Project Sanity Checklist

Alexander von Zitzewitz

hello2morrow Inc.

"If you don't know where you're going, you're unlikely to end up there." - Forrest Gump

Overview

If you are or feel responsible for a non-trivial software project with 3 or more people working on it and want to make it a smashing success, this document is for you. It will ask a couple of simple questions about your project, that you should be able to answer with a clear "yes". If your answer is "no" or "maybe" it gives you ideas how you might come to a "yes".

The list might contain silly questions, but the purpose of a checklist is to check even trivial things. It happened in the past and it will probably happen again in the future that multi-million dollar software projects fail because somebody forgot to ask some of the sillier questions in this list at the beginning of the project.

The document is split into several sections covering organizational and technical aspects of a project. Every section contains a couple of questions that you should be able to answer with a yes.

Project Organization

Are you using any kind of a development methodology or process?

This can be Scrum [SCR], any variant of agile processes, any variant of RUP [RUP], Kanban [KAN] or even something you invented yourself. The main purpose of a development methodology is to organize work into manageable units and to enable you to track the progress. It also helps with risk assessment and management by identifying the most risky and/or difficult parts of the project. Usually those are the ones you want to address first.

Another advantage is that modern methodologies have formalized points of communication where team members can address problems and discuss solutions for those problems on a regular base.

If you don't have a process you might want to have a look at agile processes like Scrum or Kanbas. Nowadays almost everybody agrees that your development process should be an iterative process with iterations not lasting longer than 4 weeks. At the end of every iteration there should be a presentable result in form of implemented project features.

Does everybody in the team know and understand the methodology?

The most efficient process is of no use if people don’t adhere to it or don’t understand it. Everybody joining the team should get some kind of a process training or at least a decent explanation of how he/she should apply the methodology.

Are the initial requirements (or user stories) clear and understood by all team members?

This question is not as silly as it might seem. Many projects fail because the requirements are fuzzy and not well defined. Don’t start a project without having a clear understanding of the initial set of requirements. That does not mean that you have to know all requirements before you start working on your code, because this would require something like a waterfall methodology. But the initial set of requirements must be clear for everybody in the team.

Do you know who is responsible for the architecture and technical quality of the project?

This is a tricky question, because basically you have to be able to answer who will get the blame if the project fails for technical reasons. Of course you can only blame a person if he or she also had the power to move things into the right direction.

In many organizations you will find a technical architect, but the responsibility for the technical success of a project is considered to be shared by the whole team. In case of a failure everybody will most likely agree that the project failed because of outside factors like management pressure or fuzzy requirements (see question above). This might be very convenient, but unfortunately it is not at all helpful in ensuring a positive outcome.

I recommend that the technical architect is the person taking the responsibility for the technical quality and viability of the project. This also means that he or she must have the power to establish and enforce technical rules and guidelines. It does not mean that those rules and guidelines will be dictated by the architect. Usually the team should agree on a set of rules and guidelines and the role of the architect is to make sure that those rules are actually respected during the development and that rule violations are addressed in a timely manner.

If you prefer the model of shared responsibility you still need to designate a person who ensures that those commonly accepted rules are respected during the development phase, the person who looks after the “big picture”. In the end that means that this person will more or less take over the responsibility for architecture and technical quality unofficially - so why not make it official in the first place.

This job is certainly not always easy because it requires some authority to enforce the rules and also the ability to screen the team from direct management pressure. It also requires the ability to explain to the stakeholders why certain features have to be delayed for the sake of the technical integrity of the project. From time to time resources will be needed to refactor the code and to pay down technical debt (fixing rule violations).

Quality and Design?

Do you have rules for code formatting and naming conventions?

In a team it is very helpful, if everybody uses the same naming conventions for variables and types and also formats the code in the same way. It makes it much easier for everybody to read the code. Most modern IDE's provide an auto format feature that will format the code according to a configurable formatting convention.

Do you have a large scale architecture definition for your system?

Managing the dependencies between the different parts of your system is probably the most important task of a software architect. Many project failures can be tracked down to code dependencies getting out of control. When you start a new project you should at least be able to identify the major components, layers and subsystems of your system. Major components can usually be mapped to projects or modules in your IDE. Layers and subsystems should be reflected in the physical organization of your code.

Moreover you should be able to define the allowed dependencies between the elements defined before making sure that there are no cyclic dependencies between them.

The large scale architecture should be available as a diagram to all team members and everybody should understand the architecture and the dependency rules deriving from it.

