Comparing eXtreme Programming and Feature Driven Development in academic and regulated environments

Serguei Khramtchenko

(Address removed)

skhramtc@wyeth.com

Final paper for CSCIE-275:
Software Architecture and Engineering

Harvard University

May 17, 2004

Abstract

This paper compares the two agile software development methodologies: eXtreme programming (XP) and Feature Driven Development (FDD). It focuses on applicability of the methodologies in an academic environment (CSCIE-275 course project) and in regulated environment (health care industry). The paper compares different aspects of project management with XP and FDD: starting from gathering user requirements and up to deployment of the complete applications.

The intended reader should be familiar with at least one of the two development methodologies. This paper is not a description of any of them: it rather highlights their differences and similarities.

Table of Contents

List of Figures

Chapter 1 Introduction

This chapter gives a high level description of eXtreme Programming and Feature Driven Development. The reader familiar with both methodologies may skip sections 1.1, 1.2 and jump to the section 1.3, which describes author's point of view to be developed in the following chapters.

In the last 10 years several "agile" software development methodologies have emerged. They position themselves as an alternative to traditional "waterfall" development model. In waterfall model the whole software project is divided into a number of stages. The typical stages are:

- Gathering user requirements;
- Design and documentation;
- Development;
- Testing;
- Deployment;

The more complete list of stages/activities may be found in CSCIE-275 lecture materials [3]. The waterfall assumes that each stage is 100% complete before the next stage starts. One of the main weaknesses of this approach is the fact the design errors are often not discovered until the deployment time. At this time the project is almost complete and the errors are usually expensive to recover from.

FDD and XP are the two of agile development methodologies. Agile development tries to avoid the main weakness of "waterfall" by doing iterative development. Each iteration is meant to be short (1-3 weeks) and includes all of the above steps. This guarantees that design errors are discovered at early stages of development.

1.1 eXtreme Programming

XP development methodology appeared as one of the attempts to simplify and improve software development methodology. In 1996 Kent Beck started a project at Daimler-Chrysler, which was based on his ideas of software development. Some of them radically change the way of development. He then published his ideas in a book titled "Extreme Programming Explained: Embrace Change" [5]. Most of software projects may be described as a meticulous implementation of user requirements, while XP stresses the customer's satisfaction.

XP project starts with collecting customer's stories. Each story is written by customer and consists of one paragraph of non-technical text. The purpose of the story is not to provide all the details of any particular scenario, but rather to be able to estimate how complex a part of the system will be, and how long it may take to implement it. All the details of the story will be clarified with the customer immediately before the story's implementation starts.

The next stage is a release plan. The release plan specifies which user stories should be implemented for which system release. Each release consists of a number of iterations. Each iteration will have part of planned user stories implemented. An iteration includes

- planning: the stories to implement are chosen, their details are clarified;
- coding: implementation of user story;
- testing: at least one unit test per class;
- acceptance test: if successful, the new functionality is considered finished; if failed, it brings to the beginning of the iteration.

While the described steps are simple, they do not say much about the spirit of XP. XP puts its spiritual principles over the rigid project plan. These principles are found on the XP web site [2]:

- **Simplicity**: try to come up with the simplest solution that works. Many projects lose momentum when the developers spend too much time on the "universal" solutions

that may accommodate any change in user requirements, deployment platform etc. These capabilities may never be demanded.

- **Communication:** this includes both communications within the development team and with customers. The communication is the key to success. It happens way too often that the developers implement something completely different from what customer expects. This may happen as a result of obscure/incomplete user requirements, misinterpretation by developers etc.
- **Testing:** automated testing is essential for XP. One of the XP practices is that every developer may change any code if needed. If any bugs are introduced, unit testing allows catching them immediately.

1.2 Feature Driven Development

Feature Driven Development is an agile software development methodology by Jeff De Luca and Peter Code. This methodology got its recognizable name in 1997. FDD followers discuss the methodology and processes in the FDD community web site [1]. FDD claims that it achieves the repeatable success in software projects. It has more formal requirements and steps than XP, but adds a precise tracking of progress.

FDD development consists of the two main stages:

- Discovering list of features to implement;
- Feature-by-feature implementation;

Discovering list of features is a critical process. The quality of this step largely defines how precise the project will be tracked, how maintainable and extensible the code will be. This process requires full-time participation of customers. The outcome of this step is the UML diagrams of problem domain. If the two-way development tool is used, than UML diagrams are backed up by the compilable code in the target programming language.

