A Tool for Supporting Feature-Driven Development

Marek Rychlý and Pavlína Tichá

Department of Information Systems,
Faculty of Information Technology, Brno University of Technology,
Božetěchova 2, 612 66 Brno, Czech Republic
rychly@fit.vutbr.cz, xticha05@stud.fit.vutbr.cz

Abstract. This paper deals with the Featured Driven Development (FDD), an agile software development method. According to the requirement analysis for the FDD method application, an information system has been created providing all team members with instruments to follow the method. This tool has been implemented as a multi-user web-based application enabling creation of feature lists, planning a project, supporting cooperation among members of a feature-team, and tracking project progress in an illustrative way. To support project management and communication with customer representatives, a wide range of reporting features has been provided.

Keywords: Agile software development, Feature driven development, Feature, Feature-team, Class ownership.

1 Introduction

Traditional software development suffers from slow interaction between the development process and evolving user requirements. It follows from application of traditional software development methods to projects with rapidly changing business requirements. In this context, approaches to software development can be broadly divided into two groups. At one extreme, there are *classical software development methods* where user requirements are obtained in the first phases of the development process and each one of the later phases follows an earlier one (e.g. "the waterfall" model). At the other extreme, there are *agile software development methods* [1], which embrace and promote evolution of user requirements throughout the entire development process (e.g. "eXtreme Programming"). The first extreme produces precisely designed and documented software systems, but those often do not match current user requirements, while the second extreme produces software systems matching the latest user requirements, but often with an inconsistent design and poor documentation.

The *Feature Driven Development* (FDD) [2] is an iterative and incremental software development process. Although the FDD method is one of agile software development methods, it is built around the traditional industry-recognised

B. Meyer, J.R. Nawrocki, and B. Walter (Eds.): CEE-SET 2007, LNCS 5082, pp. 196–207, 2008.

practices derived from software engineering, including planning, design and documentation phases with fine-grained decomposition of a system's functionality and developers' responsibilities, accurate progress reporting, frequent verification, etc. The application of the FDD method leads to better project management and consistency of a software's design, implementation and documentation.

The paper describes requirement analysis for a software supporting the FDD method [3]. The tool is designed and implemented as an information system providing all team members with instruments to follow the FDD method on real software projects run in a middle-sized software development company. An important feature of the tool is ability of tracking changes in user requirements and map them into modifications in classes and into team members responsible for implementing the changes. Using the feature contributes to an increase in safety of development process.

The remainder of the paper is organised as follows. In Section 2, we introduce the FDD process in more detail. In Section 3, we analyse the FDD process and describe requirements concerning the supporting tool. In Section 4 and Section 5, we describe the design and (briefly) implementation of the tool. In Section 6, we review main approaches that are relevant to the subject and discuss advantages and disadvantages of our system compared with the reviewed approaches. Finally, in Section 7, we summarise our approach, current results and outline the future work.

2 Feature Driven Development (FDD)

The *Feature Driven Development* (FDD) has been published in 1999 [4], after its successful application at an international bank in Singapore in 1997. The FDD is a highly iterative and collaborative agile development method that is composed of *five processes* (see Figure 1). The processes are formally described using the traditional ETVX-based (Entry-Task-Verify-eXit) process descriptions [2]. Informally, the processes can be described as follows:

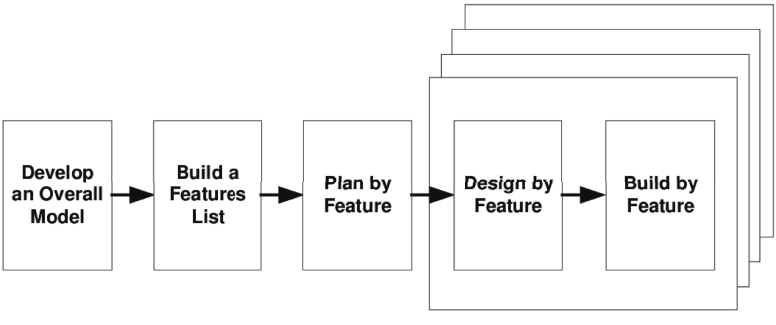

Fig. 1. The five processes of the FDD method (form [2])

Develop an Overall Model – In collaboration with domain experts and developers, an overall *domain object model* is created gradually in series of "walkthroughs" of a software system's scope and context for each area of the problem domain. It captures the key abstractions and their relationships in the system.

