The Software Engineering Team

The software engineering team involves **representatives of the software sustainment team** to ensure that the **architectural configuration and design mechanisms provide a context for customer and product support**. **The software sustainment representatives must sanction the structural design solution declaring it may be maintained, within established sustainment plans, resources, and schedule constraints.**

- **Team Goals**

Planning. Goal setting.

Building software for the mere sake of building software is not efficient. You have to know where you'll lead your product to, more or less. **Unclear instructions and miscommunications are the cause of many problems while teamworking. Different organisations have different needs, so your goals shall vary as per each's distinctions and requirements. Each team should know exactly what they're working toward including what goals and when they need to be hit.**

- **Team Structure**

The structure of a **team is like a skeleton**. **Without a strong, robust skeleton, a body cannot function at its best. The size of the team and the role each person plays must be in harmony with each other as well as the team body (goals, visions, missions, etc).**

On the other hand, a team, pretty much as a skeleton, **is quite fragile**, to a degree that even one malfunction component can cause the whole figure to collapse, which leaves you with a tremendous pressure to keep everything in control.

Understanding the complication in deciding an ideal size for your team as well as a suitable task-allocation tactic, in this part.

- **Team Size**
  - **The quantity, in the context of a team, does play an essential role in boosting the performance**.
  - To be exact, **it's a factor that directly influences the quality of your process and your product – those 2 are always in close relation.**

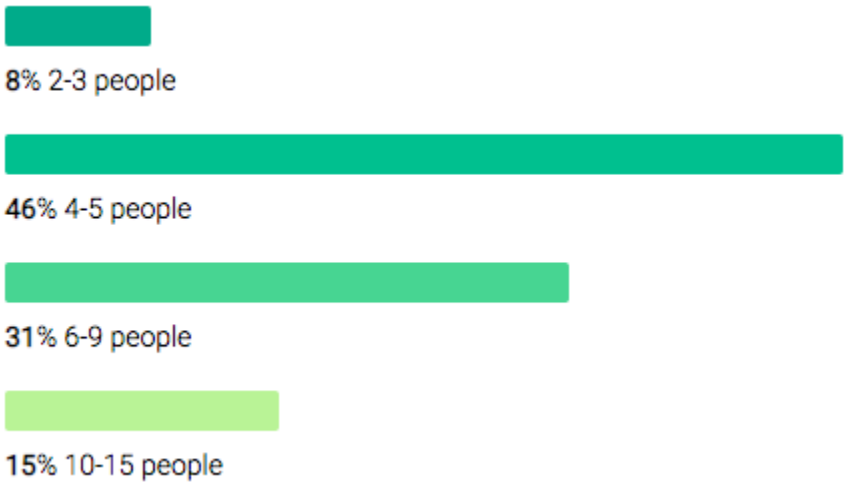
- **Size team is the key factor affecting a team's productivity**.

But when it comes to assembling the team, it all boils down to the following key factors:

- **Complexity of your project**
- **Budget**
- **Deadline**
- **Available resources**

Based on these important elements you can decide what kind of team size is your solution. **According to Scrum methodology, the optimal team size is between 3 and 9 members with 7 being the most perfect fit.** But that doesn't mean that you have to strictly follow this rule. If your software project requires a bigger team, it doesn't mean you are bound to have problems in managing it or establishing the communication. **The key here is to carefully manage your team in accordance to your project requirements.**

According to a poll on HackerLife, **the most optimal size for an IT project is within 4-5 people and on average the team comprises of around 6 people.** This conclusion is proven by an experiment called the Ringelmann effect, also called social loafing phenomenon.



➢ **How many members are there in your team?**

The experiment involves increasingly large groups of people pulling a rope, **Ringelmann deduced that for every person added beyond 5-6 people, individual contributions to the group became smaller.**

In simple words:

The bigger the team are, the less effort the workers put into common tasks as their contribution are less visually recognised.

Yet, **if a team is too large there is a potential risk of productivity of each individual employees decreasing while the composition of the team gets bigger.**

Notwithstanding, **some specific tasks require a large number of software development team members.**

For example:

Sports teams composed of a different number of players. 11 players for a football team, 6 players for a volleyball team and etc but on average not more than fifteen. Members of small teams are more devoted to others and everyone's success.

Still, in software development, larger software applications take more effort to develop than smaller ones. There are a lot of approaches to managing huge software development teams.

A long-term outsourcing development project may require up to 20 members. **However, whether a project is successful depends mostly on the human factor.**

IT development teams often conquers several problems like human resources management, lack of communication, and motivational loss.

- **Team Roles and Responsibilities**

Can work be done by a single person?

- **Yes sure. If it's not teamwork.**

In order to maintain a sense of justice and equity within your team as well as for the work to be done as fast, as excellent as possible, equipping yourself with a set of delegation skill is necessary.

Having too many people doing the same jobs, or even worse, having the wrong person to do tasks that are way too much for them are always the start of corruption, whether you can foresee it or not.

- **How to choose the right person with the right role and responsibility?**

To avoid having too many people with similar roles in the same team, **you should first list a number of candidates with regard to their technical background.**

Then, trying to evaluate their soft skills and finally, make a decision based on some extra factor like working style, personal characteristics and logical thinking.

For example:

A web development team must be composed of UX/UI designer, quality assurance (QA) tester, server administrator and project manager (PM).

In a mobile development team, developers should have experience in working with a platform of choice and its languages: Java/Kotlin for Android, Objective-C/Swift for iOS and C# for Windows Phone.

For every IT project, these roles are indispensable, so make sure you recruit wisely.

You should also do the same thing in order to avoid suffocating your members with never-in-the-wildest-dreams tasks.

It makes no sense if you assign a level-10 task to a level-4 member since the possibilities of them messing it up is high, not to mention the deterrence they get when doing something far too hard for them.

As such, the point is not to give everyone an equal number of tasks but to allocate adequate and suitable tasks to each of them.

That's left us with you doing some researches about your members beforehand. Learning your member core competence, range of professions and the level of pressure-taking are all essential steps for an effective delegation.

Divide the task in a way that's challenging, but achievable. Setting reasonable deadlines to reminds everyone to stick to their task, therefore, avoids slowing down the working progress of the whole team.

Those guarantee everyone can work their best in their power yet still be motivated to be better versions than they were yesterday.

Team Engagement

Commitment. Engagement. Sense of inclusion and alliance. All the ingredients you know should be in the pot, but find it hard to buy and to measure how much shall be added.

How about me showing you a method?

Encouraging communication

Communication is a two-way street and employees should feel like they can add to the conversation both with superiors and co-workers.

The more the team communicates, the sooner the goal may achieve.

Before starting a big project, you could plan a casual meetup beforehand that allow colleagues to get coffee or dinner with each other.

The coffee time or tea break also boost team members to get to know each other on a more personal level to increase mutual trust, respect and understanding each other's characteristics and strength.

When people build trust on a personal level, the benefits carry over into work projects.

On the other hand, the workers would not hesitate to give feedback about their works. In turn, group efforts have a higher likelihood of success.

Praising and recognising team and individuals who have the best performance will boost confidence and morale, encouraging teams and individuals to keep up the good work as well.

Make communication a priority

Communication this is the last piece of the puzzle that very often is either the reason of success or failure software development team. You can have all the other elements right on track and properly functioning but if your team lacks this last element it can sufficiently harm your product and the whole process.

As hard as your team is at work on a daily basis, they spend a sufficient amount of time interacting with each other. And if your team members can clearly communicate their needs and project demands to one another, it can boost their collaboration and improve the work process overall. It is also capable of driving creativity and innovation inside the team. It comes in handy when discussing the project's details

and requirements with stakeholders, negotiating timelines and generally, when trying to understand what is it that the stakeholders want.

Great communication skills are an important soft skill for any software engineer. But you should treat their ability to communicate as a given. Your task is to nurture and encourage the communication inside your software development team and make it a part of your normal process by practicing daily standups, design and code reviews, writing documentation, presentations and also some social events.

Helping developers grow professionally

Technology is a competitive world. This world requires developers to constantly upgrade themselves and desire for new innovation.

Attracting competent developers is a challenge, retaining them is even more difficult. That is why HR managers and company leaders are paying so many concerns about developing employees.

When you invest time and money in your developers, not only do you boost their loyalty, you also help them cultivate new skills that will benefit your business.

For example:

Some companies introduce mentoring programs, which pairing employees with mentors.

Employees would choose to pair up with a person who they feel the most comfortable with and want to learn more from mentors.

Other companies offer workshops, where they share working experience, tips for work and anything that is helpful to the audiences. These workshops and mentoring efforts not only improve employees skills but also strengthen bonds between team members.

A monthly workshop in Savvycom.

Just like in Savvycom, internal workshops are remained monthly to keep our developers updated with new technologies or to share each other's experience and to train new skills.

Besides, Savvycom leaders often send our developers to external training to improve their skills with giant technology firms like AWS, Google or Apple.

Team Performance Evaluation

Evaluation is the ace of controlling.

Without evaluation, can you realise what's good and what's bad, what to be maintained and what to be improved in order for your project to rock?

Through evaluation, you can see where your team is standing, recognise problems before things fall out in a messy bowl of soup and act on the necessities immediately so that your project could end up a success.

Evaluation is a necessary step for effective development team building.

It helps you not only track the status of the whole team, but also the progress of each individual.

Encouragement, appreciation and appraisal play a huge part in members' motivation, development, dedication and commitment, therefore, you must always have a heads-up on who are the best players, the people that are worthy of receiving credits for their hard word and prosperous results.

You shall also be aware of the falling-behind because they are likely to lose hope and productivity in their work.

They need consolations, encouragements, or simply someone who listens to them and has their back.

They need you as an empathetic, understanding manager to show them that work is still fun, that even though they are doing not too well, they are recognised for the effort, and that they have your belief and support to do better in the future.

Therefore, they would turn back to their lovely, happy, satisfied-with-work selves and function 10 times better than they did before.

But to do that? Evaluate each of them.

- **How to track down your team performance efficiently?**

Here are the tools and frameworks that you can apply right away.

- **360-degree feedback**

360-degree feedback is one of the most popular methods for evaluation because of its confidential nature.

In this framework, managers receive anonymous feedback from individuals with whom they interact frequently in the course of daily operations. These can include internal and external customers, superiors, direct reports, subordinates, vendors, etc.

Evaluators are then chosen at random from the above groups to avoid skewed results. Based on the interpretation from the evaluators, you will be able to filter out the information you need for evaluation and appraisal.

- **Balanced scorecard**

Another method you shall put into use is the balanced scorecard. This approach combines quantifiable information, such as sales quotas and budgetary requirements, with performance standards particular to the position.

This method utilizes key performance indicators, or KPIs, to track how well the employee has reached short- and long-term goals.

These take into account the employee's career growth and adherence to best practices as set forth by the individual organization.

- **Self-evaluation**

Lastly, self-evaluation. This allows the employee to rate himself/herself against the same or similar criteria used by his supervisor. Often this involves qualitative and quantitative criteria.

This method can raise the credibility level of the process in the view of the employee; especially when the employee's self-assessment score lines up closely with that of the supervisor.

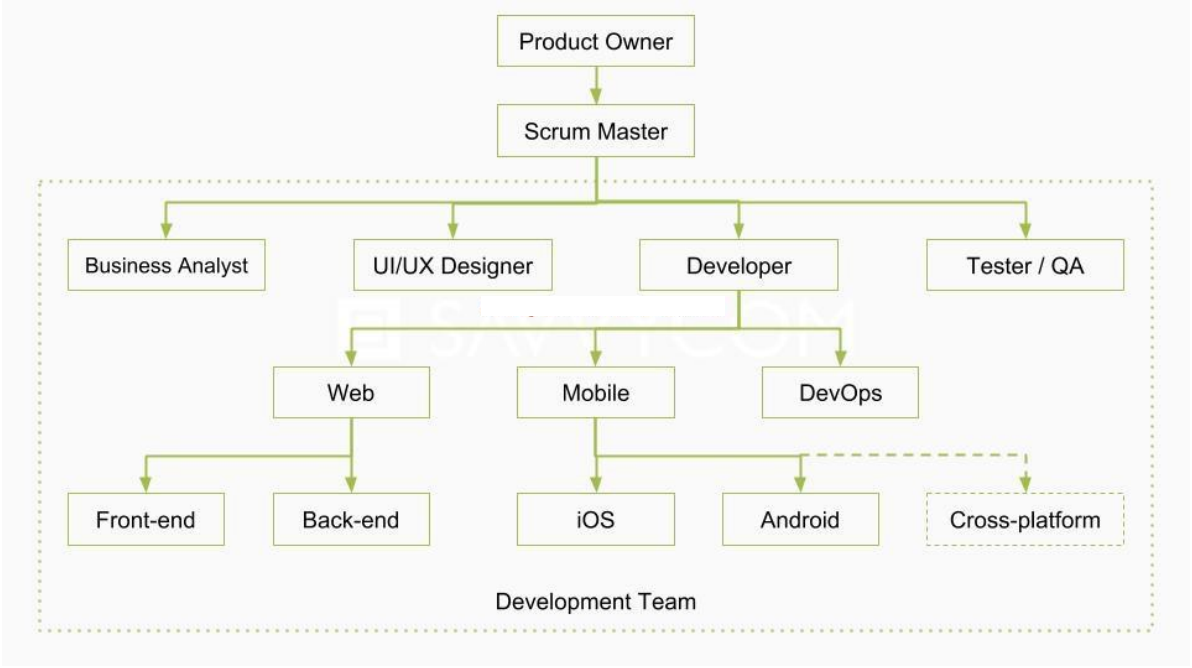
When the scores are somewhat at odds with one another, this tool offers discussion processes whereby these differences can be discussed in a safe, constructive manner.

