COMP 6481 – Fall 2006
System Requirement Specifications

Instructor Name: **Dr. Olga Ormandjieva**

Software Requirements Engineering using Agile Methods

Project Report

Name: Muhammad F. Ahmed	ID: 5774454
Name: Nick Simmons	ID: 4439449
Name: Rima Alaoui	ID: 5814855
Name: Tanvir Khan	ID: 6019285
Name: Rajbir Padda	ID: 6004628

Due Date: 7th December 2006

Table of Contents

Introduction

A software requirement "defines a capability needed by a user to solve a problem to achieve an objective"[1]. The defined requirements of a system act as the link between end-users and developers.

Requirements engineering - which encompasses all of the tasks of gathering, defining, analyzing, and managing the requirements for a software system - is the first, and arguably the most important, phase of a development process. The primary goal when developing a system is to ensure that the finished product matches the user's needs and wants as close as possible. This is considered a qualitative measurement of the quality of a system, and in the software industry quality has traditionally been a weak point. Thus it is apparent that proper requirements specification leads to improved quality.

Requirements engineering can be roughly divided into two sub-tasks: Requirements development and requirements management[1]. The development phase involves gathering, analyzing, specifying, and validating the requirements. The management phase involves gathering, analyzing, and making decisions on requested changes to the requirements baseline.

- Requirements gathering may involve market analysis, or specific customer requests.
- Requirements analysis could be handled by doing a survey of the market to determine if a certain requirement is really needed, or maybe a feasibility study is done to determine if it's even possible to implement the requirement using today's technology.
- Requirements specifications may be written using visual models, natural languages, or even formal languages such as Z or Larch.
- Validation may be as simple as a revision of the requirements by stakeholders or as comprehensive as using formal verification tools when dealing with formalized specifications.
- Requirements management can vary significantly depending on the goals and important factors surrounding a software project.
- Risk management can play a key role in the decision process for changes, or customer satisfaction and delivering everything that is deemed necessary takes precedence over delivery dates.

The entire requirements engineering process itself can be handled in several different ways. For example, the requirements can be developed entirely up-front and contractually agreed upon (waterfall method), or an iterative approach can be taken where the requirements grow over time.

It should be apparent now that requirements engineering is a vital and expansive discipline that can be approached in many different ways. The goal always remains the same however. Requirements should be used as a means of helping developers deliver what the end-users are looking for no matter what the approach may be.

Agile is a relatively new methodology that keeps this key goal closely in mind while offering a significantly different approach when compared with traditional software development practices.

I. Agile Methodology

A. Overview

Agile methodologies began evolving in the 1990's. These new methodologies were created by industry practitioners who, from practical experience, were able to identify the pitfalls in existing software development processes and proposed new solutions for these problems. "Each had a different combination of old ideas, new ideas, and transmuted old ideas. But they all emphasized several key characteristics: close collaboration between programmers and business experts, face-to-face communication, frequent delivery of new deployable business value, close working self-organizing teams, and ways to craft the code and the team such that the inevitable requirements churn was not a crisis"[3].

In 2001 the major practitioners of these new methodologies came together and created a generalized, common understanding about what these new ideas had in common. The end result was the "Agile Manifesto". This published collection of values and principles has become the backbone for a paradigm shift in the software industry that to this day is still in its infancy.

Agile Manifesto

The Agile Manifesto contains four statements of shared values. They read as follows:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan[4]

These statements are saying that while there is value in the items on the right, the items on the left are deemed to be the most valuable. These four statements alone define the general methodology behind Agile and its applied approaches.

There is also list of principles that accompanies these statements. The principles follow the four key value statements, but add some breadth to the definition of

Agile. Following are the ways to handle requirement engineering using Agile principles:

Agile Principles	Requirements With Agile
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	Customer satisfaction is critical and requirements play an important role in ensuring customer satisfaction so clearly requirements must play as vital a role in Agile as they do in traditional methods.
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	This principle is against the response most people have when changes are introduced on a software project. Agile tries to improve the requirements management task by making change an expected situation instead of an exceptional one.
The best architectures, requirements, and designs emerge from self-organizing teams.	It identifies a key concept in how Agile teams work best.
Simplicity--the art of maximizing the amount of work not done--is essential.	This is an interesting concept that is against many of the traditional methods of requirements specification and management.
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	This principle is more of a psychological theory than a principle, but has a direct effect on Agile requirements engineering when combined with the last principle.
Business people and developers must work together daily throughout the project.[4]	Face to face collaboration between end-users and developers on a regular basis will result in higher quality software that meets the end-user's vision.

