Prof. Dr. Klaus Quibeldey-Cirkel

MOOC Web Engineering

Part III: Web Application Quality Assurance

MOOC Web Engineering

Part III: Web Application Quality Assurance

Contents

Articles

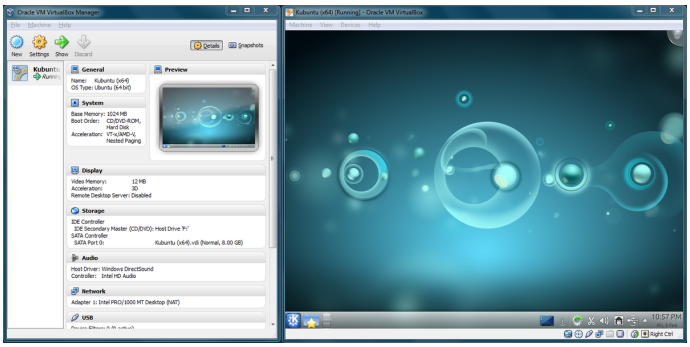
Chapter 21: Starter Kit: Virtualised Development Environments with Vagrant and Puppet

VirtualBox

"Virtual box" redirects here. For virtual computers in general, see virtual machine.

VirtualBox

Logo of Oracle/Sun/innotek VirtualBox since 2010.



Running Kubuntu Live CD with Oracle VM VirtualBox on Windows 7

Original author(s)	innotek GmbH
Developer(s)	Oracle Corporation
Initial release	15 January 2007
Stable release	4.3.12 (16 May 2014) [±] [1]
Written in	C, C++
Operating system	Microsoft Windows, Mac OS X, Linux and Solaris
Size	86–115 MB depending on platform
Type	Virtual machine
License	Base Package: GNU General Public License version 2 (Optionally CDDL for most files of the source distribution), "Extension Pack": PUEL [2]
Website	www.virtualbox.org [3]

Oracle VM VirtualBox (formerly **Sun VirtualBox**, **Sun xVM VirtualBox** and **innotek VirtualBox**) is a virtualization software package for x86 and AMD64/Intel64-based computers from Oracle Corporation as part of its family of virtualization products. It was created by innotek GmbH, purchased in 2008 by Sun Microsystems, and now developed by Oracle. It is installed on an existing host operating system as an application; this host application allows additional guest operating systems, each known as a *Guest OS*, to be loaded and run, each with its own virtual

environment.

Supported host operating systems include Linux, Mac OS X, Windows XP, Windows Vista, Windows 7, Windows 8, Solaris, and OpenSolaris; there are also ports to FreeBSD and Genode. Supported guest operating systems include versions and derivations of Windows, Linux, BSD, OS/2, Solaris, Haiku and others. Since release 3.2.0, VirtualBox also allows limited virtualization of Mac OS X guests on Apple hardware, though OSX86 can also be installed using VirtualBox.

Since version 4.3, Windows guests on supported hardware can take advantage of the recently implemented WDDM driver included in the guest additions; this allows Windows Aero to be enabled along with Direct3D support.

History

Logo of VirtualBox OSE, 2007-2010

VirtualBox was initially offered by the innotek GmbH under a proprietary software license, making one version of the product available at no cost for personal or evaluation use, subject to the VirtualBox Personal Use and Evaluation License (PUEL). In January 2007, based on counsel by LiSoG, innotek GmbH released VirtualBox Open Source Edition (OSE) as free and open-source software, subject to the requirements of the GNU General Public License (GPL), version 2.

The innotek GmbH also contributed to the development of OS/2 and Linux support in virtualization and OS/2 ports of products from Connectix which were later acquired by Microsoft. Specifically, innotek developed the “additions” code in both Microsoft Virtual PC and Microsoft Virtual Server, which enables various host-guest OS interactions like shared clipboards or dynamic viewport resizing.

Sun Microsystems acquired innotek in February 2008.

Oracle Corporation acquired Sun in January 2010 and re-branded the product as "Oracle VM VirtualBox".

Licensing

With version 4 of VirtualBox, released in December 2010, the core package is free software released under GNU General Public License version 2 (GPLv2). This is the fully featured package, excluding some proprietary components not available under GPLv2. These components provide support for USB 2.0 devices, Remote Desktop Protocol (RDP) and Preboot Execution Environment (PXE) for Intel cards and are released as a separate "VirtualBox Oracle VM VirtualBox extension pack" under a proprietary Personal Use and Evaluation License (PUEL), which permits use of the software for personal use, educational use, or evaluation, free of charge.

Oracle defines personal use as any situation in which one person installs the software, and only that individual, and their friends and family, use the software. Oracle does not care if that use is for commercial or non-commercial purposes. Oracle would consider it non-personal use, for example, if a network administrator installed many copies of the software on many different machines, on behalf of many different end-users. That type of situation would require purchasing a special volume license.

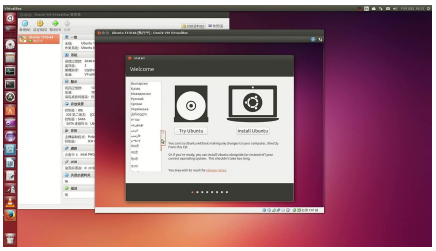
Prior to version 4, there were two different packages of the VirtualBox software. The full package was offered free under the PUEL, with licenses for other commercial deployment purchasable from Oracle. A second package called the *VirtualBox Open Source Edition (OSE)* was released under GPLv2. This removed the same proprietary components not available under GPLv2.

Virtualbox requires the use of the Open Watcom compiler to build the BIOS since version 4.2.

Although VirtualBox has experimental support for Mac OS X guests, the end user license agreement of Mac OS X does not permit the operating system to run on non-Apple hardware, enforced within the operating system by calls to the Apple System Management Controller (SMC) in all Apple machines, which verifies the authenticity of the hardware.

Emulated environment

Users of VirtualBox can load multiple guest OSs under a single host operating-system (host OS). Each guest can be started, paused and stopped independently within its own virtual machine (VM). The user can independently configure each VM and run it under a choice of software-based virtualization or hardware assisted virtualization if the underlying host hardware supports this. The host OS and guest OSs and applications can communicate with each other through a number of mechanisms including a common clipboard and a virtualized network facility. Guest VMs can also directly communicate with each other if configured to do so.



Running Ubuntu Live CD with Oracle VM VirtualBox on Ubuntu

Software-based virtualization

In the absence of hardware-assisted virtualization, VirtualBox adopts a standard software-based virtualization approach. This mode supports 32-bit guest OSs which run in rings 0 and 3 of the Intel ring architecture.

- The system reconfigures the guest OS code, which would normally run in ring 0, to execute in ring 1 on the host hardware. Because this code contains many privileged instructions which cannot run natively in ring 1, VirtualBox employs a Code Scanning and Analysis Manager (CSAM) to scan the ring 0 code recursively before its first execution to identify problematic instructions and then calls the Patch Manager (PATM) to perform *in-situ* patching. This replaces the instruction with a jump to a VM-safe equivalent compiled code fragment in hypervisor memory.
- The guest user-mode code, running in ring 3, generally runs directly on the host hardware in ring 3.

In both cases, VirtualBox uses CSAM and PATM to inspect and patch the offending instructions whenever a fault occurs. VirtualBox also contains a dynamic recompiler, based on QEMU to recompile any real mode or protected mode code entirely (e.g. BIOS code, a DOS guest, or any operating system startup).

Using these techniques, VirtualBox can achieve a performance comparable to that of VMware.

Hardware-assisted virtualization

VirtualBox supports both Intel's VT-x and AMD's AMD-V hardware-virtualization. Making use of these facilities, VirtualBox can run each guest VM in its own separate address-space; the guest OS ring 0 code runs on the host at ring 0 in VMX non-root mode rather than in ring 1.

VirtualBox supports some guests (including 64-bit guests, SMP guests and certain proprietary OSs) only on hosts with hardware-assisted virtualization.

Device virtualization

The system emulates hard disks in one of three disk image formats:

1. a VirtualBox-specific container format, called "Virtual Disk Image" (VDI), storing files (with a .vdi suffix) on the host operating system
2. VMware Virtual Machine Disk Format (VMDK)
3. Microsoft Virtual PC VHD format

A VirtualBox virtual machine can, therefore, use disks previously created in VMware or Microsoft Virtual PC, as well as its own native format. VirtualBox can also connect to iSCSI targets and to raw partitions on the host, using either as virtual hard disks. VirtualBox emulates IDE (PIIX4 and ICH6 controllers), SCSI, SATA (ICH8M controller) and SAS controllers to which hard drives can be attached.

VirtualBox has supported Open Virtualization Format (OVF) since version 2.2.0 (April 2009).

Both ISO images and host-connected physical devices can be mounted as CD/DVD drives. For example, the DVD image of a Linux distribution can be downloaded and used directly by VirtualBox.

By default VirtualBox provides graphics support through a custom virtual graphics-card that is VESA compatible. The Guest Additions for Windows, Linux, Solaris, OpenSolaris, or OS/2 guests include a special video-driver that increases video performance and includes additional features, such as automatically adjusting the guest resolution when resizing the VM window or desktop composition via virtualized WDDM drivers .

For an Ethernet network adapter, VirtualBox virtualizes these Network Interface Cards:

- AMD PCnet PCI II (Am79C970A)
- AMD PCnet-Fast III (Am79C973)
- Intel Pro/1000 MT Desktop (82540EM)
- Intel Pro/1000 MT Server (82545EM)
- Intel Pro/1000 T Server (82543GC)

The emulated network cards allow most guest OSs to run without the need to find and install drivers for networking hardware as they are shipped as part of the guest OS. A special paravirtualized network adapter is also available, which improves network performance by eliminating the need to match a specific hardware interface, but requires special driver support in the guest. (Many distributions of Linux ship with this driver included.) By default, VirtualBox uses NAT through which Internet software for end-users such as Firefox or ssh can operate. Bridged networking via a host network adapter or virtual networks between guests can also be configured. Up to 36 network adapters can be attached simultaneously, but only four are configurable through the graphical interface.

For a sound card, VirtualBox virtualizes Intel HD Audio, Intel ICH AC'97 and SoundBlaster 16 devices.

A USB 1.1 controller is emulated so that any USB devices attached to the host can be seen in the guest. The closed-source extension pack adds a USB 2.0 controller and, if VirtualBox acts as an RDP server, it can also use USB devices on the remote RDP client as if they were connected to the host, although only if the client supports this VirtualBox-specific extension (Oracle provides clients for Solaris, Linux and Sun Ray thin clients that can do this, and have promised support for other platforms in future versions).

Feature set

- 64-bit guests (hardware virtualization support is required)
- Snapshots
- Seamless mode - the ability to run virtualized applications side by side with your normal desktop applications
- Shared clipboard
- Shared folders
- Special drivers and utilities to facilitate switching between systems
- Command line interaction (in addition to the GUI)
- Public API (Java, Python, SOAP, XPCOM) to control VM configuration and execution
- Nested paging for AMD-V and Intel VT (only for processors supporting SLAT and with SLAT enabled)
- Limited support for 3D graphics acceleration (including OpenGL up to (but not including) 3.0 and Direct3D 9.0c via Wine's Direct3D to OpenGL translation)
- SMP support (up to 32 virtual CPUs per virtual machine), since version 3.0
- Teleportation (aka Live Migration)
- 2D video output acceleration (not to be mistaken with video decoding acceleration), since version 3.1

Storage emulation features

- NCQ support for SATA, SCSI and SAS raw disks and partitions
- Pass-through mode for solid-state drives

- Pass-through mode for CD/DVD/BD disks - allows to play audio CDs, burn optical disks, play encrypted DVD disks
- Can disable host OS I/O cache
- Allows to limit IO bandwidth
- PATA, SATA, SCSI, SAS, iSCSI, floppy disk controllers

Storage support

- Raw hard disk access – allows physical hard disk partitions on the host system to appear in the guest system
- VMware Virtual Machine Disk (VMDK) format support – allows VirtualBox to exchange disk images with VMware
- Microsoft VHD support
- QEMU qed and qcow disks
- HDD format disks (only version 2; version 3 and 4 are not supported) used by Parallels virtualization products

Since version 3.2

- Mac OS X Server guest support – experimental
- Memory ballooning (not available on Solaris hosts)
- RAM deduplication (Page Fusion) for Windows guests on 64-bit hosts
- CPU hot-plugging for Linux (hot-add and hot-remove) and certain Windows guests (hot-add only)
- Deleting snapshots while the VM is running
- Multi-monitor guest setups in the GUI, for Windows guests
- LSI Logic SAS controller emulation
- Remote Desktop Protocol (RDP) video acceleration
- Run and control guest applications from the host – for automated software deployments

Since version 4.0

- The PUEL/OSE separation was given up in favor of an open source base product and a closed source extension pack that can be installed on top of the base product. As part of this change, additional components of VirtualBox were made open source (installers, documentation, device drivers)
- Intel HD audio codec emulation
- Intel ICH9 chipset emulation
- A new VM storage scheme where all VM data is stored in one single folder to improve VM portability
- Several UI enhancements including a new look with VM preview and scale mode
- On 32-bit hosts, VMs can each use more than 1.5 GB of RAM
- In addition to OVF, the single file OVA format is supported
- CPU use and I/O bandwidth can be limited per VM
- Support for Apple DMG images (DVD)
- Multi-monitor guest setups for Linux/Solaris guests (previously Windows only)
- Resizing of disk image formats from Oracle, VDI (VirtualBox disk image), and Microsoft, VHD (Virtual PC hard disk)

Since version 4.1

- Windows Aero support (experimental)
- Virtual machine cloning

Since version 4.2

- Virtual machine groups - allows you to manage a group of virtual machines as a single unit (power them on or off, snapshot them, etc.)
- Some VM settings can be altered during a VM execution
- Support up to 36 NICs in case of the ICH9 chipset

- Support for limiting network IO bandwidth
- Can automatically run VMs on a host system startup (except on Windows host)

Since version 4.3

- VM video capturing support
- Host touch devices support (GUI passes host touch-events to guest)/USB virtualization of such devices

The extension pack

Some features require the installation of the closed-source "VirtualBox Extension Pack":

- Support for a virtual USB 2.0 controller (EHCI)
- VirtualBox RDP: support for proprietary remote connection protocol developed by Microsoft and Citrix.
- PXE boot for Intel cards

References

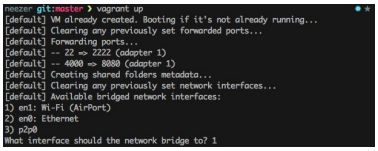
[1] http://en.wikipedia.org/w/index.php?title=Template:Latest_stable_software_release/VirtualBox&action=edit
[2] https://www.virtualbox.org/wiki/VirtualBox_PUEL
[3] https://www.virtualbox.org/

External links

- Official website (https://www.virtualbox.org/)
- Sub-site at Oracle (http://www.oracle.com/us/technologies/virtualization/oraclevm/061976.html)
- Virtualbox Downloads (https://www.virtualbox.org/wiki/Downloads)
- All downloads index (not to be removed). E.g. for guest additions (http://download.virtualbox.org/virtualbox)

Vagrant (software)

Vagrant



Vagrant starting a virtual machine using 'vagrant up'

Original author(s)	Mitchell Hashimoto
Developer(s)	Mitchell Hashimoto and John Bender
Stable release	1.6.1 (May 6, 2014) [±] [1]
Development status	Active
Written in	Ruby
Operating system	Microsoft Windows Mac OS X Linux
Available in	English
License	MIT License
Website	vagrantup.com [2]

Vagrant is free and open-source software for creating and configuring virtual development environments. It can be considered a wrapper around virtualization software such as VirtualBox and configuration management software such as Chef, Salt and Puppet.

Since version 1.1, Vagrant is no longer tied to VirtualBox and also works with other virtualization software such as VMware, and supports server environments like Amazon EC2. Although written in Ruby, it is usable in projects written in other programming languages such as PHP, Python, Java, C# and JavaScript.

There is a plugin called vagrant-libvirt, which adds support for libvirt to Vagrant.

References

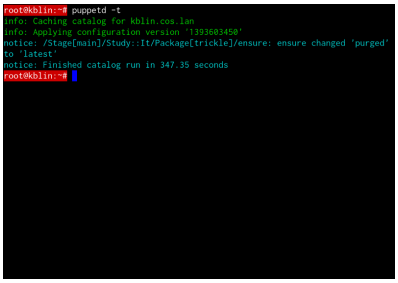
[1] http://en.wikipedia.org/w/index.php?title=Template:Latest_stable_software_release/Vagrant_(software)&action=edit
[2] http://vagrantup.com/

External links

- Official website (http://www.vagrantup.com/)
- List of Vagrant boxes (http://www.vagrantbox.es/)

Puppet (software)

Puppet



Puppet manually invoked on a client

Developer(s)	Puppet Labs
Initial release	2005
Stable release	3.5.1 (April 16, 2014) [±] [1]
Preview release	3.2.1-rc1 (May 17, 2013) [±] [2]
Written in	Ruby
Operating system	GNU/Linux, Unix-like, Windows
Type	Configuration management
License	Apache for >2.7.0; GPL for prior versions.
Website	http://www.puppetlabs.com/

Puppet is an open source configuration management tool. It is multi-platform in that it works on many Unix-like systems as well as on Microsoft Windows, and includes its own declarative language to describe system configuration.

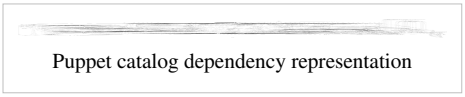
Origin

Puppet is produced by Puppet Labs, founded by Luke Kanies in 2005. It is written in Ruby and released as free software under the GPL until version 2.7.0 and the Apache 2.0 license after that.

Purpose

Puppet is a tool designed to manage the configuration of Unix-like and Microsoft Windows systems declaratively. The user describes system resources and their state, either using Puppet's declarative language or a Ruby DSL (domain-specific language). This information is stored in files called "Puppet manifests". Puppet discovers the system information via a utility called Facter, and compiles the Puppet manifests into a system-specific catalog containing resources and resource dependency, which are applied against the target systems. Any actions taken by Puppet are then reported.

Puppet language



Puppet catalog dependency representation

Puppet consists of a custom declarative language to describe system configuration, which can be either applied directly on the system, or compiled into a catalog and distributed to the target system via client–server paradigm (using a REST API), and the agent uses system specific providers to enforce the resource specified in the manifests. The resource abstraction layer enables administrators to describe the configuration in high-level terms, such as users, services and packages without the need to specify OS specific commands (such as rpm, yum, apt).

Platform support

Built to be cross-platform, it works on Linux distributions, including CentOS, Debian, Fedora, Mandriva, Oracle Linux, RHEL, Scientific Linux, SUSE and Ubuntu, as well as multiple Unix systems (Solaris, BSD, Mac OS X, AIX, HP-UX), and has Microsoft Windows support.[3][4]

It is model-driven, requiring limited programming knowledge to use.[5]

Users

Puppet is used by the Wikimedia Foundation,[6] ARIN, Mozilla, Reddit,[7] Dell, Rackspace, Zynga, Twitter, the New York Stock Exchange, PayPal, Disney, Citrix Systems, Oracle, Yandex, the University of North Texas, the Los Alamos National Laboratory, Stanford University, Lexmark and Google, among others.[8]

References

[1] http://en.wikipedia.org/w/index.php?title=Template:Latest_stable_software_release/Puppet_(software)&action=edit

[2] http://en.wikipedia.org/w/index.php?title=Template:Latest_preview_software_release/Puppet_(software)&action=edit

[3] System Requirements (http://docs.puppetlabs.com/pe/2.0/install_system_requirements.html)

[4] Supported Platforms (http://docs.puppetlabs.com/guides/platforms.html)

[5] Deploying Tomcat Applications With Puppet (http://www.tomcatexpert.com/blog/2010/04/29/deploying-tomcat-applications-puppet)

[6] Ever wondered how the Wikimedia servers are configured? — Wikimedia blog (http://blog.wikimedia.org/2011/09/19/ever-wondered-how-the-wikimedia-servers-are-configured/). Blog.wikimedia.org (2011-09-19). Retrieved on 2013-12-09.

[7] We are sysadmins @ reddit. Ask us anything! : sysadmin (http://www.reddit.com/r/sysadmin/comments/r6zfv/we_are_sysadmins_reddit_ask_us_anything/). Reddit.com (2012-03-21). Retrieved on 2013-12-09.

[8] Google, VMware, Cisco stuff Puppet with $8.5M (http://www.theregister.co.uk/2011/11/30/puppet_labs_google_vmware_funding/)

External links

- Puppet Project Home (http://projects.puppetlabs.com/projects/puppet)
- Official Puppet Labs YouTube Channel (http://www.youtube.com/user/PuppetLabsInc)
- Pulling Strings with Puppet: Configuration Management Made Easy (http://www.apress.com/9781590599785) (ISBN 978-1590599780)
- Pro Puppet (http://www.apress.com/9781430230571) (ISBN 978-1430230576)
- https://github.com/puppetlabs

Chapter 22: Automated Performance Testing with Apache JMeter

Software performance testing

In software engineering, **performance testing** is in general testing performed to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

Performance testing is a subset of performance engineering, an emerging computer science practice which strives to build performance into the implementation, design and architecture of a system.

Testing types

Load testing

Load testing is the simplest form of performance testing. A load test is usually conducted to understand the behaviour of the system under a specific expected load. This load can be the expected concurrent number of users on the application performing a specific number of transactions within the set duration. This test will give out the response times of all the important business critical transactions. If the database, application server, etc. are also monitored, then this simple test can itself point towards bottlenecks in the application software.

Stress testing

Stress testing is normally used to understand the upper limits of capacity within the system. This kind of test is done to determine the system's robustness in terms of extreme load and helps application administrators to determine if the system will perform sufficiently if the current load goes well above the expected maximum.

Soak testing

Soak testing, also known as endurance testing, is usually done to determine if the system can sustain the continuous expected load. During soak tests, memory utilization is monitored to detect potential leaks. Also important, but often overlooked is performance degradation. That is, to ensure that the throughput and/or response times after some long period of sustained activity are as good or better than at the beginning of the test. It essentially involves applying a significant load to a system for an extended, significant period of time. The goal is to discover how the system behaves under sustained use.

Spike testing

Spike testing is done by suddenly increasing the number of or load generated by users - by a very large amount - and observing the behaviour of the system. The goal is to determine whether performance will suffer, the system will fail, or it will be able to handle dramatic changes in load.

Configuration testing

Rather than testing for performance from the perspective of load, tests are created to determine the effects of configuration changes to the system's components on the system's performance and behaviour. A common example would be experimenting with different methods of load-balancing.

Isolation testing

Isolation testing is not unique to performance testing but involves repeating a test execution that resulted in a system problem. Often used to isolate and confirm the fault domain.

Setting performance goals

Performance testing can serve different purposes.

- It can demonstrate that the system meets performance criteria.
- It can compare two systems to find which performs better.
- Or it can measure what parts of the system or workload causes the system to perform badly.

Many performance tests are undertaken without due consideration to the setting of realistic performance goals. The first question from a business perspective should always be "why are we performance testing?". These considerations are part of the business case of the testing. Performance goals will differ depending on the system's technology and purpose however they should always include some of the following:

Concurrency/throughput

If a system identifies end-users by some form of log-in procedure then a concurrency goal is highly desirable. By definition this is the largest number of concurrent system users that the system is expected to support at any given moment. The work-flow of a scripted transaction may impact true concurrency especially if the iterative part contains the log-in and log-out activity.

If the system has no concept of end-users then performance goal is likely to be based on a maximum throughput or transaction rate. A common example would be casual browsing of a web site such as Wikipedia.

Server response time

This refers to the time taken for one system node to respond to the request of another. A simple example would be a HTTP 'GET' request from browser client to web server. In terms of response time this is what all load testing tools actually measure. It may be relevant to set server response time goals between all nodes of the system.

Render response time

A difficult thing for load testing tools to deal with as they generally have no concept of what happens within a node apart from recognizing a period of time where there is no activity 'on the wire'. To measure render response time it is generally necessary to include functional test scripts as part of the performance test scenario which is a feature not offered by many load testing tools.

Performance specifications

It is critical to detail performance specifications (requirements) and document them in any performance test plan. Ideally, this is done during the requirements development phase of any system development project, prior to any design effort. See Performance Engineering for more details.