The list of the features is derived from UML diagrams. The features are expressed in a language that is clear to both development community and customers. For example, imagine a classical shopping cart example where the shopper logs in to an online store to buy some goods. The UML diagram may include classes like *ShoppingCart*, *Item*, *Shopper*. The resulting list of features may include:

- Create new *ShoppingCard* for *Shopper*;
- Add new *Item* to *ShoppingCart*;
- List all *Items* in *ShoppingCart*;
- Calculate the total price of the *Items* in the *ShoppingCart*;

This list of features presents a value to a customer (sponsor), since it directly reflects the functionality that will be available in a software application. This list also contains granular units of work for developers, where every feature is small enough, so the development may be done in short iterations.

The implementation starts from grouping the related features into work package. A work package should be complete in one iteration, usually 1 to 3 weeks long. A complete work package presents the working piece of software that the customer may play with. Each iteration includes:

- Kickoff meeting for a work package: details of included features are clarified.
- Design: required classes/methods/documentation created.
- Design review: either accepts of rejects the offered design.
- Development: implementation and unit tests are created.
- Code review meeting: peer coder review is performed.
- Release meeting: implemented features are released into a build process.

1.3 Vision statement.

This section states the author's vision that is supported by the rest of this paper. Both FDD and XP are lightweight software methodologies. These methodologies recognize the fact that the development process became over-managed over the years. There are too many rules to follow. Adding new steps to the process does not improve it, but makes it harder to follow. Both FDD and XP choose only a subset of steps that are required in a traditional waterfall process. However, the chosen steps are often quite different. They define the area of applicability of each methodology.

- XP seem to be better suited for volatile projects, where user requirements are obscure or change often. XP deals with such projects better since it deliberately avoids any activities that are not immediately required for current implementation stage.
- XP seems to be less scalable than FDD. XP heavily relies on communications within the team, which becomes harder as team grows.
- FDD offers better predictability if requirements for the project are more stable.
- FDD has methods to track project's progress, which is more appealing in corporate environment.

Chapter 2 Gathering User Requirements.

This chapter discusses the process of collecting user requirements for software projects. Traditionally the user requirements are written in a document that has a detailed description of what has to be implemented by software system. Such a document becomes the basement for the following stages in the process:

- implementation must address all the requirements listed in the document.
- acceptance test becomes a comparison of what is written in a document and what is implemented.

Use cases, usage scenarios, non-functional requirements and other means to describe the interaction of users with the system often accompany the user requirement.

The main weakness of such traditional way to collect user requirements is that they are written in stone. Once the requirements document is complete, customer stops participating in the project, or plays the role of an observer. The errors and/or omissions in the user requirements, as well as misinterpretation by developers often propagate to the last stages of the project, when they are expensive to correct.

Both XP and FDD reject the traditional way of collecting user requirements and replace it with their own methods. However, the methodologies put a different weigh on this stage.

XP: Gathering user's stories.

XP puts the minimum efforts into defining user requirements. They are collected in a written form of short stories. The purpose of the stories is to understand the scope of the project and to estimate the implementation time with low probability of error. The process of collection user stories depends on the project and takes not longer than a couple of weeks. 60 to 100 stories should be enough for one release. Details of each story

will be clarified at iteration planning meeting with the customer. Such a light way of collecting user requirements allows to easily adopting changes.

FDD: Building problem domain.

Gathering user requirements is critical for the success of FDD project. It is so important that the process has a proud name of "Process One". The outcome of the process is a UML diagram and a list of features. UML diagram(s) define the problem domain of the software system. If implemented well, it covers all the relevant areas of business and allows to easily adding features for subsequent releases. The list of features becomes the basement for the project timelines: it is used to track the progress. In this respect FDD depends on user requirements as much as waterfall does. Having understood this, FDD changes the process of collecting requirements: the customer and developers create them interactively.

The process involves developers familiar with UML and customer representatives, including problem domain experts. The modeling utilizes UML in color. This is a version of UML introduced by Peter Code and described in [4]. The short introduction is given in the Appendix 1. The modeling starts with a short introduction to UML for domain experts. While they are not expected to become UML gurus I witnessed amazingly quick learning of color UML.

