How do programmers work together on a project?

Asked 14 years, 6 months ago Modified 9 years, 5 months ago

Viewed 25k times Part of CI/CD Collective

89

I've always programmed alone, I'm still a student so I never programmed with anyone else, I haven't even used a version control system before.

I'm working on a project now that requires knowledge of how programmers work together on a piece of software in a company.

How is the software compiled? Is it from the version control system? Is it by individual programmers? Is it periodic? Is it when someone decides to build or something? Are there any tests that are done to make sure it "works"?

Anything will do.

CI / CD

unit-testing version-control compilation continuous-integration

Share

Improve this question

edited Jul 20, 2013 at 21:24

Follow

Cjxcz Odjcayrwl
22.8k ● 42 ● 149 ● 228

asked Jun 8, 2010 at 18:33

Leo Jweda
2,487 ● 3 ● 23 ● 34

25 I removed the "not-programming-related" tag because how people program as a team is definitely programming related. – Brendan Long Jun 8, 2010 at 18:47

@Brendan Long: Thank you for the retag. – Leo Jweda Jun 9, 2010 at 5:06

13 Answers

Sorted by: Highest score (default)

66

Actually, there are as many variations on these processes as many companies there are. Meaning: every company has a little bit different conventions than others, but there are some common best practices that are generally used in most places.

Best practices that are always useful

- **All** the source code of the project and anything that is required to build it is under *version control* (also called source control). **Anyone** should be able to build the entire project with one click. Furthermore, unnecessary files (object files or compiled binaries) should **not** be added to the repository, as they can be regenerated quite easily and would just waste space in the repo.

- Every developer should **update** and **commit** to the version control a few times per day. Mostly when they have finished the task they are working on and tested it enough so they know that it doesn't contain trivial bugs.
- Again: **anyone** should be able to build the project with a single click. This is important and makes it easy to test for everyone. Big advantage if non-programmers (eg. the boss) are able to do so, too. (It makes them feel to be able to see what the team is working on exactly.)
- Every developer **should test** the new feature or bug fix they are adding **before** they commit those to the repository.
- Set up a server that regulary (in predetermined intervals) updates itself from the repository and tries to build **everything** in the **entire project**. If it fails, it sends e-mails to the team along with the latest commits to the version control (since which commit did it fail to build) to help debug the issue.
  This practice is called *continuous integration* and the builds are also called *nightly builds*.
  (This doesn't imply that developers should not build and test the code on their own machines. As mentioned above, they should do that.)
- Obviously, **everyone** should be familiar with the basic design/architecture of the project, so if

something is needed, different members of the team doesn't have to reinvent the wheel. Writing reusable code is a good thing.

- Some sort of **communication** is needed between the team members. Everyone should be aware of what the others are doing, at least a little. The more, the better. This is why the **daily standup** is useful in SCRUM teams.
- **Unit testing** is a very good practice that makes testing the basic functionality of your code automatically.
- A **bug tracking software** (sometimes called *time tracking software*) is a very good means of keeping track what bugs are there and what tasks the different team members have. It is also good for testing: the alpha/beta testers of your project could communicate with the development team this way.

These simple things ensure that the project doesn't go out of control and everyone works on the same version of the code. The continuos integration process helps when something goes terribly bad.

It also prevents people from committing stuff that don't build to the main repository.
If you want to include a new feature that would take days to implement and it would block other people from building (and testing) the project, use the **branches** feature of your version control.

If that is not enough, you can set it up to do automated testing, too, if that is possible with the project in question.

Some more thoughts

The above list can be very heavyweight at first glance. I recommend that you follow it on an **as-needed** basis: start with a version control and a bug tracker, then later on set up the continuous integration server, if you need it. (If it's a large project, you're gonna need it very soon.) Start writing unit tests for the most important parts. If it's not enough, then write more of them.

Some useful links:
Continuous integration, Daily builds are your friends, Version control, Unit testing

Examples:

For version control, I tend to use Git for my personal projects nowadays. Subversion is also popular, and for example, VisualSVN is quite easy to set up if you use a Windows server. For client, TortoiseSVN works best for many people. Here is a comparison between Git and SVN.

For bug tracking software, Jira and Bugzilla are very popular. We also used Mantis at a previous workplace.

For continuous integration software, there is Teamcity for one (also, CruiseControl and its .NET counterpart are notable).