B. Advantages and Disadvantages

One important thing to keep in mind when discussing any software development methodology or approach is that there is no "silver bullet" approach. Every methodology has its advantages and disadvantages. Agile has evolved from pragmatic professionals in search of a better solution to the problems they encounter in their projects. But since every project is different the proposed solutions may work very well in one project and very poorly in another. With

specific regard to requirements engineering it is important to understand when Agile is worth considering.

The advantages of Agile methods lie in the promise of being able to adapt quickly to constantly changing requirements, while accepting that the final scope, cost, and time required to complete the system are at best a ballpark estimate at the start of a project and only become fixed at the very end of the lifecycle. They also facilitate rapid development by reducing the effort spent on certain tasks that are deemed to be aside from the actual goal of the project – producing working software. Some of these tasks include detailed written requirements specifications, utilizing well defined but complex and abstract formal languages for specifying the requirements, and requirements traceability and verification.

These tradeoffs seem reasonable when working in an environment where it is expected that requirements will adapt and flexibility is of the utmost importance. It is also much easier to accept when the development is relatively small and detailed requirements specifications and formal verification can be replaced by direct verbal communication. However, in some cases this approach is not beneficial or even feasible.

Example 1: When requirements are provided by the customer in great detail and are contractually agreed upon. In this case requirements are not expected to change and Agile methods will not prove to be as beneficial, especially if the contract requires detailed estimated costs and schedules.

Example 2: a project that involves safety critical systems such as medical or aircraft navigational software. These types of systems have strict guidelines and verification procedures that must be followed. Since Agile tends to lean away from detailed written requirements and formalized languages that can be exhaustively verified using proven formal techniques and tools, these types of systems should not be developed using Agile.

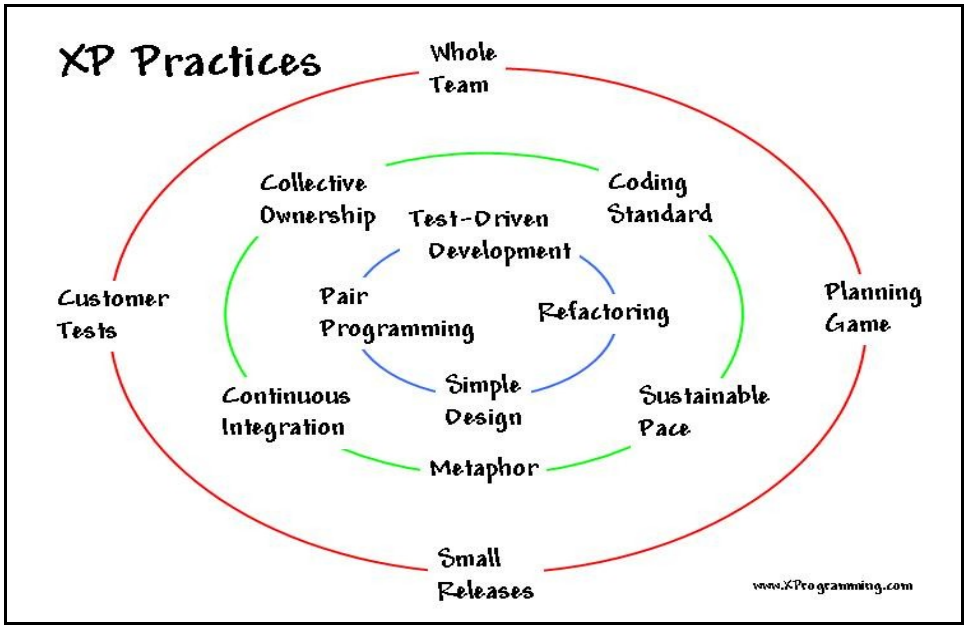
C. Practical Examples

There exist many practical applications of Agile methodology that can be directly applied by a software development team. Extreme programming, Agile Unified Process, Feature Driven Development and Scrum are some examples to name a few. Most of these processes predate Agile, as it was the founders of these applied methods that collaboratively created the Agile Manifesto. Each has its own unique approach to software development, and more specifically requirements, but all share the same common values of the Agile Manifesto. A careful analysis of each process and approach to requirements engineering will provide some insight into the Agile methods that are directly applied in industry.