The modeling itself is split into iterations. Each iteration starts from user's story about a small fragment of business. The team then splits into several modeling groups; each of them includes developers and domain experts. Each group creates UML model that reflects the story. The teams present and discuss the models. At the end of iteration the best one is chosen/merged. The modeling iterations continue until the full problem domain model that satisfies all the stories is created.

The good problem domain covers customer's business. It supposed to be stable: user requirements for the application may change, while business usually stays the same. In addition to creating object-oriented model of business, Process One indirectly indicates the duration of development phase. FDD rule of thumb says: the development team will spend approximately 6 months of development for each 2 weeks of modeling.

Chapter 3 Design and documentation.

XP ignores this step almost entirely. The recommendation from XP people is this: “We suggest that you should write the program to need as little documentation as possible”.

FDD does not require create design documentation either. At the end of Process One the developers usually create a description of UML diagrams, documenting the alternative ways that where rejected and reasons behind the decisions. The document becomes useful later: in a long project people may forget details of initial decision, so the document serves as a reminder. If customer requests the formal user requirements may also be written on the basis of this document.

While FDD does require many formal documents to be created, it strives to make the project information publicly available. The good place for it is an internal website that contains all the information about the project: list of developers, sponsors and domain experts, UML model and comments, discussion forums, coding conventions, list of tools/libraries used, unit test reports, progress reports etc.

Chapter 4 Development and Testing.

Both FDD and XP develop the project in short (1 to 3 weeks) iterations. This chapter describes the content of the iteration in both methodologies. It also highlights the distinctive development practices of the methodologies. The description of FDD practices is mainly based on author's experience. As exposure to XP was limited, the XP description is a combination of personal experience and recommendations from XP web site [4].

The development team has a different structure in XP and FDD. FDD requires a number of chief programmers (CPs) to be selected. CPs are most experienced programmers in the team that play the role of technical leaders. They also take some additional responsibilities. In the team hierarchy CPs stand in between the developers and a project manager. XP development team does not require such a hierarchy.

4.1 Iteration planning

Iteration starts with the planning meeting. While both methodologies use these meeting as a formal start of iteration, the performed activities differ.

4.1.1 XP planning.

The customer chooses a set of user stories to implement. The team communicates with the customer to clarify the details of the activities. The user stories are rewritten in developer's language and required time is estimated. Failed unit tests and bugs from the previous iterations are also included into the list of tasks for current iteration.

4.1.2 FDD planning.

CP prepares iteration planning meeting by grouping the right amount of relevant features into a work package. Normally the team has a number of CPs, each of them conducts iterations independently. Each iteration does not have to include the whole

development team. Instead, the new iteration team is formed for each iteration. CP selects iteration team members based on availability. Each developer receives a subset of features; strong class ownership is encouraged. The iteration team goes through the list of features, which should be familiar from Process One. If needed, the team contacts the customer to clarify any obscure features. In the complex cases the team may create sequence diagrams. The planning meeting also sets timeline for iteration milestones.

4.1.3 Comparing the differences.

The main difference is that in XP the iteration includes the whole team, while FDD forms the small volatile teams of 3-5 developers for each iteration. This fact may explain better scalability of FDD: since the iteration team is always small, the communication does not become a problem.

XP iteration seems to be better self-regulated. Next iteration includes bug fixes and failed unit tests from the previous iteration. If one of the iterations tried to grab too many user stories and failed to produce quality code/tests, the implementation amount of next iteration will be automatically reduced.

FDD has the two regulating mechanisms:

1) experience of chief programmer;
2) if developer(s) do not complete the task on time, it does not slow down the whole team: other iteration teams will be forming without still-busy developers.

XP does not seem to have the granularity inside of the iteration: the day-by-day activities are the same through the duration of an iteration. FDD on the other hand distinguishes these milestones:

- iteration planning meeting
- design phase
- design review meeting
- coding phase
- code review meeting
- promote to build moment.

FDD allows for better tracking of the iteration, especially if iteration is relatively long (3 weeks). However, frequent meetings add an overhead that is especially visible if the iterations are short (1 week)

4.2 Design and Review.

XP does not have the formal design stage. The design may be performed at the beginning of the iteration and later on as-need basis.

