Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Technologies

Diploma Thesis

Version control systems in software engineering

Bc. Yevhenii Samoilov

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Yevhenii Samoilov

Informatics

Thesis title

Version control systems in software engineering

Objectives of thesis

The main objective of the thesis is to perform an analysis of the existing version control systems and their role in the software development process.

Partial goals are such as:
- to create general overview of the current version control system solutions, their advantages and disadvantages
- to analyze usage of Microsoft Team Foundation Server as a source control management tool
- to evaluate the version control systems from a perspective of corporate software development

Methodology

Methodology of the thesis is based on analysis of information resources and usage of the found solutions in the simulated software development examples. Some of the examples will include real life situations from software development cycles. Development and management experiences will be reflected and evaluated. Final conclusion and recommendations will be given based on the practical part outcomes as well as theoretical analysis of the found solutions.

The proposed extent of the thesis

60 – 80 pages

Keywords

Source code management, version control, team development, software engineering, open-source, Integrated Development Environment, Team Foundation Server

---

Recommended information sources

Arora, Tarun. Microsoft Team Foundation Server 2015 Cookbook. Washington : Packt Publishing, 2016. 978-1-78439-696-1.

Microsoft. Microsoft API and reference catalog. MSDN. [Online] 01 01, 2017 . [Cited: 02 16, 2017.] https://msdn.microsoft.com/en-us/library/ms123401.aspx.

Schildt, Herbert. C# 4.0: The complete reference. s.l. : McGraw-Hil, 2010. 978-0-07-174117-0.

Sink, Eric. Version Control by Example. Champaign, Illinois : Pyrenean Gold Press, 2011. 978-0-9835079-1-8.

Wright, David. Software Life Cycle Management Standards. s.l. : IT Governance Ltd, 2011. 9781849282062.

Expected date of thesis defence

2017/18 SS – FEM

The Diploma Thesis Supervisor

Ing. Miloš Ulman, Ph.D.

Supervising department

Department of Information Technologies

Electronic approval: 31. 10. 2017

Ing. Jiří Vaněk, Ph.D.

Head of department

Electronic approval: 1. 11. 2017

Ing. Martin Pelikán, Ph.D.

Dean

Prague on 29. 03. 2018

Declaration

I declare that I have worked on my diploma thesis titled "Version control systems in software engineering" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the diploma thesis, I declare that the thesis does not break copyrights of any third person.

In Prague on 29/03/2018 Bc. Yevhenii Samoilov _____

Acknowledgement

I would like to thank my thesis supervisor Ing. Miloš Ulman, for his guidance during the research. His advices and remarks, as well as patience and support were of incredible help. Dr. Ulman was always available for feedback on my ideas, whenever I needed a consultation. He is an excellent advisor and very supportive and kind person.

I would also like to thank my family for their love and support and for giving me all these wonderful opportunities to follow my dreams and expand my knowledge.

Thanks to my girlfriend Sofia, who's love and encouragement have got me through all the challenges and difficulties throughout these last few years and to my good friend Shura, who was always there for me whenever I needed motivation and support.

To Nvision Czech Republic staff; thank you for all your contribution, advices and feedback.

Version control systems in software engineering

Summary

The main objective of the thesis is to perform an analysis of the existing version control systems and their role in the software development process.

Firstly, author makes a literature review on the topic of source control and gives all the necessary definitions of different mechanisms of version control software work.

Then author creates a general overview of the current version control system solutions, their advantages and disadvantages.

Next, in order to analyze usage of Microsoft Team Foundation Server as a source control management tool, the case study is presented. Development environment of a Czech telecommunication software company is described and its staff's experiences with version control are evaluated.

Then, version control systems are evaluated from a perspective of corporate software development. It is shown how this type of software can contribute to the development process and how it can be used in order to achieve more than it was originally intended by its authors. This part of the research is targeted mainly to a large software development companies

Keywords: Source code management, version control, team development, software engineering, open-source, Integrated Development Environment, Team Foundation Server

Table of content

List of tables

List of figures

1 Introduction

Version control systems (also revision control or source control systems) are an important part of software development process. They allow a team of developers to work on the same code simultaneously, change files and see the history of changes whenever needed. Each team member does his own work through a process called “branching”. It keeps the work of one developer isolated from his teammates until he decides to submit it to the shared code base.

Managing code for multiple projects without VCS would have been a tiresome task for a small team of developers. For a big software company with many branches working on different projects that often overlap well-thought-out source control is a necessity without which the whole development process would have been too slow and complicated.

Large software development companies are currently facing many problems which source control software can help solve. These problems lie mostly in a field of communication, access and needless effort.

In big organizations, multiple teams are mostly scattered across multiple locations. A solution to a given problem can be discovered at multiple locations practically simultaneously. If employees at these different locations don’t have proper means of communication and/or don’t have fast access to each other’s work base then the problem of needless effort arises.

These problems can be successfully solved with modern workflow models, careful and smart product and team management and necessary software systems. Good source control management system can provide better team cooperation, even if people in these teams are physically very far apart. It can also provide means of communication and fast access to the code base of the product.

Version control systems can integrate with different software development tools, including IDEs and build automation software. Nowadays they are powerful and agile tools that help to store a huge amount of information. They might also include a module that can help developers with managing their application lifecycle. VCS’s give companies numerous advantages and allow for more releases in smaller time frames.

2 Objectives and Methodology

2.1 Objectives

The main objective of the thesis is to perform an analysis of the existing version control systems and their role in the software development process and propose the best solution for a company.

One of the partial goals of the thesis is to perform general overview of current version control system solutions, their advantages and disadvantages. Analysis of Microsoft Team Foundation Server in-built source control system - Team Foundation Version Control - will serve as a reference example. Other popular today systems will also be review to determine their advantages and disadvantages.

2.2 Methodology

Methodology of thesis is based on the analysis of multiple information resources and critical points from various literature topics. Views of different authors as well as multiple opinions of users will be analysed and reflected.

Some of the examples will include real life situations from software development cycles of the existing company located in Czech Republic. All experiences will be reflected and evaluated. Final conclusion and recommendations will be given based on the practical part outcomes as well as theoretical analysis of indicated solutions.

3 Literature Review

In this chapter critical literature topics will be reviewed to gain a better understanding of version control systems, their main characteristics and capabilities. Necessary definitions and technical terms from literature sources will also be provided. At the end of the chapter, the problem will be fully formulated and different solutions will be introduced.

3.1 Version control system definition

First thing we need to do before we go into deeper analysis is to define version control systems (VCS) in general.

G2 crowd, business solution review aggregator, states that: "Version control systems are used to track changes to software development projects, and allow team members to change and collaborate on the same files." [2]

Also, from the same resource: "Version control systems allow developers to work simultaneously on code and isolate their own work…" [2]

Developer and the founder of SourceGear software company Eric Sink, author of the "Version Control by example" defines version control systems as: "… a piece of software that helps the developers on a software team work together and also archives a complete history of their work". [1]

Git documentation gives definition of VCS as follows: "Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later".

While all these definitions use slightly different words to describe revision software, we can agree on the basic notion that it is some system that is used to track history of file changes and systemize developer cooperation.

Now that we have some basic definitions of the reviewed software, let's elaborate on what are the actual goals of its existence.

In the above mentioned "Version Control by example" Sink states that there are three basic goals of a version control system. The first and probably the most important

one is to maximize concurrency. That means that each developer should write code in his own thread and ideally never have one thread blocked by another.

Second goal is to minimize the number of conflicts between submitted code changes. Developers need to be sure that changes they submit are not going to overwrite somebody else's code as well as to be sure that their own work is not going to be erased or manipulated without their knowledge.

Third goal is to archive every version of every file that ever existed during the development process along with complete information on changes that that file underwent, who made those changes and why.

Before we continue with the examination of VCSs let us define necessary terms that are going to be used throughout this section.

3.2 Necessary terms

Presented below are the main operations that can be used when working with version control [1].

Lock

Using the definition, given in the "Version Control by example": "The lock operation is used to get exclusive rights to modify a file" [1]

This definition is self-explanatory. Whenever a user wants to modify a file he can "lock" it, which will give him full control on what changes that file should or should not undergo. This will prevent other users from modifying the same file until the lock is on.

Branch

Branch is a separate line of development. At certain point team might want their software to have multiple development lines. The most common example is when the new version of software is being developed, it might be kept separately from the previous version bug-fixes.

Merge

Following Eric Sinks definition: "Apply changes from one branch to another" [1]

Whenever there's a need to combine changes from different branches, it's possible through the process called merging.

Changeset

Change set is a group of changes that are being submitted to the repository.

Commit, check-in or submit

"Apply the modifications in the working copy to the repository as a new change set" [1].

From the definition it is clear that commit is an operation that submits the change set to the code base.

Repository

Repository is a central file storage location.

Checkout

Checkout operation creates a new copy for a repository.

Conflict

Conflict is a situation when several users are trying to make changes to the same files, but one of them has already submitted his changes, and the rest are not.

Head

The main version is the latest version for the branch / trunk, located in the storage.

Rebase

Moving the version from which the branch starts to a later version the trunk.

Revision

The version of the file. Versions vary in numbers and are assigned automatically by version control systems.

Shelving

Delaying changes and saving the on the server without their fixation. It makes it possible not to create branches.

Tag

A label that can be assigned to a specific version. Label represents the name for the files that user decided to merge into one group.

Trunk, mainline, master

Trunk is the main branch of the project development.

Update

Most often this action means updating the working copy to the fresh state of the repository. However, if necessary, it's possible to synchronize a working copy and to an older state.

Working copy

A working copy is a local copy of the storage or a part of it.

3.3 A history of version control

The history of version control can be divided into three generations, as shown on the table below [1].

Generation	Networking	Operations	Concurrency	Examples
First	None	One file at a time	Locks	RCS, SCCS
Second	Centralized	Multi-file	Merge before commit	Subversion, TFS
Third	Distributed	Change sets	Commit before merge	Git, Mercurial

Table 1 – History of source control
Source: [1]

First generation tools used locks as a method of controlling the development concurrency, which meant that only one user could be working on a file at a time.

With second generation of systems came a more freedom. People could work on the same files at the same time with one restriction. Before committing changes files had to be merged with their current versions.

Third generation tools completely separated the processes of merging and committing. [1]

3.4 Existing types of version control systems

Currently, there are 3 main types of version control systems that are available to users: local, centralized and decentralized. They all have their advantages and disadvantages. The choice between them is mostly organizational.

3.4.1 Local VCS

The most preferred VCS for home development would be a local type. To solve this problem, special VCS were developed with a simple database, in which all local files are stored. VCS of other types can also be used in this way, there are possibilities to install and use them locally. [3]

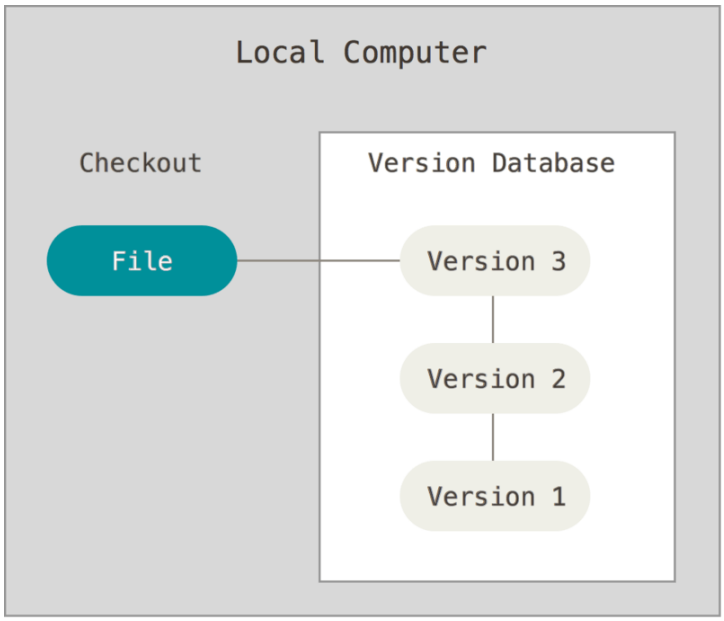

Figure 1 – Local VCS Scheme
Source: [3]

3.4.2 Centralized VCS

Traditional version control systems use a centralized model when there is a single document storage that is managed by a special server that performs most of the versioning

functions. The user working with documents must first obtain the version of the document he needs from the repository; usually a local copy of the document is created, the so-called "working copy". It is possible to get the latest version or any of the previous ones, which can be selected by version number or creation date, sometimes by other attributes. After the necessary changes have been made to the document, the new version is placed in the repository. Unlike simple file saving, the previous version is not erased, but also remains in the repository and can be retrieved from there at any time. [3]

In such systems, for example CVS or Subversion, there is a central server on which all files under version control are stored and a number of clients that receive copies of files from it. For many years this was the standard for version control systems. [3]

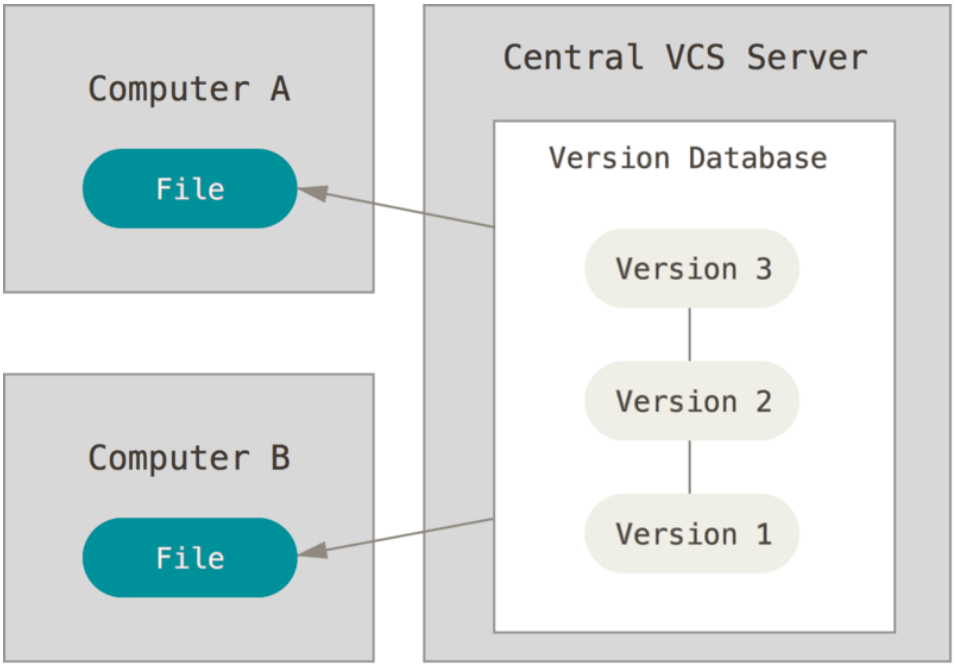

Figure 2 - Centralized version control scheme
Source: [3]

This approach has many advantages, especially over local VCS. For example, everyone knows who is currently working on the project what files are being modified. Administrators have a clear access control and configuration process of CVCS is generally much easier than repeating the same process on each client's machine locally.

