Continuous integration

End of the big bang integration era

Patrick Laurent
Partner
Technology & Enterprise Applications
Deloitte

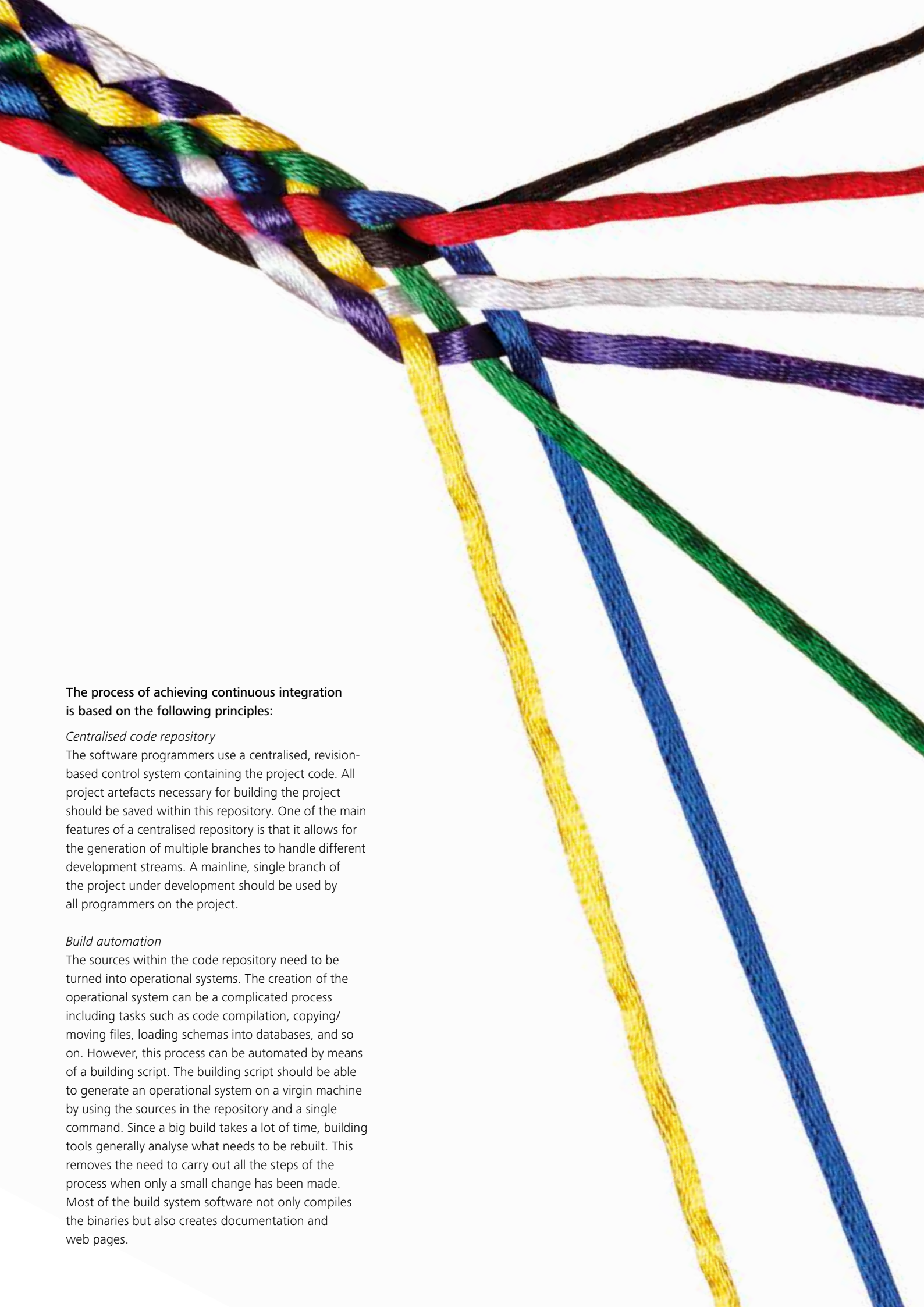
Mario Deserranno
Manager
Technology & Enterprise Applications
Deloitte

The integration of software systems is often a long and painful process that is in most cases postponed until the different components have been built and delivered separately. This does not necessarily have to be the case. Integration can also be performed in an iterative way as from the start of the project, rather than via 'big bang integration' where software modules are developed in isolation and integrated at the end of the project.

