(IJIRSE) International Journal of Innovative Research in Science & Engineering
ISSN (Online) 2347-3207

Modularity and Reuse

Charles Edeki, Ph.D
American Intercontinental University, Department of Information Technology
School of Information Technology
231 North Martingale, 6th Floor
Schaumburg, IL 60173.

Abstract **:- As software development teams evolve, projects begin to borrow from knowledge previously acquired from working on older projects. A well-organized team will keep copies of these various projects on hand, just in case those may be reused later. While there might be times where contractual obligations or limitations might not allow a team to use specific code for other projects, that doesn't mean the team can't use their lessons-learned to assure that the next project goes more smoothly. Strict adherence to object-oriented (OO) design approaches helps this in that OO fosters modular development where individual modules can be replaced without affecting the rest of the code or architecture. Regardless of the specific project, it will often be the case when a customer asks for something very similar to something a team has worked on before. When that happens, they can look through their records to determine whether they can base the project on older work, or if they have to start from scratch.**

I. Dust off the Repository

In a hypothetical scenario, a company asks a software team to develop an application for them that is very similar to something the team has accomplished before. Hopefully, the team employs some form of version control system, such as Subversion, to allow them to maintain a detailed history of the development of the application as a whole. The same organized team will also maintain not only a log of each major revision of the software, but also maintain tagged copies of each, so that when this situation arises, they may perform a checkout of that revision as a starting point for the new project. Often, when a company's needs are only 20% different from a project that has been completed before in the past, these differences will be more superficial features over architectural differences. If this is the case, many of those features can be added without much work to the underlying system. However, even if a major architectural change is needed, such as the requirement to use Oracle over SQL, a properly designed system is still developed in such a way where the implications on other code are minimal.

II. Layered Architecture

The layered architecture concept attempts to divide the system into various sections that deal with the same purpose. For instance, a typical layered architecture is the "3-Tiered Architecture" (Papagelis 2014). In this architecture, the data store is considered to be one layer. This layer is responsible for all actions that read, write or delete data, typically as a database. The next layer is the business layer. This layer maintains all the logic of how the system operates, including any algorithms, business processes and possibly security/authentication. This layer serves as an intermediary step between the data store and the final layer of 3-Tiered Architecture, the Presentation layer. Presentation focuses solely on how the user interacts with the system. Those interactions are then translated into inputs that the business modules use to execute the appropriate business processes. While this is one of the more common architectures used today, especially in web applications, a system may have any number of layers so long as they are meaningful and it adds value to keep them separate.

The major benefit to a layered architecture approach in this scenario is that the system itself was built to be modular. Even if the 20% difference was a major consideration such as a change to the database, a properly developed data model layer would allow the team to focus on that layer without having to change the rest of the system. This is especially true if the data model utilizes the repository pattern which subdivides the data layer into the actual database code and a repository that converts the results of the database queries into business objects (Millet 2010).

III. Object-Oriented Design and Reuse

Going deeper than good architectural design is good system design. A team using an object-oriented approach to the design of their system will benefit from the 4 key elements of the philosophy: Abstraction, Encapsulation, Modularity & Hierarchy (Booch et al. 2007). Chief among these four elements is modularity. With modularity comes the ability to remove and replace sections of code without affecting the rest of the system. So long as the inputs and outputs match what the system is expecting, this module can be replaced to help support new functionality from a customer request that is similar to older projects. Additionally, in a good modular system, new modules can be added to a business process by changing a few method calls. By doing this, extra business process steps can be included into old projects to support a business that has, perhaps, a more complex internal process. In either case, adhering to good object-oriented design, teams can save a lot of time by knowing how to reuse their old systems to support new requirements.

IV. To App or Not To App

A growing trend in the past decade has been the rise of the mobile app. In this scenario, a company requests that a mobile app version of the software be made available so that their crews may report from customers' homes. Mobile applications can be useful when the device itself serves some purpose. For example, if the device's accelerometer, camera, GPS, etc are needed for the app to function properly, developing a mobile app can certainly add value to the overall system. However, one caveat is that often times, companies might be asking for an iPhone/Android app when they don't really need one. The primary concern in this scenario is cost. It is rare when a programmer can create an online system and an app for both the iPhone and Android. As such, more developers will be needed so that the apps themselves can be developed, as well as some additional developers and quality assurance members to assure that the integration between the two systems is functioning properly. From the requirements given, the team only needs to update a status remotely. This can easily be done in a web browser, without the need to develop an application. Indeed, extra care must be given to ensure that the design of the site is responsive to devices of various form factors, but that will still be immensely more cost-effective than hiring additional developers per platform. This concept is supported by Tim Berners-Lee, founder of HTTP protocol and often called "the father of the Internet". In one keynote, Berners-Lee emphasized the power of HTTP protocol and web applications by boldly suggesting that virtually anything that can be done on a mobile app can be done in a web browser (2013). While this is certainly possible, it is important to consider that local assets on the device itself are still much easier to access through a mobile app's framework than through a browser, due solely to security limitations placed on the browser.

V. When All Else Fails

It is certainly possible that a team will not have an application available to them that fits the needs of a new requirement, or that they do not have Subversion or that they are simply not legally allowed to reuse the code. In either case, a team in this scenario can still tap into their own experience to improve the quality and development efficiency of their next project. Freeman & Pryce write about the importance of integrating customer feedback into the specifications of the project after each iteration in so that the team may learn more about it and evolve the system along with the customer's understanding of their needs (2009). Perhaps development teams should integrate a feedback loop on themselves after each project, considering the lessons learned and using that feedback to modify their processes and techniques for future iterations (projects). Indeed, the very nature of iterative development cycles allow developers to learn as they progress, so too should they apply that same mindset on their own processes to ensure they are the highest quality team possible.

References

[1] Berners-Lee, Tim (2013). Open Web Platform. SXSW. Lecture Conducted from Austin, TX.
[2] Booch, G., Maksimchuk, R., Engle, M., Young, B., Conallen, J. & Houston, K. (2007). Object-Oriented Analysis and Design with Applications Third Edition. Addison-Wesley.
[3] Freeman, Steve & Pryce, Nat (2009). Growing Object-Oriented Software, Guided By Tests. Pearson Education.
[4] Millet, Scott (2010). Professional ASP.NET Design Patterns. Wrox.
[5] Papagelis, Manos (2014). Web App Architectures: Multi-Tier (2-Tier, 3-Tier) & MVC.
[6] University of Toronto. Retrieved From: http://queens.db.toronto.edu/~papaggel/courses/csc309/docs/lectures/web-architectures.pdf

AUTHORS PROFILE

Dr.Charles Edeki received his Ph.D degree from Capella University in IT with specialization in Data Mining and Bioinformatics in 2012, MS in Computer Science in 2001 from Long Island University and MS in Biotechnology from University of Maryand. He has been teaching computer science and math courses since 1999. He is member of Association for Computing Machinery (ACM) and Association for the Advancement of Artificial Intelligence (AAAI).