Answer to your question "who decides the main design of the project?"

Of course, that would be the lead developer.
In companies, the lead developer is the person who talks to the financial / marketing people of the project, and decides the arcitheсture according to the financial capability of the company, the planned features the requirements from users, and the time that is available.

It is a complex task, and usually more than one people are involved. Sometimes members of the team are also asked to participate or brainstorm about the design of the entire project or specific parts.

Share Improve this answer Follow

edited May 23, 2017 at 11:54
Community Bot
1 ● 1

answered Jun 8, 2010 at 19:00
Venemo
19k ● 13 ● 88 ● 128

4 You should consider Git over Subversion. Subversion is better than CVS, and Git is far more superior than Subversion. Also, subversion has some quircks, even though easy to learn. Especially in the forms of plugins. Even though TortoiseSVN is sweet (when working on Windows systems).
– Shyam Jun 8, 2010 at 19:34

5 @Shyam - Actually, I've heard about Git. It has its advantages, but I wouldn't say "far more superior". Especially when considering that it doesn't have a decent Windows client yet. Still, it is more of a personal preference, so I added it to my answer. – Venemo Jun 8, 2010 at 19:52

@Venemo: Doesn't that make programming boring? Not being able to test your code immediately and having to wait for the build and then see little or no effect of what you wrote? Also, who decides the main design of the project? (language, features, libraries, frameworks, architecture, etc) – Leo Jweda Jun 9, 2010 at 6:13

2 @Laith J: 1. Work in general is boring 2. Being patient is a virtue. 3. Doesn't matter who designs the project, the customer decides. No matter 'how' innovative, fantastic or fabulous idea you have. Deliverables. Work to be alive, don't be alive to work. – Shyam Jun 9, 2010 at 8:06

1 @Shyam - I personally don't know a thing about Git, and I don't judge things that I know little about. No need to turn this into flaming. The differences are well explained here: stackoverflow.com/questions/871/... and I won't change to Git until this simple and day-to-day thing is so hard to achieve: stackoverflow.com/questions/2733873/... . I also like the approach of SVN better and it is much simpler to learn. And I like simple. – Venemo Jun 9, 2010 at 8:29

13

I'm a student as well, who completed a software engineering course recently where the entire semester consisted of a giant group project. Let me just start by saying we could have done with 3 people what it took 12 of us the whole semester to do. Working with people is a tough thing. Communication is key.

Definitely utilize a repository. Each person can remotely access all the code, and add/delete/change anything. But the best part about subversion is that if someone breaks the code, your can revert to an earlier version and assess what went wrong from there. Communication is still key though, know what your teammates are doing so that there are no conflicts. Don't sit on your code either, make quick, meaningful commits to the repository to be the most effective.

**I'd also recommend a bug tracker, such as Redmine. You can set up accounts for everyone, and assign people tasks with different priorities, and also track and see if people have taken care of certain problems, or if more have come up.

And, as has been said before, unit testing will help greatly. Best of luck! Hope this helped :-)

Share Improve this answer Follow

answered Jun 8, 2010 at 19:41

Elaina R
133 ● 7

---

it looks as if you learned a really valuable lesson in the group project so well done to your college. Working with people IS tough but you're going to spend a lot of time doing it. And it can be rewarding too. – High Performance Mark Jun 9, 2010 at 8:07

+1 for bug tracker - the one we use (don't know others) allow us to add notes, and we can follow the entire history of the bug and remember things a lot better if something comes up 3 months after it was fixed – Rox Jun 9, 2010 at 13:08