II. XP – Extreme Programming

A. Overview

Extreme Programming (XP) is the most well known Agile method. It offers a comprehensive process that covers all phases of the development cycle and proposes radically different ideas that go against almost all previously existing software processes. It was "created by Kent Beck, Ward Cunningham, and Ron Jeffries during their work on the Chrysler Comprehensive Compensation System (C3) payroll project. Kent Beck became the C3 project leader in March 1996 and began to refine the development methodology used on the project. Kent Beck wrote a book on the methodology, and in October 1999, *Extreme Programming Explained* was published."[7]



Extreme Programming http://www.xprogramming.com/xpmag/whatisxp.htm

XP is "based on the values of simplicity, communication, feedback and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation."[8] In addition to iterative development cycles it recommends twelve core practices:

- Planning Game
- Small, frequent releases
- System metaphors
- Simple design

- Testing (Automated using Test-Driven Design)
- Frequent re-factoring
- Pair programming
- Team code ownership
- Continuous integration
- Sustainable pace
- Whole team together
- Coding standards[8]

B. Requirements Development

The core values of XP have a direct influence on requirements development. The early stages of the XP lifecycle (Exploration and Planning) are focused on gathering and analyzing the requirements.

Requirements come directly from the customer who writes “user stories” on paper index cards. These cards are then used to generate a task list for iteration. Further elaboration is done by developers who use index cards to document class responsibilities and collaborators (CRC). Some object modeling is done on a whiteboard with everyone involved. This promotes understanding and collaboration, reducing the need for detailed documented specifications.

Validation also comes directly from the customer in the form of acceptance tests. These tests are identified along with the requirements and are automated by the developers during the implementation.

“Test-Driven Design”, a key practice in XP, ensures that the product being developed conforms to the tests provided by the customer. The idea is to write an automated test for a requirement before writing the actual implementation code. The test is run and should fail (Since the code to make it pass does not exist yet!). Just enough coding is done to make the test pass and from that point on the code can easily be verified.

C. Requirements Management

Requirements are managed by the customer directly during the planning phase of iteration by selecting which story cards to implement. The developers break the stories into fine grained tasks and estimate the time required to complete each task. The total task-level estimates are combined and adjustments are made to the scope or time of the iteration if needed.

Changes can be made at any time because of the nature of an XP environment. Developers get direct input and feedback from the onsite customer. There is very

little detailed up-front design so changes made before implementation have nearly no effect on existing documentation. Changes made after the implementation can be made by modifying the automated acceptance tests and re-factoring the code just enough to make all the tests pass.

III. AUP – Agile Unified Process

A. Overview

The Agile Unified Process (AUP) is a modified version of the Rational Unified Process (RUP) that was initially created by Rational Software. It uses Agile techniques and concepts while following the Unified Process model. Like RUP, there are four main phases:

- Inception
- Elaboration
- Construction
- Transition

Each phase has specific objectives. AUP differs from RUP in that it attempts to reduce some of the overhead of RUP. AUP is based on some key philosophies that define it as an Agile process:

- Experienced team members
- Simplicity
- Agility
- Focus on high-value activities
- Tool independence
- AUP is a guideline that should be tailored[9]

The goal of AUP is to use a structured but flexible collection of activities and artifacts. The structure comes from the very well defined Unified Process model that contains numerous disciplines, activities and artifacts. The flexibility comes from the philosophies that it values. AUP recommends reducing the set of disciplines, activities and artifacts to a minimum that are deemed to be of high value to the specific project at hand. All UP artifacts are optional and should only be created if they are considered to add value. AUP also recommends Agile techniques for key RUP activities that are typically of high value but very high maintenance. One example is Agile Modeling – an approach to modeling that states “The purpose of modeling is primarily to *understand*, not to document”[10].

B. Requirements Development

The Unified Process defines approximately 50 artifacts that can be created during a project, but in AUP all of these are optional and a "less is better" approach is taken.