However, with this approach, there are several serious shortcomings. The most obvious - a centralized server is a vulnerable location of the entire system. If the server shuts down for an hour, then within that hour developers can't interact and no one can save a new version of their work nor can they get any version from the server. If the disk

with the central database is damaged and there is no backup - the entire history of the project can be lost, with the exception of several working versions that have been preserved on the users' work machines [3].

3.4.3 Distributed VCS

Such systems use the distributed model instead of traditional client-server. In general, they do not need a centralized repository: the entire history of document changes is stored on each computer in the local storage and if necessary, individual fragments of the local storage history are synchronized with the same storage on another computer. Therefore, in the event that the server through which the work is being performed is disconnected, any client repository can be copied back to the server in order to restore the database. In some of the systems, the local storage is located directly in the working copy directories. [3]

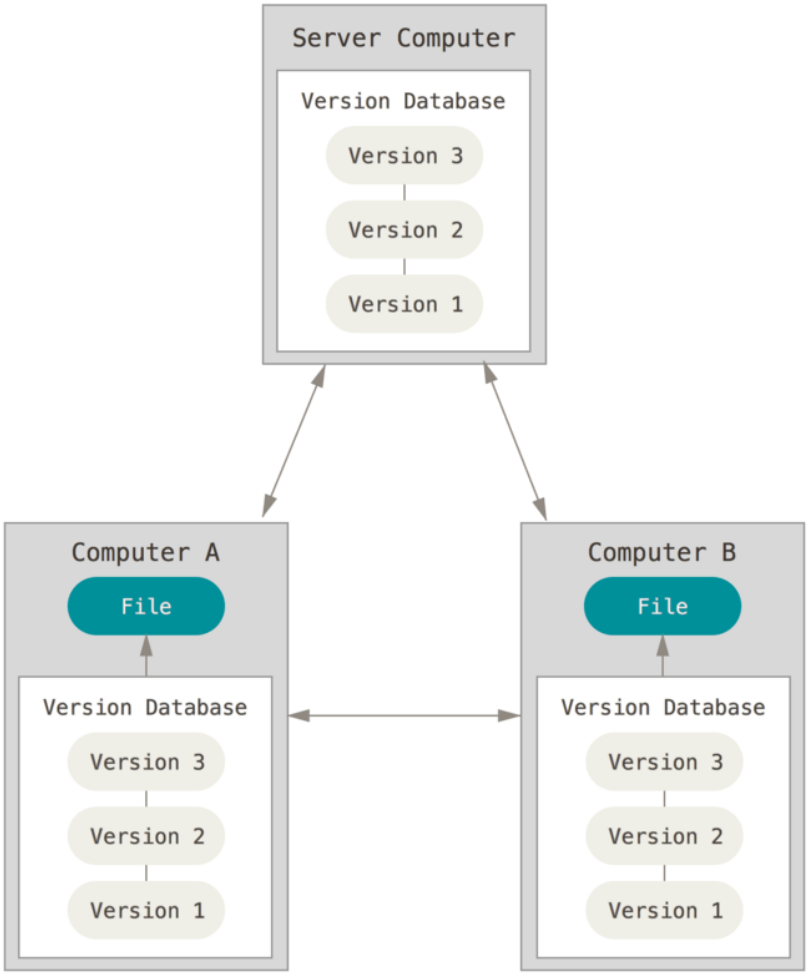

Figure 3 - Distributed model of VCS
Source: [3]

When a user of such system performs ordinary actions, such as extracting a certain version of a document, creating a new version and so on, it works with its local copy of the repository. As changes are made, the repositories belonging to different developers start to differ and there is a need for their synchronization. Such synchronization can be carried out by exchanging patches or so-called sets between users [1].

The described model is logically close to creating a separate branch for each developer in the classical version control system. The difference is that before the synchronization, other developers of this branch do not see it. While the developer changes only his branch, the work he does doesn't affect other project participants and vice versa. When the isolated part of the work is completed, the changes made to the branch are merged with the main (general) branch. As with the merging of branches or when synchronizing different storages, version conflicts are possible. In this case in all systems there are some methods for detecting and resolving merger conflicts [1].

From the user's point of view, distributed system differs mainly by the need to create a local repository and the presence in the command language of two additional commands: the command to retrieve a repository from a remote computer and the transfer of his repository to a remote computer. The first command merges the changes of the remote and local repositories with the placement of the result in the local repository. The second, on the contrary, merges the changes of two repositories and places the result in a remote repository. As a rule, merge commands in distributed systems allow you to choose which sets of changes will be transferred to or removed from another repository, correct merger conflicts directly during the operation or after its unsuccessful completion, repeat or resume an unfinished merge. Usually, the transfer of their changes to another repository ends successfully only if there are no conflicts. If conflicts arise, the user must first merge the versions in his repository and only then transfer them to others [1].

It is usually recommended to organize work with the system in such way so that users always or mostly perform merges in their own repository. This is very different from centralized systems where users transmit their changes to the central server when they think it is necessary. In distributed systems the order is more natural because the merge of the versions is initiated by the one who needs to get its result. For example, the developer managing the assembly server.

The main advantages of distributed systems are their flexibility and a much larger (in comparison with centralized systems) autonomy of a separate workplace. Each developer's computer is, in fact, an independent and full-featured server. From such computers it is possible to build a system of an arbitrary structure and a desired level of complexity by specifying the synchronization order [1].

The disadvantages of distributed systems include an increased amount of required of disk storage: each computer has to store a complete version history, while in a centralized system only a working copy is stored on the development computer. A less obvious but equally unpleasant disadvantage is that in a distributed system it is practically impossible to implement some of the functionality provided by centralized systems, such as [1]:

- locking a file or group of files. It means that special administrative measures are needed in case to work with binary files that are not suitable for automatic merging;
- tracking a specific file or group of files;
- single numbering system of file versions in which the version number increases monotonically. In distributed systems, local version number designations have to be bypassed and tags, style and content of which is determined by either an agreement between developers or corporate standards of the company, have to be applied;
- user has to work with a separate, usually smaller sample of data compared to the significant in size and internal complexity sample of the storage on the remote server.

Following typical situations can be distinguished in which the use of a distributed system gives noticeable advantages:

- periodic synchronization of several computers under senior developer's supervision. Usage of a distributed system eliminates the need to allocate one of the computers as a server in such situation;
- joint work on the project of a small geographically distributed group of developers without allocation of common resources. As in the previous case, the scheme is implemented without the main server, and repositories are kept up-to-date by periodic synchronizations;
- a large distributed project, the participants of which can work on their own part individually for a long time without having a permanent connection to the company

network. Such a project can use a centralized server, with which copies of all its participants are synchronized. It is possible to work without "group" servers - then the developers of one group synchronize the changes among themselves, after which any of them transfers the changes to the central server.

3.5 Typical work process with VCS

Each VCS has its own particular features that are reflected in the set of commands, the order of operations and administration process. Nevertheless, the general order of work with the majority of VCSs is usually typical. In the following example it is assumed that the hypothetical project already exists and the server hosts its repository to which the developer gets access [1].

3.5.1 Getting started with the project

The first step that a developer must perform is extracting a working copy of the project or the part of the project that he is going to work with. This action is performed using the standard version extraction command (checkout or clone) or a special command that performs the same action. The developer sets the version that should be copied, the most recent one is usually set as default (or selected by the administrator as the main).

Upon the completion of the extraction command a connection to the server is established and the project in the form of a directory tree and files is copied to the developer's computer. It is a common practice to duplicate a working copy: in addition to the main directory with the project, another copy of the project is added to the local disk. While working with the project, the developer changes only the files of the main working copy. The second local copy is stored unchanged, allowing at any time without contacting the server to determine what changes were made to a particular file or project as a whole and from which version a working copy was made; as a rule, any attempt to manually change this copy results in errors in the operation of the VCS software.

3.5.2 Daily work cycle

With some variations, determined by the details and developments of the adopted business process, the usual work-cycle during the day is as follows:

1) Update the working copy. As changes are made to the main version of the project, the working copy on the developer's computer becomes obsolete: the discrepancy

with the main version of the project increases. This increases the risk of conflict change. Therefore, it is convenient to maintain a working copy in a state closest to the current main version, for which the developer performs the operation of updating the working copy (update) as often as possible;

2) Modification of the project. The developer modifies the project, changing the files included in a working copy in accordance with a task. This work is done locally and does not require calls to the VCS server;

3) Fixation of changes. Having completed the next stage of the task, developer commits his changes, transferring them to the server or to the main branch if the work on the task is completed or to a separate branch of development of this task. The VCS may require the developer to create a copy of the current version before committing. If the system has support for deferred changes, (shelving) they can be transferred to the server without a commit. If the current company policy of allows this then committing of changes may not be carried out every day, but only after the completion of work on the assignment; in this case, until the work is completed, all changes associated with the task are saved only in the local working copy of the developer.

3.5.3 Branching

It is common to make small corrections in the project by directly editing the working copy and then fixing the changes directly in the main branch (trunk) on the server. However, when performing a significant amount of work on a single task, this procedure becomes inconvenient: the absence of a possibility to commit intermediate changes to the server does not allow the developer to work on anything in the group mode and the risk of losing changes in local accidents increases, while the ability to analyze and return to previous versions is lost. To avoid such problems, the practice of creating branches has to be implemented. A version of a project or a part of it, the development in which is conducted in parallel with the changes in the main version is called a branch. When the work for which the branch was created is completed, the branch is reintegrated into the main version. This can be done by a merge command or by creating a patch containing the changes made during the development of the branch and applying this patch to the current major version of the project.

3.5.4 Version merging

Three things can happen during the source control system usage that can lead to the need to combine changes:

- Update of the working copy;

- Commit of changes;

- Merge of branches.

In each system, there is an automatic merge mechanism that works based on the following principles:

- Changes can consist of file content modifications, creation of new files or directories or removal/renaming of pre-existing file or directory in the project;

- If two changes concern to different and unrelated files or directories, they can always be applied simultaneously. Changes made in each version of the project are copied to the merged version;

- Creation, removal and renaming of files in project directories can be combined, unless they conflict with each other. In this case, the changes made in each version of the project are copied to the joint version.

Actions that usually conflict with each other:

- Removal or modification of the same file or directory;

- Removal or renaming of the same file or directory;

- Creation of different versions of a file with the same name and different contents;

- Changes within a single text file are always conflicting.

In all cases, the base version for the merge is the one in which separate branches were created. In case of commit operation the base version will be the one of the last update before commit, in case of update - the version of the previous update and in case of merge - the version from which the particular branch was created.

The majority of modern version control systems are oriented to software development projects in which the main type of file content is text. Accordingly, the mechanisms for automatic merge changes are oriented to the processing of text files.

A mechanism for line comparison is used to decide if it's possible to merge changes within the same text file. It compares the merged version with the base one and builds a list of changes that took place within the file. The sets of changed rows that do not intersect are considered compatible and their merging is done automatically. If the merged files contain changes that affect the same line in the file then it results in a conflict. Such files can only be combined manually. Any files except text in terms of VCS are binary and do not allow automatic merging.

3.5.5 Conflicts and their resolution

A situation when a merge of several versions of the same file causes the changes made in them to cross is called a conflict. If there are conflicting changes, the version control system cannot automatically create the merged project and is forced to contact the developer. As mentioned above, conflicts can occur at the stages of commit, update or merging of branches. In all cases when a conflict is detected the corresponding operation is suspended before its resolution.

To resolve a conflict, the system offers the developer three versions of the file that caused it to choose from: basic, local and server. Conflicting changes are either shown to the developer in a special program module or simply marked with special markup directly in the text of the merged file.

Conflicts in the file system are usually much simpler. They can only occur when delete operations is applied simultaneously with one of the other operations.

3.5.6 Locks

Locking mechanism allows one of the developers to select a file or group of files and "lock" them for their own use to make changes to them. While the file is locked, it remains accessible to all other developers only for reading and any attempt to make changes to it is rejected by the server. Technically, a lock can only be organized in a number of different ways. The following mechanism is typical for modern systems:

- Files that require locking are marked with a special "blocked" flag;

- If the file is marked as locked and the working copy is extracted from the server, it gets the read-only attribute on the local file system, which prevents it from being accidentally edited.