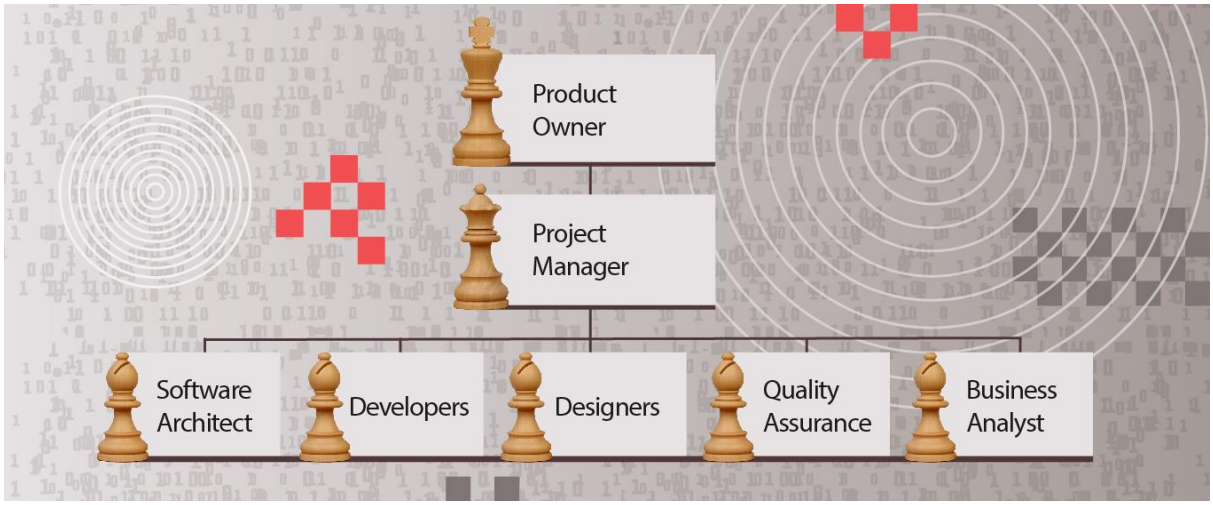
- **Applying Agile Methodology and Developing a Scrum Team**

With nearly 10 years of bringing the most advanced technology solutions into the business world, Savvycom is proudly confident to provide you with a professional working process, an awesome team structure.

Software Project Team Roles and Responsibilities

Successful development projects take careful planning, a talented team and collaboration of a project's team members, both internal and external (client representatives). Software projects only move forward when the key team members are in place.





Software development team diagram

- **Product Owners**

Located at the top of the diagram, in case of an outsourced company, could be the company's client. They have a vision of what their product is, whom it serves, and how it should be. Product owner, in the case of an outsourced project, this is the client with a vision of how the end-product should look, who are the end-users and what it should do.

Commonly, they will give the project manager a set of product requirements and expectations that he/she desires their products will have.

Leading and managing the whole team is a Project Manager (PM), who identifies where and how the team members will go to achieve their goals.

Project Manager

- Develop a project plan
- Manage deliverables according to the plan
- Recruit project staff
- Lead and manage the project team
- Determine the methodology used on the project
- Establish a project schedule and determine each phase
- Assign tasks to project team members
- Provide regular updates to upper management

PMs are right under the Product Owner in the diagram. They are in charge of optimising the work efficiency and ensuring the product meets all the requirements.

PMs will take charge of recruiting team members, identify their strengths and weaknesses to assign prosper tasks for everyone in the team. More importantly, they are also responsible for tracking team members' workflow and make sure they strictly adhere to the timeline.

Project manager is a person responsible for managing and leading the whole team. Their role is to efficiently optimize the work of the team, ensure the product is meeting the requirements and identify the goals for the team.

Software architect is a highly-skilled software developer that has to think through all the aspects of the project and is responsible for making high level design choices, as well as select technical standards (for instance, determines the technology stack to use).

Analyst

The Analyst is responsible for ensuring that the requirements of the business clients are captured and documented correctly before a solution is developed and implemented. In some companies, this person might be called a Business Analyst, Business Systems Analyst, Systems Analyst or Requirements Analyst.

- **Business Analyst Business Analysts (BA).**

Business Analyst's role is to uncover the ways to improve the product. They interact with stakeholders to understand their problems and needs, and later document and analyze them to find a solution.

BA acts as a bridge to connect business challenges with technology solutions. Their role is making a set of questions to ask their clients in order to have a deeper understanding of the project.

This can be done by an array of assignments such as asking clients for explaining requirements for developers, gathering comprehensive information about project needs and conducting user acceptance test.

Functional manager:

- Assign project
- Discuss how well person is doing that work and if person wants to continue doing it (providing opportunities for growth)
- Gather information from other PMs to write the evaluation
- Work with employee to set and coach on career goals

Operational Manager

An operations manager is a senior role which involves overseeing the production of goods and/or provision of services.

It's an operations manager's job to make sure an organization is running as well as it possibly can, with a smooth efficient service that meets the expectations and needs of customers and clients.

Main Job Tasks and Responsibilities

- **Coordination and Supervision** — Coordinate, manage and monitor the workings of various departments in the organization.
- **Financial** — Review financial statements and data. Utilize financial data to improve profitability. Prepare and control operational budgets. Control inventory. Plan effective strategies for the financial well-being of the company.

- **Best Practices** — Improve processes and policies in support of organizational goals. Formulate and implement departmental and organizational policies and procedures to maximize output. Monitor adherence to rules, regulations and procedures.
- **Human Resources** — Plan the use of human resources. Organize recruitment and placement of required staff. Establish organizational structures. Delegate tasks and accountabilities. Establish work schedules. Supervise staff. Monitor and evaluate performance.
- **Production** — Coordinate and monitor the work of various departments involved in production, warehousing, pricing and distribution of goods. Monitor performance and implement improvements. Ensure quality of products. Manage quality and quantity of employee productivity. Manage maintenance of equipment and machinery. Provide technical support where necessary.
- **Communication** — Monitor, manage and improve the efficiency of support services such as IT, HR, Accounts and Finance. Facilitate coordination and communication between support functions.
- **Sales and Marketing and Customer Service** — Manage customer support. Plan and support sales and marketing activities.
- **Strategic Input** — Liaison with top management. Assist in the development of strategic plans for operational activity. Implement and manage operational plans.
  - Planning and controlling change.
  - Managing quality assurance programs.
  - Researching new technologies and alternative methods of efficiency.
  - Setting and reviewing budgets and managing cost.
  - Overseeing inventory, distribution of goods and facility layout.

Business analyst duties:

- Assist in defining the project
- Gather requirements from business units or users
- Document technical and business requirements
- Verify that project deliverables meet the requirements
- Test solutions to validate objectives

QA Manager

The QA role works with the Functional Analyst (FA) and the Solutions Architect (SA) to convert the requirements and design documents into a set of testing cases and scripts, which can be used to verify that the system meets the client needs. This collection of test cases and scripts are collectively referred

to as a test plan. The test plan document itself is often simple providing an overview of each of the test cases. The testing cases and scripts are also used to validate that there are no unexplained errors in the system.

The test plan is approved by the Subject Matter Experts (SMEs) and represents the criteria to reach a project closing. If the test cases and scripts in the test plan are the agreed upon acceptance criteria for a project then all that is necessary is for project closure is to demonstrate that all of the testing cases and scripts have been executed successfully with passing results.

A test case is a general-purpose statement that maps to one or more requirements and design points. It is the overall item being tested. It may be a specific usability feature, or a technical feature that was supposed to be implemented as a part of the project.

Test scripts fit into the test cases by validating that case. Test scripts are step-by-step instructions on what to do, what to look for, and what should happen. While the test cases can be created with nearly no input from the architecture or design, the test scripts are specific to how the problem was solved by the software development team and therefore they require an understanding of not only the requirements, but also the architecture, design, and detailed design.

Change Control Board/change manager

The Change Control Board is usually made up of a group of decision makers authorized to accept changes to the projects requirements, budget, and timelines. This organization would be helpful if the project directly impacted a number of functional areas and the sponsor wanted to share the scope change authority with this broader group. The details of the Change Control Board and the processes they follow are defined in the project management processes.

Client

This is the people (or groups) that are the direct beneficiaries of a project or service. They are the people for whom the project is being undertaken. (Indirect beneficiaries are probably stakeholders.) These might also be called “customers”, but if they are internal to the company, LifecycleStep refers to them generically as clients. If they are outside your company, they would be referred to as “customers”.

Client Project Manager

If the project is large enough, the business client may have a primary contact that is designated as a comparable project manager for work on the client side. The IT project manager would have overall responsibility for the IT solution. However, there may be projects on the client side that are also needed to support the initiative, and the client project manager would be responsible for those. The IT project manager and the client project manager would be peers who work together to build and implement the complete solution.

Solution team/software architect

- define, document, and communicate it
- make sure everyone is using it, and using it correctly
- make sure that it comes out in stages in a timely way so that the overall organization can make progress before it's complete
- make sure the software and system architectures are in synchronization
- act as the emissary of the architecture
- make sure management understands it (to the detail necessary)
- make sure the right modeling is being done, to know that qualities like performance are going to be met
- give input as needed to issues like tool and environment selection
- identify and interact with stakeholders to make sure their needs are being met
- make sure that the architecture is not only the right one for operations, but also for deployment and sustainment
- resolve disputes and make tradeoffs
- resolve technical problems
- maintain morale, both within the architecture group, and externally as well. The latter is done by providing a sound design, when needed, and providing good presentations and materials to let everyone know the organization is on the right track.
- understand and plan for evolutionary paths
- plan for new technology insertion
- manage risk identification and risk mitigation strategies associated with the architecture

Process Analyst:

The process analyst leads and coordinates business use-case modeling by outlining and delimiting the organization being modeled; for example, establishing what business actors and business use cases exist and how they interact. The business process analyst is responsible for the business architecture. He/she is shown below as responsible for Artifact: Business Object Model because of this overall architectural responsibility, even though Role: Business Designer creates and maintains it.

Main Duties

Commonly, the main duties of this role include:

- Stakeholder Relationship Management
- Documentation, Training and Support
- Process Review and Enhancement

Duties :

- Assess the situation of the target organization where the project's end-product will be deployed.
- Understand customer and user requirements, their strategies, and their goals.
- Facilitate modeling of the target organization.
- Discuss and facilitate a business engineering effort, if needed.
- Perform a cost/benefit analysis for any suggested changes in the target organization.
- Discuss and support those who market and sell the end-product of the project.

Test analyst/Tester/test Engineer

The Test Analyst role is responsible for initially identifying and subsequently defining the required tests, monitoring the test coverage and evaluating the overall quality experienced when testing the Target Test Items. This role also involves specifying the required Test Data and evaluating the outcome of the testing conducted in each test cycle. Sometimes this role is also referred to as the Test Designer, or considered part of the Tester role. QA or tester is responsible for the Quality Assurance and makes sure the product is ready to use.

This role is responsible for:

- Identifying the Target Test Items to be evaluated by the test effort
- Defining the appropriate tests required and any associated Test Data
- Gathering and managing the Test Data

- Evaluating the outcome of each test cycle
- **Work process**

In general, after thoroughly understand user requirements, UI/UX designer will illustrate design ideas using process flows, storyboards, and sitemaps as well as design graphical user interface elements like menu, tab, and widgets.

On the premise of UI/UX, developers will use an appropriate programming language/required technologies to create clean, efficient codes based on specifications.

The team members must collaborate well with each other to build and implement ideal functional programs.

In this phase, we have to differentiate three kinds of developers.

they are

- Web developers
- App developers
- DevOps.

These two first terms are simply distinguished by their platforms.

Web application developers engaged in developing and designing web/network applications and make sure they work well on the internet.

As a web contains both the interface – what users can see and the code – which is only be seen by the developers, it is typically divided into Front-end developers and Back-end developers.

Front-end developers manage everything that users visually see first in their browser or application, in other words, they are responsible for the users' sense.

Back-end developers manage the server-side of an application and everything that communicates between the database and the browser.

Mobile applications developers include iOS and Android developer teams, sometimes could be cross-platform developers who can play well on both iOS and Android system.

Another essential part of a software developer is DevOps. They are a combination of Development & Operations team.

The main goal of DevOps is to increase the quality of the product to a great extent and to increase the collaboration of Development and Operation team as well so that the workflow within the organisation becomes smoother.

Finally, Quality Assurance (QA) is at the bottom level in the diagram.

After having the first version of the product, the QA department will implement testing whether the draft has any bugs or not.

They convert requirements and documents into a set of testing cases and scripts, which can be used to verfiy if the application meets the client's expectation.

These testing cases will be applied continuously until there is no unexplained error found in the system.

Building an effective team is nowhere near easy. It requires a rotation of planning, organising, leading and controlling, in combination with other interpersonal skills and management of the various environment.

Lots of efforts, persistence, observation and improvision must be exerted while facing up with the changing situation within the team, therefore you shall never put your guard off.

Developers or software engineers are team members that apply their knowledge of engineering and programming languages in software development.

Experience designers ensure that the product is easy and pleasant to use. They conduct user interviews, market research, and design a product with end-users in mind.

Software Architecture:

The basic objectives of a solution architect could be as following.

