2002

Systems Thinking in Extreme Programming

Peter Wendorff
ASSET GmbH, p.wendorff@t-online.de

SYSTEMS THINKING IN EXTREME PROGRAMMING

Peter Wendorff

ASSET GmbH, Am Flasdieck 5, 46147 Oberhausen, Germany
Tel. +49 208 6287220, Fax. +49 208 6287221
p.wendorff@t-online.de

ABSTRACT

Extreme Programming (XP) is a new software development method (SDM) that claims to be superior to other methods in some situations that are characterised by vague requirements and rapid change. XP has quickly gained a remarkable degree of acceptance in parts of the software engineering community. Interestingly, it has provoked a vivid and often controversial exchange of opinions, ranging from enthusiastic support to vigorous criticism. In this paper we look at possible sources of such fundamental disagreement. We use general systems theory as an integrative theoretical framework to analyse SDMs and expose their underlying, often implicit assumptions. We discuss some examples where the assumptions made by XP are fundamentally different from the assumptions made by most other SDMs. These fundamentally different assumptions indicate fundamentally different mental models. Mental models are central to systems thinking because they have a strong influence on our perception of reality and our behaviour. If they act as unconscious filters that limit our perception to what we expect, they can seriously impede our ability to learn. We conclude that mental models play a crucial role in the introduction of XP.

1. INTRODUCTION

Extreme Programming (XP) is a new software development method (SDM) that has become popular in the late 1990s (Beck, 2000). XP claims to be superior to other methods in some situations that are characterised by vague requirements and rapid change. It belongs to a new class of methods, called "agile" SDMs (Cockburn, 2001). These methods share a core of values and principles published as the "Manifesto for Agile Software Development" on the World Wide Web (The Agile Alliance, 2001). XP is by far the most widely used agile SDM at the moment, and even the first ever dynabook of the Institute of Electrical and Electronics Engineers (2000) has been devoted to it.

XP has quickly gained a remarkable degree of acceptance in parts of the software engineering community. Interestingly, it has provoked a vivid and often controversial exchange of opinions, for example, published in the dynabook mentioned above. The observation by Jawed Siddiqi of "both enthusiastic support and equally vigorous criticism" of XP in this dynabook indicates the need for an integrative theoretical framework for SDMs in software engineering. In this paper we use general systems theory as an integrative theoretical framework to analyse SDMs and their underlying assumptions.

2. SYSTEMS THINKING

General systems thinking was originally proposed in the 1950s as an analytical paradigm to stress the common foundations of different scientific disciplines like biology, psychology, the social sciences, etc. (Schermerhorn, 2001) The systems approach has proven very valuable in the study of organisa-

203

tions. For example, Scott (1998) uses it as a framework for the classification of different types of organisational designs. Another very influential writer has been Senge (1994), who has used the systems approach to focus on the abilities an organisation needs to develop in order to survive in a complex and turbulent environment. Weinberg (1991) proposed an application of systems thinking to software engineering management.

2.1. Organisations as Systems

The systems thinking view in organisational analysis reflects the need for adequate descriptions of organisations as social systems. Scott (1998) describes the historical view of organisations using three perspectives on social systems. The first perspective is the "rational system" view of organisations, that stresses the predominance of clear organisational goals that are pursued by almost all members of the organisation in a formally predefined way. In reality, however, clear organisational goals do often not exist or are sometimes ignored by the organisational members, and this realisation gave rise to the second perspective, the "natural system" view of organisations. The natural system view of organisations tends to focus on their internal affairs, whereas in reality any system operates in an environment, and often even the survival of an organisation may depend on an appropriate reaction to environmental stimuli. This has led to a third perspective, the "open system" view of organisations, which focuses on the dependence of an organisation on its environment.

2.2. Systems as Models

The systems theory perspective is widely used in management to reason about organisations as social systems. Schermerhorn (2001, pp. 81) defines: "A system is a collection of interrelated parts that function together to achieve a common purpose. A subsystem is a smaller component of a larger system."

Systems thinking uses systems as finite models of reality to study phenomena that can be observed in the real world. Naturally, these systems have a limited number of parts and interrelations. This results in an inevitable discrepancy between the models and reality. The models represent assumptions about the relevant parts and interrelationships that are appropriate to describe the real world. Only if these assumptions hold, the resulting models will have predictive value. Therefore, systems thinking as a discipline encourages to make key assumptions in models explicit.

2.3. Models and Decisions

As Pugh and Hickson (1997) point out the work by Simon, March, and others has shaped the understanding of decision making in management. One major conclusion of their work has been that the complexity of management problems usually by far exceeds the problem solving capacity of human minds, famously stated in the principle of "bounded rationality". As a consequence the mental models people use in the decision making process are often deficient. These flawed mental maps constrain human decision making resulting in suboptimal outcomes. Two common flaws in mental maps are elaborated by Weinberg (1991): First, humans routinely assume linear cause-effect relationships when in fact non-linear forces are at work. Second, the feedback effects that operate within a system are not fully understood.