FDD has design stage and design review meeting. During the design stage the developers declare main classes, methods and properties required to implement the given feature. They have to be well documented in the form of code comments, like java-doc. When design is complete, each team member receives a copy of other developer's code printout for review. The team comments on each other's design during the design review meeting. Note: the actual review has to be completed before the meeting to keep it short.

FDD design review meetings allow catching the wrong decisions early in the iteration. This prevents the whole iteration from going astray. The team proceeds with development if design of each team member is accepted. To avoid an overhead of doing large amount of reviews, iteration team must be small: no more than 5 developers including CP. The benefit of FDD is that it produces the sufficient amount of code documentation.

XP does not believe in code documentation. While XP does not have dedicated design phase, the development process is monitored in the daily meetings. Additionally, pair development reduces the risk of wrong design decisions.

4.3 Coding practices.

Both development methodologies pay attention to this stage of development, though the priorities are different. Both XP and FDD stress the importance of coding standards. The standards allow to read each other's code easily. The code that is written according to the standard is less error prone and can be reviewed more efficiently.

Unit tests are the essential part of both methodologies. Unit tests may be written either before the main code is ready (Test Drive Development style) or after. XP insists that unit tests should precede implementation, while FDD does not have any preferences. The essential requirement is to have at least one test per class.

4.3.1 Coding in XP.

Writing code is the main focus of XP. XP strips out many of project activities for the benefits of the coding stage. The most notable difference is that coding is done in pairs. The two developers are sitting in front of the computer, one does the typing, and both do the thinking. At a first glance, it may seem like the waste of time (two people typing may produce more code). The practice shows opposite effect. This may be explained by the following:

- Challenging coding tasks are discussed on the fly (brain-storming works);
- Coding undergoes instant code review;
- The practice allows bringing less experienced developers up to speed quickly;
- People tend to spend less time on non-related activities like reading emails, browsing Internet etc.

XP promotes refactoring. Refactoring is a process of changing code to make it better, without breaking the existing functionality. The resulting code has higher quality.

XP rejects the idea of code ownership. While designated programmers create the initial implementation, anyone can modify the code. The practice of browsing/modifying the code of other developers widens knowledge about the system as a whole. If someone leaves the team, the negative impact of lost knowledge is minimized. There is a natural restriction on collective code ownership: one should never modify the code he/she does not understand. This restriction becomes visible in larger projects.

4.3.2 Coding in FDD.

Coding process in FDD is not as exciting and challenging as it is in XP. This happens because by the coding time the features have been extensively discussed during Process One, iteration kick-off meeting, design review meeting. Classes and methods are

defined by now, their purpose is described in code documentation. Coding often becomes a mechanical process.

Unlike XP FDD strongly discourages refactoring. The main argument against refactoring here is that it takes time and does not bring any value to the customer. The quality of code is addressed during code review meetings.

FDD encourages strong code ownership. The main idea is that every developer knows the owned code and better realizes the consequence of changes. FDD fights the problem of leaving team members from the different angle:

- Sufficient code documentation simplifies understanding somebody else's code;
- Developers know what other people's code does, since they reviewed the design.
- Developers will look at each other's code during code review;

4.4 Code review.

Code review serves several important goals:

- To find errors in code: code review is known to catch as many errors as unit testing, especially in the areas where unit testing is difficult, like User Interface code.
- To enforce coding standards.
- To bring less experienced developers up to speed, and share the best programming practices.
- To familiarize developers with each other's code.

While doing XP development during implementation of BRE-275 project, we did not perform code review. The review is supposed to be done on the fly, while working in pairs. The academic environment makes pair programming difficult or impossible.

FDD has a formal meeting, called "Code Review Meeting" in its iteration schedule. Every developer of the iteration team prints out his/her code and distributes it

among the team members. Actual code review happens before the meeting. The meeting itself is used to discuss found problems and figure out the best ways to solve them.

Both development methodologies have their strong and weak sides regarding code review:

- Pair programming in XP naturally enforces code review. It does not matter how much code is written – it all will be reviewed.
- FDD exposes the code to more eyes. The practice shows that quite often only one member of the iteration team catches some particular bug. The problem in the code may stay unnoticed if only two people are looking at the code.
- Code review in FDD is separated from coding process. If the iterations are long (3 weeks) and iteration team is 5 or more developers then the code review process becomes less efficient. During a long iteration the team members may create hundreds of pages of code. It is hard to review such amount of code during single code review: the process becomes long, people loose concentration and bugs may go unnoticed.

