

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

Appendix K Application of the analytical framework

This appendix describes five agile methods, DSDM, XP, Scrum, ASD and Crystal Methods, using the comparative analytical framework described in Chapter 4. The source for the method description is the major publication listed for each method in its identifier table. Opinions or statements not made by the method author listed in the identifier are clearly stated as my own opinion or referenced to another source.

Table of Contents for Appendix K

1	Dynamic Systems Development Method.....	2
2	Extreme Programming	13
3	Scrum	25
4	Adaptive Software Development	36
5	Crystal Methods	50
	Bibliography for Appendix K	61

List of Figures

Figure 1: DSDM process diagram from the DSDM Consortium ("Dynamic Systems Development Method Ltd.; the DSDM lifecycle", 1997 - 2005).....	9
Figure 2: XP values, principles, activities and practices (Beck, 2000).....	18
Figure 3: XP practices and their interactions from Beck (2000, p. 70)	20
Figure 4: Empirical Management Model (Schwaber & Beedle, 2002, p. 101)	28
Figure 5: Summary of Scrum phases adapted from Schwaber & Beedle (2002, p. 8) ...	33
Figure 6: The Detailed Adaptive Life Cycle (J. A. Highsmith, 2000, p. 85).....	44
Figure 7: Characterising projects by communication load and criticality	53
Figure 8: Cockburn's model of concurrent development (Cockburn, 2002, p. 132).....	58

List of Tables

Table 1: Identification of the source material for DSDM	2
Table 2: Identification of the source material for XP	13
Table 3: Project risks and how they are addressed in XP. Adapted from Beck (2000)	16
Table 4: Identification of the source material for Scrum	25
Table 5: Project risks and how they are addressed in Scrum	27
Table 6: Identification of the source material for ASD.....	36
Table 7: Identification of the source material for Crystal methods	50
Table 8: Cockburn's Crystal methods (Cockburn, 2002)	59

1 Dynamic Systems Development Method

Identifier

Table 1: Identification of the source material for DSDM

Method Name Alternative(s)	Dynamic Systems Development Method DSDM
Author	Jennifer Stapleton
Date of first publication	1995 ("Dynamic Systems Development Method, Version 2", 1995)
Major publication	Stapleton (1997) This source was selected because it contains "the essential foundations of DSDM that are not expected to change in future versions" (Stapleton, 1997, p. xi)
Country of origin	United Kingdom

Philosophy

Paradigm

This method is primarily objectivist as there is an assumption that a software product will be produced. The subjectivist component of the method is shown in the concern for the people who will work with the system.

Assumptions and values

The stated assumptions are:

- Development is a team effort. It must combine the users' knowledge of the business requirements with the technical skills of IT professionals.
- High quality demands fitness for purpose as well as technical robustness.
- Development can be incremental. Not everything has to be delivered at once, and delivering something earlier is often more valuable than delivering everything later.
- The law of diminishing returns applies- resource must be spent developing the features of most value to the business.
- DSDM is about people not tools. It is about truly understanding the needs of the business and delivering solutions that work – and delivering them as quickly and as cheaply as possible (Stapleton, 1997, p. xiv).

These assumptions are formalised in a set of nine principles:

1. *Active user involvement is imperative.* This is the most important principle. There are 'a few knowledgeable users who support and participate in the development team throughout

the project' (ibid, p. 11). This involvement is continuous and is designed to reduce communication problems between users and developers regarding requirements.

2. *DSDM teams must be empowered to make decisions.* The main constraint to team-level decision making is the budget, but within that constraint frequent small decisions should be made by the team.
3. *The focus is on frequent delivery of products.* This may be weekly and consist of functioning software or other artefacts of development. The delivered artefact does not need to be complete. Frequent delivery ensures that the delivered product meets the needs of the users and helps manage control of the process.
4. *Fitness for business purpose is the essential criterion for acceptance of deliverables.* This is to avoid the problem of 'gold-plated' solutions that attempt to cover all possible user needs.
5. *Iterative and incremental development is necessary to converge on an accurate business solution.* Systems evolve using this method, a subset of functionality is delivered early to the user and additional functionality is developed in further passes of the process. Having a user in the team ensures fast feedback on the work quality and that errors are captured early.
6. *All changes during development are reversible.* It is accepted to discard or rework any project artefact which does not meet its intended purpose.
7. *Requirements are base lined at a high level.* The requirements captured in the business study become the agreed high-level scope of the project. To provide project control these broad requirements are gathered and 'frozen' (left unchanged), prototyping is then used to elaborate the detail of requirements during process iterations.
8. *Testing is integrated throughout the lifecycle.* "Test as you go" (ibid, p. 16) is possible because of the early and constant delivery of software (partial system components). All forms of testing are carried out incrementally throughout the project.
9. *A collaborative and cooperative approach between all stakeholders is essential.* This includes relationships between users, developers, parts of the business, IT organisations (both internal and external), purchasers and suppliers. Compromise is considered important when negotiating new functionality within budgetary and time constraints as some functionality must be left out as new functionality is discovered.

Perspective

The method “addresses the needs of all participants in RAD: project manager, developers, end users, user management and quality assurance personnel.” (ibid, p. xiv). The method is even-handed in its perspective and takes no one viewpoint.

Objectives

DSDM is designed to provide a controlling framework for Rapid Application Development (RAD) tools and techniques. The framework has all of the characteristics of a method, but the description of how to use it is stated at a high level so that users can tailor the method to any technical or business environment.

A primary goal of the method is to shorten development times and to deliver what will have the greatest business benefit first. One stated aim is to provide “a way of developing application systems that truly serve the needs of the business” (ibid, p. xiii).

DSDM also aims to remove the ‘quick and dirty’ image of RAD. This is achieved by using techniques designed to deliver only what is needed, on time and at an agreed level of maintainability. There are three maintainability (quality) levels. The level of each project is decided at the start of development. The levels are:

1. The system must be maintainable from its first delivery into the operational environment.
2. Maintainability is not initially guaranteed, but will be addressed after delivery.
3. The system is a temporary solution and therefore will not be maintainable. The developers reserve the right to remove the system from production once it has served its immediate purpose.

In addition “DSDM is more than anything about improving communications between all parties involved in the development of a system” (Stapleton, 1997, p. 65).

In conclusion DSDM has the objectives of providing a structured, framework for RAD techniques in order to shorten development time and provide the required system at a negotiated level of quality.

Domain

The domain of this method is computationally straightforward business problems with high user interface needs.

Target

“There are classes of system to which the method is most easily applied” (ibid, p. 19).

Size of project – small and large, but large projects must be able to be split into functional components for incremental delivery.

Type of problem – business problems

Type of organisation – not specified

Size of organisation – large or small

Type of development – not specified

Type of application – interface intensive business systems

Technology environment – not specified

The method states specific criteria which should be met before DSDM is considered:

1. Functionality is reasonably visible at the user interface. This makes user verification of the system using prototyping a more useful method for requirements gathering.
2. All classes of end users can be identified. This is because it is considered essential in this method to have a representative of the potential end user population working alongside the development team at all times. The aim is to have complete coverage of all relevant user views within the development team.
3. The application is computationally straightforward (not complex).
4. Large applications can be meaningfully split into smaller functional components.
5. The project is time constrained.
6. The requirements are flexible and specified at a high level.

There is a ‘suitability filter’ published in the DSDM Manual ("Dynamic Systems Development Method, Version 2", 1995) which is a series of questions to consider the use of the framework. This list however does not give a definitive answer to whether the method should be used or not, rather it raises questions, the answers to which provide useful project information for planning purposes. The filter has questions about the business, the system to be developed and technical considerations.

Model

The primary model is readable, well-documented code; no other model is specified in DSDM. The creation of any type of models, structured or object-oriented, is acceptable. A minimal and essential set of analysis, design and test documents are defined by the team at the start of development and they are reviewed as development progresses. It is acceptable to create

additional support models and documents but they should not be reviewed. Recommended documents are a system overview, a context diagram showing the systems interfaces with other systems, a description of system components and how they are linked, the physical data structures and the design decisions that were taken and why.

Techniques

Stapleton states “There are no prescribed techniques, but suggested paths are supplied for implementers of both structured and object-oriented approaches” and in addition “DSDM describes project management, estimating, prototyping, timeboxing, configuration management, testing, quality assurance, roles and responsibilities (of both users and IT staff), team structures, tools environments, risk management, building for maintainability, reuse and vendor/purchaser relationships”. (ibid, p. xiv).

Evolutionary prototyping is a fundamental technique which is controlled using documented evaluation criteria and timeboxing. It is used to overcome the problem of communication between IT people and business people and resolve the requirements to a fine level of detail by clarifying the interface, its look, feel and properties. The prototype is not a traditional “throw away” type, but is a partial system component elaborated in increments until it becomes the final system. Different types of prototype are defined: business prototypes to demonstrate business functionality, usability prototypes for investigating the HCI aspects of the system, performance and capacity prototypes for workload assessment and capability/design prototypes for trying out a particular design approach. The prototype is always accompanied by a review document where user feedback is recorded by a scribe.

Iteration is another technique used in DSDM and a timeboxing technique is used to control the iterations. Incremental delivery is a technique that is accepted as necessary in DSDM but it is not discussed in any detail. It is expected that a full set of system artefacts is produced at each increment including a complete and consistent set of documentation, working software, user manuals, and training materials.

Analysis and design techniques are not specified in the method, only how to manage such activities. Time management is given detailed consideration in DSDM because it is important to complete the project in short time frame. In order to keep to a strict time frame the following practices are recommended:

- “The aim should be to work within the normal working day and keep weekends and evenings free” (ibid, p. 26).

- The scope of the project is clearly agreed and documented in a business study carried out at the start of the project. If new functionality is added than some other functionality is removed to maintain time and budget limits.
- Regular and frequent team meetings are used to ensure development is on track. Daily meetings are recommended which take no longer than ½ hour.
- Development only includes what is absolutely necessary to the business users.

A technique called MoSCoW rules is used for prioritising requirements. All requirements are assigned a priority of; must have, should have, could have, want to have but will not have this time round. This prioritisation is used to decide what will be achieved in each time box. Priority is based on considerations of the importance of the business requirements, technical risk, and the difficulty of the task for the developers.

Each project is divided into a series of timeboxes which control the development of functionality and iterations. Each timebox has a start and end date and consists of a list of prioritised functionality. The timebox is controlled using a three stage process of:

- Investigation – considers the status of previous timeboxes and any work that impacts on this new timebox. What is to be produced in the timebox according to the set priorities is decided, priorities are reviewed to check that they are still correct and quality criteria for deliverables are set.
- Refinement – this is the stage when development is carried out.
- Consolidation – this stage is used to ensure that all deliverables and quality criteria (tests) have been met for this timebox.

Each stage has an objectives-setting meeting at the beginning and another meeting at the end to check that objectives have been met. To control each stage, forms checklists and descriptions of objectives are used. Each timebox contains a mix of priorities to provide some flexibility if things are not completed or problems arise.

Joint Application Development (JAD) is a meeting of stakeholders to discuss and agree on project details such as requirements definition, prioritisation of requirements, prototyping the user interface, and benefits analysis.

Stakeholder training in DSDM principles and practices is recommended. Team and user collocation is another technique recommended in DSDM to maintain effective communication and reduce the need for formal documentation. Testing occurs in each timebox and includes unit, integration, system, acceptance and regression tests. Testing is carried out by the team (user or developer) and is not passed to a third party. Automated testing is recommended.

Metrics are collected to determine project progress and time spent on the various activities. The data can be gathered using a questionnaire and the results graphed and publicly displayed.

There are no proscribed metrics.

Tools

The ideal support environment consists of automated testing tools, configuration management tools and documentation production tools. These tools should be an integrated set, although the author states that this integrated set is not available at the time of writing. The set is listed as:

1. Common user interface (presentation integration).
2. Development tools - including tools for requirements analysis, system prototyping, design, construction, testing, and reverse engineering
3. Requirements management tools
4. Configuration management tools
5. Project/process management tools
6. Documentation tools
7. Shared repository (data integration)
8. Virtual operating environment (platform integration)

Scope

The development process has five phases as shown in Figure 1.

1. Feasibility study

This is a traditional assessment of the feasibility of the project but is very brief (two weeks are recommended) and includes an assessment of the feasibility of using DSDM for this particular project. The feasibility study also includes an outline plan and a 'fast prototype' (ibid, p. 5) to provide evidence of the technical feasibility of the project.

2. Business study

This study provides understanding of the business and technical constraints, the business processes to be automated and the information needs of the new system. JAD sessions are held to gather initial high level requirements for this study and to prioritise the requirements. The content of the study can be changed as the project progresses and as the need arises. The contents are:

Business Area Definition: describes at a high level the processes to be automated, the business users who will be affected by the system, the prioritisation of the business requirements and initial system models.

System Architecture Definition: describes the development and target platforms and the software architecture, its major components and interfaces.

Outline Prototyping Plan: defines the prototyping strategy and the configuration management plan.