However, performance testing is frequently not performed against a specification i.e. no one will have expressed what the maximum acceptable response time for a given population of users should be. Performance testing is frequently used as part of the process of performance profile tuning. The idea is to identify the "weakest link" – there is inevitably a part of the system which, if it is made to respond faster, will result in the overall system running faster. It is sometimes a difficult task to identify which part of the system represents this critical path, and some test tools include (or can have add-ons that provide) instrumentation that runs on the server (agents) and report transaction times, database access times, network overhead, and other server monitors, which can be analyzed together with the raw performance statistics. Without such instrumentation one might have to have someone crouched over Windows Task Manager at the server to see how much CPU load the performance tests are generating (assuming a Windows system is under test).

Performance testing can be performed across the web, and even done in different parts of the country, since it is known that the response times of the internet itself vary regionally. It can also be done in-house, although routers would then need to be configured to introduce the lag that would typically occur on public networks. Loads should be introduced to the system from realistic points. For example, if 50% of a system's user base will be accessing the system via a 56K modem connection and the other half over a T1, then the load injectors (computers that simulate real users) should either inject load over the same mix of connections (ideal) or simulate the network latency of such connections, following the same user profile.

It is always helpful to have a statement of the likely peak number of users that might be expected to use the system at peak times. If there can also be a statement of what constitutes the maximum allowable 95 percentile response time, then an injector configuration could be used to test whether the proposed system met that specification.

Questions to ask

Performance specifications should ask the following questions, at a minimum:

- In detail, what is the performance test scope? What subsystems, interfaces, components, etc. are in and out of scope for this test?
- For the user interfaces (UIs) involved, how many concurrent users are expected for each (specify peak vs. nominal)?
- What does the target system (hardware) look like (specify all server and network appliance configurations)?
- What is the Application Workload Mix of each system component? (for example: 20% log-in, 40% search, 30% item select, 10% checkout).
- What is the System Workload Mix? [Multiple workloads may be simulated in a single performance test] (for example: 30% Workload A, 20% Workload B, 50% Workload C).
- What are the time requirements for any/all back-end batch processes (specify peak vs. nominal)?

Pre-requisites for Performance Testing

A stable build of the system which must resemble the production environment as closely as is possible.

The performance testing environment should be isolated from other environments, such as user acceptance testing (UAT) or development: otherwise the results may not be consistent. As a best practice it is always advisable to have a separate performance testing environment resembling the production environment as much as possible.

Test conditions

In performance testing, it is often crucial (and often difficult to arrange) for the test conditions to be similar to the expected actual use. This is, however, not entirely possible in actual practice. The reason is that the workloads of production systems have a random nature, and while the test workloads do their best to mimic what may happen in the production environment, it is impossible to exactly replicate this workload variability - except in the most simple system.

Loosely-coupled architectural implementations (e.g.: SOA) have created additional complexities with performance testing. Enterprise services or assets (that share a common infrastructure or platform) require coordinated performance testing (with all consumers creating production-like transaction volumes and load on shared infrastructures or platforms) to truly replicate production-like states. Due to the complexity and financial and time requirements around this activity, some organizations now employ tools that can monitor and create production-like conditions (also referred as "noise") in their performance testing environments (PTE) to understand capacity and resource requirements and verify / validate quality attributes.

Timing

It is critical to the cost performance of a new system, that performance test efforts begin at the inception of the development project and extend through to deployment. The later a performance defect is detected, the higher the cost of remediation. This is true in the case of functional testing, but even more so with performance testing, due to the end-to-end nature of its scope. It is crucial for a performance test team to be involved as early as possible because key performance requisites like test environment acquisition and preparation is often a lengthy and time consuming process.

Tools

In the diagnostic case, software engineers use tools such as profilers to measure what parts of a device or software contributes most to the poor performance or to establish throughput levels (and thresholds) for maintained acceptable response time.

Technology

Performance testing technology employs one or more PCs or Unix servers to act as injectors – each emulating the presence of numbers of users and each running an automated sequence of interactions (recorded as a script, or as a series of scripts to emulate different types of user interaction) with the host whose performance is being tested. Usually, a separate PC acts as a test conductor, coordinating and gathering metrics from each of the injectors and collating performance data for reporting purposes. The usual sequence is to ramp up the load – starting with a small number of virtual users and increasing the number over a period to some maximum. The test result shows how the performance varies with the load, given as number of users vs response time. Various tools are available to perform such tests. Tools in this category usually execute a suite of tests which emulate real users against the system. Sometimes the results can reveal oddities, e.g., that while the average response time might be acceptable, there are outliers of a few key transactions that take considerably longer to complete – something that might be caused by inefficient database queries, pictures, etc.

Performance testing can be combined with stress testing, in order to see what happens when an acceptable load is exceeded –does the system crash? How long does it take to recover if a large load is reduced? Does it fail in a way that causes collateral damage?

Analytical Performance Modeling is a method to model the behaviour of a system in a spreadsheet. The model is fed with measurements of transaction resource demands (CPU, disk I/O, LAN, WAN), weighted by the transaction-mix (business transactions per hour). The weighted transaction resource demands are added up to obtain the hourly resource demands and divided by the hourly resource capacity to obtain the resource loads. Using the responsetime formula (R=S/(1-U), R=responsetime, S=servicetime, U=load), responsetimes can be calculated and calibrated with the results of the performance tests. Analytical performance modeling allows evaluation of design options and system sizing based on actual or anticipated business usage. It is therefore much faster and cheaper than performance testing, though it requires thorough understanding of the hardware platforms.

Tasks to undertake

Tasks to perform such a test would include:

- Decide whether to use internal or external resources to perform the tests, depending on inhouse expertise (or lack thereof)
- Gather or elicit performance requirements (specifications) from users and/or business analysts
- Develop a high-level plan (or project charter), including requirements, resources, timelines and milestones
- Develop a detailed performance test plan (including detailed scenarios and test cases, workloads, environment info, etc.)
- Choose test tool(s)
- Specify test data needed and charter effort (often overlooked, but often the death of a valid performance test)
- Develop proof-of-concept scripts for each application/component under test, using chosen test tools and strategies
- Develop detailed performance test project plan, including all dependencies and associated time-lines
- Install and configure injectors/controller
- Configure the test environment (ideally identical hardware to the production platform), router configuration, quiet network (we don't want results upset by other users), deployment of server instrumentation, database test sets developed, etc.
- Execute tests – probably repeatedly (iteratively) in order to see whether any unaccounted for factor might affect the results
- Analyze the results - either pass/fail, or investigation of critical path and recommendation of corrective action

Methodology

Performance testing web applications

According to the Microsoft Developer Network the Performance Testing Methodology [1] consists of the following activities:

- **Activity 1. Identify the Test Environment.** Identify the physical test environment and the production environment as well as the tools and resources available to the test team. The physical environment includes hardware, software, and network configurations. Having a thorough understanding of the entire test environment at the outset enables more efficient test design and planning and helps you identify testing challenges early in the project. In some situations, this process must be revisited periodically throughout the project's life cycle.
- **Activity 2. Identify Performance Acceptance Criteria.** Identify the response time, throughput, and resource utilization goals and constraints. In general, response time is a user concern, throughput is a business concern, and resource utilization is a system concern. Additionally, identify project success criteria that may not be captured by those goals and constraints; for example, using performance tests to evaluate what combination of configuration

settings will result in the most desirable performance characteristics.
- **Activity 3. Plan and Design Tests.** Identify key scenarios, determine variability among representative users and how to simulate that variability, define test data, and establish metrics to be collected. Consolidate this information into one or more models of system usage to be implemented, executed, and analyzed.
- **Activity 4. Configure the Test Environment.** Prepare the test environment, tools, and resources necessary to execute each strategy as features and components become available for test. Ensure that the test environment is instrumented for resource monitoring as necessary.
- **Activity 5. Implement the Test Design.** Develop the performance tests in accordance with the test design.
- **Activity 6. Execute the Test.** Run and monitor your tests. Validate the tests, test data, and results collection. Execute validated tests for analysis while monitoring the test and the test environment.
- **Activity 7. Analyze Results, Tune, and Retest.** Analyse, Consolidate and share results data. Make a tuning change and retest. Improvement or degradation? Each improvement made will return smaller improvement than the previous improvement. When do you stop? When you reach a CPU bottleneck, the choices then are either improve the code or add more CPU.

External links

- The Art of Application Performance Testing - O'Reilly ISBN 978-0-596-52066-3 [2] (Book)
- Performance Testing Guidance for Web Applications [3] (MSDN)
- Performance Testing Guidance for Web Applications [4] (Book)
- Performance Testing Guidance for Web Applications [5] (PDF)
- Performance Testing Guidance [6] (Online KB)
- Enterprise IT Performance Testing [7] (Online KB)
- Performance Testing Videos [8] (MSDN)
- Open Source Performance Testing tools [9]
- "User Experience, not Metrics" and "Beyond Performance Testing" [10]
- "Performance Testing Traps / Pitfalls" [11]

References

[1] http://msdn2.microsoft.com/en-us/library/bb924376.aspx
[2] http://oreilly.com/catalog/9780596520670
[3] http://msdn2.microsoft.com/en-us/library/bb924375.aspx
[4] http://www.amazon.com/dp/0735625700
[5] http://www.codeplex.com/PerfTestingGuide/Release/ProjectReleases.aspx?ReleaseId=6690
[6] http://www.codeplex.com/PerfTesting
[7] http://www.perftesting.co.uk
[8] http://msdn2.microsoft.com/en-us/library/bb671346.aspx
[9] http://www.opensourcetesting.org/performance.php
[10] http://www.perftestplus.com/pubs.htm
[11] http://www.mercury-consulting-ltd.com/wp/Performance_Testing_Traps.html

Black-box testing

Black-box diagram

Black-box testing is a method of software testing that examines the functionality of an application (e.g. what the software does) without peering into its internal structures or workings (see white-box testing). This method of test can be applied to virtually every level of software testing: unit, integration, system and acceptance. It typically comprises most if not all higher level testing, but can also dominate unit testing as well. Wikipedia:Citation needed

Test procedures

Specific knowledge of the application's code/internal structure and programming knowledge in general is not required. The tester is aware of *what* the software is supposed to do but is not aware of *how* it does it. For instance, the tester is aware that a particular input returns a certain, invariable output but is not aware of *how* the software produces the output in the first place.

Test cases

Test cases are built around specifications and requirements, i.e., what the application is supposed to do. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are primarily *functional* in nature, *non-functional* tests may also be used. The test designer selects both valid and invalid inputs and determines the correct output without any knowledge of the test object's internal structure.

Test design techniques

Typical black-box test design techniques include:

- Decision table testing
- All-pairs testing
- State transition Analysis
- Equivalence partitioning
- Boundary value analysis
- Cause–effect graph
- Error guessing

Hacking

In penetration testing, black-box testing refers to a methodology where an ethical hacker has no knowledge of the system being attacked. The goal of a black-box penetration test is to simulate an external hacking or cyber warfare attack.

References

External links

- BCS SIGIST (British Computer Society Specialist Interest Group in Software Testing): *Standard for Software Component Testing* (http://www.testingstandards.co.uk/Component Testing.pdf), Working Draft 3.4, 27. April 2001.

Functional testing

For functional testing in manufacturing, see Functional Testing (Manufacturing).

Functional testing is a quality assurance (QA) process and a type of black box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered (not like in white-box testing).[1] Functional Testing usually describes *what* the system does.

Functional testing differs from system testing in that functional testing "*verifies* a program by checking it against ... design document(s) or specification(s)", while system testing "*validate[s]* a program by checking it against the published user or system requirements" (Kaner, Falk, Nguyen 1999, p. 52).

Five steps

Functional testing typically involves five stepsWikipedia:Citation needed

1. The identification of functions that the software is expected to perform
2. The creation of input data based on the function's specifications
3. The determination of output based on the function's specifications
4. The execution of the test case
5. The comparison of actual and expected outputs
6. To check whether the application works as per the customer need.

References

[1] Kaner, Falk, Nguyen. *Testing Computer Software*. Wiley Computer Publishing, 1999, p. 42. ISBN 0-471-35846-0.

Smoke testing (software)

See also: Build verification test

In computer programming and software testing, **smoke testing** is preliminary testing to reveal simple failures severe enough to reject a prospective software release. A subset of test cases that cover the most important functionality of a component or system is selected and run, to ascertain if the most crucial functions of a program work correctly.[1] For example, a smoke test may ask basic questions like "Does the program run?", "Does it open a window?", or "Does clicking the main button do anything?" The purpose is to determine whether the application is so badly broken that further testing is unnecessary. As the book "Lessons Learned in Software Testing" [2] puts it, "smoke tests broadly cover product features in a limited time ... if key features don't work or if key bugs haven't yet been fixed, your team won't waste further time installing or testing".[3]

Smoke testing performed on a particular build is also known as a *build verification test*.[4]

A daily build and smoke test is among industry best practices.[5] Smoke testing is also done by testers before accepting a build for further testing. Microsoft claims that after code reviews, "*smoke testing* is the most cost-effective method for identifying and fixing defects in software".[6]

Smoke tests can either be performed manually or using an automated tool. When automated tools are used, the tests are often initiated by the same process that generates the build itself.Wikipedia:Citation needed

Smoke tests can be broadly categorized as functional tests or unit tests. Functional tests exercise the complete program with various inputs. Unit tests exercise individual functions, subroutines, or object methods. Both functional testing tools and unit testing tools tend to be third-party products that are not part of the compiler suite.Wikipedia:Citation needed Functional tests may be a scripted series of program inputs, possibly even with an automated mechanism for controlling mouse movements. Unit tests can be implemented either as separate functions within the code itself, or else as a driver layer that links to the code without altering the code being tested.Wikipedia:Citation needed

References

[1] Dustin, Rashka, Paul. "Automated Software Testing -Introduction, Management, and Performance". Addison-Wesley 1999, p. 43-44. ISBN 0-201-43287-0.

[2] Cem Kaner, James Bach, Bret Pettichord, *Lessons learned in software testing: a context-driven approach*. Wiley, 2001

[3] Kaner, Bach, and Pettichord. "Lessons Learned in Software Testing". Wiley Computer Publishing, 2002, p. 95. ISBN 0-471-08112-4

[4] Author unknown (date unknown). "How to: Configure and Run Build Verification Tests (BVTs)". *MSDN Library for Visual Studio 2005*. Retrieved on 2010-11-20 from http://msdn.microsoft.com/en-us/library/ms182465%28v=VS.80%29.aspx.

[5] McConnell, Steve. "Rapid Development". Microsoft Press, p. 405

[6] Author unknown (date unknown). "Guidelines for Smoke Testing". *MSDN Library for Visual Studio 2005*. Retrieved on 2010-11-20 from http://msdn.microsoft.com/en-us/library/ms182613(VS.80).aspx.

External links

- PC Mag's Definition (http://www.pcmag.com/encyclopedia_term/0,2542,t=smoke+test&i=51556,00.asp)

Load testing

Load testing is the process of putting demand on a system or device and measuring its response. Load testing is performed to determine a system's behavior under both normal and anticipated peak load conditions. It helps to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation. When the load placed on the system is raised beyond normal usage patterns, in order to test the system's response at unusually high or peak loads, it is known as stress testing. The load is usually so great that error conditions are the expected result, although no clear boundary exists when an activity ceases to be a load test and becomes a stress test.

There is little agreement on what the specific goals of load testing are.Wikipedia:Citation needed The term is often used synonymously with concurrency testing, software performance testing, reliability testing, and volume testing. *Load testing* is usually a type of non-functional testing although it can be used as a functional test to validate suitability for use.

Software load testing

Main article: Software performance testing

The term *load testing* is used in different ways in the professional software testing community. *Load testing* generally refers to the practice of modeling the expected usage of a software program by simulating multiple users accessing the program concurrently. As such, this testing is most relevant for multi-user systems; often one built using a client/server model, such as web servers. However, other types of software systems can also be load tested. For example, a word processor or graphics editor can be forced to read an extremely large document; or a financial package can be forced to generate a report based on several years' worth of data. The most accurate load testing simulates actual use, as opposed to testing using theoretical or analytical modeling.

Load testing lets you measure your website's QOS performance based on actual customer behavior. Nearly all the load testing tools and frame-works follow the classical load testing paradigm, which is listed in Figure 1. When customers visit your web site, a script recorder records the communication and then creates related interaction scripts. A load generator tries to replay the recorded scripts, which could possibly be modified with different test parameters before replay. In the replay procedure, both the hardware and software statistics will be monitored and collected by the conductor, these statistics include the CPU, memory, disk IO of the physical servers and the response time, throughput of the System Under Test (short as SUT), etc. And at last, all these statistics will be analyzed and a load testing report will be generated.

Load and performance testing analyzes software intended for a multi-user audience by subjecting the software to different numbers of virtual and live users while monitoring performance measurements under these different loads. Load and performance testing is usually conducted in a test environment identical to the production environment before the software system is permitted to go live.

As an example, a web site with shopping cart capability is required to support 100 concurrent users broken out into following activities:

- 25 Virtual Users (VUsers) log in, browse through items and then log off
- 25 VUsers log in, add items to their shopping cart, check out and then log off
- 25 VUsers log in, return items previously purchased and then log off
- 25 VUsers just log in without any subsequent activity

A test analyst can use various load testing tools to create these VUsers and their activities. Once the test has started and reached a steady state, the application is being tested at the 100 VUser load as described above. The application's performance can then be monitored and captured.

The specifics of a load test plan or script will generally vary across organizations. For example, in the bulleted list above, the first item could represent 25 VUsers browsing unique items, random items, or a selected set of items depending upon the test plan or script developed. However, all load test plans attempt to simulate system performance across a range of anticipated peak workflows and volumes. The criteria for passing or failing a load test (pass/fail criteria) are generally different across organizations as well. There are no standards specifying acceptable load testing performance metrics.

A common misconception is that load testing software provides record and playback capabilities like regression testing tools. Load testing tools analyze the entire OSI protocol stack whereas most regression testing tools focus on GUI performance. For example, a regression testing tool will record and playback a mouse click on a button on a web browser, but a load testing tool will send out hypertext the web browser sends after the user clicks the button. In a multiple-user environment, load testing tools can send out hypertext for multiple users with each user having a unique login ID, password, etc.

The popular load testing tools available also provide insight into the causes for slow performance. There are numerous possible causes for slow system performance, including, but not limited to, the following:

- Application server(s) or software
- Database server(s)
- Network – latency, congestion, etc.
- Client-side processing
- Load balancing between multiple servers

Load testing is especially important if the application, system or service will be subject to a service level agreement or SLA.

User Experience Under Load test

In the example above, while the device under test (DUT) is under production load - 100 VUsers, run the target application. The performance of the target application here would be the User Experience Under Load. It describes how fast or slow the DUT responds, and how satisfied or how the user actually perceives performance.

Load testing tools

Tool Name	Company Name	Notes
Apache JMeter	An Apache Jakarta open source project	Java desktop application for load testing and performance measurement.
BlazeMeter	BlazeMeter Ltd.	BlazeMeter is a JMeter compatible, self-service, load testing platform for websites, web apps, mobile and databases, supporting any user scenario. Scalable load up to 200,000 concurrent simulated browser users from across eight geographical locations. Can also be used for integration and functional testing.
Blitz	Spirent Communications	Blitz is a service for load and performance testing of websites, mobile, web apps and REST APIs in the cloud. It allows to simulate up to 50,000 simultaneous virtual users from different worldwide locations.
Gatling	Open Source	JVM application with scenarios as code and portable HTML reports.
Loader.io	SendGrid Labs	Cloud based load testing service for developers to test performance and scalability with their web applications and APIs.
Load Impact	Load Impact - AB	Cloud based performance testing SaaS tool primarily used for executing large volume performance tests - up to 1.2 million concurrent users from 10 locations simultaneously. Also does automated mobile app testing, API testing, web/app testing. Ability to monitor backend resources during testing and create custom load scripts. Simple UI caters to developers and novices.
LoadRunner	HP	Performance testing tool primarily used for executing large numbers of tests (or a large number of virtual users) concurrently. Can be used for unit and integration testing as well. Licensed.

Load Test (included with Soatest)	Parasoft	Performance testing tool that verifies functionality and performance under load. Supports SOAtest tests, JUnits, lightweight socket-based components. Detects concurrency issues.
loadUI	SmartBear Software	Open source and cross-platform load testing tool, targeted mainly at web services. Integrates with soapUI.
Login VSI	Login VSI B.V.	Performance testing software for Windows based virtualized desktops by simulating user workloads. Licensed.
NeoLoad	Neotys	Load testing tool for web and mobile applications. Load can be generated from local agents or from the cloud. Licensed.
OpenSTA	Open System Testing Architecture	Open source web load/stress testing application, licensed under the Gnu GPL. Utilizes a distributed software architecture based on CORBA. OpenSTA binaries available for Windows.
Rational Performance Tester	IBM	Eclipse based large scale performance testing tool primarily used for executing large volume performance tests to measure system response time for server based applications. Licensed.
Silk Performer	Borland	Application performance tool with cloud and local virtual agents. Supports most protocols and applications. Licensed.
Test Studio	Telerik	Test Studio is a load testing tool that enables you to get a better understanding of how your website would perform if visited by a large number of users at the same time, thus helping you assess if your web apps meet business needs for availability and user satisfaction.
Visual Studio Ultimate edition	Microsoft	Visual Studio Ultimate edition includes a load test tool which enables a developer to execute a variety of tests (web, unit etc...) with a combination of configurations to simulate real user load.

Physical load testing

Many types of machinery, engines , structures , and motors are load tested. The load may be at a designated safe working load (SWL), full load, or at an aggravated level of load. The governing contract, technical specification or test method contains the details of conducting the test. The purpose of a mechanical load test is to verify that all the component parts of a structure including materials, base-fixings are fit for task and loading it is designed for.

Several types of load testing are employed

- Static testing is when a designated constant load is applied for a specified time.
- Dynamic testing is when a variable or moving load is applied.
- Cyclical testing consists of repeated loading and unloading for specified cycles, durations and conditions.

The *Supply of Machinery (Safety) Regulation 1992 UK* state that load testing is undertaken before the equipment is put into service for the first time. Performance testing applies a safe working load (SWL), or other specified load, for a designated time in a governing test method, specification, or contract. Under the *Lifting Operations and Lifting Equipment Regulations 1998 UK* load testing after the initial test is required if a major component is replaced, if the item is moved from one location to another or as dictated by the *Competent Person*

Car charging system

A load test can be used to evaluate the health of a car's battery. The tester consists of a large resistor that has a resistance similar to a car's starter motor and a meter to read the battery's output voltage both in the unloaded and loaded state. When the tester is used, the battery's open circuit voltage is checked first. If the open circuit voltage is below spec (12.6 volts for a fully charged battery), the battery is charged first. After reading the battery's open circuit voltage, the load is applied. When applied, it draws approximately the same current the car's starter motor would draw during cranking. Based on the specified cold cranking amperes of the battery, if the voltage under load falls below a certain point, the battery is bad. Load tests are also used on running cars to check the output of the car's alternator.

References

External links

- Modeling the Real World for Load Testing Web Sites (http://www.methodsandtools.com/archive/archive.php?id=38) by Steven Splaine
- What is Load Testing? (http://smartbear.com/products/qa-tool/what-is-load-testing/) by Tom Huston

Scenario testing

Scenario testing is a software testing activity that uses scenarios: hypothetical stories to help the tester work through a complex problem or test system. The ideal scenario test is a credible, complex, compelling or motivating story the outcome of which is easy to evaluate. These tests are usually different from test cases in that test cases are single steps whereas scenarios cover a number of steps.

History

Kaner coined the phrase scenario test by October 2003. He commented that one of the most difficult aspects of testing was maintaining step-by-step test cases along with their expected results. His paper attempted to find a way to reduce the re-work of complicated written tests and incorporate the ease of use cases.

A few months later, Buwalda wrote about a similar approach he had been using that he called "soap opera testing". Like television soap operas these tests were both exaggerated in activity and condensed in time. The key to both approaches was to avoid step-by-step testing instructions with expected results and instead replaced them with a narrative that gave freedom to the tester while confining the scope of the test.

Methods

System scenarios

In this method only those sets of realistic, user activities that cover several components in the system are used as scenario tests. Development of system scenario can be done using:Wikipedia:Citation needed

1. Story lines
2. State transitions
3. Business verticals
4. Implementation story from customers

Use-case and role-based scenarios

In this method the focus is on how a user uses the system with different roles and environment.Wikipedia:Verifiability

References

Test plan

A **test plan** is a document detailing a systematic approach to testing a system such as a machine or software. The plan typically contains a detailed understanding of the eventual workflow.

Test plans

A test plan documents the strategy that will be used to verify and ensure that a product or system meets its design specifications and other requirements. A test plan is usually prepared by or with significant input from test engineers.

