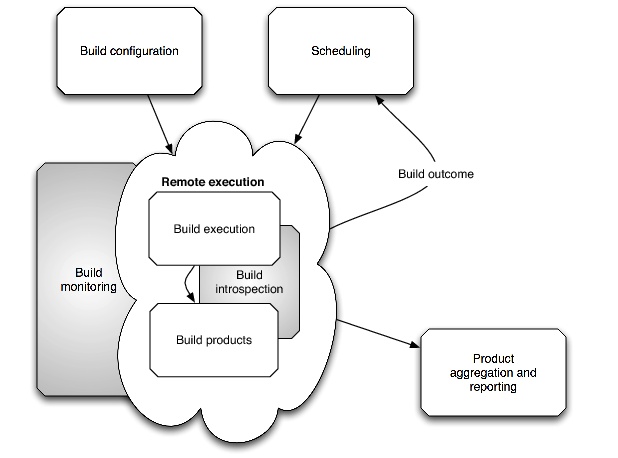
1. Continuous Integration (CI) systems are systems that build and test software automatically and regularly
2. avoiding long periods between build and test runs
3. simplify and automate the execution of many otherwise tedious tasks
4. Continuous integration is a timely subject, not least
5. The first, Buildbot, is a master/slave system; the second, CDash is a reporting [server](http://www.aosabook.org/en/integration.html) model; the third Jenkins, uses a hybrid model; and the fourth, Pony-Build, is a Python-based decentralized reporting server that we will use as a foil for further discussion
6. ***What Does Continuous Integration Software Do?*** The core functionality of a continuous integration system is simple: build software, run tests, and report the results.
7. in UNIX, for example, this entire process can be implemented for most Python packages in a seven line script ***(***do a build, and then run the tests***):***





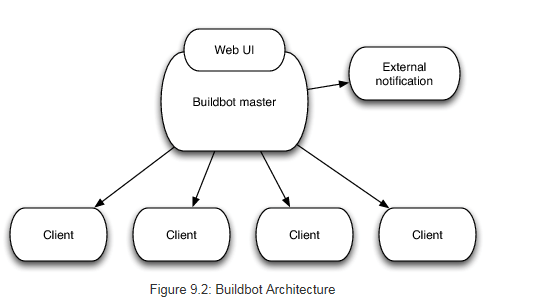
1. ***External Interactions***

Continuous integration systems also need to interact with other systems. There are several types of potential interactions:

* *Build notification*: The outcomes of builds generally need to be communicated to interested clients, either via pull (Web, RSS, RPC, etc.) or push notification (e-mail, Twitter, PubSubHubbub, etc.) This can include notification of all builds, or only failed builds, or builds that haven't been executed within a certain period.
* *Build information*: Build details and products may need to be retrieved, usually via RPC or a bulk [download[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://www.aosabook.org/en/integration.html)](http://www.aosabook.org/en/integration.html) system. For example, it may be desirable to have an external analysis system do an in-depth or more targeted analysis, or report on code coverage or performance results. In addition, an external test result repository may be employed to keep track of failing and successful tests separately from the CI system.
* *Build requests*: External build requests from users or a code repository may need to be handled. Most VCSs have post-commit hooks that can execute an RPC call to initiate a build, for example. Or, users may request builds manually through a Web interface or other user-initiated RPC.
* *Remote control of the CI system*: More generally, the entire runtime may be modifiable through a more-or-less well-defined RPC interface. Either ad hoc extensions or a more formally specified interface may need to be able to drive builds on specific platforms, specify alternate source branches in order to build with various patches, and execute additional builds conditionally. This is useful in support of more general workflow systems, e.g. to permit commits only after they have passed the full set of CI tests, or to test patches across a wide variety of systems before final integration. Because of the variety of bug tracker, patch systems, and other external systems in use, it may not make sense to include this logic within the CI system itself.

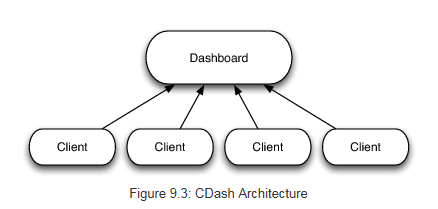
1. Buildbot and CDash have chosen opposite architectures
2. ***Implementation Model: Buildbot***

* Buildbot uses a master/slave architecture, with a single central server and multiple build slaves
* the master configuration specifies the command to be executed on each remote system, and runs them when each previous command is finished



1. ***Implementation Model: Cdash***

* In contrast to Buildbot, CDash implements a reporting server model
* In this model, the CDash server acts as a central repository for information on remotely executed builds
* Builds run on remote clients on their own schedule, and submit build reports in an XML format



1. ***Implementation Model: Jenkins***

* Jenkins operates in a hybrid mode
* Defaulting to master-server build execution but allowing a variety of methods for executing remote builds, including both server- and client-initiated builds. Like Buildbot
* **it is primarily designed for central server control, but has been adapted to support a wide variety of distributed job initiation mechanisms, including virtual machine management**.