Build a Features List – According the initial domain object model, a *list of features* is created where each feature describes an object or its method in the domain model. A "feature" is defined as a small, deliverable client-valued piece of a system's functionality, which can be implemented in no more than 2 weeks. The features are grouped into *feature-sets*, which represent business processes or work-flows, and the feature-sets are grouped into *subject-areas* (or *subject-domains*), which represent business functions or business domains implementing core capabilities of the system. There are recommended *formats for descriptions* of the features, feature-lists and subject-areas, which facilitate its mapping into objects and methods [4]. The recommended format of the descriptions is

- for a feature: *action* the *result* (by, for, of, to, ...) a(n) *object* [(of, for, with, ...) *parameters*], e.g. "verify the password for an user with the login",
- for a feature-list: *action* (for, of, ...) a(n) *object*, e.g. "authentication of an user",
- and for a subject-area: *object* management, e.g. "user management".

Plan By Feature – The feature-sets and features are analysed and their time intensities are estimated. The feature-sets are sorted according to priorities assigned by a customer's representatives, estimated time intensities and technical dependencies. Then they are assigned to individual ad-hoc *feature-teams*. Inside the feature-teams, classes from the domain model are assigned to individual developers. Each *developer is responsible* for creation and maintenance of his classes. The developer is a member of all feature-teams, which have assigned the features related to the developer's classes.

Design By Feature and Build By Feature – Those two processes are iterated for each work package, i.e. for a small group of features from one feature-set. The work package is processed by one feature-team and it is a unit of integration with other feature-sets and feature-teams. Members of the feature-team collaborate to create sequence diagrams and another useful design models for their features, and design interfaces and declarations for corresponding classes and methods. After that, individual developers implement and integrate their classes or parts of the classes into a system.

The FDD processes uses software engineering "best practices" such as domain object modelling (the model is a primary representation of knowledge), developing by feature (iterative and incremental), individual class ownership, ad-hoc set up teams of developers, inspection and reviews (of an overall model, feature-lists, design models and a code), regular builds and verification by a customer's representatives, progress reporting, etc.

2.1 Roles and Responsibilities in the FDD Processes

The FDD method defines more roles than many of other agile methods [1]. The roles can be classified into three categories: key roles, supporting roles and additional roles [2]. One team member can act as multiple roles, and a single role can be shared by several team members.

The six *key roles* in a FDD project are: project manager (the leader of a project), chief architect (overall design of a system), development manager (the coordinator of teams), chief programmer (the leader of a feature-team preparing work packages), class owner (designer, coder, tester and documentarist of its classes) and domain experts (detailed knowledge of user requirements and problem domains).

The five *supporting roles* comprise release manager (controls progress of the process), language lawyer/language guru (has detail knowledge of programming language or technologies), build engineer (responsible for build process and version management), tool-smith and system administrator (technical support of a project).

The three *further roles* that are needed in some projects are: testers (verify that a system fulfils requirements), deployers (maintain data compatibility and prepare new releases) and technical writers (prepare user documentation).

3 Software Support for the FDD Method

To *support of the FDD method*, we need to track the five processes, which are described in Section 2, from points of view of the roles that are listed in Section 2.1. A software support for the FDD method should control application of the FDD processes to real software projects.

The *first FDD process* is focused on the developing of an overall domain model. According to the original presentation of the FDD method [4], it is recommended to use UML visual models [5] with coloured objects. The initial domain model provides a basis for feature-lists, that will be created in the second FDD process. However, in the most projects, many features are arising during the later processes, especially in the processes "design by feature" and "build by feature" as modifications and extensions of the existing features. This issue forbids description of the domain model in details during the first process and requires modifications of the model in later processes. The software support of the FDD method should allow modification of domain models in relation to feature management.