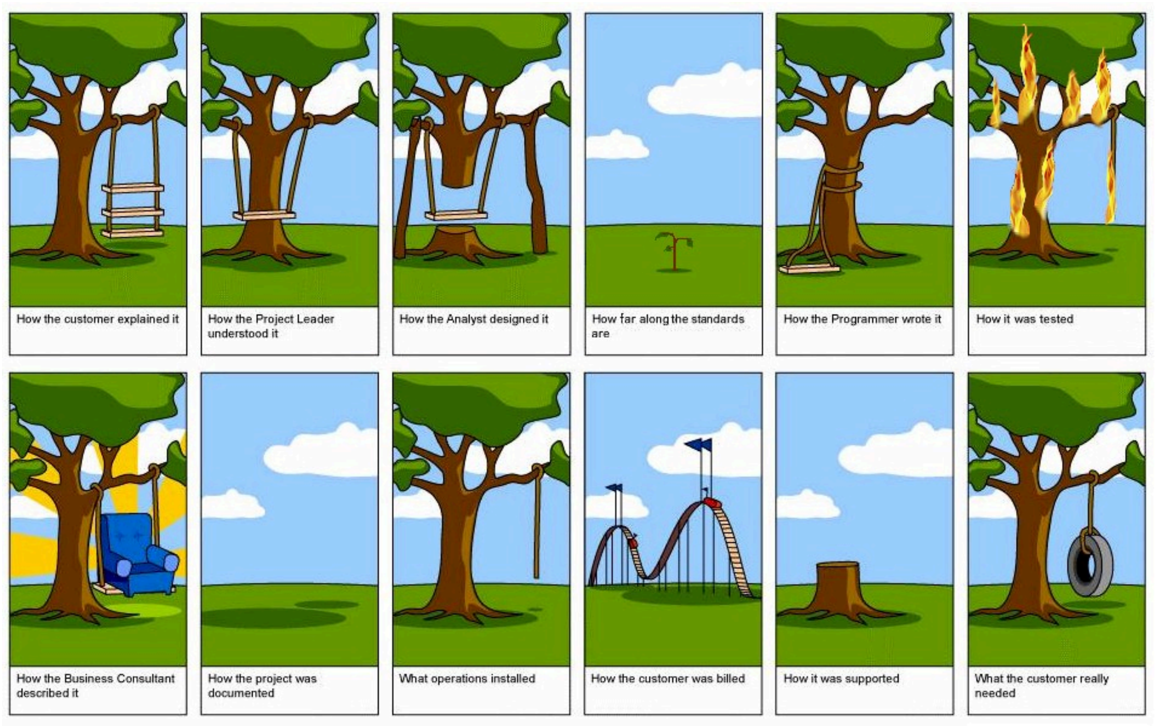
When I was at uni, every single project that involved a group failed to deliver because of group dynamics, communication or weak links. The advantage when you work for a company is that *totally* incompetent or unmotivated colleagues (usually) don't hang around that long. – Benjol Jun 9, 2010 at 13:30

@Benjol - Couldn't agree more! Thanks for the feed-back everyone :-) – Elaina R Jun 9, 2010 at 13:52

9

> how programmers work together on a piece of software in a company

Developers never work as a team. Teams suck. Dilbert is funny not because he's a comical character like Goofy. He's funny because he's real and people recognize the situations he's in.


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How far along the standards are
How the Programmer wrote it
How it was tested
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Share Improve this answer edited Jun 9, 2010 at 13:01

Follow

answered Jun 9, 2010 at 12:27

Andomar
238k ● 53 ● 385 ● 410

8

Generally it is good practice not to check build artifacts into the repository. The repository will contain the source tree, build configuration, etc - anything written by a human. Software engineers will check out a copy of their code onto their local filesystem and build it locally.

It is also good practice to have unit tests which are run as part of the build process. This way, a developer will know instantly if his changes have invalidated any of the unit tests, and will have the opportunity to fix them before checking in his changes.

You might like to look into the documentation for a version control system (one of Subversion, CVS, Git, etc) and for a build system (for example, in Java there are Ant and Maven).

Share Improve this answer

Follow

answered Jun 8, 2010 at 18:37

danben
83.1k ● 18 ● 126 ● 146

If the build results are not in the repository, how will developers of large projects be able to run the test build? I'm not sure if this is my mentality only 'cause I've always worked alone, but I always check that things "work" when I run my program. – Leo Jweda Jun 9, 2010 at 6:15

Test builds (and data created by the build process) will either be bundled up in a installer or as a flat file system on a network share so it can be simply copied across. This is useful where you have dedicated testers so they can provide feedback on bug fixes, etc. – graham.reeds Jun 9, 2010 at 8:38

8

The big things are:

- **A plan** — If people don't know where they're going, they won't go anywhere. The start of any project therefore needs a few people (often the project graybeards) to get into a huddle and come up with a plan; the plan need not be very detailed, but it's still required.
- **Version control system** — Without this, you aren't working together. You also need the firm commitment that if things aren't committed, they don't count. “Oh, it's in one of my sandboxes” is just a lame excuse.
- **Issue tracker** — You can't keep track of these things by email folders. Should definitely be database-backed.
- **Notification system** — People need to know when things are committed to code that they maintain or comments are made to bugs they are responsible for. Email *can* work for this, as can IRC (provided everyone uses it, of course).