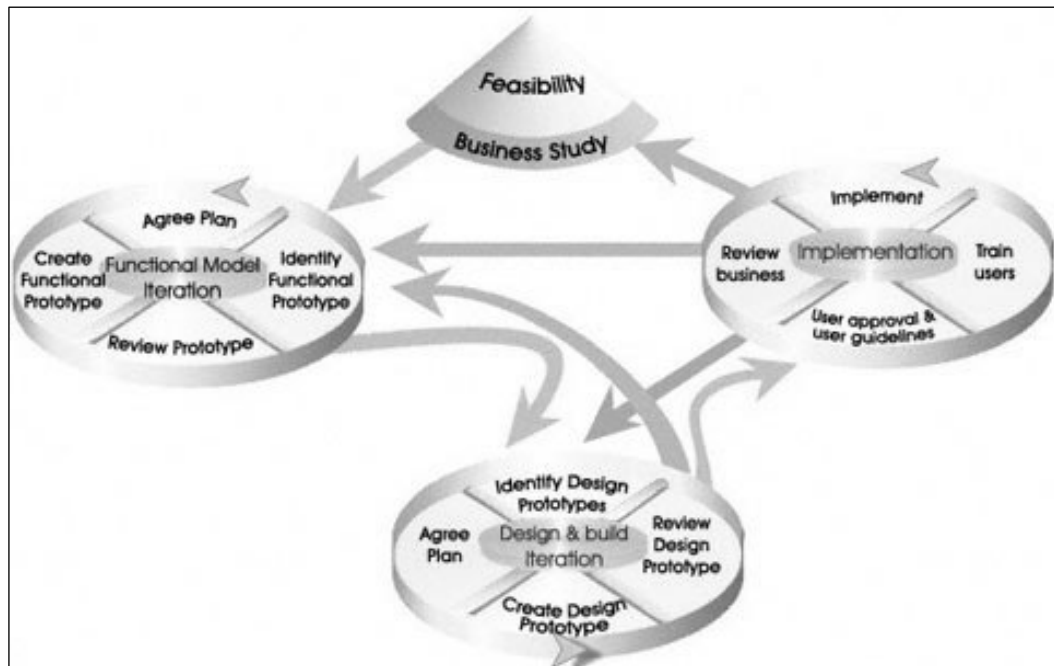


Figure 1: DSDM process diagram from the DSDM Consortium ("Dynamic Systems Development Method Ltd.; the DSDM lifecycle", 1997 - 2005)

3. Functional model iteration

This iteration consists of the creation of analysis models, software components, and prototyping activities. Each iteration has four activities: identification of tasks, agreement of task allocation, carrying out tasks, and a review of the completion of tasks (document review, prototype demonstration and software testing).

Contents:

- Prioritised functions
- Functional prototyping review documents
- Non-functional requirements
- Risk analysis of future development

4. System design and build iteration; this is when the system is refined to standard suitable for implementation. The activities of the functional model iteration are continued.

5. Implementation; this is when the system is installed into the operational environment, users are trained, a User Manual and Project Review Document are produced and recommendations for future system refinement are documented. If further work is needed the project returns to an earlier phase to complete the work.

The feasibility and business study are carried out sequentially at the beginning of project. The latter phases are iterative; their exact progress depends on individual project needs.

Output

The output is a fully functional system which meets user requirements and includes documentation suitable for maintenance purposes, training schemes and user manuals.

The main product is working software of an agreed quality. In addition a series of documents that explain the project, and enough models and documents to enable maintenance to be carried out at a more sophisticated level than studying the code, are produced.

Practice

Background

DSDM is formulated and controlled by a UK- based not-for-profit consortium of organisations of all types who are involved in RAD development of software systems. The consortium was formed in 1994. DSDM is a practitioner-based methodology.

DSDM and standards

TickIT

DSDM can be used in the TickIT environment. TickIT is a UK scheme which provides certification and third party auditing procedures around the ISO 9001 standard and ISO9000-3 notes for guidance for software development. The British Standards Institution produces a specific guide for the application and assignment of DSDM in a TickIT environment.

CMM

DSDM can help an organisation achieve a CMM (Capability Maturity Model) process maturity level of 2 (the repeatable process level) according to the DSDM Consortium.

DSDM certification

A six month course is available for IT staff which is conferred jointly by the Consortium and the countries examining body.

Roles and responsibilities

The development team consists of user representatives, developers and project management staff. Individual roles specified are:

Senior developer; leads the team and is experienced in RAD techniques. Sets timeboxes, carries out analysis, design, coding and testing (component tests, integration tests, regression tests).

Developer; a less experienced senior developer

Technical coordinator; defines system architecture, ensures system quality, maintains technical controls (such as configuration management). Normally a senior technical expert.

Ambassador user; participates in prioritisation of functionality, provides communication between the user group and the developer team and carries out acceptance testing. This representative comes from the community that will use the system.

Advisor user; anyone who has an interest in the final system. They have an *ad hoc* role in providing business requirements for the project and supplement the knowledge provided by the Ambassador user in specialist areas.

Visionary; this person is responsible for defining and championing the new system. They clearly see the business need for the system and participate in the feasibility and business study phases.

Executive sponsor; the ultimate controller of the finances of the project and the final decision maker.

Project manager; “the autocratic project manager has no place heading up a DSDM project” (Stapleton, p. 38) . This person prepares the feasibility study and the business study for the project with input from the user representatives and other development team members. They maintain logistic support for the project. The project manager works full time on one project.

The recommended team size is 2-6 people. A project may have up to six teams working in parallel. Dedicated technical staff rather than part-time staff are recommended.

Difficulties with DSDM

The authors state that DSDM is unsuitable for scientific or engineering applications as it is untried in these environments. Difficulties are likely to occur when the target environment (see Target section above) is not present.

Skill levels

Each team should have the technical and business skills necessary to carry out the development with additional specialists be called in as required. Every member of a DSDM team must be able to work in a cooperative and collaborative way with all other members of the team and with end users. There is no clear delineation between the various roles within a team and all members should be capable of carrying out the main development tasks (user communication, analysis, design, code and test).

Tailorability

“For DSDM to be successful, all of these [nine] principles must be applied in a project (ibid, p. 11). If one of them is ignored, the whole basis of DSDM is endangered. Some projects may find that one or more of the principles is difficult to apply in which case the user of DSDM should be seriously reconsidered. At the very least, an approach to mitigating the consequences of non-conformance to the principles needs to be thought out” (ibid, p. 19). Although DSDM is not tailorable at a high level tailoring is expected when it is applied to individual projects as: “It [DSDM] is a method in as much as it defines a process and a set of products, but these have been deliberately kept at a high level so that they can be tailored for any technical and business environment” (ibid, p. xiv). One of the case studies in Stapleton (1997) states: “be prepared to adapt everything [in the method] to suit the needs of the team and to play to your strengths” (Stapleton, 1997, p. 89). I conclude from this that tailoring the method for a project is considered acceptable as long as the overall principles are adhered to.

2 Extreme Programming

Identifier

Table 2: Identification of the source material for XP

Method Name Alternative(s)	Extreme Programming eXtreme programming, XP
Author	Kent Beck
Date of first publication	1999 (Beck, 1999)
Major publication	Extreme programming explained: embrace change (Beck, 2000)
Country of origin	USA

Philosophy

Paradigm

“XP takes commonsense practices and principles to extreme levels” (Beck, 2000, p. xv); which explains the name of the method.

XP is primarily objectivist because it is ‘scientific’ and disciplined (Beck, 2000, p. xvii) and is “concerned with engineering a system to achieve its objectives” (Checkland, 1999, p. A48). It is scientific because explicit practices are mandated and metrics are recommended. It is disciplined in the sense that the people using the method have clearly defined roles and responsibilities and in order for the methodology to work they must carry out these roles and responsibilities consistently and correctly. The objectivist philosophy is also apparent in the planning game technique which assumes a shared understanding will arise from a meeting of developers and the systems business stakeholders. This positivist stance indicates an objectivist approach (D.E. Avison & Fitzgerald, 1995). XP is subjectivist in that emergent properties (“The whole is greater than the sum of the parts” (D. E. Avison & Fitzgerald, 2003, p. 557)), in the sense of Checkland (1999, Chap 3) will occur when the principles and practices of the method are used together, supporting each other and leading to a balance. As Beck says “The practices and the principles work together with each other to create a synergy that is greater than the sum of the parts.” (Beck, 2000, p. 150). Exactly what these synergistic benefits are is left unexplored in the method description.

Assumptions and values

The method is built on an assumption about the ‘cost of change’ and four values. The values are supported by fifteen principles which are implemented with twelve practices and four activities.

In XP the belief is that the traditional ‘cost of change’ relationship is no longer valid in most system development projects. Traditionally the cost of a change in a system rises exponentially over time as the system moves through the phases of analysis, design, implementation and testing. Beck believes that this assumption is no longer valid due to changes in technology such as the use of object-oriented systems development, integrated development environments and object databases which have reduced the impact of system changes on cost and time. If the ‘cost of change’ assumption is no longer true then making changes late in the development process is acceptable. This assumption explains why many of the practices of XP are possible. Once it is acceptable to make a change to the system at any time during development then practices such as simplicity in design (you can always make a change later if it proves necessary), acceptance of late changes to systems due to changing requirements, constant code refactoring, and constant retesting of the code base also become acceptable.

The values of XP are communication, simplicity, feedback and courage. Communication between the customer and the development team and communication between team members is critical to project viability as it reduces misunderstandings and problems within the project. Simplicity is implemented by creating the simplest coded solution to a requirement and not designing the system for future flexibility or extension. The purpose of this is to reduce development time. Feedback is important because it is used to assess project progress and to ensure the system under creation is correct. Courage is needed by the development team to make changes to code structure as and when needed, to throw away code that is not working, to try different code designs and to communicate project problems openly with management whenever they occur.

Further assumptions are that the important variables that act on projects are cost, time, quality and scope. Beck assumes that any three of these variables can be controlled in a project but not all of them at once. The customer of an XP project is asked to set the value of any three of the variables. Then the development team adjusts the additional variable to enable the project to be carried out. Another assumption of XP is that “Everything in software changes. The requirements change. The design changes. The business changes. The technology changes.

The team changes” (Beck, 2000, p. 28) . Responding to and controlling the impact of these changes on the development environment is a goal of XP.

Perspective

“The programmer is the heart of XP” (Beck, 2000, p. 141).

There is a programmer-centric view of development in XP. It underlies some of the reasons for the practices used particularly those concerned with team morale and communication and 40 hour weeks.

XP aims to reduce project risk, improve responsiveness to business changes, improve team productivity throughout the life of the system, to make working in teams to create software ‘fun’, to have better relationships with customers and to have “stable, more productive programmers”. (Beck, 2000, p. xvii)

Objectives

This statement summarises the objectives of the method. When introducing the method Beck analyses the risk associated with software development projects and explains how XP is designed to address these risks with appropriate practices. Table 3 shows these risks and associated practices of XP.

The prime objectives of the method are; to develop the system as rapidly as possible; to produce software which meets the customer’s needs and has some acceptable level of quality while mitigating risks; to produce quality software in small efficient teams using established software development techniques; to have high morale and a good working environment for developers. Software quality is important. It is negotiated with the customer and supported with specific techniques. High morale and an appropriate working environment are objectives not considered in other methodologies. Even the participative ETHICS methodology of Mumford (1995) makes no mention of developers although employees who will use a system are fully catered for.

I believe the objective of the method is narrow, it does not include any strategic analysis of the need for the system, general problem solving, the impact of the system on the business, its effect on business productivity, or the work lives of users, all of which are addressed historically by a variety of methodologies. The objective is to meet the customers stated business needs by producing a software solution and no further problem analysis is undertaken.

Domain

XP is a specific problem-solving methodology because it is designed to enable a team of developers and customers to produce solutions for specific business problems. There is an underlying assumption that there is a specific problem to be addressed which is clearly specified before the project begins.

Table 3: Project risks and how they are addressed in XP. Adapted from Beck (2000)

Software Project Risk	How XP practice addresses the risk
Schedule slippage	Short release cycles of a few months at most 1-4 week iterations within a release consisting of customer-requested features 1-3 day tasks within an iteration Feature prioritisation carried out by customer
Project cancellation	Releases are short and provide evidence of software's usefulness
Maintenance problems	Unit test suites developed for each code segment Integration tests carried out daily
High initial defect rates	Unit tests for each code segment Integration tests daily Customers write and carry out function tests
Business misunderstood	Customer on-site Late change requests acceptable
Business changes	Short release cycle produces software early for feedback
False feature rich	Function prioritisation reduces unnecessary features
Staff turnover	Programmers estimate and complete their own work Team contact and interaction encouraged Collective ownership of code spreads knowledge of the system around the team

Target

The target environment for XP is described as follows:

Type of problem to be addressed by the project –

- Projects that are not constrained by an existing computing environment
- Projects with vague requirements
- Projects with constant changes in requirements

Type of organisation for which the software is developed – any type

Type of organisation in which the software is developed – any type

Size of organisation – any size

Size of project - projects that can be carried out by two to ten programmers.

Type of development

- Outsourced software
- Fix-price contract software development
- In-house development

Type of application XP is designed for - application frameworks for external use, software applications, web-based systems, shrink-wrapped software

Technology platform - Systems developed using object-oriented concepts and programming languages.

Model

No model is specified in XP. There is an assumption that XP is used in an environment with object-based systems development, and class models, CRC cards (Beck & Cunningham, 1989) and sketches are mentioned. These models are intended to be used when required as an aid to communication and understanding, otherwise any appropriate models can be used but no guidance is provided on what models should be used. In addition there is no formal design phase or documentation requirement specified in XP. Models are created and thrown away once the code and its unit test are completed.

The nearest thing to a model is the ‘system metaphor’ which is used to describe the system architecture, but its development and use is not explained in any detail.

Simple design and refactoring¹ are used in place of formal modelling. Beck specifies what he means by ‘simple design’; this is when the system meets a specified set of constraints (in priority order) (Beck, 2000, p. 109):

1. The system must communicate everything you want it to communicate.
2. The system must contain no duplicate code.
3. The system should have the fewest possible classes.
4. The system should have the fewest possible methods.