Depending on the product and the responsibility of the organization to which the test plan applies, a test plan may include a strategy for one or more of the following:

- *Design Verification or Compliance test* - to be performed during the development or approval stages of the product, typically on a small sample of units.
- *Manufacturing or Production test* - to be performed during preparation or assembly of the product in an ongoing manner for purposes of performance verification and quality control.
- *Acceptance or Commissioning test* - to be performed at the time of delivery or installation of the product.
- *Service and Repair test* - to be performed as required over the service life of the product.
- *Regression test* - to be performed on an existing operational product, to verify that existing functionality didn't get broken when other aspects of the environment are changed (e.g., upgrading the platform on which an existing application runs).

A complex system may have a high level test plan to address the overall requirements and supporting test plans to address the design details of subsystems and components.

Test plan document formats can be as varied as the products and organizations to which they apply. There are three major elements that should be described in the test plan: Test Coverage, Test Methods, and Test Responsibilities. These are also used in a formal test strategy.

Test coverage

Test coverage in the test plan states what requirements will be verified during what stages of the product life. Test Coverage is derived from design specifications and other requirements, such as safety standards or regulatory codes, where each requirement or specification of the design ideally will have one or more corresponding means of verification. Test coverage for different product life stages may overlap, but will not necessarily be exactly the same for all stages. For example, some requirements may be verified during Design Verification test, but not repeated during Acceptance test. Test coverage also feeds back into the design process, since the product may have to be designed to allow test access (see Design For Test).

Test methods

Test methods in the test plan state how test coverage will be implemented. Test methods may be determined by standards, regulatory agencies, or contractual agreement, or may have to be created new. Test methods also specify test equipment to be used in the performance of the tests and establish pass/fail criteria. Test methods used to verify hardware design requirements can range from very simple steps, such as visual inspection, to elaborate test procedures that are documented separately.

Test responsibilities

Test responsibilities include what organizations will perform the test methods and at each stage of the product life. This allows test organizations to plan, acquire or develop test equipment and other resources necessary to implement the test methods for which they are responsible. Test responsibilities also includes, what data will be collected, and how that data will be stored and reported (often referred to as "deliverables"). One outcome of a successful test plan should be a record or report of the verification of all design specifications and requirements as agreed upon by all parties.

IEEE 829 test plan structure

IEEE 829-2008, also known as the 829 Standard for Software Test Documentation, is an IEEE standard that specifies the form of a set of documents for use in defined stages of software testing, each stage potentially producing its own separate type of document.

- Test plan identifier
- Introduction
- Test items
- Features to be tested
- Features not to be tested
- Approach
- Item pass/fail criteria
- Suspension criteria and resumption requirements
- Test deliverables
- Testing tasks
- Environmental needs
- Responsibilities
- Staffing and training needs
- Schedule
- Risks and contingencies
- Approvals

The IEEE documents that suggest what should be contained in a test plan are:

- 829-2008 *IEEE Standard for Software and System Test Documentation*
  - 829-1998 *IEEE Standard for Software Test Documentation* (superseded by 829-2008)
  - 829-1983 *IEEE Standard for Software Test Documentation* (superseded by 829-1998)
- 1008-1987 *IEEE Standard for Software Unit Testing*
- 1012-2004 *IEEE Standard for Software Verification and Validation*
  - 1012-1998 *IEEE Standard for Software Verification and Validation* (superseded by 1012-2004)
  - 1012-1986 *IEEE Standard for Software Verification and Validation Plans* (superseded by 1012-1998)
- 1059-1993 *IEEE Guide for Software Verification & Validation Plans* (withdrawn)

References

External links

- Public domain RUP test plan template at Sourceforge (http://jdbv.sourceforge.net/RUP.html) (templates are currently inaccessible but sample documents can be seen here: DBV Samples (http://jdbv.sourceforge.net/Documentation.html))
- Test plans and test cases (http://www.stellman-greene.com/testplan)

Apache JMeter

Apache JMeter

Developer(s)	Apache Software Foundation
Stable release	2.11 / January 5, 2014
Preview release	nightly build [1] / nightly build
Written in	Java
Operating system	Cross-platform
Type	Load Testing
License	Apache License 2.0
Website	http://jmeter.apache.org/

Apache JMeter is an Apache project that can be used as a load testing tool for analyzing and measuring the performance of a variety of services, with a focus on web applications.

JMeter can be used as a unit test tool for JDBC database connections, FTP, LDAP, Webservices, JMS, HTTP, generic TCP connections and OS Native processes. JMeter can also be configured as a monitor, although this is typically considered an *ad hoc* solution in lieu of advanced monitoring solutions. It can be used for some functional testing as well.

JMeter supports variable parameterization, assertions (response validation), per thread cookies, configuration variables and a variety of reports.

JMeter architecture is based on plugins. Most of its "out of the box" features are implemented with plugins. Off-site developers can easily extend JMeter with custom plugins.

Releases

Apache JMeter versions

Version	Release Date	Description
1.0.2	2001-03-09	earliest in archive
...	...	
2.3RC3	2007-07-11	
2.3RC4	2007-09-02	
2.3	2007-09-24	
2.3.1	2007-11-28	
2.3.2	2008-06-10	
2.3.3	2009-05-24	
2.3.4	2009-06-21	Java 1.4+
2.4	2010-07-14	Java 5+
2.5	2011-08-17	Java 5
2.5.1	2011-10-03	Java 5+

2.6	2012-02-01	Java 5+
2.7	2012-05-27	Java 5+
2.8	2012-10-06	Java 5+
2.9	2013-01-28	Java 6+
2.10	2013-10-21	Java 6+
2.11	2014-01-05	Java 6+

References

[1] http://people.apache.org/builds/jakarta-jmeter/nightly/

External links

- Official JMeter website (http://jmeter.apache.org/)
- Example of JMeter Custom Plugins (http://jmeter-plugins.org/)

Chapter 23: Automated Browser Testing with Selenium

Regression testing

Regression testing is a type of software testing that seeks to uncover new software bugs, or *regressions*, in existing functional and non-functional areas of a system after changes such as enhancements, patches or configuration changes, have been made to them.

The intent of regression testing is to ensure that a change such as those mentioned above has not introduced new faults. One of the main reasons for regression testing is to determine whether a change in one part of the software affects other parts of the software.

Common methods of regression testing include rerunning previously completed tests and checking whether program behavior has changed and whether previously fixed faults have re-emerged. Regression testing can be performed to test a system efficiently by systematically selecting the appropriate minimum set of tests needed to adequately cover a particular change.

Background

Experience has shown that as software is fixed, emergence of new and/or re-emergence of old faults is quite common. Sometimes re-emergence occurs because a fix gets lost through poor revision control practices (or simple human error in revision control). Often, a fix for a problem will be "fragile" in that it fixes the problem in the narrow case where it was first observed but not in more general cases which may arise over the lifetime of the software. Frequently, a fix for a problem in one area inadvertently causes a software bug in another area. Finally, it may happen that, when some feature is redesigned, some of the same mistakes that were made in the original implementation of the feature are made in the redesign.

Therefore, in most software development situations, it is considered good coding practice, when a bug is located and fixed, to record a test that exposes the bug and re-run that test regularly after subsequent changes to the program. Although this may be done through manual testing procedures using programming techniques, it is often done using automated testing tools.[1] Such a test suite contains software tools that allow the testing environment to execute all the regression test cases automatically; some projects even set up automated systems to automatically re-run all regression tests at specified intervals and report any failures (which could imply a regression or an out-of-date test). Common strategies are to run such a system after every successful compile (for small projects), every night, or once a week. Those strategies can be automated by an external tool.

Regression testing is an integral part of the extreme programming software development method. In this method, design documents are replaced by extensive, repeatable, and automated testing of the entire software package throughout each stage of the software development cycle.

In the corporate world, regression testing has traditionally been performed by a software quality assurance team after the development team has completed work. However, defects found at this stage are the most costly to fix. This problem is being addressed by the rise of unit testing. Although developers have always written test cases as part of the development cycle, these test cases have generally been either functional tests or unit tests that verify only intended outcomes. Developer testing compels a developer to focus on unit testing and to include both positive and negative test cases.

Uses

Regression testing can be used not only for testing the *correctness* of a program, but often also for tracking the quality of its output. For instance, in the design of a compiler, regression testing could track the code size, and the time it takes to compile and execute the test suite cases.

> "Also as a consequence of the introduction of new bugs, program maintenance requires far more system testing per statement written than any other programming. Theoretically, after each fix one must run the entire batch of test cases previously run against the system, to ensure that it has not been damaged in an obscure way. In practice, such *regression testing* must indeed approximate this theoretical idea, and it is very costly."
>
> — Fred Brooks, *The Mythical Man Month, p. 122*

Regression tests can be broadly categorized as functional tests or unit tests. Functional tests exercise the complete program with various inputs. Unit tests exercise individual functions, subroutines, or object methods. Both functional testing tools and unit testing tools tend to be third-party products that are not part of the compiler suite, and both tend to be automated. A functional test may be a scripted series of program inputs, possibly even involving an automated mechanism for controlling mouse movements and clicks. A unit test may be a set of separate functions within the code itself, or a driver layer that links to the code without altering the code being tested.

References

[1] Automate Regression Tests When Feasible (http://safari.oreilly.com/0201794292/ch08lev1sec4), Automated Testing: Selected Best Practices, Elfriede Dustin, Safari Books Online

External links

- Microsoft regression testing recommendations (http://msdn.microsoft.com/en-us/library/aa292167(VS.71).aspx)
- Gauger performance regression visualization tool (https://gnunet.org/gauger/)
- What is Regression Testing (http://smartbear.com/products/qa-tools/what-is-regression-testing/) by Scott Barber and Tom Huston

Web browser engine

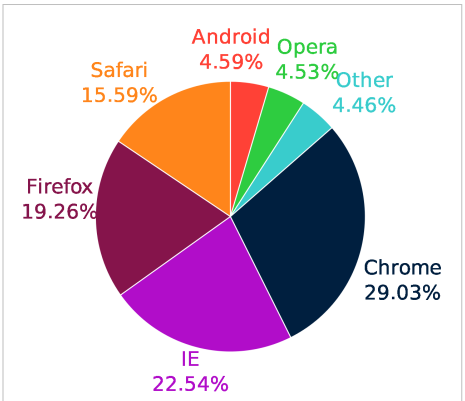
A **web browser engine** (sometimes called **layout engine** or **rendering engine**) is a software component that takes marked up content (such as HTML, XML, image files, etc.) and formatting information (such as CSS, XSL, etc.) and displays the formatted content on the screen. It draws on the content area of a window, which is displayed on a monitor or a printer. A layout engine is typically embedded in web browsers, e-mail clients, e-book readers, on-line help systems or other applications that require the displaying (and editing) of web content. Engines may wait for all data to be received before rendering a page, or may begin rendering before all data is received. This can result in pages changing as more data is received, such as images being filled in or a flash of unstyled content if rendering begins before formatting information is received.

Non-mobile web browser statistics on Wikipedia projects	
Chrome (Blink)	43.00%
Internet Explorer (Trident)	25.80%
Firefox (Gecko)	18.22%
Safari (WebKit)	5.90%
Opera (Blink)	2.31%
Others	4.77%

Non-mobile web browser usage for Wikimedia visitors as of February 2014[1].

Examples

KDE's open-source KHTML engine is used in KDE's Konqueror web browser and was the basis for WebKit, the rendering engine in Apple's Safari and Google's Chrome web browsers, which is now the most widely used browser engine according to StatCounter. Current versions of Chromium/Chrome (except iOS version) and Opera are based on Blink, a fork of WebKit.



Web browser usage (as of 2012) on Wikipedia project servers

Gecko, the Mozilla project's open-source web browser engine, is used by a variety of products derived from the Mozilla code base, including the Firefox web browser, the Thunderbird e-mail client, and SeaMonkey internet suite.

Trident, the web browser engine from Internet Explorer, is used by many applications on the Microsoft Windows platform, such as netSmart, Outlook Express, some versions of Microsoft Outlook, and the mini-browsers in Winamp and RealPlayer.

Opera Software's proprietary Presto engine is licensed to a number of other software vendors, and was used in Opera's own web browser.

Technical operation

The first web browsers were monolithic. They used various techniques inherited from text processing, such as regular expressions, to parse HTML into a visual representation. Later they adopted a more modular approach and were split into a host application and an engine.

- The **engine** does most of the work. It essentially takes a URL and a set of window content-area rectangle coordinates as arguments. It then retrieves the document corresponding to the URL and paints a graphical representation of it in the given rectangle. It also handles links, forms, cookies, client-side scripting, plug-in loading and other matters.
- The **host application** provides the menu bar, address bar, status bar, bookmark manager, history and preferences functionality (among other things). It embeds the engine and serves as an interface between the user, the engine, and the underlying operating system. Since it provides the graphical elements surrounding the area in which the engine paints documents, programmers sometimes use the term *chrome* to refer to its user interface (like the chrome surrounding a car).

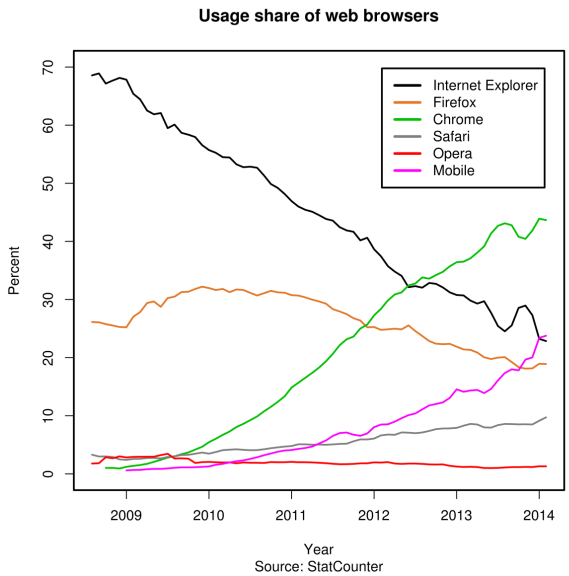
This modular approach has the advantage that it then becomes easy to embed web-browser engines in a variety of applications. For example, the same engine used by a web browser can be used by an email client to display HTML email. On-line help systems integrated in applications have largely moved from using custom formats to using standard HTML displayed with a web-browser engine. The EPUB 3 e-book standard uses a layout engine to render XHTML and CSS.

References

[1] http://en.wikipedia.org/w/index.php?title=Web_browser_engine&action=edit

Comparison of web browser engines

The following tables compare general and technical information for a number of layout engines. While these are mainly used in web browsers, they are also used in email clients for rendering HTML email, and used to render EPUB e-books, for example. This article is not all-inclusive—please see individual "Comparison of layout engine" articles for detailed comparisons of HTML5 feature support, CSS feature support, and the like. Unless otherwise specified in footnotes, comparisons are based on the stable versions without any add-ons, extensions or external programs.



Usage share as of 2013 by percent of layout engines/web browsers.

General information

Basic general information about the engines.

Engine	Developer(s)	Software license	Leading application	Target application(s)	Programming language
Blink[1]	Google, Opera, Samsung, Intel, others	GNU LGPL, BSD-style	Google Chrome	Google Chrome & Opera from 15.0	C++
Gecko	Netscape/Mozilla Foundation	MPL	Mozilla Firefox	Mozilla Firefox & Mozilla Thunderbird	C++
GtkHTML[2]	GNOME	GNU LGPL	Novell Evolution	Novell Evolution	C
Hubbub	Andrew Sidwell	MIT	NetSurf	NetSurf	C
iCab	Alexander Clauss	Proprietary	iCab	iCab	*?*
KHTML	KDE	GNU LGPL	Konqueror	Konqueror & KMail	C++
NetFront	Access Co.	Proprietary	NetFront	NetFront	*?*
Presto	Opera Software	Proprietary	Opera	Opera[3]	C++
Prince XML	YesLogic Pvt Ltd	Proprietary	Prince XML	Prince XML	Mercury
Robin	Ritlabs	Proprietary	The Bat!	The Bat!	Delphi
Tasman	Microsoft	Proprietary	Microsoft Entourage	Internet Explorer for Mac & Microsoft Entourage	*?*
Trident	Microsoft	Proprietary	Internet Explorer	Internet Explorer	C++
WebKit[4]	Apple, KDE, Nokia, BlackBerry, Palm, others	GNU LGPL, BSD-style	Apple Safari	Apple Safari	C++
XEP	RenderX	Proprietary	XEP	XEP	Java

Release history

A brief overview of the release history.

Engine	First public release		First stable release		Latest stable release	
	Date	Version	Date	Version	Date	Version
Blink	3 April 2013	No number	3 April 2013	No number	N/A	SVN version only
Gecko	7 December 1998	"Preview"	19 March 1999	M3	14 May 2013	21.0
GtkHTML	2000	?	2000	?	14 December 2009	3.28.2
Hubbub	22 April 2002	?	17 May 2007	1.0	20 April 2013	3.0
iCab	1998	?	1998	?	1 January 2008	3.0.5[5]
KHTML	October 2000	?	October 2000	?	4 August 2009	4.3
NetFront	1995	?	1995	?	13 January 2010	4.0
Presto	13 November 2002	1.0	28 January 2003	1.0	5 November 2012	2.12.388
Prince XML	April 2003	1.0	April 2003	1.0	May 2010	7.1
Robin	27 April 2000	1.32	27 April 2000	1.32	24 August 2009	4.2.10
Tasman	27 March 2000	0	27 March 2000	0	11 May 2004	1.0
Trident	April 1997	No number	October 1997	No number	17 October 2013	7.0
WebKit	7 January 2003	48	23 June 2003	85	N/A	SVN version only
XEP	1999	fo2pdf	?	?	March 2010	4.18

Operating system support

The operating systems the engines can run on without emulation.

Engine	Windows	OS X	Linux	BSD	Unix	Symbian OS
Blink[6]	Yes	Yes	Yes	Yes	Yes	No
Gecko	Yes	Yes[7]	Yes	Yes	Yes	No
GtkHTML	Yes	Yes	Yes	Yes	Yes	No
Hubbub	No	Yes	Yes	Yes	Yes	No
iCab	No	Yes	No	No	No	No
KHTML	Yes	Yes	Yes	Yes	Yes	Yes
NetFront	Partial[8]	No	Yes	No	No	Dropped[9]
Presto	Dropped (12.16)	Dropped (12.16)	Yes	Yes	Yes	Yes
Prince XML	Yes	Yes	Yes	Yes	Yes	No
Robin	Yes	No	No	No	No	No
Tasman	No	Dropped (5.2.3)	No	No	No	No
Trident	Yes	Dropped (4.0)	No[10]	No	Dropped (5.0)	No
WebKit	Yes	Yes	Yes	Yes	Yes	Yes
XEP[11]	Yes	Yes	Yes	Yes	Yes	No

Notes

[1] Blink was created by Google by forking WebKit

[2] This engine is not currently being developed.

[3] Opera switched to Webkit, then followed Google to Blink.

[4] WebKit was created by Apple by forking KHTML. Subsequently Apple released it as an open source project.

[5] The iCab 4 browser uses the WebKit engine - iCab 3.0.5 was the final release of the Carbon (API)-based iCab engine

[6] Blink cannot be used alone and must be used via Chromium's content layer which has wide platform support.

[7] Although dropped in current version, older versions of the Gecko web browser engine for Mac OS 8.6 and Mac OS 9 are still available for download from Netscape's Archived Products site (http://browser.netscape.com/ns8/download/archive70x.jsp). An updated port of the Mozilla Application Suite for classic Mac OS systems is maintained as Classilla.

[8] NetFront supports Windows CE and Android, but is mainly used as an embedded browser on low-end mobile phones.

[9] From Access website (http://www.access-company.com/support/netfront_support.html) (2010/06/19): "Access no longer offers nor supports NetFront Browser for Symbian."

[10] Through the use of the Wine Libraries some version of IE can be started.

[11] XEP is written in Java, with a dedicated release line for Windows.

References

Selenium (software)

Selenium

Stable release	2.40 / February 19, 2014
Development status	Active
Written in	Java
Operating system	Cross-platform
Type	Software testing framework for web applications
License	Apache License 2.0
Website	seleniumhq.org [1]

Selenium is a portable software testing framework for web applications. Selenium provides a record/playback tool for authoring tests without learning a test scripting language (**Selenium IDE**). It also provides a test domain-specific language (**Selenese**) to write tests in a number of popular programming languages, including Java, C#, Groovy, Perl, PHP, Python and Ruby. The tests can then be run against most modern web browsers. Selenium deploys on Windows, Linux, and Macintosh platforms.

History

Selenium was originally developed by Jason Huggins in 2004, who was later joined by other programmers and testers at ThoughtWorks. It is open-source software, released under the Apache 2.0 license, and can be downloaded and used without charge.

The name comes from a joke made by Huggins in an email, mocking a competitor named Mercury, saying that you can cure mercury poisoning by taking Selenium supplements. The others that received the email took the name and ran with it.

The latest side project is Selenium Grid, which provides a hub allowing the running of multiple Selenium tests concurrently on any number of local or remote systems, thus minimizing test execution time.

Selenium components

Selenium IDE

Selenium IDE is a complete integrated development environment (IDE) for Selenium tests. It is implemented as a Firefox Add-On, and allows recording, editing, and debugging tests. It was previously known as Selenium Recorder. **Selenium-IDE** was originally created by Shinya Kasatani and donated to the Selenium project in 2006.Wikipedia:Citation needed

Scripts may be automatically recorded and edited manually providing autocompletion support and the ability to move commands around quickly. Scripts are recorded in *Selenese*, a special test scripting language for Selenium. Selenese provides commands for performing actions in a browser (click a link, select an option), and for retrieving data from the resulting pages.

Selenium Client API

As an alternative to writing tests in Selenese, tests can also be written in various programming languages. These tests then communicate with Selenium by calling methods in the Selenium Client API. Selenium currently provides client APIs for Java, C#, Ruby and Python.

With Selenium 2, a new Client API was introduced (with *WebDriver* as its central component). However, the old API (using class *Selenium*) is still supported.

Selenium Remote Control

Selenium Remote Control (**RC**) is a server, written in Java, that accepts commands for the browser via HTTP. RC makes it possible to write automated tests for a web application in any programming language, which allows for better integration of Selenium in existing unit test frameworks. To make writing tests easier, Selenium project currently provides client drivers for PHP, Python, Ruby, .NET, Perl and Java. The Java driver can also be used with JavaScript (via the Rhino engine). A new instance of selenium RC server is needed to launch html test case - which means that the port should be different for each parallel run.Wikipedia:Citation needed However, for Java/PHP test case only one Selenium RC instance needs to be running continuously.Wikipedia:Citation needed

Selenium Remote Control was a refactoring of **Driven Selenium** or **Selenium B** designed by Paul Hammant, credited with Jason as co-creator of Selenium. The original version directly launched a process for the browser in question, from the test language of Java, .Net, Python or Ruby. The wire protocol (called 'Selenese' in its day) was reimplemented in each language port. After the refactor by Dan Fabulich, and Nelson Sproul (with help from Pat Lightbody) there was an intermediate daemon process between the driving test script, and the browser. The benefits included the ability to drive remote browsers, and the reduced need to port every line of code to an increasingly growing set of languages. **Selenium Remote Control** completely took over from the **Driven Selenium** code-line in 2006. The browser pattern for 'Driven'/'B' and 'RC' was response/request, which subsequently became known as Comet.

With the release of Selenium 2, Selenium RC has been officially deprecated in favor of Selenium WebDriver.

Selenium WebDriver

Selenium WebDriver is the successor to Selenium RC. Selenium WebDriver accepts commands (sent in Selenese, or via a Client API) and sends them to a browser. This is implemented through a browser-specific **browser driver**, which sends commands to a browser, and retrieves results. Most browser drivers actually launch and access a browser application (such as Firefox or Internet Explorer); there is also an HtmlUnit browser driver, which simulates a browser using HtmlUnit.

Unlike in Selenium 1, where the Selenium server was necessary to run tests, Selenium WebDriver does not need a special server to execute tests. Instead, the WebDriver directly starts a browser instance and controls it. However, Selenium Grid can be used with WebDriver to execute tests on remote systems (see below).

In practice, this means that the Selenium 2.0 API has significantly fewer calls than does the Selenium 1.0 API. Where Selenium 1.0 attempted to provide a rich interface for many different browser operations, Selenium 2.0 aims to provide a basic set of building blocks from which developers can create their own Domain Specific Language. One such DSL already exists: the Watir project in the Ruby language has a rich history of good design. Watir-webdriver implements the Watir API as a wrapper for Selenium-Webdriver in Ruby. Watir-webdriver is created entirely automatically, based on the WebDriver specification and the HTML specification.

