Quality Assurance

Eddie S. Jackson

Kaplan University

IT510: Systems Analysis and Design

03/16/2014

Quality Assurance

As I am the systems analyst for this project, it is my responsibility to take in all the available information, and determine what the best course of action will be to complete this project. This would include the scheduled layoffs, the reduction in budget, as well as listening to personnel complaints, concerns, and suggestions. It is my job to bring people together, combine our resources, and complete the Quality Assurance and Implementation phase. In our case, we need techniques that have been tried and proven at the company (rather than introducing some new technique). There are two well-known, and quite practical, techniques that the company could utilize to complete this project; they are the System Development Life Cycle (SDLC) using computer aided software engineering (CASE) tools, and pseudocode. SDLC is a type of process that is used in the software development process to create higher quality software (Tutorials Point, n.d.). SDLC provides a phased, systematic approach to effectively guide the analysis and design of information systems (Kendall & Kendall, 2011). CASE tools offer software engineers a way to collaborate with colleagues on software development projects without getting bogged down in source code (Select Business Solutions, n.d.). Pseudocode, as the name implies, is a type of code that is fake. Pseudocode is a great way to allow multiple programmers to work together on software development projects without worrying about the coding specifics.

Employing the use of SDLC with CASE tools and pseudocode would be very convenient because they are already being used, or at least somewhat understood, by our team members. The benefits of using SDLC with CASE tools would be establishing clearly defined milestones, having access to a highly structured, organized methodology, and concise requirements definitions are considered upfront (Kendall & Kendall, 2011). SDLC encompasses

the entire software cycle methodology from planning, defining, designing, building, testing, and deployment. Of course, pseudocode has many great attributes of its own. With pseudocode, our programmers will not have to think about syntax, and can instead focus on the logical components of the software (Community College of Rhode Island, n.d.). Pseudocode can also make the documentation process easier by using simple or common terms when referring to parts of the software. In our particular case, both of these solutions could help in the Quality Assurance and Implementation phase. Specifically, SDLC and CASE tools could address the concern of consistency in documentation, as well as aid in creating a new clear and concise strategy for completing the Quality Assurance and Implementation phase. Pseudocode could help our programmers meet and discuss the logical components of our software system and not focus only on syntax, this in turn will speed up the design, testing, and implementation of our software.

After weighing each of the possible solutions, the best choice for the ongoing project will be SDLC using CASE tools. The reasons I have selected SDLC and CASE tools over pseudocode, is because the SDLC approach has an inherent, and quite comprehensive, methodology for testing and deploying software into a production environment. Utilizing SDLC and CASE tools is ideal for our particular situation because the core principles are about accomplishing as much as you can, with as little effort as possible. There is also user acceptance testing (UAT) that be added to the Quality Assurance and Implementation phase. This will ensure that personnel will have a better understanding of the final product, as well as allowing employees to provide critical feedback before the software system goes live. In contrast, pseudocode does offer many great ways to save money and time; however, its main focus is software development, and not the full scope of the project.

The Systems Test

Testing is vitally important in the software development lifecycle. We need to test to detect possible failures before actual go live date. Otherwise, workflow could seriously be interrupted due to downed systems. By not testing, we could potentially not catch problematic portions of the system that could cost the company much more money to fix after the go-live date. It is always cheaper and easier to address problems before a system enters into a production environment. Additionally, the testing phase gives us an opportunity to engage our end-users. We want the employees to be excited and to fully understand the new system; we want the adoption rate to be as high as possible. This can accomplish by including personnel in the final testing, collecting their feedback, and compiling the data to reach the best possible outcome (Atlassian, n.d.). Finally, this new system is a reflection of the personnel that have worked many long hours getting it up and running. The last thing we want to do is to skip the testing phase, and possibly deliver a flawed system to the enterprise.

When considering what I will do to make sure all the testing gets completed, one thing I can do is not wait until the end. Testing is important, and should be done throughout the software development lifecycle; thus, software testing must be factored into the overall design and implementation strategy. It will be my job as a systems analyst to develop a good rapport with the team members, and to open up the lines of communication utilizing email, instant messaging, telephone, video conferencing, etc. Subsequently, by establishing a good rapport and communicating effectively and efficiently, I can make sure the testing is progressive, and I can remain up to date at all times. I would make sure testing is added to the PERT chart, in all the right areas, so that everyone will be on the same page throughout the rest of the Quality Assurance and Implementation phase. As for maintaining my busy schedule, to make sure

development and testing stays on schedule, I could use Microsoft Project as a standalone solution, or even better, an online solution like Daptiv. Daptiv offers a way to manage projects on a single, collaborative platform, without the need to purchase and manage individual software licenses (Daptiv, 2014).