3. EXTREME PROGRAMMING

Extreme Programming has been developed in the late 1990s. It has been made popular by a book published by Beck (2000). In the introduction to that book he states: "The book is written as if you and I were creating a new software development discipline together. We start by examining our basic assumptions about software development. We then create the discipline itself. We conclude by

204

examing [sic!] the implications of what we have created - how it can be adopted, when it shouldn't be adopted, and what opportunities it creates for business." (Beck, 2000, p. xix)

Beck (2000) limits the applicability of XP to small projects with up to 10 programmers, where all required functionality of the system can be verified through automated tests, and where a customer representative is continually available on-site as part of the software development team. Beck (2000, p. xv) claims that XP was particularly efficient "in the face of vague or rapidly changing requirements."

In the following we regard a SDM as an abstract description of a social system that is supposed to produce software. Assuming a systems perspective we focus our attention on key assumptions about software development implied by SDMs. We present some examples of constituting assumptions made by XP and will compare them to assumptions that we believe underly most other SDMs.

As said, XP is only applicable to a fraction of all software projects, and the following discussion is generally limited to that subset, as well as to those other SDMs that are applicable in the given situation.

3.1. Uncertainty

A look at critical success factors in software projects reveals that many reasons for project failure result from circumstances outside the project, like misunderstood users' needs, technological changes, business needs changes, and users' resistance to change (Reel, 1999).

The strategy employed by most SDMs is to analyse the project's environment until a reasonable degree of certainty is achieved. For example, Jacobson et al. (1998) propose that at least 80% of the requirements should be elaborated before software design can begin. There are two crucial assumptions that underly these approaches. First, it is assumed that most of the requirements can be determined upfront. Second, it is assumed that the early determination of requirements is cost-efficient, because the cost of change increases in later phases of the development process, and that the cost growth is approximately exponential (Beck, 2000).

In XP both these assumptions are challenged (Beck, 2000). First, XP is based on the assumption that vague or volatile requirements make it inefficient to determine most of the requirements prior to the design phase. Attempts to elicit requirements under these circumstances may even result in distrust and defensive behaviour, because customers may shirk responsibility. Second, one of the most controversial assumptions of XP is the "flattened change cost curve" (Beck, 2000, pp. 23), that directly contradicts the above hypothesis of exponential cost growth.

The strategy employed by XP to deal with uncertainty is to define four guiding values, namely communication, simplicity, feedback, and courage (Beck, 2000, pp. 29). First, if the environment is not predictable, communication is important to ensure that decisions take into account all relevant factors. Second, simplicity helps to avoid the development of unnecessary features that may be rendered useless by later changes in the environment anyway. Third, feedback is vital to minimise the time lag between a change in the environment and the reaction by the software project. Fourth, decisions that turn out to be unfavourable later are an inevitable problem, and courage is required to deal with these issues candidly.

3.2. Design Information

Gharajedaghi (1999) points out that the elements of a social system are "information-bonded". This emphasises the crucial role of information in social systems. One important aspect in a software project is the storage of design information for later use.

The strategy employed by most software development methods is to develop and maintain detailed models of artefacts that are produced during the project using natural language or special description

205

languages. The resulting documentation is developed and maintained in parallel to the source code of the executable computer programs. Often these approaches are characterised as "document-centric". One well-known problem with these approaches is the inherent redundancy of information in the documentation that can give rise to inconsistencies. The underlying assumption of these approaches is that the value of the documentation exceeds the cost of its management.

In XP this assumption is challenged. Instead, it is assumed that the source code together with automated test cases is able to convey almost all of the information needed during the development and subsequent maintenance of computer programs. Moreover, in the presence of vague or volatile requirements the generation and maintenance of documentation consumes resources that should better be spent on the source code or the test cases. This view is supported by Cockburn (1998), who notes that the management of documentation is usually only feasible on the basis of CASE tools, and that the evidence for productivity gains through these tools is rather weak in many cases.

The strategy employed by XP to deal with the storage of design information emphasises the role of source code and automated test cases. This is supported by two practices, permanent and aggressive refactoring and complete and fully automated test coverage of all requirements (Beck, 2000).

3.3. Decision Making

The degree of centralisation of decision making is a major characteristic of any organisation.

Most software development methods rely on specialised roles and a deep division of labour. The software process defined by Jacobson et al. (1998) is a typical example of this approach. The resulting procedures rely on differentiation into separate tasks and corresponding integration of partial results. This approach usually results in rather centralised decision making, where important design decisions are made by specialised and formally appointed team members. The assumption behind this approach is that the gain from specialisation of developers exceeds the cost incurred by differentiation and integration efforts.