As of early 2012, Simon Stewart (inventor of WebDriver), who was then with Google and now with Facebook, and David Burns of Mozilla were negotiating with the W3C to make WebDriver an internet standard. In early 2013, the working draft was released. As such, Selenium-Webdriver (Selenium 2.0) aims to be the reference implementation of the WebDriver standard in various programming languages. Currently Selenium-WebDriver is fully implemented

and supported in Python, Ruby, Java, and C#.

Selenium Grid

Selenium Grid is a server that allows tests to use web browser instances running on remote machines. With Selenium Grid, one server acts as the **hub**. Tests contact the hub to obtain access to browser instances. The hub has a list of servers that provide access to browser instances (**WebDriver node**s), and lets tests use these instances. Selenium Grid allows running tests in parallel on multiple machines, and to manage different browser versions and browser configurations centrally (instead of in each individual test).

References

[1] http://seleniumhq.org/

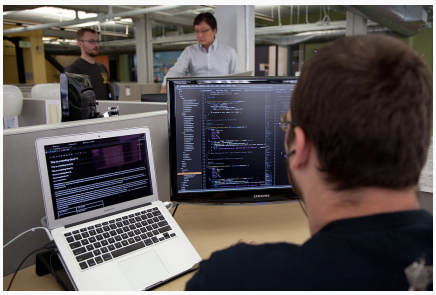
External links

- Selenium Project home page (http://seleniumhq.org/)

Chapter 24: QA Automation with Jenkins, Apache Maven, and SonarQube

Build automation

Software development process



A software developer at work

Core activities
• Requirements • Specification • Architecture • Construction • Design • Testing • Debugging • Deployment • Maintenance
Methodologies
• Waterfall • Prototype model • Incremental • Iterative • V-Model • Spiral • Scrum • Cleanroom • RAD • DSDM • RUP • XP • Agile • Lean • Dual Vee Model • TDD

• FDD • DDD • MDD
Supporting disciplines
• Configuration management • Documentation • Quality assurance (SQA) • Project management • User experience
Tools
• Compiler • Debugger • Profiler • GUI designer • Modeling • IDE • Build automation
• v • t • e [1]

Build automation is the act of scripting or automating a wide variety of tasks that software developers do in their day-to-day activities including things like:

- compiling computer source code into binary code
- packaging binary code
- running tests
- deployment to production systems
- creating documentation and/or release notes

History

Historically, developers used build automation to call compilers and linkers from inside a build script versus attempting to make the compiler calls from the command line. It is simple to use the command line to pass a single source module to a compiler and then to a linker to create the final deployable object. However, when attempting to compile and link many source code modules, in a particular order, using the command line process is not a reasonable solution. The make scripting language offered a better alternative. It allowed a build script to be written to call, in a series, the needed compile and link steps to build a software application. GNU Make also offered additional features such as "makedepend" which allowed some source code dependency management as well as incremental build processing. This was the beginning of Build Automation. Its primary focus was on automating the calls to the compilers and linkers. As the build process grew more complex, developers began adding pre and post actions around the calls to the compilers such as a check-out from version control to the copying of deployable objects to a test location. The term "build automation" now includes managing the pre and post compile and link activities as well as the compile and link activities.

New breed of tools

In recent years, build management tools have provided even more relief when it comes to automating the build process. Both commercial and open source tools are available to perform more automated build and workflow processing. Some tools focus on automating the pre and post steps around the calling of the build scripts, while others go beyond the pre and post build script processing and also streamline the actual compile and linker calls without much manual scripting. These tools are particularly useful for continuous integration builds where frequent calls to the compile process are required and incremental build processing is needed.

Advanced build automation

Advanced build automation offers remote agent processing for distributed builds and/or distributed processing. The term "distributed builds" means that the actual calls to the compiler and linkers can be served out to multiple locations for improving the speed of the build. This term is often confused with "distributed processing".

Distributed processing means that each step in a process or workflow can be sent to a different machine for execution. For example, a post step to the build may require the execution of multiple test scripts on multiple machines. Distributed processing can send the different test scripts to different machines. Distributed processing is not distributed builds. Distributed processing cannot take a make, ant or maven script, break it up and send it to different machines for compiling and linking.

The distributed build process must have the machine intelligence to understand the source code dependencies in order to send the different compile and link steps to different machines. A build automation tool must be able to manage these dependencies in order to perform distributed builds. Some build tools can discover these relationships programmatically (Rational ClearMake distributed, Electric Cloud ElectricAccelerator), while others depend on user-configured dependencies (Platform LSF lsmake)

Build automation that can sort out source code dependency relationships can also be configured to run the compile and link activities in a parallelized mode. This means that the compiler and linkers can be called in multi-threaded mode using a machine that is configured with more than one core.

Not all build automation tools can perform distributed builds. Most only provide distributed processing support. In addition, most products that do support distributed builds can only handle C or C++. Build automation products that support distributed processing are often based on make and many do not support Maven or Ant.

The deployment task may require configuration of external systems, including middleware. In cloud computing environments the deployment step may even involve creation of virtual servers to deploy build artifacts into.

Advantages

The advantages of build automation to software development projects include

- Improve product quality
- Accelerate the compile and link processing
- Eliminate redundant tasks
- Minimize "bad builds"
- Eliminate dependencies on key personnel
- Have history of builds and releases in order to investigate issues
- Save time and money - because of the reasons listed above.[2]

Types

- **On-Demand automation** such as a user running a script at the command line
- **Scheduled automation** such as a continuous integration server running a nightly build
- **Triggered automation** such as a continuous integration server running a build on every commit to a version control system.

Makefile

One specific form of build automation is the automatic generation of Makefiles. *See* List of build automation software.

Requirements of a build system

Basic requirements:

1. Frequent or overnight builds to catch problems early.[3][4][5]
2. Support for Source Code Dependency Management
3. Incremental build processing
4. Reporting that traces source to binary matching
5. Build acceleration
6. Extraction and reporting on build compile and link usage

Optional requirements:[6]

1. Generate release notes and other documentation such as help pages
2. Build status reporting
3. Test pass or fail reporting
4. Summary of the features added/modified/deleted with each new build

References

[1] http://en.wikipedia.org/w/index.php?title=Template:Software_development_process&action=edit
[2] http://www.denverjug.org/meetings/files/200410_automation.pdf
[3] http://freshmeat.net/articles/view/392/
[4] http://www.ibm.com/developerworks/java/library/j-junitmail/
[5] http://buildbot.net/trac
[6] http://www.cmcrossroads.com/content/view/12525/120/

Notes

- Mike Clark: *Pragmatic Project Automation*, The Pragmatic Programmers ISBN 0-9745140-3-9

Revision control

For the Wikipedia revision control system, see Wikipedia:Revision control.

Revision control, also known as **version control** and **source control** (and an aspect of software configuration management), is the management of changes to documents, computer programs, large web sites, and other collections of information. Changes are usually identified by a number or letter code, termed the "revision number", "revision level", or simply "revision". For example, an initial set of files is "revision 1". When the first change is made, the resulting set is "revision 2", and so on. Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged.

The need for a logical way to organize and control revisions has existed for almost as long as writing has existed, but revision control became much more important, and complicated, when the era of computing began. The numbering of book editions and of specification revisions are examples that date back to the print-only era. Today, the most capable (as well as complex) revision control systems are those used in software development, where a team of people may change the same files.

Version control systems (VCS) most commonly run as stand-alone applications, but revision control is also embedded in various types of software such as word processors and spreadsheets, e.g., Google Docs and Sheets and in various content management systems, e.g., Wikipedia's Page history. Revision control allows for the ability to revert a document to a previous revision, which is critical for allowing editors to track each other's edits, correct mistakes, and defend against vandalism and spamming.

Software tools for revision control are essential for the organization of multi-developer projects.

Overview

In computer software engineering, revision control is any kind of practice that tracks and provides control over changes to source code. Software developers sometimes use revision control software to maintain documentation and configuration files as well as source code.

As teams design, develop and deploy software, it is common for multiple versions of the same software to be deployed in different sites and for the software's developers to be working simultaneously on updates. Bugs or features of the software are often only present in certain versions (because of the fixing of some problems and the introduction of others as the program develops). Therefore, for the purposes of locating and fixing bugs, it is vitally important to be able to retrieve and run different versions of the software to determine in which version(s) the problem occurs. It may also be necessary to develop two versions of the software concurrently (for instance, where one version has bugs fixed, but no new features (branch), while the other version is where new features are worked on (trunk).

At the simplest level, developers could simply retain multiple copies of the different versions of the program, and label them appropriately. This simple approach has been used on many large software projects. While this method can work, it is inefficient as many near-identical copies of the program have to be maintained. This requires a lot of self-discipline on the part of developers, and often leads to mistakes. Consequently, systems to automate some or all of the revision control process have been developed.

Moreover, in software development, legal and business practice and other environments, it has become increasingly common for a single document or snippet of code to be edited by a team, the members of which may be geographically dispersed and may pursue different and even contrary interests. Sophisticated revision control that tracks and accounts for ownership of changes to documents and code may be extremely helpful or even indispensable in such situations.

Revision control may also track changes to configuration files, such as those typically stored in `/etc` or `/usr/local/etc` on Unix systems. This gives system administrators another way to easily track changes made

and a way to roll back to earlier versions should the need arise.

Structure

Revision control manages changes to a set of data over time. These changes can be structured in various ways.

Often the data is thought of as a collection of many individual items, such as files or documents, and changes to individual files are tracked. This accords with intuitions about separate files, but causes problems when identity changes, such as during renaming, splitting, or merging of files. Accordingly, some systems, such as git, instead consider changes to the data as a whole, which is less intuitive for simple changes, but simplifies more complex changes.

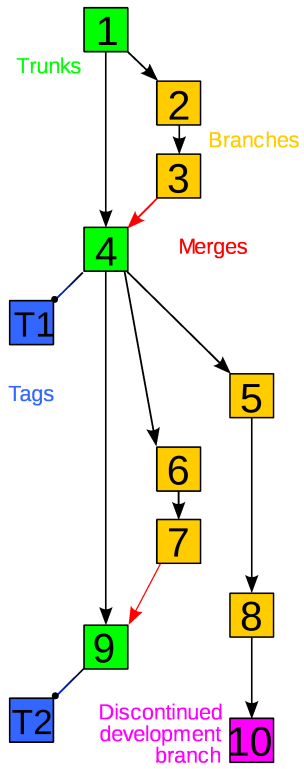
When data that is under revision control is modified, after being retrieved by *checking out,* this is not in general immediately reflected in the revision control system (in the *repository*), but must instead be *checked in* or *committed.* A copy outside revision control is known as a "working copy". As a simple example, when editing a computer file, the data stored in memory by the editing program is the working copy, which is committed by saving. Concretely, one may print out a document, edit it by hand, and only later manually input the changes into a computer and save it. For source code control, the working copy is instead a copy of all files in a particular revision, generally stored locally on the developer's computer;[1] in this case saving the file only changes the working copy, and checking into the repository is a separate step.

If multiple people are working on a single data set or document, they are implicitly creating branches of the data (in their working copies), and thus issues of merging arise, as discussed below. For simple collaborative document editing, this can be prevented by using file locking or simply avoiding working on the same document that someone else is working on.

Revision control systems are often centralized, with a single authoritative data store, the *repository,* and check-outs and check-ins done with reference to this central repository. Alternatively, in distributed revision control, no single repository is authoritative, and data can be checked out and checked into any repository. When checking into a different repository, this is interpreted as a merge or patch.

Graph structure

In terms of graph theory, revisions are generally thought of as a line of development (the *trunk*) with branches off of this, forming a directed tree, visualized as one or more parallel lines of development (the "mainlines" of the branches) branching off a trunk. In reality the structure is more complicated, forming a directed acyclic graph, but for many purposes "tree with merges" is an adequate approximation.



Example history graph of a revision-controlled project; trunk is in green, branches in yellow, and graph is not a tree due to presence of merges (the red arrows).

Revisions occur in sequence over time, and thus can be arranged in order, either by revision number or timestamp.[2] Revisions are based on past revisions, though it is possible to largely or completely replace an earlier revision, such as "delete all existing text, insert new text". In the simplest case, with no branching or undoing, each revision is based on its immediate predecessor alone, and they form a simple line, with a single latest version, the "HEAD" revision or *tip*. In graph theory terms, drawing each revision as a point and each "derived revision" relationship as an arrow (conventionally pointing from older to newer, in the same direction as time), this is a linear graph. If there is branching, so multiple future revisions are based on a past revision, or undoing, so a revision can depend on a revision older than its immediate predecessor, then the resulting graph is instead a directed tree (each node can have more than one child), and has multiple tips, corresponding to the revisions without children ("latest revision on each branch").[3] In principle the resulting tree need not have a preferred tip ("main" latest revision) – just various different revisions – but in practice one tip is generally identified as HEAD. When a new revisions is based on HEAD, it is either identified as the new HEAD, or considered a new branch.[4] The list of revisions from the start to HEAD (in graph theory terms, the unique path in the tree, which forms a linear graph as before) is the *trunk* or *mainline*.[5] Conversely, when a revision can be based on more than one previous revision (when a node can have more than one *parent*), the resulting process is called a *merge,* and is one of the most complex aspects of revision control. This most often occurs when changes occur in multiple branches (most often two, but more are possible), which are then merged into a single branch incorporating both changes. If these changes overlap, it may be difficult or impossible to merge, and require manual intervention or rewriting.

In the presence of merges, the resulting graph is no longer a tree, as nodes can have multiple parents, but is instead a rooted directed acyclic graph (DAG). The graph is acyclic since parents are always backwards in time, and rooted because there is an oldest version. However, assuming that there is a trunk, merges from branches can be considered as "external" to the tree – the changes in the branch are packaged up as a *patch,* which is applied to HEAD (of the trunk), creating a new revision without any explicit reference to the branch, and preserving the tree structure. Thus, while the actual relations between versions form a DAG, this can be considered a tree plus merges, and the trunk itself is a line.

In distributed revision control, in the presence of multiple repositories these may be based on a single original version (a root of the tree), but there need not be an original root, and thus only a separate root (oldest revision) for each repository, for example if two people starting working on a project separately. Similarly in the presence of multiple data sets (multiple projects) that exchange data or merge, there isn't a single root, though for simplicity one may think of one project as primary and the other as secondary, merged into the first with or without its own revision history.

Specialized strategies

Engineering revision control developed from formalized processes based on tracking revisions of early blueprints or bluelines. This system of control implicitly allowed returning to any earlier state of the design, for cases in which an engineering dead-end was reached in the development of the design. A revision table was used to keep track of the changes made. Additionally, the modified areas of the drawing were highlighted using revision clouds.

Version control is also widespread in business and law. Indeed, "contract redline" and "legal blackline" are some of the earliest forms of revision control,Wikipedia:Citation needed and are still employed in business and law with varying degrees of sophistication. An entire industry has emerged to service the document revision control needs of business and other users, and some of the revision control technology employed in these circles is subtle, powerful, and innovative. The most sophisticated techniques are beginning to be used for the electronic tracking of changes to CAD files (see product data management), supplanting the "manual" electronic implementation of traditional revision control.

Source-management models

Traditional revision control systems use a centralized model where all the revision control functions take place on a shared server. If two developers try to change the same file at the same time, without some method of managing access the developers may end up overwriting each other's work. Centralized revision control systems solve this problem in one of two different "source management models": file locking and version merging.

Atomic operations

Main article: Atomic commit

Computer scientists speak of *atomic* operations if the system is left in a consistent state even if the operation is interrupted. The *commit* operation is usually the most critical in this sense. Commits are operations that tell the revision control system you want to make a group of changes final and available to all users. Not all revision control systems have atomic commits; notably, the widely used CVS lacks this feature.

File locking

The simplest method of preventing "concurrent access" problems involves locking files so that only one developer at a time has write access to the central "repository" copies of those files. Once one developer "checks out" a file, others can read that file, but no one else may change that file until that developer "checks in" the updated version (or cancels the checkout).

File locking has both merits and drawbacks. It can provide some protection against difficult merge conflicts when a user is making radical changes to many sections of a large file (or group of files). However, if the files are left exclusively locked for too long, other developers may be tempted to bypass the revision control software and change the files locally, leading to more serious problems.

Version merging

Main article: Merge (revision control)

Most version control systems allow multiple developers to edit the same file at the same time. The first developer to "check in" changes to the central repository always succeeds. The system may provide facilities to merge further changes into the central repository, and preserve the changes from the first developer when other developers check in.

Merging two files can be a very delicate operation, and usually possible only if the data structure is simple, as in text files. The result of a merge of two image files might not result in an image file at all. The second developer checking in code will need to take care with the merge, to make sure that the changes are compatible and that the merge operation does not introduce its own logic errors within the files. These problems limit the availability of automatic or semi-automatic merge operations mainly to simple text based documents, unless a specific merge plugin is available for the file types.

The concept of a *reserved edit* can provide an optional means to explicitly lock a file for exclusive write access, even when a merging capability exists.

Baselines, labels and tags

Most revision control tools will use only one of these similar terms (baseline, label, tag) to refer to the action of identifying a snapshot ("label the project") or the record of the snapshot ("try it with baseline *X*"). Typically only one of the terms *baseline*, *label*, or *tag* is used in documentation or discussionWikipedia:Citation needed; they can be considered synonyms.

In most projects some snapshots are more significant than others, such as those used to indicate published releases, branches, or milestones.

When both the term *baseline* and either of *label* or *tag* are used together in the same context, *label* and *tag* usually refer to the mechanism within the tool of identifying or making the record of the snapshot, and *baseline* indicates the increased significance of any given label or tag.

Most formal discussion of configuration management uses the term *baseline*.

Distributed revision control

Main article: Distributed revision control

Distributed revision control systems (DRCS) take a peer-to-peer approach, as opposed to the client-server approach of centralized systems. Rather than a single, central repository on which clients synchronize, each peer's working copy of the codebase is a bona-fide repository. Distributed revision control conducts synchronization by exchanging patches (change-sets) from peer to peer. This results in some important differences from a centralized system:

- No canonical, reference copy of the codebase exists by default; only working copies.
- Common operations (such as commits, viewing history, and reverting changes) are fast, because there is no need to communicate with a central server.

Rather, communication is only necessary when pushing or pulling changes to or from other peers.

- Each working copy effectively functions as a remote backup of the codebase and of its change-history, providing inherent protection against data loss.

Integration

Some of the more advanced revision-control tools offer many other facilities, allowing deeper integration with other tools and software-engineering processes. Plugins are often available for IDEs such as Oracle JDeveloper, IntelliJ IDEA, Eclipse and Visual Studio. NetBeans IDE and Xcode come with integrated version control support.

Common vocabulary

Terminology can vary from system to system, but some terms in common usage include:[6]Wikipedia:Citing sources

Baseline

: An approved revision of a document or source file from which subsequent changes can be made. See baselines, labels and tags.

Branch

: A set of files under version control may be *branched* or *forked* at a point in time so that, from that time forward, two copies of those files may develop at different speeds or in different ways independently of each other.

Change

: A *change* (or *diff*, or *delta*) represents a specific modification to a document under version control. The granularity of the modification considered a change varies between version control systems.

Change list

: On many version control systems with atomic multi-change commits, a *change list*, *change set*, *update*, or *patch* identifies the set of *changes* made in a single commit. This can also represent a sequential view of the source code, allowing the examination of source "as of" any particular changelist ID.

Checkout

: To *check out* (or *co*) is to create a local working copy from the repository. A user may specify a specific revision or obtain the latest. The term 'checkout' can also be used as a noun to describe the working copy.

Commit

: To *commit* (*check in*, *ci* or, more rarely, *install*, *submit* or *record*) is to write or merge the changes made in the working copy back to the repository. The terms 'commit' and 'checkin' can also be used as nouns to describe the new revision that is created as a result of committing.

Conflict

: A conflict occurs when different parties make changes to the same document, and the system is unable to reconcile the changes. A user must *resolve* the conflict by combining the changes, or by selecting one change in favour of the other.

Delta compression

: Most revision control software uses delta compression, which retains only the differences between successive versions of files. This allows for more efficient storage of many different versions of files.

Dynamic stream

: A stream in which some or all file versions are mirrors of the parent stream's versions.

Export

: *exporting* is the act of obtaining the files from the repository. It is similar to *checking out* except that it creates a clean directory tree without the version-control metadata used in a working copy. This is often used prior to publishing the contents, for example.

Head

Also sometimes called *tip*, this refers to the most recent commit, either to the trunk or to a branch. The trunk and each branch have their own head, though HEAD is sometimes loosely used to refer to the trunk.

Import

importing is the act of copying a local directory tree (that is not currently a working copy) into the repository for the first time.

Label

See *tag*.

Mainline

Similar to *trunk*, but there can be a mainline for each branch.

Merge

A *merge* or *integration* is an operation in which two sets of changes are applied to a file or set of files. Some sample scenarios are as follows:

- A user, working on a set of files, *updates* or *syncs* their working copy with changes made, and checked into the repository, by other users.[7]
- A user tries to *check in* files that have been updated by others since the files were *checked out*, and the *revision control software* automatically merges the files (typically, after prompting the user if it should proceed with the automatic merge, and in some cases only doing so if the merge can be clearly and reasonably resolved).
- A set of files is *branched*, a problem that existed before the branching is fixed in one branch, and the fix is then merged into the other branch.
- A *branch* is created, the code in the files is independently edited, and the updated branch is later incorporated into a single, unified *trunk*.

Promote

The act of copying file content from a less controlled location into a more controlled location. For example, from a user's workspace into a repository, or from a stream to its parent.

Repository

The *repository* is where files' current and historical data are stored, often on a server. Sometimes also called a *depot* (for example, by SVK, AccuRev and Perforce).

Resolve

The act of user intervention to address a conflict between different changes to the same document.

Reverse integration

The process of merging different team branches into the main trunk of the versioning system.

Revision

Also *version*: A version is any change in form. In SVK, a Revision is the state at a point in time of the entire tree in the repository.

Share

The act of making one file or folder available in multiple branches at the same time. When a shared file is changed in one branch, it is changed in other branches.

Stream

A container for branched files that has a known relationship to other such containers. Streams form a hierarchy; each stream can inherit various properties (like versions, namespace, workflow rules, subscribers, etc.) from its parent stream.

Tag

A *tag* or *label* refers to an important snapshot in time, consistent across many files. These files at that point may all be tagged with a user-friendly, meaningful name or revision number. See baselines, labels and tags.

Trunk

The unique line of development that is not a branch (sometimes also called Baseline, Mainline or Master)

Update

An *update* (or *sync*) merges changes made in the repository (by other people, for example) into the local *working copy*.[7] *Update* is also the term used by some CM tools (CM+, PLS, SMS) for the change package concept (see *changelist*).

Working copy

The *working copy* is the local copy of files from a repository, at a specific time or revision. All work done to the files in a repository is initially done on a working copy, hence the name. Conceptually, it is a *sandbox*.

Notes

[1] In this case, edit buffers are a secondary form of working copy, and not referred to as such.

[2] In principle two revisions can have identical timestamp, and thus cannot be ordered on a line. This is generally the case for separate repositories, though is also possible for simultaneous changes to several branches in a single repository. In these cases, the revisions can be thought of as a set of separate lines, one per repository or branch (or branch within a repository).

[3] The revision or repository "tree" should not be confused with the directory tree of files in a working copy.

[4] Note that if a new branch is based on HEAD, then topologically HEAD is no longer a tip, since it has a child.

[5] "Mainline" can also refer to the main path in a separate branch.

[6] Collins-Sussman, Fitzpatrick & Pilato 2004.

[7] Collins-Sussman, Fitzpatrick & Pilato 2004, 1.5: SVN tour cycle resolve (http://svnbook.red-bean.com/en/1.5/svn.tour.cycle.html#svn.tour.cycle.resolve): 'The G stands for merGed, which means that the file had local changes to begin with, but the changes coming from the repository didn't overlap with the local changes.'

References

Bibliography

- Collins-Sussman, Ben; Fitzpatrick, BW; Pilato, CM (2004), *Version Control with Subversion* (http://svnbook.red-bean.com/), O'Reilly, ISBN 0-596-00448-6

External links

- "Visual Guide to Version Control" (http://betterexplained.com/articles/a-visual-guide-to-version-control/), *Better explained*.
- "Comparison" (http://better-scm.berlios.de/comparison/), *Better SCM Initiative*, DE: Berlios. A useful summary of different systems and their features.
- Sink, Eric, "Source Control" (http://www.ericsink.com/scm/source_control.html), *SCM* (how-to). The basics of version control.