- **Build system** — It doesn't really matter *how* this happens, so long as with one action you can get a complete build of the current state of things, both of your development sandbox and of the main repository. The best option for this depends on what language(s) you're using.
- **Test suite** — A test suite helps people avoid silly errors. It needs to be as easy to run as the build (being part of the build is *good*). Note that tests are only a crude substitute for correctness, but they're a heck of a lot better than nothing.

Finally, you need a willingness to work together toward fulfilling the plan. That's all too often the tough part.

Share Improve this answer

Follow

answered Jun 8, 2010 at 20:00

Donal Fellows
137k ● 19 ● 156 ● 221

Itemized requirements can help, as can a continuous integration system, but they're really aspects of the other things listed. – Donal Fellows Jun 8, 2010 at 20:01

3

There is no standard for the things you're asking about. Rather, there are conventions and these depend heavily on the size and maturity of the organization. If you're in a small organization, say a couple of programmers, then things will probably be somewhat informal with the individual developers doing coding, builds, and test.

In larger organizations, there may be a dedicated build engineer and process. This kind of organization will usually do a formal build on a regular basis, say once a day, using whatever source code is checked in. The process will also usually include BVT (Build Validation Tests) and perhaps some regression tests. Developers will check out the code from the repository, work on their own portion locally, then check it in.

In the largest organizations, like Microsoft or Google, they will have a completely dedicated group and full lab that will build on a more-or-less continual basis, making the results of each run available. These organizations have very formal processes and procedures in place about what gets checked in and when, what the code review processes are, etc.

Share Improve this answer Follow

answered Jun 8, 2010 at 18:43

jfawcett

885 ● 7 ● 6

How do developers in MS and Google test their code then? Let's face it, no matter what we do, unless we compile and run our code we can never be sure it works. – Leo Jweda Jun 9, 2010 at 6:07

As above, it depends on the team you're on. The biggest teams, like Windows, will have a big and complicated branch structure that facilitates local testing of individual fixes, with early identification of integration problems as the change is moved up the branches until it gets to WinMain. That's what you have to do when you've got something like 5,000 devs and SDETs working on the product. BTW, SDETs at MS do actually program, a lot. Their test code is checked in alongside the product code and must meet similar coding and quality standards. – jfawcett Jun 9, 2010 at 20:00

3

There is no cookbook for working with software development, but in general the version control system should be the heart of your build system, even if you are working in a project where you are the only developer. Even in this case, being able to revert versions and read the version log is very welcome help in fixing bugs. This is not the only feature of a version control system, but this alone justifies installing, configuring and maintaining a version control system.

The build can be done either by each developer when adding new code, or periodically by a "build server". The last approach requires more setup, but helps finding out build errors sooner.

Share Improve this answer Follow

answered Jun 8, 2010 at 22:18

gclello

96 ● 2

2

The short answer - "It depends".

Currently, I'm working on a project by myself, so I'm the one who builds/uses VCS. I know of other places that you have teams working on the project together by *shudder* email. Or big (+5) teams using VCS.

On that note, I highly recommend learning at least some VCS, and Joel Spolsky has a great introductory tutorial for Mercurial. Bazaar (my personal choice) is similar, and then Git is the next nearest in terms of similarity, but probably more popular than either (at least ATM). After that you have SVN which is pretty weak in comparison.

Actually, Joel talks about most of your questions - I'd recommend reading the 10 years of archives he has - it's all highly useful information, and most of it pertinent to your current and near-future situation.