This set of guidelines is inadequate for most practical purposes as criteria 1, 3, and 4 are subjective. The refactored code and its associated unit test scripts are the only artefacts of the system which are kept.

Techniques

The techniques of XP are related to the philosophical values of courage, feedback, simplicity and communication. Beck calls the techniques ‘practices’ so that name will be used here.

There are fifteen principles which support the values and four activities and twelve practices

¹ Refactoring is the process of reworking existing code to simplify it and improve its design.

which are used to achieve these principles. Values, principles, practices, activities and the relationship between them are shown in Figure 2.

Four activities are central to XP; coding, testing, listening, and designing. Coding is used to produce the end product but it is also used as a mechanism for communication between team members as it explains system and program logic when pair programming and it is used as a learning mechanism for trying out different designs and structures for the system. The source code is valuable because, along with its unit tests, it forms an ‘operational specification’ (Beck, 2000, p. 45) of the system.

The twelve practices of XP are:

Coding standards – programmers write code in accordance with rules which emphasise communication through code readability.

Collective ownership – “anyone can change any code anywhere in the system at any time” (ibid, p. 99). This technique is used so that knowledge about the system is shared amongst the team.

Continuous integration – the system is integrated and tested whenever a task is completed which may be many times per day. This provides rapid feedback on system development progress.

40 hour week – must be the norm. Never work overtime a second week in a row. This is to reduce worker stress and low morale.

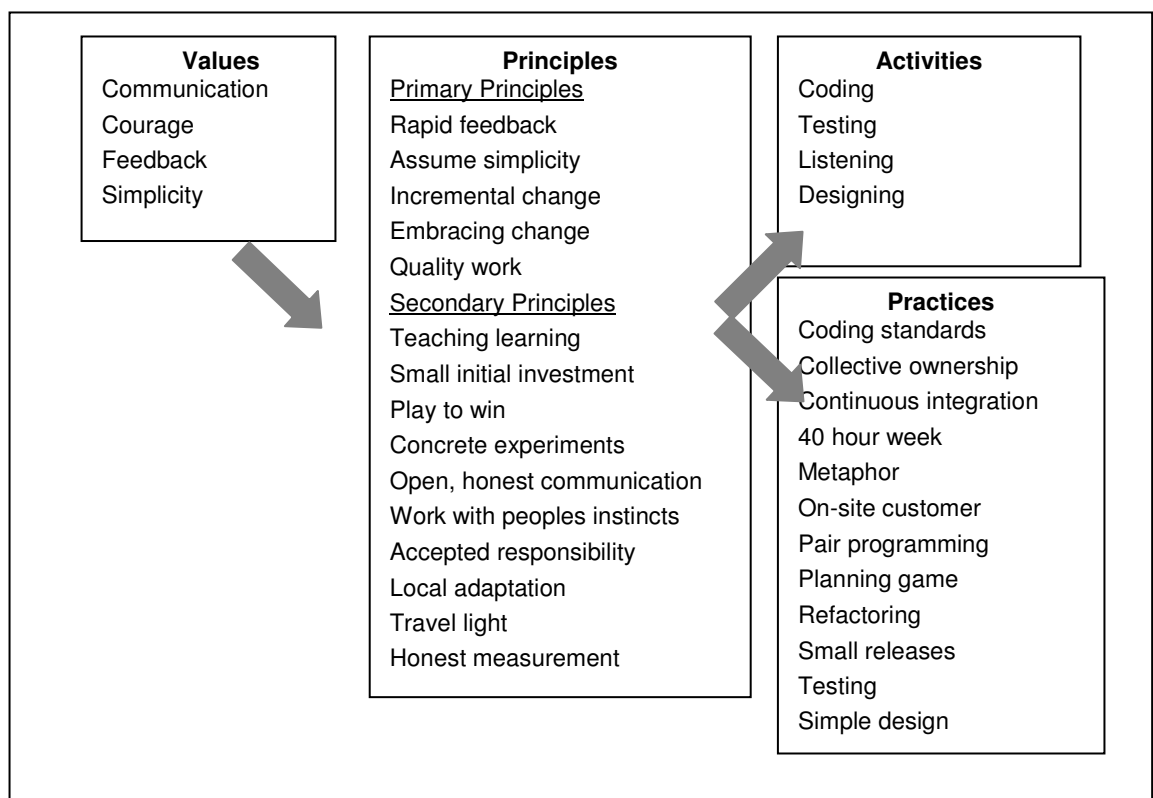


Figure 2: XP values, principles, activities and practices (Beck, 2000)

Metaphor – a single shared story of how the whole system works that describes the system architecture and guides development. The system metaphor is “A story that everyone – customers, programmers, and managers, can tell about how the system works.” (ibid, p. 179). There are no examples or further discussion about this practice making it weakest of the XP practices.

On-site customer – a customer is part of the team so that they are available at all times to answer questions. This reduces the chance of misinterpreted requirements and provides rapid response when requirements are unclear or problems occur.

Pair programming – all production code is written using two programmers sitting at one machine. The benefits of this practice are that it improves code quality, reduces any detrimental effects of personal ownership of code and supports a shared knowledge base throughout the team.

Planning game – planning is carried out at the beginning of each release and each iteration within a release. The scope of the next release is determined quickly by combining business priorities and technical estimates.

Refactoring – programmers restructure the system without changing its behaviour to remove duplication, improve communication (code readability), simplify, or improve flexibility.

Small releases – a simple system is put into production quickly and new versions are released on the shortest cycle possible. The aim is to find a balance between implementing the most valuable business requirements and the highest risk requirements first.

Testing – programmers continually write unit tests before writing the code and these tests must run correctly before development can continue. Integration testing of units is carried out at least daily. Customers write function tests so that features are demonstrated as complete.

Beck explains that: “The practices support each other. The weakness of one is covered by the strengths of others.” (ibid, p. 63). How each of the practices support and interact with other practices is explained in detail in the method (see Figure 3). There are two further practices that are not included in the list above; metrics and room arrangements. Metrics are managed by the tracker role, one of the major tasks for this role is to implement and manage a metrics programme. Metrics are discussed as a mechanism for process control and feedback but only one metric is described: “the ratio between estimated development time and calendar time” (ibid, p. 72).

Any other metrics can be used and are posted publicly. Facilities Strategy or room layout is also specified in the method “We will create an open workspace for our team, with small private

spaces around the periphery and a common programming area in the middle” (ibid, p. 77). The reason for this is to enable pair programming and improve informal communication during development.

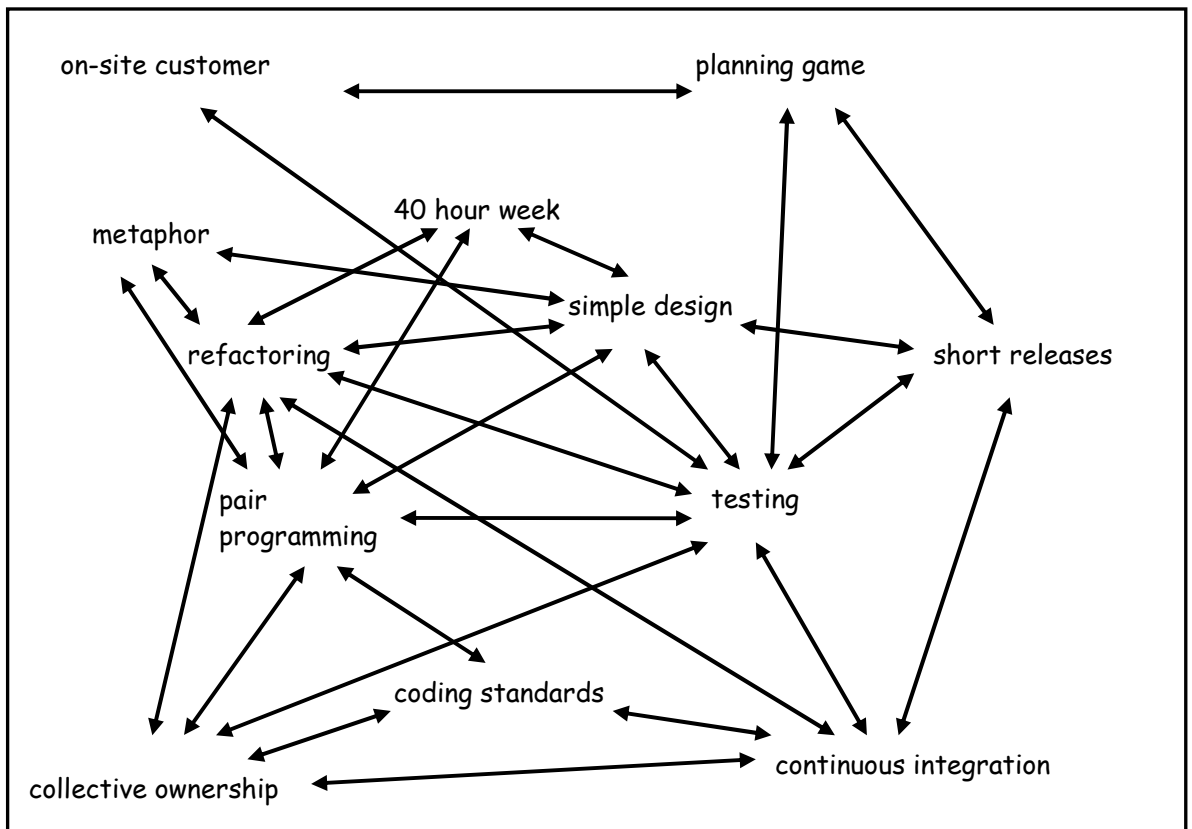


Figure 3: XP practices and their interactions from Beck (2000, p. 70)

Tools

Automated unit testing tools must be available for unit testing and integration testing as without such tools the whole method is cumbersome and impractical. This makes the availability of such tools a fundamental requirement for successful XP adoption.

Scope

Detailed explanations of the phases needed to carry out the planning, scoping, and prioritising of functionality into releases and iterations is provided. Phases are: (Beck, 2000, chap 21)

Exploration; this includes team formation, testing the development technology, exploring possibilities for system architecture by prototyping, estimating task implementation times, and practicing the technique of story writing.

Planning; consists of customers and developers agreeing on a date for the end of the first release which should ideally be between 2 and 6 months long. Requirements are written by the customer on story cards. These main stories are assigned to releases. Estimates of the length of time for each story to be implemented are made by the developers to provide an overall estimate of the length of the project. The customer decides which of the stories are most important and these are implemented first. Where technical factors affect the priority then negotiation occurs between the developer and the customer to decide on a reasonable time frame or priority for the story. Customer involvement is crucial in the planning phase as negotiation is both mandated and clearly described in XP.

Iterations to first release; this breaks the release into one-to-four week iterations. The initial iteration “must result in a system that runs end-to-end, however embryonically” (ibid, p. 91). Customers prioritise and pick the story to be implemented in each iteration. At the beginning of each iteration the developers divide a story into tasks on task cards, a developer selects a task and estimates the time needed to implement the task. A task typically takes 1-2 days. Customers write functional tests for each story which are run at the end of the iteration.

Productionizing occurs at the end of the last iteration and is when releases of software occur. Each release begins with an exploration phase and risk analysis is carried out to determine which changes should be included in the release. Daily stand-up meetings are recommended during releases to maintain communication amongst team members. Iterations continue and software is evolved more slowly during this phase. Performance tuning begins along with refactoring of the code base.

Maintenance; this is when the system is in production and is considered ‘the normal state of an XP project’ (Beck, 2000, p. 135). New functionality is produced during this phase and refactoring continues to improve the code structure. Developers may take a ‘help desk’ role in addition to programming. Iterations and releases continue.

Death occurs when new customer stories have stopped or for other business reasons. At this point a brief document is produced that describes the overall system. The purpose of this is as an aid to future maintenance or when creating similar systems.

In summary XP specifies a set of phases and activities such as analysis (story writing), design (using CRC cards and refactoring), testing and writing code that are carried out during each phase.

Output

The only outputs specified in XP are source code and associated unit tests. Any other outputs are negotiated with the customer. The main XP product is working software. Another outcome of using the practices is a development team with good morale. The team will also be familiar with the method which could be considered another product. Any other product is negotiable with the customer.

Practice

Background

XP is based on the development experience of a single author and is aimed at developers; therefore it is a practitioner-based method. The practices used in XP are all traditional software engineering practices except pair programming, test first development and refactoring.

Roles and responsibilities

Programmer; estimates the time needed to implement features, estimates the consequences of technical alternatives, sets the detailed scheduling of tasks within a release, writes unit tests and code, works as a pair programmer and communicates understanding to other programmers, performs functional tests and integration tests.

Tester; helps customer choose and write functional tests and performs any additional types of test.

Tracker; gathers and maintains information and metrics about the project progress. Checks that schedule estimates for stories were correct and that scheduled iterations and releases are likely to be met. This person also keeps a record of reported defects, defects assigned to programmers and test cases used.

Coach; understands, implements and maintains the method and process. Implements a process for reviewing XP practices and is normally a senior technical person.

Consultant; an expert called in to solve a particular problem.

Big Boss; hires and fires employees and takes overall control of the project.

Customer; sets the timing and scope of releases (defines features/functionality to be implemented in a release), sets the relative priorities and scope of proposed features. An expert user is recommended for this role.

Difficulties with XP

Aspects of XP that are difficult to adopt are collaboration with others in the team, and using simple designs. These involve overcoming people's tendency to avoid collaboration and introduce unnecessary complexity in system structure (Beck, 2000).

