SOFTWARE PROJECT MANAGEMENT

Lesson 4

ABOUT SCRUM IN DETAILS

CONTENTS

1. WHAT IS SCRUM? WHY SCRUM OCCURRED?

The Scrum approach was developed to improve a frustrating situation related to IT projects in the early 1990s.

The Chaos report of 1995, issued by The Standish Group, presented the following statistics: only 16.2% of IT projects met the deadline and the budget, 31.1% of IT projects were canceled for a variety of reasons. Calculations in this report state that American companies have lost more than $81 billion in the canceled IT projects in 1995, and the initial budgets of the completed IT projects have gone over $59 billion. (https://www.standishgroup.com/sample_research_files/chaos_report_1994.pdf).

Members of the teams involved in the development of IT products meditated on the experience they've gained with the waterfall model, they wanted to find other approaches to project management. The article "The New New Product Development Game" by Hirotaka Takeuchi and Ikujiro Nonaka was published in 1986 in *HARVARD BUSINESS REVIEW*. It described the "rugby" approach (https://hbr.org/1986/01/the-new-new-product-development-game).

The waterfall model of development of IT projects used a sequential approach, when a team works as a relay team passing the results from one stage to another. The approach called "rugby" implies that the team covers the whole distance of the project together, as if passing the ball to each other

The authors of the article examined project management in multinational companies, such as Fuji-Xerox, Canon, Honda, NEC, Epson, Brother, 3M, Xerox, and Hewlett-Packard and analyzed the development process of six products. The result of this analysis was a description of the iterative model of product development and six characteristics in managing the project that, according to the authors, helped the teams to manage the development of new products.

This article introduced a new term **"rugby approach"**, which was renamed in 1991. DeGrace and Stahl referred to it as to **Scrum** method in their book "Wicked Problems, Righteous Solutions".

And in 1995, Jeff Sutherland and Ken Schwaber introduced the world to a structured Scrum approach at a conference.

> **Scrum is an approach that includes a set of principles on which the development process is built, three roles that must be in the project, five types of events, and two documents.**

This set of roles, documents, and events was enough to implement IT projects and create good IT products.

A document that describes how Scrum works is called **Scrum Guide** and it is available at: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf.

In 2001, adherers of the iterative model gathered in USA to discuss common things their approaches have. This meeting resulted in the manifesto for Agile and 12 principles behind the Agile.

According to the 12th annual State of Agile report, Scrum framework is the most popular Agile approach in the world:

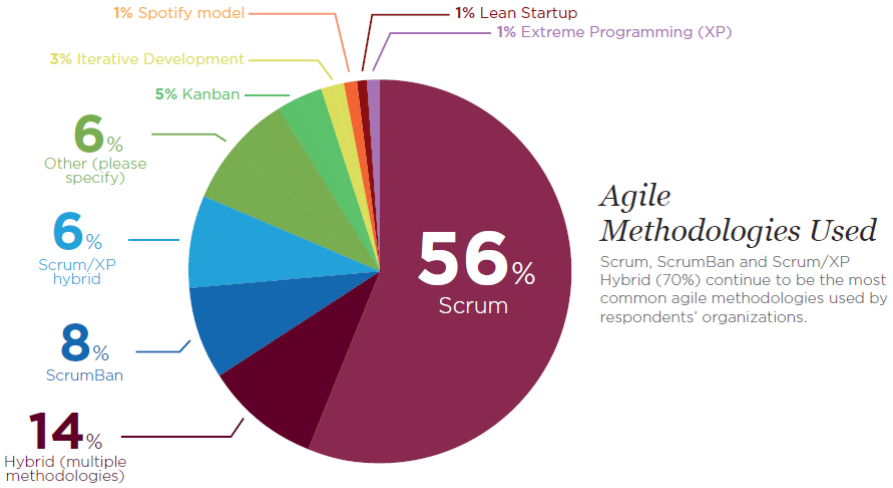

Figure 1

All components of the Scrum are given in the figure below:

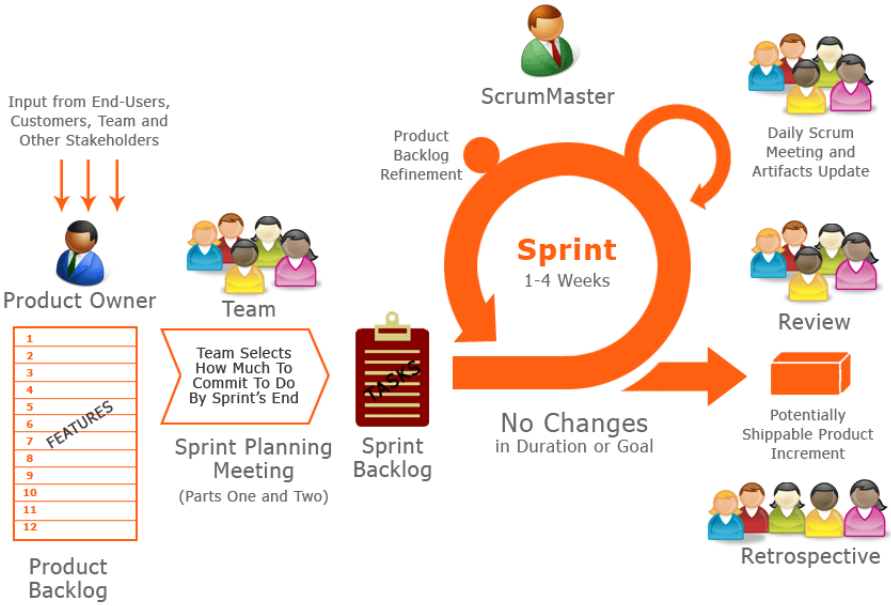

Figure 2

2. ROLES IN SCRUM

The first thing that may surprise you when you start getting acquainted with Scrum is that there is no project manager role. Many people, who take part in Scrum trainings, are bewildered at this point: how can you manage a project if nobody is responsible for the success? We've discussed what success in a project is in the Lesson 2. If the project objectives were achieved within the set deadline and budget, it is considered to be successful.

Scrum Guide describes only three roles: a **Product Owner**, a **Scrum Master**, and the **Development Team**. These three roles are included in the **Scrum team**:

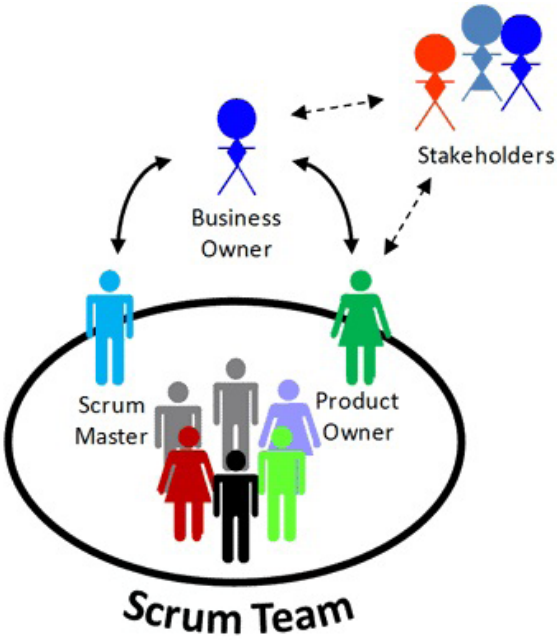

Figure 3

Product Owner is responsible for maximizing the value of the product.

The best person for the role of a Product Owner is a customer representative who is more interested than any other person in achieving results from using the software product created within the project. For instance, a Product Owner for the project "Development of an Online Bank" can be a Director of the Section for Retail Service.

If the customer company doesn't have a person able to take on the role of a Product Owner, the Development Team may choose a **Proxy Product Owner** to do most of the work for the customer representative, secure approval for priorities in the product backlog, and demonstrate the result once a month. A member who has an experience as a business analyst or project manager can be a good Proxy Product Owner.

In order to create the right product, the Product Owner uses a **product backlog** and manages its content. The Product Owner is also responsible for the budget of the project, because they decide on what product items one should spend money of the project and what items are out of budget by managing priorities in the product backlog.

To prioritize items in the product backlog, the Product Owner must develop and adopt an approach to prioritization. The question "Can a product have multiple Product Owners?" the Scrum Guide says "No". Fun fact: the Product Owner may execute some items from the product backlog, but more often than not they.

Tasks of the Product Owner:

- Focuses on “What?” instead of “How?”;
- Responsible for cost effectiveness indicators of the project;
- Approves product requirements;
- Prioritizes product requirements;
- Participates in Sprint planning;
- Accepts the work of the Sprint.

Mission of the Scrum Master is to create an effective team and teach it the Scrum approach.

In order to be an effective Scrum Master, you have to be skilled in coaching, meeting facilitation, and group dynamics management. If a candidate doesn't have the appropriate skills, Scrum may stall or be a failed experiment. So the best solution for a team that uses Scrum approach for the first time is to invite an experienced Scrum Master to teach all team members how to work in the new paradigm and teach one of the team members how to work as a Scrum Master.

Alternatively, an experienced Development Team member, who worked as a Scrum Master before, wants to take on this role and combine it with the role of a team member.

Scrum Master helps the Product Owner to:

- adopt best practices for prioritizing in the Product Backlog;
- convey the product vision to the Development Team;
- develop the team's skills in correct formulation of items in the Product Backlog.

Scrum Master for the team is an unofficial leader, who helps to grasp Agile values and principles, adopt Scrum events and documents.

Scrum Master helps the Development Team to:

- teach the Development Team members self-organization, time management, estimate complexity of the Backlog items and tasks;
- clear obstacles that prevent the Development Team from gaining in productivity;
- facilitate Scrum events.

A facilitator is required for all Scrum events in order to reach the goal without exceeding the allotted time. The facilitator directs the discussion flow and adds four values to the discussion:

1) full participation;
2) mutual understanding;
3) mutually acceptable solutions;
4) joint responsibility.

The Development Team in Scrum must be responsible for execution of all tasks planned for this Sprint. The Development Team is efficient if it corresponds to the following characteristics:

- There are no other roles in the Development Team other than the role of a developer.
- The team is cross-functional, meaning that all specializations, required for the creation of a product, must be presented. Inside the team, there may be specializations of labor, but the team is responsible for the result. Many teams stall when specialists in the team are narrow focus specialized.
- The Development Team is self-organizing. This means that

the Scrum Master and the Product Owner should NOT tell the team how to do the job and nominate performers within the Sprint. It is common practice for each member to choose the tasks they will work on within the Sprint.

- The Development Team doesn't have any sub-teams. And the size of the team doesn't go over nine people.
- The Development Team has a collective responsibility for the Sprint result.

3. DOCUMENTS IN SCRUM

Scrum minimizes the number of documents required for reaching the objective of the project. Thus, only two documents are used in Scrum:

1. **Product backlog**.
2. **Sprint backlog**.

Product Backlog

Product Backlog is a document that includes everything to be executed by the Development Team. The backlog also contains error messages, ideas about new functions, and so on.

There are two types of product requirements in Scrum: **epic** (requirements cannot be implemented within one Sprint) and **story** (requirements can be implemented within one Sprint).

Story is described as follows:

Being a <user type>, I want to <specific objective> in order to <specific reason>.

How to Create and Manage a Product Backlog

Scrum team works with requirements orally rather than in writing. Although the Product Backlog has written requirements, the team writes each of them in one sentence instead of giving a detailed description of each item (like in SRS). As soon as a user story goes up in the Product Backlog so that it is highly possible that it will appear in the nearest Sprint, the Development Team decides who will work on this story, and this person asks questions about it during the Spring Planning.

A Product Backlog is never complete. The earliest development of it lays out the initially known and best-understood requirements. So, to create the first version of the Product Backlog, the Development Team and the Product Owner just need to extract business requirements, range them, and elaborate the most important to make user requirements out of them (according to the Wieger's classification).

Let's look at characteristics of a good Product Backlog:

- **Optimum level of details in user stories.**

Stories in the Product Backlog that will appear in one of the nearest Sprints must be described and estimated so well that the team can implement them within the next Sprint. Stories that will be implemented in three Sprints or later must be described in less details.

- **Effort estimates.**

Backlog items (stories, epics, errors) at the bottom of the Product Backlog are not worked through by the analyst, as a rule. So these estimates are less precise compared to the estimates at the top of the Backlog.

- **Ever changing**.

The Product Backlog changes as the product develops: it has new user stories, some stories are deleted, priority of the story's changes.

- **Backlog priority.**

Product Backlog must be sorted so that the most important items are at the top of the list, and less important at the bottom. It is assumed that if the Product Owner has set the correct

priority of the items, the team can maximize the value of the product or system.

Requirements can be extracted and analyzed by both the Product Owner and the team, but the priority is a responsibility of the Product Owner only.