- Requirements gathering and analysis begins in the Inception phase, where the vision is created and the architecturally and/or riskiest requirements are captured in detail.
- The elaboration phase is where the majority of the requirements are analyzed and specified. The customer actively participates in the "requirements workshops". Requirements are specified by writing use cases and performing Agile Modeling.
- Validation is done by producing many prototypes starting as early as the inception phase.
- Testing is done continuously and customer feedback is gathered often.

Since the Unified Process is an iterative process customer feedback from previous iterations act as input for future iterations. The result is a product that is refined iteration by iteration.

C. Requirements Management

Requirements can be managed using existing Agile methods (Scrum for example). Continuous testing, prototyping, and customer feedback leads to change requests. Changes are managed through disciplined configuration management, version control, and a change request protocol[6]. Requirements are continuously re-organized based on risk, priority, and status. Each iteration focuses on implementing the requirements that have the highest priority and risk first. The status of each requirement is tracked to known what requirements are finished, in process or not started yet.

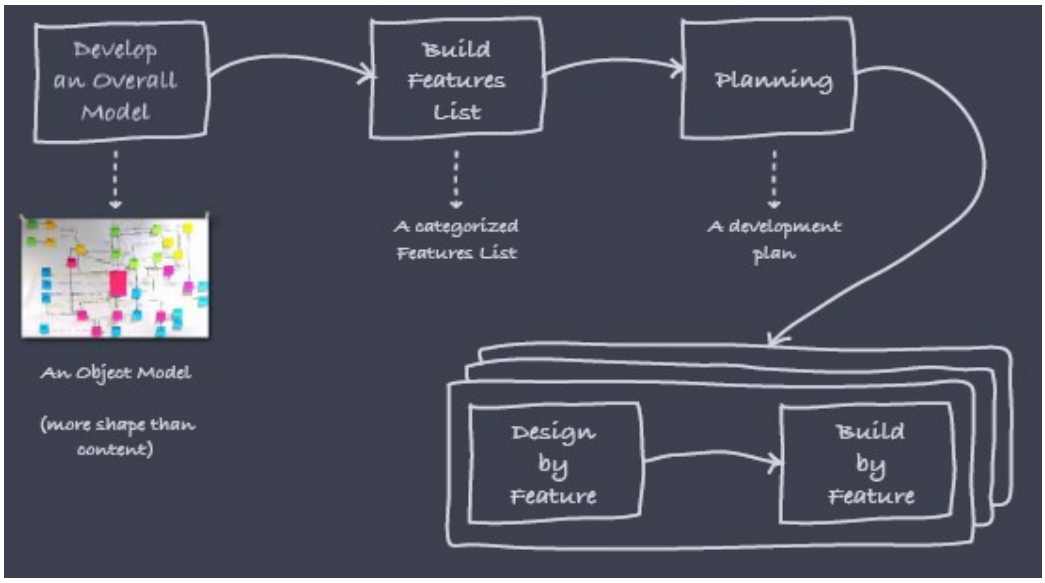
IV. FDD – Feature Driven Development

A. Overview

Feature Driven Development is "an Agile method that blends a number of industry-recognized best practices into a cohesive whole" [11]. FDD is based around the concept of client-valued units of functionality what are called "features". Its goal is "to deliver working software repeatedly in a timely manner" [11].

Feature Driven Development was created by Jeff De Luca while working on a software development project in 1997. FDD was introduced to the world in a book co-authored by De Luca called “Java Modeling in Color with UML”. FDD methodology is comprised of 5 basic activities:

- Develop an overall model
- Build a features list
- Plan by feature
- Design by feature
- Build by feature



Feature Driven Development(FDD)

The first three activities are done once at the beginning of a project and form the startup phase. The last two activities form the construction phase and are done iteratively. Between the two phases is a key milestone. “The FDD startup phase contains predictive metrics to project future effort based on early available indicators.” [13] Once the short startup phase is completed results are seen quickly. Features are by definition less than two weeks work to complete, so a constant stream of features and feature sets are being delivered throughout the construction phase.