Beck bases his guidelines on when not to use XP on his own experiences. The main contraindications are: "big teams, distrustful customers and technology that doesn't support 'graceful change'" (Beck, 2000, p. 155) . Other detrimental environmental conditions include: a business culture that is very hierarchical and uses command and control mechanisms for making decisions, a culture where an explicit analysis, design or system specification must be produced before development begins, a culture where complete documentation of a system is required, a workplace where it is normal to spend long hours at work, or where communication between programmers is actively discouraged and a project that needs more than 20 programmers.

Many of the practices of XP cannot be achieved easily with large teams, without customer and management support, or with systems built without objects. Practices such as continuous integration, planning game, and customer on-site become unmanageable as the amount of code and the number of participants in the development increases. Large existing systems which are complex and highly coupled are also not amenable to the practices of XP. Cultures with large slow quality assurance processes that lengthen feedback cycles are also not recommended. Meeting the physical environment specifications is also considered crucial to implementing XP effectively and XP is not recommended for geographically separated teams.

Skill levels

The skill levels needed are those of average programmers. Coaches are necessary team members and experience with the method along with previous system development experience is needed for this role. The skill level required of the customer who joins the team is also high. They must be capable of carrying out a number of activities in development and decision making which require either training or experience to carry out competently.

Tailorability

To gain the maximum benefits from XP all practices must be adopted completely.

Contradicting this is the principle of 'local adaptation' which says that the method should be adapted for local conditions. Adaptation guidelines are brief and general; initially pick the most difficult problem; use only the planning game and test-first development practices; as each practice is learned then adopt more practices. The only other advice is to adapt the practices to the situation as needed, for example if the team consists of only three or four programmers then some of the coordination practices can be omitted (e.g. the iteration planning game).

Tailoring is an accepted practice in XP, but details on how to adapt the method for various situations is not given.

3 Scrum

Identifier

Table 4: Identification of the source material for Scrum

Method Name Alternative(s)	Scrum none
Author	Schwaber, K. , Beedle, M.
Date of first publication	Schwaber (1995)
Major publication	Agile software development with Scrum (Schwaber & Beedle, 2002)
Country of origin	USA

Philosophy

Paradigm

Scrum is objectivist in that it assumes that an automated solution to the problem is needed. Scrum consists of constant objective assessment of progress towards the goal of a completed software project. It is subjectivist because it is concerned with the people who will create the system and the emergent properties of the method. What these emergent properties are is not explored.

Assumptions and values

“Scrum is based on an empirical process control model rather than the traditional defined process control model”
(Schwaber & Beedle, 2002, p. 89).

In Scrum the assumption is that “software development is like new product development and not like manufacturing”(Schwaber & Beedle, 2002, p. 106). Consequently software development should not follow a repeatable and defined manufacturing process; the software development process should involve creativity, research and learning and be managed using empirical methods. Scrum is based on process control theory and ideas formulated in a study of the development practices of six large Japanese and American companies who produced innovative products during the late 1970s and early 1980s. This study by Takeuchi and Nonaka (1986) determined six characteristics of new product development processes:

1. “Built-in instability
2. Self-organizing project teams
3. Overlapping development processes

4. “Multilearning”
5. Subtle control
6. Organizational transfer of learning” (Takeuchi & Nonaka, 1986, p. 138)

Takeuchi and Nonaka (1986) likened traditional sequential development methods to a relay race and their new method to a rugby game (rugby is popular in Japan). A scrum is a technique used in rugby to gain control of the ball.

Schwaber uses these ideas to explain the reasons for the Scrum techniques. Traditional project management and ‘heavy weight’ systems development methodologies try to impose an abstract repeatable process model on software development; this type of process cannot be effective for software development when each project is unique. Scrum offers a solution to managing and controlling software projects using an empirical process control model. This involves regular inspection of development activities to observe the state and progress of development, and then adjustment of the activities to produce the desired and predicted outcomes. Figure 4 shows this empirical management model. The model illustrates how a self-organising team uses the technology, requirements and multi-learning as input to the development process, called a Sprint, which is controlled using observation and adjustment of progress. The output is an increment of the total software product.

Scrum has stated values which emerge when Scrum is used; commitment, focus, openness, respect and courage. The techniques of Scrum support these values:

Commitment – team members commit to the project because they are given autonomy to solve a problem in whichever way they choose.

Focus – the team must be able to focus to solve a problem and Scrum gives them the environment where this is possible by removing distractions.

Openness – there is communication and visibility for all tasks, problems, responsibilities, and lines of authority.

Respect – all team members, who all have different skills, backgrounds, education levels and abilities, respect one another

Courage – the team members have the courage to make decisions and to work out how to meet the cost, schedule, quality and functionality commitments of development.

Scrum has a team-oriented philosophy shown by statements such as; “Scrum deals primarily at the level of the team” (Schwaber & Beedle, 2002, p. 2); “Scrum is about deep social interactions that build trust among team members”. (ibid, p. 106). The published method discusses the effect of using Scrum on the development team which showed signs of ‘emergent properties’ of

hyper-productivity, and developing better solutions; “The personal lives of the people were changed. People said they would never forget working on such a project and they would always be looking for another experience like it. It induced open, team-oriented, fun-loving behaviour in unexpected persons and eliminated those who were not productive from the team through peer embarrassment” (ibid, p. 15).

Perspective

This method has a project manager and programmer team perspective of development.

Objectives

Scrum provides technological solutions in environments with certain risk factors. These risks and associated Scrum practices are shown in Table 5.

Table 5: Project risks and how they are addressed in Scrum

Software Project Risk	How Scrum practice addresses the risk
Not pleasing the customer	Customer can constantly see the product Customer on-site preferred. Customer sees working software at least every Sprint Sprint planning meeting used to prioritise and allocate tasks according to customer choice Sprint review meeting used to assess visible progress
Functionality incomplete	Functionality prioritised for each Sprint. Only low priority functionality is missed.
Poor estimating and planning	Techniques for evaluating progress and prioritising tasks: <ul style="list-style-type: none"> • Daily Scrums, • Product Backlog • Monthly Sprints • Sprint planning meeting • Sprint review meeting
Not resolving issues	The management role is facilitated by attendance at daily meetings
Not completing the development cycle	Working software is delivered at each Sprint
Taking too much work and changing expectations	Changes to Product Backlog not allowed during a Sprint

Scrum objectives are;

- To provide an effective alternative to traditional methodologies and processes.
- To provide techniques for estimating, planning, tracking and managing software development projects.

- To form “a self-empowered team where everyone had a global view of the product begin built” (ibid, p. 11).
- To enable teams, guided by knowledge and experience rather than a formal project plan, to cooperate effectively to produce complex, sophisticated products.
- To provide exponential productivity gains.
- To produce working functionality within one month and in consistent increments thereafter.
- To produce software that meets business needs.
- “to wrestle working systems from the complexity of emerging requirements and unstable technology” (Schwaber & Beedle, 2002, p. 154).
- To act as a wrapper to other methods. Scrum has been used with XP on projects. Scrum “is superimposed on and encapsulates whatever engineering practices already exist” (ibid, p. 18).

Another Scrum goal is to control the four variables of development; time, cost, quality and functionality. The first three are normally fixed for a project and the fourth is varied to meet the goal set for an each iteration. This is done by increasing or decreasing the scope or depth of the functionality delivered.

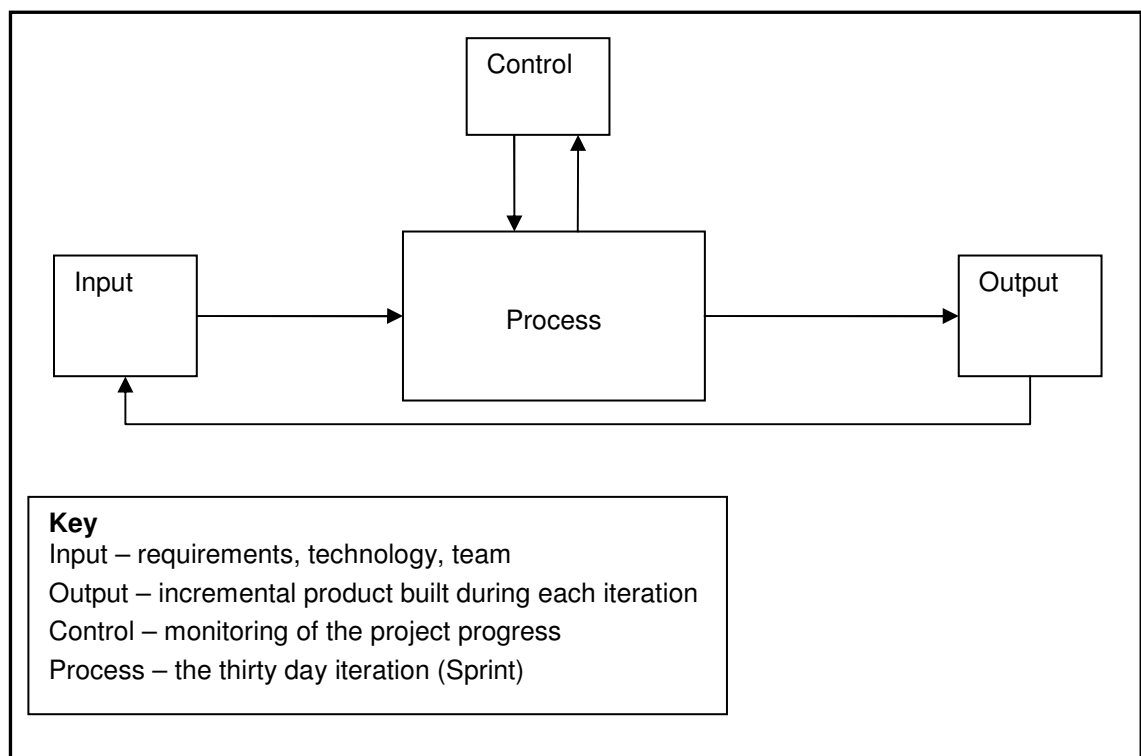


Figure 4: Empirical Management Model (Schwaber & Beedle, 2002, p. 101)

Domain

Scrum is designed to address existing well-defined problems.

“Scrum works in any environment and can scale into programming in the large”
(Schwaber & Beedle, 2002, p. 17).

Target

Type of problem to be addressed by the project –

- Projects that are complex
- Projects with vague requirements
- Projects with constant changes in requirements

Type of organisation for which the software is developed – any type

Type of organisation in which the software is developed – any type

Size of organisation – any size

Size of project - projects that can be carried out by five to nine developers.

Type of development – any type

Type of application Scrum is designed for – any type but with particular use in

- Application frameworks designed for reuse
- Software applications for web deployed wireless technologies
- Web-based systems
- Object-oriented systems

Technology platform – any

Scrum is a general method suitable for any type of development.

Model

There is no ‘model’ specified in this method. “Architecture and design emerge across multiple Sprints, rather than being developed during the first Sprints” (Schwaber & Beedle, 2002, p. 9).

It is recommended that the first Sprint includes the development of an initial system framework to show that the chosen solution is possible given the technology and team available. Object-oriented development and design techniques are assumed to be the paradigm in which most development takes place. No guidance is provided in their use and it is assumed that the team will have this knowledge in place before development begins. This is not explicitly stated in the method but a number of case studies in the main Scrum publication discuss the technology used

and it is typically object-oriented. In addition it is stated that the foundation of Scrum is sashimi (incremental development) and scrum as described by Takeuchi and Nonaka (1986) which are techniques “that uniquely fit object-oriented implementation of software” (ibid, p. 11). There is some criticism of the ‘use-case driven’ approach as it is seen to obscure a lot of the tasks that must be completed to create the system solution.

The case studies described in the method publication offer an insight into the importance of the model in Scrum development: “Stop worrying about documentation. Document the system after you go into production” (ibid, p. 134).

Models are used when needed and primarily as a guide to thinking about the solution.

Techniques

“Teams develop products incrementally and empirically”
(Schwaber & Beedle, 2002, p. 2).

Scrum uses an empirical process control model which “provides and exercises control through frequent inspection and adaptation for processes that are imperfectly defined and generate unpredictable and unrepeatable outputs.” (ibid, p. 25). The techniques are designed to enable management to carry out control by observation and incremental adjustment while the team carries out the development unhindered. Scrum has a number of techniques for managing the project:

Product Backlog is a publicly available list of everything the system should include and address, including functionality, features and technology, enhancements, bug fixes, and issues. The list is gathered from all stakeholders and prioritised by the Product Owner. The Product Owner estimates the amount of tasks in a release and in a Sprint, and in consultation with the team, estimates the time in days to implement each Product Backlog item. Because functionality is prioritised in this way, any functionality not completed in an increment is likely to be low priority. The Product Backlog grows as items are added and shrinks as Sprints are completed.

Sprint is a thirty day period (iteration) in which a subset of the Product Backlog is implemented to create an increment of product functionality. At the end of a Sprint another subset of the Product Backlog is selected for implementation and a new Sprint begins. Working software is delivered to the customer at every Sprint.

Sprint Goal is the ultimate goal of a sprint which is set during the Sprint Planning meeting.

Sprint Backlog is a subset of the Product Backlog which is selected at the beginning of a Sprint for implementation during that Sprint. The Sprint Backlog is maintained by the team. Each task is estimated by its developer and estimates can be adjusted as the development progresses.

Sprint Planning meeting occurs at the start of each Sprint. Customers, users, management, the Product Owner and the Scrum Team determine the Sprint goal and functionality at this meeting. The Product Owner selects tasks to complete in the Sprint from the Product backlog. The team then allocates the individual tasks that must be performed to build the product increment to meet the Sprint goal.