Scrum Team decides how and when the Product Backlog is to be refined to prepare stories for the next Sprint Planning: refine and estimate requirements to stories and other Backlog items. This activity usually takes no more than 10% of the time the Scrum Team has.

The Product Backlog structure may be as follows:

Name	Priority	Estimate	How to show	Notes
Being a car driver, I want to listen to my letters from the specified email as I drive	100	40	A driver guns the engines, presses the email button on the display, then the Play Email button. The computer reads messages from email box configured before.	Selection of multiple email boxes is out of scope for now
Being a car driver, I want the computer to tell me when I change lanes to avoid falling asleep while driving	99	50	The driver turns on the button"Inform about lane chang"; while driving on the high road, he twists the wheel and the car changes lanes. The computer informs: "You have changed the lane."	

The **Priority** box is filled by the Product Owner, the **complexity** is estimated by the Development Team. This

estimate is considered to be initial because it can be refined during Sprint Planning.

A Product Backlog exists until the product is ready and accepted by the Product Owner.

If two or more teams work on the product, all future works are described in the same Product Backlog.

How to Estimate Tasks in a Product Backlog?

There are several ways for initial estimation of items in the Product Backlog. One of them is **Planning Poker**.

The Development Team, a Scrum Master, and a Product Owner gather in one room. The Product Owner introduces the Team to stories with the highest priority in the Product Backlog. Each team member has a special deck of cards. You can use a card deck for Planning Poker showing Fibonacci sequence.

Figure 4

Not all numbers correspond to the Fibonacci sequence in this deck. For example, 1/2, 20, 40, and 100 are not in the Fibonacci sequence since they are not a sum of the two previous numbers. There is a ? card that means that the participant doesn't understand the story and cannot estimate it, and a coffee cup card means that the participant is tired and wants to take a break, a 0 card means that the item is already implemented.

The Product Owner introduces the team to the story with the highest priority. Team members ask questions and the Product Owner answers them. After this the team members split the story into tasks and estimate each task. Participants agree on the units of effort estimation: **ideal hours** or **story points**.

> **Story points** are abstract measures of effort required to implement a user story. At estimation, the team compares this story to a baseline story assigned 1 story point. If your story is three times more complicated than the baseline story, it gets *3 story points*.

Each member thinks how much time they personally would need to implement the task and then they take the card with the corresponding estimate. But, in order to avoid an anchoring effect, the participants put them face down. For instance, one member assigns 4 man-hour to the task. In this case, they take two cards from the deck: 3 and 1 and puts them face down.

After all members put their cards, they are revealed and the members see the results.

The goal of Planning Poker is to bring together opinions of the members, so the Product Owner asks to speak the member with the lowest estimate and the member with the highest estimate. After these members discuss, requirements are refined as a rule, the best approach to implementation is selected, and the task is estimated for the second time. After the second round, the estimate range narrows. To give a single estimate, calculate the arithmetic mean value and go to the next task.

The team estimates all the stories and tasks in this way.

Sprint Backlog

The document contains tasks and user stories the team selected for the next Sprint. In order to select stories and tasks for a Sprint, the team needs to know the priority of the stories in the Product Backlog, scope of the Sprint, and initial estimates of the Product Backlog items.

Scope of the Sprint is the amount of time all team member have for the next Sprint. Scrum Master evaluates the scope of the Sprint based on the polling the Development Team.

The Sprint Backlog may be as follows:

Tasks	Estimate	Assigned to	IS IT DONE?
Develop a service document (similar to the work order in the old system). Add planned cash inflow to the work order, use the catalog of natural persons	8	Sergei	no
Load the stock-list data from Excel to working DB	2	Maksim	yes

Tasks	Estimate	Assigned to	IS IT DONE?
Fix the bug related to creating a check based on the client's account in the business process	2	Irina	yes

BurnDown Diagram

In order to understand the team manages to implement everything planned for Sprint within the set deadline, you should know how much work you have left before the Sprint ends and how much should be left according to the estimates.

A **BurnDown diagram** is used for this. See the example below:

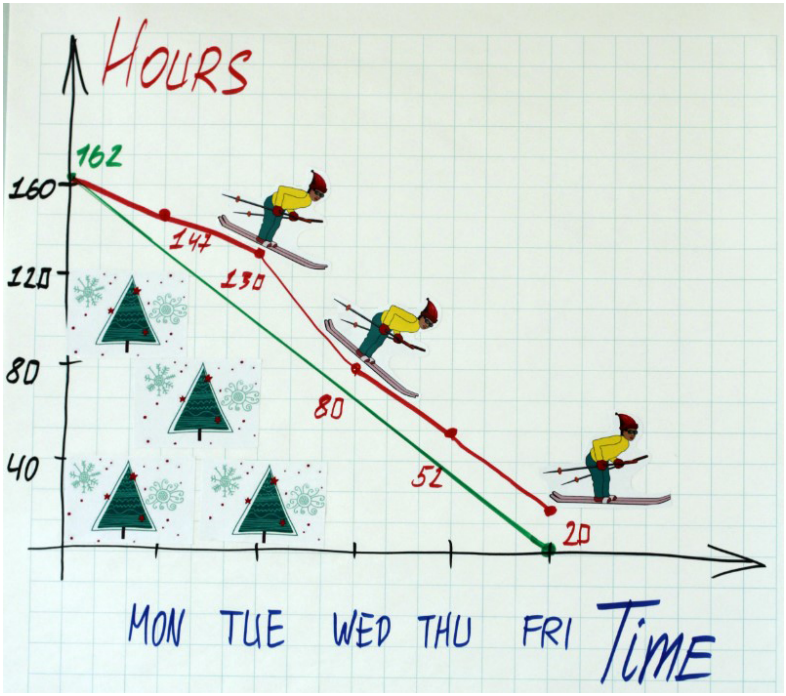

Figure 5

The team planned 162 ideal hours for a Sprint. Green line on the diagram shows what scope of work in man-hours the team should ideally have left every day. Red line shows the real scope of work the team has left. Thus, if the red line is above the green, the team is behind the schedule, otherwise it is ahead of the schedule or sticks to it (if the lines coincide).

If the team members do not get lazy and estimate the scope of work remained by the tasks they have already started, the diagram shows clearly whether the team can do everything planned for this sprint on time or not.

What is a Scrum Board?

The team can use a **Scrum Board** to visualize the flow of tasks and to manage tasks in a Sprint. See an example of such board in Figure 6.

All the stories that have appeared in the Sprint are described on white sticky notes, and white sticky notes have tasks that the story is broken down into (see Figure 7).

Each story on the board has a path, where tasks of this story move from left to right.

There are three columns of the board that describe different states of the tasks: **Not Checked Out**, **Checked Out**, **Done**. As soon as one of the team members starts working on a task, they can move the sticky note with this task to the **Checked Out** column. When the task is completed, the sticky note goes to the **Done** column. This way, all sticky notes gradually move to the **Done** column.

You can place a **BurnDown diagram** next to these columns.

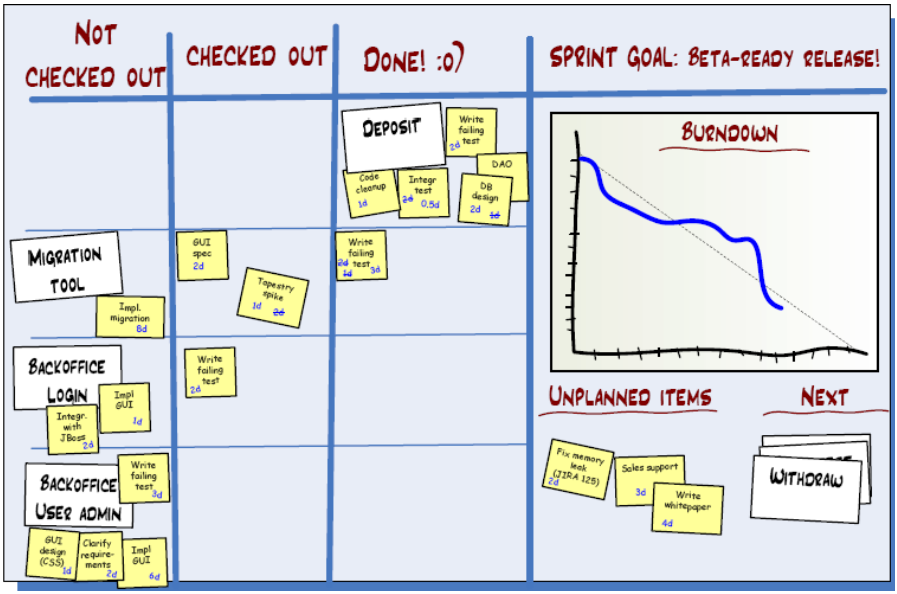

Figure 6

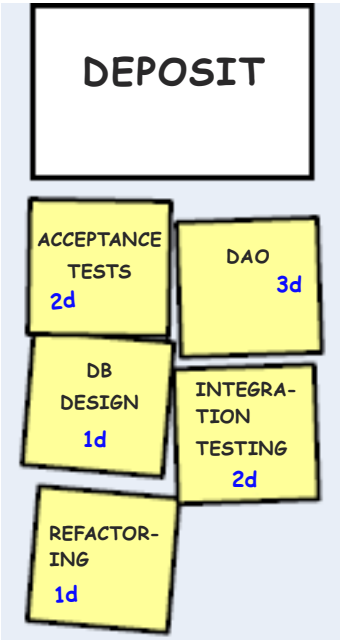

Figure 7

4. SCRUM EVENTS

What Is a Sprint?

Scrum uses the term **Sprint** to denote a time period from one to four weeks allocated for creation of functions of the product:

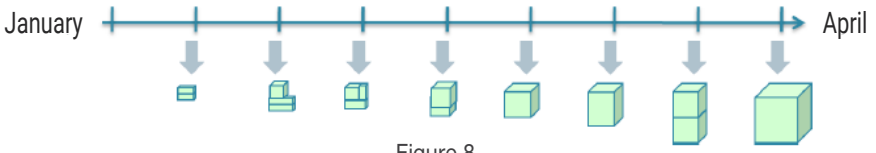

Figure 8

The Product Owner and the team specify the Sprint duration, which does not change as the project is being developed.

Sprint Planning

The result of a Sprint Planning is a list of tasks that the team shall complete within a Sprint. This list of tasks goes to the Sprint Backlog. Sprint Planning, as a rule, consists of two stages:

1. **The first stage prepares the team for Sprint Planning**: epics are broken down into stories, stories are broken down into tasks and estimated.

2. **The second stage is for meeting on Sprint Planning**, where the team decides how much work it can do within a Sprint. To do this, the team must understand scope of the Sprint: how much man-hours all team members hove for the next Sprint and an initial estimate of the task complexity.

   It is recommended to spend 1 to 2 hours on Sprint Planning.

Scrum Master facilitates this meeting, but they can offer this to any member of the Development Team in order for them to become competent in meeting facilitation and to make sure that the team members of the project are interchangeable.

The Product Owner must attend the Sprint Planning meeting in order to:

- answer the team's questions about requirements to a story or an epic;
- decide what to do if one of the Product Backlog items doesn't fit in a Sprint, but the Product Owner needs at least some work on this item to be done the next Sprint;
- ask questions to the members of the Development Team about why the estimate of task complexity is so if the Product Owner doubts its adequacy.

Daily Scrum

This meeting solves the following problems:

- lowers the student syndrome that the majority of people suffer from.

Student syndrome is when humans start tasks as late as possible; the curve below illustrates the intensity of work:

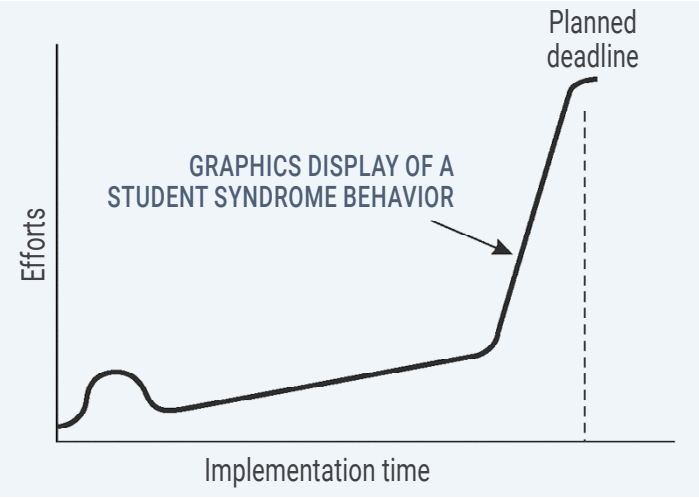

Figure 9

- helps the developer to find a solution to the problem which they didn't discuss with the colleagues trying to to solve it on their own do it all alone.