Usually the architecture will be refined or modified as the project progresses. So it is not required to have a complete architecture definition at the beginning of the project. But for every iteration the architectural rules needed for that iteration must be defined in a formal way. There is nothing wrong with starting with a simple set of basic architecture rules and refining them later.

The concept and importance of large scale software architecture is described nicely in [AUD] and [LSD].

Do you have a commonly agreed set of quality related programming rules?

Programming rules are essential to maintain a certain level of technical quality in a software system. Typical rules would be “don't let a source file grow over x lines” or “do not allow empty catch blocks”. It is better to have a fairly small set of commonly accepted and useful rules than to have a large set of rules that nobody is able to remember of enforce. The DZone architecture essentials reference card describes a set of rules that have proven to be effective in many projects [AER].

Do you check and enforce architecture and dependency rules automatically?

This is often overlooked, but nevertheless quite important. The most elegant and beautiful architectural blueprint is useless if it only exists on paper and is not reflected by the code base. And if you don’t check conformance of your code to the blueprint it is safe to assume that rule violations will creep in at a rate that will grow over time. For Java you can find some some basic automated dependency checking in Sonar [SON], which is an open source tool that I recommend highly. For other languages or if you want to have more powerful features like architecture and dependency visualization or IDE integration you have to look at commercial tools, e.g. our own flagship product SonarJ [SNJ], which integrates nicely with Sonar. Since the cost of structural erosion is so high, these tools provide real value by minimizing your technical debt so that they are definitely worth the money you have to invest into them. Don’t think that you can get away with manual checking. This only works for very small projects.

Do you check and enforce your programming rules automatically?

Your rules are of no use if you do not check and enforce them in some automatic way. Java programmers are better of because there are a lot of open source programming rule checkers available. Again I recommend Sonar, because it works as a umbrella tool for quality management integrating all the popular open source rule checkers. And the best thing is that it can be expanded by writing plugins. For other languages or if you want very specific checks you will have to look at commercial tools.

Do you measure your technical quality on a regular base?

If you were able to answer the two previous questions with “yes”, then you have everything you need.

The number of rule violations reported by the automated checks gives you a good indication about the amount of accumulated technical debt and the overall technical quality of your project.

Do you give your team the time needed to cleanup rule violations on a regular base?

If the number of violations gets too high it is wise to use a good part of an iteration for code cleanup (pay back technical debt). Of course sometimes your deadlines might not give you the time needed to do that and it is a deliberate choice to increase technical debt to meet a deadline. Just make sure that you keep technical debt under control and use time and resources to reduce it on a regular base. Otherwise the interest payment (additional effort on code changes caused by increased complexity) will reduce your team productivity significantly.

Technical Environment

Do you have a version control system?

Ok, I admit this question should not even be asked, but it is a checklist after all. Should you answer with “no” there are lots of excellent and free options available (e.g. Subversion or Git).

Can you build your system outside of the IDE?

It must be possible to build your system without any user interaction from the sources checked out from the version control system (VCS). If you can’t build without the IDE you will have to create a build script or a makefile. For Java you can use Ant or Maven, for the C family of languages you should use `make`, `nmake` or `msbuild`. Other languages usually also come with tools for building and deploying systems.

Do you have a build server?

Having a build server is very useful because it allows you to build the complete system after each change committed to the version control system, but at least once per night (nightly build). The nightly build is also an ideal place to run tools that automatically measure quality and rule compliance of the code. Again, if you don’t have it set up there are excellent open source build servers available. Two recommendations are Hudson [HUD] and Apache Continuum [CON]. They also are able to build non Java systems by running any shell script.

Do you have automated tests and are you able to run them automatically with your build script?

Every project should cover all complex code sections with unit tests. The tests should be run automatically at least once per night. If a change causes a test case to fail it “breaks” the build. Of course you want to know that as early as possible. Most modern programming languages have unit test frameworks available for free.

References

[SCR] http://en.wikipedia.org/wiki/Scrum_(development)

[RUP] http://en.wikipedia.org/wiki/IBM_Rational_Unified_Process

[KAN] http://en.wikipedia.org/wiki/Kanban_(development)

[SON] http://www.sonarsource.org/

[SNJ] http://www.hello2morrow.com/products/SonarJ

[HUD] http://hudson-ci.org/

[CON] http://continuum.apache.org/

[AUD] Applying UML And Patterns, Craig Larman, Prentice Hall 2002

[LSD] Large-Scale C++ Software Design, John Lakos, Addison-Wesley 1996

[AER] DZone Architecture Essentials Refcard