Daily Scrum is a short 15 minute meeting held daily where individual progress is reviewed and impediments to progress are reported to the Scrum Master who deals with them. Anyone (managers and users) may attend the meetings but only the team and the Scrum Master can speak. Management, in the form of the Scrum Master ensures active management of the project. This meeting is to give management first-hand observation of teams and project progress.

Sprint Review meeting is a four hour informational meeting held at the end of each Sprint. Management, the team, customers, users, and the Product Owner inspect the product increment, which may be either kept for further development, scavenged for parts to be reused, or thrown away. This means that if the team is unable to develop the required functionality, only one month is lost from the whole development process. The Product Backlog may be reprioritised at this stage depending on any additional requirements arising during the Sprint. Releases of software occur when the product is deemed ready and may occur during a Sprint.

Release Backlog is a subset of product Backlog that is selected for a release of finalised software product.

Product Backlog Graph is a graph used by management to track project progress and assist in decision making; it shows the estimated days of work remaining for a release. The Product Owner updates these values weekly. The graphs dependent axis is 'estimated work remaining'; the independent axis is the project or release time scale.

Sprint Backlog graph is the same as the Product Backlog Graph but for a single Sprint.

Sprint signatures are backlog graphs which show patterns unique to a team.

Tasks are subsets of functionality worked on by team members and are usually four to sixteen hours long.

Other recommended practices:

- Customer on-site: “Wherever possible , Scrum prefers to have a customer on-site but it mandates that the customer sees working software at least every Sprint.”(Schwaber & Beedle, 2002, p. 109). The purpose of this is to validate the progress of the project.
- A large open work space where communication is face-to-face and development progress is posted on public notice boards.
- Daily builds and tests.
- Configuration management
- Regression testing
- System testing
- Release management

The only metrics described for Scrum are those used to monitor progress; the Product Backlog and the Sprint Backlog.

This analysis shows that Scrum prescribes various techniques for controlling the project and the team and provides little further guidance on other aspects of development, such as analysis and design techniques, leaving such details to the team to decide. It provides mainly project management practices for iterative and incremental development in environments where object-oriented technologies are in use.

Tools

No tools are specified for the method but adequate tools and infrastructure are recommended (Schwaber & Beedle, 2002).

Scope

Scrum uses iterative, incremental systems development. The phases are clearly stated and are shown schematically in Figure 5Figure 5: Summary of Scrum phases adapted from Schwaber & Beedle (2002, p. 8)

.

Output

The output of the method is working software which meets the requirements of the customer. The customer receives working software at the end of each 30 day period. Any other products can be requested by the customer.

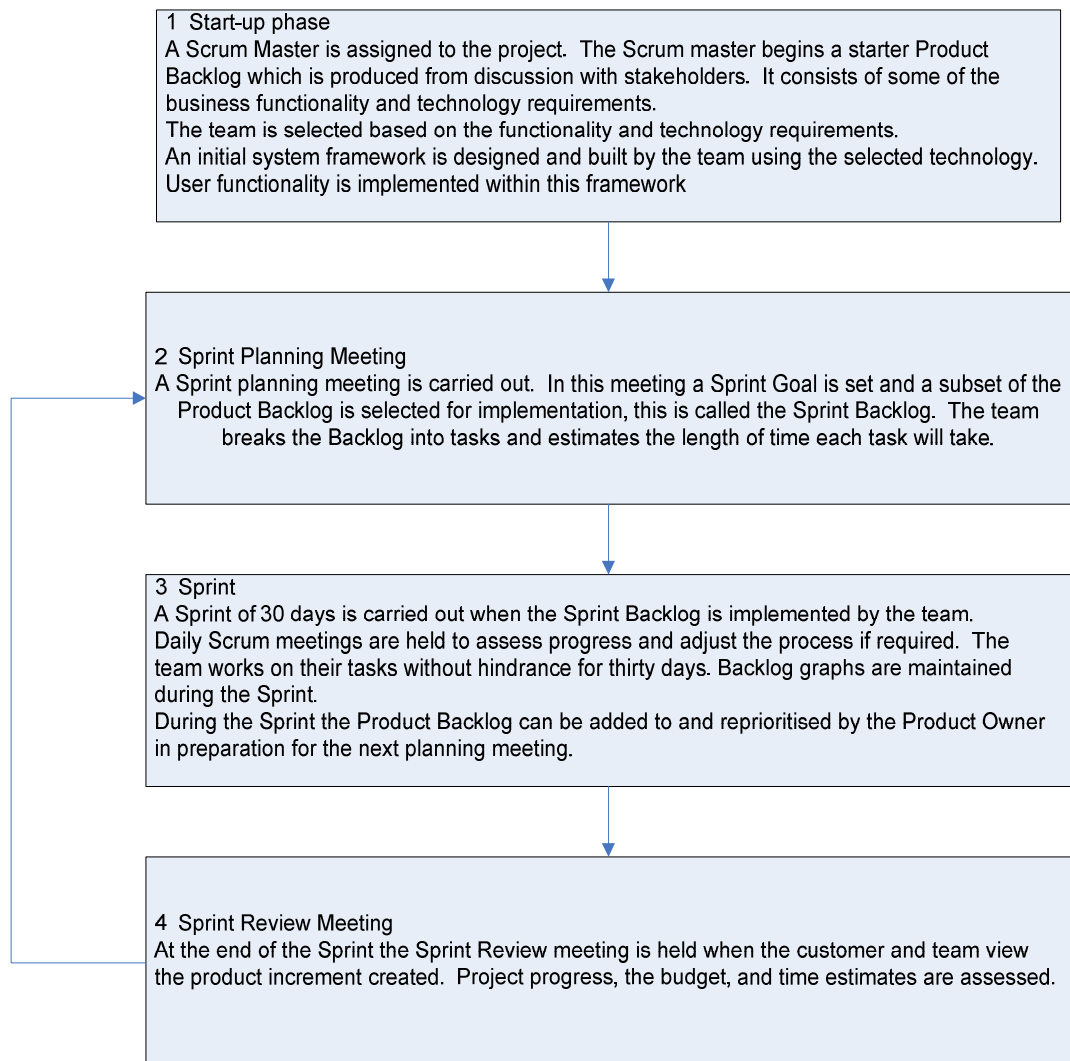


Figure 5: Summary of Scrum phases adapted from Schwaber & Beedle (2002, p. 8)

Practice

Background

Scrum is practitioner-based because it is based on the development experiences of the authors who report on a number of examples where they have used the method to develop systems.

There is some grounding in management and process theory as shown in the philosophical discussion above.

Roles and responsibilities

Scrum Master; a management representative (project leader or project manager) who enforces Scrum values, practices and rules, helps the team make decisions and solves problems arising for the team. This person acquires needed resources and acts as an interface between the team and the rest of the organisation. This person forms Scrum teams in association with management; works with the Product Owner and the team to create Product Backlog for a Sprint; calls and attends daily meetings to determine progress; removes impediments to progress; works with management to gauge progress and reduce backlog; coordinates and conducts the Sprint Review meeting.

Product Owner is the person who is officially responsible for the project and decides the order in which the system functionality will be implemented. Only they can prioritise the Product Backlog which they must keep visible to the whole team. This person works with the team and others (e.g. quality assurance people, technical writers) to estimate the time backlog items will take to implement.

Scrum Teams are self-organising and fully autonomous, their focus exclusively on the currently selected Product Backlog. The team selects the amount of backlog that it believes it can handle in a Sprint based on estimates for each item. This becomes the Sprint Backlog which the team commits to turn into a working product. The team may consist of programmers, analysts, testers, consultants with specialised expertise and any other specialists required. Other points about teams:

- Team size, 7+-2 less than 3 too small for benefits, larger than 8 too large for control mechanisms to work properly.
- The team is cross-functional, all members are responsible for analysis, design, coding, testing and user documentation and all members work on all tasks. Team composition may change at the end of a Sprint.
- The team makes all decisions about how the Sprint goal will be achieved within the constraints of any charters, standards, conventions, architectures and technology specified.
- Only the team can change the contents or the estimates of a Sprint Backlog during a Sprint. They must keep the Sprint Backlog up to date.

- The team must attend Daily Scrum meetings; teams may be located in different locations but still report to the daily meeting using telecommunication media.

Management; their role is to manage the four variables of cost, date, functionality and quality as the development proceeds.

Difficulties with Scrum

No difficulties are discussed. Scrum can be used in all situations, and all difficulties expect management cancellation of the project are assumed to be surmountable.

Skill levels

The skill levels are not stated.

Tailorability

The method is designed to be used as stated but can be tailored if needed. It is recommended that until experience is gained, the full set of techniques is used before tailoring is carried out. How to tailor the method for different environmental conditions is not described. “Scrum scales to any size” (ibid, p. 16) and general guidance is provided on how to adjust the method for larger projects, reuse of components, and how to use it for geographically distributed projects. For larger projects the advice is to structure the development into multiple teams who then develop product increments in parallel, all teams working from the same Product Backlog. Each team has there own Scrum Master and the Scrum Masters meet frequently to communicate progress and problems. Scrum can be used in conjunction with XP or any other existing engineering practices without tailoring.

4 Adaptive Software Development

Identifier

Table 6: Identification of the source material for ASD

Method Name Alternative(s)	Adaptive Software Development ASD
Author	James Highsmith
Date of first publication	Highsmith (1997)
Major publication	Highsmith (2000)
Country of origin	USA

Philosophy

Paradigm

ASD combines the philosophy and practices of Rapid Application Development with the ideas of complex adaptive systems theory to provide a framework for developing software systems. The framework is most effective for project environments of constant change and high time pressure. I believe that RAD methods reflect an objectivist approach as the goal of the practices is always to provide software systems for business problems. Adaptive systems theory, as it relates to human activity systems, provides the rationale for the techniques of ASD. I believe this shows a subjectivist approach because this theory is based on the holistic thinking of complex adaptive systems.

Assumptions and values

Complex adaptive systems theory is usually associated with the actions of living entities and their relationships (e.g. cells). Successful software development projects carried out in accelerated and uncertain environments (extreme projects) tend to act as complex adaptive systems and show the properties of emergent order. This is “a property of complex adaptive systems that creates some greater property of the whole (system behaviour) from the interaction of the parts (agent behaviour). This emergent system behaviour cannot be fully explained from the measured behaviours of the agents”² (J. A. Highsmith, 2000, p. 8). The property of emergence is also identified as a characteristic of systems in the Soft Systems Methodology of Checkland (1999). Software projects can be characterised as systems, similar to a living

² Highsmith notes “there is no scientific *proof* that emergence is a characteristic of organizational systems. However there is growing evidence that emergence is a characteristic that helps explain observed organisational behaviours. It is usable and actionable, and it helps organizations achieve their stated missions, the behaviours are not accidental – they follow an understandable pattern.” (J. A. Highsmith, 2000, p. 283)

organisms, with agents as the people taking part in the project. Approaching software development in this way “provides a better model for managing extreme software projects. Such an approach produces better products more quickly, and at the same time fosters healthier organisms ready to tackle the next project” (J. A. Highsmith, 2000, p. 11). Projects that behave as complex adaptive systems show self-organisation and a high degree of collaboration leading to emergent order and the ability to adapt quickly to change. The practices and techniques of ASD are designed to support self-organising teams and collaboration; within the team, between teams and between the team management and customers. The techniques support a software development process that is adaptable when changes occur.

The method is based on three models:

Adaptive Conceptual Model – defines the properties of complex adaptive systems. The system is viewed as an ensemble of independent agents with these properties:

- Agents interact to form an ecosystem;
- their interaction is defined by the exchange of information;
- their actions are based on internal rules;
- they self-organise to product emergent results;
- they exhibit characteristics of both order and chaos;
- and they evolve over time.

Adaptive Development Model – a life-cycle model designed to accommodate uncertainty and change. The model is based on the RAD spiral, iterative and evolutionary lifecycle and includes the concept of emergent order and component-based development. Using this model projects are not controlled directly but guided to completion. There are three phases, speculate, collaborate and learn. The speculate phase involves defining the mission and sharing it among the stakeholders, developing a detailed adaptive life cycle plan and producing versions of the product in an iterative manner. The collaborate phase involves active stakeholder participation characterised by unfettered information flow and good leadership. The learn phase comes from the ideas of the learning organisation (Senge, 1990) and organisational adaptation (Holland, 1995). Highsmith believes that learning organisations and teams are those which are capable of changing with the times because the organisation has the ability to critically examine itself and improve itself based on that knowledge. In a software development project this is characterised by critical feedback from stakeholders to the team about the product, and feedback on the progress of component development from the team to management. Feedback allows any mistakes to be found early and corrected in short iterative cycles.

Adaptive (Leadership-Collaboration) Management Model – an adaptive organisation treats continuous change as the norm and reflects this in their practices. Adaptation is nurtured by leadership and collaboration rather than command-and-control. Leadership focuses on creating the cultural environment in which adaptation and collaboration are supported, and on creating a collaborative structure in which multiple teams and virtual teams can interact effectively.

Important concepts in the method are having the appropriate organisational culture, leadership style and team characteristics to support collaboration. For successful projects and teams the organisational culture should be human-centred and view people, participation and relationships as vital to success. An organisation should be viewed, not as a mechanistic deterministic machine, but as a complex adaptive system. To support this system management should encourage creativity, innovation and adaptability, and maintain an unstructured environment. This ‘chaordic’ environment is created when the organisation and its projects are balanced between chaos and lack of control, and order and control.

If the business environment is changing rapidly the organisation must also change, along with the development teams. So managers and developers need to understand the organisational goals and the business environment so they can understand the strategies the organisation uses in different environments.

