Guide to coding ajcw

Learn how to create your own artistic images and animations and display them in our online gallery, which has now been expanded to suit self-students. Our world is becoming more digitized. For many of us, barely a day goes by without recording videos, taking and editing photos, and sharing digital content in multiple apps. But how well do we understand the technology we use and how digital information is created and manipulated? With many careers associated today with some form of computing, there is a growing need for people to move beyond digital literacy to understand how digital technologies work, and to develop literacy in code. This course will help you purchase it. In this course, you will not only learn about the inner workings of your digital world, but you will also create and manipulate images using code, creating new works of art and interactive animation. Your images and animations will be displayed in an online art gallery, forming part of a dynamic learning community. You will also develop effective computational thinking and concept skills, transferable to other coding environments and programming languages.Computational skills of introductory programming concepts such as sequencing, iteration and selection skills and knowledge on how to create art and basic animations with ProcessingJS Preparing to study computer science or other programming languages Get an instructor signed a certificate with the institution's logo to test your achievements and increase your employment prospects. , or post it directly on LinkedInGive itself an additional incentive to complete the courseEdX , a non-profit organization, relies on proven certificates to help fund free education for all around the world As web developers we all love code; That's why we do what we do. I guess we all strive to be the best we can be. Working in a fast-paced environment at BKWLD, our development team must learn to adapt at the moment to meet deadlines, most of which arrive a little faster than we would like. I often have to try to bridge the line between doing something well and doing it quickly. It is expected that they can both be achieved, which is sometimes true. More often than not, however, I have to lean more one way, choosing to either do something clean and beautiful, or do something that is complete when the customer needs it. Which approach is better? Our CTO, Justin Jewett, summed it up perfectly when he told me: We need fewer killers and more street fighters. Jewett points out that we need people who can code quickly, roll with punches and do the best work possible - something that's especially difficult when things get heated and customers are less This has led to many intense discussions about which approach is the right one. Poetry is good There is a reason good code is considered a form of poetry. It's elegant, clean, clean, read, and have fun writing. These are all exceptional qualities that we must strive for every day. This approach is philosophically correct. If the code is structured well from the start, then, at the end of the game, things are easier to find and edit. For example, creating a JavaScript file for stored config level variables is a good practice, making setting up things like animation speed and delaying the duration of later wind. The speed of goodSpeed is often overlooked and/or argued about among developers. An easy way to do something is often seen as bad or amateurish. Labels and hacks are further frowned upon, and their practitioners are, according to the community, bad developers. I'm a proponent of speedy development for many reasons, the boss of which is getting things done on time - or early. This leaves more room for polishing, and can make both manufacturers and customers very happy. Not everything fits the convention.' There are times when simple image tags, tables, or even (dare say it?) frames are a quick solution to a problem that will take much longer to build using standards or some new innovative workflows. I worked on sites that were too complex for their need and context. Not everything requires complex environments, Python framework, or mined hashing scenarios. All of these things have their place for specific projects, but a good developer should choose what is best for the scope of the project, rather than just using the most sophisticated technology in all cases. Find out what's right for the project, given the project you're working on, think about what the needs are and where most of the time should be spent. For example, if you don't need a complex JavaScript on your site, don't add scripts and modules to the script load that will take time and energy to set up. Instead, a simple script file or even some inline JavaScript will work just fine. So the requirements are met and you can spend more time on the rest of the site. If the project is personal, which you are very passionate about, spend all the time making sure that every line of code is where it needs to be and comes down to its clean form. If the project is for a three-month campaign that is due to be completed next week, the shortest path to the finish line is probably better. I have only been a developer for five years, and 95 percent of my professional projects are the last. We must complete the quality of work as soon as possible. Words: Matt AebersoldMatt Aebersold is a developer at BKWLD. This article originally appeared in the clean issue of 246.Liked this? Read