- Be mindful of the scope of the project.
- Across the SAP modules, he would do the followings.
  - Understand client's requirements and its business viabilities

- Determine the gaps between requirements and SAP functionalities and define workable solutions to bridge the gaps
- Design the solutions in detail with the help from the consultants and business
- Plan the closure of the solutions and see through the implementation of the solutions

Database Administrator

A Database Administrator is a specialist that models, designs and creates the databases and tables used by a software solution. This role combines Data Administrator (logical) and DBA (physical).

The designer is responsible for understanding the business requirements and designing a solution that will meet the business needs. There are many potential solutions that will meet the client's needs. The designer determines the best approach. A designer typically needs to understand how technology can be used to create this optimum solution for the client. The designer determines the overall model and framework for the solution, down to the level of designing screens, reports, programs and other components. They also determine the data needs. The work of the designer is then handed off to the programmers and other people who will construct the solution based on the design specifications.

Developer

The Developer is responsible for the actual building of the solution. For more information on this role,

Project Team

The project team consists of the full-time and part-time resources assigned to work on the deliverables of the project. This includes the analysts, designers, programmers, etc. They are responsible for:

- Understanding the work to be completed
- Planning the assigned activities in more detail if needed
- Completing assigned work within the budget, timeline and quality expectations
- Informing the project manager of issues, scope changes, risk and quality concerns
- Proactively communicating status and managing expectations

The project team can consist of staff within one functional organization, or it can consist of members from many different functional organizations. A cross-functional team has members from multiple organizations. Having a cross-functional team is usually a sign that your organization is utilizing matrix management.

Sponsor (Executive Sponsor and Project Sponsor)

This is the person who has ultimate authority over the project. The Executive Sponsor provides project funding, resolves issues and scope changes, approves major deliverables and provides high-level direction. They also champion the project within their organization. Depending on the project and the organizational level of the Executive Sponsor, they may delegate day-to-day tactical management to a Project Sponsor. If assigned, the Project Sponsor represents the Executive Sponsor on a day-to-day basis and makes most of the decisions requiring sponsor approval. If the decision is large enough, the Project Sponsor will take it to the Executive Sponsor for resolution.

Stakeholder

These are the specific people or groups who have a stake, or an interest, in the outcome of the project. Normally stakeholders are from within the company, and could include internal clients, management, employees, administrators, etc. A project may also have external stakeholders, including suppliers, investors, community groups and government organizations.

Steering Committee

A Steering Committee is a group of high-level stakeholders who are responsible for providing guidance on overall strategic direction. They do not take the place of a Sponsor, but help to spread the strategic input and buy-in to a larger portion of the organization. The Steering Committee is usually made up of organizational peers and is a combination of direct clients and indirect stakeholders. Some members on the Steering Committee may also sit on the Change Control Board.

Subject Matter Expert

A Subject Matter Expert (SME) has superior (expert) knowledge of a discipline, technology, product, business process or entire business area.

SME Duties:

- Review test cases for integration testing associated with the inventory management system.
- Help validate user requirements for payroll application.

- Conduct code walkthrough for accounts payable interface to legacy system.
- Review requirements traceability matrix, and ensure that requirements have coverage.
- Help refine and determine feasibility, correctness, and completeness of end-user's requirement.
- Provide input for the design and construction of test cases and business scenarios.
- Help answer questions associated with the design of the status quo application, its features, and its capabilities.
- Validate executed test results.

Suppliers / Vendors

Suppliers and vendors are third party companies or specific people that work for third parties. They may be subcontractors who are working under your direction, or they may be supplying material, equipment, hardware, software or supplies to your project. Depending on their role, they may need to be identified on your organization chart. For instance, if you are partnering with a supplier to develop your requirements, you probably want them on your organization chart. On the other hand, if the vendor is supplying a common piece of hardware, you probably would not consider them a part of the team.

Tester

The Tester ensures that the solution meets the business requirements and that it is free of errors and defects.

Users

These are the people who will actually use the deliverables of the project. These people may also be involved heavily in the project in activities such as defining business requirements. In other cases, they may not get involved until the testing process. Sometimes you want to specifically identify the user organization or the specific users of the solution and assign a formal set of responsibilities to them, like developing use cases or user scenarios based on the needs of the business requirements.

Steps to Building an Effective Software Development Team

When kicking off a new software development project, naturally we all intend for it to be a success. But in order to be one you have to rely on a strong core — your software development team. While it is an undeniable truth that all teams are different in terms of their work style and the unique ecosystem within, there are some pretty common elements to it. We all expect our team to consist of highly

experienced and skilled individuals. But is it the only prerequisite? In order to make a group of professions a truly effective software development team you need to remember about some elements to take into account:

Define the kind of team type that fits for your project

First things first, before we dive deeper into what pay attention to when building a software development team, you need to decide on the kind of team you want to create. Establishing a clear **software development team structure** is an important first step into an overall success of your project.

Here you have to choose from the three main types:

Generalists — are Jack of all trades, so to say. Generalists possess a broad range of knowledge and expertise. Generally, these types of teams are designed to handle end-to-end solutions. The advantage of generalists is in the fact that they can provide a complete solution to the problem. However, they also have some drawbacks — if your project requires a higher level of expertise in some area, generalists will find themselves at a loss as they may lack the knowledge and skills.

Specialists — unlike generalists, this type of teams consists of members that are highly skilled in a particular field. The advantage of specialists is clear — they can address a specific matter with all their knowledge and expertise, resulting in more efficient and effective work. On the other hand, communication is not exactly a forte of this type of software development team. Often times being a very narrow specialist, team members may lack general understanding of what are the roles of other team members, and thus making communication between them somewhat ineffective.

Hybrid team — as you've probably guessed, this type of team is a mix of the previous two. It seems like this approach combines the best from the two worlds, where specialists focus on functional parts, and generalists are responsible for the communication and cooperation inside the team. This dream of a team, however, comes with a constraint — usually this type of software development team is more difficult to gather in terms of time and financial resources.

Decide on the software development team size

Now that you've established what kind of team you want to build, let's talk about the size. There really isn't a magical number when it comes to the size of your software development team. Smaller teams are easier to manage on the one hand, but in this case, every team member plays a crucial role in the

project and losing even one person can have a significant impact on the overall result. With bigger teams the challenge is in managing the communication.

Adopt agile

Agile methodology makes your team more flexible; it allows to adapt and respond to some unforeseen changes that might come without damaging the whole process.

There's no single right formula how to build your agile software development team. Even within this approach to software development there are two types of methodology: some prefer Scrum, while others use Kanban.

The later practices real-time communication and full work transparency. It relies greatly on visualization. Actually, it's in the core of this framework. Work is constantly visually represented on Kanban board to help the team understand at what stage they currently are. On top of that, by using visualization, they can work out what are they stumbling blocks, what slows them down and find ways to overcome them.

Scrum is the most popular Agile framework that breaks down a large project into smaller chunks (sprints) and reviews and adapts them along the way. Sprints can last from a week to a month in duration. If you are adopting Scrum methodology, the structure of your software development team will include a very important element — Scrum master. He or she makes sure that the team sticks to the agile values and principles and follows the process that the team has agreed on.

Regardless of the framework you choose, agile will help your team to deliver faster and efficiently.

Agood team isn't something that happens on its own. But why do we need one at all? Don't good professionals form a good team by default? The thing is, it's not enough to lock them in a room and give them a deadline. An efficient team isn't just about professionalism; it's also about how the team members interact. Only a truly efficient team can deal with the workload faster and be more productive. And generally, it's always nice to work with a team that, well, doesn't mess up.

Statistics show that, in fact, the main reason why projects fail is a lack of confidence in the project's success: "75% of respondents admit that their projects are either always or usually "doomed right from the start." But why do you think it happens, and how can you change it? In order to function properly, the team needs to know all the aspects of the process, their duties and responsibilities, and believe in

what they do — and you're the one to convince them. Here's how to put together a team you can rely on, an effective team you're confident about, one that you can look at and say, "We're going to change the world!"

Creating a Software Development Team: Where Do You Start?

Like any other big task that carries a lot of responsibility, building a successful software development team may seem daunting at first. Where do you start? How do you know who you need and where to look for them? First things first: write down what business task the team is supposed to solve. Front-end or back-end development? What is the team's role? As soon as you have that clear, start shaping the team.

Just so you have an idea of how this process unfolds, here are a couple of examples from our own experience.

If our task is to finish or fix a project started by someone else, first we need to examine the existing code and understand the complexity of any potential changes. For cases like these, all we need at the start is to involve a back-end developer and a front-end developer.

If you need to make test cases, then you'll have to call for a QA (Quality Assurance) to backup the project. Meanwhile, if you're working with a project from scratch, usually two back-end and one front-end developer, one QA, and one project manager should be enough.

Done putting together the team? The next step is to define the team's objectives and/or results, as the goals and tasks should be transparent and clear to everyone. You can do it based on business goals, and adapt all the processes within the team.

Effective Teams: What to Strive For and How to Achieve It

Before we go any further, let's do a quick turn to what makes up a really effective team. Consider these five steps if you want to create a truly successful development team that will help you achieve your goal.

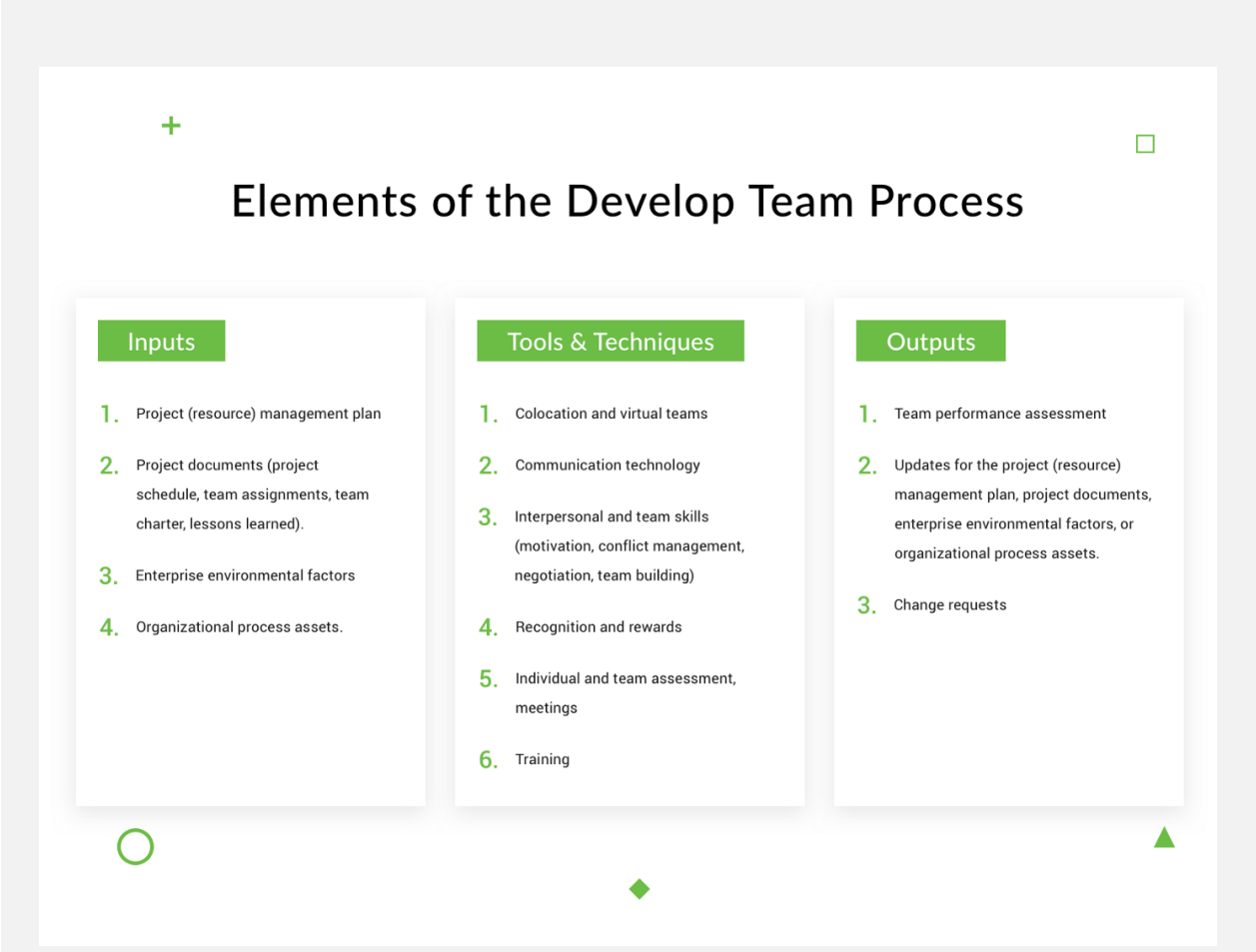
Step 1. Appreciate the power of teamwork. Be smart about identifying your employee's skills and assigning tasks that match their abilities. But also make sure the team members appreciate each other's functions in the team and are aware of how their personalities complement each other. The group's cohesion will lend them the true power to focus on the common goal and reach it.

Step 2. Find the right people. Choose candidates who bring varied experience and perspective to your project. The more sides you cover, the better your product will be at launch and the less you will have to change or improve later on.