Continuous integration

Continuous integration (**CI**) is the practice, in software engineering, of merging all developer working copies with a shared mainline several times a day. It was first named and proposed as part of extreme programming (XP). Its main aim is to prevent integration problems, referred to as "integration hell" in early descriptions of XP. CI can be seen as an intensification of practices of periodic integration advocated by earlier published methods of incremental and iterative software development, such as the Booch method. CI isn't universally accepted as an improvement over frequent integration, so it is important to distinguish between the two as there is disagreement about the virtues of each.Wikipedia:Citation needed

CI was originally intended to be used in combination with automated unit tests written through the practices of test-driven development. Initially this was conceived of as running all unit tests and verifying they all passed before committing to the mainline. This helps avoid one developer's work in progress breaking another developer's copy. If necessary, partially complete features can be disabled before committing using feature toggles.

Later elaborations of the concept introduced build servers, which automatically run the unit tests periodically or even after every commit and report the results to the developers. The use of build servers (not necessarily running unit tests) had already been practised by some teams outside the XP community. Nowadays, many organisations have adopted CI without adopting all of XP.

In addition to automated unit tests, organisations using CI typically use a build server to implement *continuous* processes of applying quality control in general — small pieces of effort, applied frequently. In addition to running the unit and integration tests, such processes run additional static and dynamic tests, measure and profile performance, extract and format documentation from the source code and facilitate manual QA processes. This continuous application of quality control aims to improve the quality of software, and to reduce the time taken to deliver it, by replacing the traditional practice of applying quality control *after* completing all development. This is very similar to the original idea of integrating more frequently to make integration easier, only applied to QA processes.

In the same vein the practice of continuous delivery further extends CI by making sure the software checked in on the mainline is always in a state that can be deployed to users and makes the actual deployment process very rapid.

Theory

When embarking on a change, a developer takes a copy of the current code base on which to work. As other developers submit changed code to the source code repository, this copy gradually ceases to reflect the repository code. Not only can the existing code base change, but new code can be added as well as new libraries, and other resources that create dependencies, and potential conflicts.

The longer a branch of code remains checked out, the greater the risk of multiple integration conflicts and failures when the developer branch is reintegrated into the main line. When developers submit code to the repository they must first update their code to reflect the changes in the repository since they took their copy. The more changes the repository contains, the more work developers must do before submitting their own changes.

Eventually, the repository may become so different from the developers' baselines that they enter what is sometimes referred to as "merge hell", or "integration hell", where the time it takes to integrate exceeds the time it took to make their original changes. In a worst-case scenario, developers may have to discard their changes and completely redo the work.

Continuous integration involves integrating early and often, so as to avoid the pitfalls of "integration hell". The practice aims to reduce rework and thus reduce cost and time.

A complementary practice to CI is that before submitting work, each programmer must do a complete build and run (and pass) all unit tests. Integration tests are usually run automatically on a CI server when it detects a new commit.

All programmers should start the day by updating the project from the repository. That way, they will all stay up-to-date.

The rest of this article discusses best practice in how to achieve continuous integration, and how to automate this practice. Build automation is a best practice itself.

Principles

Continuous integration – the practice of frequently integrating one's new or changed code with the existing code repository – should occur frequently enough that no intervening window remains between commit and build, and such that no errors can arise without developers noticing them and correcting them immediately. Normal practice is to trigger these builds by every commit to a repository, rather than a periodically scheduled build. The practicalities of doing this in a multi-developer environment of rapid commits are such that it's usual to trigger a short time after each commit, then to start a build when either this timer expires, or after a rather longer interval since the last build. Many automated tools offer this scheduling automatically.

Another factor is the need for a version control system that supports atomic commits, i.e. all of a developer's changes may be seen as a single commit operation. There is no point in trying to build from only half of the changed files.

To achieve these objectives, continuous integration relies on the following principles.

Maintain a code repository

Main article: Revision control

This practice advocates the use of a revision control system for the project's source code. All artifacts required to build the project should be placed in the repository. In this practice and in the revision control community, the convention is that the system should be buildable from a fresh checkout and not require additional dependencies. Extreme Programming advocate Martin Fowler also mentions that where branching is supported by tools, its use should be minimized. Instead, it is preferred for changes to be integrated rather than for multiple versions of the software to be maintained simultaneously. The mainline (or trunk) should be the place for the working version of the software.

Automate the build

Main article: Build automation

A single command should have the capability of building the system. Many build-tools, such as make, have existed for many years. Other more recent tools are frequently used in continuous integration environments. Automation of the build should include automating the integration, which often includes deployment into a production-like environment. In many cases, the build script not only compiles binaries, but also generates documentation, website pages, statistics and distribution media (such as Debian DEB, Red Hat RPM or Windows MSI files).

Make the build self-testing

Once the code is built, all tests should run to confirm that it behaves as the developers expect it to behave.

Everyone commits to the baseline every day

By committing regularly, every committer can reduce the number of conflicting changes. Checking in a week's worth of work runs the risk of conflicting with other features and can be very difficult to resolve. Early, small conflicts in an area of the system cause team members to communicate about the change they are making. Committing all changes at least once a day (once per feature built) is generally considered part of the definition of Continuous Integration. In addition performing a nightly build is generally recommended.Wikipedia:Citation needed These are lower bounds, the typical frequency is expected to be much higher.

Every commit (to baseline) should be built

The system should build commits to the current working version in order to verify that they integrate correctly. A common practice is to use Automated Continuous Integration, although this may be done manually. For manyWikipedia:Avoid weasel words, continuous integration is synonymous with using Automated Continuous Integration where a continuous integration server or daemon monitors the revision control system for changes, then automatically runs the build process.

Keep the build fast

The build needs to complete rapidly, so that if there is a problem with integration, it is quickly identified.

Test in a clone of the production environment

Having a test environment can lead to failures in tested systems when they deploy in the production environment, because the production environment may differ from the test environment in a significant way. However, building a replica of a production environment is cost prohibitive. Instead, the pre-production environment should be built to be a scalable version of the actual production environment to both alleviate costs while maintaining technology stack composition and nuances.

Make it easy to get the latest deliverables

Making builds readily available to stakeholders and testers can reduce the amount of rework necessary when rebuilding a feature that doesn't meet requirements. Additionally, early testing reduces the chances that defects survive until deployment. Finding errors earlier also, in some cases, reduces the amount of work necessary to resolve them.

Everyone can see the results of the latest build

It should be easy to find out whether the build breaks and, if so, who made the relevant change.

Automate deployment

Most CI systems allow the running of scripts after a build finishes. In most situations, it is possible to write a script to deploy the application to a live test server that everyone can look at. A further advance in this way of thinking is Continuous deployment, which calls for the software to be deployed directly into production, often with additional automation to prevent defects or regressions.

History

In 1997, Kent Beck and Ron Jeffries invented Extreme Programming (XP) while on the Chrysler Comprehensive Compensation System project, including continuous integration. Beck published about continuous integration in 1998, emphasising the importance of face-to-face communication over technological support. In 1999, Beck elaborated more in his first full book on Extreme Programming. CruiseControl was released in 2001.

Advantages and disadvantages

Advantages

Continuous integration has many advantages:

- When unit tests fail or a bug emerges, developers might revert the codebase to a bug-free state, without wasting time debugging

- Developers detect and fix integration problems continuously — avoiding last-minute chaos at release dates, (when everyone tries to check in their slightly incompatible versions).
- Early warning of broken/incompatible code
- Early warning of conflicting changes
- Immediate unit testing of all changes
- Constant availability of a "current" build for testing, demo, or release purposes
- Immediate feedback to developers on the quality, functionality, or system-wide impact of code they are writing
- Frequent code check-in pushes developers to create modular, less complex codeWikipedia:Citation needed
- Metrics generated from automated testing and CI (such as metrics for code coverage, code complexity, and features complete) focus developers on developing functional, quality code, and help develop momentum in a teamWikipedia:Citation needed

Disadvantages

- Initial setup time required
- Well-developed test-suite required to achieve automated testing advantages

Many teams using CI report that the advantages of CI well outweigh the disadvantages. The effect of finding and fixing integration bugs early in the development process saves both time and money over the lifespan of a project.

Software

Main article: Comparison of continuous integration software

Software tools for continuous integration include:

- Apache Continuum: continuous integration server supporting Apache Maven and Apache Ant.
- Apache Gump: continuous integration tool.
- AutomatedQA Automated Build Studio: proprietary automated build, continuous integration and release management system
- Atlassian Software Systems Bamboo: proprietary continuous integration server.
- Buildbot: Python/Twisted-based continuous build system.
- BuildHive: free cloud-hosted continuous integration service for GitHub projects, based on Jenkins.
- CABIE (Continuous Automated Build and Integration Environment): open source, written in Perl
- Cascade: proprietary continuous integration tool.
- codeBeamer: proprietary collaboration software with built-in continuous integration features.
- CruiseControl: Java-based framework for a continuous build process and .NET-based automated continuous integration server.
- CruiseControl.rb: lightweight, Ruby-based continuous integration server that can build any codebase, not only Ruby; released under Apache Licence 2.0.
- Electric Cloud ElectricCommander: proprietary continuous integration and release management product.
- FinalBuilder Server: VSoft Technologies proprietary automated build and continuous integration server.
- Hudson: MIT-licensed, written in Java.
- Jenkins (Fork of Hudson): MIT-licensed, written in Java.
- IBM Rational Team Concert: proprietary software development collaboration platform with built-in build engine.
- IBM Rational Software SCLM: software configuration management system for z/OS.
- Microsoft Team Foundation Server: proprietary software development collaboration platform with continuous integration server and source code repository.
- Shippable: a Saas platform that provides Continuous Integration/Deployment to Github repositories.
- Strider [1]: An open source continuous integration and deployment server written in node.js; BSD License
- TeamCity: a closed source, Java-based build management and continuous integration server from JetBrains

- Thoughtworks Go: open source (previously proprietary) agile build and release management software.
- Tinderbox: Mozilla-based product written in Perl.
- Travis CI: a distributed build system for the open source community.
- Xcode 5 also contains a proprietary continuous integration system, works closely with OS X Server for OS X Mavericks.

References

[1] http://stridercd.com

Further reading

- Duvall, Paul M. (2007). *Continuous Integration. Improving Software Quality and Reducing Risk*. Addison-Wesley. ISBN 0-321-33638-0.

External links

- Fowler, Martin. "Continuous integration" (http://www.martinfowler.com/articles/continuousIntegration.html) (article) (introduction).
- "Continuous Integration" (http://www.c2.com/cgi/wiki?ContinuousIntegration) (wiki). *Portland Pattern Repository* (a collegial discussion). C2.
- "Cross platform testing" (http://c2.com/cgi/wiki?CrossPlatformTesting) (wiki). *Portland Pattern Repository*. C2.
- *Continuous Integration Server Feature Matrix (a guide to tools)* (http://confluence.public.thoughtworks.org/display/CC/CI+Feature+Matrix). Thought works.
- Richardson, Jared. "Continuous Integration: The Cornerstone of a Great Shop" (http://www.methodsandtools.com/archive/archive.php?id=42). (introduction).
- Flowers, Jay. "A Recipe for Build Maintainability and Reusability" (http://jayflowers.com/joomla/index.php?option=com_content&task=view&id=26).
- Duvall, Paul. "Continuous Integration anti-patterns" (http://www.ibm.com/developerworks/java/library/j-ap11297/). *Developer works*.
- "Integrate often" (http://www.extremeprogramming.org/rules/integrateoften.html). *Rules*. Extreme programming.
- *Version lifecycle* (http://www.mediawiki.org/wiki/Version_lifecycle). MediaWiki.

Software metric

A **software metric** is a measure of some property of a piece of software or its specifications. Since quantitative measurements are essential in all sciences, there is a continuous effort by computer science practitioners and theoreticians to bring similar approaches to software development. The goal is obtaining objective, reproducible and quantifiable measurements, which may have numerous valuable applications in schedule and budget planning, cost estimation, quality assurance testing, software debugging, software performance optimization, and optimal personnel task assignments.

Common software measurements

Common software measurements include:

- Balanced scorecard
- Bugs per line of code
- Code coverage
- Cohesion
- Comment density
- Connascent software components
- Coupling
- Cyclomatic complexity (McCabe's complexity)
- DSQI (design structure quality index)
- Function point analysis
- Halstead Complexity
- Instruction path length
- Number of classes and interfaces
- Number of lines of code
- Number of lines of customer requirements
- Program execution time
- Program load time
- Program size (binary)
- Robert Cecil Martin's software package metrics
- Weighted Micro Function Points
- Function Points and Automated Function Points, an Object Management Group standard
- CISQ automated quality characteristics measures

Limitations

As software development is a complex process, with high variance on both methodologies and objectives, it is difficult to define or measure software qualities and quantities and to determine a valid and concurrent measurement metric, especially when making such a prediction prior to the detail design. Another source of difficulty and debate is in determining which metrics matter, and what they mean. The practical utility of *software* measurements has thus been limited to narrow domains where they include:

- Schedule
- Size/Complexity
- Cost
- Quality

Common goal of measurement may target one or more of the above aspects, or the balance between them as indicator of team's motivation or project performance.

Acceptance and public opinion

Some software development practitioners point out that simplistic measurements can cause more harm than good. Others have noted that metrics have become an integral part of the software development process. Impact of measurement on programmers psychology have raised concerns for harmful effects to performance due to stress, performance anxiety, and attempts to cheat the metrics, while others find it to have positive impact on developers value towards their own work, and prevent them being undervalued. Some argue that the definition of many measurement methodologies are imprecise, and consequently it is often unclear how tools for computing them arrive at a particular result, while others argue that imperfect quantification is better than none ("You can't control what you can't measure."). Evidence shows that software metrics are being widely used by government agencies, the US military, NASA, IT consultants, academic institutions, and commercial and academic development estimation software.

References

External links

Covers a minimal set of essential metrics for a successful product delivery.

- Definitions of software metrics in .NET (http://www.ndepend.com/Metrics.aspx)
- International Function Point Users Group (http://www.ifpug.org)
- What is FPA (http://www.nesma.org/section/fpa/) at Nesma website
- Further defines the term Software Metrics with examples. (http://www.sqa.net/softwarequalitymetrics.html)
- Software Engineering Metrics: What do they measure and how do we know (http://www.kaner.com/pdfs/metrics2004.pdf) - An intellectually rigorous treatment of software engineering metrics

Code coverage

In computer science, **code coverage** is a measure used to describe the degree to which the source code of a program is tested by a particular test suite. A program with high code coverage has been more thoroughly tested and has a lower chance of containing software bugs than a program with low code coverage. Many different metrics can be used to calculate code coverage; some of the most basic are the percent of program subroutines and the percent of program statements called during execution of the test suite.

Code coverage was among the first methods invented for systematic software testing. The first published reference was by Miller and Maloney in *Communications of the ACM* in 1963.

Coverage criteria

To measure what percentage of code has been exercised by a test suite, one or more *coverage criteria* are used. Coverage criteria is usually defined as a rule or requirement, which test suite needs to satisfy.

Basic coverage criteria

There are a number of coverage criteria, the main ones being:

- **Function coverage** - Has each function (or subroutine) in the program been called?
- **Statement coverage** - Has each statement in the program been executed?
- **Branch coverage** - Has each branch of each control structure (such as in *if* and *case* statements) been executed? For example, given an *if* statement, have both the true and false branches been executed? Another way of saying this is, has every edge in the program been executed?
- **Condition coverage** (or predicate coverage) - Has each Boolean sub-expression evaluated both to true and false?

For example, consider the following C++ function:

```
int foo (int x, int y)
{
    int z = 0;
    if ((x>0) && (y>0))
    {
        z = x;
    }
    return z;
}
```

Assume this function is a part of some bigger program and this program was run with some test suite.

- If during this execution function 'foo' was called at least once, then *function coverage* for this function is satisfied.
- *Statement coverage* for this function will be satisfied if it was called e.g. as `foo(1,1)`, as in this case, every line in the function is executed including `z = x;`.
- Tests calling `foo(1,1)` and `foo(0,1)` will satisfy *branch coverage* because, in the first case, the 2 `if` conditions are met and `z = x;` is executed, while in the second case, the first condition `(x>0)` is not satisfied, which prevents executing `z = x;`.
- *Condition coverage* can be satisfied with tests that call `foo(1,1)`, `foo(1,0)` and `foo(0,0)`. These are necessary because in the first two cases, `(x>0)` evaluates to `true`, while in the third, it evaluates `false`. At the same time, the first case makes `(y>0)` `true`, while the second and third make it `false`.

Condition coverage does not necessarily imply branch coverage. For example, consider the following fragment of code:

```
if a and b then
```

Condition coverage can be satisfied by two tests:

- `a=true`, `b=false`
- `a=false`, `b=true`

However, this set of tests does not satisfy branch coverage since neither case will meet the `if` condition.

Fault injection may be necessary to ensure that all conditions and branches of exception handling code have adequate coverage during testing.

Modified condition/decision coverage

Main article: Modified Condition/Decision Coverage

Combination of function coverage and branch coverage is sometimes also called **decision coverage**. This criteria requires that every point of entry and exit in the program have been invoked at least once, and every decision in the program have taken on all possible outcomes at least once. In this context the decision is a boolean expression composed of conditions and zero or more boolean operators. This definition is not the same as branch coverage,[1] however, some do use the term *decision coverage* as a synonym for *branch coverage*.[2]

Condition/decision coverage requires that both decision and condition coverage been satisfied. However, for safety-critical applications (e.g., for avionics software) it is often required that **modified condition/decision coverage (MC/DC)** be satisfied. This criterion extends condition/decision criteria with requirements that each condition should affect the decision outcome independently. For example, consider the following code:

```
if (a or b) and c then
```

The condition/decision criteria will be satisfied by the following set of tests:

- a=true, b=true, c=true
- a=false, b=false, c=false

However, the above tests set will not satisfy modified condition/decision coverage, since in the first test, the value of 'b' and in the second test the value of 'c' would not influence the output. So, the following test set is needed to satisfy MC/DC:

- a=**false**, b=**false**, c=true
- a=**true**, b=false, c=**true**
- a=false, b=**true**, c=**true**
- a=false, b=true, c=**false**

Multiple condition coverage

This criterion requires that all combinations of conditions inside each decision are tested. For example, the code fragment from the previous section will require eight tests:

- a=false, b=false, c=false
- a=false, b=false, c=true
- a=false, b=true, c=false
- a=false, b=true, c=true
- a=true, b=false, c=false
- a=true, b=false, c=true
- a=true, b=true, c=false
- a=true, b=true, c=true

Parameter value coverage

Parameter value coverage (PVC) requires that in a method taking parameters, all the common values for such parameters been considered. The idea is that all common possible values for a parameter are tested.[3] For example, common values for a string are: 1) null, 2) empty, 3) whitespace (space, tabs, newline), 4) valid string, 5) invalid string, 6) single-byte string, 7) double-byte string. It may also be appropriate to use very long strings. Failure to test each possible parameter value may leave a bug. Testing only one of these could result in 100% code coverage as each line is covered, but as only one of seven options are tested, there is only 14.2% PVC.

Other coverage criteria

There are further coverage criteria, which are used less often:

- **Linear Code Sequence and Jump (LCSAJ) coverage** - has every LCSAJ been executed?
- **JJ-Path coverage** - have all jump to jump paths[4] (aka LCSAJs) been executed?
- **Path coverage** - Has every possible route through a given part of the code been executed?
- **Entry/exit coverage** - Has every possible call and return of the function been executed?
- **Loop coverage** - Has every possible loop been executed zero times, once, and more than once?
- **State coverage** - Has each state in a finite-state machine been reached and explored?

Safety-critical applications are often required to demonstrate that testing achieves 100% of some form of code coverage.

Some of the coverage criteria above are connected. For instance, path coverage implies decision, statement and entry/exit coverage. Decision coverage implies statement coverage, because every statement is part of a branch.

Full path coverage, of the type described above, is usually impractical or impossible. Any module with a succession of n decisions in it can have up to 2^n paths within it; loop constructs can result in an infinite number of paths. Many paths may also be infeasible, in that there is no input to the program under test that can cause that particular path to be executed. However, a general-purpose algorithm for identifying infeasible paths has been proven to be impossible (such an algorithm could be used to solve the halting problem).[5] Methods for practical path coverage testing instead attempt to identify classes of code paths that differ only in the number of loop executions, and to achieve "basis path" coverage the tester must cover all the path classes.

In practice

The target software is built with special options or libraries and/or run under a special environment such that every function that is exercised (executed) in the program(s) is mapped back to the function points in the source code. This process allows developers and quality assurance personnel to look for parts of a system that are rarely or never accessed under normal conditions (error handling and the like) and helps reassure test engineers that the most important conditions (function points) have been tested. The resulting output is then analyzed to see what areas of code have not been exercised and the tests are updated to include these areas as necessary. Combined with other code coverage methods, the aim is to develop a rigorous, yet manageable, set of regression tests.

In implementing code coverage policies within a software development environment, one must consider the following:

- What are coverage requirements for the end product certification and if so what level of code coverage is required? The typical level of rigor progression is as follows: Statement, Branch/Decision, Modified Condition/Decision Coverage(MC/DC), LCSAJ (Linear Code Sequence and Jump)
- Will code coverage be measured against tests that verify requirements levied on the system under test (DO-178B)?
- Is the object code generated directly traceable to source code statements? Certain certifications, (i.e. DO-178B Level A) require coverage at the assembly level if this is not the case: "Then, additional verification should be

performed on the object code to establish the correctness of such generated code sequences" (DO-178B) para-6.4.4.2.

Test engineers can look at code coverage test results to help them devise test cases and input or configuration sets that will increase the code coverage over vital functions. Two common forms of code coverage used by testers are statement (or line) coverage and branch (or edge) coverage. Line coverage reports on the execution footprint of testing in terms of which lines of code were executed to complete the test. Edge coverage reports which branches or code decision points were executed to complete the test. They both report a coverage metric, measured as a percentage. The meaning of this depends on what form(s) of code coverage have been used, as 67% branch coverage is more comprehensive than 67% statement coverage.

Generally, code coverage tools incur computation and logging in addition to the actual program thereby slowing down the application, so typically this analysis is not done in production. As one might expect, there are classes of software that cannot be feasibly subjected to these coverage tests, though a degree of coverage mapping can be approximated through analysis rather than direct testing.

There are also some sorts of defects which are affected by such tools. In particular, some race conditions or similar real time sensitive operations can be masked when run under code coverage environments; and conversely, and reliably, some of these defects may become easier to find as a result of the additional overhead of the testing code.

Usage in industry

Code coverage is one consideration in the safety certification of avionics equipment. The guidelines by which avionics gear is certified by the Federal Aviation Administration (FAA) is documented in DO-178B[] and the recently released DO-178C.[6]

References

[1] Position Paper CAST-10 (June 2002). *What is a "Decision" in Application of Modified Condition/Decision Coverage (MC/DC) and Decision Coverage (DC)?* (http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/media/cast-10.pdf)

[2] MathWorks. *Types of Model Coverage.* (http://www.mathworks.com/help/slvnv/ug/types-of-model-coverage.html)

[3] Unit Testing with Parameter Value Coverage (PVC) (http://www.rhyous.com/2012/05/08/unit-testing-with-parameter-value-coverage-pvc/)

[4] M. R. Woodward, M. A. Hennell, "On the relationship between two control-flow coverage criteria: all JJ-paths and MCDC", Information and Software Technology 48 (2006) pp. 433-440

[5] Dorf, Richard C.: *Computers, Software Engineering, and Digital Devices*, Chapter 12, pg. 15. CRC Press, 2006. ISBN 0-8493-7340-9, ISBN 978-0-8493-7340-4; via Google Book Search (http://books.google.com/books?id=jykvlTCoksMC&pg=PT386&lpg=PT386&dq="infeasible+path"+"halting+problem"&source=web&ots=WUWz3qMPRv&sig=dSAjrLHBSZJcKWZfGa_IxYlfSNA&hl=en&sa=X&oi=book_result&resnum=1&ct=result)

[6] RTCA/DO-178C, *Software Considerations in Airborne Systems and Equipment Certification, Radio Technical Commission for Aeronautics,* January, 2012.

Technical debt

Technical debt (also known as **design debt**Wikipedia:Citation needed or **code debt**) is a neologistic metaphor referring to the eventual consequences of poor software architecture and software development within a codebase. The debt can be thought of as work that needs to be done before a particular job can be considered complete. If the debt is not repaid, then it will keep on accumulating interest, making it hard to implement changes later on. Unaddressed technical debt increases software entropy.