Management should “embrace change” (J. A. Highsmith, 2000, p. 183) and absorb it rather than control it. This strategy leads to opportunities to learn and also to get ahead of the competition. Changes needing management are requirements changes caused by new customer requests, changes in technology needs, and competitors actions. Accepting change as a normal state in a high change environment is an essential part of creating an adaptive culture.

Highsmith provides six characteristics of an adaptive culture:

1. Emergent order – which arises from self organisation.
2. Simple principles – a few simple clear guidelines are best rather than many rules and regulations.
3. Rich connections – a high degree of collaboration between individuals and teams throughout the organisation.
4. Distributed governance – distributed decision-making and team involvement.
5. Poise – building teams while honouring diversity, compromising by way of mutual concessions and decision making by finding common ground.
6. Balance – maintaining tradeoffs between product characteristics and practices.

Highsmith believes that the most critical activities in an adaptive culture are creativity, innovation and problem-solving.

Leaders in adaptive organisations must be highly talented in people management. They should understand the goals of the organisation, facilitate collaboration by providing a team network structure, encourage ideas, support relationships and encourage learning from mistakes. They should also be pragmatic, optimistic, and visionary, acknowledge risk and make hard decisions. In addition they must have the ability to balance rigor and flexibility, guide the team rather than control it, deal with the emotions of the team, have technical understanding, judgement, and be able to continuously adapt.

Teams must participate in team decision making and be accountable for outcomes, they should respect and trust the leaders and have confidence in their own technical skills, and they must build strong relationships based on collaboration. Teams should be able to react to change without the guidance of explicit documentation or specific change-control procedures. Core values are mutual trust, respect, participation and commitment.

Collaboration is another major theme in ASD. Cross-group communication and valuing the contribution and participation of others enhances collaboration. ASD provides guidance on “the interpersonal, cultural, and structural issues of collaboration” (J. A. Highsmith, 2000, p. 115) including how to enable small groups to interact effectively, how to manage complex environments and how to create an environment which supports emergence. A barrier to collaboration is the ‘command and control’ style of management where communication is vertical (up and down a hierarchy) and where rules and predictability are important and individuals are viewed as interchangeable pieces. Another barrier is the culture of individualism present throughout western organisations which rewards the individual rather than the group.

Perspective

“Adaptive software development is a management approach to delivering software product; it is not a specific development approach” (J. A. Highsmith, 2000, p. 70). I believe the perspective of the method is primarily that of the project manager.

Objectives

ASD is a framework for managing software development projects which are under high time pressure and have rapidly changing requirements. The ideas and practices in the method are designed to maintain a collaborative team environment and successfully manage projects using

RAD techniques. The primary objective is to deliver the product to the client within designated scope, schedule, resource and defect levels. Highsmith states the benefits of ASD:

- “Applications evolve in response to periodic feedback, resulting in a close match to customer requirements.
- Changing business needs are accommodated more easily.
- The development process adapts to the specified quality profile of the product.
- Customer benefits are generated earlier, for example, because the customer gets the application more quickly and can use it to increase revenue.
- The risk that major failures will occur is reduced.
- Customers gain early confidence in the project”. (J. A. Highsmith, 2000, p.40)

Another objective is to produce software systems of ‘good enough’ quality. This means that the quality of the product is negotiated at the beginning of the project. The quality of scope (including functions and performance attributes), schedule, defects and resources are all negotiable.

Minimal documentation is another aim of the method although certain documents are specified in a lot of detail as discussed below. Another stated goal is to create change-tolerant, maintainable and extensible software.

The framework aims to produce software while balancing learning, knowledge, process and people, concepts and practice, rigor and flexibility.

Domain

This method is designed for specific pre-defined problems for which a computerised system is the expected solution.

Target

ASD is recommended when the project is “a critical new business initiative (J. A. Highsmith, 2000, p. xxix), under intense time pressure and undergoing constant changes in requirements.

Type of organisation – not specified.

Size of organisation – not specified.

Size of project – small to medium-sized projects. The basic Adaptive Life Cycle is suitable for projects ranging up to 10,000 function points that can be carried out by teams of less than 10 developers. The Advanced Adaptive Life Cycle is for projects larger than this.

Type of development – component development or component assembly, using any object technology.

Type of application – e-commerce and e-business, data warehouse, and products for the Internet software market.

Technology platform – recommended for, but not restricted to; client/server, networked, Internet application server.

Model

There is no model specified by this method.

Techniques

“The practices and tools needed for effective collaboration are in fact those needed for managing continuous change!” (J. A. Highsmith, 2000, p. 185)

ASD techniques are designed to maintain effective collaboration and learning and to enable the management of iterative development.

Components – iterative development is based on the assignment of product features to ‘components’. A component is a set or group of product features, for example; object-oriented ‘objects’, business features, the GUI, or containers that are planned and implemented together.

There are three types of component:

1. *Primary components* deliver functionality to the customer e.g. produce warehouse stock report.
2. *Technology components* are components on which the primary components are built such as networks, computer hardware, operating systems, database management software.
3. *Support components* are items that support the developed product such as training manuals, data models, and data conversion programs.

High risk components, such as components implementing new technologies, are placed in early iterations to reduce the risk to the project.

Function point counting is used to size the project. Other metrics-based techniques are also recommended but the exact techniques are not specified.

Time-boxing. Timeboxes are set for the whole project and for iterations. Timeboxes are boundaries for development effort and the activities within the timebox are negotiated so that development stays within its timebox. For projects less than 9 months a cycle timebox should be 4-8 weeks and for projects longer than 9 months the timebox should be 6-10 weeks.

Timebox length is based on application sizing and estimation techniques. The overall schedule

is divided into development cycles, each with a time milestone. Trade-offs in features and resources are then made so that the time milestone can be met. The early cycles are kept to timeline in order to check the plan; later cycles can then be adjusted once experience is gained. Early cycles are shorter than later cycles to encourage customer involvement and confidence in the development, and to verify scope, requirements and project viability.

Concurrent development within iterations is a control mechanism for managing change.

Components are developed concurrently with the proviso that dependencies between components may reduce or disallow concurrency.

Learning - To support learning a number of techniques are used to increase feedback to the development team. Prototypes, customer focus-groups, software inspections and post-mortems are all carried out at regular short intervals to enhance learning. The belief is that frequency and repetition enhance learning.

Prototyping - and prototype sessions are when developers and customers meet to review or develop applications. Prototyping sessions are less formal than customer focus groups. They are a standard RAD technique used to precisely specify software requirements and to reduce misinterpretation of the customer's requirements. Highsmith believes that models of the system are not adequate and that the application itself is the only deliverable on which customers can base an evaluation. Prototyping begins in the early planning phase and continues during development. Prototypes are used to determine scope, size and cost estimates.

Software inspections are used whenever appropriate during the development cycle. Software reviews, software inspections and walkthroughs are all acceptable techniques. Any work products can be inspected, for example strategic plans or test-case scenarios. Inspections are used to locate defects, train team members in best practices, and support non-testable quality goals such as maintainability. Checklists of what to look for are created. Inspection is carried out before the inspection meeting which is used to review the identified potential defects. Facilitators are appointed to run the inspections. The results of the inspection meeting are a defect list, suggested updates to checklists and inspection metrics (such as hours spent per defect discovered).

Customer involvement and active partnerships between developers and customers are facilitated by Joint Application Development (JAD) and customer focus groups. Developers are advised to get close to the customer in order to understand their business language and needs.

Joint Application Development sessions are used to support collaboration. JAD in ASD is defined as "a facilitated workshop that brings together cross-functional groups to build collaborative relationships capable of producing high-quality deliverables during the life of a project" (J. A. Highsmith, 2000, p. 135). JAD workshops are recommended throughout the development process not just at the start of development.

Customer focus groups are formal review meetings held at the end of a major development cycle when a cross-section of customer representatives explore the working application in a facilitated meeting. The session focuses on demonstrating specific business scenarios. The developer team is present in order to learn about any product changes and to get to know the customers. The customer representatives are present to increase confidence in the product and to gain a sense ownership of the product. The results of the meeting are documented change requests that are later analysed, accepted or rejected, and then assigned to developers.

Post-mortems are facilitated meetings carried out by the core project team and managers during the review phase and again at the end of the project. The effectiveness of the team, the development process, progress-to-date and the development practices are reviewed and any successes, problems and areas for improvement are discussed. This allows for correction during the next development cycle. The post mortem document is distributed about the organisation so other teams can learn from it.

Resource requirements analysis is an analysis of resource needs. This is based on a list of questions that must be addressed by the team.

Risk assessment is addressed at the project planning stage and in the review phase. This assists in cycle planning and is used to modify plans based on management of identified risks.

Teams are colocated and dedicated to a single project. A team is made up of core members and supporting specialists. Frequent short team meetings are held in a dedicated team meeting space to support collaboration, effective communication and concurrent development. Concurrent development needs effective communication so that team members understand the progress on related components. Core members also work together on mission documents to encourage acceptance and understanding of the mission.

Other techniques and practices are recommended but not described in any detail. They include any RAD techniques as well as change management, beta testing, data modelling, training programmes, quality assurance plans, production of user documentation, development of test plans, test cases and test data and source code control.

Tools

Tools are recommended for supporting collaboration but no development tools are specified. The recommended tools include electronic mail, group calendaring and scheduling, asynchronous and synchronous data conferencing, electronic meeting systems, source-document creation tools, configuration management tools and threaded discussion group tools.

Scope

ASD development is carried out using cycles of speculate, collaborate and learn phases as shown in Figure 6. The timeboxed phases are designed to be tolerant of change. The phases used are:

Project Initiation - The project initiation phase is when the project is planned and the initial mission documents are written.

Adaptive Cycle Planning - Components are assigned to iterations during the adaptive cycle planning phase based on their importance, size and dependencies. The steps in the adaptive planning cycle are:

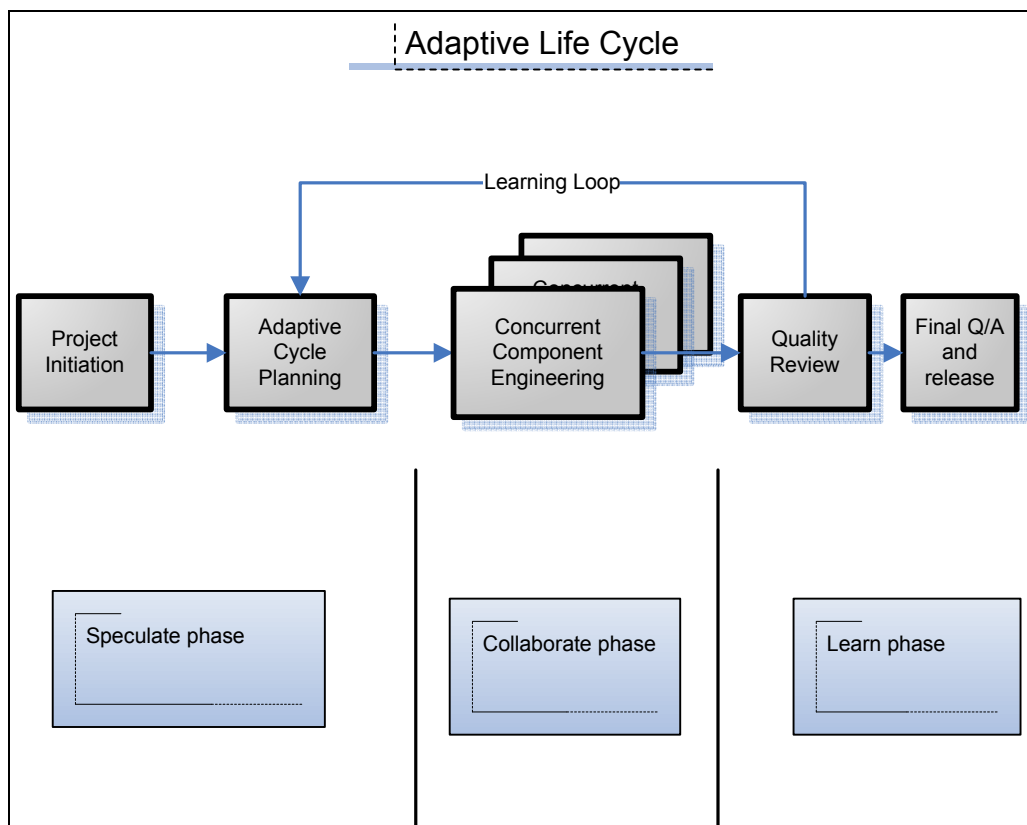


Figure 6: The Detailed Adaptive Life Cycle (J. A. Highsmith, 2000, p. 85)

1. Conduct the project initiation phase to determine the basic architecture and development technologies. Produce the mission statement and the feasibility study.
2. Determine the project time-box. Two types of dates are possible. Target dates are determined by business needs and set the overall project delivery time, committed dates are set by the development team after project initiation and planning sessions. Cycle dates only change if both the team and executive sponsor approve. A 10 to 20% time buffer is recommended to allow for unanticipated events.

3. Determine the optimal number of cycles and the time-box for each.
4. Write an objective statement for each cycle to provide the team with a goal for the iteration.
5. Assign the primary components to cycles.
6. Assign technology and support components to cycles.
7. Develop a project task list.

Concurrent Component Engineering -The goal of each concurrent component engineering phase is to work on components concurrently. Timeboxes are maintained by making trade-offs between time and delivered functionality in consultation with the stakeholders. The learning loop iterations are used to keep the product development public and consistently under review and to keep the project on track. There are three types of iteration through the lifecycle; versions, cycles and builds and each iteration consists of analysis, design, coding testing and conversion planning. A version iteration is a long iteration that produces a new version of the product which is ready to be installed. A cycle iteration is a major loop that delivers a demonstrable component to the review process. A cycle is used to both monitor the project and learn about the product. A build is a short iteration used to produce an interim deliverable which is usually only reviewed by the development team. The first cycle is a proof of concept.