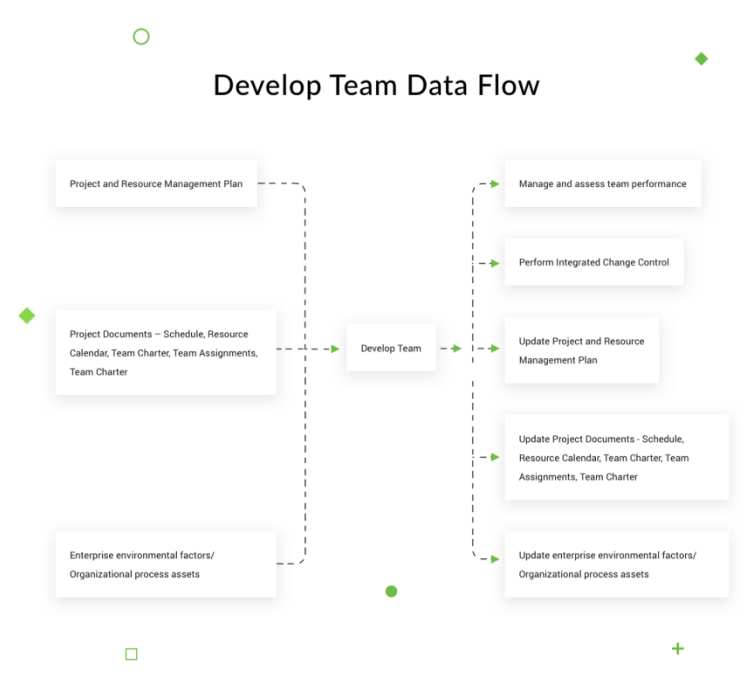
Step 3. **Learn to delegate**. Your job is basically to make sure that your team can do theirs. Once you've set goals and guidelines, let the team members do their thing. Delegate authority and provide access to tools to your development team.

Step 4. Track progress. The thing is, when people like what they do, they tend to get carried away. Your task is to monitor progress and make sure the project stays on track. Provide a means to share concerns, and discuss the project's status regularly. This way, you'll see if the team is working well or if you have to re-assign roles. However, let the team overcome some of the obstacles on their own — under your guidance. Dealing with troubles in a group brings any team closer together and build confidence among team members.

Step 5. Celebrate little victories. When your team reaches a goal — or better yet, exceeds it — don't forget to give props to the team members. This will encourage the team to work better and improve team spirit. If you're not really a celebratory type, at least schedule one final meeting to thank the people who worked on your project and helped it come true.

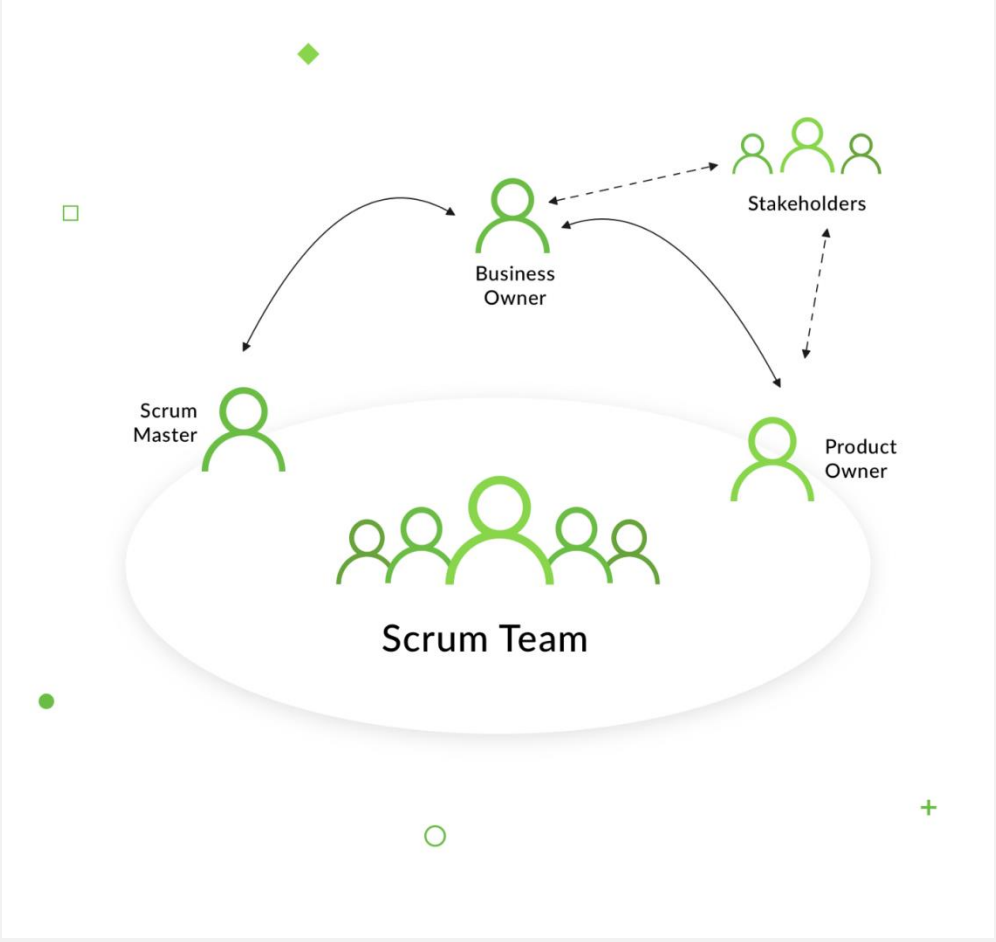


Develop Team is a process that has a data flow as well. It's based on the project management plan, project documents, and takes into account enterprise environmental factors and/or organizational process assets. All of this information is used to manage the team and the project, control changes, update project documents and enterprise environmental factors and/or organizational process assets.



Good project managers always have a set of skills they can use to maintain team spirit and effectiveness:

- Open and effective communication
- Enhancing trust between team members
- Team-building opportunities
- Constructive conflict management
- Collaborative decision making and problem-solving

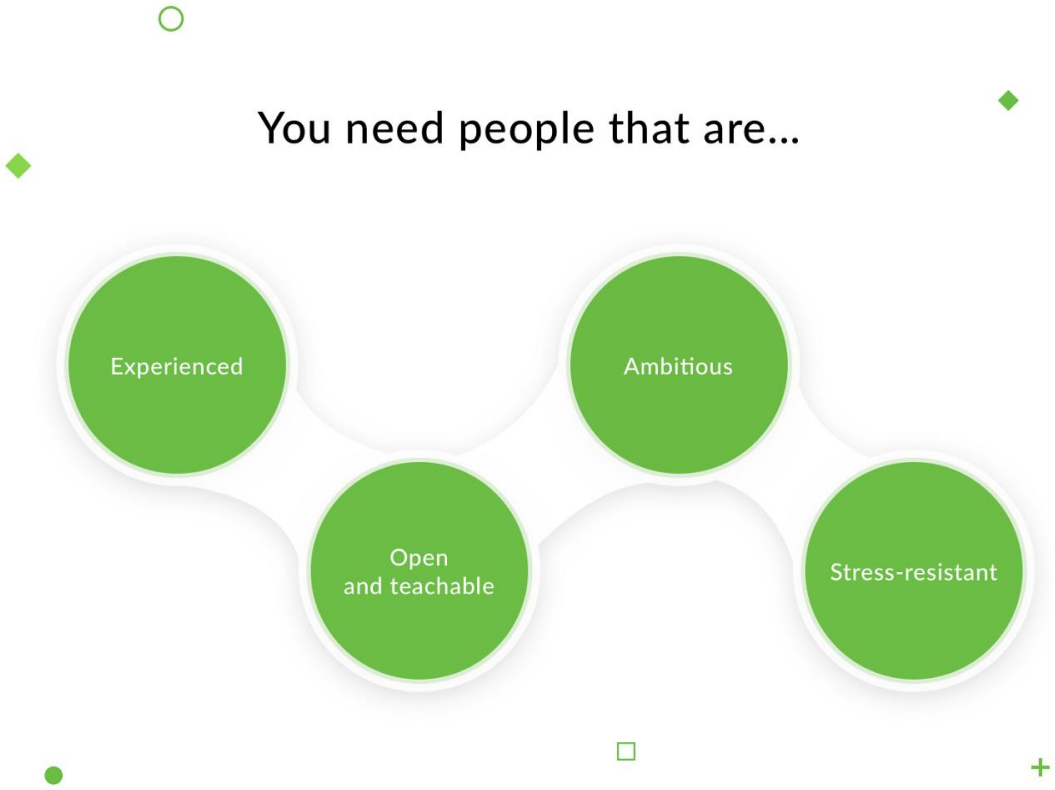


According to Scrum Guide, the Product Owner and Scrum Master are parts of the Scrum Team, too. But who makes up the rest of the team?

Do's And Dont's of Choosing a Team Lead

So, what do you do? First of all, choose candidates with vast experience and the right work ethic. Take a closer look at people who have worked for at least 5 years in the domain in question, those who have been employed by large- or medium-sized companies who value a good workplace culture.

What exactly is this culture? It's one where every single person understands that under no circumstances is it okay to bodge. Where everyone is cooperative, and the company and all of its employees are open to modern approaches to organizing work processes, and so on. So, basically, the honorary role of those large and medium IT-companies is to provide you with developers. Every more or less large company has good developers that either got stuck on their career ladder, or financially. These are exactly the people you need for your team.



And when you make the right choices in picking your team members, you'll be rewarded immensely — you'll get a team that knows what to do, is fully involved in the process, transparent in terms of doing business, and has one single goal in mind: to create a great product.

The 'dont's' are pretty easy to understand by now. Most importantly, don't assign the PM duty to people without any experience, or people coming from in-house development at rather small companies (unless they've previously had experience like what we've described above).

People that are used to small projects or in-house development, no matter how good they are, don't have the experience or outlook that will allow them to make correct big decisions (for instance, when it comes to forecasting data volumes, data load, organization of modularity, and separation of concerns). Or they will lack knowledge of the working culture of an effective team, which involves things like taking on big responsibility, planning, working Agile, reviewing design and code, testing, writing tests, etc.

On top of it, an inexperienced developer will basically learn at your expense and generate little to no value. At the same time, they will take the place of a person who could have moved your project forward in leaps and bounds.

Project Managers vs. Scrum Masters In a Software Development Team

What does a Project Manager (PM) generally do? A PM is the one who sets tasks for the team, assigns them to team members, and supervises them as they fulfill their assignments.

Scrum, however, shifts the roles. A Scrum Master is more like a guide, a nominal head of the team who looks after how the methodology is being applied. Scrum is all about independent teams. This way, a PM carries more management tasks and responsibilities than a SM. But the upside to a team led by an SM is that the team members feel a greater responsibility to their work. When people feel that their project belongs to them and them alone, instead of someone higher — like a PM — they work better, and they're more productive.

Some people mistakenly think that being a SM means abandoning the team altogether. This would inevitably lead to bad consequences, as the team isn't used to setting its own goals and finding ways to achieve them. In such cases, a Scrum Master should leave it to the team to figure out their next steps, their mistakes, but provide just enough help to enable them to get work done.

To put it shortly, in Scrum, a PM is less of a nanny, but more of a mentor for the software development team. Companies using Scrum should re-define the project manager's role in advance and be absolutely clear that the manager isn't responsible for the team completing their commitment in the Sprint.

Dedicated Development Teams: Psychological Syncing

Now you know what to look for in a Team Lead or a Project Manager, but what about the rest of the team? We can't stress enough how important it is not just to find fitting skill sets with different people, but also to match mindsets and psychological quirks. The team needs to sync on every level.

Here's our advice on how to pick team members for IT-projects at any stage with just one priority — to improve product quality:

- Look at a how the candidates' interpersonal skills go together. It's as important as their expertise, skills, achievements and experience.
- As a team leader, take into account the possibility of making additions to the team based not just on skills, but also on roles, personality types and career ambitions while maintaining the team's general direction.
- To evaluate team member compatibility, find a way to measure team relations every other month.

- Show the results of your measurements and team relations research to the team and discuss them together. This way, you'll be able to find an effective way to change the relations, if necessary.

Don't forget that all of the team members are just humans. This means, apart from professional skills, you should take into account psychological factors. Based on individual psychological characteristics, a developer team is effective when:

- The size is minimal (3 to 10 people) while including a full skill set for a particular project time frame;
- The team core is very cohesive, but lack of communication with certain team members is possible (outsourcing of minor tasks);
- The leader is highly accepted (by at least half of the team), while the member with the highest acceptance is the one with the highest level of intelligence that allows him or her to make the biggest contribution;
- The leaders' team is aimed at itself, the team members are aimed at the goal, the number of members aimed at communication is minimal;
- The team leader aims to serve, while the leaders' team's aim is mastery;
- The team has a full set of roles: idea generator, analyst, and critic. Acting roles preside over mental roles, while mental roles preside over social ones. The number of idea generators and analysts isn't large compared to other roles;
- The intellectual capital of the team should have different levels: a combination of one intellectual leader with several social leaders with lower intellectual capital is preferable.

Evaluation of Team Performance

At this point, you've learned how to build a development team. Let's say your team is working, and now it's time to see how good, how effective it actually is. Any and all evaluation of team performance should focus on two things: team results and team process.

Team results are defined by the team's main objectives. It may be anything — from better product quality, to faster delivery time, to less resources used. Team process, on the other hand, is the way the team goes about achieving results — how well team members resolve conflicts, share information, manage budgets, schedules, and interpersonal relations.

If you see how well your team works, an evaluation will allow you to set markers for future reference. But if a team faces internal obstacles, a detailed evaluation of all the team processes becomes even more critical. There are numerous methods you can use to evaluate your team's processes, but the most simple and effective include benchmarking, ongoing team discussions, and project debriefings.

As you see, building a software development team is a bit like putting together a baseball team. Everyone has and knows their role, their position on the field, and the goal everyone's longing for. It's not enough just to pick out good professionals; they have to match each other like puzzle pieces. And it's your job to really get to know each candidate to see how they work together. Learn how to appreciate teamwork, how to celebrate small achievements on your way to reaching the big goal. Create a dedicated development team that's going to be successful in this particular constellation. Better yet, build a team that will want to work together even after your project is done — this will be a true sign that you created not just a good development team, but a great one.