Each member of the Development Team tells the colleagues what they did yesterday, and what kept them from completing the planned work and receiving the expected result. The mere thought that "If I don't complete at least one task today, I won't have anything to tell my colleagues during the daily Scrum" motivates us to work.

Daily Scrum meeting is held every day, and each team member takes part in it. The team member answers these three questions:

1. What did I do yesterday?

2. What prevented me from completing the work planned for yesterday?
3. What will I do today?

In order for the meeting to take 15 minutes, everybody stands. Scrum Master keeps track of time and makes sure that only general issues are discussed during daily meetings. If there are specific problems, the Scrum Master offers to hold a separate meeting and invites only those team members who are interested in the issue under discussion and who can help.

Sprint Overview

In order to get a quick feedback from the Product Owner on the functions done, the team prepares a **demonstration** (**demo**) of everything done within a Sprint.

The whole team participates in the demo; each member must show their result to the Product Owner, and the Product Owner can refine the requirements, or they may have new thoughts on requirements.

The team does not spend too much time on SRS, it rather works on the product, quickly creates prototypes, and collects feedbacks. It is desirable to set a time limit to the demo: 4 hours or less for four-week Sprints and 1 hour or less for one-week Sprints.

During the demo, one of the members should write down the feedback received from the Product Owner. At the end of the demo, this person shows all the notes to the Product Owner and draws confirmation that everything is written correctly.

Summarizing the results of the demo, the Product Owner adds new items to the Product Backlog and prioritizes them.

Retrospective Meeting

Scrum has an idea of an ever-increasing quality of teamwork related to this methodology.

Retrospective Meetings should be held upon the completion of each Sprint and timbe-boxed to 4 hours or less.

> The Theory of Inventive Problem Solving (TRIZ) describes the **Law of increasing the degree of ideality of the system**: the development of technical systems must lead to an increased degree of their ideality.
>
> This formula was proposed to evaluate the ideality of a system:

$$\text{Degree of Ideality} = \frac{\text{Values} - \text{Loss of Quality}}{\text{Expenses}}$$

> The more values the system creates, the more ideal it is, thus generating minimum costs and minimizing the loss of quality.

This formula results in a counter-intuitive conclusion: an ideal system is a system that doesn't require costs and creates value.

Scrum Master is the person who takes care of increasing ideality of the Scrum process. To do this they arrange and hold meetings called **Retrospective Meetings**.

Three issues are discussed in classic Retrospective Meetings:

1. What worked well for us?
2. What could work better?

3. What actions can we take to improve our process going forward?

You can extend your list of questions for a Retrospective Meeting as follows:

1. Discuss the tasks set before team members in the previous Retrospective Meeting, specifically:
   - What tasks on improvement of our process were completed?
   - What tasks were failed and why?
   - What are new deadlines on the implementation of uncompleted tasks?
2. Discuss actual metric values of a team, for example, velocity and focus factor.
3. What prevented the team from improving the metric values?
4. What new tasks aimed at improvement of the process will we bring up in the next Sprint? Persons responsible and deadlines for tasks related to improvements?

To make your Retrospective Meetings fun, use specially designed games that increases employees' engagement, for example: Timeline, Color Code Dots, Mad Sad Glad, Like to Like (*see more in the booke "Agile Retrospectives: Making Good Teams Great" by Esther Derby, Diana Larsen*)

Example of a product development using Scrum is given below.

1. The Development Team and the Product Owner during the project kickoff meeting decided that a Sprint duration is one week.

2. Before the meeting, the Product Owner developed a new version of the Product Backlog and added 20 stories to it, each of them was assigned a value in the Priority box and description in the How to show box. After this the Product Owner asked the Development Team to give initial estimates. The team picked the metric ideal man-hours to estimate the story complexity. One of the team members offered to help estimate the stories, estimated the most important stories and filled in the Estimate box (see the Sprint Backlog above).
3. The first Sprint Planning was scheduled for Monday at 10:00 AM. It was estimated that the team has 100 man-hours per Sprint. After this the team has broken down stories into tasks and has refined estimates. After the general manpower effort of the tasks picked for Sprint exceeded 100 man-hours, the Product Owner stopped the Planning.

   The team created the Sprint Backlog and the team members divided and shared the Sprint tasks. The Scrum Master offered a daily Scrum Meeting at 10:00 AM, and demo and Retrospective Meeting on Monday 14:00 to 18:00 PM. The team members and the Product Owner concur with the offer.
4. After the Sprint Planning, the team members went to their workspaces and started working on the Sprint tasks.
5. The next day at 10:00 AM, everybody gathered in a room for a daily Scrum Meeting. The Scrum Master proposed anyone, who feels like it, to answer the questions. Irina was first.

   Scrum Master: *What did you do yesterday?*

   Irina: *I completed the task "Fix the bug related to creating a check based on the client's Order in the business process".*

Scrum Master: *What prevented you from the completion of another task?*

Irina: *: I don't really understand how I should implement it.*

Scrum Master: *Who can help Irina?*

Kostia: *I have an idea.*

Scrum Master: *Irina and Kostia, please stay after the meeting and discuss how to solve this problem, ok?*

Colleagues: *Ok.*

Everybody answered the three questions and the meeting was over. Kostia and Irina stayed after the meeting to find a solution to the above problem.

6. Every daily Scrum Meeting revealed problems the team members have and after the meeting, the team members solved these problems.
7. On Friday morning, the Scrum Master walked up to each team member and asked: "Do you know what and how you will show during the demo?" In case of doubts the Scrum Master helped the team member to prepare for the demo.
8. On Friday at 14:00 PM, the Demo was held in the room reserved by the Scrum Master. The Product Owner made five remarks on the completed tasks and gave two ideas on how the completed functionality can be changed. The Scrum Master wrote down all the ideas and remarks and handed them to the Product Owner. Upon the completion of the Demo, the Product Owner added all remarks and ideas to the Product Backlog within the next 15 minutes and set priorities to them. The Demo took 1 hour.
9. The Scrum Master offered a 20-minute coffee break.

10. The whole team but the Product Owner gathered in the same room at 15:20 PM.

    The Scrum Master announced the goal of the Retrospective Meeting: *We should find the techniques and tools that helped us to implement this sprint, and bring out rough spots to improve our process. We have 1 hour to discuss these questions, everybody should speak. Let's start with Irina.*

    Scrum Master: *What went well in this Sprint?*

    Irina: *I completed all my tasks.*

    Scrum Master: *Wonderful news, I'll write it in the "What went well" section.What could go better?*

    Irina: *We didn't manage to implement all set tasks.*

    Scrum Master: *What prevented us from this?*

    Irina: *It looks like we are not good at planning our business hours and arranging the task completion during the day.*

    Scrum Master: *Who agrees?*

    (everybody raises their hands)

    Scrum Master: *What can we do to change it?*

    Kostia: *I read this great book on time management, it helped me a lot. Maybe we should have a training on time management?*

    Scrum Master: *Who agrees?*

    (everybody raises their hands)

    Scrum Master: *Great. I will find a coach and send you a program book by the end of this Sprint.*

After this, the Scrum Master wrote his task "Find a training, get a program book, and send to colleagues for approval" in the "What to change?" section on the board. Deadline: by the end of the Sprint. Assignee: Scrum Master.

Retrospective Meeting took 1 hour 15 minutes.

11. The next Sprint Planning was on Monday at 10:00 AM.
12. The team worked in each next Sprint in a similar way.

5. UTILITIES AND TOOLS USED WHEN WORKING WITH PROJECTS

Version Control Systems

When a Development Team works on an IT project, there is a need to share editing of document versions of the project and provide storage of the source code of the software being developed. Of course, you can share documents and the source code in folders stored on a server, but the number of such folders may increase enormously over time, which may complicate rollbacks to the previous versions, tracking changes, etc. Version control systems are designed to solve this problem.

Version control system (VCS) allows storing old file versions of files and go back to them if necessary, view changes made to the previous versions of a file, identify the author of the changes, specify who and when might have made a mistake in the code, and so on. If you save an unwanted change or lose a file, VCS will help you to easily restore it.

One of the VCS options is a **centralized version control system**. Such systems include a central server, which stores all files under the version control, and client computers, which receive file copies from it (see Figure 10).

This VCS category includes products such as CVS and **Subversion** (as known as SVN).

Concurrent versions system (CVS) was developed in 1980s.

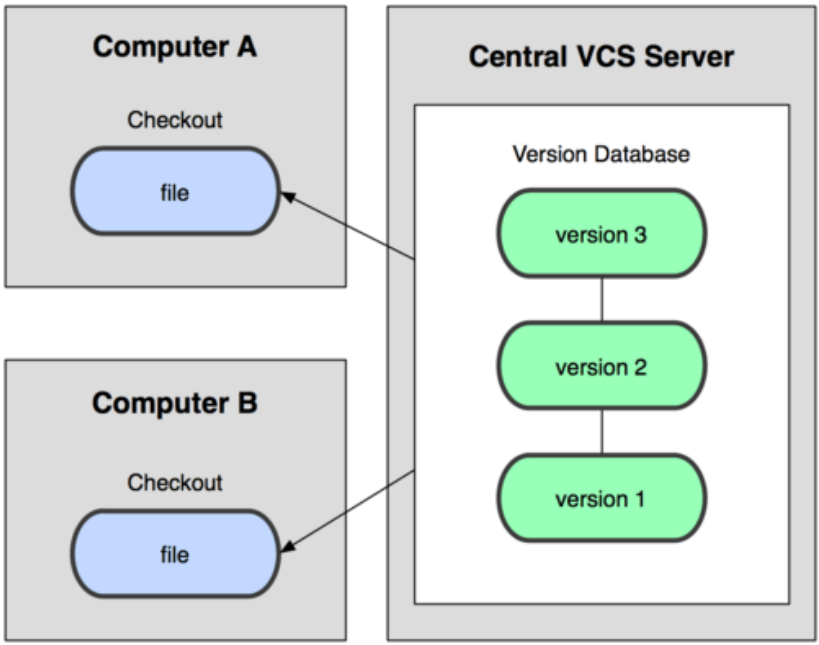

Figure 10

CVS allows **checking out**, or receiving a file version from the server, and then **checking in**, or returning the file with the made changes.

CVS was designed for shared work on a file avoiding version conflicts. Only the latest version of the code is allowed for shared work. One of the drawbacks of the CVS early versions was that the user had to quickly make changes to the repository before the other users do this. A later version of CVS allowed working with code branches, which resulted in multiple variants and functions of the same product that can be later combined. Other CVS disadvantages are:

- If the file is renamed or relocated, it is not displayed in the history;

- Security problems related to symbolic links to files;
- Atomic operations are not supported, which may lead to data corruption;
- Operations with the code branches are expensive because the VCS is not designed for long-term projects with branches of code.

CVS is a time-proved version control system provided free of charge, but its disadvantages led to the emergence of a **Subversion (SVN)** system.

The development of SVN started in 2000, when CVS was recognized by the software developers. The creators of SVN decided to use the trusted concepts and provide an easy transition for CVS users to a new product.

In general, SVN exceeds CVS by all measures except labels that address file system objects.

Disadvantages of SVN:

- Mistakes related to renaming files and directories;
- Poor set of commands to work with the repository;
- Relatively slow.

SVN Basics

Let's consider how to work with the SVN repository through the example of **TortoiseSVN**.

TortoiseSVN is a free client for the Subversion VCS made as a Windows shell.

A **repository** is a centralized storage for source code, working materials, and documentation. Any number of clients can connect to the storage, read or write these files.

A repository stores all folder and file structures. The repository stores all changes recorded in it from the creation date. To track changes time-wise, each transaction that changes a repository content is assigned a unique revision number, it also remembers the time and the author. All revisions in the repository are available to any user.