Share Improve this answer Follow

answered Jun 8, 2010 at 18:45

Wayne Werner
51.6k ● 33 ● 208 ● 300

SVN is probably the most useful to learn though, since most people don't use those new-fangled distributed VCSs (at least in a business). – Brendan Long Jun 8, 2010 at 18:49

1 Almost any version control system is better than none. Exceptions are VSS, RCS and SCCS; nobody should be using those in this day and age. (If only nobody was *actually* using them, but that's another story.) – Donal Fellows Jun 8, 2010 at 19:44

@Brendan Long: People have been using "those new-fangled distributed VCSs" (specifically git in my case) in the businesses I've worked in for the past couple years. – PTBNL Jun 8, 2010 at 20:15

@Donal Felllows: RCS is the only VCS on your list that I have used, but I think using it would be better than nothing. I do agree with your broader point that it would be better to move to a newer one. – PTBNL Jun 8, 2010 at 20:16

@PTBNL: It's very easy to migrate from RCS to CVS. The main thing you've got to get away from is having the repository locked by someone who's gone away on vacation! I've no doubt that there are better version control systems than CVS, but it most definitely can work in practice.
– Donal Fellows Jun 8, 2010 at 20:32

1

Proper programming is a deep thing that benefits greatly from experience. Pair-programming is like running multiple processors of awareness... one can overlook something seen by the other and so long as they are communicating it can result in great progress.

Share Improve this answer Follow

answered Jun 8, 2010 at 18:35
Hardryv
793 ● 7 ● 12

5 This doesn't have a thing to do with the question. – danben Jun 8, 2010 at 18:38

1 /disagree -- by my reckoning pair-programming is the most interesting and often even the most productive edition of 'programmers working together on a project'... which was essentially the question. Admittedly a microcosm of it, but indeed my favorite one. – Hardryv Jun 9, 2010 at 18:34

1

First of all, teams work by using repositories (which can be professional version control, or just a bunch of directories that is considered the 'live' one, however a revision control system is the de facto standard). Also, how the project is managed strategy depends on how you work (waterfall, agile, etc.). If you work in iterations, you build components/plugins/modules/libraries which are self-sustained, and you perform unit testing, until its signed off as finished. As a team, you work in a team which means you do don't work on the entire project everywhere at the same time. Instead, you get a task to perform inside a realm of the project. On some occasions you have to fix code that isn't yours, but that comes usually when a strange behavior occurs. Basically, you are doing the testing of the parts you develop.

Let me examplify this for you. You are inside a team of construction workers. The architect comes with a plan for a building, the foreman looks what the necessities are to construct and then hires the constructors. The mason does the walls, checks them for strength and glues them up nicely. The electrician does all the wiring inside the building so electricity can flow. Each man has their own job. Sometimes, the electrician might want to discuss with the mason if certain walls can be carved, but always in conjunction with the foreman.

I hope this is some help for you!

Share Improve this answer Follow

answered Jun 8, 2010 at 19:32

Shyam

2,377 ●8 ●33 ●44

Typically, the source control system contains the source code and usually does not have the binaries. If you want

0

to build it and run it, you would check the code out and build it on your local machine.

Some places run nightly builds to make sure everything works. There may even be some automated tests that are ran server-side. If the build or anything else fails, someone is notified automatically.

Share Improve this answer Follow

answered Jun 8, 2010 at 18:38

Donald Miner
39.9k ● 9 ● 97 ● 118

0

A good introduction to a method of using source control is Eric Sink's Source Control HOWTO http://www.ericsink.com/scm/source_control.html

In his examples he uses SourceGear Vault since he wrote it and all, but the methods can be applied to other version control systems.

Share Improve this answer Follow

answered Jun 10, 2010 at 7:23

ManiacZX
867 ● 1 ● 6 ● 11

0

This is again one good reason why one should look into Open Source projects.

The lead developers who work in big OpenSource projects (like Chromium , Mozilla Firefox, MySQL , Popular Gnu Software) are professionals. They have lot of experience and these projects have evolved over years with ideas from hundreds of such professionals.

Everything others mentioned in their answers (Plan, Version control system , Issue tracker , Notification system , Build system, Test suite,) can be found in these OpenSource projects.

If you really want an hands on experience I strongly suggest you to go through some popular & big OpenSource projects and then get the Source from any project (using Version Control) and build it your self.

PS: I'm also a student and involving in OpenSource projects is the best thing I ever did in my life. Trust me! you'll also feel the same.

Share Improve this answer Follow

edited Jun 14, 2010 at 10:20

answered Jun 14, 2010 at 10:14

claws
54k ● 60 ● 150 ● 201

Mentioning OpenSource--that's one weird way of writing it--when they check in source files with database connection information and such, how do they protect that information from people who may want to mess things up?
– Leo Jweda Jun 14, 2010 at 19:52