Forming a team that fits your project

1. Choose the team structure relevant to your project

There are 3 common types of development team structure: generalists, specialists, and a hybrid team. Each of them can be efficient in one project and get stuck regularly in another. To prevent the latter, consider your project's complexity, time and budget and then decide which team structure suits it most.

What is a "version control System"?

Version control systems are a category of software tools that helps record changes to files by keeping a track of modifications done to the code. 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. For the examples in this book, you will use software source code as the files being version controlled, though in reality you can do this with nearly any type of file on a computer.

If you are a graphic or web designer and want to keep every version of an image or layout (which you would most certainly want to), a Version Control System (VCS) is a very wise thing to use. It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also generally means that if you screw things up or lose files, you can easily recover. In addition, you get all this for very little overhead.

Use of Version Control System:

- A repository: It can be thought as a database of changes. It contains all the edits and historical versions (snapshots) of the project.
- Copy of Work (sometimes called as checkout): It is the personal copy of all the files in a project. You can edit to this copy, without affecting the work of others and you can finally commit your changes to a repository when you are done making your changes.

Types of Version Control Systems:

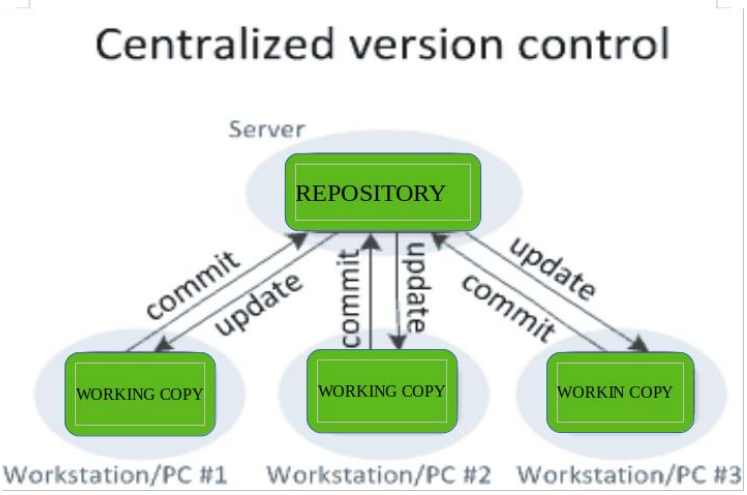
- Local Version Control Systems
- Centralized Version Control Systems
- Distributed Version Control Systems

Local Version Control Systems: It is one of the simplest forms and has a database that kept all the changes to files under revision control. RCS is one of the most common VCS tools. It keeps patch sets (differences between files) in a special format on disk. By adding up all the patches it can then re-create what any file looked like at any point in time.

Centralized Version Control Systems: Centralized version control systems contain just one repository and each user gets their own working copy. You need to commit to reflecting your changes in the repository. It is possible for others to see your changes by updating.

Two things are required to make your changes visible to others which are:

- You commit
- They update



The benefit of CVCS (Centralized Version Control Systems) makes collaboration amongst developers along with providing an insight to a certain extent on what everyone else is doing on the project. It allows administrators to fine-grained control over who can do what.

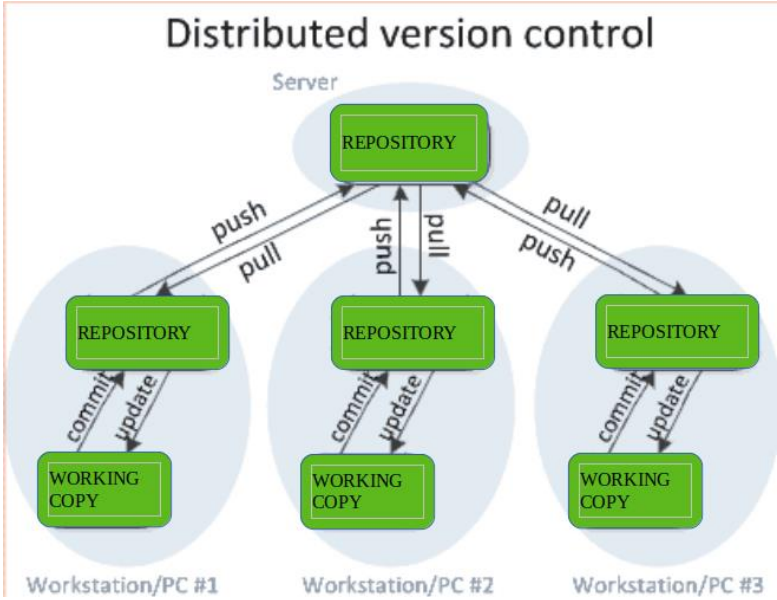
It has some downsides as well which led to the development of DVS. The most obvious is the single point of failure that the centralized repository represents if it goes down during that period collaboration and saving versioned changes is not possible. What if the hard disk of the central database becomes corrupted, and proper backups haven't been kept? You lose absolutely everything.

Distributed Version Control Systems: Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository and you need to push them in order to make them visible on the central repository. Similarly, When you update, you do not get other's changes unless you have first pulled those changes into your repository.

To make your changes visible to others, 4 things are required:

- You commit
- You push
- They pull
- They update

The most popular distributed version control systems are Git, Mercurial. They help us overcome the problem of single point of failure.



Purposeof Version Control:

- Multiple people can work simultaneously on a single project. Everyone works on and edits their own copy of the files and it is up to them when they wish to share the changes made by them with the rest of the team.
- It also enables one person to use multiple computers to work on a project, so it is valuable even if you are working by yourself.
- It integrates the work that is done simultaneously by different members of the team. In some rare case, when conflicting edits are made by two people to the same line of a file, then human assistance is requested by the version control system in deciding what should be done.
- Version control provides access to the historical versions of a project. This is insurance against computer crashes or data loss. If any mistake is made, you can easily roll back to a previous version. It is also possible to undo specific edits that too without losing the work done in the meanwhile. It can be easily known when, why, and by whom any part of a file was edited.

History of code versioning system

A brief history of version control systems We will take a look at a brief history of Version Control Systems (VCS). By looking at the background of VCS, we can better understand how much VCS has changed over the years, the motivation of development behind each VCS and the evolution from centralized systems to distributed systems. Version Control Systems have come a long way since it was first introduced. One of the earliest VCS was the Source Code Control System (SCCS), developed by Marc J. Rochkind in 1972. It only works locally and only on individual files. Although merging was not supported, its file storage technique was used by later VCS and is key to advanced merging and versioning techniques. It was later written for Unix and it became the de-facto VCS for Unix until Revision Control System (RCS) merged. RCS, developed in 1982 by Walter F. Tichy was an improved version of SCCS; it supported binary files and has better storage performance. Like SCCS, it was itself replaced by Concurrent Versions System (CVS) in 1986. CVS was written by Dick Grune as a set of RCS scripts to operate on multiple files. It was rereleased in 1990 as a client-server VCS, making it the first Centralized Version Control System (CVCS). However, it had many flaws, most notably, the lack of an atomic commit. To tackle this, Subversion (SVN) was written and sponsored by CollabNet in 2000. Its main goal, "CVS done right" was central to its purpose, and as it introduced many improvements over CVS, which made it the de-facto CVCS used nowadays.

Distributed Version Control Systems (DVCS) emerged in the 1990s with the commercial release of Sun Workshop TeamWare by Sun MicroSystems. It was the first DVCS and it was used internally by Sun for the development of the Solaris and Java platforms. It incorporated a lot of advanced features not found in earlier VCS, such as the local repository. DVCS took off and in 1998, BitKeeper was released by BitMover Inc, whose CEO Larry McVoy previously designed TeamWare. BitKeeper was built upon many TeamWare concepts. It was most notably used as the SCM tool of the Linux Kernel project from 2002 to 2005. It was used by project founder Linus Torvalds himself, despite it being a proprietary product. It withdrew from the Linux Kernel project controversially. To fill up the void left behind after BitKeeper's exit, two new DVCS was initiated around the same time, Git and Mercurial. Linus Torvalds wrote Git in 2005 within 2 months. His motivation for Git was to have it support a BitKeeper workflow instead of the CVS workflow which he disliked, while having high performance and strong safety measures against corruption of source code. It was used in the Linux kernel project after its release. Likewise, Matt Mackall developed Mercurial with similar aims and motivation as that of Git. Written in Python, it

focused on cross-platform interoperability and user-friendliness. Till this day, many developers are still continuing to adopt DVCS over CVCS. This may be attributed to several reasons.

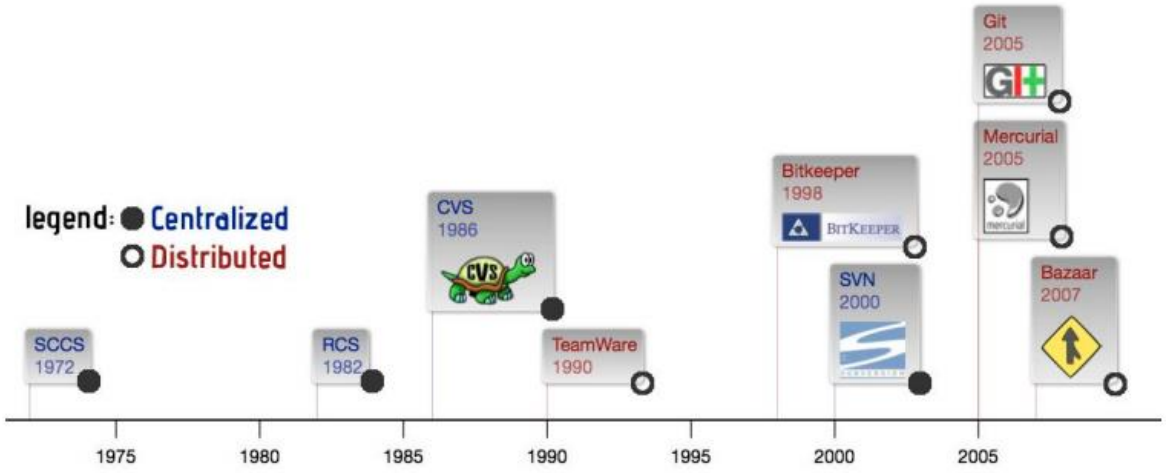


Figure 1: Timeline of the various VCS

2 Common Version Control Systems Terminologies

Entities

Repository (Repo)	The database where the files and historical data are stored, including the author of the changes and the summary of each change. Commonly called **repo** for short.
Working Copy	The local directory of your files.
Trunk	The primary location for code in the repo. Remember that a Version Control System allows for branching, the trunk is the one that usually contains the stable code that everyone is working on. Also called **master**, **main** or **default**.
Branch	A secondary location of the code in the repo. A repo can have multiple branches but usually only one trunk.
Revision	A revision is the set of changes whenever a check in is performed. Each revision is given a number. See Figure 2.

Actions

Check In (Commit)	Uploads a changed file or a set of changed files to the repository. Commonly known as **commit**.
Check Out	Downloads a file or a set of files from the repository (for the first time).
Add	Tells the Version Control System to track a file, a set of files or a directory. These tracked files do not go into the repository until the next check in.
Update	Synchronize the files in your working copy with the latest files from the repository. This is normally performed after a **check out**.
Revert	Discards all changes in the working copy and use a specified revision from the repository. See Figure 3.
Branch	The act of creating a branch. See Figure 4.
Merge	Apply the changes from one or more file(s) to another. For example, you can merge code from a branch into the trunk. See Figure 5.
Resolve	A **conflict** may occur when multiple changes to a file contradict one another. When that happens, the process of fixing these conflicts and checking in the fixed file is called **resolve**.
Tag	Label a revision for easy reference. See Figure 6.