A **repo-browser** is used to work with the repository. You can access it by navigating to the root folder of any hard drive and right-clicking on it to start the browser:

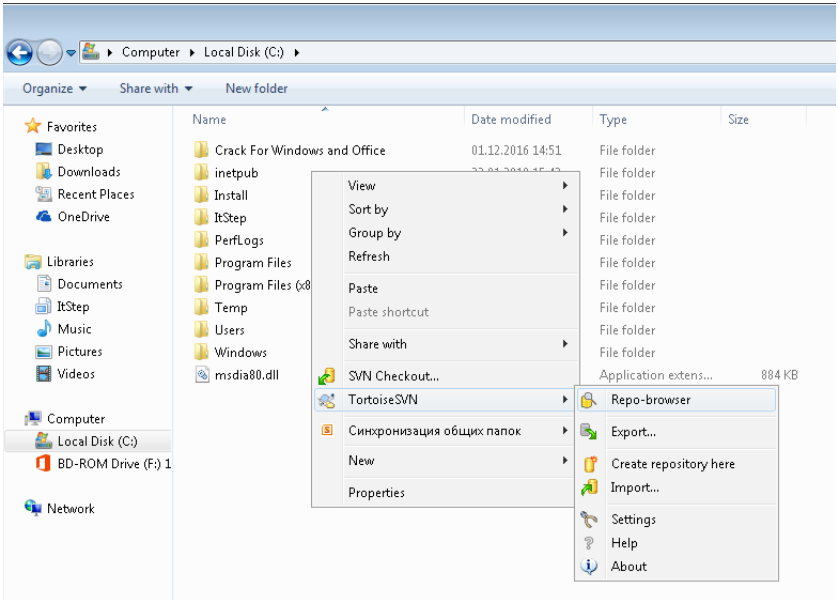

Figure 11

To view the repository, select the one you want to access:

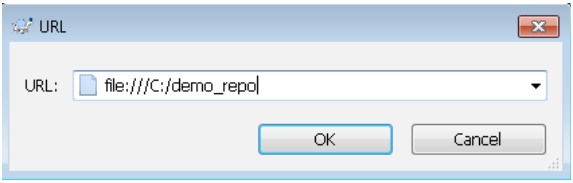

Figure 12

A newly created repository is empty:

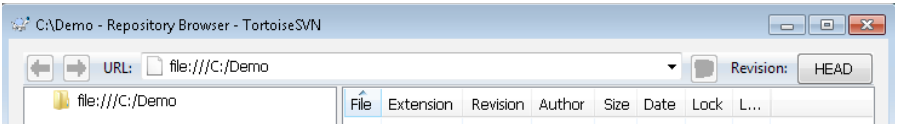

Figure 13

In order to put all your folders and files under the SVN control, you should create a project. You can do this by using the **Create folder** command in the root folder of the repository:

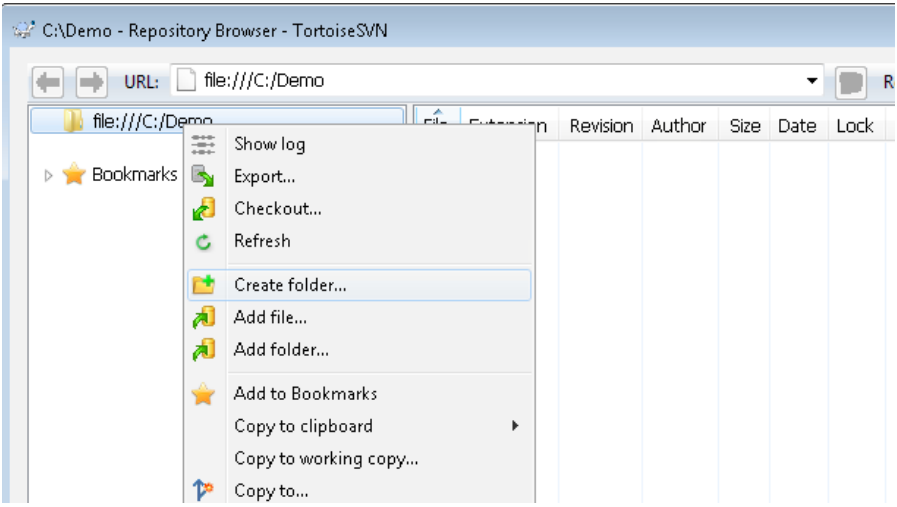

Figure 14

After this, enter a name of the project folder. Let it be **demo_project**:

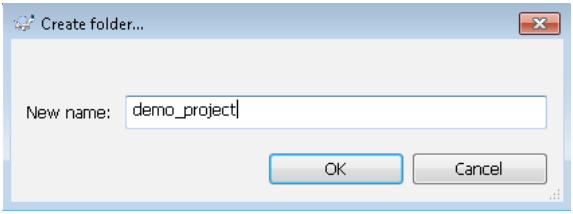

Figure 15

Create mandatory folders, which purpose is described below, in the same way. In the end we will get an empty project in the repository:

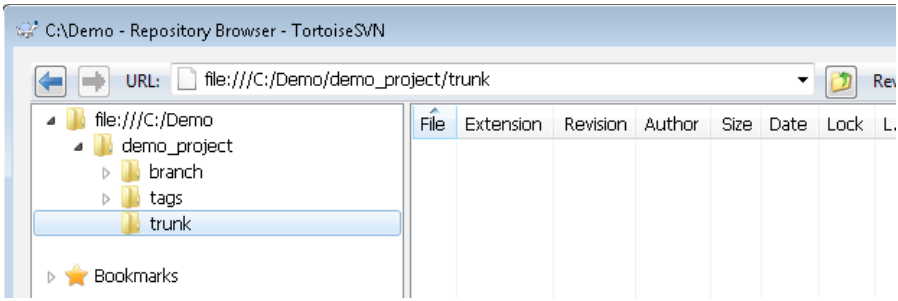

Figure 16

In order for you to start working on the project, create a working copy. For this create a root folder of the project on the hard disk. Navigate to this folder and use the **Checkout** command.

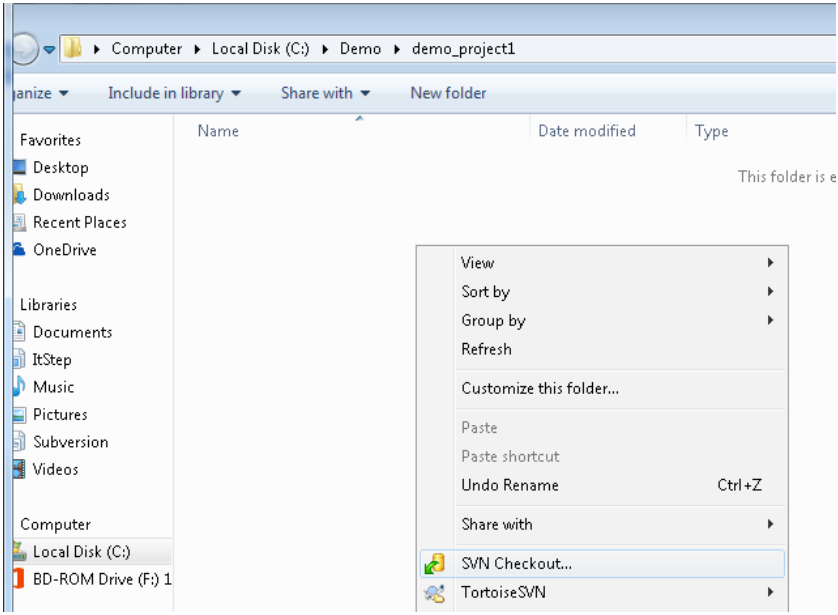

Figure 17

Use the repository browser to select a project, as well as folders and files in this project. Use **Show log** to select the required revision number (see Figure 18).

So, a working copy is created and controlled by SVN.

Now you can easily edit, delete, modify folders and files of the project, as well as add new files and folders without interfering with the work of your colleagues.

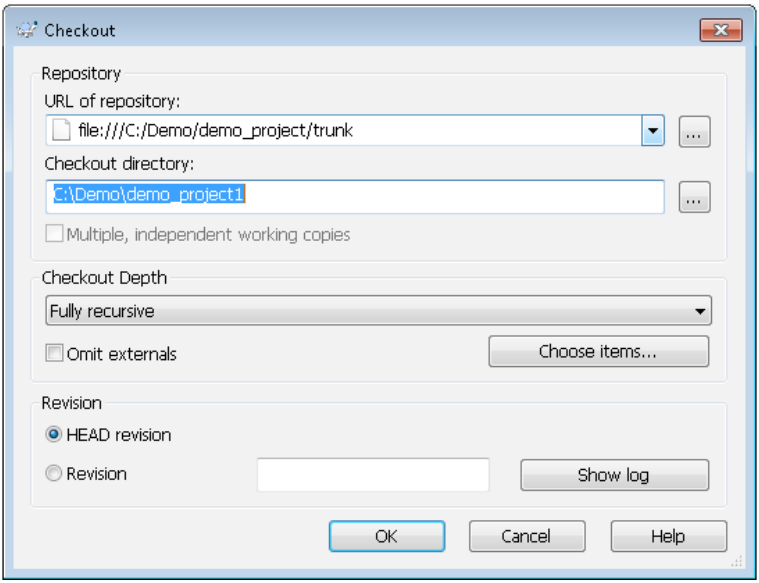

Figure 18

How to Work with the Working Copy

1. **Add files to the working copy.**

In order to add a file to source control (SVN), select the files and use **TortoiseSVN** > **Add**:

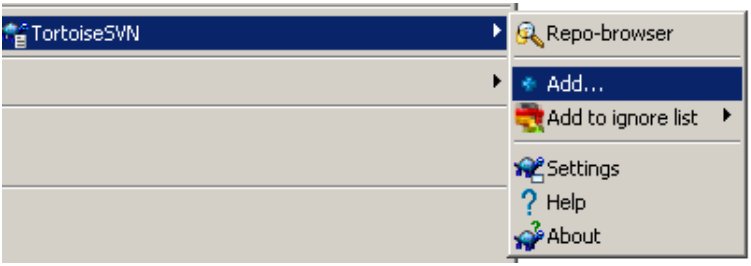

Figure 19

Before the command is executed, SVN provides a possibility to check a list of the added files. You can exclude a file if it was added by mistake.

To send changes to the working copy, use the **Commit** command.

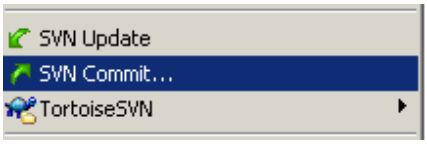

Figure 20

2. Synchronize the working copy and the repository.

Suppose two programmers independently of each other added their files to the project. In order for them to use each other's results, they have to synchronize their working copies. In SVN this is done via the repository with the **Update** command.

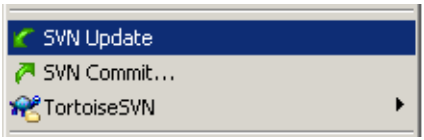

Figure 21

3. File change and rollback.

If you want to undo changes made to a file, select this file and use **Diff**.

In order to rollback to the previous revision, use the **Revert** command.

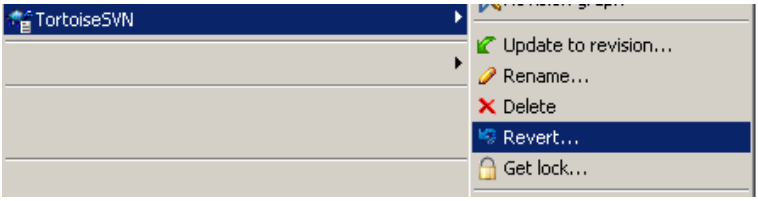

Figure 22

4. **Rename files.**

   If you want to rename a file, use **Rename**.

   To undo the renaming, use **Revert**.

5. **Resolving conflicts.**

   To resolve conflicts, use **Edit Conflicts**.

   To finalize the conflict resolution, delete all files that were automatically created to resolve the conflict. You can do this by means of the operating system, but the best way is by using **Resolved**.

6. **Locking folders and files.**

   You should use locking only if you want to be the only person who can modify locked folders and files. Select the file(s) in your working copy for which you want to acquire a lock, then select the command **TortoiseSVN** > **Get Lock**.

The considered commands should be enough for you to begin working with SVN, you will master the rest of the commands as you will proceed.

Centralized version control systems have an obvious drawback — if the centralized server is unavailable, developers cannot save their results in a new version of the file.

Distributed version control systems do not suffer from this drawback, since clients copy the whole repository. This means that if the server is unavailable for some reason, you can copy any of the client repositories back up to the server and restore the database. So when clients copy a new version of the files, they create a full copy of all data.

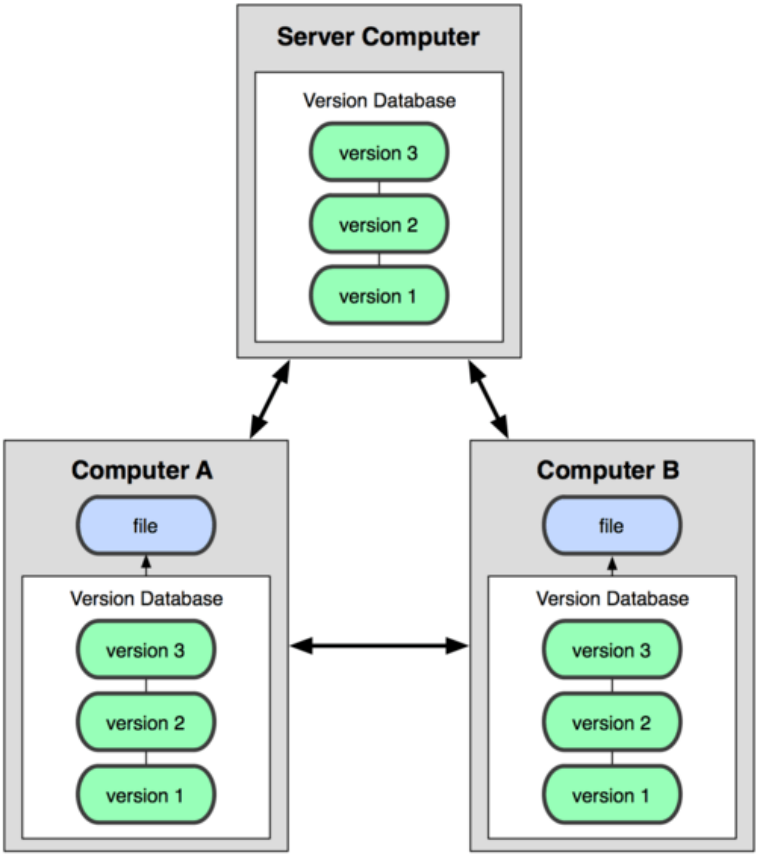

Figure 23

The majority of these systems allow working with multiple remote repositories, therefore you can work in different ways with different groups of people within the same project. Thus, you can have multiple working processes within a single project, which is impossible in centralized systems.

