Software engineering has become one of the pillars of the modern world’s economy. The necessity of up-to-date software is essential as the system must be able to handle new features and requirements. This essay discusses continuous integration, and how it can address the problem of software evolution in agile software development.

All software systems are vulnerable to errors given the ever-changing and ﬂuctuating nature of the competitive market and customers’ needs. New bugs surface every day which must be ﬁxed. New requirements emerge and need to be implemented in later versions. New versions of hardware and interactive components are released, making it necessary to adapt systems to them. Therefore, developers must keep themselves ahead of the competitors by coming up with and installing new features to software constantly. As changes are applied, a new version of the software appears. As such, most systems could be considered as a set of iterations of different software. Thus, it is critical to keep track of the versions constituting a system, which raises the need to have efﬁcient conﬁguration management. Ineffective conﬁguration management can often lead to losing track, modifying, and delivering the wrong version to the customer. There are four activities involved in conﬁguration management, which are system building, version management, release management, and change management (Sommerville, 2016).

Software development is a process of integrating different components, external libraries, data ﬁles, etc. to create a fully functional system. It involves a large amount of information adding up to a complex process that is prone to errors. In Agile development, in which the code is changed regularly by many developers simultaneously, discovering defects and repairing them early is pivotal. This problem could be addressed by continuous integration. One way of conducting this step is through continuous integration. Continuous integration is a procedure in which the building and testing of a system are carried out continually to discover problems (Sommerville, 2016). It involves rebuilding the product as soon as some changes are made in the source code. Continuous integration requires team members to integrate their work frequently, which might lead to multiple integrations in a single day. Each integration is checked by an automated build that detects errors as quickly as possible. Continuous integration is supported by various tools which are servers running automated testing suites. Those tools thus act as conﬁguration management by giving an insight into the steps required to set up and conﬁgure a particular software system (Stolberg, 2009). This reduces the integration period and hence speeds up the delivery.

Steps in continuous integration involve importing the latest source code from the version database into the developer’s workspace, then compiling and testing it to check whether the system achieves all its requirements. If it fails, then whoever updates the system last is informed to repair the bugs. After the build passes all the tests, changes are made to the system and the new system is again tested. If errors arise, the developer continues editing the system. Else, the system is implemented and tested on the build sever system. If it is stable enough, then it could be finalized and implemented into the baseline (Sommerville, 2016).

Continuous integration enables testing the system in conjunction with its development. Bugs are discovered early and are easy to track down due to shorter integration periods. This proves to be an economical solution over the whole development cycle. It also avoids last-minute malfunctions close to the release dates when developers try to implement the very last versions, which might be unstable. Furthermore, it ensures the constant availability of a build that is ready to be shipped. Since code is frequently integrated and used by various developers, the need to create more modular and less complicated code is apparent. This constant refactoring reduces the effort and increases the productivity of the developers. On top of that, continuous integration reduces the cost of testing. By incorporating quality testing within the development cycle, quality is immediately added to be the product (Stolberg, 2009).

In conclusion, continuous integration plays an indispensable role in improving the quality of software in Agile development. Developers and enterprises alike should adopt this practice to further extend efficiency and productivity during the development process.

Reference

SOMMERVILLE, I., 2016. *Software Engineering*. 10th ed. Boston: Pearson Education.

STOLBERG, S., 2009. Enabling Agile Testing through Continuous Integration. *2009 Agile Conference* [online], pp.369-374. [viewed 13 November 2021]. Available from: https://doi.org/10.1109/AGILE.2009.16