Quality Review - The review is for reflection, status determination and learning. The team considers the project status, its schedule, scope, defect-level and resources. They determine if project artefacts are valid, check that the delivered component meets customer and technical specifications and expectations and assess how the team is working. The cycle plan is also reviewed and corrected if necessary, any new requirements are assigned to components and cycles, and the completion status of the components is assessed. Each component moves through the lifecycle and enters a series of states which are used to manage the project progress:

1. Outline state – the components initial state, with some code written
2. Detail state – the component may have a prototype or model which performs the basic functions required.
3. Reviewed state – the component has been reviewed and changes have been implemented.
4. Approved state - the component is complete

The number of states used depends on the size and complexity of the project. Larger projects need all states; smaller projects can reduce the number of states.

The project management lifecycle - Project management is a separate set of activities to the software development lifecycle. ASD contains a series of questions that the project manager should ask at each step to monitor the status of the project. The steps are:

1. Initial the project
2. Plan the project
3. Manage the project
4. Close the project

Steps are performed iteratively except step 3 which is carried out continuously.

Output

The main output of the method is a software product which is delivered in an evolutionary manner. Although it is recommended to minimize formal documentation to reduce workload, a number of documents are produced during development to support shared understanding about the project. These documents define the project mission and are written by the project manager and the development team. They include the following:

Project vision – this defines what the project is about in 2 to 10 pages. The vision document contains the key business objectives, product specifications and market positioning information in the form of a feasibility study. Political, economic, technical and organisational factors are described and problems, constraints and opportunities identified. A detailed list of contents for the project vision document is provided.

Project data sheet – this document is a one page summary of key business benefits, product specifications and project management information including important scope, schedule and resource information. The data sheet is prominently displayed on a wall in the project team area.

Progress data – qualitative and quantitative measures taken of scope, schedule, defects and resources.

Product specification outline – this specifies the scope, features and functionality of the project including the architecture, project size estimates measured using traditional methods such as function points, lines of code, indications of work effort, milestones and resource estimates. The outline is reviewed at the beginning of each cycle and is used as a baseline for size estimation and to set minimal component specifications and to determine how product features (components) are assigned to development cycles.

Product mission profile – a matrix stating the priority levels for scope (features), schedule, defects and resources. Priority levels are based on market success and cover three levels: excel, improve and accept. This matrix specifies the focus of the development and is written by the

developers in consultation with the sponsors. The profile forms a contract between the development group and the executive sponsor or primary customer.

Other products are negotiated with the client and include written documentation, training, support and consultancy.

Practice

Background

ASD is a practitioner-based method although specific projects are not described.

Roles and responsibilities

Executive sponsor – articulates the objects of the project, approves resources and establishes project constraints.

Project manager – must be full time on a single project. Must be highly skilled and show the leadership qualities described above. The person in this role focuses on defining the mission, building relationships and removing obstacles to progress, rather than on prescribing tasks.

Core team – the team of developers who work on the project full-time. There is one team for each major feature set on a larger project.

Collaboration facilitator – this person organises and acts as facilitator at review meetings, JAD sessions, and on-line meetings. They also manage information flow, for example by setting up access rights for on-line communication, arranging face-to-face meetings and moderating discussion forums. This role is very important for large projects with virtual teams.

Client team members – customer representatives who are available to the team on a daily basis. Highsmith admits that this is the optimal situation which may not always be possible.

Difficulties

No difficulties or problems are described with the method. However this method would not be compatible with the CMM. Highsmith is convinced that the optimisation objective of CMM is not suitable for software development improvement. Highsmith (2000, p. 287) believes that “It is a widely accepted management axiom that strict, detailed procedures and bureaucratic rules impede innovation and creativity” and that “Optimization stifles emergence, not only because individuals feel restricted but also because optimization reduces the breadth and scope of interconnections and relationships.”

Skill levels

There is a certain level of assumed knowledge on the part of the team. Highsmith states: “in Cycle 1, the team needs to establish development guidelines such as naming standards, reuse approach, and data design considerations. Hopefully the team is experienced with the tools and those guidelines are already in place and only need to be tailored to fit the project” (J. A. Highsmith, 2000, p. 107). Highsmith (2000, p. 115) believes that: “having individuals with high software-engineering skill levels is a critical success factor for adaptive groups” along with “teams whose members collaborate well, exchange ideas, share leadership, embrace diversity, and learn from mistakes”. Teams should be picked for intelligence and must have “the right blend of skills – technical skills, business skills, problem-solving and decision-making skills, and interpersonal skills.” (J. A. Highsmith, 2000, p. 118). The aim is to create a ‘jelled’ team who have a common vision of the project, can discuss issues and make decisions, and whose members support team decisions even when they do not fully agree with them. Team members hold themselves mutually accountable for the project outcomes. The team takes an open-ended flexible approach to technical problem-solving. Open teams with these characteristics are also able to take on larger projects.

Tailorability

Most ASD techniques are advised not proscribed. Tailoring is undertaken at project startup when the team tailors the method to the project. The roles and responsibilities of the team members are defined, communication and collaboration pathways are set up, development practices and support tools are identified and plans for change management and progress assessment are written. To set up communication pathways stakeholders are identified, along with what each one needs to know and how that information will be delivered to them.

The techniques for tailoring the method for larger projects are ‘the advanced adaptive lifecycle’ and ‘structural collaboration’.

Scaling up using the advanced adaptive lifecycle involves adjustments to team organisation and to component development. The project is divided amongst a number of different teams, called a network of teams, who work concurrently on different components. Core teams consist of 5-10 members who work full-time on a project supported by part-time experts. Interim prototypes are developed and delivered at milestones which act as synchronisation points when results from different teams are brought together, reviewed and integrated into a testable product. A phase and gate approach is used to manage the increased number of components. This approach focuses on identifying and planning components, determining the dependencies and

interrelationships between components, monitoring the evolution of each component through defined completion states, and evaluating progress on component development at the end of each cycle. Techniques for managing large numbers of components include increasing the number of component states allowing closer monitoring of progress to completion, increasing the number of components monitored and increasing the formality of component documentation. To manage component dependencies during concurrent development increasingly formal communication between groups is needed. Components that cross organisational boundaries (i.e. must be worked on by more than one non-located team) are monitored more closely using a rigorous daily build process.

Structural collaboration is the second technique for managing larger projects. This is a knowledge management technique for sharing expertise and best practices across virtual teams. ASD has techniques for maintaining structural collaboration. As projects get larger and more complex rigor is increased while taking care not to stifle emergence. Rigor is first increased on final integrated product components, components that are shared across geographical groups and documents shared with geographically distributed groups. Then rigor is increased for components and documents that are used within located teams.

5 Crystal Methods

Identifier

Table 7: Identification of the source material for Crystal methods

Method Name Alternative(s)	Crystal methods
Author	Alistair Cockburn
Date of first publication	2002
Major publication	Cockburn 2002
Country of origin	USA/Europe

Philosophy

Paradigm

The paradigm is primarily objectivist because the goal is to create software solutions to given problems. There is a subjectivist aspect in that the software development team and its interactions are as important as any other aspect of software development.

Cockburn's ideas are based on the writings of Naur (1992), Musashi (2000), Weinberg (1998) and Ehn (1992).

Assumptions and values

The Cooperative Game Principle

Software development is a (resource-limited) cooperative game of invention and communication. The primary goal of the game is to deliver useful, working software. The secondary goal, the residue of the game, is to set up for the next game. The next game may be to alter or replace the system or to create a neighbouring system. (Cockburn, 2002, p. 31)

Crystal methods are based on the idea that effective software development is only possible with good communication and a methodology which is fit for the project. Techniques are described for supporting communication amongst the team, and for tailoring a methodology in a timely manner so that it is sufficient for the project and capable of evolving when needs change. The underlying assumption is that when a development methodology is focused on team member skills and communication the project will be effective and agile (capable of adjusting to change) whereas a focus on process is not so effective.

Cockburn analyses the essential elements of a systems development methodology. Then he uses this base to form the Crystal methods. The elements are:

Activities – actions that must be carried out during the project.

Deliverables – work products that are needed within the organisation or between teams.

Milestones – points in time where the progress of product development is assessed as either complete or incomplete. There are three kinds of milestone: reviews, publication and declarations (statements of completion).

Process – the sequence of activities that move the development through to completion, each with a pre and a post condition.

Quality – the desired degree of defects in the software product and other work products. This also involves metrics used to measure the quality of work products and the quality of activities of development.

Roles – the set of activities which a team member carries out. The personality traits of the people should match their role.

Skills – the abilities of team members based on experience, training and natural ability.

Standards – includes code standards but also any tool, technique, or project management decision that is applied across the whole project.

Team values – a set of values that the team embodies.

Teams – the team of people working on producing the software product.

Techniques – specific procedures that people use to accomplish tasks.

Tools – software that supports the development process

Work products– disposable and permanent artefacts of software development.

Each methodology has a particular scope which is a combination of lifecycle coverage, role coverage and activity coverage. In addition when designing a methodology the following parameters are important:

Methodology size – this is the number of control elements in the methodology. Control elements include deliverables, standards, activities, quality measures and techniques.

Ceremony – the degree of formality and completeness of the artefacts of production.

Methodology weight– a subjective measure of methodology size multiplied by ceremony.

Problem size – the number of elements in the problem and their interdependencies, also a subjective measure.

Project size – the number of staff involved in the project.

System criticality – this is the damage from undetected defects. The categories of criticality are loss of comfort, loss of discretionary money, loss of irreplaceable money and loss of life.

Precision – there are three categories low, medium and high precision and all activities and work products are completed to some degree of precision.

Accuracy – the degree of accuracy required in various work products e.g. rough sketches to completed object models with all details added.

Relevance – the area in which the methodology is relevant e.g. interface design, system architecture, user participation.

Tolerance – is the degree of variation allowed in the use of standards and in the dates for activities.

Visibility – is the degree to which an outsider can readily assess if the methodology is being followed.

Scale – the degree to which detail is hidden in models, techniques or project reporting to give a high level view.

Stability – the degree to which the project and the methodology are changing. Projects and their artefacts generally become more stable over time and also just before a design review or publication of a software product or other artefact.

Cockburn's methodology ideas are based on seven principles:

1. "Interactive, face-to-face communication is the cheapest and fastest channel for exchanging information.
2. Excess methodology weight is costly.
3. Larger teams need heavier methodologies
4. Greater ceremony is appropriate for projects with greater criticality.
5. Increasing feedback and communication reduces the need for intermediate deliverables.
6. Discipline, skills, and understanding counter process, formality, and documentation.
7. Efficiency is expendable in non-bottleneck activities" (Cockburn, 2002, p. 148)

Cockburn makes a distinction between heavy and light methodologies. Heavy methodologies tend towards greater process, formality and documentation whereas light methodologies tend toward increased skills, discipline and understanding and a reliance on tacit knowledge. The former are 'optimizing' methodologies and the later 'adapting' methodologies. Crystal methods are adapting methodologies.

The following characteristics of projects have effected Cockburn's ideas about design of methodologies (Cockburn, 2002, Chap 4):

- Adding people to a project is costly in terms of salaries and the added communication burden needed to maintain productivity.

- Team size increases in large jumps. This is based on the ideas of Brooks, author of The Mythical Man Month (1995).
- Teams should be improved, not enlarged. When productivity must be increased adding people to the team is not the optimal strategy. Better strategies are training, replacing team members with more skilled people and people who are better team members, and sitting people closer together to reduce communication lags.
- Different methodologies are needed for different projects. This is based on Cockburn's grid as shown in Figure 7. Different grids are used for different project priorities, for example productivity rather than criticality.

Cockburn provides examples which embody his ideas about methodologies. He names the methodologies for crystals; Crystal clear, Crystal yellow, Crystal orange. He ranks XP as in the C4 to E14 category. The methodologies he describes are all at the smaller project size, ranging from 1 to 40 people. The Crystal methods are based on general principles about software development, people, communication, projects, and methodology weight. Software development is viewed as a cooperative activity supported by invention and communication.

People are good at observing and taking initiative and these characteristics should be supported by the methodology used. Communication is most effective when it is face-to-face and the methodology should strive for this style of communication.





Defects cause loss of...	Methodology prioritized for criticality					
Life (L)	L6	L20	L40	L100	L200	L500
Essential money (E)	E6	E20	E40	E100	E200	E500
Discretionary money (D)	D6	D20	D40	D100	D200	D500
Comfort (C)	C6	C20	C40	C100	C200	C500
	1-6	-20	-40	-100	-200	-500
Number of people involved $\pm 20\%$						
Key Crystal clear  Crystal yellow  Crystal orange  Crystal red 						

Figure 7: Characterising projects by communication load and criticality

Adapted from Cockburn (2002, p. 162).

Projects are unique 'ecosystems' and the methodology should be adjusted to fit the unique ecosystem. The methodology should also be as light as possible while remaining sufficient to complete the project effectively. Lighter methodologies deliver working software more quickly than heavy methodologies, but larger projects need heavier methodologies to cope with the heavier communication load and greater ceremony the larger team needs to be effective. As project size increases the project manager should maintain the lightest methodology possible for the situation rather than increase weight.

Cockburn's Crystal methods are a family of methodologies with the same basic values, goals and principles. The method most suitable for a project is selected and then tailored to meet the specific details of the project. Three crystal methods are described by Cockburn because they have been used in real projects.