Basics of Git

One of the most popular distributed VCSs is **Git**.

The goal of the Git is the creation of a faster version control system compared to CVS. Git is the fastest under Linux.

Advantages of Git:

- High speed;
- Cheap operations with code branches;
- Full track of the development history, which is available offline.

Disadvantages of Git:

- Not all Git functions work properly on Windows as compared to Linux.

1. **The first thing you should do is to install Git**:
   - **Linux:** open a terminal window and install the app using the package manager for your distribution;
   - **Windows:** it is recommended to use **git for windows**, which contains a GUI client and a bash shell.
   - The simplest solution for **OS X** is to use **homebrew**.
2. **The next step is setup.**

There are many options, but you should configure the most important ones: user name and an e-mail address. This

allows outputting this information for every Git commit thus informing users who committed what.

3. **Create a new repository.**

If you are going to use Git for an existing project, then you should go to the project's directory and type:

```
$ git init
```

4. **Define the repository status.**

The **status** command displays the current state of the repository — update, new, or edited information.

Lifecycle of Files

Each file in your working repository can be in one of two states: tracked or untracked.

Tracked files are the files that were in the last project snapshot; they can be unmodified, modified, or staged. **Untracked files** are everything else — any files in your working directory that were not in your last snapshot and are not in your staging area.

When you first clone a repository, all of your files will be tracked and unmodified because Git just checked them out and you haven't edited anything yet.

As you edit files, Git sees them as modified, because you've changed them since your last commit. As you work, you selectively stage these modified files and then commit all those staged changes, and the cycle repeats.

Figure 24 depicts this lifecycle:

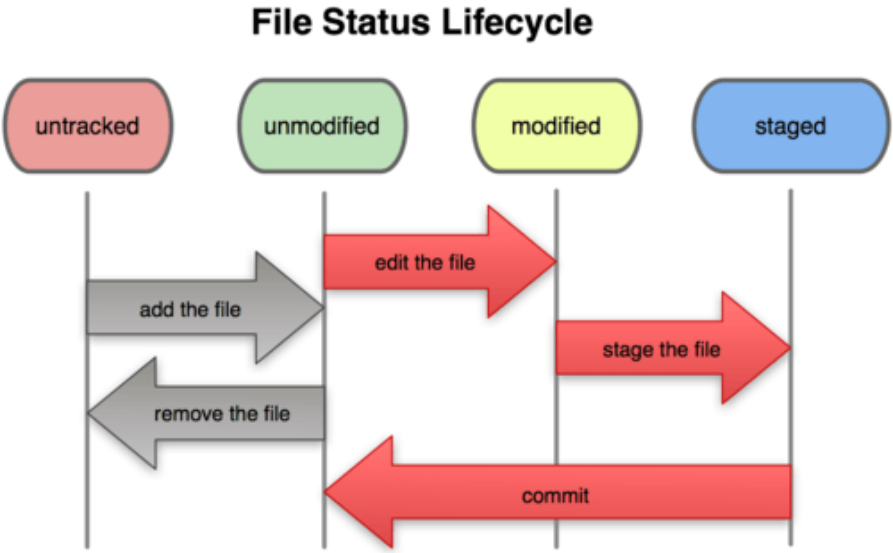

Figure 24

1. **Checking the status of your files.**

To determine states of your files, you should use the **git status** command.

2. **Tracking new files.**

In order to begin tracking a new file, you use the command **git add**. To begin tracking the **Tutorial** file, run this command:

```
$ git add Tutorial
```

3. **Staging files.**

Git has a staging area for staged files. You can visualize it as canvas where you add changes required in commit. The canvas is empty, then you add files to it (or a part of files, even

if it is just single strings) with the **add** command, and finally commit everything to the repository (take a snapshot of the state) with the **commit** command.

4. **Commit (record changes).**

Commit is a repository state at a certain point in time. It is a good practice to commit often and write informative comments.

The easiest way to record changes is to use the **git commit** command:

```
$ git commit
```

You can type your comment in the command line along with the **commit** command by specifying it after the **-m** flag, as the below example shows:

```
$ git commit –m "Story 182: Fix benchmarks for speed"
[master]: created 463dc4f: "Fix benchmarks for speed"
2 files changed, 3 insertions(+), 0 deletions(–)
create mode 100644 README
```

The commit has given you some output about itself: which branch you committed to (**master**), what SHA-1 checksum the commit has (**463dc4f**), how many files were changed, and statistics about lines added and removed in the commit.

5. **Removing files.**

To remove a file from Git, you have to remove it from your tracked files and then commit. The **git rm** command does that, and also removes the file from your working directory so you don't see it as an untracked file the next time around.

6. **Moving files.**

If you want to rename a file in Git, you can run something like:

```
$ git mv file_from file_to
```

7. **Branching.**

It is considered to be a good practice to work with a copy of the original project as you develop new functionality. Such copy is called a branch. Branches have a separate history and isolated changes until you decide to merge them. There is a set of reasons for this:

- The already working, stable version of the code is saved.
- New functions can be developed in parallel by different programmers.
- The developers can work with their branches without risking their code base to modify because of someone else's changes.
- In case of doubt, you can develop various implementation options of the same idea in different branches and then compare them.

Team members should learn to do the following:

- create a new branch;
- switch between branches;
- merge branches.

8. **Advanced skills that may appear useful.**
   The following skills may help you at shared work:
   - tracking committed changes;
   - reverse to the previous state;
   - edit commit;
   - resolve conflicts at merging.

6. BUG TRACKERS

Bug tracker systems were initially designed to automate error logging and fixing during the development of software products.

Jira entered the market in 2003. It was initially designed to track and fix errors in software products under development. In the little more than a decade Jira has become a system that can help a team to plan and manage all activities related to software development.

Today Jira is included in the Gartner Magic Quadrant for Enterprise Agile Planning Tools, which confirms its popularity.

Jira is a cloud-based service that includes the following in its main functionality:

- Gantt Chart;
- Task priority;
- Cloud storage;
- Messages on project events;
- Comments to tasks;
- Attaching files to tasks;
- Filters;
- Access setting;
- Tracking progress in percent;
- Tracking task states;
- Dashboards;
- Assignment management;

- Knowledge base;
- Email notifications;
- Frequently asked questions;
- Time tracking;
- Various reports.

The following functionality is implemented to manage an Agile project:

- sprint Backlog creation;
- backlog management;
- BurnDown diagram;
- security and confidentiality;
- Scrum board;
- Kanban board.

In terms of traditional approach to project management, Jira lacks the following functionality:

- project budgeting;
- calendars with vacations and weekends specified;
- creation of the Gantt Chart.

You can find a part of this functionality among the paid addons that extend possibilities of the product.

Jira has a good price for a team of 10 and less people ($120 a year) and much higher for a team of 11 and more people ($7 per user a month). And if you want to use **Confluence** to create a knowledge base, be ready to add $5 on top of it per user a month. You can buy and merge the functionality you are lacking with addons. Note that Jira does not have free offers.

Basics of Jira

To create a project in Jira, click **Create project**:

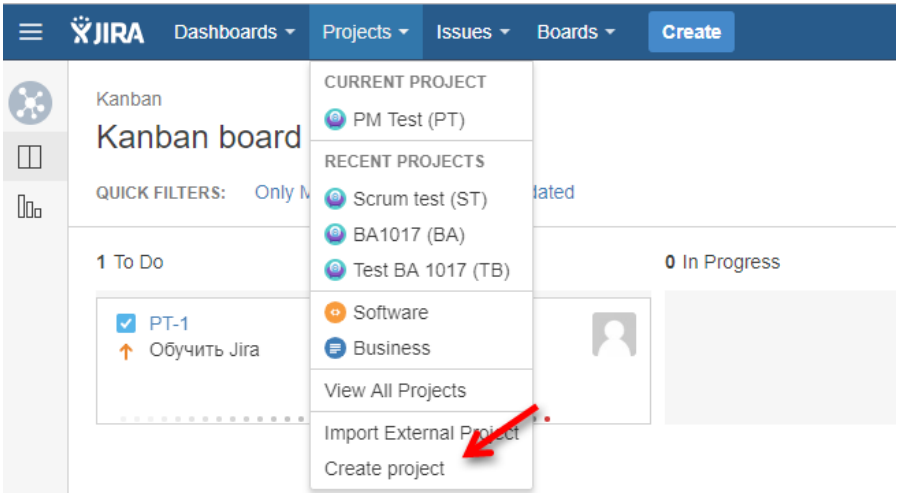

Figure 25

Then select a project type: **Scrum**, **Kanban**, or **Project Management**, for instance.

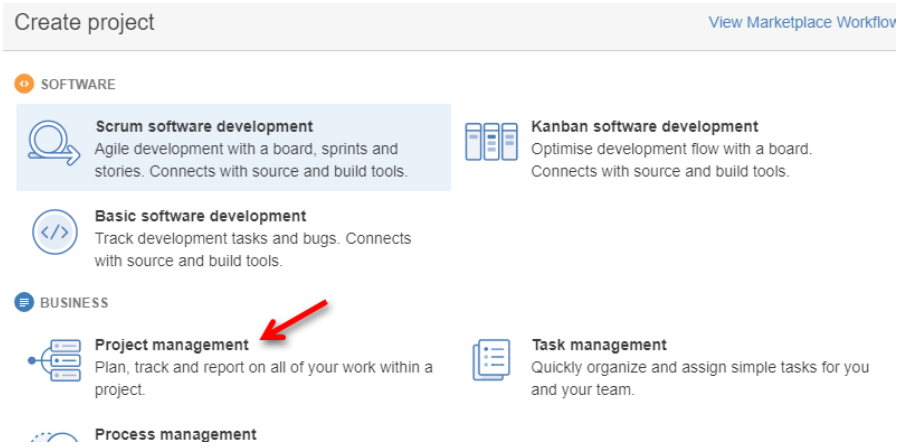

Figure 26

After this, configure the issue type:

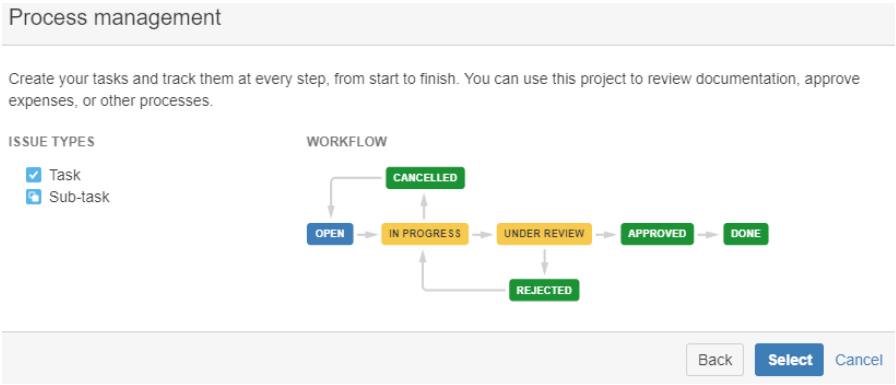

Figure 27

And enter the project parameters:

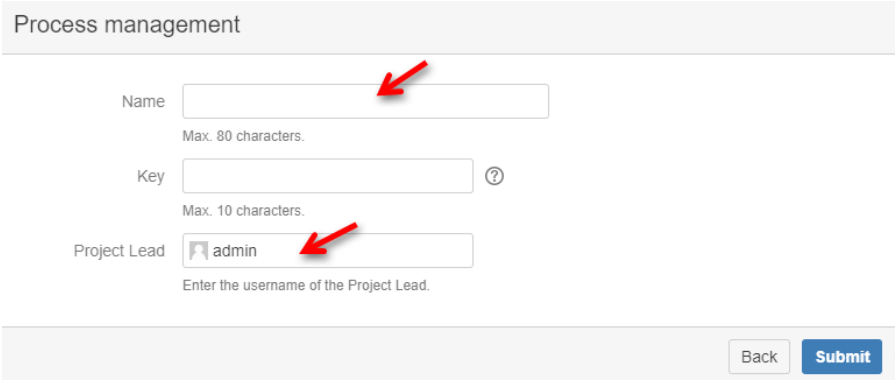

Figure 28

If your selected type of project is Scrum, you will have a Scrum board available and the following types of activity: **Epic**, **User story**, and **Task**.