Chapter 5 Deployment

In a waterfall style of development, the deployment stage is usually the last stage of project. It is performed when all code is complete and tested. The deployment phase may not depend on the development methodology: quite often the big corporation have completely isolated environments for development, testing and production.

FDD and XP both include the deployment into their iteration cycle. It may mot be the real deployment into production environment because of the reasons stated above. However the application should be up and running, and available for the customer. The customer plays with the implemented features and provides the feedback.

Sometimes the regulated corporate environment is hostile to the spirit of continues deployment. For example, the US government regulates the healthcare industry. The regulations require the formal validation step to be performed before the deployment is possible. The purpose of this validation process is to ensure that all the required functionality is implemented according to the user requirements. The validation team will not start validation until the end of the project when all the features are implemented. The intermediate releases cannot be validated, and cannot be used against production data. This may discourage the customer from looking at the partially implemented system.

XP requires that the customer representatives should become the team members. It is one of the representative's responsibilities to play with the system in progress. While FDD requires less intensive customer's evolvement, it should follow FDD spirit to ensure continues customer's feedback.

Chapter 6 Tracking the progress

Tracking the progress of software projects is important. Only small projects or in-house development may not require any progress indicators. The projects of larger scale, or those done for the external sponsor must be tracked. The coarse-grained progress tracking is a natural feature of waterfall development. The common 5-6 stages of the project are given their percentages, so project managers can easily see project's status and report it to the upper management. The new development methodologies should offer progress-tracking capabilities that are not worse than existing ones.

6.1 Progress tracking in XP

XP does not offer much in the area of progress tracking. XP only ensures that the progress is visible. Every new iteration adds the functionality that is immediately available for customer's acceptance test. However, it may be hard to answer the simple question: "When will the project be complete?" One of the ways to estimate it is to compare the number of implemented and pending user stories. This is not a precise answer though: implementation of one user story and communication with customer may lead to a new user story being created.

eXtreme Programming is extremely agile. It is best suited for the projects where user requirements are not well defined or change often. In such circumstances the question of project completion date may become meaningless, so no one bothers to answer.

6.2 Progress Tracking in FDD.

FDD offers precise progress tracking. It is based on the two facts:

- total number of features is known after the "Process One" is complete;
- each iteration has a defined lifecycle with percentages given to each step. The following percentages are found on Nebulon Ltd web site [6]

Domain Walkthrough	Design	Design Inspection	Code	Code Inspection	Promote to Build
1%	40%	3%	45%	10%	1%

Figure 6.1.1 Cost of milestones in FDD iteration.

These facts allow calculation of project completion date, as well as the creation of tracking reports. The examples of the reports are provided in the Appendix 2. These reports are expensive to create manually. FDD projects rely on project management tools to generate the reports.

FDD also addresses the question of project completion date. Jeff De Luca mentioned that FDD may accommodate up to 10% change in project requirements without slipping the deadline. The idea here is that the changing requirements for the software system do not mean that the business itself has changed. Therefore the problem domain should still represent the valid picture of business. The new requirements should not affect the majority of already implemented features.

Chapter 7 Summary and Conclusions

This section summarizes the applicability statements made in the different sections of this paper. My participation in both XP- and FDD- managed projects leads me to the following conclusions:

- Both methodologies successfully utilize iterative development to avoid main weakness of waterfall when errors are discovered at later stage;
- Both methodologies are lightweight, but XP is lighter: it does not produce code documentation and keeps code review informal.
- XP is better suited for the projects with frequently changing or poorly defined user requirements.
- FDD is not very good at shooting the moving target. However, when user requirements are rather stable, it offers higher success repeatability than waterfall.
- FDD scales better. It has a hierarchy within the development team that allows keeping iteration teams small even when the project team is big.
- As a part of the methodology FDD offers progress tracking and reporting capabilities. This comforts managers and makes it more attractive for big companies.
- FDD project might be managed manually, but the project management tools reduce the overhead of keeping the database of features and generating the reports. XP does not seem to require any special tools.
- FDD is better prepared to work with team where developers' experience varies. The most experience/productive members become chief programmers. However, in the small team of equally strong developers some recourses may

be left under-utilized: author observed the developers be idle while CP prepares list of features for the next iteration.