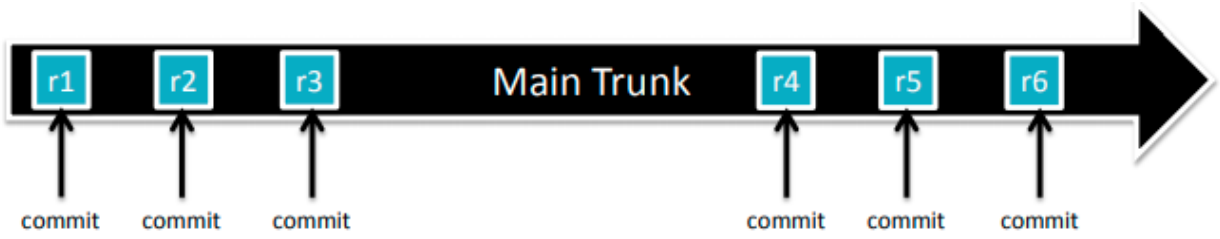


Figure 2: New revisions are created whenever a commit is made

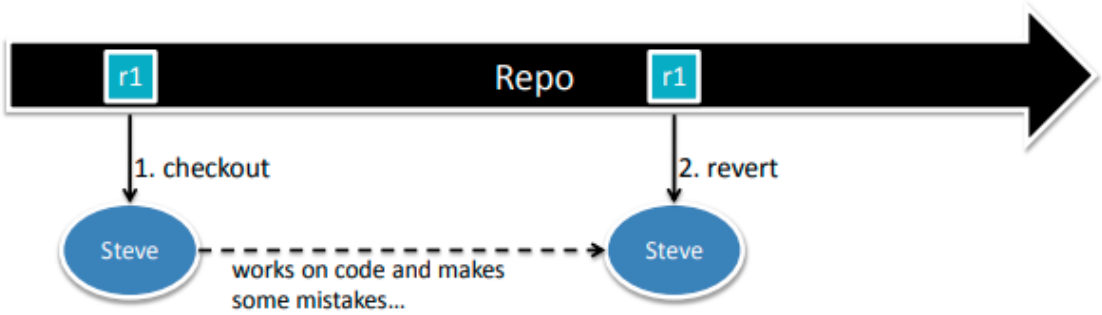


Figure 3: Reverting after making mistakes

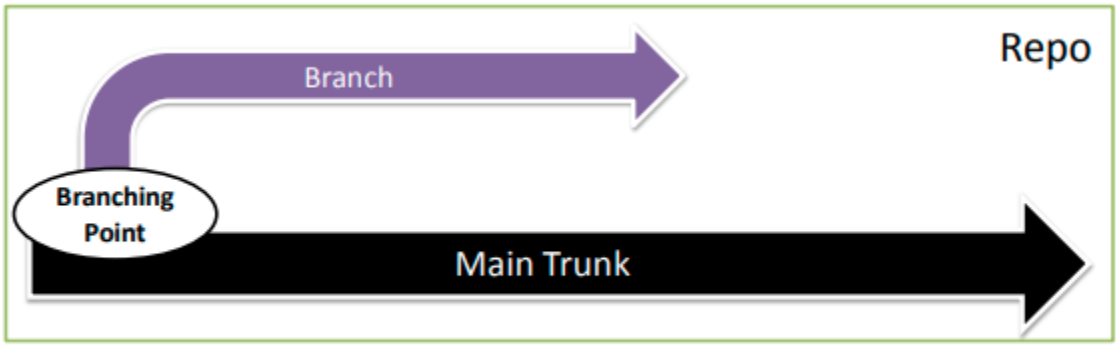


Figure 4: Branching

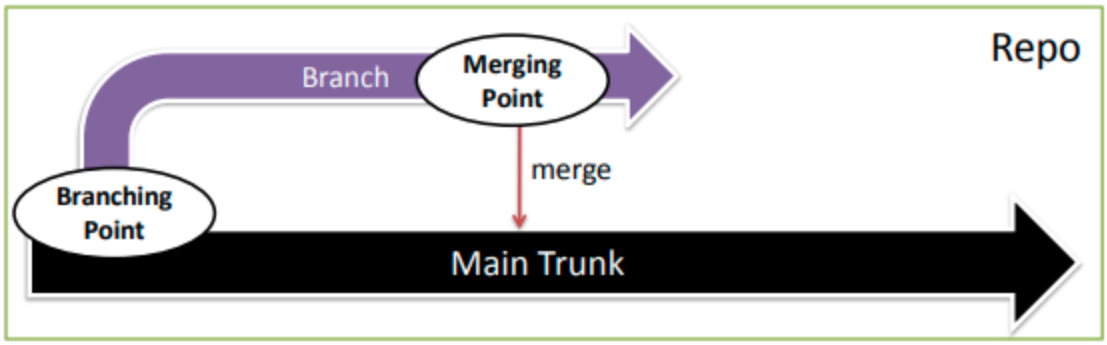


Figure 5: Merging after branching

Figure 6: Tagging Revision 1337 as 1.0

Local Version Control Systems

Many people's version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they're clever). This approach is very common because it is so simple, but it is also incredibly error prone. It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to.

To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.

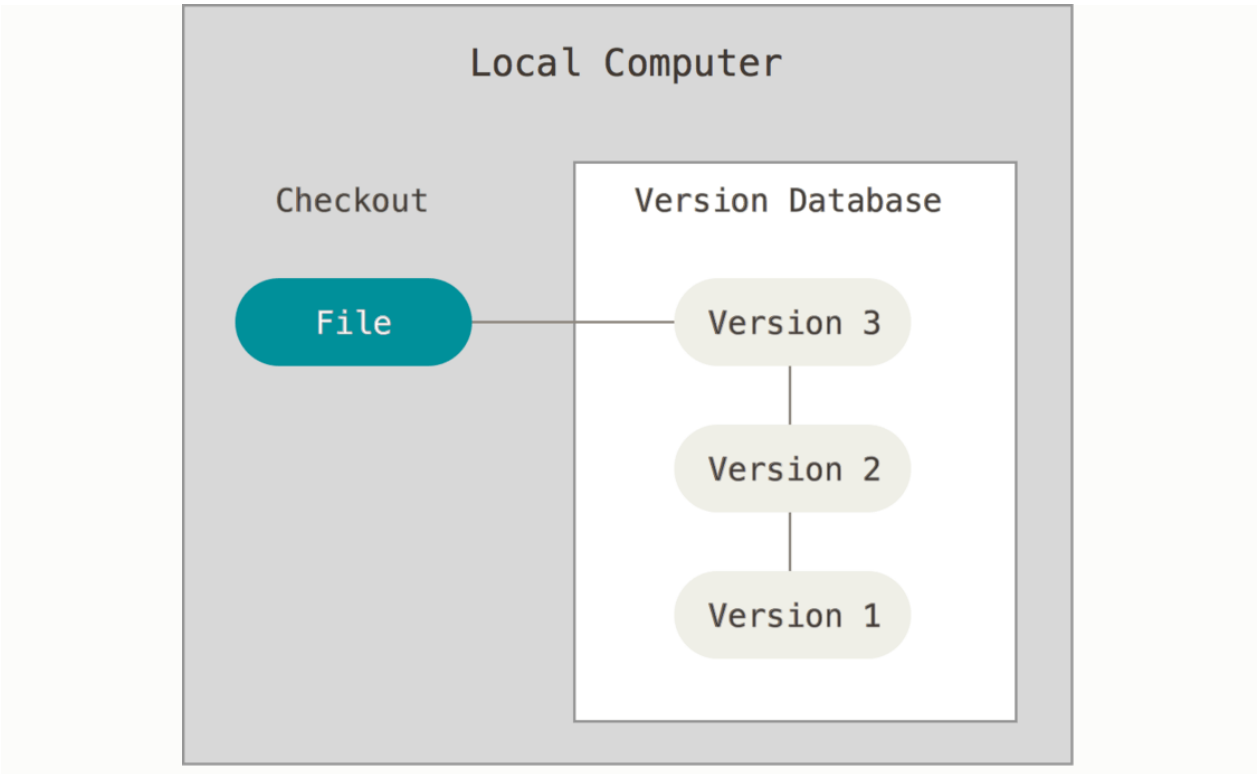


Figure 1. Local version control

One of the most popular VCS tools was a system called RCS, which is still distributed with many computers today. RCS works by keeping patch sets (that is, the differences between files) in a special format on disk; it can then re-create what any file looked like at any point in time by adding up all the patches.

Centralized Version Control Systems

The next major issue that people encounter is that they need to collaborate with developers on other systems. To deal with this problem, Centralized Version Control Systems (CVCSs) were developed. These systems (such as CVS, Subversion, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place. For many years, this has been the standard for version control.

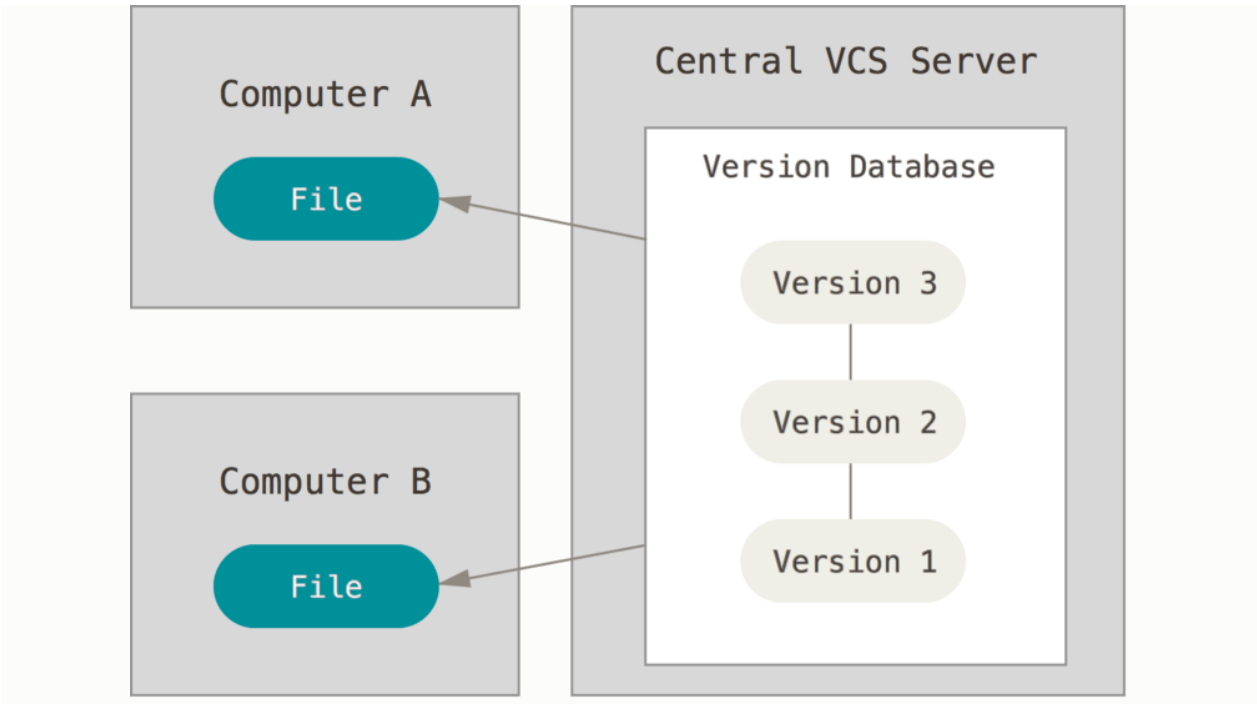


Figure 2. Centralized version control

This setup offers many advantages, especially over local VCSs. For example, everyone knows to a certain degree what everyone else on the project is doing. Administrators have fine-grained control over who can do what, and it's far easier to administer a CVCS than it is to deal with local databases on every client.

However, this setup also has some serious downsides. The most obvious is the single point of failure that the centralized server represents. If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they're working on. If the hard disk the central database is on becomes corrupted, and proper backups haven't been kept, you lose absolutely everything — the entire history of the project except whatever single snapshots people happen to have on their local machines. Local VCS systems suffer from this same problem — whenever you have the entire history of the project in a single place, you risk losing everything.

Distributed Version Control Systems

This is where Distributed Version Control Systems (DVCSs) step in. In a DVCS (such as Git, Mercurial, Bazaar or Darcs), clients don't just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history. Thus, if any server dies, and these systems were collaborating via

that server, any of the client repositories can be copied back up to the server to restore it. Every clone is really a full backup of all the data.

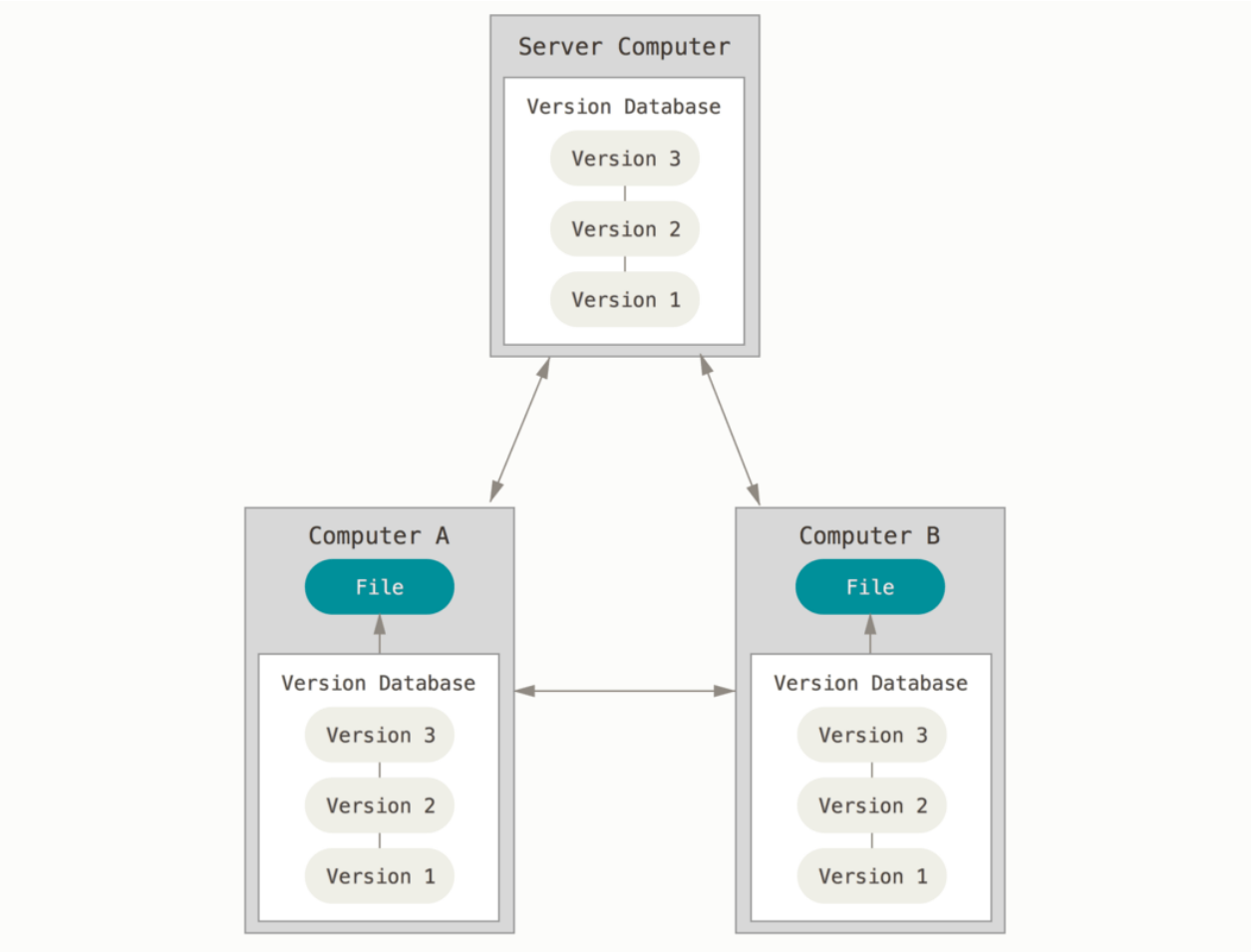


Figure 3. Distributed version control

Furthermore, many of these systems deal pretty well with having several remote repositories they can work with, so you can collaborate with different groups of people in different ways simultaneously within the same project. This allows you to set up several types of workflows that aren't possible in centralized systems, such as hierarchical models.