To support the *second FDD process*, which is aimed at building of feature-lists, a software support for the FDD method needs to keep track of all features in a project and their grouping to feature-sets and subject-areas. Generally, lists of features can be created in two ways: top-down and bottom-up, i.e. as decomposition of subject-areas to feature-sets and then to single features and as composition of features into feature-sets and feature-sets into subject-areas, respectively. The software has to support both ways.

In the *third FDD process*, a timetable for feature-sets and features is created. For each feature, it includes the time of its start and its estimated duration. There are two requirements contrary to each other: keep the start-time and duration of a whole project and permit individual planing of features in scope of feature-teams. To balance those requirements, we split the whole project into two-weeks intervals, which are appropriate to maximal time intensities of features [2]. Then, the feature-teams can plan their features within the assigned two-weeks intervals independently and the time-requirements of the whole project are complied. The supporting system should allow to assign features to intervals and their detailed planing, including control of their dependencies and balancing of a workload.

The *last two FDD processes* are focused on designing and building of individual features in compliance with the principle of feature-ownership. Owner of a feature creates a list of activities leading to implementation of the feature. Each activity can require modification of a class or its part related to the feature. The modification will be done solely by the owner of the class (a developer). This implies also ownership of a part of a project's source code. The software support of the FDD method should track those relationships between features, developers and parts of classes.

Finally, a tool for supporting the FDD method has to provide a set of *visual reports*. The reports should continuously show progress for individual features, feature-sets, subject-areas and a whole project, as well as a proper form of workload overviews for individual developers and feature-teams. This information is important for correct planing decisions.

To summarise, the basic requirements to software support of the FDD method according to its five processes are the following: a support of an initial domain model, tracking of its modification and connections of its parts to individual features; a support of features, feature-sets and subject-areas, and their composition and decomposition (top-down and bottom-up approaches); a support of individual planing of features in scope of feature-teams without substantial impact on duration of a whole project; a support of the principle of feature-ownership and class-ownership and collaboration of the owners of features and classes; and a support of wide range of visual reports required for planing decisions.

4 Design of the Tool for Supporting FDD

The Figure 2 shows the basic user roles, which cover important roles from Section 2.1 according to the requirements for a tool for supporting the FDD method (an information system) mentioned in Section 3. Besides those requirements, we define "a domain" as an independent group of projects supported by the tool, e.g. projects of one software developer company using the tool, if it is provided as an outsourced service.

The *SystemAdministrator* maintains technical issues of the whole information system (the same meaning as in Section 2.1), e.g. controls user rights and domains, while the *DomainAdministrator* maintains technical issues of a domain specific part of the system, e.g. controls users in the domain.The *ProjectAdministrator* is