As a change is started on a codebase, there is often the need to make other coordinated changes at the same time in other parts of the codebase or documentation. The other required, but uncompleted changes, are considered debt that must be paid at some point in the future. Just like financial debt, these uncompleted changes incur interest on top of interest, making it cumbersome to build a project. Although the term is used in software development primarily, it can also be applied to other professions.

Common causes of technical debt include (a combination of):

- **Business pressures**, where the business considers getting something released sooner before all of the necessary changes are complete, builds up technical debt comprising those uncompleted changes.
- **Lack of process or understanding**, where businesses are blind to the concept of technical debt, and make decisions without considering the implications.
- **Lack of building loosely coupled components**, where functions are not modular, the software is not flexible enough to adapt to changes in business needs.
- **Lack of test suite**, which encourages quick and risky band-aids to fix bugs.
- **Lack of documentation**, where code is created without necessary supporting documentation. That work to create the supporting documentation represents a debt that must be paid.
- **Lack of collaboration**, where knowledge isn't shared around the organization and business efficiency suffers, or junior developers are not properly mentored
- **Parallel development** at the same time on two or more branches can cause the buildup of technical debt because of the work that will eventually be required to merge the changes into a single source base. The more changes that are done in isolation, the more debt that is piled up.
- **Delayed refactoring –** As the requirements for a project evolve, it may become clear that parts of the code have become unwieldy and must be refactored in order to support future requirements. The longer that refactoring is delayed, and the more code is written to use the current form, the more debt that piles up that must be paid at the time the refactoring is finally done.
- **Lack of knowledge**, when the developer simply doesn't know how to write elegant code.

"Interest payments" are both in the necessary local maintenance and the absence of maintenance by other users of the project. Ongoing development in the upstream project can increase the cost of "paying off the debt" in the future.Wikipedia:Please clarify One pays off the debt by simply completing the uncompleted work.

The buildup of technical debt is a major cause for projects to miss deadlines.Wikipedia:Citation needed It is difficult to estimate exactly how much work is necessary to pay off the debt. For each change that is initiated, an uncertain amount of uncompleted work is committed to the project. The deadline is missed when the project realizes that there is more uncompleted work (debt) than there is time to complete it in. To have predictable release schedules, a development team should limit the amount of work in progress in order to keep the amount of uncompleted work (or debt) small at all times.

> "As an evolving program is continually changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it."
>
> — Meir Manny Lehman, 1980

While Manny Lehman's Law already indicated that evolving programs continually add to their complexity and deteriorating structure unless work is done to maintain it, Ward Cunningham first drew the comparison between technical complexity and debt in a 1992 experience report:

> "Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite... The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise."
>
> — Ward Cunningham, 1992

In his 2004 text, *Refactoring to Patterns*, Joshua Kerievsky presents a comparable argument concerning the costs associated with architectural negligence, which he describes as "design debt".

Activities that might be postponed include documentation, writing tests, attending to TODO comments and tackling compiler and static code analysis warnings. Other instances of technical debt include knowledge that isn't shared around the organization and code that is too confusing to be modified easily.

In open source software, postponing sending local changes to the upstream project is a technical debt.

References

External links

- *Ward Explains Debt Metaphor* (http://c2.com/cgi/wiki?WardExplainsDebtMetaphor), video from Ward Cunningham
- OnTechnicalDebt (http://www.ontechnicaldebt.com) The online community for discussing technical debt
- TechDebt (http://www.techdebt.org) The first collaborative & live benchmark on technical debt and software quality
- Experts interviews on Technical Debt: Ward Cunningham (http://blog.techdebt.org/resources-links/67/ward-cunningham-interview-about-technical-debt-sqale-agile)Wikipedia:Link rot, Philippe KRUCHTEN (http://blog.techdebt.org/interviews/156/interview-with-philippe-kruchten-on-technical-debt-rup-ubc-decision-process-architecture)Wikipedia:Link rot, Ipek OZKAYA (http://blog.techdebt.org/interviews/189/technical-debt-interview-with-ipek-ozkaya-on-technical-debt-sei-ieee-software-architecture-agile)Wikipedia:Link rot, Jean-Louis LETOUZEY (http://blog.techdebt.org/interviews/118/interview-with-jean-louis-letouzey-on-technical-debt-and-sqale)Wikipedia:Link rot
- Steve McConnell discusses technical debt (http://forums.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx)
- TechnicalDebt (http://www.martinfowler.com/bliki/TechnicalDebt.html) from Martin Fowler Bliki
- An Andy Lester talk entitled (http://www.media-landscape.com/yapc/2006-06-26.AndyLester/) "Get out of Technical Debt Now!"
- Lehman's Law (http://www.inf.ed.ac.uk/teaching/courses/rtse/Lectures/lehmanslaws.pdf)
- Limited WIP Society (http://www.limitedwipsociety.org/) website discusses techniques to avoid building up technical debt
- Managing Technical Debt Webinar by Steve McConnell (http://www.youtube.com/watch?v=lEKvzEyNtbk)

Jenkins (software)

Jenkins (software)

Jenkins	
Initial release	2 February 2011[1]
Stable release	1.563 / 11 May 2014
Written in	Java
Operating system	Cross-platform
Type	Continuous integration
License	MIT license
Website	jenkins-ci.org [2]

Jenkins is an open source continuous integration tool written in Java. The project was forked from Hudson after a dispute with Oracle.

Jenkins provides continuous integration services for software development. It is a server-based system running in a servlet container such as Apache Tomcat. It supports SCM tools including AccuRev, CVS, Subversion, Git, Mercurial, Perforce, Clearcase and RTC, and can execute Apache Ant and Apache Maven based projects as well as arbitrary shell scripts and Windows batch commands. The primary developer of Jenkins is Kohsuke Kawaguchi. Released under the MIT License, Jenkins is free software.

Builds can be started by various means, including being triggered by commit in a version control system, scheduling via a cron-like mechanism, building when other builds have completed, and by requesting a specific build URL.

History

Jenkins was originally developed as the Hudson project. Hudson's creation started in summer of 2004 at Sun Microsystems. It was first released in java.net in Feb. 2005.

Around 2007 Hudson became known as a better alternative to CruiseControl and other open-source build-servers. At the JavaOne conference in May 2008 the software won the Duke's Choice Award in the Developer Solutions category.

During November 2010, an issue arose in the Hudson community with respect to the infrastructure used, which grew to encompass questions over the stewardship and control by Oracle. Negotiations between the principal project contributors and Oracle took place, and although there were many areas of agreement a key sticking point was the trademarked name "Hudson", after Oracle claimed the right to the name and applied for a trademark in December 2010. As a result, on January 11, 2011, a call for votes was made to change the project name from "Hudson" to "Jenkins". The proposal was overwhelmingly approved by community vote on January 29, 2011, creating the Jenkins project.

On February 1, 2011, Oracle said that they intended to continue development of Hudson, and considered Jenkins a fork rather than a rename. Jenkins and Hudson therefore continue as two independent projects, each claiming the other is the fork. As of December 2013, the Jenkins organisation on GitHub had 567 project members and around 1,100 public repositories, compared with Hudson's 32 project members and 17 public repositories.

In 2011, creator Kohsuke Kawaguchi received a Google-O'Reilly Open Source Award for his work on the Hudson/Jenkins project. In 2014, Kawaguchi became the Chief Technology Officer for CloudBees.

Plugins

Plugins have been released for Jenkins that extend its use to projects written in languages other than Java.[3] Plugins are available for integrating Jenkins with most version control systems and big databases. Many build tools are supported via their respective plugins. Plugins can also change the way Jenkins looks or add new functionality.

Builds can generate test reports in various formats (JUnit is supported out-of-the-box, others via plugins) and Jenkins can display the reports and generate trends and render them in the GUI.

References

[1] Jenkins 1.396 released (http://jenkins.361315.n4.nabble.com/Jenkins-1-396-released-td3257106.html), *The first release of Jenkins is posted*, Kohsuke Kawaguchi

[2] http://jenkins-ci.org/

[3] Plugins - Jenkins (https://wiki.jenkins-ci.org/display/JENKINS/Plugins)

External links

- Jenkins homepage (http://jenkins-ci.org/)
- Jenkins tutorial video (http://www.lalitbhatt.com/Jenkins)
- Hudson free book (http://www.eclipse.org/hudson/the-hudson-book/book-hudson.pdf)
- Jenkins all-in-one installer (http://bitnami.org/stack/jenkins), virtual machine and cloud images by BitNami
- Hudson and CI related articles (http://adrian.org.ar/tag/hudson)
- Jenkins creator Kohsuke Kawaguchi on The Changelog podcast talking about the project origin and name change (http://thechangelog.com/post/3186867001/episode-0-4-8-jenkins-formerly-hudson-with-kohsuke-kawag)
- *7 Ways to Optimize Jenkins/Hudson* (http://www.cloudbees.com/sites/default/files/whitepapers/7WaysToOptimizeJenkins.pdf) from founder Kohsuke Kawaguchi (http://www.cloudbees.com/sites/default/files/whitepapers/7WaysToOptimizeJenkins.pdf)

Apache Ant

Apache Ant (Another Neat Tool)

Developer(s)	Apache Software Foundation
Initial release	July 2000
Stable release	1.9.3 / December 29, 2013
Written in	Java
Operating system	Cross-platform
Type	Build tool
License	Apache License 2.0
Website	ant.apache.org [1]

Apache Ant is a software tool for automating software build processes. It is similar to Make but is implemented using the Java language, requires the Java platform, and is best suited to building Java projects.

The most immediately noticeable difference between Ant and Make is that Ant uses XML to describe the build process and its dependencies, whereas Make uses Makefile format. By default the XML file is named `build.xml`.

Ant is an Apache project. It is open source software, and is released under the Apache License.

History

Ant ("Another Neat Tool"[2]) was conceived by James Duncan Davidson while preparing Sun's reference JSP/Servlet engine, later Apache Tomcat, for release as open source. A proprietary version of *make* was used to build it on the Solaris Operating Environment, but in the open source world there was no way of controlling which platform was used to build Tomcat; so Ant was created as a simple platform-independent tool to build Tomcat from directives in an XML "build file". Ant (version 1.1) was officially released as a stand-alone product on July 19, 2000.

Several proposals for an Ant version 2 have been made, such as AntEater by James Duncan Davidson, Myrmidon by Peter Donald and Mutant by Conor MacNeill, none of which were able to find large acceptance with the developer community.[3]

At one time (2002), Ant was the build tool used by most Java development projects.[4] For example, most open source Java developers include build.xml files with their distribution.Wikipedia:Citation needed

Because Ant made it trivial to integrate JUnit tests with the build process, Ant made it easy for willing developers to adopt test-driven development, and even Extreme Programming.

Sample `build.xml` file

Below is listed a sample build.xml file for a simple Java "Hello, world" application. It defines four targets - *clean*, *clobber*, *compile* and *jar*, each of which has an associated description. The *jar* target lists the *compile* target as a dependency. This tells Ant that before it can start the *jar* target it must first complete the *compile* target.

```
<?xml version="1.0"?>
<project name="Hello" default="compile">
    <target name="clean" description="remove intermediate files">
        <delete dir="classes"/>
    </target>
    <target name="clobber" depends="clean" description="remove all artifact files">
        <delete file="hello.jar"/>
    </target>
    <target name="compile" description="compile the Java source code to class files">
        <mkdir dir="classes"/>
        <javac srcdir="." destdir="classes"/>
    </target>
    <target name="jar" depends="compile" description="create a Jar file for the application">
        <jar destfile="hello.jar">
            <fileset dir="classes" includes="**/*.class"/>
            <manifest>
                <attribute name="Main-Class" value="HelloProgram"/>
            </manifest>
        </jar>
    </target>
</project>
```

Within each target are the actions that Ant must take to build that target; these are performed using built-in *tasks*. For example, to build the *compile* target Ant must first create a directory called *classes* (which Ant will do only if it does not already exist) and then invoke the Java compiler. Therefore, the *tasks* used are *mkdir* and *javac*. These perform a similar task to the command-line utilities of the same name.

Another task used in this example is named *jar*:

```
 <jar destfile="hello.jar">
```

This Ant task has the same name as the common Java command-line utility, JAR, but is really a call to the Ant program's built-in JAR/ZIP file support. This detail is not relevant to most end users, who just get the JAR they wanted, with the files they asked for.

Many Ant tasks delegate their work to external programs, either native or Java. They use Ant's own <exec> and <java> tasks to set up the command lines, and handle all the details of mapping from information in the build file to the program's arguments and interpreting the return value. Users can see which tasks do this (e.g. <cvs>, <signjar>, <chmod>, <rpm>), by trying to execute the task on a system without the underlying program on the path, or without a full Java Development Kit (JDK) installed.

Extensions

WOProject-Ant[5] is just one of many examples of a *task* extension written for Ant. These extensions are put to use by copying their jar files into ant's *lib* directory. Once this is done, these extension tasks can be invoked directly in the typical *build.xml* file. The WOProject extensions allow WebObjects developers to use ant in building their frameworks and applications, instead of using Apple's Xcode suite.

Antcontrib[6] provides a collection of tasks such as conditional statements and operations on properties as well as other useful tasks.[7]

Ant-contrib.unkrig.de[8] implements tasks and types for networking, Swing user interfaces, JSON processing and other.

Other task extensions exist for Perforce, .Net, EJB, and filesystem manipulations, just to name a few.[9]

Portability

One of the primary aims of Ant was to solve make's portability problems. The first portability issue in a Makefile is that the actions required to create a target are specified as shell commands which are specific to the platform on which Make runs. Different platforms require different shell commands. Ant solves this problem by providing a large amount of built-in functionality that is designed to behave the same on all platforms. For example, in the sample `build.xml` file above the *clean* target deletes the `classes` directory and everything in it. In a Makefile this would typically be done with the command:

```
rm -rf classes/
```

`rm` is a Unix-specific command unavailable in some other environments. Microsoft Windows, for example, would use:

```
rmdir /S /Q classes
```

In an Ant build file the same goal would be accomplished using a built-in command:

```
<delete dir="classes"/>
```

A second portability issue is a result of the fact that the symbol used to delimit elements of file system directory path components differs from one platform to another. Unix uses a forward slash (/) to delimit components whereas Windows uses a backslash (\). Ant build files let authors choose their favorite convention: forward slash or backslash for directories; semicolon or colon for path separators. It converts each to the symbol appropriate to the platform on which it executes.

Limitations

- Ant build files, which are written in XML, can be complex and verbose. The complex structure (hierarchical, partly ordered, and pervasively cross-linked) of Ant documents can be a barrier to learning. (A GUI called Antidote was available for a time, but never gained a following and has been retired from the Apache project.) The build files of large or complex projects can become unmanageably large. Good design and modularization of build files can improve readability but not necessarily reduce size. Other build tools, such as Maven, use more concise scripts at the expense of generality and flexibility.
- Many of the older tasks—the core ones that are used every day, such as `<javac>`, `<exec>` and `<java>`—use default values for options that are not consistent with more recent versions of the tasks. Changing those defaults would break existing Ant scripts.
- When expanding properties in a string or text element, undefined properties are not raised as an error, but left as an unexpanded reference (e.g. `${unassigned.property}`).

- Ant has limited fault handling rules, and no persistence of state, so it cannot be used as a workflow tool for any workflow other than classic build and test processes.
- Lazy property evaluation is not supported. For instance, when working within an Antcontrib <for> loop, a property cannot be re-evaluated for a sub-value which may be part of the iteration. (Some third-party extensions facilitate a workaround; AntXtras flow-control tasksets do provide for cursor redefinition for loops.)
- In makefiles, any rule to create one file type from another can be written inline within the makefile. For example, you may transform a document into some other format by using rules to execute another tool. Creating a similar task in Ant is more complex: a separate task must be written in Java and included with the Ant build file in order to handle the same type of functionality. However, this separation can enhance the readability of the Ant script by hiding some of the details of how a task is executed on different platforms.

There exists a myriad of third-party Ant extensions (called *antlibs*) that provide much of the missing functionality. Also, the Eclipse IDE can build and execute Ant scripts, while the NetBeans IDE uses Ant for its internal build system. As both these IDEs are very popular development platforms, they can simplify Ant use significantly. (As a bonus, Ant scripts generated by NetBeans can be used outside that IDE as standalone scripts.)

References

[1] http://ant.apache.org
[2] *Why do you call it Ant?* (http://ant.apache.org/faq.html#ant-name), Apache Ant FAQ
[3] Conor MacNeill -- The Early History of Ant Development (http://codefeed.com/blog/?p=98)
[4] Java Tools for eXtreme Programming, Wiley, 2002: 76
[5] WOProject-Ant - WOProject / WOLips - Confluence (http://www.objectstyle.org/confluence/display/WOL/WOProject-Ant)
[6] Ant-Contrib (http://ant-contrib.sourceforge.net)
[7] Ant-Contrib Tasks (http://ant-contrib.sourceforge.net/tasks/tasks/index.html)
[8] ant-contrib.unkrig.de (http://ant-contrib.unkrig.de)
[9] Overview of Ant Tasks (http://ant.apache.org/manual/tasksoverview.html)

Bibliography

- Loughran, Steve; Hatcher, Erik (July 12, 2007). *Ant in Action* (2nd ed.). Manning Publications. p. 600. ISBN 978-1-932394-80-1.
- Holzner, Steven (April 13, 2005). *Ant - The Definitive Guide* (http://oreilly.com/catalog/9780596006099/) (2nd ed.). O'Reilly Media. p. 334. ISBN 978-0-596-00609-9.
- Moodie, Matthew (November 16, 2005). *Pro Apache Ant* (http://www.apress.com/book/view/9781590595596) (1st ed.). Apress. p. 360. ISBN 978-1-59059-559-6.
- Bell, Alexis T. (July 7, 2005). *ANT Java Notes: An Accelerated Intro Guide to the Java ANT Build Tool* (http://www.virtualbookworm.com/mm5/merchant.mvc?Screen=PROD&Store_Code=bookstore&Product_Code=antjava) (1st ed.). Virtualbookworm.com Publishing. p. 268. ISBN 978-1-58939-738-5.
- Hatcher, Erik; Loughran, Steve (August 2002). *Java Development with Ant* (1st ed.). Manning Publications. p. 672. ISBN 978-1-930110-58-8.
- Niemeyer, Glenn; Poteet, Jeremy (May 29, 2003). *Extreme Programming with Ant: Building and Deploying Java Applications with JSP, EJB, XSLT, XDoclet, and JUnit* (http://www.informit.com/store/product.aspx?isbn=0672325624) (1st ed.). SAMS Publishing. p. 456. ISBN 978-0-672-32562-5.
- Williamson, Alan (November 1, 2002). *Ant - Developer's Handbook* (http://www.informit.com/store/product.aspx?isbn=0672324261) (1st ed.). SAMS Publishing. p. 456. ISBN 978-0-672-32426-0.
- Matzke, Bernd (September 2003). *ANT: The Java Build Tool In Practice* (http://www.powells.com/biblio?isbn=9781584502487) (1st ed.). Charles River Media. p. 280. ISBN 978-1-58450-248-7.

External links

- Official website (http://ant.apache.org)
- Apache Ant manual (http://ant.apache.org/manual/) (tasks (http://ant.apache.org/manual/tasklist.html), types (http://ant.apache.org/manual/conceptstypeslist.html)).
- Apache Ant wiki (http://wiki.apache.org/ant/FrontPage).
- WinAnt - Windows installer for Apache Ant (http://code.google.com/p/winant/).
- Introduction to Ant (http://www.exubero.com/ant/antintro-s5.html) (slide show).
- Linguine Maps visualization library will automatically produce easy to read diagrams from Ant build files. (http://www.softwaresecretweapons.com/jspwiki/Wiki.jsp?page=LinguineMapsForApacheAnt)
- antro - a profiler for Ant scripts (http://sourceforge.net/projects/antro).
- Wiki Book on learning Apache Ant.
- Ant tutorial (http://ideoplex.com/focus/java#ant).
- Ant Automation (http://hbtechs.blogspot.com/2007/06/automation-using-innovative-tools.html), a good handy example of automation with Ant.
- A simple Windows GUI for running Ant. (http://visualdrugs.net/antrunner/)

Apache Maven

Apache Maven

maven	
Developer(s)	Apache Software Foundation
Stable release	3.2.1[1] / February 21, 2014[2]
Development status	Active
Written in	Java
Operating system	Cross-platform
Type	Build tool
License	Apache License 2.0
Website	maven.apache.org [3]

Maven is a build automation tool used primarily for Java projects. Maven addresses two aspects of building software: First, it describes how software is built, and second, it describes its dependencies. Contrary to preceding tools like Apache Ant it uses conventions for the build procedure, and only exceptions need to be written down. An XML file describes the software project being built, its dependencies on other external modules and components, the build order, directories, and required plug-ins. It comes with pre-defined targets for performing certain well-defined tasks such as compilation of code and its packaging. Maven dynamically downloads Java libraries and Maven plug-ins from one or more repositories such as the Maven 2 Central Repository [4], and stores them in a local cache.[5] This local cache of downloaded artifacts can also be updated with artifacts created by local projects. Public repositories can also be updated.

Maven can be used to build and manage projects written in C#, Ruby, Scala, and other languages. The Maven project is hosted by the Apache Software Foundation, where it was formerly part of the Jakarta Project.

Maven is built using a plugin-based architecture that allows it to make use of any application controllable through standard input. Theoretically, this would allow anyone to write plugins to interface with build tools (compilers, unit test tools, etc.) for any other language. In reality, support and use for languages other than Java has been minimal. Currently a plugin for the .NET framework exists and is maintained,[6] and a C/C++ native plugin is maintained for Maven 2.[7]

Superseding technologies like gradle and sbt as build tools do not rely on XML any more, but keep the key concepts Maven introduced. With Apache Ivy a dedicated dependency manager was developed as well.

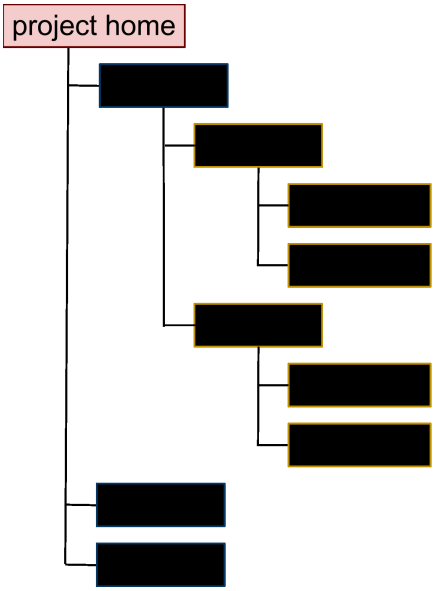
Example

Maven projects are configured using a Project Object Model, which is stored in a `pom.xml`-file. Here's a minimal example:

```
<project>
  <!-- model version is always 4.0.0 for Maven 2.x POMs -->
  <modelVersion>4.0.0</modelVersion>

  <!-- project coordinates, i.e. a group of values which
       uniquely identify this project -->
```

```
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0</version>

  <!-- library dependencies -->

  <dependencies>
    <dependency>

      <!-- coordinates of the required library -->

      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>

      <!-- this dependency is only used for running and compiling tests -->

      <scope>test</scope>

    </dependency>
  </dependencies>
</project>
```

This POM only defines a unique identifier for the project (*coordinates*) and its dependency on the JUnit framework. However, that is already enough for building the project and running the unit tests associated with the project. Maven accomplishes this by embracing the idea of Convention over Configuration, that is, Maven provides default values for the project's configuration. The directory structure of a normal idiomatic Maven project has the following directory entries:



The Maven software tool auto-generated this directory structure for a Java project.

Directory name	Purpose
project home	Contains the pom.xml and all subdirectories.
src/main/java	Contains the deliverable Java sourcecode for the project.
src/main/resources	Contains the deliverable resources for the project, such as property files.
src/test/java	Contains the testing Java sourcecode (JUnit or TestNG test cases, for example) for the project.
src/test/resources	Contains resources necessary for testing.

Then the command

```
mvn package
```

will compile all the Java files, run any tests, and package the deliverable code and resources into `target/my-app-1.0.jar` (assuming the artifactId is my-app and the version is 1.0.)

Using Maven, the user provides only configuration for the project, while the configurable plug-ins do the actual work of compiling the project, cleaning target directories, running unit tests, generating API documentation and so on. In general, users should not have to write plugins themselves. Contrast this with Ant and make, in which one writes imperative procedures for doing the aforementioned tasks.

Concepts

Project Object Model

A Project Object Model (POM) provides all the configuration for a single project. General configuration covers the project's name, its owner and its dependencies on other projects. One can also configure individual phases of the build process, which are implemented as plugins. For example, one can configure the compiler-plugin to use Java version 1.5 for compilation, or specify packaging the project even if some unit test fails.