this! What is your code philosophy? Tell us in the comments! A few other articles in this series about effective code reviews covered when you have to schedule reviews that should participate, participate, problems for groups that are not geographically distributed and the role of tools. But at some point, you end up bringing a few adults to the conference room and you'll close the door. Now what? In this article I give you clear guidelines about the process and things to look for. ULTIMATE GUIDE TO CODE REVIEWS Running Effective Code Review 5 reasons for software developers to do a code review (even if you think they're a waste of time that look in the Code Review Code Review Software Tools Help, rather than discourage how to keep the code review How not to run the Code Review Do Spot-On Code Reviews with remote teams first , set the basic rules. and get the workflow in place. This means that participants have source code over time, setting roles for meeting attendees, and creating a process to follow up after a code review. To make the review successful, says Jason Cohen, founder of Smartbear Software, eliminate parts that make developers unwilling to participate. Identify parts of the review process that are busy with work or dead time and eliminate them, he says. The only action that is clearly not a waste of time is to critically look at the code and talk about it with others. Everything else is extra. For example, the developer Adwait Ullal sends a notification a week before the code is verified, ensuring that there will be three collegiate reviewers, as well as a scribe and an author. Reviewers are notified of the artifact in question and its location. Ullal sends a reminder three days before the review, in which he tells everyone that they can bring notes to the meeting (not the artifact's source code); only the author can bring the source code. (This eliminates last-minute reading.) The scribe takes notes and sets a deadline for when corrections should be made. Once the code author fixes the problem, he sends one e-mail to the team. Make sure you budget enough time for the review, and for reviewers to examine the code in advance. If the reviewers aren't ready, says Ben Sweet, chief engineer at Lear Corporation, the experience is terrible. The result is a walk in which someone reads to the meeting participants line by line of code. It's not very effective and very boring. In these types of reviews, people will probably identify cosmetic deficiencies that don't actually affect the quality of the product. Theron Welch, a software mentor who helps Microsoft build a team in China, offers one interesting way to ensure that all profits are prepared. We have a simple rule that says: If you don't have red marks on the printouts of the code, you are not allowed in the review. There are many variations on this process and some developers get very passionate about them. (It won't surprise you. passionate?) For example, Cohen's additional reduction includes collecting files for and sending to others: source files, checklists, claims documents, related defects, testing specifications, whatever. Integrate with version management to create differences automatically, or use a commercial review tool that does the same thing, he advises. A typical homegrown version for email diff automatically when signing up, but this assumes that you want to consider after registration, not before, which most people don't like. Checklists and other documents should be placed on the intranet system so they can be linked, hopefully automatically. In addition, Cohen objects to the typical Scribe commenting process, which takes a long time, writing: On line 142 of the file //depot/foo/bar/... because both the reader and the author have to look for the code in IDE. Tables or tools help, he says; or develop a cut like filename:linenumber, adding a path only when camouflage is needed. Not everyone wants printouts. Jeff Benson, Geneca's technician, recommends using a projector connected to the source control system. So you can drill in and out of unforeseen parts of the system, he says. It's also easier and more convenient to keep a small group of developers focused if you're all looking at one place on the wall rather than flipping through a shee of handouts while trying to keep up. Set basic rules I've already considered some of the relationship issues in other articles in this series (see Running Effective Code Review), but some elements are worth mentioning during each review. Participating in advocacy to encourage code reviews to become part of the culture, says Oliver Cole, president of OC Systems, and lead to open source Eclipse testing and performance platform tools project. Make sure you keep saying things like us all know it's the right thing to do and let's be selfish. He adds: Once a throw in Wow, these code reviews really enhance the quality of our work (even if it doesn't). (Gosh, quality is hard enough to measure anyway.) One important suggestion is that the developer whose work is being considered be asked to explain its choice. Ask, don't tell me. Advised by Benny Czarney, founder and CEO of OPSWAT, which makes development tools and data services to manage the security functions of end applications, ask questions that are productive. Don't say things like: It's not right or why did you do it that way? Ask for an explanation: Can you explain what this piece of code is doing? or What led you to this