Benefits of version control

Version control systems are a category of software tools that help a software team manage changes to source code over time. Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

For almost all software projects, the source code is like the crown jewels - a precious asset whose value must be protected. For most software teams, the source code is a repository of the invaluable knowledge and understanding about the problem domain that the developers have collected and refined through careful effort. Version control protects source code from both catastrophe and the casual degradation of human error and unintended consequences.

Software developers working in teams are continually writing new source code and changing existing source code. The code for a project, app or software component is typically organized in a folder structure or "file tree". One developer on the team may be working on a new feature while another developer fixes an unrelated bug by changing code, each developer may make their changes in several parts of the file tree.

Version control helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting. Changes made in one part of the software can be incompatible with those made by another developer working at the same time. This problem should be discovered and solved in an orderly manner without blocking the work of the rest of the team. Further, in all software development, any change can introduce new bugs on its own and new software can't be trusted until it's tested. So testing and development proceed together until a new version is ready.

Good version control software supports a developer's preferred workflow without imposing one particular way of working. Ideally it also works on any platform, rather than dictate what operating system or tool chain developers must use. Great version control systems facilitate a smooth and continuous flow of changes to the code rather than the frustrating and clumsy mechanism of file locking - giving the green light to one developer at the expense of blocking the progress of others.

Software teams that do not use any form of version control often run into problems like not knowing which changes that have been made are available to users or the creation of incompatible changes between two unrelated pieces of work that must then be painstakingly untangled and reworked. If you're a developer who has never used version control you may have added versions to your files, perhaps with suffixes like "final" or "latest" and then had to later deal with a new final version. Perhaps you've commented out code blocks because you want to disable certain functionality without deleting the code, fearing that there may be a use for it later. Version control is a way out of these problems.

Version control software is an essential part of the every-day of the modern software team's professional practices. Individual software developers who are accustomed to working with a capable version control system in their teams typically recognize the incredible value version control also gives them even on small solo projects. Once accustomed to the powerful benefits of version control systems, many developers wouldn't consider working without it even for non-software projects.

Developing software without using version control is risky, like not having backups. Version control can also enable developers to move faster and it allows software teams to preserve efficiency and agility as the team scales to include more developers.

Version Control Systems (VCS) have seen great improvements over the past few decades and some are better than others. VCS are sometimes known as SCM (Source Code Management) tools or RCS (Revision Control System). One of the most popular VCS tools in use today is called Git. Git is a Distributed VCS, a category known as DVCS, more on that later. Like many of the most popular VCS systems available today, Git is free and open source. Regardless of what they are called, or which system is used, the primary benefits you should expect from version control are as follows.

- A complete long-term change history of every file. This means every change made by many individuals over the years. Changes include the creation and deletion of files as well as edits to their contents. Different VCS tools differ on how well they handle renaming and moving of files. This history should also include the author, date and written notes on the purpose of each change. Having the complete history enables going back to previous versions to help in root cause analysis for bugs and it is crucial when needing to fix problems in older versions of software. If the software is being actively worked on, almost everything can be considered an "older version" of the software.
- Branching and merging. Having team members work concurrently is a no-brainer, but even individuals working on their own can benefit from the ability to work on independent streams of changes. Creating a "branch" in VCS tools keeps multiple streams of work independent from each other while also providing the facility to merge that work back together, enabling developers to verify that the changes on each branch do not conflict. Many software teams adopt a practice of branching for each feature or perhaps branching for each release, or both. There are many different workflows that teams can choose from when they decide how to make use of branching and merging facilities in VCS.

- Traceability. Being able to trace each change made to the software and connect it to project management and bug tracking software such as Jira, and being able to annotate each change with a message describing the purpose and intent of the change can help not only with root cause analysis and other forensics. Having the annotated history of the code at your fingertips when you are reading the code, trying to understand what it is doing and why it is so designed can enable developers to make correct and harmonious changes that are in accord with the intended long-term design of the system. This can be especially important for working effectively with legacy code and is crucial in enabling developers to estimate future work with any accuracy.

While it is possible to develop software without using any version control, doing so subjects the project to a huge risk that no professional team would be advised to accept. So the question is not whether to use version control but which version control system to use.

Why is version control important?

If you are reading this, it's possible that you are updating documents that look something like this: index-v12-old2.html. Let's get away from this and on to something that will not only allow you to control your source code and files, but become more productive as a team. If this sounds familiar:

- Communicated with your team via email about updates.
- Made updates directly on your production server.
- Accidentally overwrote some files, which can never be retrieved again.

You can now look forward to this instead:

- File names and directory structures that are consistent for all team members.
- Making changes with confidence, and even reverting when needed.
- Relying on source control as the communication medium for your team.
- Easily deploying different versions of your code to staging or production servers.
- Understanding who made a change and when it happened.

The basic concepts

Tracking changes

A version control system is mostly based around one concept, tracking changes that happen within directories or files. Depending on the version control system, this could vary from knowing a file changed to knowing specific characters or bytes in a file have changed.

In most cases, you specify a directory or set of files that should have their changes tracked by version control. This can happen by checking out (or cloning) a repository from a host, or by telling the software which of your files you wish to have under version control.

The set of files or directories that are under version control are more commonly called a repository.

As you make changes, it will track each change behind the scenes. The process will be transparent to you until you are ready to commit those changes.

Committing

As you work with your files that are under version control, each change is tracked automatically. This can include modifying a file, deleting a directory, adding a new file, moving files or just about anything else that might alter the state of the file. Instead of recording each change individually, the version control system will wait for you to submit your changes as a single collection of actions. In version control, this collection of actions is known as a commit.

Revisions and Changesets

When a commit is made, the changes are recorded as a changeset and given a unique revision. This revision could be in the form of an incremented number (1, 2, 3) or a unique hash (like 846eee7d92415cfd3f8a936d9ba5c3ad345831e5) depending on the system. By knowing the revision of a changeset it makes it easy to view or reference it later. A changset will include a reference to the person who made the commit, when the change was made, the files or directories affected, a comment and even the changes that happened within the files (lines of code).

When it comes to collaboration, viewing past revisions and changesets is a valuable tool to see how your project has evolved and for reviewing teammates' code. Each version control system has a formatted way to view a complete history (or log) of each revision and changeset in the repository.

Getting updates

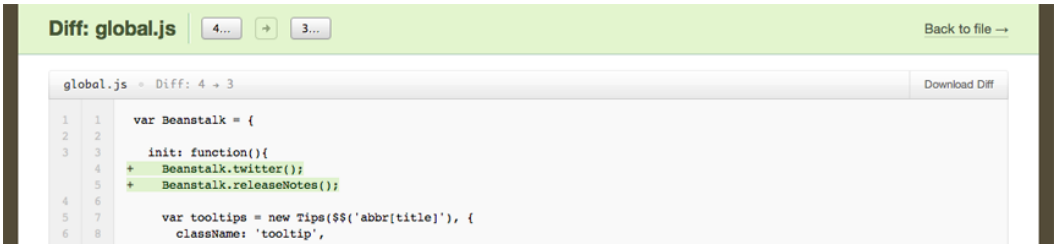
As members of your team commit changes, it is important that you have the latest version. Having the latest version reduces the chance of a conflict. Getting the latest changes from a repository is as simple as doing a pull or update from another computer (usually a hosted or centralized server). When an update or pull is requested, only the changes since your last request are downloaded.

Conflicts

What if the latest update or commit results in a conflict? That is, what if your changes are so similar to the changes that another team member made that the version control system can't determine which is the correct and authoritative change? In most cases, the version control system will provide a way to view the difference between the conflicting versions, allowing you to make a choice. You can either edit the files manually to merge the options, or allow one revision to win over the other. You may want to collaborate with the person who made the other commit to make sure you're not undoing their important work!

Diffing (or, viewing the differences)

Since each commit is recorded as a change to a file or set of files and directories, it is sometimes useful to view what changed between revisions. For instance, if a recent deployment of your web site is broken and you narrowed down the cause to a particular file, the best action to take is to see what recently changed in that file. By viewing a diff, you can compare two files or even a set of files to see what lines of code changed, when it changed and who changed it. Most version control tools let you compare two sequential revisions, but also two revisions from anywhere in the history.



Branching and merging

There are some cases when you want to experiment or commit changes to the repo that could break things elsewhere in your code (like working on a new feature). Instead of committing this code directly to the main set of files (usually referred to as trunk or master), you can create something called a branch. A branch allows you to create a copy (or snapshot) of the repository that you can modify in parallel without altering the main set. You can continue to commit new changes to the branch as you work, while others commit to the trunk or master without the changes affecting each other.

Once you're comfortable with the experimental code, you will want to make it part of the trunk or master again. This is where merging comes in. Since the version control system has recorded every

change so far, it knows how each file has been altered. By merging the branch with the trunk or master (or even another branch), your version control tool will attempt to seamlessly merge each file and line of code automatically. Once a branch is merged it then updates the trunk or master with the latest files.

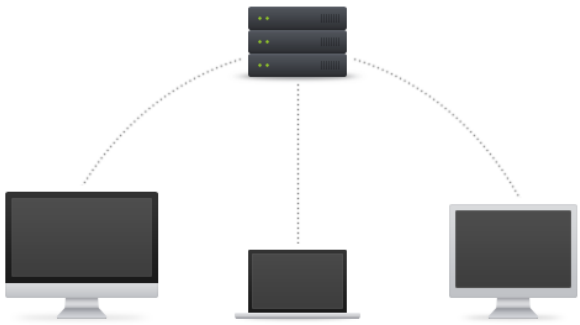
For example, let's say you want to experiment with a new layout for a web site. This may require heavy changes in many files, could break existing code and it may not turn out as expected. It could also take a long time, so you want to continue committing the changes. Instead of committing to the trunk or master set of files, you create a branch. From this point forward, any changes made in the new branch will not affect others in the trunk or master. Days or weeks may go by allowing you to commit changes, test and refine. When the new layout is working properly and you are comfortable with the result you are probably ready to make it a permanent part of the site. This is the point where you will merge the branch with the trunk or master. Once the merge is complete, it will combine the changes from the branch with the most recent version of the trunk or master.

In some cases the version control system might not be able to figure out which change to apply between two revisions when doing a merge. If this happens a conflict will arise. A conflict in this scenario is the same as the conflict mentioned above and requires manual intervention to decide which files or lines of code should remain.

Types of Version Control Systems

The three most popular version control systems are broken down into two main categories, centralized and decentralized (also known as distributed).

Centralized Version Control



At the time of this writing, the most popular version control system is known as Subversion, which is considered a centralized version control system. The main concept of a centralized system is that it works in a client and server relationship. The repository is located in one place and provides access to many clients. It's very similar to FTP in where you have an FTP client which connects to an FTP server. All changes, users, commits and information must be sent and received from this central repository.

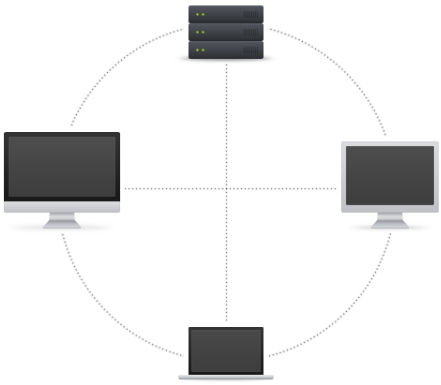
The primary benefits of Subversion are:

- It is easy to understand.
- You have more control over users and access (since it is served from one place).
- More GUI & IDE clients (Subversion has been around longer).
- Simple to get started.

At the same time, Subversion has some drawbacks:

- Dependent on access to the server.
- Hard to manage a server and backups (well, not with Beanstalk of course!)
- It can be slower because every command connects to the server.
- Branching and merging tools are difficult to use.

Distributed Version Control



Distributed systems are a newer option. In distributed version control, each user has their own copy of the entire repository, not just the files but the history as well. Think of it as a network of individual repositories. In many cases, even though the model is distributed, services like Beanstalk are used for simplifying the technical challenges of sharing changes. The two most popular distributed version control systems are Git and Mercurial.

The primary benefits of Git and Mercurial are:

- More powerful and detailed change tracking, which means less conflicts.
- No server necessary – all actions except sharing repositories are local (commit offline).
- Branching and merging is more reliable, and therefore used more often.
- It's fast.

