human-orientated sciences on computer related sciences and vice versa.