- Both methodologies require discipline and do not replace the need for good manager.
- Both methodologies offer some extremely useful techniques that may be applied in any development methodologies, even in waterfall. The two of my favorites are unit testing and code review.

References

1. Feature Driven Development web site:
   http://www.featuredrivendevelopment.com
2. Extreme Programming web site:
   http://www.extremeprogramming.org
3. CSCIE-275 Lecture materials:
   http://www.people.fas.harvard.edu/~robinson/cscie275/materials/lectures/L04-20040301.pdf
4. Java Modeling In Color With UML : Enterprise Components and Process
   By Peter Coad, Jeff de Luca, Eric Lefebvre, 1999
5. Extreme Programming Explained: Embrace Change
   by Kent Beck, Addison-Wesley, 1999.
6. Web site of Nebulon Ltd.: consulting practice of Jeff De Luca:
   http://www.nebulon.com/fdd/index.html

Appendix 1. Introduction to UML in color.

Colored UML is a regular UML with color-encoded classes. The classes are divided into 4 different categories, each category has its own color. The auxiliary classes and interfaces are colorless. A fragment of color UML is presented on figure A1.1

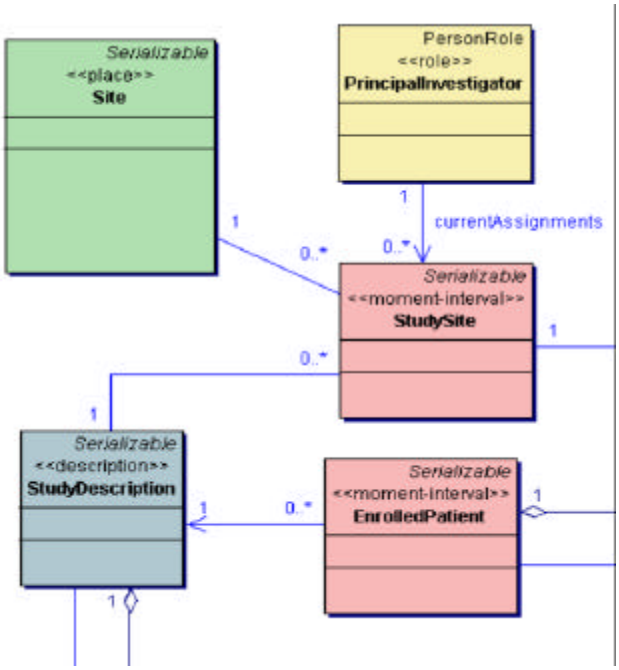

Figure A1.1: Fragment of Color UML

Yellow: a role being played, usually by a person or an organization. For example, the user of online auction site may play different roles: it can be a buyer, a seller or a system administrator.

Blue: a catalog-like description. For example, an online store may have description of the CD player that it sells. The description describes all the characteristics of the player, but it is not the player by itself.

Green: a party, place or thing. In the previous example that actual player in stock would be modeled as green. The green class usually has some identifying attributes, like serial number, person's name etc.

Pink: a moment in time or an interval of time usually associated with some business process. For example, the fact of purchase may be registered with pink class, since it has a time of sale, which is tracked by online store.

Color allows to quickly understand problem domain's dynamics. The chain of pink classes represents the main flow of business. It is usually surrounded by the blue descriptions and the green "things". The yellow roles are normally added to the design at the beginning. They are often removed later, if a person is playing only one role.

Figure 1.1 displays part of the problem domain used in healthcare industry for conducting clinical trials. The purpose of clinical trial is to test the new drug. Clinical trials are conducted at a site (hospital or doctor's office). The site is green. Each site has a principal investigator that conducts the trial. The investigator is a role therefore it is yellow. The details of the study (schedule of visits, drugs given etc.) are described by *StudyDescription*, which is blue. Hospital's participation in the study is recorded in *StudySite* class. It has start- and end- dates, so it is represented by pink interval. Patients enroll into the study at a site. Patient has a date of enrollment and date of completion, so it is pink.

Appendix 2. FDD progress reports.

A2.1 Overall progress report.