B. Requirements Development

In FDD most of the requirements are done up-front. The goal is to provide a predictive plan as soon as it is feasible to do so. The initial phase focuses on creating an overall model of the system, with active participation by the customer. The end result is a collection of class and sequence diagrams. These models

drive the next phase which is focused on building a list of all the features of the system that need to be implemented. The features list is built through functional decomposition.

The system is broken down into subject areas, then business activities, and finally business activity steps. Each business activity step is fine grained enough that its estimated time to complete is one work week or less. The completed features list is then used to estimate and plan future time-boxed iterations, and guide the design and build iterations.

C. Requirements Management

FDD assumes that a significant amount of the features can be identified up front. But once the initial up-front requirements development and planning is competed the design and build iterations are short (2 weeks or less). At the start of iteration there is a chance to adjust and refine the requirements. When a change is made the models and feature list are updated, and work continues on the next iteration.

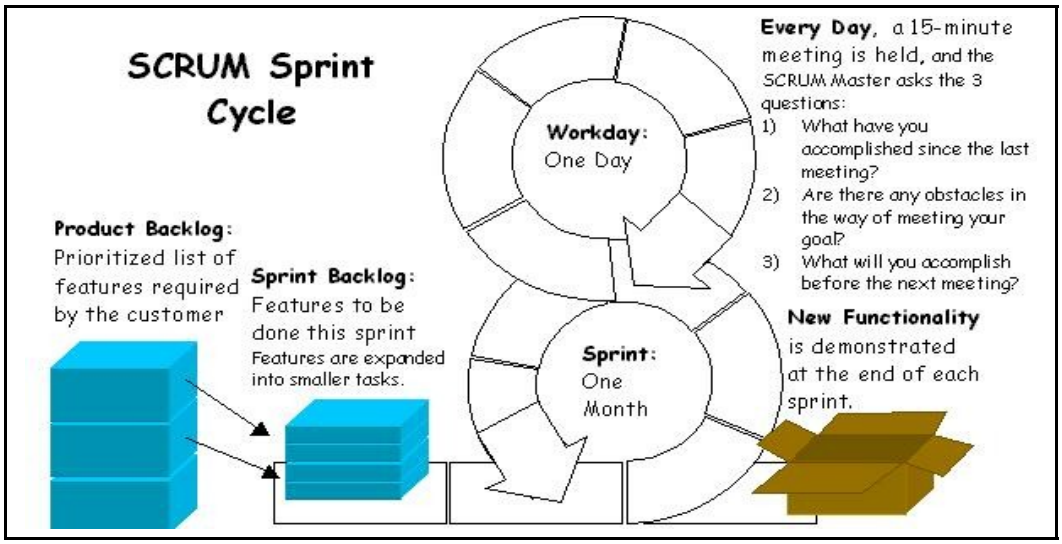
V. Scrum

A. Overview

Scrum is an agile method that radically redefines the project management aspect of software development. Scrum has been implemented by many different companies including Canon, Honda, NEC, Epson and Xerox. [14]

Scrum offers a solution for projects that are not well defined at startup. When requirements are expected to change during the process, the use of a predictive and defined model is simply is not feasible. Initial long term estimates can only be expected to represent a ballpark guess. The solution that Scrum offers is empirical, adaptive, controlled and measured with continuous rapid feedback. Some key practices of Scrum include:

- Self directed and self-organizing teams
- No external addition of work to an iteration, once accepted
- Daily stand-up meetings with special questions
- Four week time-boxed iterations called “sprints”
- Demo to external stakeholders at end of each iteration
- Each iteration is driven by adaptive client-focused planning



SCRUM

The main work products of Scrum which drives the project, are the product backlog and sprint backlog. A sprint backlog is a subset of the product backlog and is fixed during a sprint. It is the responsibility of the sprint team (the “pigs”) to manage, organize and complete all tasks listed in the sprint backlog. At the end of the sprint the sprint team presents the finished, release-quality product to the project stakeholders (the “chickens”). The time in between each sprint is used to evaluate the progress of the project and re-direct the sprint team’s effort on the next set of customer-valued items.

B. Requirements Development

Scrum defines two work products that are used for gathering, analyzing and specifying the requirements. The product backlog is populated by the product owner (which is typically the customer or someone who speaks for them). It gathers all identified requirements that will be implemented in some future sprint. It can be modified, reorganized, and re-prioritized at any time without affecting the progress of the team.