implementation? as the author answers your questions, Czarny says, has her comments inserted into the code detailing the explanation. in their opinion, there is a problem, says The King, and not only by ego. The code author can acquire development skills and draw his own conclusions about something the reviewer emphasizes. For example, let's say that the code a scenario that could become a vulnerability. A reviewer who notices this potential vulnerability asks insightful questions. As the discussion continues, the developer has a light bulb moment and suddenly recognizes the missed questions. These are the kinds of experiences you'll learn and draw in future code development and analysis sessions, Tsarney said. Senior Software Engineer Michael Doubez suggests you go to south-on to understand that each reviewer will get one of their code reviews. This makes people kinder knowing that one day they will be on the other side of the table, he says. However, do not be afraid to criticize the work in question. Senior software engineer Alexei edlov once reviewed the code, which was rather sloppy, had a lot of copy and paste and just screamed for refactoring. I had an idea of how it happened and it was far from entirely the author's fault, he says. There was no point in criticizing the developer or saying he was stupid. Instead, however, just say that the code isn't as good as it should be, and here are ways to fix it. A few more bullet points before we get into specifics: Criticism of the code, not the person. Point out good things, not just weaknesses. Don't focus on the solutions you'll find. No more than five minutes should be devoted to possible resolutions. Avoid second guessing. Don't give more than 25 words to describe the process. Prevent breaks. Look: Compliance with style and standardsSSEier experienced developers and IT managers note that a review of the code should ensure that the software follows the accepted guidelines for the style of coding. The most important aspect of the software code verification process, first, is clearly communicating company standards for coding: Here's how we develop, test and lay out code, says Jay Dickins, founder and president of Deacom, Inc., a manufacturer of ERP software for building components and the package processes industry. You should never be able to distinguish between two developers code. But to check the code to make sure that the software meets company standards, then you have to have these standards documented. To eliminate stupid code review comments such as gaps or capitalization, you need a good style guide, says Microsoft's Welch. The style guide describes how the team prefers to comment; Name classes, variables and methods and how to format. It should not try to pass on best practices software like how best to deal with memory that can't be highlighted, or tracking streams, and so on. It should focus only on style. The style guide should be short enough to be easily understood by all team members without covering every sickening detail, Welch explains. teams still need the freedom to write code the way they see fit. But with a style guide, he says: Developers are then responsible that they won't have any style problems in their code. If the team command established good style guidelines that are readily available and used frequently, this should be easy. A style guide can solve a lot of team problems and prevent silly fights by suggesting that they don't cause more of them. As software contractor Steve Silberberg notes, No person will ever agree to variable names, module formatting, or coding conventions, and they will never come to an acceptable compromise. According to Silverberg, code reviews that inculcate standards can cause resentment (sometimes serious) among those who have to change their naming conventions. For some developers, this change significantly reduces their performance, forcing them to reprogram themselves to start thinking differently. Coding standards arguments are fierce and territorial and will rip through the fabric of any team unity designed-all for the modest (at best) benefits from the ease of coding maintenance, he says. Look for: Everything except StyleIt usually switch to style issues during code review, but in fact, there are better things for you to do, say many experienced developers. It's easy to get distracted by code style, says James Pitts, vice president of software development and management at Embarcadero Technologies. The wrong style won't break the product; It's just harder to read. The key to reviewing code is not to spend more time browsing and then implementing, Pitts says. Fast and efficient is the name of the game. Among its guidelines: Pay attention to the programming language. People often make mistakes similar to typos around the misuse of the pointer, missing breaks in the switch, etc. If it is very localized, you should be able to consider quickly. If it has more influence, you should take longer. If you find it difficult to understand the code, stop and ask for help. Don't waste a lot of time trying to figure it out. E. William Horne, systems architect at William Warren Consulting, suggests that code review focuses on reuse, easy transition to other locations, using standard procedures and functions whenever possible, well-defined interfaces, system calls, and file formats. Focus on catching items that compiler, liner and other automated tools miss, said Eric W. Brown, president of