What is continuous integration?

Continuous integration is a software development practice intended to reach a high level of integrated and continuous development activity. Its goal is to reduce the scope of integration problems. The term 'continuous integration' comes from the Extreme Programing development process, where it was one of 12 original practices. The practice was intended for the automation of unit testing, verifying that all unit tests have been passed before committing to the mainline. Later expansions of the practice included the use of build servers to automate the code build process each time a developer checked in a change before unit testing was launched. By continually applying automated quality control (with each modification of the source code), organisations aim to improve the quality of the software and its delivery time.

While the concept of continuous integration has been around for several years, experts in the software industry believe it will become increasingly vital in software development in the current business environment. The cost of computing is continually declining, but human resources costs remain more or less constant. Continuous integration intends to shift the effort from developers to software to reduce overall costs.



The process of achieving continuous integration is based on the following principles:

Centralised code repository
The software programmers use a centralised, revision-based control system containing the project code. All project artefacts necessary for building the project should be saved within this repository. One of the main features of a centralised repository is that it allows for the generation of multiple branches to handle different development streams. A mainline, single branch of the project under development should be used by all programmers on the project.

Build automation
The sources within the code repository need to be turned into operational systems. The creation of the operational system can be a complicated process including tasks such as code compilation, copying/moving files, loading schemas into databases, and so on. However, this process can be automated by means of a building script. The building script should be able to generate an operational system on a virgin machine by using the sources in the repository and a single command. Since a big build takes a lot of time, building tools generally analyse what needs to be rebuilt. This removes the need to carry out all the steps of the process when only a small change has been made. Most of the build system software not only compiles the binaries but also creates documentation and web pages.

While the concept of continuous integration has been around for several years, experts in the software industry believe it will become increasingly vital in software development in the current business environment

Making the build self-testing
The best way of efficiently identifying bugs faster, is to include automated tests within the build process. These automated tests are executed with the help of self-testing code, where tests are automated within the software. These automated checks must be able to check large parts of the code for bugs. The tests need to be able to be launched with a single command, and it should be self-checking. If the tests performed result in failed test cases, the build should also fail.

This way of working integrates all pieces of software from the various development teams from the very beginning of the project, where they would traditionally be working in isolation until the integration phase.