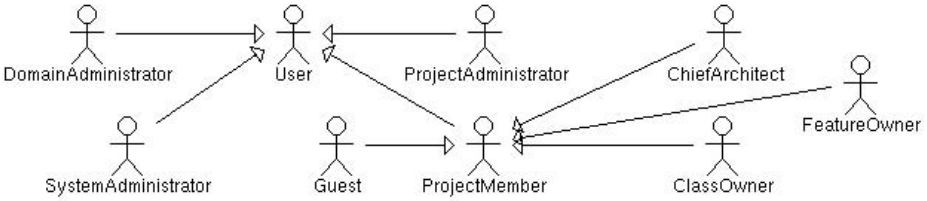

Fig. 2. The hierarchy of the well-established user roles

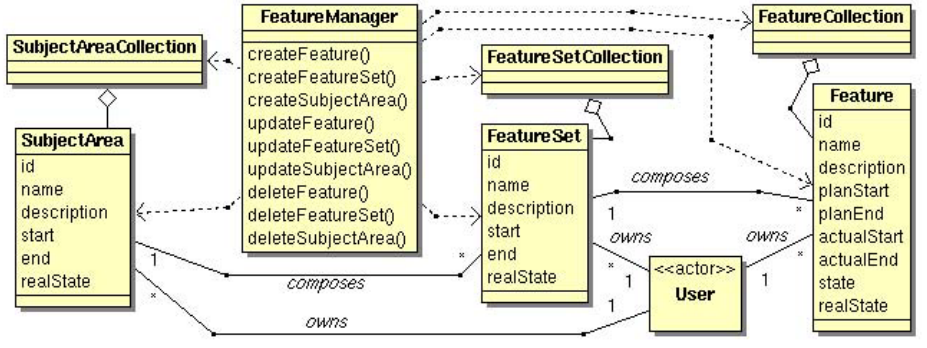

Fig. 3. The class diagram of features and feature-lists

able to create, modify and delete a project, assign and withdraw users and their roles in the project, and obtain reports relevant to the project and its parts. The *ChiefArchitect* can manage a project's domain model and features, feature-sets and subject-areas, and make planing decisions, i.e. assign individual features into two-weeks intervals (see the requirements of the third FDD process in Section 3). The *FeatureOwner* maintains a feature-team, i.e. controls activities leading to implementation of the feature, assigns classes that are modified by the activities and class-owners responsible for realisation of individual activities, watches progress of the activities and verifies finished activities. The *ClassOwner* represents a developer, who implements a part of assigned feature (an individual activity connected to the owned class) and is able to modify information about the progress of the activity. The last role, the *Guest*, represent external supervisor of a project, a customer's representative, who is able to view reports about progress of the project.

In the Figure 3, there is a part of class diagram related to features and feature-lists. The instance of the class `FeatureManager` handles for a project a collection of features (classes `FeatureCollection` and `Feature`) composed into feature-sets (classes `FeatureSetCollection` and `FeatureSet`) composed into subject-areas (classes `SubjectAreaCollection` and `SubjectArea`). Those features, feature-sets and subject-areas have attributes representing scheduled numbers of starting and ending two-weeks intervals. Moreover, the features have attributes indicating actual starting and ending two-weeks intervals and their current

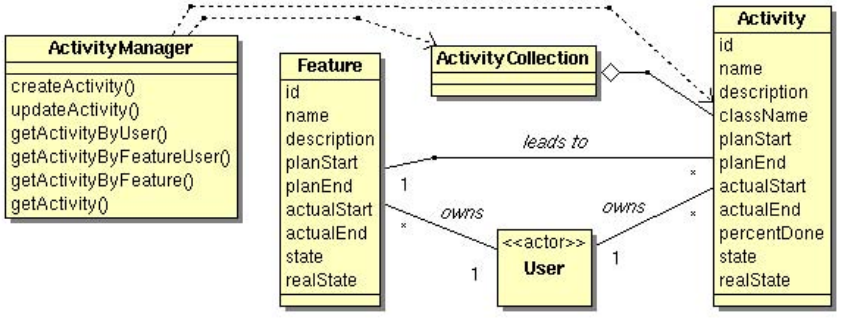

Fig. 4. The class diagram of features and activities

states[1]. Features, feature-sets and subject-areas are connected to users (instances of class `User` with stereotype `Actor`), that act as their owners.

The class diagram in the Figure 4 shows relationship of features and activities and their context. The instance of the class `ActivityManager` handles for a project a collection of activities (classes `ActivityCollection` and `Activity`). Each activity contains attributes that represent its scheduled and actual numbers of starting and ending two-weeks intervals, the name of a class, which is modified by the activity, the progress in percents and current state[1]. Activities are connected to features (instances of class `Feature`) and users (instances of class `User` with stereotype `Actor`), that act as owners of the modified classes during the activities and are responsible for realisation of the activities.

The overall entity-relationship diagram of the system with basic entities is presented in the Figure 5. The diagram connects entities for features and activities to auxiliary entities for domain management (entities `Domain` and `Project`) and user management (entities `User`, `Right`, `Role` and associative entity `UserRoleInProject`). The entities for features and activities (entities `Feature`, `FeatureSet`, `SubjectArea` and `Activity`) have been described before, as the relevant classes in the class diagrams in Figure 3 and Figure 4.

5 Implementation and Practical Results

The system has been implemented in the framework *ASP.NET 2.0* and coded in the C# language as a web-based application for the *Microsoft Internet Information Services* web-server and the relational database management system *Microsoft SQL Server* as a data storage back-end. The external database is accessed via *ADO.NET* and own data abstraction layer, which maps relational data to proper objects and vice versa (see `Manager-` classes in Figures 3 and 4).