1. ***Implementation Model: Pony-Build***

* Pony-Build is a proof-of-concept decentralized CI system written in Python
* It is composed of three core components

Clients:

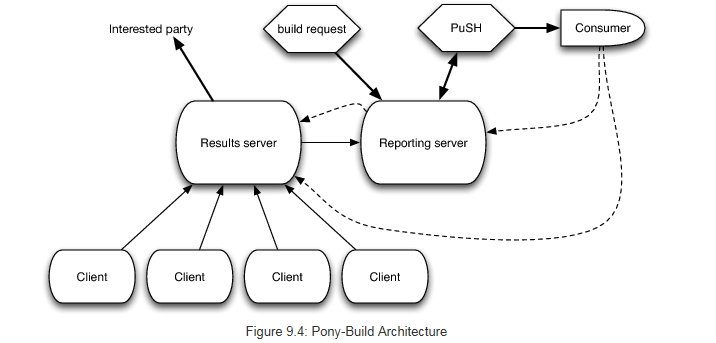
The clients independently contain all configuration information and build context

Result Server:

The results server acts as a centralized database containing build results received from individual clients and coupled with a lightweight client-side library to help with VCS repository access, build process management, and the communication of results to the server

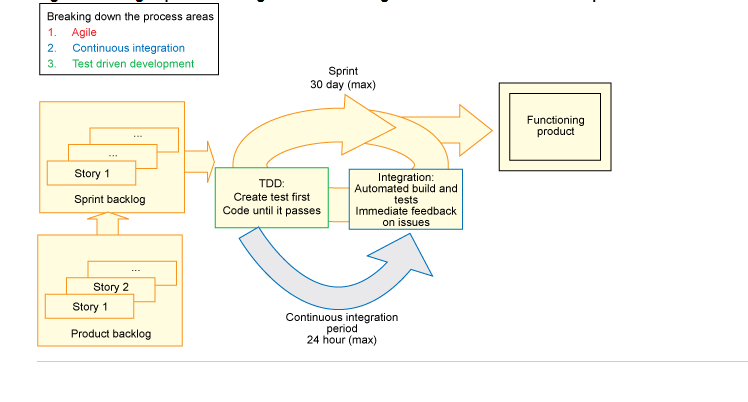
Reporting Server:

The reporting server is optional, and contains a simple Web interface, both for reporting on the results of builds and potentially for requesting new builds



**IBM best practice in agile**

1. Continuous integration in agile development Continuous integration in agile development Continuous integration in agile developmentHow continuous integration and test-driven development fit into agile practice



1. nmbmnbTypes of projects that benefit from continuous integration
   * + - Size Type:

Teams of fewer than 50 people working on less complex projects

* + - * Embedded systems development:

Today, a new car is marketed less by its horsepower and more by its embedded software technology (self-park, advanced safety warnings, fuel efficiency, infotainment system, for example)

The number of lines of code written to create a new car is higher than the number of lines of code written for an F16 fighter jet.

1. Some of usage in IBM aspect:
   * + - Architecture
       - ***Simulation***
       - Build automation
       - ***Work management***
       - Quality management
       - ***Automated testing***
       - Collaboration
2. **Advantages of CI and TDD in practice in IBM aspect**

* From a technical perspective, CI helps teams work more efficiently. These teams can be cross-functional, creating hardware and software that works together.
* They can be geographically distributed, because the constant integration work will ensure that you don't get deviating designs.
* People can work on a large team, because the different components of a complex system will more assuredly work together.
* It solves many of the early pitfalls that these nontraditional agile teams might have experienced without CI.
* Combining CI with test-driven development puts more people under the agile umbrella, because it allows agile methods to work more efficiently.
* From a business perspective, CI offers better business results by allowing teams to have their cake and eat it too.
* That is, they can bring products to market faster, by finding issues when they are young and small, not waiting until they are large and more difficult to fix.
* They can also respond better to requirements that are introduced while the product is being development. This creates a better product for the customer, which is the real promise of agility.