Benefits of continuous integration

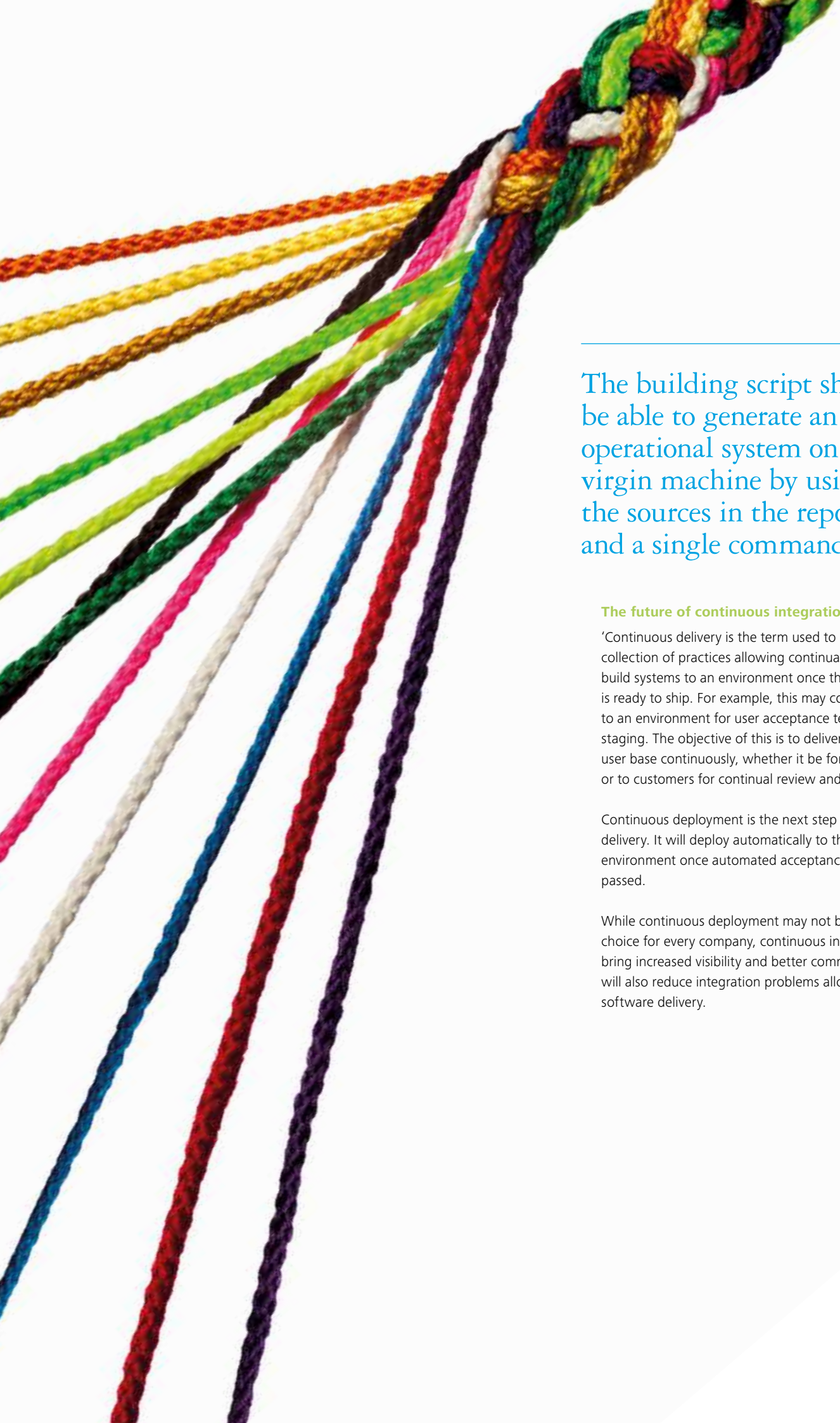
- **Builds are becoming artefacts:** a new build is created each time new code is added to the mainline and passes all tests; not only when the developer manually checks in. This process allows new builds to flow until the project is ready to be delivered
- **Test automation:** automated testing is a good way of checking the quality of a build. It helps developers detect and fix integration problems continuously, avoiding last-minute chaos on release dates (when everyone tries to check in their slightly incompatible versions)
- **Team collaboration:** it results in an improved team dynamic, with the software team, code testers and developers all responsible for writing and executing test cases. Under this system they work together in dynamic teams, rather than against each other
- **Automated test reporting:** automated reports generated by the continuous integration tool will lead to better communication of the testing progress
- **Use of version control becomes standard:** as continuous integration servers rely on version control to build and run tests, development teams now have access to a version control tool which was previously only available to large companies and teams
- **KPIs:** KPIs are generated from automated testing (e.g. code coverage, code complexity and number of features complete), helping developers focus on the delivery of quality code and helping develop the momentum in a team

A continuously evolving trend

Since its original introduction, there have been some improvements in the underlying technologies of continuous integration, especially in the field of branching and merging and within distributed development. As it stands, each branch is tested before it is merged back into the mainline, reducing the risk of having broken builds. Developers now also make use of 'feature' branches where all feature development should take place in a dedicated branch instead of the master branch. This separation allows multiple developers to work on a particular feature without disturbing the main codebase. This means that the master branch will never contain broken code, which is a huge advantage for continuous integration environments.

Distributed version control is a peer-to-peer approach to traditional source control management allowing developers to run their own repositories locally, without any connection to the central server. This will speed up the development process as developers will no longer have problems with slow connections.

Changes to integration servers and their underlying version control system mean that continuous integration is continuing to progress quickly, providing further improvements to release frequency and project stability.



The building script should be able to generate an operational system on a virgin machine by using the sources in the repository and a single command

The future of continuous integration

'Continuous delivery is the term used to describe the collection of practices allowing continual delivery of build systems to an environment once the development is ready to ship. For example, this may cover a delivery to an environment for user acceptance tests or for staging. The objective of this is to deliver builds to a user base continuously, whether it be for QA purposes or to customers for continual review and inspection.

Continuous deployment is the next step of continuous delivery. It will deploy automatically to the production environment once automated acceptance tests are passed.

While continuous deployment may not be the right choice for every company, continuous integration will bring increased visibility and better communication, and will also reduce integration problems allowing faster software delivery.

Typical Continuous Integration process