Developer who wants to change a locked file runs a special lock command with the name of the file. The result of this command is the following:

1) Server checks to see if the file has already been blocked by another developer; if so, the lock command fails;

2) The file on the server is marked as "blocked", with the identifier of the developer who blocked it and the time of the lock being saved;

3) If the lock on the server was successful, the "read-only" attribute is removed from the working copy on the local file system, which allows the user to start editing it.

When the user is finished with the file modification, he can call the unlock operation to release the lock. All changes made to the file will be canceled, the local file will return to the "read only" state, the "blocked" attribute will be removed from the file on the server and other developers will be able to modify this file. Normally, the lock is automatically removed when the users commits his changes.

The version control system provides storage of all existing variants of files and, as a result, all variants of the project as a whole that have taken place since the moment of its development. But the very concept of "version" in different systems can be treated in two different ways.

3.5.7 Project versions, tags

Version control systems provide storage of all existing variants of files and as a result, all variants of the project as a whole that existed since the moment of its development. But the very concept of a "version" in different systems can be treated in two different ways.

Some systems support the versioning of files. This means that any file that appears in the project gets its own version number. During each commit by the developer, the corresponding part of changes is applied to the file and the file gets a new version number, usually and increment of the previous one.

For other systems, the term "version" refers not to a single file, but to the entire repository. The newly created empty repository has version 1 or 0 and any commit of changes leads to an increase of this number. Version numbers for separate files simply do not exist in this type of VCS.

However, for convenient marking versioning of projects numbering is not enough. That is why version control systems support the concept of tags.

A tag is a symbolic label that can be associated with a specific version of a file or directory in the repository. Using the appropriate command, a given label can be assigned to all or part of the project files that meet certain conditions. Thus, it is possible to identify the version of the project, when fixing its state at any given time. As a rule, the system of tags is flexible enough and allows the user identify files from different project versions with the same tag. It makes it possible to compose a project from different versions of its parts in any desired way.

3.6 Software life-cycle

Software life-cycle is a series of events that occur with the system in the process of its creation and further use. In other words, this is the time from the initial moment of software creation until the end of its development and implementation. It can be represented in the form of models [4].

3.6.1 Typical life-cycle

There are 5 main stages of software life-cycle that are present in every model [4]:

1) **Requirements analysis**

Life cycle of software development begins with the analysis stage, during which process participants discuss the requirements for the final product. The purpose of this stage is to determine the detailed requirements for the system. In addition, it is necessary to make sure that all participants correctly understood the tasks and how exactly each requirement will be implemented in practice.

Often, the discussion also includes testing specialists who, at the design stage, can make their own wishes and, if necessary, adjust the process.

Depending on the chosen development model, the approaches to determining the moment of transition from one stage to another may differ.

2) **Design**

At the design stage (also called the stage of design and architecture) programmers and system architects, guided by the requirements, develop a high-level system design.

A variety of technical issues arising in the design process are discussed with the customer. The technologies that will be used in the project, restrictions, time frame and the budget are determined. In accordance with specified requirements, the most suitable design solutions are selected.

Approved design of the system determines the list of software components to be developed, interaction with third parties, functional characteristics of the program, databases to be used and much more.

3) **Implementation / Coding**

After requirements and design of the product are approved, a transition to the development stage of the life cycle occurs. Programmers begin to writhe program code in accordance with previously defined requirements.

In addition, programmers write Unit-tests to verify the correctness of work of each system component, review the written code, create builds and deploy the finished software in a software environment. This cycle is repeated until all requirements are implemented.

4) **Testing**

During this phase bugs that were missed during development are looked for and reported. Upon finding a defect, tester generates an error report that is passed on to developers. After the bug is corrected testing is repeated - but this time in order to make sure that the problem was fixed and the fix itself did not cause new bugs in the product.

5) **Deployment and maintenance**

When the program is tested and there are no more serious defects in it, it is officially released and delivered to the end users.

After the release of the new version of the program, technical support department provides help and consultations to users.

In the event that certain post-release bugs are discovered, information about them is transmitted in the form of error reports to development team, which, depending on the severity of the problem, either immediately issues a fix (hot fix), or defers it to the next version of the program.

In addition, technical support team helps to collect and systematize various metrics - program's performance in real conditions.

3.6.2 Waterfall model

Cascade model (otherwise known as the waterfall model) is a model of software development in which the process looks like a flow that goes through phases of requirements analysis, design, implementation, testing, integration and support [21].

In this model, developer moves from one stage to another strictly in sequence. First, the "requirements definition" stage is completed, resulting in a list of requirements for the software. Once requirements are fully defined, a transition to design stage takes place, during which detailed documentation and plans for future implementations are created. After the design stage is completed, programmers implement the project. At the next stage of the process integration of individual components developed by different teams of programmers occurs. After the implementation and integration are completed, the product is tested and debugged. At this stage all the defects that appeared in the previous stages of development are eliminated. After that, the software product is implemented and its support is provided [21].

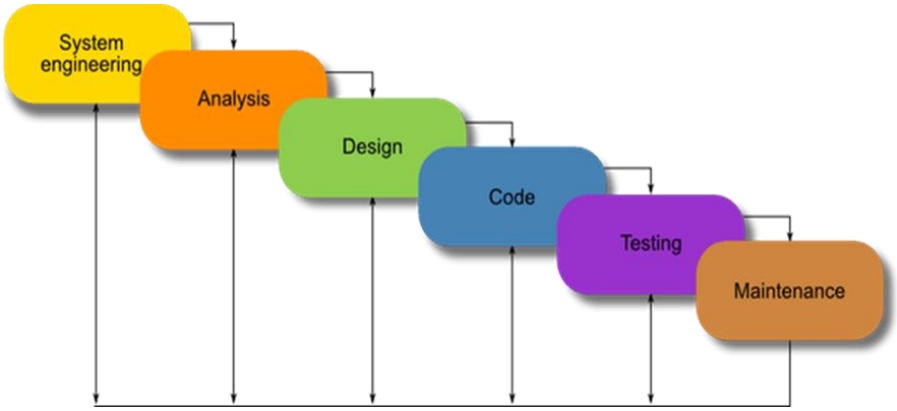

Figure 4 – Basic Waterfall diagram
Source: [20]

This model implies one-time execution of each phase of the project. The transition from one phase to another is possible only after successful completion of a previous stage. Each stage implies detailed planning and full correctness of the result of the stage. Such strict sequence constraints allow for as transparent development process as possible.

Downside of this method is the need to maintain and constantly update the product documentation. Any change must be agreed with the customer. Insufficient level of elaboration of requirements brings with it an increase in the budget and timing of the project, which are quite difficult to assess in advance.

Today, the waterfall model of software development is rarely used because of its small flexibility. However, there are still some reasons for this model to be used because of its high transparency. Due to the high level of formalization, it is much easier to manage such a project. It is generally believed that this development model reduces risks and clarifies the development process when a few dozen people work on the project [21].

Cascade model is suitable for development of complex projects and systems with strictly defined functionality. Common examples of such projects are large state orders or scientific developments. [21]

Disadvantages of the waterfall model:

- Very non-flexible methodology;

- Can create a wrong impression about the percentage of complete work (for example, the phrase "50% completed" does not mean that project is half-finished);

- The customer does not have the opportunity to familiarize himself with the system in advance;

- All requirements must be known at the beginning of the project;

- There is need for strict management and regular monitoring;

- There is no possibility for alteration, the entire project is done at once.

Advantages of the waterfall model:

- High transparency of the development and phases of the project;

- Clear sequence;

- Stability of requirements;

- Strict project management control.

3.6.3 Agile software development

Agile software development is a series of approaches aimed at using interactive development, dynamic formation of requirements their implementation as a result of constant interaction within self-organizing working groups consisting of specialists of various profiles. There are several methods related to the class of flexible development methodologies, in particular, extreme programming, DSDM, Scrum [22].

Most flexible methodologies are aimed at minimizing risks by bringing development to a series of short cycles called iterations, which usually last two to three weeks. Each iteration in itself looks like a software project in miniature and includes all the tasks necessary to provide an increase in functionality: planning, requirements analysis, design, programming, testing and documentation. Although a single iteration is usually not enough to release a new version of the product, a smaller version of a flexible software project is ready for release at the end of each iteration.

Main principles of agile [22]**:**

- Increased customer's satisfaction due to early and uninterrupted delivery of software;

- A change in requirements can be appropriate even at the end of development (this can increase the competitiveness of the product);

- A quick delivery of working software (every month or week or even more often);

- Daily communication of the customer with developers throughout the project;

- Project involves motivated individuals who are provided with the necessary working conditions, support and trust;

- Recommended method of information transfer - personal conversation (face to face);

- Working software is the best measure of progress;

- Constant adaptation to changing circumstances.

Main advantages of agile:

- Risk minimization

A large project enables a customer to pay several iterations of it and to understand that he will receive in time exactly what he wants. Waterfall models (with the use of specifications and technical specifications) do not provide such opportunities.

Customer always has the opportunity to observe the development process, adjust the project's functionality, test or run it or even stop it at any time.

- Quality improvement

Involving the customer in the process of each iteration makes it possible to correct the process, which improves the quality.

- Higher development speed

Iteration lasts no more than 3 weeks, by the end of this period there is necessarily a result.

3.6.4 Scrum

Scrum - is a development process built on a set of principles, which allow in a number of short-term iterations, called sprints, to provide the end user with running software with new capabilities for which the highest priority is determined. The new opportunities that will be worked on during the sprint are determined at the planning stage and cannot be changed throughout its entire length. Sprint time is also regulated at the planning stage and invariably, which makes the product creation process predictable and flexible. [5]

Main roles of project participants, developed with Scrum:

- A product owner is a person who knows how he wants a product to look and forms a list of requirements that are entered in the backlog of the product;
- A team that directly participates in the development. It is very important for it to be self-organizing and self-governing and that it could also independently identify which of the proposed requirements should be developed first;

- Scrum Master is a person in charge of carrying out all the main procedures according to the rules of Scrum.

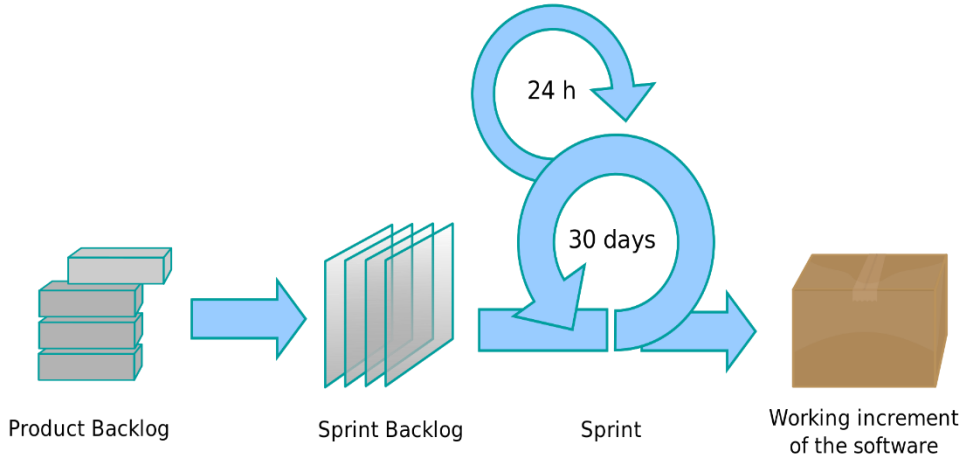

Figure 5 – Basic SCRUM diagram
Source: [5]

Sprint planning is the first stage. Here, the functionality that needs to be developed and how long it will take is decided. Then the sprint begins.

Sprint - iteration in the scrum, during which the functional growth of software is created. It is rigidly fixed in time, its duration is from 2 to 4 weeks. However, different teams select the sprint length according to the specifics of their work, team composition and requirements, often by trial and error. It is believed that the shorter the sprint, the more flexible the development process is, releases go out more often, feedback from the consumer comes faster and less time is spent working in the wrong direction. The tasks that are solved during the sprint are taken from the backlog.

Product backlog is a list of requirements for functionality, ordered by their importance, to be implemented. Elements of this list are called user stories or backlog items. Project backlog is open for editing for all participants of the scrum.

Sprint backlog – contains functionality selected by the owner from the project. All functions are divided into tasks, each of which is evaluated by the scrum team. Every day, team evaluates the amount of work that needs to be done to complete the sprint.

Daily meetings are also an important part of scrum. During each of these meetings team members explain who did what and ask each other questions if necessary.

4 Practical part

In this part we evaluate the role of version control in software development.

Firstly, the analysis of four popular version control systems is presented. Their comparison is made and with authors opinion on which system suits a particular type of software development company better.

Secondly, Microsoft Team Foundation server will be described and its usage is to be presented in the case study.

Thirdly, Version control is to be analysed from the corporate software development perspective. In order to achieve this goal, interviews will be conducted with different members of a large telecommunications software company based in Prague and their experiences reflected and evaluated.

Fourthly, findings as to what functions of source control play an important role in the software development and how these functions can be used to the advantage of programmers who use these systems are presented and conclusions are made.

4.1 Version control systems overview

4.1.1 Team Foundation Server

Team Foundation Server (TFS is designed to integrate development tools for faster and more comfortable work.