If your selected type of project is Kanban, you can set up process stages, set WIP limits for each stage, etc.

Create activity in the project by clicking **Create**:

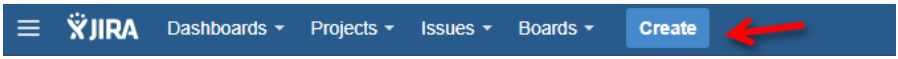

Figure 29

After this you can select a type of the activity and fill in the boxes.

Within the Scrum projects, you can drag the task cards across the board:

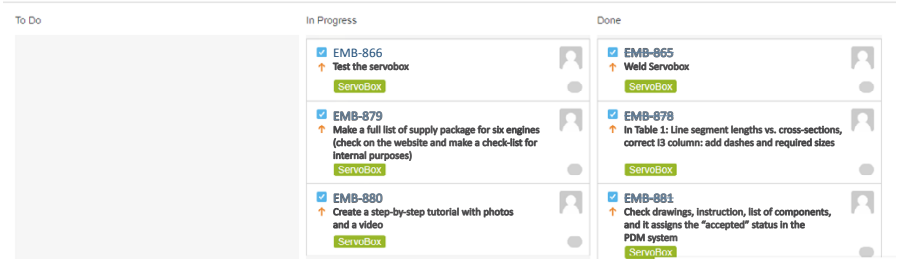

Figure 30

Each task enables tracking of the time spent and time left, and you can use a BurnDown diagram to manage the Sprint:

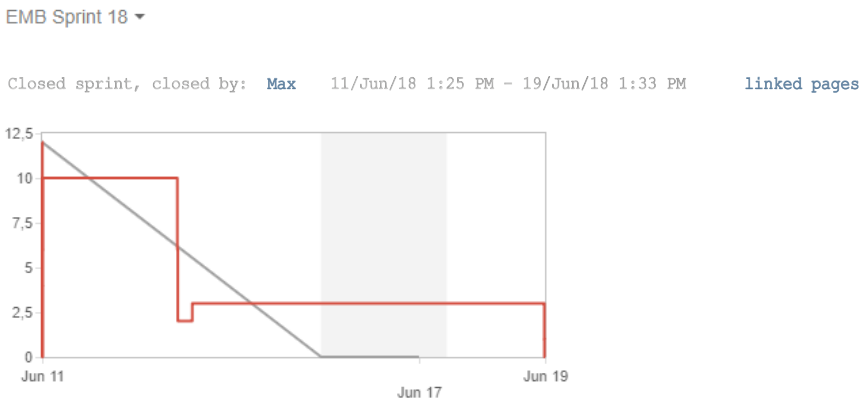

Figure 31

To complete a Sprint, press this button:

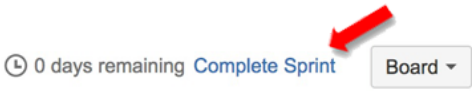

Figure 32

In order to master all Scrum techniques, the user should spend a couple of hours in Jira and he will understand everything.

It is worth noting the configuration and administration of Jira is a quite complicated process, so one might need hundreds of hours to understand them.

Mantis BT

Mantis BT is a free bug tracker that allows creating error messages and track how they are further handled. You can use it as a request and incidents tracking system. You can integrate it with the wiki engine for documentation (**DokuWiki**).

Mantis BT is a cloud system that works via web browser. It has the following advantages:

- it is free;
- PHP code is easily modified;
- customizable fields;
- convenient filters.

7. PROJECT MANAGEMENT SOFTWARE SYSTEMS

MS Project

MS Project is a product by Microsoft that was for a long time considered to be one of the most efficient products for deadline and budget management. On the market of the single-user solutions Microsoft Project was a leader up to 2010 and earned more than $900 a year from their 20,000,000 users.

This software supports almost all methods and time and cost management tools described in PMBOK such as:

Project planning

Decomposition of tasks

Effort estimates

Network diagram and critical path method

Resource structures

Functionality designed for resource optimization Resource leveling

Cost planning

Estimate of risk influence through the "What If?" modeling technique

Interactive optimization of project plans

Tracking and managing a project

Possibility to save the baseline project

Plan/fact analysis

Possibility to enter actual and remaining efforts on project tasks

Gantt Chart with tracking

Earned value method

It is worth noting that MS Project is one of the most efficient software in the world for project simulation within limited resources. However, **MS Project Standard** is not enough for shared work on a project. That is why the MS Project developers offer **MS Project Server** that implies a client-server architecture or a cloud product, **Project Online**.

The main benefit of MS Project is the possibility to create a model of the project that includes all existing limitations. Let's view examples of creating such a project model.

Example of MS Project for planning an IT project

We will learn the MS Project possibilities through the example of the **MS Project 2013 Standard**.

Suppose an IT company received a request from a client, who wants to implement an IT system within 3 months. The CEO of this IT company has to figure out whether this deadline is feasible, considering the employee load on other projects.

1. **To answer this question, the project manager simulated the project in MS Project.**

When you enter a default program, you can see the **Gantt Chart**:

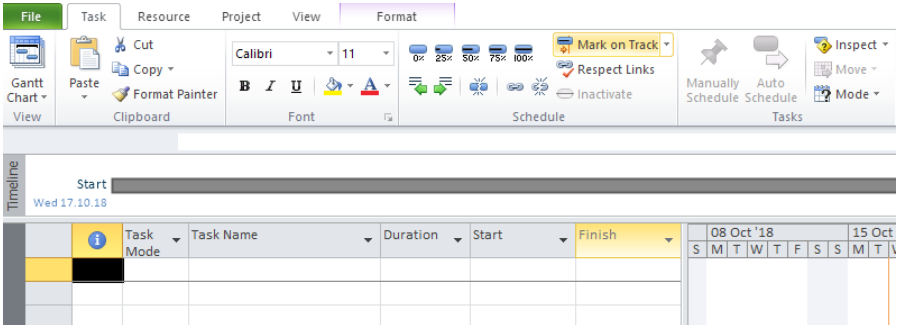

Figure 33

To switch between views — table or diagram — press the **Gantt Chart** button.

Using the creation of the task hierarchy, the project manager first developed a structure of works on the project in the **Gantt Chart** view. For this, he entered a list of tasks in the **Gantt Chart** and pressed the **Indent** button to create a hierarchy in the list:

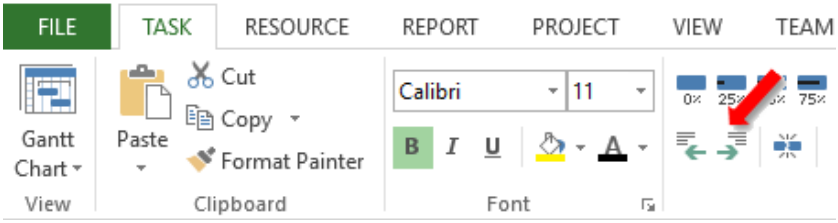

Figure 34

Here is how the hierarchy of works on the project looks like:

Project Management	**0 days?**	**Fri 14.09.18**
Plan Sprints, Retrospectives, Standups	0 days?	Fri 14.09.18
Meet with the Customer on the Project State	0 days?	Fri 14.09.18
Develop and Coordinate Documents	0 days?	Fri 14.09.18
Improve and Implement the Salary Unit		**Fri 14.09.18**
Primary Population of the Information Base		**Mon 17.09.18**
Analysis of Requirements and Writing SRS	0 days?	Mon 17.09.18
Loading	0 days?	Mon 17.09.18
Tests	0 days?	Mon 24.09.18
Bug Fixes	0 days?	Mon 22.10.18
Primary Setup of the System by the Accounting Policies of 3 Companies	0 days?	Mon 17.09.18
Salary in Foreign Exchange		**Fri 14.09.18**
Requirements Analysis	0 days?	Fri 14.09.18
Improve Calculations	0 days?	Fri 21.09.18
Tests	1 days?	Tue 25.09.18
Demo	1 days?	Thu 27.09.18
Bug Fixes	1 days?	Mon 01.10.18

Figure 35

After this he added a **Work** column (right-click on the **Duration** field, select **Add column** and select a **Work** field

from the drop-down menu), and for each task added data on planned efforts per task in man-hours.

Name	Work	Duration	Start	Finish
Project Management	**32 h**	**50 days**	**Fri 14.09.18**	**Thu 22.11.18**
Improve and Implement the Salary Unit	**205 h**	**56 days?**	**Fri 14.09.18**	**Fri 30.11.18**
Primary Population of the Information Base	**70 h**	**35 days**	**Mon 17.09.18**	**Fri 02.11.18**
Primary Setup of the System by the Accounting Policies of 3 Companies	25 h	20 days	Mon 24.09.18	Fri 19.10.18
Salary in Foreign Exchange	**10 h**	**13 days**	**Fri 14.09.18**	**Tue 02.10.18**
Requirements Analysis	2 h	5 days	Fri 14.09.18	Thu 20.09.18
Improve Calculations	4 h	2 days	Fri 21.09.18	Mon 24.09.18
Tests	1 h	2 days	Tue 25.09.18	Wed 26.09.18
Demo	1 h	2 days	Thu 27.09.18	Fri 28.09.18
Bug Fixes	2 h	2 days	Mon 01.10.18	Tue 02.10.18
Improve Accountance of Fringe Benefits for Employees, Reimbursements	**15 h**	**15 days**	**Wed 03.10.18**	**Tue 23.10.18**
Requirements Analysis	2 h	4 days	Wed 03.10.18	Mon 08.10.18
Improve Calculations	9 h	5 days	Tue 09.10.18	Mon 15.10.18
Tests	1 h	2 days	Tue 16.10.18	Wed 17.10.18
Demo	1 h	2 days	Thu 18.10.18	Fri 19.10.18
Bug Fixes	2 h	2 days	Mon 22.10.18	Tue 23.10.18
Improve the Payroll Requirements, Bonuses, Sport, Financial Help	**45 h**	**25 days**	**Mon 29.10.18**	**Fri 30.11.18**
Requirements Analysis	10 h	5 days	Mon 29.10.18	Fri 02.11.18

Figure 36

You can see (Figure 35) that such sequence of creating a project results in **Duration** field's value of **1 days?**; the question mark in this field means that the duration is set automatically, and the user has never changed data in it.

Task duration is calculated automatically after the efforts (**Work** field) are set and people are assigned.

2. **At the next step, the project manager created a list of the project members in the Resource Sheet.**

To do this, he clicked the **Gantt Chart** button and selected the **List Sheet** command from the drop-down menu:

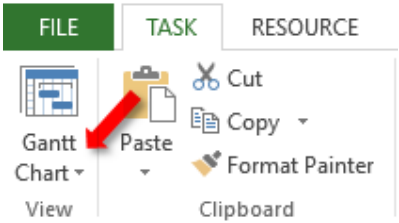

Figure 37

He has also set a man-hour cost for each of them (**Standard Rate** field) and the amount of available time in % the member has for a project (**Max Units** field):

Name	Type	Max Units	Standard Rate
Developer 1	Work	30%	$13,00/h
Developer 2	Work	50%	$13,00/h
Analyst	Work	50%	$5,00/h
Project Manager	Work	20%	$30,00/h
	Work	100%	$0,00/h
<New Resource>	Work	100%	$0,00/h

Figure 38

Data in the **Max Units** field specify maximum percentage or number of units representing the maximum capacity for which a resource (a person) is available to accomplish any tasks.

For instance, Developer 1 works 8 hours a day and can spend no more than 2.4 hours a day on the project. If you assign to this Developer tasks that require more time, the software will notify you that this resource is overallocated.

3. **After this, the project manager went back to the Gantt Chart view and assigned developers of the project to tasks.**

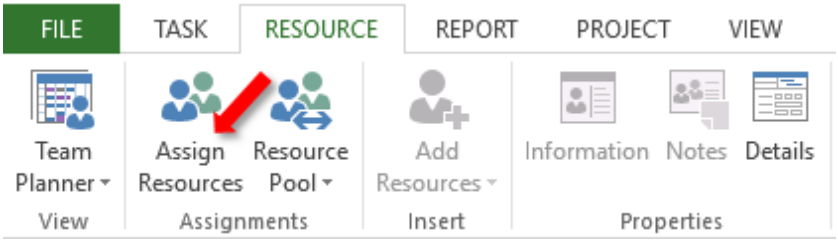

Figure 39

As you assign developers, the software calculates duration, including efforts and capacity percentage specified in the **Max Units** field in the **Resource Sheet**.