Larger projects should be divided into several modules, or sub-projects, each with its own POM. One can then write a root POM through which one can compile all the modules with a single command. POMs can also inherit configuration from other POMs. All POMs inherit from the Super POM[8] by default. The Super POM provides default configuration, such as default source directories, default plugins, and so on.

Plugins

Most of Maven's functionality is in plugins. A plugin provides a set of goals that can be executed using the following syntax:

```
mvn [plugin-name]:[goal-name]
```

For example, a Java project can be compiled with the compiler-plugin's compile-goal[9] by running `mvn compiler:compile`.

There are Maven plugins for building, testing, source control management, running a web server, generating Eclipse project files, and much more.[10] Plugins are introduced and configured in a <plugins>-section of a `pom.xml` file. Some basic plugins are included in every project by default, and they have sensible default settings.

However, it would be cumbersome if the archetypical build sequence of building, testing and packaging a software project required running each respective goal manually:

```
mvn compiler:compile
mvn surefire:test
mvn jar:jar
```

Maven's lifecycle-concept handles this issue.

Plug-ins are the primary way to extend Maven. Developing a Maven plug-in can be done by extending the org.apache.maven.plugin.AbstractMojo class. Example code and explanation for a Maven plug-in to create a cloud-based virtual machine running an application server is given in the article *Automate development and management of cloud virtual machines*.

Build lifecycles

Build lifecycle is a list of named *phases* that can be used to give order to goal execution. One of Maven's standard lifecycles is the *default lifecycle*, which includes the following phases, in this order:[11]

```
process-resources
compile
process-test-resources
test-compile
test
package
install
deploy
```

Goals provided by plugins can be associated with different phases of the lifecycle. For example, by default, the goal "compiler:compile" is associated with the compile-phase, while the goal "surefire:test" is associated with the test-phase. Consider the following command:

```
mvn test
```

When the preceding command is executed, Maven runs all goals associated with each of the phases up to and including the test-phase. In such a case, Maven runs the "resources:resources"-goal associated with the process-resources-phase, then "compiler:compile", and so on until it finally runs the "surefire:test"-goal.

Maven also has standard phases for cleaning the project and for generating a project site. If cleaning were part of the default lifecycle, the project would be cleaned every time it was built. This is clearly undesirable, so cleaning has been given its own lifecycle.

Standard lifecycles enable users new to a project the ability to accurately build, test and install every Maven-project by issuing the single command:

```
mvn install
```

Dependencies

A central feature in Maven is dependency management. Maven's dependency-handling mechanism is organized around a coordinate system identifying individual artifacts such as software libraries or modules. The POM example above references the JUnit coordinates as a direct dependency of the project. A project that needs, say, the Hibernate-library simply has to declare Hibernate's project coordinates in its POM. Maven will automatically download the dependency and the dependencies that Hibernate itself needs (called transitive dependencies) and store them in the user's local repository. Maven 2 Central Repository [4] is used by default to search for libraries, but one

can configure the repositories to be used (e.g., company-private repositories) within the POM.

There are search engines such as The Central Repository Search Engine[12] which can be used to find out coordinates for different open-source libraries and frameworks.

Projects developed on a single machine can depend on each other through the local repository. The local repository is a simple folder structure which acts both as a cache for downloaded dependencies and as a centralized storage place for locally built artifacts. The Maven command `mvn install` builds a project and places its binaries in the local repository. Then other projects can utilize this project by specifying its coordinates in their POMs.

Maven compared with Ant

The fundamental difference between Maven and Ant is that Maven's design regards all projects as having a certain structure and a set of supported task work-flows (e.g., getting resources from source control, compiling the project, unit testing, etc.). While most software projects in effect support these operations and actually do have a well-defined structure, Maven requires that this structure and the operation implementation details be defined in the POM file. Thus, Maven relies on a convention on how to define projects and on the list of work-flows that are generally supported in all projects.

This design constraint resembles the way that an IDE handles a project, and it provides many benefits, such as a succinct project definition, and the possibility of automatic integration of a Maven project with other development tools such as IDEs, build servers, etc.

But one drawback to this approach is that Maven requires a user to first understand what a project is from the Maven point of view, and how Maven works with projects, because what happens when one executes a phase in Maven is not immediately obvious just from examining the Maven project file. In many cases, this required structure is also a significant hurdle in migrating a mature project to Maven, because it is usually hard to adapt from other approaches.

In Ant, projects do not really exist from the tool's technical perspective. Ant works with XML build scripts defined in one or more files. It processes targets from these files and each target executes tasks. Each task performs a technical operation such as running a compiler or copying files around. Targets are executed primarily in the order given by their defined dependency on other targets. Thus, Ant is a tool that chains together targets and executes them based on inter-dependencies and other Boolean conditions.

The benefits provided by Ant are also numerous. It has an XML language optimized for clearer definition of what each task does and on what it depends. Also, all the information about what will be executed by an Ant target can be found in the Ant script.

A developer not familiar with Ant would normally be able to determine what a simple Ant script does just by examining the script. This is not usually true for Maven.

However, even an experienced developer who is new to a project using Ant cannot infer what the higher level structure of an Ant script is and what it does without examining the script in detail. Depending on the script's complexity, this can quickly become a daunting challenge. With Maven, a developer who previously worked with other Maven projects can quickly examine the structure of a never-before-seen Maven project and execute the standard Maven work-flows against it while already knowing what to expect as an outcome.

It is possible to use Ant scripts that are defined and behave in a uniform manner for all projects in a working group or an organization. However, when the number and complexity of projects rises, it is also very easy to stray from the initially desired uniformity. With Maven this is less of a problem because the tool always imposes a certain way of doing things.

Note that it is also possible to extend and configure Maven in a way that departs from the Maven way of doing things. This is particularly true for Maven 2 and newer releases, such as Mojos [13] or more formally, plugins and custom project directory structures.

IDE integration

Add-ons to several popular Integrated Development Environments exist to provide integration of Maven with the IDE's build mechanism and source editing tools, allowing Maven to compile projects from within the IDE, and also to set the classpath for code completion, highlighting compiler errors, etc. Examples of popular IDEs supporting development with Maven include:

- Eclipse
- NetBeans
- IntelliJ IDEA
- JBuilder
- JDeveloper (version 11.1.2)
- MyEclipse

These add-ons also provide the ability to edit the POM or use the POM to determine a project's complete set of dependencies directly within the IDE.

Some built-in features of IDEs are forfeited when the IDE no longer performs compilation. For example, Eclipse's JDT has the ability to recompile a single java source file after it has been edited. Many IDEs work with a flat set of projects instead of the hierarchy of folders preferred by Maven. This complicates the use of SCM systems in IDEs when using Maven.[14][15][16]

History

Maven, created by Sonatype's [17] Jason van Zyl, began as a subproject of Apache Turbine in 2002. In 2003, it was voted on and accepted as a top level Apache Software Foundation project. In July 2004, Maven's release was the critical first milestone, v1.0. Maven 2 was declared v2.0 in October 2005 after about six months in beta cycles. Maven 3.0 was released in October 2010 being mostly backwards compatible with Maven 2.

Future

Maven 3.0 information began trickling out in 2008. After eight alpha releases, the first beta version of Maven 3.0 was released in April 2010. Maven 3.0 has reworked the core Project Builder infrastructure such that the POM's file-based representation is now decoupled from its in-memory object representation. This has expanded the possibility for Maven 3.0 add-ons to leverage non-XML based project definition files. Languages suggested include Ruby (already in private prototype by Jason van Zyl), YAML, and Groovy.

Special attention has been paid to ensuring compatibility between Maven 2 and 3. For most projects, an upgrade to Maven 3 will not require any adjustments of their project structure. The first beta of Maven 3 saw the introduction of a parallel build feature which leverages a configurable number of cores on a multi-core machine and is especially suited for large multi-module projects.

References

[1] Maven 3.2.1 (http://maven.apache.org/docs/3.2.1/release-notes.html)
[2] Maven Releases History (http://maven.apache.org/docs/history.html)
[3] http://maven.apache.org/
[4] http://central.sonatype.org
[5] Maven 2 Central Repository (http://repo1.maven.org/maven2/)
[6] .NET Maven Plugin (http://doodleproject.sourceforge.net/mavenite/dotnet-maven-plugin/index.html)
[7] maven-native C/C++ plugin (http://mojo.codehaus.org/maven-native/native-maven-plugin/index.html) and maven-nar C/C++ plugin (http://duns.github.com/maven-nar-plugin/)
[8] Super POM (http://maven.apache.org/guides/introduction/introduction-to-the-pom.html#Super_POM)
[9] Maven Compiler Plugin (http://maven.apache.org/plugins/maven-compiler-plugin/)
[10] Maven - Available Plugins (http://maven.apache.org/plugins/index.html)
[11] Maven Build Lifecycle Reference (http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle_Reference)
[12] The Central Repository Search Engine (https://search.maven.org/),
[13] http://maven.apache.org/guides/introduction/introduction-to-plugins.html
[14] Eclipse plugins for Maven (http://maven.apache.org/eclipse-plugin.html)
[15] IntelliJ IDEA - Ant and Maven support (http://www.jetbrains.com/idea/features/ant_maven.html#Maven_Integration)
[16] Best Practices for Apache Maven in NetBeans 6.x (http://wiki.netbeans.org/MavenBestPractices)
[17] http://www.sonatype.com

Further reading

- A free online book - O'Brien, et al., Tim. "Maven: The Complete Reference" (http://www.sonatype.com/books/mvnref-book/reference/). *Sonatype.com*. Sonatype. Retrieved 15 March 2013.
- The anteater book: *Maven: The Definitive Guide* (http://books.google.com/books?id=cBvZ4s72Z0gC). Sonatype Company. O'Reilly Media, Inc. 2009. p. 470. ISBN 9780596551780. Retrieved 2013-04-17.
- A printed book - Van Zyl, Jason (2008-10-01), *Maven: Definitive Guide* (first ed.), O'Reilly Media, p. 468, ISBN 0-596-51733-5

External links

- Official website (http://maven.apache.org)
- The Maven 2 tutorial: A practical guide for Maven 2 users (http://docs.codehaus.org/display/MAVENUSER/The+Maven+2+tutorial) - tutorial at Codehaus.org (http://www.codehaus.org)
- Building Web Applications with Maven 2 (http://today.java.net/pub/a/today/2007/03/01/building-web-applications-with-maven-2.html)
- The Maven 2 POM demystified (http://www.javaworld.com/javaworld/jw-05-2006/jw-0529-maven.html) - article at JavaWorld
- Maven for PHP (http://www.php-maven.org/)

Convention over configuration

Convention over configuration (also known as **coding by convention**) is a software design paradigm which seeks to decrease the number of decisions that developers need to make, gaining simplicity, but not necessarily losing flexibility.

The phrase essentially means a developer only needs to specify unconventional aspects of the application. For example, if there's a class Sale in the model, the corresponding table in the database is called “sales” by default. It is only if one deviates from this convention, such as calling the table “sale”, that one needs to write code regarding these names.

When the convention implemented by the tool matches the desired behavior, it behaves as expected without having to write configuration files. Only when the desired behavior deviates from the implemented convention is explicit configuration required.

Motivation

Some frameworks need multiple configuration files, each with many settings. These provide information specific to each project, ranging from URLs to mappings between classes and database tables. A large number of configuration files with lots of parameters is often an indicator of an unnecessarily complex application design.[1]

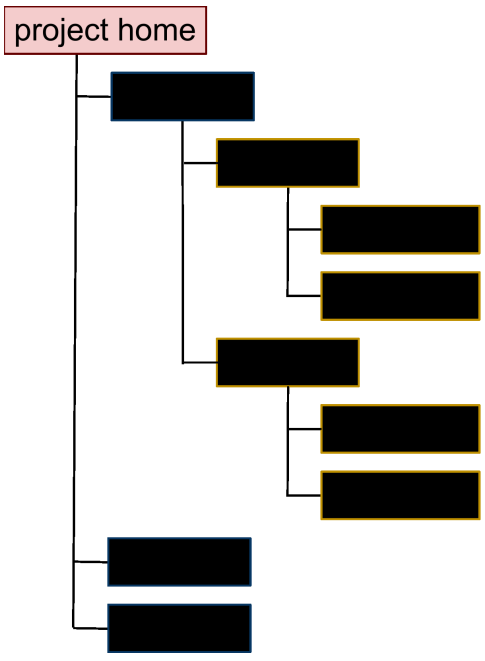
For example, early versions of the well-known Java persistence mapper Hibernate mapped entities and their fields to the database by describing these relationships in XML files. Most of this information could have been revealed by conventionally mapping class names to the identically named database tables and the fields to their columns, respectively. Later versions did away with the XML configuration file and instead employed these very conventions, deviations from which can be indicated through the use of Java annotations (see JavaBeans specification, linked below).

Usage

Many modern frameworks use a *convention over configuration* approach.

The concept is older, however, dating back to the concept of a default, and can be spotted more recently in the roots of Java libraries. For example, the JavaBean specification relies on it heavily. To quote the JavaBeans specification 1.01:[2]

> "As a general rule we don't want to invent an enormous java.beans.everything class that people have to inherit from. Instead we'd like the JavaBeans runtimes to provide default behaviour for 'normal' objects, but to allow objects to override a given piece of default behaviour by inheriting from some specific java.beans.something interface."



The Maven software tool auto-generated this directory structure for a Java project.

References

[1] C2 Wiki (1 September 2009). Too Many Parameters. C2 Wiki, 1 September 2009. Retrieved from http://c2.com/cgi/wiki?TooManyParameters.

[2] Sun (). JavaBeans specification (http://www.cs.vu.nl/~eliens/documents/java/white/beans.101.pdf), section 1.4.

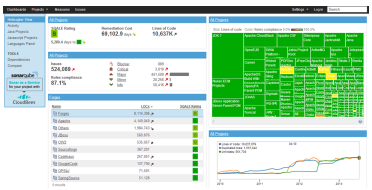
- Bachle, M., & Kirchberg, P. (2007). "Ruby on rails". Software, IEEE, 24(6), 105-108. DOI 10.1109/BCI.2009.31 (http://dx.doi.org/10.1109/BCI.2009.31).
- Miller, J. (2009). "Design For Convention Over Configuration". Microsoft, Retrieved April 18, 2010.
- Chen, Nicholas (2006). "Convention over configuration".

External links

- Detailed information on CoC (http://softwareengineering.vazexqi.com/files/pattern.html)

SonarQube

SonarQube



Developer(s)	SonarSource [1]
Stable release	4.2 [2] / March 26, 2014
Development status	Active
Written in	Java
Operating system	Cross-platform
Type	Software Analytics quality
License	Lesser GNU General Public License
Website	http://www.sonarqube.org

SonarQube (formerly **Sonar**) is an open source platform for Continuous Inspection of code quality.

Features

- Supports 25+ languages [3]: Java, C/C++, C#, PHP, Flex, Groovy, JavaScript, Python, PL/SQL, COBOL, etc.
- Can also be used in Android development.
- Offers reports on duplicated code, coding standards, unit tests, code coverage, complex code, potential bugs, comments and design and architecture.
- Records metrics history and provides evolution graphs ("time machine") and differential views.
- Provides fully automated analyses: integrates with Maven, Ant, Gradle and continuous integration tools (Atlassian Bamboo, Jenkins, Hudson, etc.).
- Integrates with the Eclipse development environment
- Integrates with external tools: JIRA, Mantis, LDAP, Fortify, etc.
- Is expandable with the use of plugins.
- Implements [4] the SQALE methodology to compute technical debt.
- Supports Tomcat. However, end of support to Tomcat is planned for SonarQube 4.1. The standalone mode [5] is now the only mode that is supported. The standalone mode embeds a Tomcat server.

Reception

In 2009 Sonar received the Jolt Awards under testing tools category. Featured in continuous integration tools. Used by the Andalusian Autonomous Government, eXo Platform, Apache Software Foundation, Eclipse Foundation, Marvelution.

References

[1] http://www.sonarsource.com
[2] http://www.sonarqube.org/downloads/
[3] http://docs.codehaus.org/display/SONAR/Plugin+Library
[4] http://www.sonarsource.com/products/plugins/governance/sqale/
[5] http://sonarqube.15.x6.nabble.com/End-of-support-of-Tomcat-is-planned-for-SonarQube-4-1-end-of-October-td5016113.html

External links

- SonarQube Web Site (http://www.sonarqube.org)
- SonarSource Web Site (http://www.sonarsource.com/)
- Live SonarQube instance (http://nemo.sonarsource.org/)
- SonarQube Plugin Library (http://sonar-plugins.codehaus.org/)
- SonarQube IDE Integration (http://sonar-ide.codehaus.org/)

Further reading

- SonarQube Tutorial (http://www.javatips.net/blog/2013/10/sonarqube-tutorial)
- Eclipse Sonar Tutorial (http://www.javatips.net/blog/2013/10/eclipse-sonar-tutorial)

Minification (programming)

Minification (also **minimisation** or **minimization**), in computer programming languages and especially JavaScript, is the process of removing all unnecessary characters from source code without changing its functionality. These unnecessary characters usually include white space characters, new line characters, comments, and sometimes block delimiters, which are used to add readability to the code but are not required for it to execute.

Minified source code is especially useful for interpreted languages deployed and transmitted on the Internet (such as JavaScript), because it reduces the amount of data that needs to be transferred. Minified source code may also be used as a kind of obfuscation, though the term obfuscation may be distinguished as a form of false cryptography while a minified code instance may be reversed using a pretty-printer. In Perl culture, aiming at extremely minified source code is the purpose of the Perl golf game.

Minified source code is also very useful for HTML code. As an example, successive whitespace characters in HTML are rendered as a single space, so replacing all whitespace sequences with single spaces can considerably reduce the size of a page.

Minification can be distinguished from the more general concept of data compression in that the minified source can be interpreted immediately without the need for an uncompression step: the same interpreter can work with both the original as well as with the minified source.

Types

Tools

JavaScript optimizers such as JSMin[1] and Packer[2] are specially designed for modern web programming techniques, and are able to understand and preserve conditional comments, and similar. Packer, for instance, can optionally Base64 compress the given source code in a manner that can be decompressed by regular web browsers, as well as shrink variable names that are typically 5–10 characters to single letters, which reduces the file size of the script and, therefore, makes it download faster.[3] Google has released their Closure Compiler, which also provides minification as well as the ability to introduce more aggressive renaming, removing dead code, and providing function inlining.[4] In addition, certain online tools, such as Microsoft Ajax Minifier,[5] the Yahoo! YUI Compressor or Pretty Diff,[6] can compress CSS files.Wikipedia:Citation needed There is a PowerShell script named "minifyPS"[7] that is able to shrink PowerShell script code as well as JavaScript code. There is a free online tool that can minify JS[8] with UglifyJS and has the ability to combine multiple files at BlimptonTech.com.[9]

Web development

Components and libraries for Web applications and websites have been developed to optimize file requests and quicken page load times by reducing the size of various files.

JavaScript and CSS resources may be minified, preserving their behavior while considerably reducing their file size. The Closure Tools [10] project is an effort by Google engineers to open source the tools used in many of Google's sites and web applications for use by the wider Web development community. Closure Compiler [11] compiles JavaScript into compact, high-performance code, and can perform aggressive global transformations in order to achieve high compression and advanced optimization. Other libraries available online are also capable of minification and optimization to varying degrees.

Some libraries also merge multiple script files into a single file for client download. This fosters a modular approach to development.Wikipedia:Citation needed

A novel approach to server-side minification is taken by Ziproxy, a forwarding, non-caching, compressing HTTP proxy targeted for traffic optimization. It minifies and optimizes HTML, CSS, and JavaScript resources and, in addition, re-compresses pictures.

Content encoding is an approach taken by compatible web servers and modern web browsers to compress HTML and related textual content, often in the gzip format.

An alternative to content encoding in the server-client layer is given by the off-line CrunchMe tool, which can create self extracting JavaScript programs using the DEFLATE compression algorithm.Wikipedia:Citation needed

JavaScript source maps can make code readable and more importantly debuggable even after its been combined and minified.[12]

References

[1] JSMin (http://www.crockford.com/javascript/jsmin.html). Crockford.com (4 December 2003).
[2] Packer (http://dean.edwards.name/packer/). Dean.edwards.name.
[3] Packer version 3.0 feature list (http://dean.edwards.name/weblog/2007/04/packer3/). Dean.edwards.name.
[4] Google Closure Compiler (http://code.google.com/closure/compiler/). Code.google.com (2 July 2012).
[5] Microsoft Ajax Minifier (http://ajaxmin.codeplex.com/). Ajaxmin.codeplex.com (13 September 2012).
[6] Pretty Diff (http://prettydiff.com/?m=minify). Pretty Diff.
[7] minifyPS (http://minifyps.codeplex.com). Minifyps.codeplex.com (22 February 2012).
[8] Online JS Minify (http://toolswebtop.com/javascript/minify)
[9] BlimptonTech (http://www.blimptontech.com/). BlimptonTech.com (17 July 2013).
[10] https://developers.google.com/closure/
[11] http://closure-compiler.appspot.com/home
[12] http://www.html5rocks.com/en/tutorials/developertools/sourcemaps/

Google Closure Tools

Not to be confused with Clojure.

Google Closure Tools

Original author(s)	Google
Initial release	November 5, 2009[1]
Available in	JavaScript
Type	Ajax framework
License	Apache License 2.0
Website	developers.google.com/closure/ [10]

Google Closure Tools[2] is a set of tools to help developers build rich web applications with JavaScript. It was developed by Google for use in their web applications such as Gmail, Google Docs and Google Maps.[3]

Closure Compiler

The Closure Compiler [10] is a tool for making JavaScript download and run faster. It optimizes JavaScript. It does not compile from JavaScript to machine code, but rather compiles from JavaScript to better JavaScript (for machine, not human). It parses JavaScript, analyzes it, removes dead code and rewrites and minimizes what's left. It also checks syntax, variable references, and types, and warns about common JavaScript pitfalls. The compiler is provided in three ways:

- Command-line:
  - This Java application can be invoked from the command line, and passed the list of JS files to be compiled.
- Interactive-way:
  - Closure Compiler service website [11] provides a form for user to input a URL pointing a JavaScript source or input a JavaScript source in a textbox and the website will response with the *optimized JavaScript* on right side for user to copy.
- HTTP POST API:
  - Closure Compiler server website [4] is waiting HTTP POST with severals HTTP POST parameters, see complete list [5]. One is *js_code* or *code_url* which contains the string of JavaScript to be optimized. In return to the HTTP POST, the optimized JavaScript code will be the response content of application/x-www-form-urlencoded.

Closure Library

The Closure Library [6] is a JavaScript library, written specifically to take advantage of the Closure Compiler, based on a modular architecture. It provides cross-browser functions for DOM manipulations and events, Ajax and JSON, as well as more high-level objects such as User Interface widgets and Controls.

Closure Templates

Closure Templates [7] are a templating system for dynamically generating HTML in both Java[8] and JavaScript.[9]

Because the language was apparently referred to as "Soy" internal to Google, and "Soy" remains in some of the documentation and classes,[10] sometimes Closure Templates are referred to as "Soy Templates".

References

[1] http://googlecode.blogspot.com/2009/11/introducing-closure-tools.html
[2] Bolin, Michael, "Closure: The Definitive Guide", O'Reilly Media Inc., Sebastopol, CA, 2010
[3] https://developers.google.com/closure/faq#gwt
[4] http://closure-compiler.appspot.com/compile
[5] https://developers.google.com/closure/compiler/docs/api-ref
[6] http://code.google.com/closure/library/docs/gettingstarted.html
[7] http://code.google.com/closure/templates/
[8] http://code.google.com/closure/templates/docs/helloworld_java.html
[9] http://code.google.com/closure/templates/docs/helloworld_js.html JavaScript
[10] http://code.google.com/p/closure-templates/source/browse/trunk/javascript/soyutils.js

External links

- Official website (https://code.google.com/closure/)
- Template:Package for TYPO3.Flow

JSHint

JSHint

Original author(s)	Anton Kovalyov, forked from original code by Douglas Crockford
Initial release	December 16, 2010
Stable release	2.1.10 / August 15, 2013
Development status	Active
Written in	JavaScript
Operating system	Cross-platform
Available in	English
Type	Static code analysis
License	Modified MIT license
Website	jshint.com [1]

JSHint is a static code analysis tool used in software development for checking if JavaScript source code complies with coding rules. It was forked from Douglas Crockford's JSLint project, as it was felt that the original did not allow enough customization options. There is also an internet version available at its official website in which users can paste code to run the application online. A command-line version of JSHint, distributed as a Node.JS module, makes it possible to automate one's "linting" process and integrate JSHint into the website's development workflow.