This report shows the progress of the whole project. The related features are combined into feature sets. The report shows one block per feature set:

- Green body color: features are complete;
- Blue body color: work in progress;
- Red body color: behind schedule;
- White body color: not started;
- Letters above the right corner: initials of chief programmer;
- Number in parenthesis: number of features in feature set;
- Percent: the percent of completed features;
- Month in the bottom part: the month when the feature set is scheduled for development;

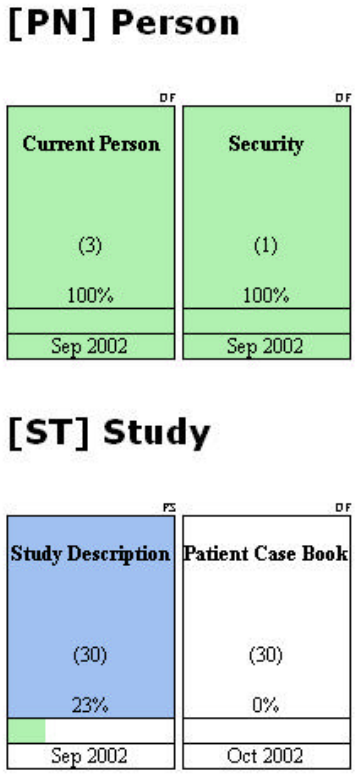

Figure A2.1.1 FDD Overall progress report.

A2.2 Work package progress report

Submit Form (21)													M AL
ID	Description	Walkthrough		Design		Design Inspection		Development		Code Inspection		Promote to Build	
		Plan	Actual	Plan	Actual	Plan	Actual	Plan	Actual	Plan	Actual	Plan	Actual
FM011	create a new PatientVisit of a atientVisitDescription for a PatientAspect	7/17/02	7/17/02	7/19/02	7/23/02	7/22/02	7/24/02	7/25/02	7/25/02	7/25/02	7/29/02	7/26/02	7/31/02
FM012	create a to-be-submitted Form of a FormDescription for a PatientVisit	7/17/02	7/17/02	7/19/02	7/23/02	7/22/02	7/24/02	7/25/02	7/25/02	7/25/02	7/29/02	7/26/02	7/31/02
FM013	determine if a Form is scheduled for a PatientVisit	7/17/02	7/17/02	7/19/02	7/23/02	7/22/02	7/24/02	7/25/02	7/25/02	7/25/02	7/29/02	7/26/02	7/31/02
FM014	duplicate for modification a to-be-submitted Form from a received Form	7/17/02	7/17/02	7/19/02	7/23/02	7/22/02	7/24/02	7/25/02	7/25/02	7/25/02	7/29/02	7/26/02	7/31/02
FM015	add a Section to a to-be-submitted Form	7/17/02	7/17/02	7/19/02	7/23/02	7/22/02	7/24/02	7/25/02	7/25/02	7/25/02	7/29/02	7/26/02	7/31/02
FM016	remove a Section from a to-be-submitted Form	7/17/02	7/17/02	7/19/02	7/23/02	7/22/02	7/24/02	7/25/02	7/25/02	7/25/02	7/29/02	7/26/02	7/31/02
FM017	add a QuestionGroup to a Section	7/17/02	7/17/02	7/19/02	7/23/02	7/22/02	7/24/02	7/25/02	7/25/02	7/25/02	7/29/02	7/26/02	7/31/02
FM022	determine the Form from which to copy continuing data for a Form	7/17/02	7/17/02	7/19/02	7/23/02	7/22/02	7/24/02	7/25/02	7/25/02	7/25/02	7/29/02	7/26/02	8/1/02
		unit test not completed yet (AS)											
FM023	determine a Form for copying continuing data for a PatientVisit	7/17/02	7/17/02	7/19/02	7/23/02	7/22/02	7/24/02	7/25/02	7/25/02	7/25/02	7/29/02	7/26/02	8/1/02
		unit test not completed yet (AS)											
FM030	perform univariate checks for a Response	7/17/02	7/17/02	7/19/02	7/23/02	7/22/02	7/24/02	7/25/02	7/25/02	7/25/02	7/29/02	7/26/02	7/31/02
FM031	submit to external system a Form	7/17/02	7/17/02	7/19/02	7/23/02	7/22/02	7/24/02	7/25/02	7/25/02	7/25/02	7/29/02	7/26/02	7/31/02

Expected completion date : Jul 2002

Figure A2.2.1 FDD Work package progress report.

This progress report shows the progress of each feature by comparing planned and actual dates for iteration's milestones. The planned dates are set at iteration kick-off meeting. Green color denotes features that are on schedule. Red color – behind schedule.