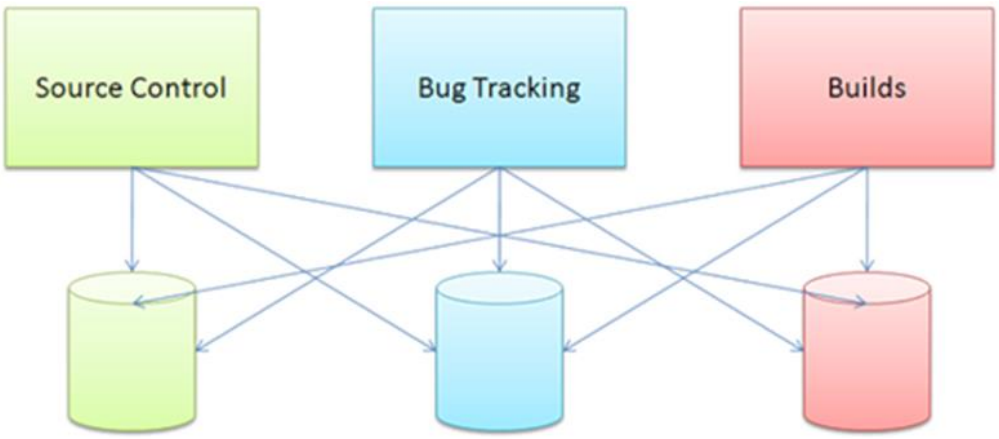

Figure 6 – TFS components depicted separately
Source: [23]

In the case described in the figure [6], each system has its own data store, its own set of identification data, its own commands and tools. Although such a system can exist and be used, switching between components and their support would take a lot of time.

TFS is a system that integrates all the necessary components together.

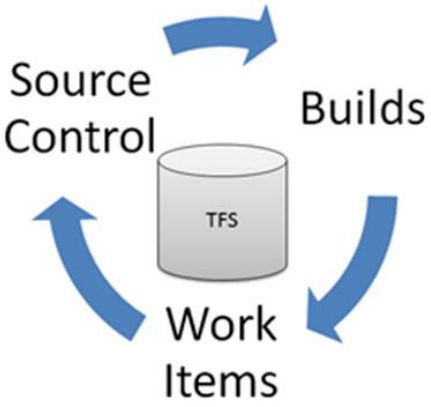

Figure 7 – TFS components
Source: [23]

This integration covers the most common scenarios. On a typical day, programmer will most likely have to add and edit the source code, build, test, log errors and fix them. When the entire workflow occurs in one integrated environment, each of the process elements can be linked. For example, if a programmer copies files to the repository in which he corrected an error, he might want to make a note of this in the report. The full version of TFS makes it possible to do everything described above and also includes: automatic testing, deployment of the virtual lab and testing of the application architecture.

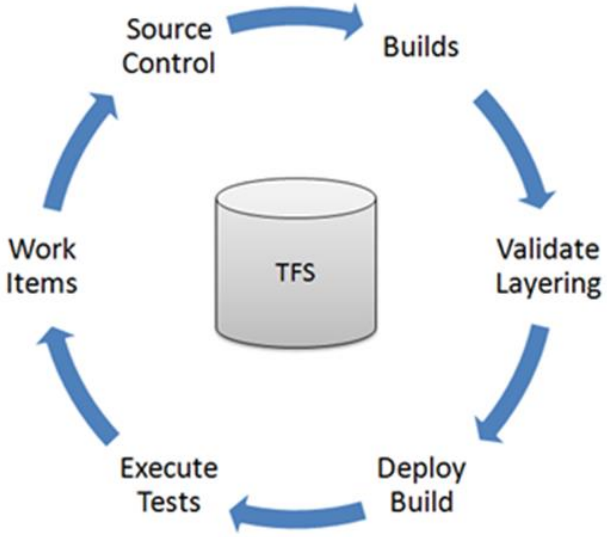

Figure 8 – Full TFS components
Source: [23]

Depending on the need it is possible to use only a part of the components. There are many ways to access the TFS functionality. For programmers the most comfortable way would be through Visual Studio. For a tester Team Explorer can be used as a client, without Visual Studio installation. A project manager can access information through a web browser, Excel or Microsoft Project.

Part of TFS that is responsible for the source control is called Team Foundation Version Control (TFVC). Although, it is worth mentioning that oftentimes, when users refer to the version control capabilities of TFS they refer to it as "TFS" rather than "TFVC".

In order to utilize distributed source control features it is possible to use other features of TFS, such as bug tracking and build automation in conjunction with Git. In this case, Git would replace TFVC as a source control part of TFS thus making it distributed version control system. An example of the programming group that can benefit from such configuration is given in the case study.

4.1.2 Git

Distributed file version control system. Created by Linus Torvalds to monitor and manage the development of the Linux kernel, the first version was released on April 7, 2005. This version control system is designed as a set of programs that are designed to be used in scripts, which makes it convenient to create version control systems based on Git or user interfaces for specific tasks.

Git has support for quick merging and distribution of versions, as well as tools for working with the development history. Git allocates a local copy of the entire development history for each developer and the changes are copied between the repositories. The Git kernel is a set of command-line utilities with parameters and all system settings are contained in the configuration files. In this regard, Git easily ported to any platform and has the possibility of integration into other systems (graphical git clients with any desired interface could be an example).

4.1.3 Mercurial

Mercurial is a cross-platform distributed version control system designed to work effectively with very large code repositories. First of all, it is a console program.

The Mercurial system is written in Python, although the performance-sensitive parts (for example, its own diff implementation) are implemented as extension modules in C. Mercurial was originally written for Linux, later ported to Windows, Mac OS X and most Unix systems. Mercurial repositories are managed using the hg command-line utility, but there are also GUI interfaces.

Along with the traditional capabilities of version control systems, Mercurial supports completely decentralized work (there is no concept of the main code repository), branching (it is possible to keep several branches of one project and copy changes between branches), merge repositories (which is the "distribution" of work). It supports the exchange of data between repositories via HTTP / HTTPS, SSH and manually with the help of packaged change sets.

4.1.4 Subversion

Subversion is a centralized system (unlike distributed systems such as Git or Mercurial) therefore the data is stored in a single repository. The repository can reside on a local disk or on a network server. Work in Subversion differs very little from other centralized version control systems. Clients copy files from the repository, creating local working copies, then making changes to the working copies and fixing these changes in the repository. Multiple clients can access the repository at the same time.

4.1.5 Assessment

Direct comparison between all described systems can be considered incorrect due to fundamental differences between centralized and decentralized systems. However, certain well known advantages and disadvantages of these VCS approaches can be described and certain recommendations can be made as to which benefits particular software development company type more.

Differences between representatives of each of these categories are more subtle. However, preference is largely opinion based.

4.1.6 Centralized vs Decentralized

Since decentralized systems are younger than centralized it can be seen that in many ways they improve on previous concepts.

Advantages of decentralized systems:

- Possibility to share changes with programmers before commit;
- Commit action can be local, other developers may not see the change set unless committing user chooses so;
- Internet connection is not required, therefore remote work is possible for distributed teams.

Major difference of decentralized source control is that such systems operate with changes, not versions.

Based on the ideas which this type of systems brings to the software development a clear description of companies that can benefit from such approach can be made. They are largely distributed with a frequent necessity of their teams or individual employees to work remotely or independently from one another.

Despite the fact that decentralized approach is preferred by more and more developers and online communities nowadays, companies and products where development process simply does not permit for the decentralized approach still exist. These companies can be described as monolithic, with strict centralized management. Oftentimes products developed by them simply cannot allow for a possibility of confusion between change sets and file versions or even existence of full local repositories that differ from one another. Such products consist of big platforms with many interconnected modules which almost always work in conjunction with one another and are highly sensitive to changes. Example of such company and product is given in the section where the case study is presented.

4.1.7 Centralized systems. TFS vs SVN

Software	Repository	Concurrency	Platforms	Cost
TFS	Centralized/Distributed	Merge/Lock	Windows or online	Non-free (varies)
SVN	Centralized	Merge/Lock	Windows, Unix, Os X	Free

Table 2 – TFS and SVN
Source: author

System	Advantages	Disadvantages
TFS	• Perfect IDE integration (Visual Studio) • More than source control (bug tracking, build automation, etc.)	• Price • Installation complexity • Developers must always be connected to the server
SVN	• Developers do not have to stay connected to the server • Free of charge • Easy and quick installation process	• External tools for graphic interface • Better small project flexibility

Table 3 – TFS and SVN comparison
Source: author

Considering given advantages and disadvantages it can be said that TFS is a better choice for large teams with a smaller amount of projects that work within the IDE and use primarily Microsoft products.

For smaller teams with many projects that vary in nature, when cost is an issue – SVN is a better choice.

4.1.8 Distributed version control. Git vs Mercurial

Software	Repository	Concurrency	Platforms	Cost
Git	Distributed	Merge	Windows, Unix, Os X	Free
Mercurial	Distributed	Merge	Windows, Unix, Os X	Free

Table 4 – Git and Mercurial
Source: author

When it comes to distributes systems, both Git and Mercurial provide similar level of features so the choice is largely a matter of preference by a particular group of users or a company. Both of these products have a large fan-base and are used extensively throughout the industry. Technical differences are largely low-level implementation

nuances and are only noticeable to experienced users who have used both systems for a long time. Many developers notice that Git, although more technically sophisticated, has a steep learning curve and the decision on whether or not to use it or prefer Mercurial often comes down to how well the team is acquainted with Git commands and their syntax.

4.2 Case study (Nvision Czech Republic)

4.2.1 General overview

Brief history

NVision Czech Republic is a Czech telecommunications company. It was founded on August 31, 1993, as STROM telecom. Until 1998, the company was located in the city of Benesov, since mid-1998 in Dobravitsy, from 2007 its main office is located in Prague. On February 13, 2007 it was renamed SITRONICS Telecom Solutions, Czech Republic. On December 16, 2013 it was renamed into NVision Czech Republic [6]. The company had changed its name because it was bought by a Russian company Nvision Group.

Nvision Group

NVision Group is a Russian developer and provider of information and communication solutions, services and services, including cloud services. In the year of 2013 NVision Group was the 2nd in the provision of IT services [7], as well as among the largest consulting groups in Russia [8] and one of the top 5 companies of the Russian IT market [9].

In December 2015, NVision Group was bought by MTS. As a result, MTS became the sole shareholder of NVision Group and all its subsidiaries, including Russian Sitronics Telecom Solutions and Czech Nvision Czech Republic. [10]

MTS

MTS is one of the leaders among telecommunication companies in Russia. The operator provides mobile telephony services and cable television. The subscriber base of the company includes more than 102 million subscribers. MTS provides services in all regions of Russia, Armenia, Belarus, Ukraine and Turkmenistan. The company regularly

develops and provides new tariff plans and additional products: data transmission in mobile and fixed networks, digital TV broadcasting, geo-positioning, etc.

As a result of acquisition of Sitronics Telecom Solutions and Nvision Czech Republic, the telecommunications company received full control over its own billing. Before that Nvision Group used to be the exclusive developer and owner of MTS billing (FORIS BSS / OSS, MEDIO SCP) and its main contractor development of IT solutions. At the end of 2014, more than 50% of the revenue of the NVision Group of Companies brought projects commissioned specifically by MTS. Back in 2012, the operator's share in the orders of the IT company was somewhat less - 24% in monetary terms. [11]

In the annual report of MTS for 2015 it was said that the acquisition of NVision Group is part of the long-term strategy of the operator aimed at business diversification and innovations development. At the expense of the purchase, the company expects to improve the service of its subscriber base through investments in the modernization of billing, to save on IT costs through integration and to strengthen its position in the corporate market. MTS plans to offer customers integrated solutions in the form of a full range of IT services, telecom services and system integration. [13]

At the time of the announcement of the acquisition of NVision Group it was already noted by numerous sources that the company had long received most of its orders from MTS. With the deal closed, NVision de-facto became an insourcing systems integrator for the MTS.

4.2.2 Main products and developments

OSS/BSS definition

OSS/BSS is an abbreviation of the Operation Support System / Business Support System [15]. This is a class of software products that are used by telecom operators, TV companies, energy enterprises and other organizations that regularly and personally interact with customers: maintain individual accounts, monitor the consumption of services and regularly bill their subscribers. A telecommunications company cannot exist without the processes that OSS / BSS provides, it is the core of its business.[15]

OSS/BSS solutions are responsible for two aspects of the telecommunications company work: infrastructure and resource management, and interaction with

subscribers. The main function of such solutions working in a complex is to ensure that services are provided and accounted for. This task is functionally divided into several parts. Proper operation of the network infrastructure and equipment (networks, subnets, switches, PBX, base stations, etc.) is what the OSS part is responsible for. [15]

Interaction with subscribers (accounting services by tariffs, monitoring of account status, billing, etc.) occurs in the second part of the system - BSS. The basis of BSS is a billing system in which all financial settlements with subscribers take place. Also, this complex can include a CRM (Customer Relationship Management) system, responsible for customer relations, which stores various data for each subscriber used for marketing purposes. Also, the BSS can include the Enterprise Resource Planning (ERP) system, which is used to manage enterprise resources, maintain accounting, financial accounting, project management, organizational structure, etc. [15]

As can be seen, these two aspects of the telecommunications company's activities are built on various business processes, each of which can be provided with a separate software product from a separate vendor. In this case, integration difficulties inevitably arise. Data from one program must be transferred quickly and without loss to another. But if the tools are released by different manufacturers, it becomes difficult to organize. There is a need to optimize systems, which requires money and time. Business development requires the development of an information system, but often this is solved only by creating a "patch" - by adding the next software product. So the problems accumulate, like a snowball, keep the IT department in constant tension, and increase financial costs. A much simpler and more optimal way is to use the full complex of OSS / BSS software products from one vendor, who has already made sure that individual modules interact as efficiently and quickly as possible.