The sprint backlog is created at the beginning of each “sprint” from the highest priority items found in the product backlog. Unlike the product backlog, the sprint backlog cannot be modified by people outside of the sprint team once it has been agreed upon and the sprint has started. At the beginning of every sprint the “pigs” and “chickens” analyze the product backlog and choose a subset of items to complete for the next sprint. The sprint backlog is created with detailed task estimates (each less than 24 hours) by the sprint team. Adjustments to the scope are then made until the selected items fit within the sprint deadline (typically 30 calendar days or less). Additional specification of requirements can be done, but is not explicitly defined by Scrum.

Scrum focuses more on the management side of a project and suggests that another process (ideally an Agile one) be used inside of the Scrum framework to fill in the gaps. Validation is done at the end of the sprint when the finished product is presented to the “chickens”.

C. Requirements Management

Requirements can be managed outside of a sprint by the product owner at any time. Only the sprint backlog is locked during a sprint and cannot be modified by anyone but the sprint team. But sprints are short time-boxed iterations and changes to the items that were done within a sprint can be handled in the next sprint. In the worst case, a sprint can be terminated and a new sprint commenced by re-starting the sprint planning effort. All changes are discussed during the sprint planning and decisions are made using everyone’s input.

Conclusions

Agile methodologies offer a radical new approach to software development that seems to fly in the face of most existing philosophies. They promise high productivity, realistic adaptive planning, and the ability to react to change with open arms. However, these methodologies are very new and have yet to be proven to be any more effective than long established processes.

The Agile Manifesto is simply a collection of best practices and core values that have been accepted by many as the solution to the pitfalls of existing processes. How to apply these values and principles in an applied process that works is quite difficult. The solutions to some problems often result in new problems. Many of the Agile processes discussed make assumptions that are not always feasible or even possible.

The most important thing to be learned from Agile methodology is that a project team should focus on what works. If something isn’t working, it should be replaced with something else that is agreed upon by the team. Something that is working should be kept and reused until it no longer works. It should be clear that with all of the different processes that exists today, some in radical contrast to others, that there is no one process that will solve the needs of all projects. Rather, every project is unique and requires a unique process to fit the needs of everyone involved.

Bibliography

COMP6481 Week1 lecture notes, Dr. Olga Ormandjieva
http://www.cs.concordia.ca/~comp6481_2/COMP6481-2006/COMP6481-F06-Week1.ppt

[2] WikiPedia – Requirement analysis page, Various authors
http://en.wikipedia.org/wiki/Requirements_analysis

[3] An Agile Roadmap, Agile Alliance
http://www.agilealliance.org/resources/roadmap/

[4] Manifesto for Agile Software Development, Various authors
http://agilemanifesto.org/

[5] WikiPedia – Agile software development, Various authors
http://en.wikipedia.org/wiki/Agile_software_development

[6] Agile and Iterative Development: A Manager's Guide, Craig Larman, Addison-Wesley, © 2004

[7] WikiPedia – Extreme programming, Various authors
http://en.wikipedia.org/wiki/Extreme_programming

[8] What is Extreme Programming?, Ron Jeffries
http://www.xprogramming.com/xpmag/whatisxp.htm

[9] WikiPedia – Agile Unified Process, Various authors
http://en.wikipedia.org/wiki/Agile_Unified_Process

[10] Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, Craig Larman, Prentice Hall PTR, © 2005

[11] WikiPedia – Feature Driven Development, Various authors
http://en.wikipedia.org/wiki/Feature_Driven_Development

[12] Feature Driven Development Overview, Nebulon Pty. Ltd. © 2005
http://www.nebulon.com/articles/fdd/download/fddoverview.pdf

[13] How to Reduce the Risk of Fixed-Price Estimates, Nebulon Pty. Ltd. © 2003
http://www.featuredrivendevelopment.com/node/527

[14] WikiPedia – Scrum (development), Various authors
http://en.wikipedia.org/wiki/Scrum_(development)

[15] Agile Software Development with Scrum, Ken Schwaber, Mike Beedle
Prentice Hall, © 2002

[16] The latest FDD Processes, Nebulon Pty. Ltd. © 2005
http://www.nebulon.com/articles/fdd/latestfdd.html