Saugus.net. This may seem basic, but most new code reviewers don't follow it, and the time spent on such items is mostly wasted, Brown says. It's much better for code reviewers to look for logic errors, he says; the main mandate is to find this kind of glitch when the code doesn't do exactly what it should (even if it's clean and seems to work in most cases). Others agree. J Schwan, Managing Partner Consulting notes that code reviews are not intended to replace specific testing or functional testing - there are more efficient and automated ways to do so. Nor should it be a formatting exercise. Some of the worst code reviews I've sat through involved measuring backed spaces or attempting to go through the logic of code that covers multiple modules in their head, trying to find errors in logic. Both of these exercises are something that can be performed much faster than a human computer. (For more information about using the tool during the code review process, see reviewers to track down potential issues where the work code might break, brown says. And if it hadn't been fixed back during the review it would have resulted in a very difficult error, he says. Page 2 Code Review should reflect the concerns of users or stakeholders, and they vary according to the roles and experiences of the developers. As longtime developer Chuck Brooks explains, a business owner (such as an independent software provider) needs to implement a storyboard or other block-level (and graphic) description of the requirement. The issues that need to be addressed are whether implementation is documented; Whether to reuse the use of code from the object library what's new that hasn't come from the library; how the software is tested and whether the test (s) will be automated for subsequent regressive testing in updates and point releases. In contrast, Brooks notes, the contract consultant working on the new development should look for clarity of code. For the development of maintenance-oriented contracts, the main concern is understanding policy before we come to a meeting, he adds. This focuses on champion issues, which include version management, testing, rollback procedures and documentation and training. New developers should be warned to focus on the technology, Brooks says. Coders don't need to know about business practices, and business analysts get really paranoid. Make sure you don't duplicate the function, says Dickins. For example, the code review process should ask, Do we do the same thing several times without using a common feature? At Deacom, we're always going backwards and trying to simplify the code, Says Dickins. When you're dealing with a complex issue like an ERP application, it's hard to say: Wow, it's done and then go back through the code again and look for patterns and see how we can make it easier. But it's an important performance issue as well as providing managed code, he says, because you'll live with that code for a very long time That's why the code-checking process shouldn't matter much. This should be the last step of simplification, he adds. A Lot who answered my questions about how to run effective code reviews put particular emphasis on topics that are closest to their own hearts or their own domain experiences. For example, Paddy Sreenivasan, co-founder and vice president of design at zmanda, a provider of backup and open source software, is committed to focusing on security, bug handling, and user messages in code. But all experienced developers have a broader perspective, too; Sreenivasan also checks to make sure that any code changes are documented (i.e. comments to the code) and have unitary tests. All this suggests that the process is working. But it's not, always. For more information (and moaning in understanding) see how not to run the review code. And 26 ways to find out your software development project is doomed while you're at it. Follow-Up and MetricsDon't forget to work through the process of what happens after the code is checked, or the whole exercise will be wasted. Lear's Sweet assumes that at the end of the meeting, participants consider and rank the found items as immediate change, change by (some date or milestone of the project) and recommendation (change at the developer's discretion). The leader should keep an eye on the developer to make sure the issues on the list are resolved, Sweet adds. Microsoft Welch suggests you calculate the effectiveness of your code reviews by tracking two parts of the information: the average rate of bug fixes for a team member (which it says is usually one or two per day), and the number of major problems the review code found. Suppose you found five major problems in your latest code review, and the main level of fixing your team's bugs is one per day, he says. If there was no code review, it would have cost about five days to fix these errors (the number of errors once, how long it takes to fix a serious problem). If there were five participants in the code review, each of whom was preparing for one hour and the review of the meeting lasted one hour, the total time was 10 hours. It's much less than five days away, so your review of the code was worth it! These aren't the only things you can keep in mind when you check the code, of course. You'll find several other suggestions in other articles in this series Review of Effective Code:

sistema de encendido computarizado nissan
normal_5f90c72527463.pdf
normal_5f8a3ec5b4afd.pdf