Nvision

On the official web-site of Nvision Czech Republic it says that:

“We supply telecom operators with end-to-end systems and services that enable their strategic, operational and business activities. We focus on billing and BSS solutions based on our own convergent billing solution and professional services surrounding it “[14]

Necessary terms

An intelligent network (IN) is a service-independent telecommunications network. That is, intelligence is taken out of the switch and placed in computer nodes that are distributed throughout the network. This provides the network operator with the means to develop and control services more efficiently. New capabilities can be rapidly introduced into the network. Once introduced, services are easily customized to meet individual customer's needs. [16]

OCS is a specialized communications function that allows a service provider to charge a user for services in real-time. The OCS handles the subscribers account balance, rating, charging transaction control and correlation. With the OCS, a telecom operator ensures that credit limits are enforced and resources are authorized on a per transaction basis. [17]

A physical or virtual node within the intelligent network, the Service Control Point (SCP) contains the active database of customer records. This database is actively queried from either the Service Switching Points (SSPs) or Service Platform Trigger Points (STPs) within the network to seek and obtain service-completion information. The SCP queries the service data point (SDP) which holds the actual database and directory. SCP, using the database from the SDP, identifies the geographical number to which the call is to be routed. [18]

Regressive testing – a testing method that checks how old functionality that had already been tested in the past was affected by new changes.

Real-time charging

One of the main products that Nvision Czech Republic is continually expanding and supporting is MSCP platform. It is essentially a hardware-software complex for real-time charging.

On the Nvision CZ official website, the platform is described like this:

"MEDIO IN/SCP is a flexible and modular next-generation intelligent network platform that provides the necessary basic to launch services quickly and efficiently. It is based on an efficient Service Execution Environment (SEE) that is able to run IN services and perform online and off-line charging. The ease of service development and

customization is due to a user-friendly graphic service creation environment. The on-line-charging system (OCS) is operated on one or several adjacent MEDIO IN/SCP platforms. The MEDIO IN/OCS allows for the unique opportunity of providing operators with real-time fully convergent billing of both value-added services and regular voice and data traffic…" [19]

OSS/BSS development sector

This part of Nvision Czech Republic (Nvision CZ from now on) is responsible for the development of new services for the MSCP platform as well as for support and bug fix of already existing ones. It is divided in three main logical sectors each of which is responsible for their own part, but is nevertheless inseparable from the others. Those three sectors are development, architecture and testing.

Figure 9 – Nvision CZ development sector
Source: [author]

Waterfall

For years since the development of the first versions of the MSCP platform the company used the waterfall model of development, while releasing approximately one version of the platform a year.

The release was basically an increment of the platform. It contained all the changes made to the source code throughout the year.

Firstly, the MTS would develop a new service or an old service alteration for its customer base. Then it would send the newly created requirement to the Business Analytics department of Nvision CZ. After the detailed analysis and agreements with MTS, BA department would release a new version of the High Level Architecture (HLA)

Design document which contained all the necessary information for the architecture department to start the development of the actual service business logic.

The result of the architecture departments' work would then be presented in front of the whole team to give them an idea about the upcoming change and its scope.

From this stage, the development department would actively participate in the process. Firstly, group leaders would assess the amount of work. Than they would create and distribute tasks between programmers. Programmers would start working, while actively consulting their assigned architects and group leaders. After the programming of major parts is completed, compiled platform modules would be sent to testing department.

By this moment, testing department would be ready for the upcoming module updates, having their simulated platform environments and test-cases ready and waiting. Just like in the development department, testing group leaders would assign appropriate amounts of work to each tester. Testing process would inevitably identify problems that have to be fixed by programmers. Therefore, testing and bug fix are closely connected, because bugs are fixed as soon as they are found, prior to release.

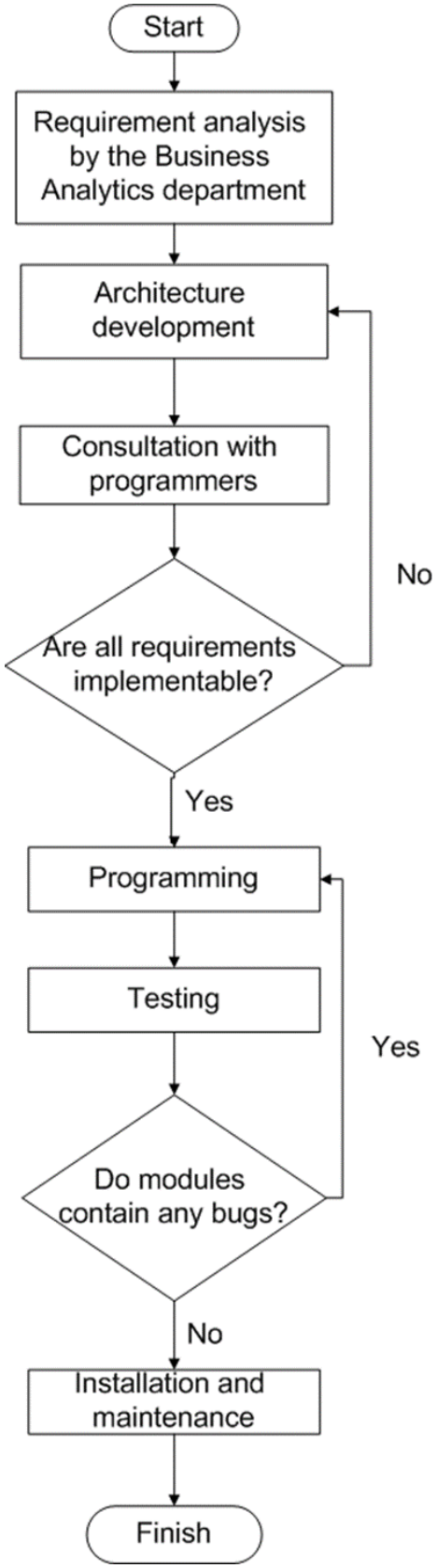

Figure 10 – Nvision CZ waterfall release model
Source: [author]

When the testing is finished and all found bugs and problems are fixed, technical support team would prepare release for installation on major platforms. All the teams would be in stand-by mode during the installation, ready to fix any upcoming problems. When the platform is successfully updated, the whole team would gather for a retrospective and soon the process would start all over again (Figure 10).

Scrum

With the acquisition of Nvision Czech Republic by MTS it er was decided that such model is no longer acceptable. The main client wanted more yearly releases and it was decided that SCRUM methodology should be implemented to achieve faster development cycles and agility of requirement implementation.

So that the production of new releases wouldn't be interrupted, it was decided to implement the new model gradually and to the extent that is possible. Some features that were implemented are such as:

- Sprint length – 2 weeks;

- Demo and retrospective at the end of each sprint;

- Partial releases or "hot-fixes" at least 4 times a year;

- Small feature/changes scope, with enables constant interaction between programmers, architect, analytics and the product owner;

- Agility of requirement implementation and definition;

- Division of work by different SCRUM teams, which enables higher resource allocation efficiency with improved work-load balance;

- Unit-testing for major modules is now possible because of improved programmer-tester interaction.

Outcome of the transition

Scrum methodology, although adopted, didn't prove to be as effective as was expected. During the interviews, it was reported by a number of employees that they preferred waterfall model better, as scrum methodology is currently putting too much overhead (everyday meetings, consultations etc.) while not providing a significant enough

boost in productivity. Main reason for that, they noted, is the fact that 70% of their work time they perform tasks related to bug fix and small features improvement which are unrelated to their scrum team user stories. On top of that, these tasks often have higher priority as they often originate during direct usage of older releases and require immediate implementation or correction.

Nature of work and its organization

Official name of the programming department is "Services Control Group" (SCG from now on). Its main goals are implementation of new services for the MEDIO IN/SCP platform and bug fixing for the currently installed and running versions. Its employees are mostly wide-profile back-end programmers with basic understanding of telecommunication protocols and procedures.

There are two main subgroups of specialists in this department – SEE (Service Execution Environment) and Services programmers.

First subgroup works mainly with parallel implementations of lower-level means of communication and control routines for the platform. Second subgroup writes actual modules which provide or enhance functionality of the platform users. Both these groups are coding in C# programming language using .NET Framework.

4.2.3 TFS usage in practice

Where it's used the most

To determine in which stages of the company's software development life-cycle VCS is used the most, what functions are used and how often and what is expected from this type of software in general it was decided to conduct a number of interviews with different team members of departments that are most likely to use VCS on a daily basis. Department choices are based on the previous section: Architecture, Programming and Testing. Table 5 provides a list of people with whom the interviews were conducted.

Architecture	Programming	Testing
Senior Solution Architect	Group Leader	Group Leader
Telco Services Architect	Software Engineer	Test Engineer

Table 5 – List of interviewees
Source: author

List of questions that were asked each of the interviewees:

1) What are your responsibilities?
2) How often do you use version control?
3) What tools are essential for you?
4) What purpose do you use TFS for?
5) Describe your daily routine process with TFS. How much does this process vary based on the tasks that you're performing at that day?
6) Would you be willing to change TFS to another system, provided the alternative had the same capabilities?
7) What are your thoughts on centralized/decentralized version control? Which would do you think suits the company and why?

The interviews were conducted in person, on the company premises. In case that prepared questions didn't provide enough insight, additional questions were asked, depending on a particular situation.

Question number 7, however, turned out to be a lot more discussion provoking. Result and brief summary of these discussions are provided in the separate section ("Centralized approach").

Interview results

Needless to say that answers varied heavily depending on the position of the employee and the department.

Architecture department seemed mostly interested in the viewing and comparison capabilities of the TFS. Their daily routine consists mostly of analysis of requirements and business logic implementations in the form of diagrams, class interface/method logic descriptions and so on. Result of their work is a technical specification that can be used by the programming department to implement an actual working solution in code. About 20% of their time at work they spend in consultation with programmers and testers. They often use version control to look at how particular design decisions of theirs were implemented or to make necessary changes to their diagrams, in case that implementation

had to use different logic than was initially described in the technical specification they provided.

As it was expected, programmers turned out to be most interested in version control capabilities of TFS as well as active users of its management and bug-tracking parts. Functions that they use on a daily basis are described in more detail in one of the next sections.

Testers turned out to be the only group completely uninterested in version control part of the TFS. They mostly use it to look at their tasks (work items) and to communicate with programmers through these tasks. Testers often create new bugs, provide descriptions of them, attach all the necessary information for the troubleshooting and send these bugs over to Services Control Group.

What was common among these three groups is that they all seemed absolutely disinterested in alternatives to TFS. Their justifications of this opinion, however, varied.

Group leaders of both programming and testing departments said that change of version control tool would mean that more than 10 years of history and bug-tracking would have to be somehow migrated to the new platform, which was simply not worth it, considering the fact that they use TFS in conjunction with other Microsoft tools and are completely satisfied with them.

Programmers also seemed reluctant to the idea of change. Their main programming language and framework are C# and .NET. Their opinion was that TFS provides the best integration of all tools that they would have to use anyway, so there is no point in changing it.

4.2.4 Daily routine

Let’s take a look at the typical process of working with Team Foundation Server during the standard sprint in the Service Control Group.

The task is to implement new service counters. A service counter is simply a method that increments a global scope variable that is used for statistical purposes as well as for possible troubleshooting if needed.

In the Figure 11 general task window is depicted, along with additional information.

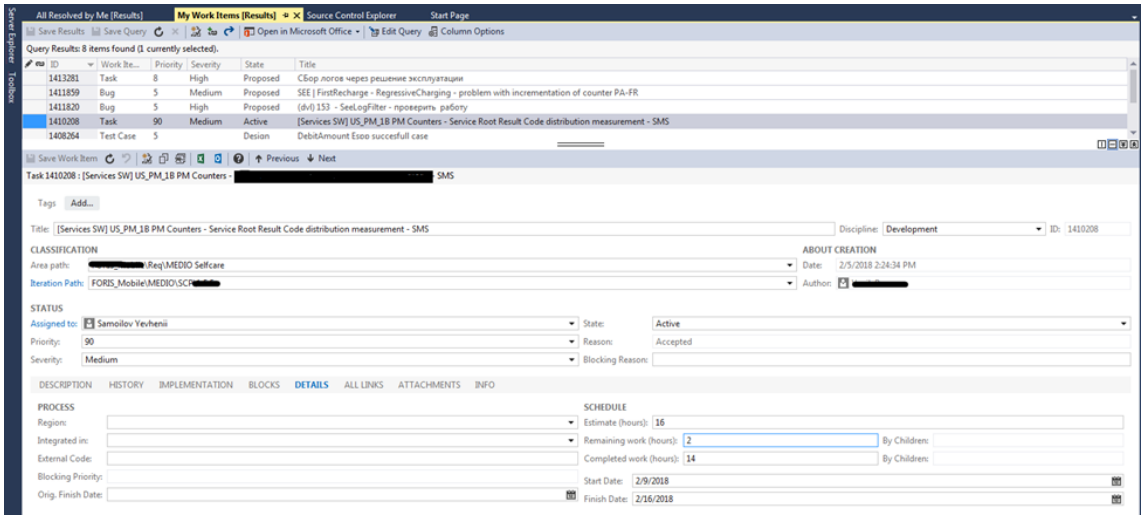

Figure 11 – My Work Items
Source: [author]

After the requirement is passed from the architecture department to programmers, group leader creates a “Lead Work Item” with general description of the requirement in it. For this work item he creates a number of normal work items and splits the work between them. After that he assigns each of them to the programmers who are going to program their parts.

Figure 12 depicts Schedule table of the Details tab of the work item. As can be seen, it contains basic information about the amount of work in hours and approximate start and finish dates. Estimate field is usually filled by a group leader, but in most cases is very approximate and can differ significantly from the actual amount of time spent on the task.