In the end, the project manager has got the following calculations:

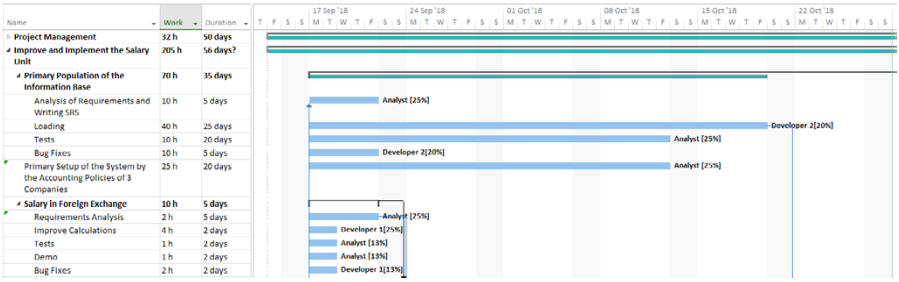

Figure 40

The diagram to the right of the table (see Figure 40) illustrates duration of each task on a time bar; the diagram shows start and finish dates for all project tasks, this is a Gantt Chart.

4. **At the next step, the project manager specified task links.**

For this he selected a predecessor task, held down **Ctrl** and selected the next, successor task. After this he pressed the **Link Tasks** button:

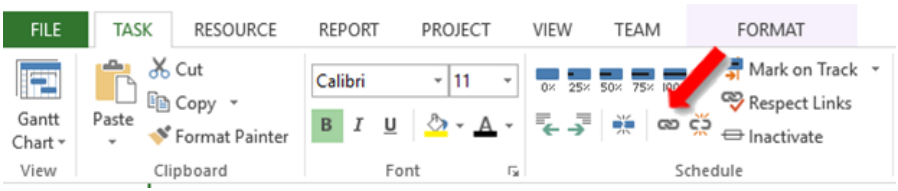

Figure 41

And the project manager got the following estimate for the project implementation (see Figure 42).

As Figure 42 shows, the Gantt Chart has relationships between tasks now, and if these relationships are correct, the project manager can assume that the most optimistic finish date is 30.11.2018.

5. **To check the project correctness, make sure that no project member is overallocated.**

This software can do this if you enable the **Resource Usage** view. No resource is red in the table (see Figure 43), which means that no resource is overallocated. But even the above action doesn't make our project resistant to external or internal impact, so you want to calculate risks, add risk management solutions to the project, and recalculate time and budget to increase the success probability.

Name	Work	Duration	Start	Finish
Account Automatization	**237 h**	**56 days?**	**Fri 14.09.18**	**Fri 30.11.18**
Project Management	**32 h**	**50 days**	**Fri 14.09.18**	**Thu 22.11.18**
Improve and Implement the Salary Unit	**205 h**	**56 days?**	**Fri 14.09.18**	**Fri 30.11.18**
Primary Population of the Information Base	**70 h**	**35 days**	**Mon 17.09.18**	**Fri 02.11.18**
Primary Setup of the System by the Accounting Policies of 3 Companies	25 h	20 days	Mon 24.09.18	Fri 19.10.18
Salary in Foreign Exchange	**10 h**	**13 days**	**Fri 14.09.18**	**Tue 02.10.18**
Requirements Analysis	2 h	5 days	Fri 14.09.18	Thu 20.09.18
Improve Calculations	4 h	2 days	Fri 21.09.18	Mon 24.09.18
Tests	1 h	2 days	Tue 25.09.18	Wed 26.09.18
Demo	1 h	2 days	Thu 27.09.18	Fri 28.09.18
Bug Fixes	2 h	2 days	Mon 01.10.18	Tue 02.10.18
Improve Accountance of Fringe Benefits for Employees, Reimbursements	**15 h**	**15 days**	**Wed 03.10.18**	**Tue 23.10.18**
Requirements Analysis	2 h	4 days	Wed 03.10.18	Mon 08.10.18
Improve Calculations	9 h	5 days	Tue 09.10.18	Mon 15.10.18
Tests	1 h	2 days	Tue 16.10.18	Wed 17.10.18
Demo	1 h	2 days	Thu 18.10.18	Fri 19.10.18
Bug Fixes	2 h	2 days	Mon 22.10.18	Tue 23.10.18

Figure 42

Name	Work
Developer 1	31 h
Developer 2	88 h
Analyst	86 h
Project Manager	30 h

Figure 43

6. **To calculate prime cost of the project, the project manager adds the Cost fields in the table and gets data on the project's prime cost:**

Name	Work	Duration	Cost
Account Automatization	**237 h**	**56 days?**	**$2 577,00**
Project Management	**32 h**	**50 days**	**$600,00**
Improve and Implement the Salary Unit	**205 h**	**56 days?**	**$1 977,00**
Primary Population of the Information Base	**70 h**	**35 days**	**$750,00**
Primary Setup of the System by the Accounting Policies of 3 Companies	25 h	20 days	$125,00
Salary in Foreign Exchange	**10 h**	**13 days**	**$98,00**
Requirements Analysis	2 h	5 days	$10,00
Improve Calculations	4 h	2 days	$52,00
Tests	1 h	2 days	$5,00
Demo	1 h	2 days	$5,00
Bug Fixes	2 h	2 days	$26,00
Improve Accountance of Fringe Benefits for Employees, Reimbursements	**15 h**	**15 days**	**$163,00**
Requirements Analysis	2 h	4 days	$10,00
Improve Calculations	9 h	5 days	$117,00
Tests	1 h	2 days	$5,00
Demo	1 h	2 days	$5,00
Bug Fixes	2 h	2 days	$26,00

Figure 44

The described case shows the main capabilities of MS Project as software for planning and estimating time and resources of an IT project. This software also helps simulate scenarios such as, "How will time change if we allocate this many resources for a project?"

It also allows calculating budget if we have work load and man-hour cost estimated. The software provides the possibility to set a fixed cost input per task or stage (e. g., cost of a contract on performing a task).

After the customer is happy with the estimated time and budget, the project manager saves the project baseline and can make a plan-fact analysis on time and budget. For example, to analyze the project progress in terms of time, you can use the **Tracking Gantt** within the Gantt Chart:

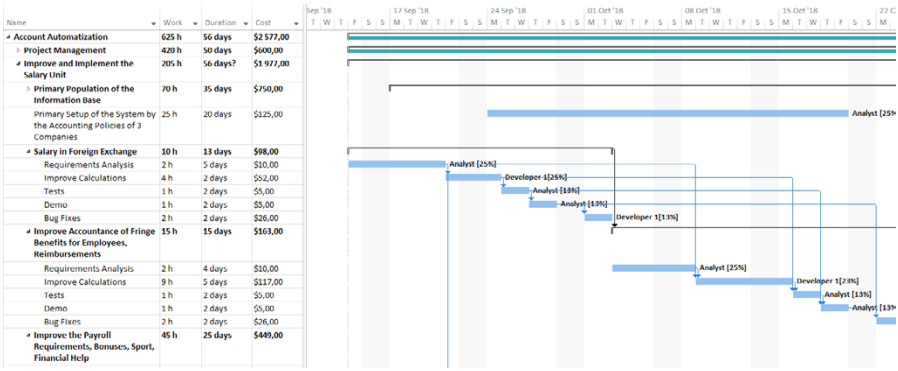

Figure 45

Gray pieces are time for baseline, and blue and red pieces are time for the current plan. The diagram shows that the project falls behind.

The **Cost** table displays data on baseline cost vs actual cost of the project tasks:

Name	Fixed Cost	Fixed Cost Accrual	Cost	Baseline Cost	Cost Variance	Actual Cost
Project Management	**$0,00**	**Prorated**	**$600,00**	**$0,00**	**$600,00**	**$0,00**
Improve and Implement the Salary Unit	**$0,00**	**Prorated**	**$1 977,00**	**$0,00**	**$1 977,00**	**$0,00**
Primary Population of the Information Base	**$0,00**	**Prorated**	**$750,00**	**$0,00**	**$750,00**	**$0,00**
Analysis of Requirements and Writing SRS	$0,00	Prorated	$50,00	$0,00	$50,00	$0,00
Loading	$0,00	Prorated	$520,00	$0,00	$520,00	$0,00
Tests	$0,00	Prorated	$50,00	$0,00	$50,00	$0,00
Bug Fixes	$0,00	Prorated	$130,00	$0,00	$130,00	$0,00
Primary Setup of the System by the Accounting Policies of 3 Companies	$0,00	Prorated	$125,00	$0,00	$125,00	$0,00
Salary in Foreign Exchange	**$0,00**	**Prorated**	**$98,00**	**$0,00**	**$98,00**	**$0,00**
Requirements Analysis	$0,00	Prorated	$10,00	$0,00	$10,00	$0,00
Improve Calculations	$0,00	Prorated	$52,00	$0,00	$52,00	$0,00
Tests	$0,00	Prorated	$5,00	$0,00	$5,00	$0,00
Demo	$0,00	Prorated	$5,00	$0,00	$5,00	$0,00
Bug Fixes	$0,00	Prorated	$26,00	$0,00	$26,00	$0,00
Improve Accountance of Fringe Benefits for Employees, Reimbursements	**$0,00**	**Prorated**	**$163,00**	**$0,00**	**$163,00**	**$0,00**
Requirements Analysis	$0,00	Prorated	$10,00	$0,00	$10,00	$0,00
Improve Calculations	$0,00	Prorated	$117,00	$0,00	$117,00	$0,00
Tests	$0,00	Prorated	$5,00	$0,00	$5,00	$0,00
Demo	$0,00	Prorated	$5,00	$0,00	$5,00	$0,00

Figure 46

Within its field, MS Projects was one of the most popular products in the world for a good many years.

However, shared work in MS Project required installation and administration of MS Project Server, which was challenging for IT administrators.

New project management systems began to appear in 2006 and 2007. And by early 2010, they displaced the systems that required installation on servers of a company. The developers of MS Project have missed this trend, and by the time of the MS Project Online (cloud MS Project) release, the battle was lost. For the past five years (beginning with 2013), Microsoft Project Online was outside the Magic Quadrant for Cloud-Based

IT Project and Portfolio Management Services by Gartner.

That is why many IT companies use MS Project when they are planning the project, and after this they switch to **Jira** or **Easy Projects** for shared work. Easy Projects provides a possibility to upload a file created in MS Project and proceed working on tasks together and edit the general plan of the project.

Easy Projects

Easy Projects is a cloud project that supports both traditional project management tools (Gantt chart, earned value management, etc.) and Agile approaches, for instance, Kanban.

The main advantages of this service is ease of use, balanced functionality, and existence of paid and free workplaces, which many competitors lack. Easy Projects allows you to upload an MS Project file and proceed you work in the Easy Projects environment. It has many integrations with popular products such as Slack, MS Outlook, Power BI (one of the world's best BI systems), and Zappier which allows you to integrate many services on your own with no programming skills required.

The Easy Projects' clients include well-known companies such as Lenovo, Hewlett Packard, Symantec, Toshiba, etc.

Easy Projects' cost varies from $10 to $26 a month per user, depending on the functionality; there are also free offers, where the users can view list of tasks they are assigned, attach files to tasks, complete and comment tasks.

Team Foundation Server

Team Foundation Server (abbreviated to **TFS**) is a Microsoft product representing a complex solution that incorporates version control system, data collection, reporting, tracking statuses and changes made to the project. This software is designed for shared work on software development projects.

TFS automates the following functions:

- **Source control**.
- **Reporting**.

The **Reporting** layer allows you to create many reports based on the joined information about working elements, change sets, information. You can export reports, created with SQL Server Reporting Services, to different formats, including Excel, XML, PDF, and TIFF. Reports are viewed with Visual Studio or a web portal.

- **Project portal**.

TFS uses **SharePoint** possibilities to create a website for a project. This website allows you to track the project's progress, supervise working elements and documents in the project library.

- **Shared services.**

TFS supports many services that you can potentially integrate with third-party applications such as project management systems.

If you want to use the Scrum methodology to organize the work of your team, you can do it in TFS by using the Scrum process template. Microsoft states that this template

was developed in consultation with experts from **Scrum.Org**, one of the world's leading consulting companies specialized in Scrum.

