Software build

In the field of software development, the term **build** is similar to that of any other field. That is, the construction of something that has an observable and tangible result.[1] Historically, build has often referred either to the process of converting source code files into standalone software artifact(s) that can be run on a computer, or the result of doing so. However, this is not the case with technologies such as Perl, Ruby or Python which are examples of interpreted languages.

Contents

Functions

Building software is an end-to-end process that involves many distinct functions. Some of these functions are described below.

Version control

The version control function carries out activities such as workspace creation and updating, baselining and reporting. It creates an environment for the build process to run in and captures metadata about the inputs and outputs of the build process to ensure repeatability and reliability.

Tools such as Git, AccuRev or StarTeam help with these tasks by offering tools to tag specific points in history as being important, and more.

Code quality

Also known as static program analysis/static code analysis this function is responsible for checking developers have adhered to the seven axes of code quality: comments, unit tests, duplication, complexity, coding rules, potential bugs and architecture & design.[2]

Ensuring a project has high-quality code results in fewer bugs and influences nonfunctional requirements such as maintainability, extensibility and readability, which have a direct impact on the ROI for your business.[3]

Compilation

This is only a small feature of managing the build process. The compilation function turns source files into directly executable or intermediate objects. Not every project will require this function.

While for simple programs the process consists of a single file being compiled, for complex software the source code may consist of many files and may be combined in different ways to produce many different versions.

Build tools

The process of building a computer program is usually managed by a build tool, a program that coordinates and controls other programs. Examples of such a program are make, Gradle, Meister by OpenMake Software, Ant, Maven, Rake, SCons and Phing (http://www.phing.info). The build utility typically needs to compile the various files, in the correct order. If the source code in a particular file has not changed then it may not need to be recompiled (may not rather than need not because it may itself depend on other files that have changed). Sophisticated build utilities and linkers attempt to refrain from recompiling code that does not need it, to shorten the time required to complete the build. A more complex process may involve other programs producing code or data as part of the build process.

See also

- Build automation
- List of build automation software
- Software versioning

References

1. Lee, Kevin.A (1996). *The Buildmeister's Guide - Achieving Agile Software Delivery*. Lulu.com. p. 21. ISBN 978-1847283733.
2. "SonarQube™ software" (http://www.sonarqube.org/). Retrieved 4 January 2014.
3. Muschko, Benjamin. *Gradle in Action*. Manning Pubns Co. ISBN 9781617291302.