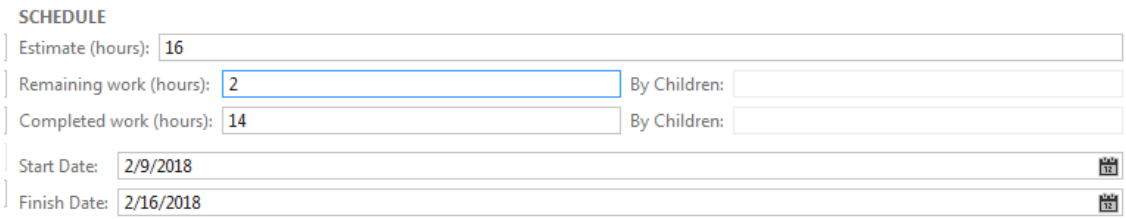

Figure 12 – Schedule of a work item
Source: [author]

In the “History” tab (Figure 13) all the changes ever made that referenced this particular work item are stored. Each change is headlined with its authors name. It is possible to write comments. This feature is useful for questions that inevitably come out during complex implementations and also for feedback and reports.

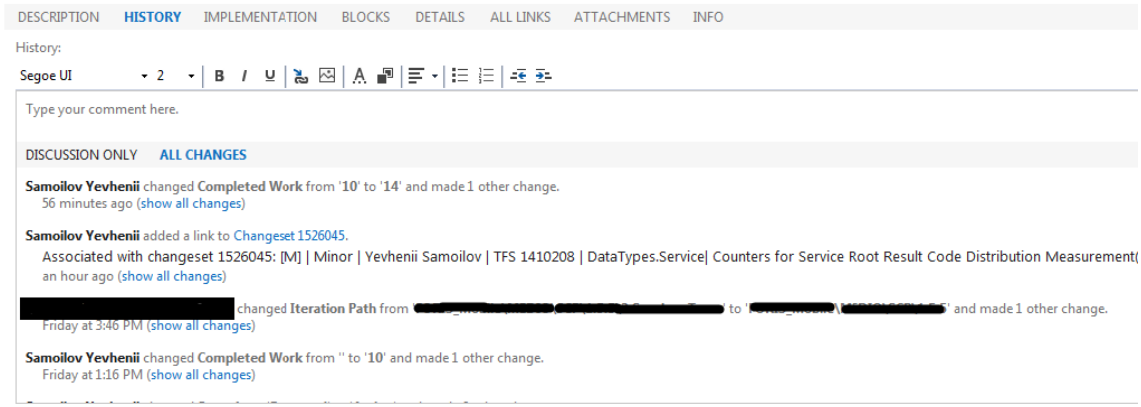

Figure 13 – History tab
Source: [author]

"All links" tab (Figure 14) shows all work items that are somehow related to the current one, as well as change sets that were checked-in.

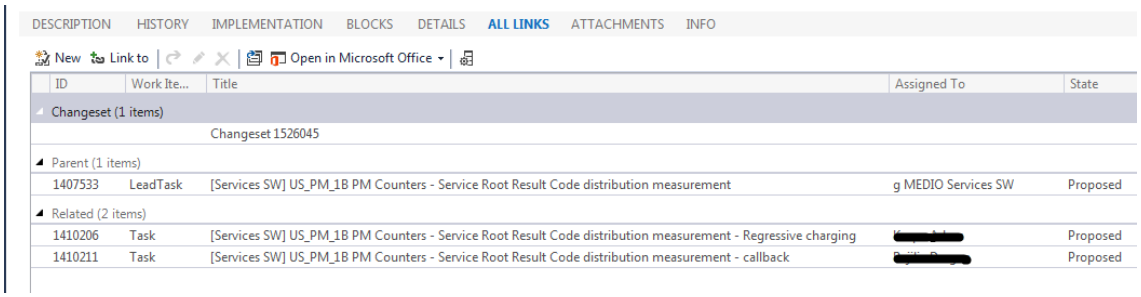

Figure 14 – All links tab
Source: [author]

To start working, first thing that a programmer needs to do is to open the **centralized** repository and get the latest version of the project, which he is going to add his changes to (Figure 15). As can be seen in the figure, other options are also available at this moment, such as checking out of the file, renaming it or discarding the changes that were already made to it, but have not yet been checked in.

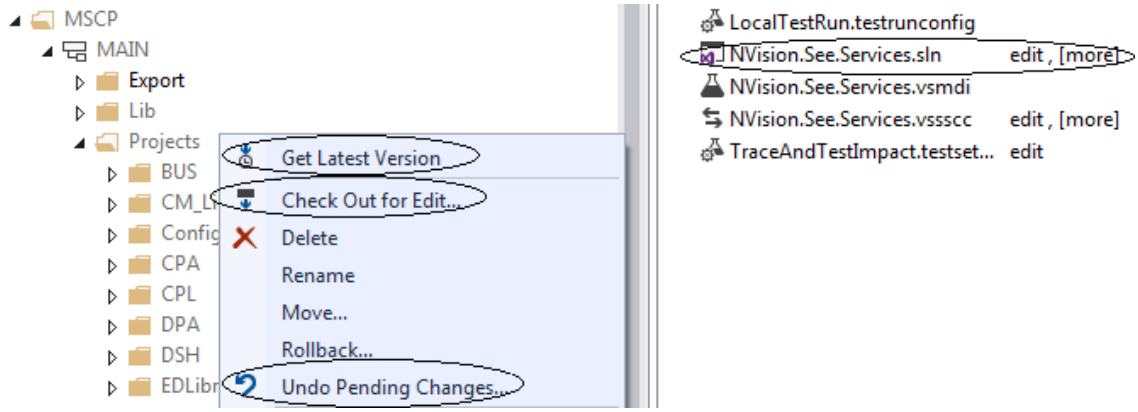

Figure 15 – Repository action menu
Source: [author]

After the latest version of the needed solution has been downloaded to the hard drive, programmer can start the editing process. First thing that need to be done is a file check out. Files can be checked out automatically when the editing process starts or it can be done manually at any moment. Any operation that somehow modifies the solution leads to the check-out of affected files, regardless of what operation was performed (edit, add, delete, rename, etc.).

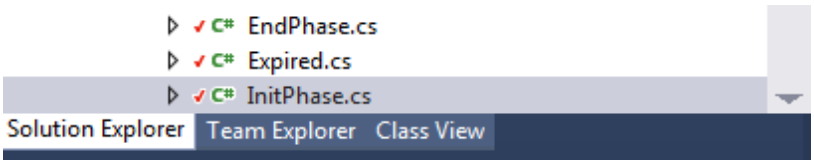

Figure 16 – Files that are checked out for edit
Source: [author]

After the file modification is completed, programmer needs to check in his changes for them to be saved to the repository and be available to others (Figure 17).

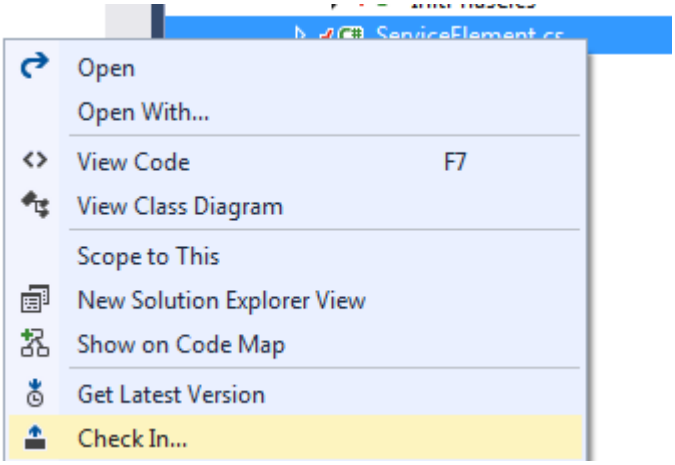

Figure 17 – Check in procedure in the submenu
Source: [author]

When the check in button is pressed Pending Changes menu is shown (Figure 18). This menu allows to select files that user wants to include and exclude from the operation or add specific comments to the change set.

It is possible (in the example it is necessary) to associate it with different work items, so that the change set appears in their history. It makes it extremely easy for any user of the TFS to monitor the work progress on any task. Tracking changes also becomes a very comfortable procedure since information pieces about work items, change sets and specific file histories are all interrelated.

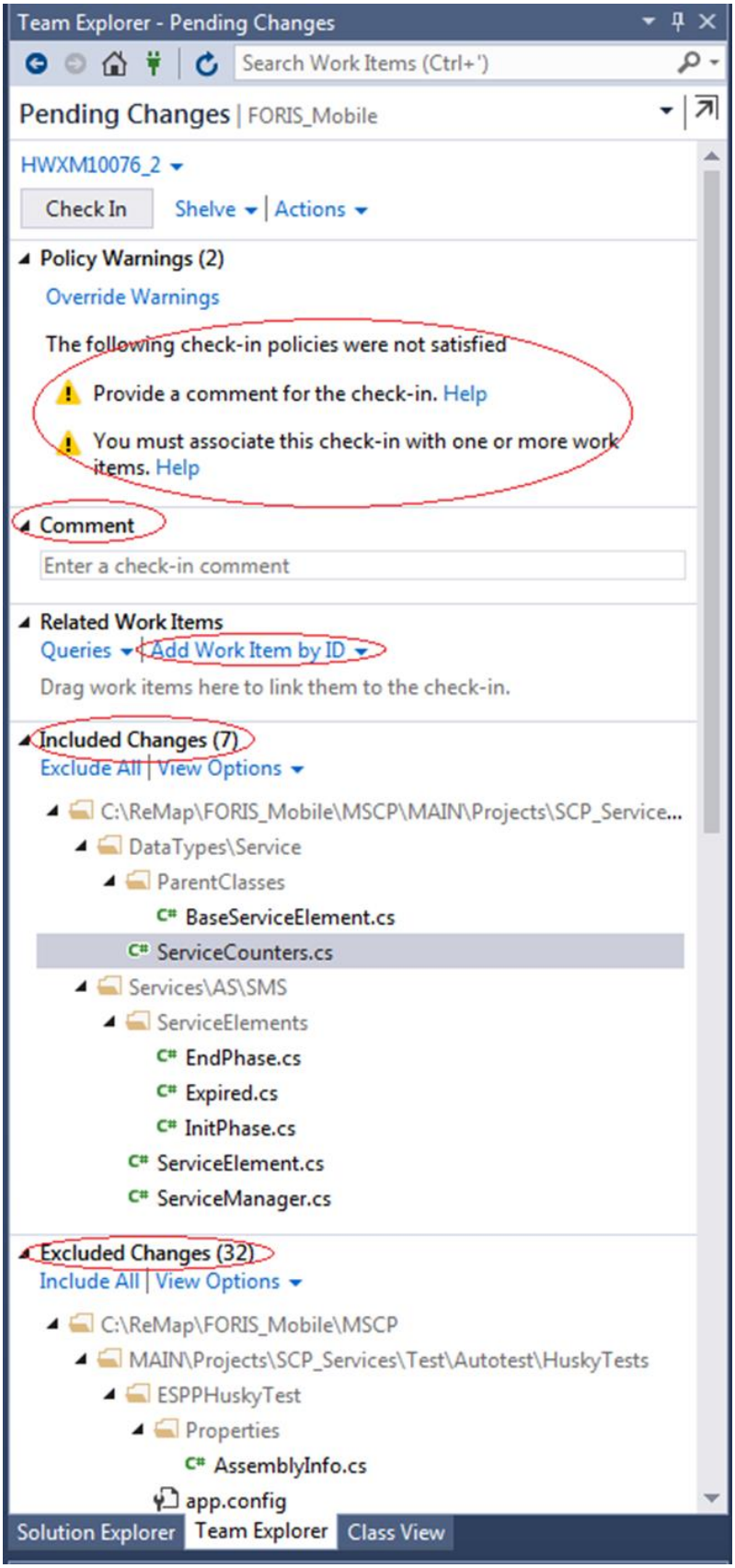

Figure 18 – Pending changes menu
Source: [author]

It is possible to compare different file-version from the Pending Changes menu to make sure that all the necessary changes were made (Figures 19).

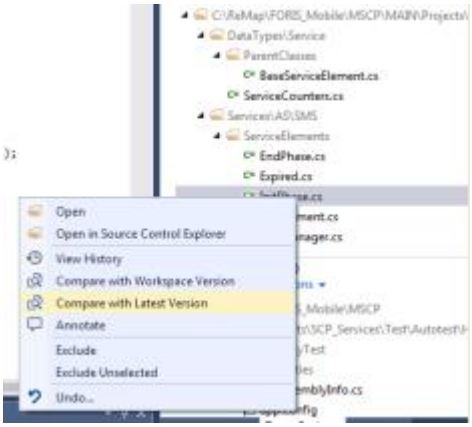

Figure 19 – Comparison of file versions
Source: [author]

Comparison of files in Visual Studio is presented in the split screen mode. On the left side server version of the file shown (Figure 20). On the right side – the local copy (Figure 21). Differences in code are highlighted in red and green, so that it's easier to notice the differences and find them.