The data abstraction layer providing objects based on the relational data also generates some *dynamic attributes* of those objects. A good example is the `realState` attribute of classes `Activity`, `Feature`, `FeatureSet`, `SubjectArea`

[1] The states are: “not started”, “in progress”, “attention” and “completed”.

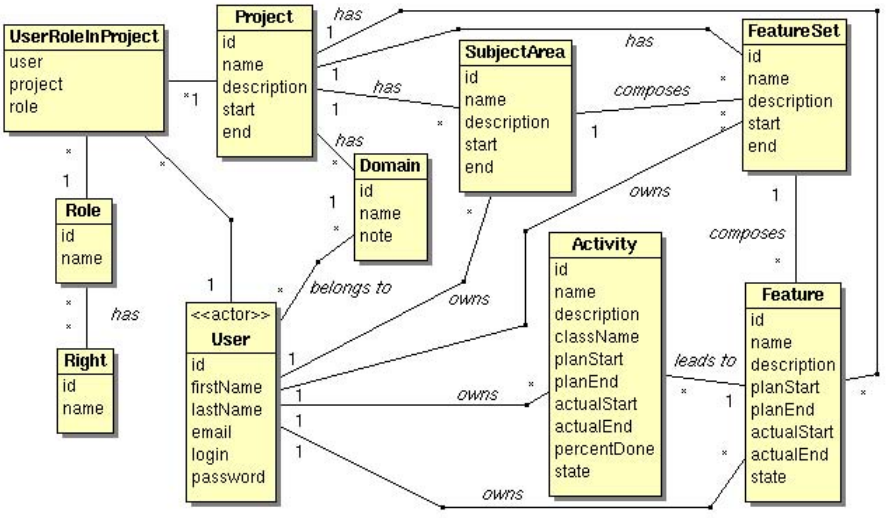

Fig. 5. The entity-relationship diagram with basic entities (the UML notation)

and `Project`. The attribute represents the actual state of an activity, one or more features or a whole project, and can assume values "not started", "in progress", "attention" and "completed", i.e. the same values as `state` attributes. For instances of classes `Activity` and `Feature`, the attribute is computed form attributes `state`, `-Start`/`-End` attributes and the actual two-week interval. The attribute `realState` has value "attention" if the number of the actual two-week interval is greater than `planStart` and `state` is not "in progress" or "completed", or it is greater than `planEnd` and `state` is not "completed", otherwise the attribute `realState` reflects attribute `state`.

Attributes `realState` of instances of classes `FeatureSet`, `SubjectArea` and `Project` are derived in the hierarchy of those objects in the bottom-up direction. For object A (e.g. `FeatureSet`), which composes from objects $B_1, \ldots, B_n$ (e.g. `Feature`-s), attribute `realState` of A has value "attention" or "in progress" if at least one of attributes `realState` of $B_1, \ldots, B_n$ has value "attention" or "in progress"[2], respectively, otherwise the attribute of A reflects values of the attributes of $B_1, \ldots, B_n$.

The result of such "automatic state analysis" of a project's parts can be reported by the system as the project park diagram[3] (see Figure 6). Moreover, values of the `realState` attributes of objects in time can be aggregated through a whole project into the project development roadmap. The roadmap indicates a development plan and real states of completed features in percentages on Y-axis and in time on X-axis of the graph (see Figure 7).

[2] If $B_1, \ldots, B_n$ are "completed" but at least one "not started", A is also "in progress".

[3] The values of progress bars in the boxes of the project park diagram representing `Feature`-s are computed as arithmetical averages of `percentDone` attributes of `Activity`-ies, which are grouped in the boxes at this level.