We can all appreciate the amount of testing that is done in an offline lab environment, but it is just not the same as running tests on live data. The potential problems that could arise if we were to skip the live data testing could be extremely serious and costly. For example, it is possible that test data and live data may have slightly different data lengths and sizes. This could cause application crashes, issues in saving the data, as well as just data loss. Because test data is not being updated as regularly as live data would be (or not updated at all), there could be potential issues in how live data is retrieved, displayed, and stored; this has to do with the differences in the inputs and outputs of test data versus the live data. Once live testing begins, there are factors such as user reactions to prompts and error messages that can be observed (Kendall & Kendall, 2011). Live data testing also presents a unique opportunity to monitor the application in a live environment, also known as performance monitoring. Performance monitoring would include the speed of data transfers, processing, and output. If the testing is skipped, we would be bringing a software system into a production environment with numerous unknown variables, which could ultimately result in disaster.

But let us be realistic. The shorter timeline will continue to be a challenge until the Quality Assurance and Implementation phase is over. This raises the question, "Can we just skip some steps of the software development lifecycle?" The short answer is no; steps cannot be skipped or omitted just because the project is running behind. To elaborate further, a good systems analyst does not skip steps, because those steps exist for a reason. There are many

technical and interpersonal components that I would consider essential to the software deployment lifecycle. These would include establishing requirements, designing, coding, testing, implementing/deploying, and maintenance (Kendall & Kendall, 2011). There are also subcategories to each of these stages, as well as a long list of collaborative efforts that must transpire to lead to the successful implementation of a software system. If I decide to skip or condense any part of the process, I would be setting myself of failure. For instance, if I decided to skip UAT, there is the possibility that something important was missed before full-scale integration. If I omitted documentation, users would not be provided the proper materials for learning the new software. And, if I cut the planning stage or the designing stage out of the project, it is almost guaranteed that the end product would be riddled with problems, and most likely be unusable.

Training Users

The obvious problem that I see is that management and other employees are in the training sessions together; this is causing multiple issues. For instance, you have managers talking about advertising, whereas the regular employees are just wanting some hands-on training. One major complaint was that the training was boring; it was like sitting in lectures in school. Training should be engaging, and should provide the employee with the best possible learning experience. One last problem I noticed is that you have highly skilled people grouped in with non-skilled, or lowly skilled employees; this is not the best way to create a learning environment that will be the most effective for the attendees.

Now, due to scheduling constraints, there has to be some creative planning and scheduling that I must think about. First, I would split up the novices and experts (or highly skilled) employees (Kendall & Kendall, 2011). Even with tight schedules, I can divide everyone

up into separate rooms. This way I can tailor each of the levels of training accordingly. Additionally, I would provide hands-on training to all novice users or anyone else that requests it, as well as a written manual that employees would be able to carry back to their departments. A combination of training methods is the best approach (Kendall & Kendall, 2011). I also like the idea of open communication, so I would give all employees my work telephone number and email address, just in case if they have questions later; I would want them to contact me directly. Even if I did not know the answer when they contacted me, I could find it for them. The idea is to make the employees feel as comfortable as possible during the training session.

The advice that my team would ignore for the training session would be grouping novices and experts together, and not making the training interesting (or fun). Because timelines and schedules really are tight, my team would decide not to take the time to split people up based upon their skillsets. Likewise, the short timelines would limit the amount of creativity put into the training itself. The end result will be everyone will be piled into training rooms and training will seem boring to some employees.

References

Atlassian. (n.d.). Software Testing: A Culture of Quality. Retrieved from https://www.atlassian.com/software-testing

Community College of Rhode Island. (n.d.). Pseudocode: An Introduction. Retrieved from http://faculty.ccri.edu/mkelly/COMI1150/PseudocodeBasics.pdf

Computer Aided Software Engineering (CASE Tool). (n.d.). Retrieved from http://www.selectbs.com/analysis-and-design/computer-aided-software-engineering-case-tool

Daptiv. (2014). Portfolio Management. Retrieved from http://www.daptiv.com

Kendall, Kenneth. E., & Kendall, Julie. E. (2011). *Systems Analysis and Design Eighth Edition* Prentice Hall.

Tutorials Point. (n.d.). SDLC Overview. Retrieved from http://www.tutorialspoint.com/sdlc/sdlc_overview.htm