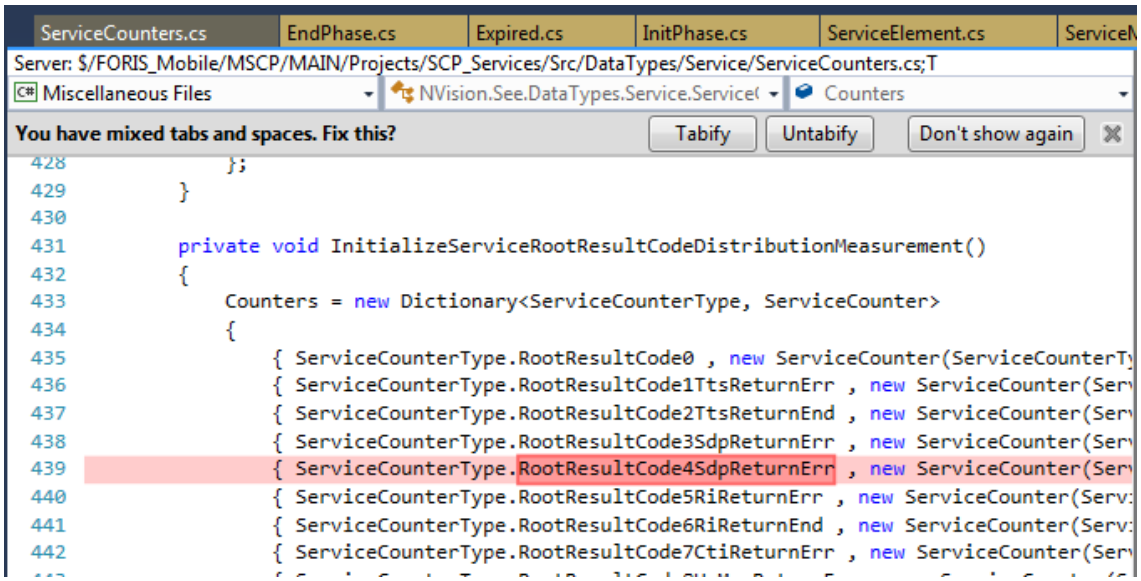

Figure 20 – Comparison window. Left side
Source: [author]

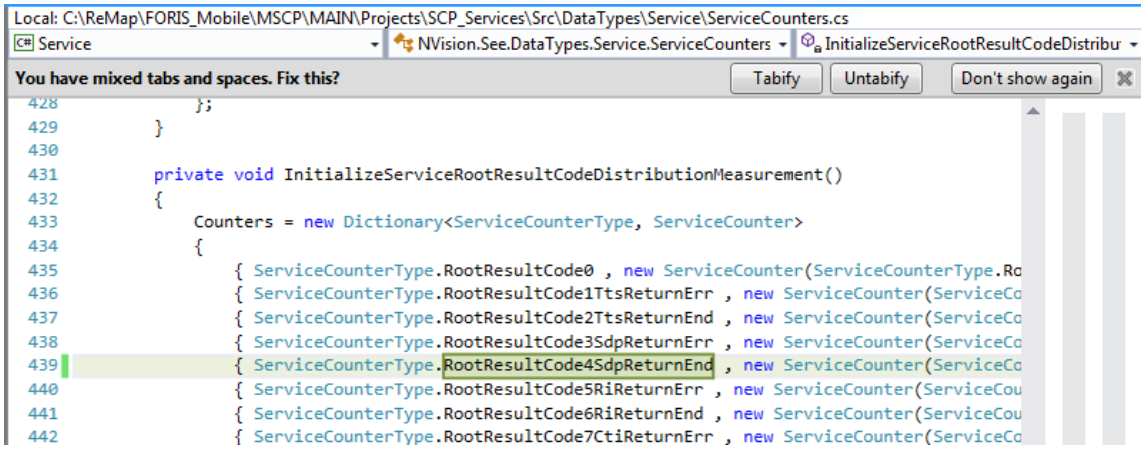

Figure 21 – Comparison window. Right side
Source: [author]

History of each individual file is represented as a list of change sets. It is possible for each individual item to view the details, comments, which files were affected by the particular changeset and how (Figure 22). It is also possible to compare two versions of the file from different change sets. The process is identical to the Local/Server comparison.

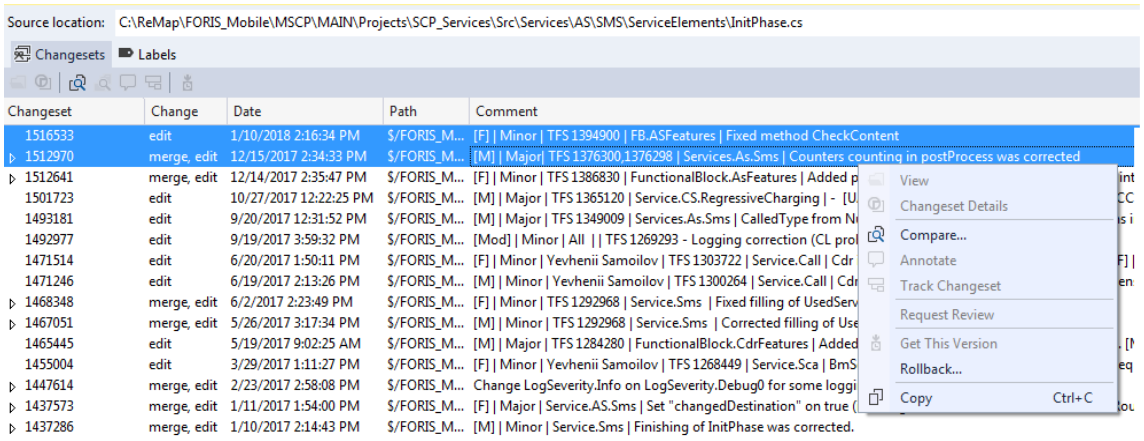

Figure 22 – History of a file
Source: [author]

If the modification need to be transferred to other branches of the product – merging operation is used. Request to merge files is handled by the Merge Wizard (Figure 23).

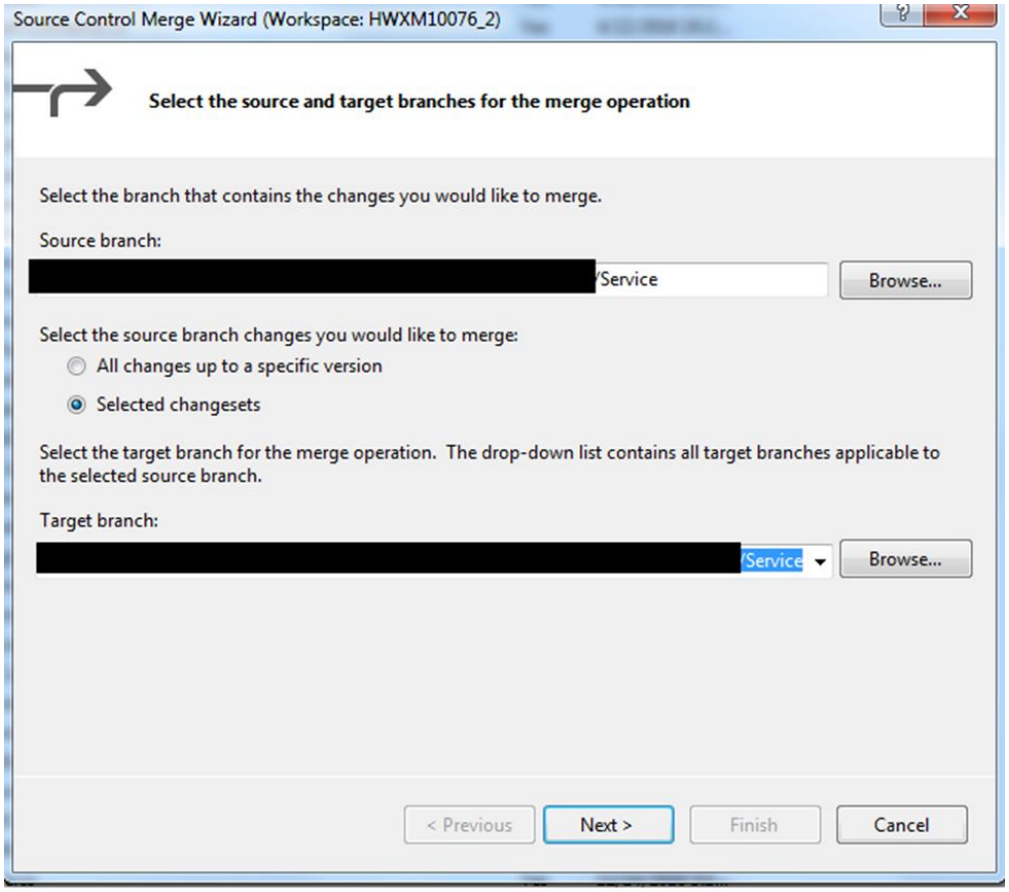

Figure 23 – Merge wizard
Source: [author]

4.2.5 Centralized approach

General monolithic architecture of the MEDIO/OCS product is a huge deciding factor in favor of the centralized approach. Not only distribution of the team and prolonged independent work of its members on their local tasks is not frequently present, it is actually mostly undesirable. Sensitivity of the system modules is rather high which is further proved by the number of regressive testing cycles that are performed before each release and sometimes after.

Integrity of the product is a number one priority and the risk of having to retest many features after unsynchronized changes are made is not something that managers of the product are willing to take on a daily basis. Every department from the development sector is affected by changes, so every change set is result of a collaboration of architecture, testing and programming departments which oftentimes have to work in parallel. It is not uncommon for a programmer to finish one feature just to find out that

part of the functionality that was just implemented is already obsolete, as somebody else had to update it during a critical bug fix.

It is necessary to note that most of the work is done on the premises of the company. Face-to-face communication plays a big role in decision making when architecture and design problems are being solved.

Considering all above said, centralized approach is justified and is both beneficial and convenient for the work style of Nvision Czech Republic. Strict enforcement of "here and now" work policies is perhaps the best idea given the circumstances.

4.2.6 Microsoft

The fact the Team Foundation Server is the best solution at hand for the company is justified by both the fact that company uses Microsoft technologies and C# programming language and the size of the project. Nvision Czech Republic perfectly fits the description of the company that can benefit from this product, provided in the assessment of centralized version control systems section of thesis.

4.2.7 User interface group

Part of the programming department of Services Control Group is a user interface group. It works mostly on utilities for debug and platform analysis that are written on C# and use Windows Forms and Windows Presentation Foundation graphical user interface technologies. They are largely independent from other programmers and oftentimes work from home, as their projects are long-term and smaller in size than modules that are written by service programmers. As this team works with Microsoft technologies almost exclusively and sometimes receive secondary tasks for MEDIO/OCS platform they also have to use Team Foundation Server. But given their nature of work, they are the group that can benefit from advantages of distributed version control systems by utilizing Git integration with TFS.

4.3 Indirect VCS functions

4.3.1 Communication

One of the most important indirect uses of TFS that was discovered during the case study is communication. Part of TFS that contributes to this function the most is Work Items.

During the interviews, employees of Nvision CZ were asked about how they use TFS every day and what functions are most important to them. Work Items turned out to be something that every interviewed employee uses on a daily basis. Almost everyone noted that it is probably the most convenient way of communication with other employees.

To demonstrate how bug tracking is related to professional communication in a software development company, let's take a look at its other most common means and their negative aspects and compare them to Work Items communication.

Face-to-face/telephone conversation

Classic way of human interaction. Its main advantage is clarity and speed of information transfer. Whatever one person wants to ask another it is possible to do immediately and receive a reply almost without a delay. In case information doesn't fully answer the question or additional details are needed, it is possible to express that immediately.

However, there are disadvantages to this medium. One of them is that speed of interpersonal communication implies immediate response. This gives a respondent very little time to compose the answer and control the amount of details that he might want to include. Second disadvantage is the inability of the receiver to reexamine the message after the conversation is finished. For the information to be processed again it has to be either remembered or the conversation must be repeated.

Email

Communication through email possesses both described advantages and disadvantages of face-to-face/telephone communication but in reverted manner.

Its main advantage is the ability of a sender to compose, edit and rehearse the message before sending it. Its another advantage is the possibility to make the information look presentable and include many additional information sources such as documents, links, pictures or previous emails. Receiver of the message can also revisit the information at any time.

Its main disadvantage is that of a possible significant delay as it may take time for a receiver to notice the message, understand it and then to compose the reply. Further, if the reply doesn't fully answer the question or is not comprehended by the inquirer the process has to be repeated.

Communication through TFS using Work Items

This communication medium is somewhat similar to email. It possess the same advantages as the email, however, it greatly improves on them.

Its main improvement from email is the ability to manage large amounts of *task-specific* data. Everything in Work Items revolves around concrete entities of bugs, tasks or requirements. Information that is written in task history is task specific as are all the additional attachments. Let's demonstrate that on an example.

In one of the scum teams of Services Control Group there is a programmer who works on a complex task. The task is to implement a new logic of a specific service.

After the implementation is finished, programmer compiles an executable library of the service that he's prepared and sends it to the tester from his scrum team.

The time that tester may need to prepare the testing platform and check the solution using test-cases may vary, depending on the complexity of changes that were implemented. That means, that the programmer can now switch to a different task or a bug, assigned to him by his scrum-master.

It often happens, that both the programmer and the tester are working on two bugs or tasks in parallel. When one is implementing the changes, another is testing how the previous problem was resolved. In order to communicate all the necessary information through email, a number of chains of emails must exist. As names of bugs or tasks do not always reveal the actual content it may be difficult for both programmer and tester to

follow these chains. If for some reason, the consultation with an architect is needed, it adds additional copies of emails to the conversation.

This problem can be resolved easily if team members use Work Items to communicate. Every work item has two fields that reference people: “Assigned To” and “Created By”. By changing the “Assigned To” field, user can transfer a work item to anybody else connected to TFS. That means that information about the task and its current state will always travel between participants as a whole, rather than brief excerpts from conversations.

Anybody who has a work item number and connection to TFS can get the information about its current state, who is working on it and for how long, see all the associated files and history.

Although largely a matter of taste, one disadvantage of this approach compared to email can be noted. Information in history is not as viewable as in email, so it is generally a better practice to keep history messages short and to the point.

To summarize, it is suffice to say that usage of Work Items for communication is a better practice and should be used on a regular basis when the work on the task is in progress. Email communication should be limited to the matters that are not directly related to tasks or are more general and expanded or require more space and presentation.

4.3.2 Learning

Another important indirect function of VCS that was discovered in the process of case study is learning.

Separate interviews were conducted with junior programmers from Services Control Group to find out their views on source control usage in the company. They were asked the same questions as others, but the results were somewhat different.

Five programmers were chosen for the interview. Their age varied from 23 to 35. All five of them didn’t have any professional programming experience prior to their employment by Nvision CZ. Most of them, as they themselves have put it, came to “learn on a job”. This however, presented them with some difficulties.