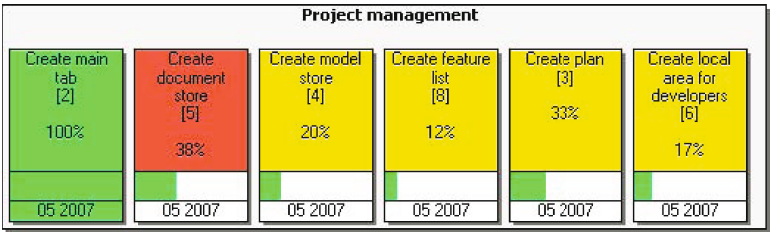

Fig. 6. The project park diagram example where colours of the boxes represent `realState` attributes of a project's parts (a part of real report from the presented tool)

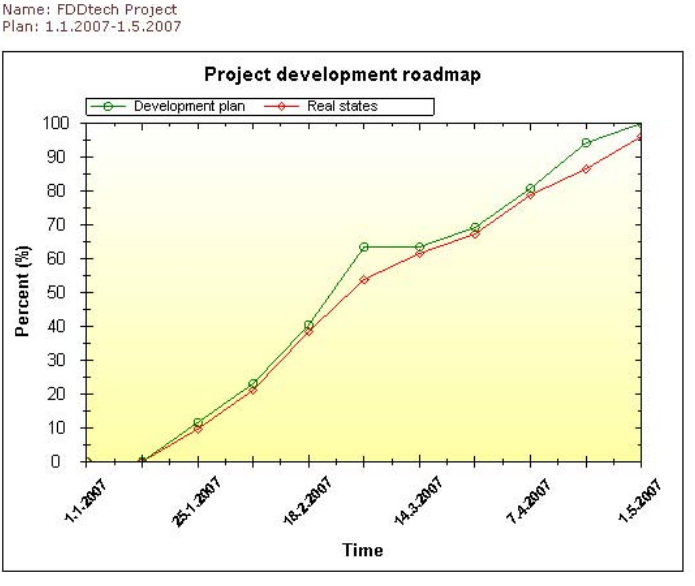

Fig. 7. The project development roadmap (a part of real report from the tool)

The system can export also other reports (e.g. summary progress and trend reports). All reports are available in HTML, PDF and RTF formats, suitable for project managers as well as a customer's representatives.

The source code of the described tool for supporting the FDD method is licensed under the GNU General Public License (GNU GPL) and will be available as an open source project[4].

6 Discussion and Related Work

In this section, we briefly review four projects that support the FDD method of software development and compare them with our approach. Table 1 compares

[4] See `http://www.fit.vutbr.cz/homes/rychly/fdd-tech/`

Table 1. The comparison of the tools that support the FDD method of software development (notes: *the names marked by a star are open source projects)

Name	Type	Users	Feature (de)composition	Progress reporting
*FDD Tools Project**	desktop	single user, projects	features, feature-sets, subject-areas (top-down)	project park diagram (interactive)
FDD Tracker	desktops (shared DB)	roles, projects, domains	work-packages, features, feature-sets, subject-areas (top-down)	project park diagram, plan view report, progress summary report, project dev. roadmap (overall and weekly), defect graph
Cognizant FDD	Visual Studio TF Server RTM	roles, projects	modifications of components, features, feature-sets, subject-areas (top-down and bottom-up)	progress report, defect report, prioritised feature list, component ownership matrix
*FDDPMA**	web-based	roles, projects	work-packages, features, feature-sets, subject-areas (top-down)	project park diagram, progress summary report, project development roadmap, plan view report, feature completion trend
*Our tool**	web-based	roles, projects, domains	work-packages, activities (modifications of classes), features, feature-sets, subject-areas (top-down and bottom-up)	project park diagram, progress summary report, project development roadmap, trend report

the basic features of the reviewed projects[5], which are described in more detail bellow.

The *FDD Tools Project* [6] is an open-source Java-based desktop application providing only basic support for the FDD method. It provides one progress report for features (a project park diagram) and there are no means of decomposition of the features into partial tasks. The project seems not to be actively developed anymore.

The *FDD Tracker* [7] is a commercial desktop application executable on Microsoft Windows NT-based operating systems. It provides complex multi-user multi-domain support of the FDD method including support of different roles and views, support of intervals of variable length, inspection management, defect tracking and reporting, etc. The FDD Tracker does not support web-based user interface, does not allow bottom-up composition of features into feature-lists and does not connect parts of features to parts of a source code.

[5] The names of progress reports are not standardised and each project uses its own terminology. In the table, we rename some of the progress reports according their formats and provided information in order to allow direct comparison of the projects.