The template consists of the following work items:

- **Product Backlog Item (PBI)**: tracks requirements to the whole software product.
- **Bug**: an error found during development.
- **Sprint**: records Sprint dates, goals, and Retrospective Meetings. You can indicate Sprint with the iteration hierarchy only, which unfortunately doesn't allow you to specify start and finish date of a Sprint.
- **Impediment**: problems, risks, and anything that may influence the team work but not related directly to properties of the product being developed.
- **Test Case**: description of a test case for **PBI** Tests are usually bound to a specific **PBI**. Like **Shared Steps**, this work item is not specified in Scrum directly but is required for correct operation of TFS.
- **Shared Steps**: test steps shared among multiple **Test Cases**.

Teams use the **Product Backlog Item** (PBI) and **Bug** work items to plan Sprints (see Figure 47).

Note that **Product Backlog Item** and **Bug** are Product Backlog items of an equal value in terms of cost estimate and planning, that is why they have almost identical set of fields.

Task. This item allows adding the required details for **Product Backlog Item**. Of course, tasks also can be split into subtasks.

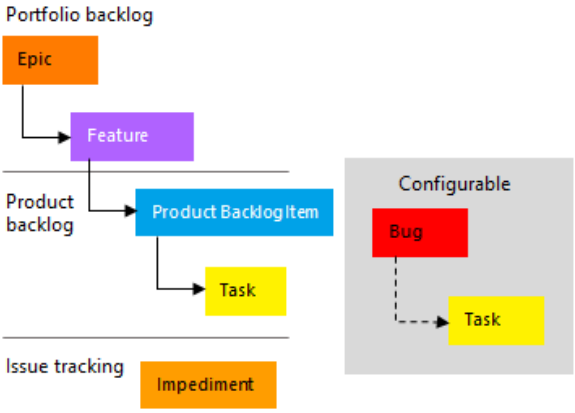

Figure 47

You can quickly create **Product Backlog Item** and **Bug** on the Backlog page:

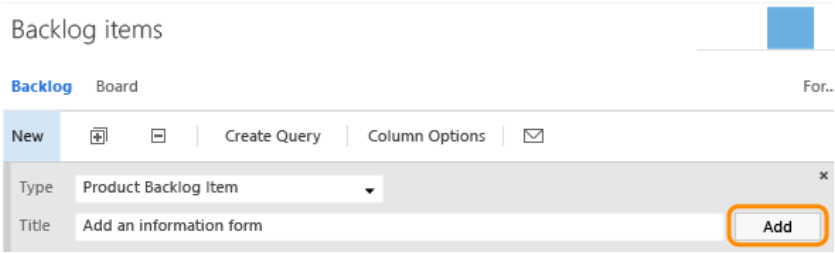

Figure 48

The Product Owner can set priorities for each **PBI** and **Bug** in the **Backlog Priority** field. In addition, the Development Team fills in the following in the **Backlog Item** form (see Figure 49):

Field/Tab Name	Description
Efforts	Estimate of scope of work to perform PBI. Any units are good: Story points, man-hours. This field is required for the reports: **Release Burndown** and **Velocity**

Field/Tab Name	Description
Value Area	A number that represents a relative PBI value compared to other PBIs. The greater the number, the higher its value
Description (PBI)	In-depth information about a user story or an error. The amount of information is right if the team can create tasks and test cases to implement the item
Acceptance Criteria	Write criteria for a team to be guided by when checking whether a PBI is implemented in full or whether a bug is fixed

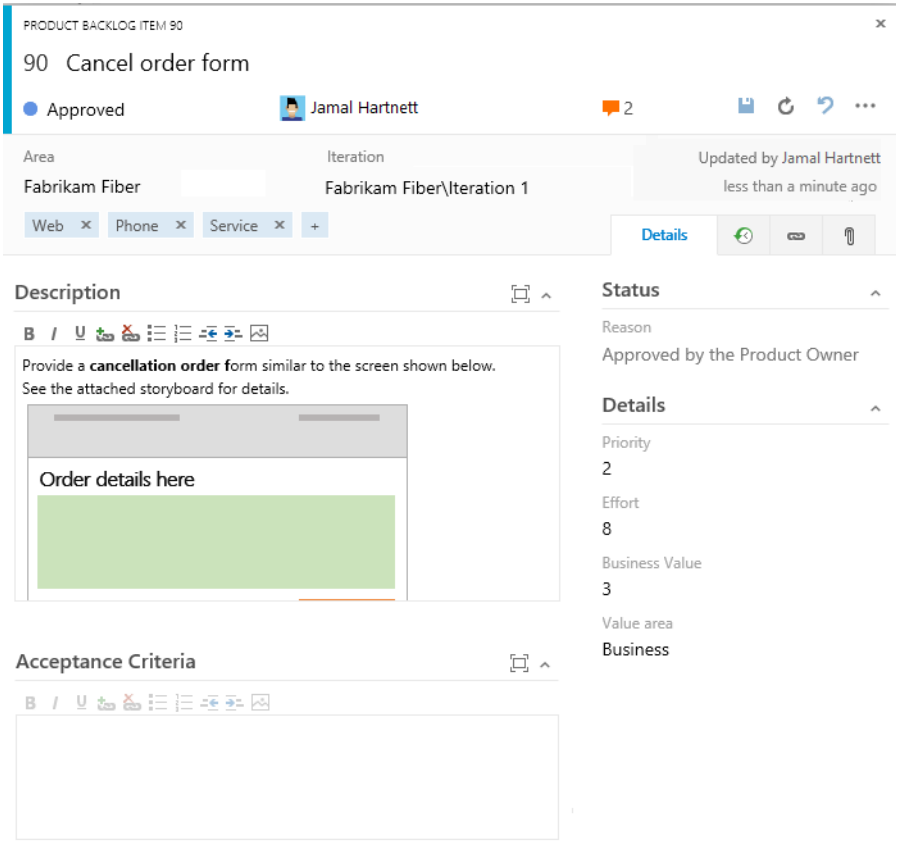

Figure 49

How to arrange Scrum in TFS

1. Create an initial **PBI** list and set priorities for each item.
2. Select the most important items from the **PBI** list and estimate their complexity (**Effort** field). Send **PBIs** that you struggle to estimate (unclear or don't answer some questions) for rework.
3. The team forms the first Sprint based on the priorities and values of **PBI** from the full set of **Product Backlog** that the team already knows how to implement.
4. PBIs that appeared in **Sprint** are split into **Tasks**. Tasks have an implementation deadline.
5. The found errors are recorded with the **Bug** item, they also appear in the **Product Backlog**.
6. Completed **PBI** and Bugs have the **Done** status.
7. The next Sprint is planned.

Tracking the Sprint progress

The team can use a Kanban board to track the progress of PBI implementations and bug fixes. If you drag an item to a new column, the **State** field updates (see Figure 50).

The item switches to **Done** when all the tasks related to it are completed and the Product Owner accepted this PBI according to the Acceptance criteria.

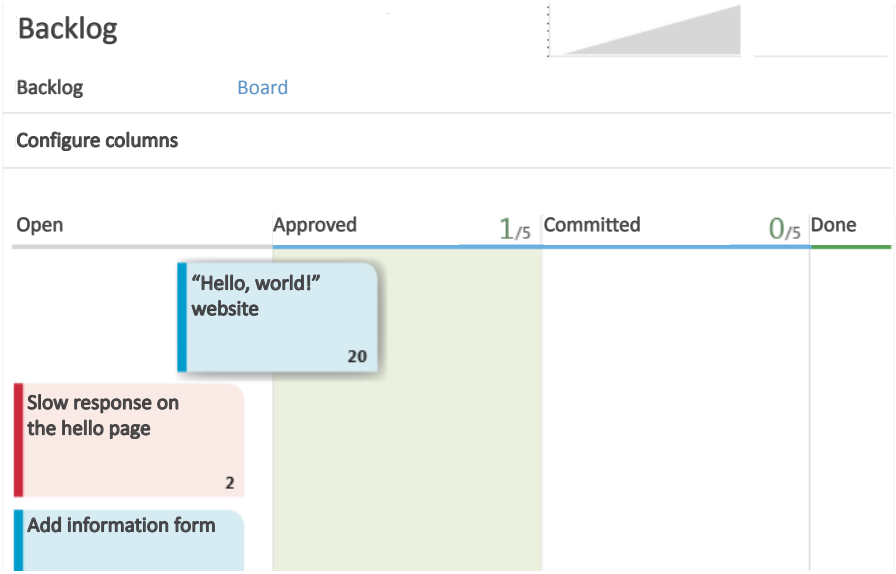

Figure 50

Reports in the Scrum template

Almost all fields defined in the Scrum work items appear in the Team Foundation Server analytics and allow creating several important reports that reflect the project progress.

1. **Release Burndown** (see Figure 51)**.**

   Total works must decrease from Sprint to Sprint.

2. **Velocity report** (see Figure 52)**.**

The Velocity graph shows planned efforts of the team for the completed tasks in Sprint. The source of the raw data is your product backlog. Each sprint that has been assigned to the team project or to the team appears along the horizontal axis. The vertical axis indicates the sum of all effort for all backlog items assigned to the indicated sprint that have been

closed (State=Done). The vertical axis shows effort in whatever unit your team uses (for example, man-hours, story points).

The graph displays a horizontal line that represents the average velocity across all the sprints:

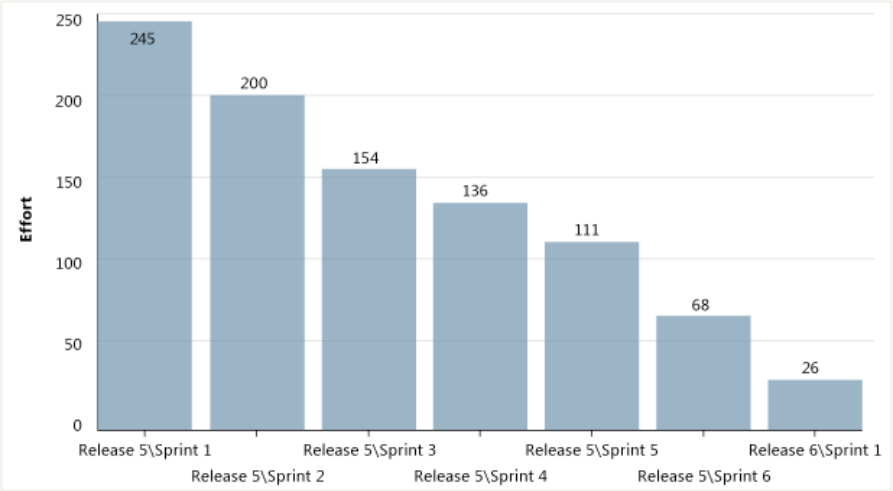

Figure 51

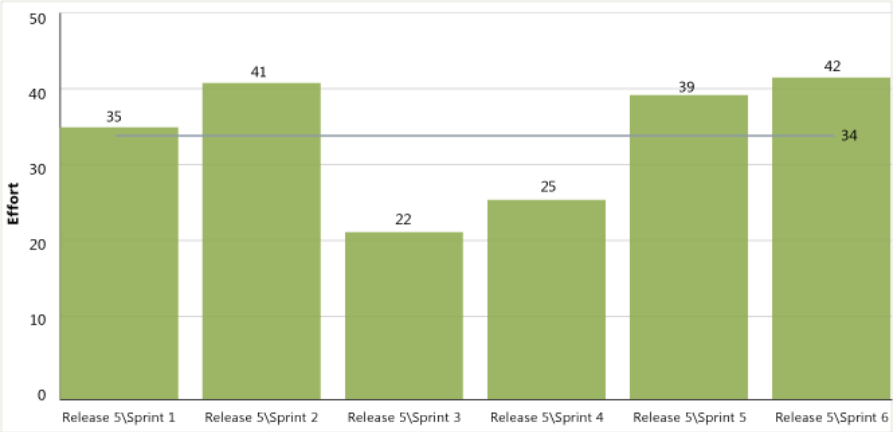

Figure 52

3. **Build Success Over Time, Build Summary reports**.

They display the build states of the project in terms of platforms and tests run. These reports allow you to track progress, assess the scope of changes and product quality.

The Scrum template has enough functionality to organize and automate team's work with TFS according to the Scrum principles.

8. HOMEWORK

Task 1

Install a trial version of MS Project 2016.

In the task list, add links between tasks and task duration, as described in the table below.

A **predecessor task** is a task that must be finished before a successor task can start.

Task Code	Duration	Predecessor Task
	2	—
B	15	A
C	10	A
D	13	A
E	18	A
F	15	C, D
G	10	F, B
H	5	E, G

What is the project implementation deadline?

Task 2

In the resource list, create a resource, name it **Developer**, and set its available load to 100%.

Assign it to all tasks of the project.

Even the load of this resource using the **Level All** button on the **Resource** tab.

Did you manage to remove the resource overload?

What project implementation deadline have you got after leveling?

Lesson 4

ABOUT SCRUM IN DETAILS

© Maksim Yakubovich
© STEP IT Academy
www.itstep.org