It is worth mentioning, that all five interviewed juniors had a very different educational and professional backgrounds. Although none of them had much programming experience, some took IT classes in the university and had a good insight into how IT—industry works. Two reported having developed Microsoft Windows Forms programs before, three said that their main programming experience was with MVC (Model-View-Controller development model) websites for their school and/or friends or family member businesses.

Based on their feedback, it could be said that the company has a somewhat underdeveloped employee enrollment procedure. All five programmers described their first weeks in the company as "a bit chaotic". In their own words they were "thrown in the water to learn how to swim". That meant that the first few weeks of their work they had to perform tasks without full or sometimes even partial understanding of how their work contributes to the development or how it may affect other system components. This situation was further complicated by the fact that oftentimes they had to implement or change parts of complex telecommunication protocols for which they couldn't find a lot of beginner-friendly documentation in company's archives.

Group leaders of the Services Control Group, when asked about the problem, appeared fully aware of it. They expressed their concern towards the situation and said that the solution to this is currently being worked on and includes more user-friendly technical specification, numerous presentations about company's procedures and methodologies and that they try to improve the situation as best as they can by trying new introduction technics with every new employee. Their explanation of why this problem came to existence in the first place is also quite interesting and relates to the nature of work that company currently performs as well as to numerous historical reasons. It is provided below.

Problem roots

The reason why introduction of new workers happens to be problematic at this stage of company's existence relates to the fact that after a number of acquisitions and budget cuts, Services Control Group of Nvision CZ is basically working on a number of versions of one large project. This project, as already been mentioned, is MEDIO IN/SCP platform.

The reason why it is so difficult for new employees to get acquainted with the source code is because most of it is already written. It may seem that that supposed to make their job easier, as most of their time they spent modifying features that already exist and work, but in reality it complicates things because from the global perspective, the MEDIO IN/SCP project is already in the maintenance phase as it has been for years. That means, that there exists a certain way about how things are written in source code and that every new employee has to learn his way around a gigantic code base before he can start making meaningful modifications to it. Without a proper beginner-friendly documentation this task becomes tiresome and difficult, as junior-programmers have to go through thousands of lines of code and pages of documentation in search of the part they have to modify or rewrite.

The size of the project however is so big, that its content and features have their own development lifecycles, which can naturally be at different stages at any given moment. When new programmers join the team they become a part of these development cycles, which not only may already be in one of the final phases, but are in turn a part of even bigger software solution.

Another problem that arises is the fact that every programmer has to work with all parts of the source code. There is no division of responsibilities. This management decision is justified by a number of facts, such as:

- workload on different parts of the project is not constant, therefore having programmers that specialize only in small parts of the platform would be a waste of recourse

- critical bug fixing is one of the major daily tasks and oftentimes has to be performed within 1-2 hours and it's impossible to guarantee that the employee who would be specializing in the particular part of code is going to be available

Programmers' solution

Given the problem described, it was very interesting to find out how these programmers overcame their initial obstacles. As they said themselves, they don't experience any of the above described problems anymore.

All five junior programmers noted that despite the lack of proper introduction procedures atmosphere in the team is extremely friendly and supportive. Most of their questions about telecommunication protocols were answered and explained by architects, who based their solutions on these protocols. Their co-workers from Services Control Group were also extremely supportive, spending as much as half of their working time during the first weeks of the employment explaining the programming concepts used in the source code.

Each of them were asked which tool helped them the most while they were performing their first tasks and getting acquainted with the solution. Their answer was unanimous – TFS.

The way that each particular junior programmer was using TFS to learn varied. However, based on their answers, we can show what tools of version control they used the most and find out how they helped them. General learning technique can also be described and used later as a part of the Services Control Group new employee introduction procedure.

Main learning tools

Three main sets of features distinguish VCS as a useful learning tool. They are:

- timeline

- commentaries

- comparison tools

Let’s take a look at these features in more detail.

Version control systems are capable of tracking every change that has ever been made by anybody who worked with the source code. That means that for each individual file it is possible to see a history of changes made to it. This history of changes usually contains a link to the change set in which this particular modification was implemented and the name of the programmer who made that change.

It can help a new programmer to learn in a number of ways. But the most obvious is to track change sets which contained similar modifications to the one that is required to implement.

Let’s take a look at the example from the daily routine section as if performed by a junior programmer.

1) New programmer receives the task to implement a portion of new service counters for the solution.

2) First thing that the programmer can do is take a look at the class which contains logic where other service counters are utilized. Then he can take a look at the history of changes that were made to the file (Figure 20).

3) Using *Commentaries* column, he can easily find the change set where another service counters were added in the past and then compare this version of the file with the previous one using comparison tool of Visual Studio (Figure 20).

4) Then he will immediately see the portions of code that were added during the implementation of service counters. Programmer may also choose to consult his coworker who made that change (Figure 18, 19).

5) Programmer may also choose to look at the work item linked to a change set to see how the task under which the feature was implemented resembles or differs from his current one (Figure 20).

6) This information the programmer can use while performing his own task.

General learning routine

It is possible to develop a series of tasks for new employees which may help them to get acquainted with the source code much faster and gain a better understanding about the structure of the solution during their first weeks in the company. In order to do that, three main components of learning through TFS have to be utilized – timeline, comparison tools, commentaries. Proposed description of such procedure can be performed with various deviations which group leader of new employees may see as necessary.

First, feedback has to be gathered from programmers about which tasks they perform most often. Opinions of the group members who have not yet worked in the company for a long time (<1 year) are especially valuable, since they can provide a clear

and fresh insight as to what information would have been more useful at the beginning of their employment and which can be postponed until later stages.

After that, implementations of such features must be found and numbers of change sets which contain them extracted from the TFS history. If no clear or easily understandable example can be found it has to be created in the form of reimplementation of an already existing feature.

When the necessary amount of change sets and implementations is gathered, special tasks which require the programmer to make something similar, but with some alterations have to be created. These tasks can be performed in a specially created for educational purposes branch of the product or in the real production version, depending on the level of progress shown by the employee.

Learning by experienced members

Junior programmers are not the only category who can benefit from learning possibilities of VCS. As it was mentioned earlier, version control systems are capable of tracking every change that has ever been made by anybody who worked with the source code. That itself makes them a very useful and agile learning tool.

It was already shown on the junior training example that VCS can be effectively used as a shared knowledge base by various users with different levels of experience. But it can also be used by seasoned professionals to improve their coding styles by revising their old implementations and monitoring their progress throughout their employment with the company.

An experienced senior programmer may want to take a look at his code that is several years old to compare how his coding habits changed and use the knowledge gathered during various problem-solving attempts to determine which solutions worked well in the long run and which did not.

5 Results and discussion

Analysis provided in the practical part of the thesis along with the case study demonstrate situation on the market of version control systems.

Firstly, author describes four most common version control systems on the market. As it is shown in the comparison of each type of source control software, there is no ultimate answer as to what system is the best, nor can an objective comparison be made between each of them, since most of the selected system have their own users.

While worldwide tendencies mostly support distributed version control as a newer and superior form, centralized VCS still to this day find their usage and sometimes are a better, or simply the only, choice for companies with strict centralized management and big monolithic products that helps to maintain integrity of software components.

In the case study, author makes a detailed analysis of a Czech software development company and how its development sector uses Microsoft Team Foundation Server on a daily basis. Daily routine with it is also described.

Also shown in the thesis are indirect VCS functions that were discovered during the case study that are not often thought or spoken about on a daily basis in developer communities, but that are of direct help and usage to most programmers who use version control in their work.

The author then identifies three types of work related communication and shows how they are different and what advantages and disadvantages every one of them possesses.

Then, learning features of version control are shown. It is explained, how junior programmers can benefit from different history tracking capabilities of source control and how those can be utilized to develop a new employee training procedure. Author also notes on how senior programmers can benefit from these methods in their work. As mentioned in the article “Strategic Importance of Building an Enterprise Training Program” written by Kenny and Company consulting company[24], a well-developed procedure of this kind can be of great benefit to the company in general and help it to establish itself throughout the industry. Such methods are of great value to big companies with large staff.

6 Conclusion

Goals of the thesis are to perform an analysis of the existing version control systems and their role in the software development process, to create general overview of the current version control system solutions, their advantages and disadvantages, to analyse usage of Microsoft Team Foundation Server as a source control tool, to evaluate version control systems from a perspective of corporate software development.

In the first chapter general information about source control and why it's used by developers is presented. Next chapter explains objectives and methodology in details.

Literature review consists of terms that are necessary for understanding of version control software work principles, description of different types of this software and describes typical work process with it. Software life-cycle definitions and its variations are also described and are used in the case study.

Practical part describes several of the most popular today version control systems. Analysis of their advantages and disadvantages is one of the crucial points of the chapter as it shows how direct comparison might prove itself ineffective when it comes to source control software and how the type, rather than concrete product, is much more important when the decision is being made about which VCS a company should use.

Case study is a reflection of authors work experience. Microsoft Team Foundation Server usage by the development sector of a Czech telecommunication software company is described and evaluated. Company's management policies and development life-cycle transition outcomes are included, along with interview results from different employees on order to prove that Team Foundation Server is currently the best choice for the company on the market.

One of the most important parts of the thesis are indirect VCS functions. Detailed explanation as to how those findings can be used in order to improve development practises are given in the last section of the practical part. Although applicable to any version control software, they are perhaps of a bigger value to a large company with smaller number of bigger products similar to Nvision Czech Republic.

7 References

1. **Sink, Eric.** *Version Control by Example.* Pyrenean Gold Press, 2011. 978-0-9835079-1-8.
2. **Best version control systems in 2017**. g2crowd.com. [Online] [Cited: 06.08.2017.] https://www.g2crowd.com/categories/version-control-systems.
3. **Getting Started - About Version Control.** git-scm.com. [Online] [Cited: 08.08.2017.] https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control.
4. **What are the Software Development Life Cycle (SDLC) phases?** istqbexamcertification.com. [Online] [Cited: 16.08.2017.] http://istqbexamcertification.com/what-are-the-software-development-life-cycle-sdlc-phases.
5. **What Is Scrum In Agile Methodology (Agile Development).** www.yodiz.com. [Online] [Cited: 22.08.2017.] https://www.yodiz.com/blog/what-is-scrum-in-agile-methodology-agile-development.
6. **Úplný výpis z obchodního rejstříku. NVision Czech Republic a.s.** or.justice.cz. [Online] [Cited: 03.09.2017.] https://or.justice.cz/ias/ui/rejstrik-firma.vysledky?subjektId=71444&typ=UPLNY.
7. **IT services.** kommersant.ru. [Online] [Cited: 06.08.2017.] https://www.kommersant.ru/doc/2446438.
8. **List of the largest consulting groups in Russia in 2013.** expert.ru. [Online] [Cited: 08.09.2017.] http://expert.ru/ratings/spisok-krupnejshih-konsaltingovyih-grupp-rossii-po-itogam-2013-goda.
9. **Companies of the Russian IT market**. kommersant.ru. [Online] [Cited: 08.09.2017.] https://www.kommersant.ru/doc/2446436.
10. **Rusyaeva, Polina.** *MTS announced the acquisition of one of the largest Russian IT companies*. rbc.ru. [Online] July 17, 2015. [Cited: 05.09.2017.] https://www.rbc.ru/technology_and_media/17/07/2015/55a917359a79470e8f603917.
11. **NVision Group.** tadviser.ru. [Online] [Cited: 31.10.2017.] http://www.tadviser.ru/index.php/Компания:NVision_Group_(Энвижн_Груп).

12. **MTS is integrating losses of Sistema.** kommersant.ru. [Online] [Cited: 01.11.2017.] https://www.kommersant.ru/doc/2769365.
13. **Annual Report 2015 MTS.** [Online] [Cited: 01.11.2017.] http://kgo.rcb.ru/2016/otchet/mts_2015_rus.pdf.
14. **Nvision Czech Republic official website**. [Online] [Cited: 31.10.2017.] http://www.nvision-ems.cz.
15. **The Definition of OSS and BSS.** ossline.com [Online] December 5, 2010. [Cited: 25.09.2017.] https://www.ossline.com/2010/12/definition-oss-bss.html.
16. **Shunxing Chen, Linfeng Yang**. *Intelligent Networks.* [Online] March 23, 1999. [Cited: 25.09.2017.] http://www.tml.tkk.fi/Studies/Tik-110.300/1998/Essays/in_2.html.
17. **Online Charging System.** dialogic.com. [Online] [Cited: 26.09.2017.] https://www.dialogic.com/glossary/online-charging-system-ocs.
18. **Service Control Point.** dialogic.com. [Online] [Cited: 27.09.2017.] https://www.dialogic.com/glossary/service-control-point-scp.
19. **Real-time Charging.** nvision-ems.cz. [Online] [Cited: 27.09.2017.] http://www.nvisioncz.com/ict/solutions/real-time-charging.
20. **Custom Application Development**. star-technologies.co.in. [Online] [Cited: 22.08.2017.] http://star-technologies.co.in/software-development.
21. **What is Waterfall model- advantages, disadvantages and when to use it?** istqbexamcertification.com. [Online] [Cited: 22.08.2017.] http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it.
22. **Agile vs Waterfall Differences in Software Development Methodologies**. yodiz.com. [Online] [Cited: 26.08.2017.] https://www.yodiz.com/blog/agile-vs-waterfall-differences-in-software-development-methodologies.
23. **Team Foundation Server.** visualstudio.com. [Online] [Cited: 12.11.2017.] https://www.visualstudio.com/tfs.
24. **Strategic Importance of Building an Enterprise Training Program**. michaelskenny.com. [Online] [Cited: 15.12.2017.] https://michaelskenny.com/points-of-view/strategic-importance-of-building-an-enterprise-training-program.