The *Cognizant Feature-Driven Development* [8] is a commercial tool for supporting the FDD method, which is integrated in Microsoft Visual Studio Team Foundation Server RTM. It allows defining of a software project in Visual Studio as a collection of individual features with connection to the project's source code and tracking of its defects. Cognizant FDD extends five processes of the FDD method with a new process "certify by feature", which follows the fifth FDD process "build by feature" in each iteration, and a new iterative process "release" closing the whole development process [9]. Drawback of Cognizant FDD can be its tight integration with Visual Studio, which prevents managers (non-developers) and a customer's representatives from interaction with products of the FDD method.

The *FDD Project Management Application* (FDDPMA) [10,11] is open-source Java-based application and the only web-based tool among reviewed projects. Unfortunately, the FDDPMA does not allow bottom-up composition of features into feature-lists and does not support relationships between features and classes or their parts (i.e. no class ownership).

In comparison of our tool with the reviewed projects, we can find many similar features that are recommended by the FDD method description (see Section 2). In addition to that, our tool supports top-down and bottom-up approaches to creation of feature-lists, provides decomposition of features into activities, which represent tasks needed to accomplish a feature, connection of the activities and relevant classes or their parts that must be implemented by developers to complete an activity, and detailed hierarchical tracking of the state and progress of features from developers (i.e. activities) to managers (i.e. feature-sets and a project). Usage of the web-based user interface allows easy remote access to project data, especially for domain experts and a customer's representatives that should participate in a project. Drawback of our system can be isolation from other development tools and absence of advanced FDD tools such as tracking of features verification process and defects or support for feature-teams collaboration.

7 Conclusion and Future Work

The paper describes requirements analysis, design, and implementation of a tool supporting the FDD method. The described system covers the five FDD processes, features and feature-lists management, support for decomposition of the features to activities connected to individual classes or their parts, support for feature-owners and class-owners, control of progress at different levels of hierarchy, user management with well-established and user-defined roles, and support of various reports (the park diagram, the project development roadmap, summary progress and trend reports). Some of those features (e.g. support for activities) are novel and have not been used yet.

Incremental development of a software system with strictly defined features and related modifications of parts of classes (i.e. mapping of user requirements into modifications of a source code) allows better tracking of impact of individual

increments on quality of a whole software system and contributes to an increase in safety of development process.

Future work is mainly related to integration of human-resource management into the tool (e.g. appointment of feature-team members according to their past experiences and current workload) and support for source code management (i.e. more detailed tracking of source code modifications caused by a feature realisation).

Acknowledgement. This research has been supported by the Research Plan No. MSM 0021630528 "Security-Oriented Research in Information Technology".

References

1. Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J.: Agile software development methods: Review and analysis. VTT Publications 478, Espoo, Finland: Technical Research Centre of Finland (2002)
2. Palmer, S.R., Felsing, M.: A Practical Guide to Feature-Driven Development. Prentice Hall PTR, Upper Saddle River (2002)
3. Tichá, P.: Information system supporting Feature Driven Development. Master's thesis, Brno University of Technology, Faculty of Information Technology, Department of Information Systems (May 2007)
4. Coad, P., Lefebvre, E., Luca, J.D.: Feature-Driven Development. In: Java Modeling in Color with UML: Enterprise Components and Process, ch. 6, Prentice Hall PTR, Upper Saddle River (1999)
5. OMG: UML superstructure specification, version 2.0. Document formal/05-07-04, The Object Management Group (August 2005) Also available as ISO/IEC 19501:2005 standard
6. SourceForge.net: FDD tools project (September 2006), `http://fddtools.sf.net/`
7. IT Project Services: FDDTracker (2007), `http://www.fddtracker.com/`
8. Cognizant Technology Solutions: Cognizant Feature-Driven Development (2007), `http://www.cognizant.com/html/content/microsoft/techfddvsts.asp`
9. Cognizant Technology Solutions: Implementing Cognizant Feature-Driven Development using Microsoft Visual Studio Team System. Technology white-paper, Cognizant.NET Center of Excellence (2005)
10. FDDPMA Development: FDD project management application (2007), `http://www.fddpma.net/`
11. Khramthchenko, S.: A project management application for Feature Driven Development (FDDPMA). Master's thesis, Harvard University (June 2005)