In the Crystal methods the main principles are that people and communication are of primary importance for project success and development must be incremental and concurrent. The techniques of the methods are designed to support the positive traits of people most specifically:

- People are good at observing their environment (e.g. problems that arise, things that need attention) and should be encouraged to take the initiative.
- People have the ability to learn and they are malleable.
- People take pride in work, in accomplishment and in contributing to some outcome.
- People like to be good citizens (e.g. being punctual, following code conventions, using code libraries, informally training others).
- People need feedback that is clear and frequent.
- Peoples' personalities should be matched to their role.
- Effective collaborative groups work by consensus.
- Only the people on the team can deduce and decide what will work in that particular environment and then tune it.

Perspective

I believe that ASD takes the perspective of the project manager and the developers. Any group that wants to create and document their own methodology would also find this set of methods and the philosophy behind them useful.

Objectives

The purpose of Cockburn's book is to describe how to create, document and distribute a methodology that an organisation has developed along with how to select and tailor a methodology appropriately at the start of a project. Advice on how to document and distribute changes to a methodology is also provided. The application of Cockburn's theories are designed to lead to "methodologies whose priorities are being productive and responsive to change" (Cockburn, 2002, p. 171). The main objective is to produce software for business problems and the secondary objective is to set up for the next project by providing documentation appropriate for maintenance, understanding of the development process and any lessons learned.

The objective is also to structure a team and its environment in such a way that the team is highly productive and satisfied during development.

Domain

Crystal methods are designed for business environments with specific defined problems.

Target

Type of problem - any

Technology environment – object technology is assumed by the methods are designed to work in when developing any technology

Size of organisation – not specified

Type of organisation – not specified

Size of project – any size, but smaller projects of 2-8 people are desirable

Type of development – business systems and applications

Type of application – software deployed using the Web but not restricted to this

A number of 'sweet spots' for a project are listed. These are factors that enable a project to be carried out in the most agile manner. They are to be aimed for rather met.

- Two to eight developers in one room to support communication flow.
- Onsite usage experts available at all times to provide feedback to the developers.
- One month increments to provide rapid feedback on project and product progress. This allows any changes to requirements to be quickly accommodated and any adjustments to be taken after a short time period.

- Fully automated regression tests (unit and/or functional tests). This enables the state of the product to be assessed rapidly.
- Experienced developers as this allows for the smallest possible development team.

Model

There is no model specified in this set of methodologies. Whatever the developers need at the time is appropriate. This is likely to include UML as it is a set of models that are suitable for the current technologies.

Techniques

Detailed techniques for development are not given. Any techniques can be used as long as they promote concurrency, communication, iterative and incremental development and effective team work. Techniques from standard software engineering and project management techniques are acceptable but the precision of their use is adjusted depending on the precision, criticality, accuracy, visibility and scale of the individual project. Techniques from other agile methods are acceptable such as techniques from XP, Scrum, DSDM and ASD. Techniques from RUP (Kruchten, 2000) and Design for Use interface design (Constantine & Lockwood, 1999) are also acceptable. Cockburn's advice on techniques is:

Concurrent development – this is used to reduce overall development time and to reduce the negative effect on development time of unexpected changes in requirements. This involves carrying out analysis, design coding and testing at the same time. Some lag between phases is acceptable to allow work products from an earlier phase to be brought to a level where they are useful to the next phase.

Iterative development – this is proscribed with increments of 4 months or less with a preference for 1 – 4 month increments.

Adapting the methodology at project start up – this involves a series of interviews and that are carried out at the beginning of the project. The development team or project leader interview stakeholders and developers about project priorities, work product quality, how iterations should be managed, and how communication will be organised. A detailed list of questions is provided and the answers are used to form a base methodology. The methodology will then meet the needs of management, customers and the team with regard to iteration length, artefacts to be produced, workflow, roles, software quality level, and all of the other elements of a methodology as described above in the section 'assumptions and values'. A final meeting held by the team determines the base methodology. This should take ½ to 1 day and no more.

Reflective workshops – these are meetings used for adapting the method during development. Reflective meetings are held at the middle and end of each increment to discuss what has

occurred, what the team has learnt and what they should change. All of the methodology elements should be assessed. Mid iteration workshops are not needed if increments are 3 weeks or less. A method for running a reflective workshop is provided.

Techniques for minimising documentation – these are; making visits in person, using printing whiteboards to record decisions and designs and focusing on producing minimal documentation whenever documentation is needed.

Techniques to support effective communication - face-to-face communication should be supported whenever possible. This is supported by collocation of people all in one room, the use of information boards for communication about system status, work progress and anything else important to the team and the project. Room arrangements should be made in a way that allows an open common space for communication and private or quiet areas so that developers can have privacy when necessary.

Publishing a methodology – a new methodology must be documented so that it can be reused, adjusted, improved and distributed effectively. The publication can be organised by role-deliverable-milestone. The text, to minimize size, should be made up of examples of work products, technique guides with no text but references to other publications describing how to use a technique, a set of role descriptions, and work product descriptions.

Training using role plays – this technique is described so that teams can experience a methodology before using it on a project.

Techniques for Crystal Clear

- One team, seated in one room or in close proximity
- Software delivered in increments at 2-3 month intervals
- Progress tracked using milestones
- Automated regression testing
- Direct user involvement
- Two user viewings per release
- Phases are started as soon as the previous phases products are stable enough to review (i.e. programming begins as soon as there are some stable requirements and UI and object design)
- Product and methodology tuning workshops are held at the start and middle of each increment

Techniques for Crystal Orange

- Time to market is important
- Time and cost must be kept down

- Communication with present and future staff is needed
- Three or four technologies are used
- Work products are developed to the degree where they are understandable by team members and can be peer reviewed.
- Policy standards as for crystal clear
- Incremental delivery period – 3-4 months

Work product templates, coding style, user interface standards and regression testing details are left for the team to decide upon.

Tools

A minimal tool set is specified. This includes a compiler, versioning and configuration management tools and regression testing tools and a printing whiteboard for documenting plans, designs and decisions.

Scope

Crystal methods assume no particular phases but only insist on incremental and iterative delivery. Phases of development in Crystal methods are at the discretion of the team. However in a discussion of ‘upstream’ and ‘downstream’ activities the concurrent activities listed are; requirements, UI and object design, programming and testing. Cockburn’s model of successful concurrent development is shown in Figure 8. Cockburn believes that a methodology should have phases but they can consist of any combination of activities that move the project to completion.

Outputs

The output is working software delivered incrementally and any other deliverables defined by the customer. Other artefacts are negotiable with the customer. The two methods described by Cockburn have the following specified outputs (see Table 8).

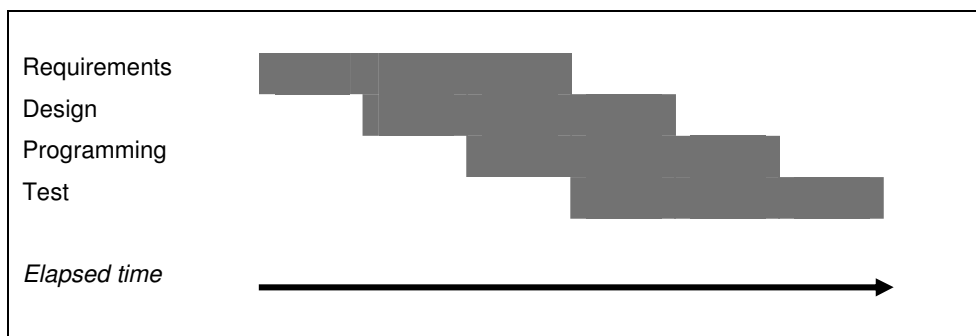


Figure 8: Cockburn’s model of concurrent development (Cockburn, 2002, p. 132)

Table 8: Cockburn's Crystal methods (Cockburn, 2002)

Crystal Clear	Crystal Orange
	Requirements document
Release sequence	Release sequence
Schedule of user viewings and deliveries	Schedule
Annotate use case or feature descriptions	
Design sketches and notes as needed	
Screen drafts	UI design document
Common object model	Common object model
Running code	Source code
Migration code	Migration code
Test cases	Test cases
User manual	User manual
	Status reports
	Inter-team specs
and optionally: Templates for work products Standards for code and user interface Standards for regression testing	

Practice

Background

Crystal methods are based on Cockburn's experience as a developer and his investigation of successful software development projects.

Roles and responsibilities

The number of roles is dependent on the size of the project. Examples for Crystal clear and crystal orange are provided but the activities undertaken by the people in the roles are allocated by the project team.

Crystal Clear – sponsor, senior designer programmer, designer programmer, user.

Crystal Orange – sponsor, business expert, usage expert, technical facilitator, business analyst/designer, project manager, architect, design mentor, lead designer-programmer, other designer programmers, UI designer, reuse point, writer, tester.

In larger projects such as that for Crystal orange people are arranged into teams for systems planning, project monitoring, architecture, technology, functions, infrastructure, external test. Each team is cross-functional containing a business analyst, analyst designer, designer programmers, database designer, other technology experts and tester.

Difficulties with Crystal methods

Difficulty arises with multi-site teams that are in different countries and time zones and with virtual teams at multiple sites because of the difficulties of communicating effectively without heavy documentation. Crystal methods have not been used with large geographically separated teams or with life-critical systems.

Skill levels

No explicit levels are mentioned although Cockburn recommends that the lighter the methodology the higher the level of skills that are needed. Training is recommended to improve the skills of the team.

Tailorability

A base method is developed by the team at the start of the project then tailoring is carried out during the project. This is achieved by reflecting every few weeks on what works well and what should be changed. The method is normally tailored when the 'sweet spots' are not achievable. However scaling to very large projects and large geographically distributed projects is covered in theory by Cockburn's tailoring mechanism but it has not been tried in practice.

An alternative to making your own method is to select one of the Crystal methods based on the size of the project and the criticality of the project. Then the method is tailored using the same review techniques as those described above to adjust the Crystal method to your unique project.

The techniques recommended by Cockburn can be substituted with those from similar methodologies if they have the same effect on development. "Substitution of elements from similar methodologies is permitted. For example, the team could decide to use Scrum or DSDMs timeboxing and dynamic prioritization policies, Scrums daily stand-up meetings, pair programming from XP and so on" (Cockburn, 2002, p. 203).

Bibliography for Appendix K

- Avison, D. E., & Fitzgerald, G. (1995). Chapter 7 Methodologies: issues and frameworks. In *Information systems development: methodologies, techniques and tools* (2 ed.). London: The McGraw-Hill Companies.
- Avison, D. E., & Fitzgerald, G. (2003). *Information systems development: Methodologies, techniques and tools* (3 ed.). London: McGraw-Hill.
- Beck, K. (1999). Embracing change with Extreme Programming. *Computer*, 32(10), 70-77.
- Beck, K. (2000). *Extreme programming explained: Embrace change*. Boston: Addison-Wesley.
- Beck, K., & Cunningham, W. (1989). A laboratory for teaching object oriented thinking, *Conference proceedings on Object-oriented programming systems, languages and applications, OOPSLA '89* (pp. 1-6). New York, NY, USA: ACM Press.
- Brooks, F. (1995). *The mythical man month: Essays on software engineering*. Reading, MA: Addison-Wesley.
- Checkland, P. (1999). *Systems Thinking, Systems Practice. Soft systems methodology: A 30-year retrospective*. Chichester: John Wiley & Sons, Ltd.
- Cockburn, A. (2002). *Agile software development*. Boston: Addison-Wesley.
- Constantine, L., & Lockwood, L. (1999). *Software for Use*. Reading MA: Addison-Wesley.
- Dynamic Systems Development Method Ltd.; the DSDM lifecycle. (1997 - 2005). Retrieved 8 May, 2005, from <http://na.dsdm.org/en/about/lifecycle.asp>
- Dynamic Systems Development Method, Version 2. (1995). (2 ed.). Ashford: Tesseract Publishing.
- Ehn, P. (1992). Scandinavian design: On participation and skill. In *Usability: turning technologies into tools* (pp. 96-132). New York, NY: Oxford University Press.
- Highsmith, J. (1997). Messy, exciting, and anxiety-ridden: Adaptive software development. *American Programmer*, 10(4), 23-29.
- Highsmith, J. A. (2000). *Adaptive software development: A collaborative approach to managing complex systems*. New York, NY: Dorset House Publishing.
- Holland, J. H. (1995). *Hidden order: How adaptation builds complexity*. Reading, Massachusetts: Addison-Wesley Publishing Co.
- Kruchten, P. (2000). *The Rational Unified Process: An introduction* (2 ed.). Boston: Addison-Wesley Longman.
- Mumford, E. (1995). *Effective requirements analysis and systems design: The ETHICS method*. Basingstoke, UK: Macmillan.
- Musashi, M. (2000). *The book of five rings* (T. Cleary, Trans.). Boston, MA: Shambhala Publications.
- Naur, P. (1992). Programming as theory building. In *Computing: A human activity* (pp. 37-48). Reading, MA: Addison-Wesley.
- Schwaber, K. (1995). SCRUM software development process. Retrieved 3 October, 2004, from <http://www.controlchaos.com/old-site/scrumwp.htm>

- Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum*. Upper Saddle River, New Jersey: Prentice Hall.
- Senge, P. (1990). *The fifth discipline: The art and practice of the learning organisation*. New York: Currency Doubleday.
- Stapleton, J. (1997). *DSDM Dynamic Systems Development Method*. Harlow, England: Addison-Wesley.
- Takeuchi, H., & Nonaka, I. (1986). The new new product development game. *Harvard Business Review*, 64(1), 137-146.
- Weinberg, G. (1998). *The psychology of computer programming*. New York, NY: Silver Edition, Dorset House.