At the same time, Git and Mercurial do have some drawbacks:

- The distributed model is harder to understand.
- It's new, so not as many GUI clients.
- The revisions are not incremental numbers, which make them harder to reference.
- It can be easier to make mistakes until you are familiar with the model.

Which one is right for you? This really depends on how you work. We usually recommend Subversion if you are just getting started, are not comfortable with the command line or do not plan to do a lot of branching and merging. Otherwise, we encourage you to try Git or Mercurial and see if you like it. With Beanstalk, you're free to go back and forth as needed without switching hosts.

Version Control Software Tools

#1) Git

Git is one of the best version control tools that is available in the present market.

Features

- Provides strong support for non-linear development.
- Distributed repository model.
- Compatible with existing systems and protocols like HTTP, FTP, ssh.
- Capable of efficiently handling small to large sized projects.
- Cryptographic authentication of history.
- Pluggable merge strategies.
- Toolkit-based design.
- Periodic explicit object packing.
- Garbage accumulates until collected.

Pros

- Super-fast and efficient performance.
- Cross-platform
- Code changes can be very easily and clearly tracked.
- Easily maintainable and robust.
- Offers an amazing command line utility known as git bash.
- Also offers GIT GUI where you can very quickly re-scan, state change, sign off, commit & push the code quickly with just a few clicks.

Cons

- Complex and bigger history log become difficult to understand.
- Does not support keyword expansion and timestamp preservation.

Open Source: Yes

Cost: Free

Web : https://git-scm.com/

#2) CVS

It is yet another most popular revision control system. CVS has been the tool of choice for a long time.

Features

- Client-server repository model.
- Multiple developers might work on the same project parallelly.
- CVS client will keep the working copy of the file up-to-date and requires manual intervention only when an edit conflict occurs
- Keeps a historical snapshot of the project.
- Anonymous read access.
- 'Update' command to keep local copies up to date.
- Can uphold different branches of a project.
- Excludes symbolic links to avoid a security risk.
- Uses delta compression technique for efficient storage.

Pros

- Excellent cross-platform support.
- Robust and fully-featured command-line client permits powerful scripting
- Helpful support from vast CVS community
- allows good web browsing of the source code repository
- It's a very old, well known & understood tool.
- Suits the collaborative nature of the open-source world splendidly.

Cons

- No integrity checking for source code repository.
- Does not support atomic check-outs and commits.
- Poor support for distributed source control.
- Does not support signed revisions and merge tracking.

Open Source: Yes

Cost: Free

Web : http://savannah.nongnu.org/projects/cvs

#3) SVN

Apache Subversion, abbreviated as SVN aims at to be a best-matched successor to the widely used CVS tool that we just discussed above.

Features

- Client-server repository model. However, SVK permits SVN to have distributed branches.
- Directories are versioned.
- Copying, deleting, moving and renaming operations are also versioned.
- Supports atomic commits.
- Versioned symbolic links.
- Free-form versioned metadata.
- Space efficient binary diff storage.
- Branching is not dependent upon the file size and this is a cheap operation.
- Other features – merge tracking, full MIME support, path-based authorization, file locking, standalone server operation.

Pros

- Has a benefit of good GUI tools like TortoiseSVN.
- Supports empty directories.
- Have better windows support as compared to Git.
- Easy to set up and administer.
- Integrates well with Windows, leading IDE and Agile tools.

Cons

- Does not store the modification time of files.
- Does not deal well with filename normalization.
- Does not support signed revisions.

Open Source – Yes

Cost: Free

Official Website. https://subversion.apache.org/

#4) Mercurial

Mercurial is a distributed revision-control tool which is written in python and intended for software developers. The operating systems that it supports are Unix-like, Windows and macOS.

Features

- High performance and scalability.
- Advanced branching and merging capabilities.
- Fully distributed collaborative development.
- Decentralized
- Handles both plain text and binary files robustly.
- Possesses an integrated web interface.

Pros

- Fast and powerful
- Easy to learn
- Lightweight and portable.
- Conceptually simple

Cons

- All the add-ons must be written in Python.
- Partial checkouts are not allowed.
- Quite problematic when used with additional extensions..

Open Source: Yes

Cost: Free

Official Website.: https://www.mercurial-scm.org/

5) Monotone

Monotone, written in C++, is a tool for distributed revision control. The OS that it supports includes Unix, Linux, BSD, Mac OS X, and Windows.

Features

- Provides good support for internationalization and localization.
- Focuses on integrity over performance.
- Intended for distributed operations.
- Employs cryptographic primitives to track file revisions and authentications.
- Can import CVS projects.
- Uses a very efficient and robust custom protocol called netsync.

Pros

- Requires very low maintenance
- Good documentation
- Easy to learn
- Portable design
- Works great with branching and merging
- Stable GUI

Cons

- Performance issues observed for some operations, most visible was an initial pull.
- Can't commit or checkout from behind the proxy (this is because of a non-HTTP protocol).

Open Source: Yes

Cost: Free

Web : https://www.monotone.ca/

6) Bazaar

Bazaar is a version control tool that is based on a distributed and client-server repository model. It provides cross-platform OS support and is written in Python 2, Pyrex and C.

Features

- It has commands similar to SVN or CVS.
- It allows you to be working with or without a central server.
- Provides free hosting services through the websites Launchpad and Sourceforge.
- Supports file names from the entire Unicode set.

Pros

- Directories tracking is supported very well in Bazaar (this feature is not there in tools like Git, Mercurial)
- Its plugin system is fairly easy to use.
- High storage efficiency and speed.

Cons

- Does not support partial checkout/clone.
- Does not provide timestamp preservation.

Open Source: Yes

Cost: Free

Web : http://bazaar.canonical.com/en/

7) TFS

TFS, an acronym for team foundation server is a version control product by Microsoft. It is based on client-server, distributed repository model and has a proprietary license. It provides Windows, cross-platform OS support through Visual Studio Team Services (VSTS).

Features

- Provides entire application lifecycle support including source code management, project management, reporting, automated builds, testing, release management and requirement management.

- Empowers DevOps capabilities.
- Can be used as a backend for several IDEs.
- Available in two different forms (on-premises and online (known as VSTS)).

Pros

- Easy administration. Familiar interfaces and tight integration with other Microsoft products.
- Allows continuous integration, the team builds and unit test integration.
- Great support for branching and merging operations.
- Custom check-in policies to aid in implementing a steady & stable codebase in your source control.

Cons

- Frequent merge conflicts.
- Connection to the central repository is always required.
- Quite slow in performing a pull, check-in, and branching operations.

Open Source: No

Cost: Free of cost for up to 5 users in the VSTS or for open source projects via codeplex.com; else paid and licensed through MSDN subscription or direct buy.

The server license can be bought for around $500 and the client licenses are also nearly the same.

Web : https://azure.microsoft.com/en-us/services/devops/server/

8) VSTS

VSTS (Visual Studio Team Services) is a distributed, client-server repository model based version control tool provided by Microsoft. It follows the Merge or Lock concurrency model and provides cross-platform support.

Features

- Programming Language: C# & C++
- Changeset storage method.

- File and Tree scope of change.
- Network protocols supported: SOAP over HTTP or HTTPS, Ssh.
- VSTS offers elastic build capabilities thru build hosting in Microsoft Azure.
- DevOps enables

Pros

- All the features that are present in TFS are available in VSTS in the cloud.
- Supports almost any programming language.
- Instinctive User Interface
- Upgrades get automatically installed.
- Git access

Cons

- Signed revisions are not allowed.
- The “work” section is not very well optimized for large teams.

Open Source: No, it is a proprietary software. But, free trial version is available.

Cost: Free for up to 5 users. $30/mo for 10 users. Also offers a lot of free and paid extensions.

Website : https://azure.microsoft.com/en-us/services/devops/

9) Perforce Helix Core

Helix Core is a Client-server and distributed revision control tool developed by Perforce Software Inc. It supports Unix-like, Windows and OS X platforms. This tool is mainly for large-scale development environments.

Features:

- Maintains a central database and a master repository for the file versions.
- Supports all file types and sizes.
- File-level asset management.
- Maintains a single source of truth.
- Flexible branching
- DevOps ready

Pros

- Git accessible
- Lightning fast
- Massively scalable

- Easy to track the change list.
- Diff tools make it very easy to identify code changes.
- Works well with the visual studio through the plugin.

Cons

- Managing multiple workspaces is quite difficult.
- Perforce Streams makes managing multiple workspaces quite simple. Users are only seeing data that is relevant, and it adds traceability.
- Rollbacking changes are troublesome if its split across multiple change-lists.
- We do offer the ability to undo a submitted changelist (in P4V) where a user can just right-click a given changelist and perform that action.

Open Source: No, it's proprietary software. But, a free trial version for 30 days is available.

Cost: Helix Core is now always free for up to 5 users and 20 workspaces.

Website.: https://www.perforce.com/products/helix-core

10) IBM Rational ClearCase

ClearCase by IBM Rational is a client-server repository model based on software configuration management tool. It supports a lot of Operating systems including AIX, Windows, z/OS (limited client), HP-UX, Linux, Linux on z Systems, Solaris.

Features:

- Supports two models i.e UCM and base ClearCase.
- UCM stands for Unified Change Management and offers an out-of-the-box model.
- Base ClearCase offers basic infrastructure.
- Capable of handling huge binary files, a large number of files, and big repository sizes.
- Allows branching, labeling, and versioning of directories.

Pros

- Simple UI
- Integrates with Visual Studio.

- Handles parallel development.
- ClearCase Views are very convenient as they allow to switch between projects and configurations as opposed to local workstation model of the other version control tools.

Cons

- Slow recursive operations.
- Evil Twin problem – Here, two files with the same name get added to the location instead of versioning the same file.
- No advanced API

Open Source: No, it is a proprietary tool. But, free trial version is available.

Cost: $4600 for each floating license (detained automatically for a 30-minutes minimum for each user, can be surrendered manually)

Website. : https://www.ibm.com/in-en/marketplace/rational-clearcase

11) Revision Control System

Revision Control system (RCS), developed by Thien-Thi Nguyen works on the local repository model and supports Unix-like platforms. RCS is a very old tool and was first released in 1982. It is an early version of VCS(Version Control System).

Features:

- Was originally intended for programs, but, is also helpful for text documents or config files that often get revised.
- RCS can be considered as a set of Unix Commands that permits various users to build and maintain program code or documents.
- Allows revision of documents, committing changes and merging docs together.
- Store revisions in a tree structure.

Pros

- Simple architecture
- Easy to work with
- It has local repository model, so the saving of revisions is independent of the central repository.

Cons

- Less security, version history is editable.
- At a time, only one user can work on the same file.

Open Source: Yes

Cost: Free

Website. : https://www.gnu.org/software/rcs/

12) Visual SourceSafe(VSS)

VSS by Microsoft is a Shared folder repository model based revision control tool. It supports Windows OS only.

It is intended for small software development projects.

Features

- Creates a virtual library of computer files.
- Capable of handling any file type in its database.

Pros

- Fairly easy to use interface.
- It lets a single user system to be assembled with fewer configurations when compared to any other SCM systems.
- Easy backup process.

Cons:

- Lacks many important features of a multi-user environment.
- Database corruption is one of the serious problems noted with this tool.

Cost: Paid. Nearly $500 for each license or single license which is comprised of every MSDN subscription.

Website. : https://www.microsoft.com/en-in/download/details.aspx?id=291

13) CA Harvest Software Change Manager

This is a revision control tool provided by CA technologies. It supports many platforms including Microsoft Windows, Z-Linux, Linux, AIX, Solaris, Mac OS X.

Features

- Changes are made to a “change package”. Harvest supports both version control as well as change management.
- Has a pre-defined lifecycle from Test to Production stages.
- Fully customizable project environments. Project means ‘entire control framework’ in Harvest.

Open Source: No, this tool comes with Proprietary EULA License. However, a free trial is available.

Pros

- Helps very well in tracking the application flow from dev to prod environments. The biggest asset of this tool is this lifecycle feature.
- Deployment in a safe manner.
- Stable and scalable.

Cons

- Could be more user-friendly.
- Merging feature could be improved.
- Handling Polar Requests For Code Reviews Is challenging.

Cost: Not disclosed by the vendor.

Website. : https://www.softwaretestinghelp.com/version-control-software/

14) PVCS

PVCS (an acronym for Polytron Version Control System), developed by Serena Software is a client-server repository model based version control tool. It supports Windows and Unix-like platforms. It provides version control of source code files. It is mainly intended for small development teams.

Features

- Follows locking approach to concurrency control.
- No built-in merge opera.tor but has a separate merge command.
- Supports multi-user environment.

Pros

- Easy to learn and use
- Manages the file versions regardless of the platforms.
- Gets easily integrated with Microsoft Visual Studio .NET and Eclipse IDEs.

Cons

- Its GUI has some quirks.

Open Source: No, it is a proprietary software.

Cost: Not disclosed by the vendor.

Website. : https://www.microfocus.com/en-us/products/pvcs/overview

15) darcs

darcs (Darcs Advanced Revision Control System), developed by The Darcs team is a distributed version control tool that follows merge concurrency model. This tool is written in Haskell and supports Unix, Linux, BSD, ApplemacOS, MS Windows platforms.

Features

- Capable of selecting which changes to accept from other repositories.
- Communicates with local and remote repositories through SSH, HTTP, email or unusually interactive interface.
- Works on the concept of linearly ordered patches.

Pros

- Has fewer and more interactive commands when compared to other tools like git and SVN.
- Offers send system for direct mailing.

Cons

- Performance issues related to merging operations.
- Installation takes a long time.

Open Source: Yes

Cost: This is a free tool.

Website. : http://darcs.net/