In XP this assumption is challenged. Instead, it is assumed that a deep division of labour results in centralised decision making that may lead to communication overhead, misunderstandings, a lack of responsibility, and dissatisfied customers and developers.

The management strategy adopted by XP is decentralised decision making. This is supported by the definition of rather abstract decision criteria that leave much room for individual decisions by team members (Beck, 2000, pp. 71). For example, the practice of simple design is such an abstract decision criterion that leaves the actual decision when and how to carry this out to the discretion of the individual developers.

3.4. Social Learning

As Gharajedaghi (1999, p. 86) notes, "social learning is not the sum of the isolated learning of each member. It is the members' shared learning as manifested in a notion of shared image and culture."

Most software development methods concentrate on technical processes and rather ignore issues of social learning. The assumption behind this is that either these social issues do not arise, or that somehow they are resolved informally, for example by a skilful project leader. This attitude is questionable for two reasons. First, the popular writings by Brooks (1995), DeMarco and Lister (1999), and Weinberg (1998) show that social processes can have a strong impact on software project performance. Second, social research has shown that many teams do not have the ability to resolve social issues effectively (Weinberg, 1998).

In XP the assumption that social issues were of minor importance is challenged. Instead, in XP there is a dedicated role called "coach" that is responsible for the management of social team processes. Thereby XP is built on two assumptions. First, in most software projects social issues will arise at

some point. Second, in many cases the technically oriented team will not succeed in resolving these issues itself, and therefore a special role is needed to ensure the smooth operation of the process.

4. CONCLUSION

XP is a new SDM that claims to be superior to other methods in some situations that are characterised by vague requirements and rapid change. XP has provoked reactions ranging from enthusiastic support to vigorous criticism. In this paper we have investigated possible sources of such fundamental disagreement.

A SDM can be regarded as an abstract description of a social system that is supposed to produce software. This systems perspective on a SDM focuses attention on its underlying, often implicit assumptions about software development. We have presented some examples where the assumptions made by XP are fundamentally different from the assumptions made by most other SDMs. These fundamentally different assumptions indicate fundamentally different mental models. These mental models represent individual, deeply ingrained human characteristics like the attitude toward uncertainty and risk, the preferred way of communication, the desired degree of centralisation in the decision-making process, and the attitude toward social learning. All these issues represent highly individual and usually very stable human characteristics that are acquired during socialisation.

Mental models are central to systems thinking because they have a strong influence on our perception of reality and our behaviour. If they act as unconscious filters that limit our perception to what we expect, they can seriously impede our ability to learn. We conclude that mental models play a crucial role in the introduction of XP. This suggests that the transition from another SDM to XP will often require a corresponding adjustment of mental models. Conversely, if this adjustment of mental models fails, then the successful transition to XP as a SDM is rather unlikely.

REFERENCES

The Agile Alliance (2001). *Manifesto for Agile Software Development.* http://agilemanifesto.org/ (last visited on 15/03/2002).

Beck, K. (2000). *Extreme Programming Explained: Embrace Change.* Longman Higher Education.

Brooks, F. P. (1995). *The Mythical Man-Month.* Addison Wesley Longman.

Cockburn, A. (1998). *Surviving Object-Oriented Projects: A Manager's Guide.* Addison Wesley Longman.

Cockburn, A. (2001). *Agile Software Development.* Addison Wesley Longman.

DeMarco, T. and Lister, T. (1999). *Productive Projects and Teams*. Dorset House Publishing.

Gharajedaghi, J. (1999). *Systems Thinking: Managing Chaos and Complexity.* Butterworth-Heinemann.

Institute of Electrical and Electronics Engineers (2000). *Dynabook on Extreme Programming.* http://computer.org/seweb/dynabook/Index.htm (last visited on 15/03/2002).

Jacobson, I., Booch, G., and Rumbaugh, J. (1998). *The Unified Software Development Process.* Addison Wesley Longman.

Pugh, D. S. and Hickson, D. J. (1997). *Writers on Organizations*. SAGE Publications.

Reel, J. S. (1999). Critical Success Factors in Software Projects. *IEEE Software*, May/June 1999, pp. 18-23.

Schermerhorn, J. R. (2001). *Management*. John Wiley & Sons.

Scott, W. R. (1998). *Organisations: Rational, Natural, and Open Systems.* Prentice-Hall.

Senge, P. N. (1994). *The Fifth Discipline: The Art and Practice of the Learning Organisation.* Doubleday Books.

Weinberg, G. M. (1991). *Quality Software Management: Systems Thinking.* Dorset House Publishing.

Weinberg, G. M. (1998). *The Psychology of Computer Programming.* Dorset House Publishing.

207