License

JSHint is distributed under an MIT license, except for one file still is under the JSLint License which is a slightly modified version of the MIT license. The additional clause specifies that the software shall be used for Good and not Evil and makes the Software proprietary[2]

References

[1] http://jshint.com/

[2] http://www.gnu.org/licenses/license-list.en.html#JSON see the comment about the JSON license

Further reading

- Zakas, Nicholas (May 2012). *Maintainable JavaScript* (http://books.google.com/books?id=bHhlCrvbqSoC&pg=PA142&dq=JSHint&hl=en&sa=X&ei=1OvOT_zgB-qg2QX_wfTIDA&ved=0CDkQ6AEwAQ#v=onepage&q=JSHint&f=false) (1 ed.). O'Reilly Media. ISBN 978-1-449-32768-2.
- Otero, Cesar (May 2012). *Professional jQuery* (http://books.google.com/books?id=Ei7Kl8nIEpAC&pg=PA15&dq=JSHint&hl=en&sa=X&ei=1OvOT_zgB-qg2QX_wfTIDA&ved=0CD4Q6AEwAg#v=onepage&q=JSHint&f=false) (1 ed.). John Wiley & Sons. ISBN 978-1-118-02668-7.
- Ullman, Larry (February 2012). *Modern JavaScript: Develop and Design* (http://books.google.com/books?id=ExJ9_sor87QC&pg=PA83&dq=JSHint&hl=en&sa=X&ei=1OvOT_zgB-qg2QX_wfTIDA&ved=0CEoQ6AEwBA#v=onepage&q=JSHint&f=false) (1 ed.). Peachpit Press. ISBN 978-0321812520.
- "JSHint - the (gentler) JavaScript code quality tool" (http://www.i-programmer.info/news/90-tools/2024-jshint-the-gentler-javascript-code-quality-tool.html). IProgrammer. February 21, 2011. Retrieved June 6, 2012.

External links

- Official website (http://jshint.com)
- Why I forked JSLint to JSHint (http://anton.kovalyov.net/2011/02/20/why-i-forked-jslint-to-jshint/)
- JSHint: An Community Driven Fork of JSLint (http://news.ycombinator.com/item?id=2236417)

Chapter 25: App Development in a Distributed Virtual Team with Redmine

Virtual team

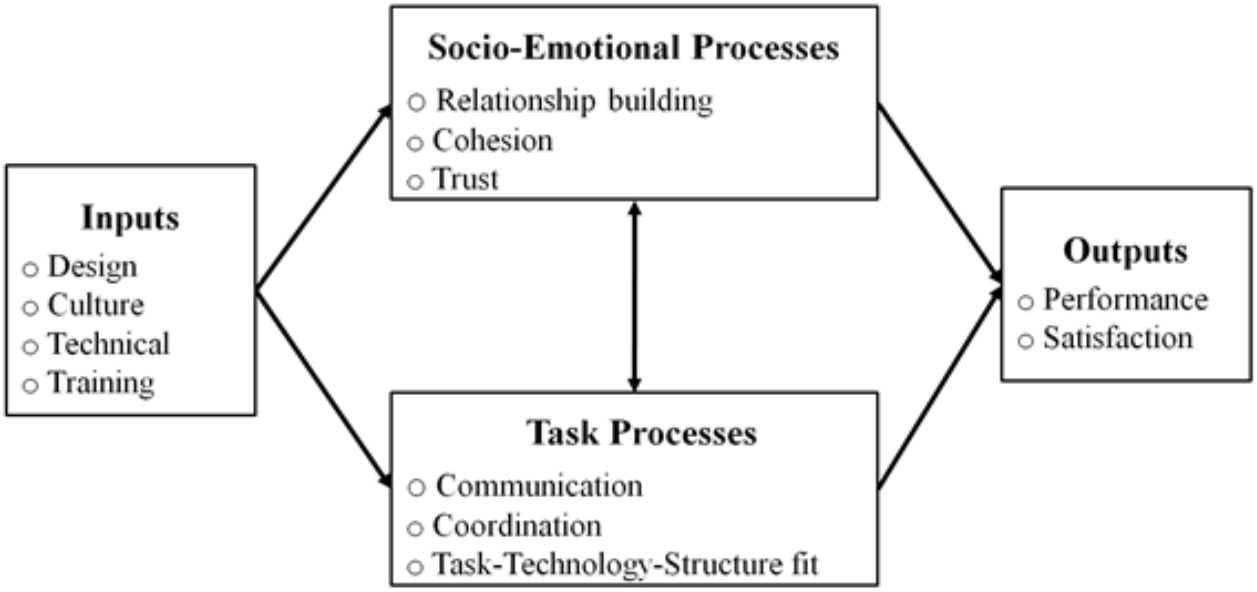
A **virtual team** (also known as a **geographically dispersed team**, **distributed team**, or **remote team**) is a group of individuals who work across time, space and organizational boundaries with links strengthened by webs of communication technology. Powell, Piccoli and Ives define virtual teams in their literature review article "as groups of geographically, organizationally and/or time dispersed workers brought together by information and telecommunication technologies to accomplish one or more organizational tasks."[1] Ale Ebrahim, N., Ahmed, S. & Taha, Z. in a 2009 literature review paper, added two key issues to definition of a virtual team "as **small temporary groups** of geographically, organizationally and/ or time dispersed knowledge workers who coordinate their work predominantly with electronic information and communication technologies in order to accomplish one or more organization tasks". Members of virtual teams communicate electronically and may never meet face-to-face. Virtual teams are made possible by a proliferation of fiber optic technology that has significantly increased the scope of off-site communication.[2] Virtual teams allow companies to procure the best talent without geographical restrictions. According to Hambley, O'Neil, & Kline (2007), "virtual teams require new ways of working across boundaries through systems, processes, technology, and people, which requires effective leadership... despite the widespread increase in virtual teamwork, there has been relatively little focus on the role of virtual team leaders."

Model

There are three main aspects to a virtual team - purpose, people and links.[3] While purpose is an important aspect for all organizations, it's the most critical aspect for virtual teams; purpose is what holds a virtual team together. Virtual teams do not have hierarchy or any other common structures because they may not be from the same organization, and purpose here brings and holds the team together. Purpose is generally translated into certain action steps for people to work on with a defined structure consisting of common goals, individual tasks and results. A number of factors may impact the performance of members of a virtual team. For example, team members with a higher degree of focused attention and aggregate lower levels of temporal dissociation (or flow) may have higher performance. Further, members with higher degrees of attention focus may prefer asynchronous communication channels, while those with low levels of flow may prefer synchronous communication channels.

Structure

Powell, Piccoli and Ives[4] found and investigated 43 articles about virtual teams and concluded that the current research have found four main focus areas of it.



Inputs

Design of a virtual team means simply that forming a VT should be planned. This means structuring the interactions; what kind of communication tools are used, how much face-to-face time will be possible, etc. Research has found that team building exercises,[5] the establishment of shared norms (Sarker et al., 2001, p. 50) and the establishment of a clear team structure[6] helps the team to succeed.[7] Kirkman *et al.*[8] [9] found empirically that having more face-to-face meetings improved the empowerment of virtual teams, which leads to better learning. Numerous communication problems can be diverted by creating shared knowledge databases in order to allow all the team members to have the same information and to know that others have it, too.[10] As an added bonus, shared knowledge databases also share the same language and mental models, which are substitutes for the all important face-to-face time. Furthermore, shared mental models can be focused through designing, requiring the teams to create goals and strategies. This has been shown clearly to improve the teams[11]

With **cultural differences** also coordination problems and obstacles to effective communication can be involved.[12] These problems may be solved by actively understanding and accepting differences in cultures.[13]

The **technical expertise** of a team seems to have a positive effect on the team"s performance and the satisfaction of belonging to the team.[14] At the same time, high trust is found to develop.[15] On the other hand, "the relationship between technology and task performance is found to be more dependent on experience with technology and with group membership than the type of task on which the group was working".[16]

Diverse technological skills can create conflict among the team.[17] This is why teams should have consistent **training** to improve team performance.[18] For instance, mentoring is a good way to make personal ties to more experienced virtual team professionals.[19] According to Tan et al.,[20] consistent training fosters cohesiveness, trust, team work, commitment to team goals, individual satisfaction and higher perceived decision quality. In their article, they taught a communication technique called the dialogue technique. It is created through three stages: small talk, sharing mental models and norm building.

Socio-emotional processes

This section introduces the emotional problems involved and mitigation tactics needed to achieve cohesion and trust among team members. Overall, the research about this reports "a positive link between socio-emotional process and outcomes of the virtual team project."[21] Because of geographical distribution, face-to-face time occurs only rarely. This, according to research, results in weaker social links between team-mates and leads the team to be more task-focused than socially focused.[22] If face-to-face meetings are feasible, meetings should be held as much as possible at the beginning of the team formation in order to bring team-mates closer and form interpersonal bonds. These meetings should focus more on **relationship building** than on actual business.[23] However, with socializing different cultural preferences have to be remembered.[24] If face-to-face meetings are not possible or feasible to the desired extent, other approaches can be applied. Social-bonding can be done partially via electronic communication tools. Jarvenpaa and Leidner's[25] study found that if teams communicate more socially they achieve higher trust and better social and emotional relationships. Leaders can help foster relationship building and general team building in many ways, e.g. by providing continuous feedback, listening to team members‟ opinions and suggestions, clearly stating the team member roles and having consistency in their leadership style.[26]

Cohesion means the sense of unity in a team. It is found to be important, but there are no conclusive results on how to support it in the virtual team context.

Trust is particularly problematic subject with virtual teams, because it is arguable whether people can be expected to trust each other if they have never met face–to-face.[27] Furthermore, trust is noted to be crucial in successful teams, but usually there is not much time to build it little by little because often the teams are short-lived in projects. Jarvenpaa and Leidner[28] describe a mechanism of how people solve the trust problem in a short time. It is called the swift trust paradigm and it suggests that team members assume from the beginning that the other team members are trustworthy. They adjust that assumption during the lifetime of the team. Jarvenpaa and Leidner also researched the differences between teams that had a high level of trust in the beginning and teams with a high amount of trust in the end and compared them. To achieve high trust early in the group‟s life, the team had social and enthusiastic communication and they coped well with technical uncertainty and took individual initiatives. The groups that enjoyed trust later had predictable communication, timely responses, positive leadership and the ability to move from social communication to task-focused communication.

Task processes

Task processes are the different functions that happen when a team is doing its work. **Communication** is one of the most crucial things in virtual teams. It starts from selecting excellent communicators for the team members and the right technology for them to use.[29] Some empirically found challenges in successful communication in virtual teams are failure to communicate due to wrong or lacking contextual information, unevenly distributed information, interpretation of the meaning of silence and technical problems.[30] Because of the lack of face-to-face time, the team can miss nonverbal communication altogether. The extensive reliance on communication technology leads to reduced impact and difficulties in management compared to the traditional teams.[31] Researchers have found some solutions for these problems. One company has created a reward system for team cooperation to encourage people to actively and accurately communicate.[32] On the other hand, according to Pink's[33] research on rewarding creativity, rewarding communication is not a sustainable way to encourage cooperation. In another company, they emphasized the need to debate as well as merely share information.[34] Predictability and feedback also frequently improve communication effectiveness, creating trust and better team performance. In addition, in one study researchers tested the question of whether adding video to electronic communication helps to explain a detailed task (a map route) to another person.[35] They found that for native speaker pairs it did not bring any additional benefits, but for non-native speaker pairs it brought significant improvement to the task.[36]

It is, naturally, more difficult to **coordinate** virtual teams in different time zones, cultures and mental models. Collaboration norms have to develop for the team to function well.[37] As mentioned before, periodical face-to-face

meetings are a good way to form relationships and also a good vehicle to coordinate activities and to drive the project forward.[38] When face-to-face meetings are not feasible, one alternative is to develop coordination protocols with communication training.[39] Ramesh and Dennis[40] have suggested standardizing the team"s inputs, processes and/or outputs. This should help the team to coordinate and help the other party.

The **task-technology-structure** fit examines "the possible fit between various technologies available..."; Studies have hypothesized that the technology fit depends on individual preferences, e.g. experience of use and the urgency of the task;[41] Majchrzak et al.[42] found that face-to-face meetings or phone calls are suitable for ambiguous tasks, managing conflicts, managing external resources, brainstorming and strategic talks. Electric communication is more suitable for more structured tasks such as routine analysis, examining design tradeoffs and monitoring project status. Interestingly, in their study the team first adjusted their organization to the technology at hand, but later also adjusted the technology to their organization.

Outputs

Output in virtual teams means all the things that come out of the work processes of the team. When comparing **the performance** of traditional and virtual teams, the results are mixed. Some studies find traditional teams and some virtual teams to be better. The majority of studies have found the teams to be about at the same level.[43] Powell, Piccoli and Ives[44] list many studies that have found different factors, which make virtual teams successful. The found factors are:

- Training
- Strategy/goal setting
- Developing shared language
- Team building
- Team cohesiveness
- Communication
- Coordination and commitment of the teams
- The appropriate task-technology fit
- Competitive and collaborative conflict behaviors (conversely, the same study found that avoidance and compromise conflict behavior had a negative impact)

The results from different student studies are mixed concerning working in a virtual team. Tan et al.[45] found that teams which used their dialogue technique were more **satisfied** with decisions made in the team. One study found that a traditional team started out more satisfied than a virtual team. Then, in less than a year, the satisfaction of the virtual team rose and exceeded the satisfaction of the traditional team.[46] Furthermore, some studies have found that women, generally, are happier in virtual teams than men.[47]

Types

Below are the most common types of virtual teams.[]

1. Networked teams
2. Parallel teams
3. Project development teams
4. Work, production or functional teams
5. Service teams
6. Offshore ISD teams

Networked teams

Generally, networked teams[48] are geographically distributed and not necessarily from the same organization. These teams are frequently created and just as frequently dissolved; they are usually formed to discuss specific topics where members from the area of expertise, possibly from different organizations, pitch their ideas in the same discussion. Depending on the complexity of the issue, additional members to the team may be added at any time. The duration these teams last may vary significantly depending on how fast or slow the issue is resolved.

Parallel teams

Parallel teams are highly task oriented teams that usually consist of specialized professionals. While they are generally only required for very short span of time, unlike networked teams, they are not dissolved after completion of the tasks. The team may be either internal or external to the organization.

Project development teams

Similar to parallel teams, these teams are geographically distributed and may operate from different timezones. Project development teams are mainly focused on creating new products, information systems or organizational processes for users and/or customers. These teams exist longer than parallel teams and have the added ability to make decisions rather than just make recommendations. Similar to networked teams, project development teams may also add or remove members of their team at any given time, as needed for their area of expertise.

Work, production or functional teams

These teams are totally function specific where they only work on a particular area within an organization (i.e. finance, training, research, etc.). Operating virtually from different geographical locations, these teams exist to perform regular or ongoing tasks.

Service teams

Service teams are geographically located in different time-zones and are assigned to a particular service such as customer support, network upgrades, data maintenance, etc. Each team works on providing the particular service in their daylight hours and at the end of day, work is delegated to the next team which operates in a different timezone so that there is someone handling the service 24 hours a day.

Offshore ISD teams

Offshore ISD outsourcing teams are independent service provider teams that a company can subcontract portions of work to. These teams usually work in conjunction with an onshore team. Offshore ISD is commonly used for software development as well as international R&D projects.

Advantages

Increased productivity: Virtual teams often see an increase in productivity because more personal flexibility is achieved, commute time is reduced, and work is not limited by the traditional 9-5 work day schedule. In turn, the company never sees an off hour. The team on the other side of the globe simply picks up where the prior team left off. This approach is commonly referred to as "Follow the Sun Approach". This advantage can translate to a much faster time to market for new products and technology.Wikipedia:Citation needed

Extended market opportunity: This is a major benefit of geographically dispersed teams due to direct access to different market opportunities. With work teams located in different parts of the globe, organizations are able to establish their presence with customers worldwide. This also gives small business owners the ability to compete on a global scale as well without being limited to a particular customer base.Wikipedia:Citation needed

Knowledge transfer: This is one of the most important benefits of a virtual team; utilizing people with different types of knowledge spread out across the globe can be very beneficial to any organization. Online meetings, remote computer access, wireless technology, and conferencing systems offer a way for participants to join a complex discussion from anywhere in the world. This benefit can enable most companies to compete on a global scale.Wikipedia:Citation needed

Statistics Related to Virtual Work [49] Both fully virtual teams and organizations that employ some virtual workers experience a high return on investment in retention, company loyalty and valuable output.

Disadvantages

Communication deficiency: The biggest disadvantage that any virtual team can suffer from is the lack of efficiency in communication, partly due to constraints in virtual communication mediums. This is also primarily due to the fact that humans communicate better when they are able to communicate with their body language. Inevitably, virtual teams may face obstacles due to restrictions of the Internet which in turn may lead to incorrect assumptions if a message is not laid out clearly. Failure to properly communicate and clearly address messages or emails could lead to frustration and eventually failure.Wikipedia:Citation needed

Poor leadership and management: Poor leadership can result in the failure of any team, whether virtual or not; however, it becomes a much more prominent problem in virtual teams. Messages must be sent across accurately and clearly. Inability to effectively communicate to members of the team can all greatly affect a project .[50]

Incompetent team members: Virtual teams should only consist of competent and experienced team members due to the distance factor which can overtly affect the timing and completion date of a project. Projects are more likely to fail if the team consists of individuals who are lazy or lack sufficient knowledge to complete their assigned tasks. It only takes one incompetent team member to have a negative effect on the rest of the team.Wikipedia:Citation needed

References

[1] Anne Powell, Gabriele Piccoli, and Blake Ives. Virtual teams: a review of current literature and directions for future research. The DATA BASE for Advances in Information Systems - Winter Vol. 35, issue 1, 2004

[2] Vlaar, P. (2008). Co Creating Understanding And Value In Distributed Work. *MIS Quarterly*, 32, 227-255.

[3] (http://proquest.umi.com.ezproxy1.lib.depaul.edu/pqdlink?did=38782926&Fmt=6&clientId=31663&RQT=309&VName=PQD), Jessica Lipnack, & Jeffrey Stamps. (1999). Virtual teams: The new way to work. Strategy & Leadership, 27(1), 14-19. Retrieved November 2, 2010, from ABI/INFORM Global. (Document ID: 38782926).

[4] see Powell, Piccoli and Ives (2004) p.8, Anne Powell, Gabriele Piccoli, and Blake Ives. Virtual teams: a review of current literature and directions for future research. The DATA BASE for Advances in Information Systems - Winter Vol. 35, issue 1, 2004 .

[5] Sarker et al. (2000) p.80, Suprateek Sarker, Francis Lau, and Sundeep Sahay. Using an adapted grounded theory approach for inductive theory building about virtual team development. SIGMIS Database vol. 32, issue 1, 2000.

[6] Sarker et al. (2000) p.81, Suprateek Sarker, Francis Lau, and Sundeep Sahay. Using an adapted grounded theory approach for inductive theory building about virtual team development. SIGMIS Database vol. 32, issue 1, 2000.

[7] Powell, Piccoli and Ives (2004) p.8, Anne Powell, Gabriele Piccoli, and Blake Ives. Virtual teams: a review of current literature and directions for future research. The DATA BASE for Advances in Information Systems - Winter Vol. 35, issue 1, 2004 .

[8] Kirkman et al. (2004) p.186, Bradley L. Kirkman, Benson Rosen, Paul E. Tesluk, Cristina B. Gibson, The Impact of Team Empowerment on Virtual Team Performance: The Moderating Role of Face-to-Face Interaction, The Academy of Management Journal, Vol. 47, No. 2 (Apr., 2004), p. 175-192.

[9] http://web.gsm.uci.edu/~cgibson/Publication%20files/Articles/Team%20Empowerment%20in%20Virtual%20Teams.pdf

[10] Crampton, C. (2001) p.355-359, Catherine Cramton, The Mutual Knowledge Problem and its Consequences for Dispersed Collaboration, Organization Science, Vol. 12, issue 3, 2001, p. 346-371.

[11] Suchan and Hayzak (2001) p.185, Jim Suchan, Greg Hayzak, The Communication Characteristics of Virtual Teams: A Case Study, IEEE Transactions on Professional Communication, Vol. 44, issue 3, p. 174-186.

[12] see Powell, Piccoli and Ives (2004) p.9, Anne Powell, Gabriele Piccoli, and Blake Ives. Virtual teams: a review of current literature and directions for future research. The DATA BASE for Advances in Information Systems - Winter Vol. 35, issue 1, 2004 .

[13] Robey, Khoo and Powers (2000) p.58, Daniel Robey and Huoy Min Khoo and Carolyn Powers, Situated Learning in Cross-Functional Virtual Teams, IEEE Transactions on Professional Communication, 2000, 43, p. 51-66.

[14] Van Ryssen and Godar (2000) p. 55-56, Stefaan Van Ryssen, Susan Hayes Godar, Going international without going international: multinational virtual teams, Journal of International Management, Volume 6, Issue 1, 2000, p. 49-60.

[15] Jarvenpaa and Leidner, (1999) p. 807, Sirkka Jarvenpaa and Dorothy E. Leidner, Communication and Trust in Global Virtual Teams, Organization Science; Special Issue: Communication Processes for Virtual Organizations, Vol. 10, issue 6, 1999, p. 791-815.

[16] Hollingshead, McGrath and O"Connor (1993) p.328, Hollingshead, A., McGrath, J., and O"Connor, K. , Group Task Performance and Communication Technology: A Longitudinal Study of Computer mediated versus Face-to-face Groups, Small Group Research, Vol. 24, issue 3, 1993, p. 307-333.

[17] Sarker and Sahay (2002) p.4-5, Sarker, Suprateek and Sahay, Sundeep, Information systems development by US-Norwegian virtual teams: implications of time and space, System Sciences, HICSS. Proceedings of the 35th Annual Hawaii International Conference on System Sciences, 2002.

[18] Kaiser et al. (2000) p.80, Paula R. Kaiser, William L. Tullar and Diana McKowen, Student Team Projects by Internet, Business Communication Quarterly, Volume 63, issue 4, 2000, pages 75–82.

[19] Suchan and Hayzak (2001) p.183, Jim Suchan, Greg Hayzak, The Communication Characteristics of Virtual Teams: A Case Study, IEEE Transactions on Professional Communication, Vol. 44, issue 3, p. 174-186.

[20] Tan et .al (2000) p.160, Bernard Tan, Kwok-Kee Wei, Wayne Huang, Guet-Ngoh Ng, A Dialogue Technique to Enhance Electronic Communication in Virtual Teams, IEEE Transactions on Professional Communication, Vol. 43, issue 2, 2000, p. 153-165.

[21] Powell, Piccoli and Ives (2004) p.9-10, Anne Powell, Gabriele Piccoli, and Blake Ives. Virtual teams: a review of current literature and directions for future research. The DATA BASE for Advances in Information Systems - Winter Vol. 35, issue 1, 2004.

[22] see Powell, Piccoli and Ives (2004) p.10, Anne Powell, Gabriele Piccoli, and Blake Ives. Virtual teams: a review of current literature and directions for future research. The DATA BASE for Advances in Information Systems - Winter Vol. 35, issue 1, 2004.)

[23] Robey, Khoo and Powers (2000) p.59, Daniel Robey and Huoy Min Khoo and Carolyn Powers, Situated Learning in Cross-Functional Virtual Teams, IEEE Transactions on Professional Communication, 2000, vol 43, p. 51-66.

[24] Powell, Piccoli and Ives (2004) p.10, Anne Powell, Gabriele Piccoli, and Blake Ives. Virtual teams: a review of current literature and directions for future research. The DATA BASE for Advances in Information Systems - Winter Vol. 35, issue 1, 2004.)

[25] Jarvenpaa and Leidner, (1999) p.807 , Sirkka Jarvenpaa and Dorothy E. Leidner, Communication and Trust in Global Virtual Teams, Organization Science; Special Issue: Communication Processes for Virtual Organizations, Vol. 10, issue 6, 1999, p. 791-815.

[26] Kayworth and Leidner (2001) p.25 , Timothy R. Kayworth and Dorothy E. Leidner, Leadership Effectiveness in Global Virtual Teams Journal of Management Information Systems Vol. 18, issue 3, 2001/2002, pp. 7–40.

[27] McDonough, Kahn, Barczak (2000) p.115-116, Edward F. McDonough III, Kenneth B. Kahn, Gloria Barczak, An investigation of the use of global, virtual, and collocated new product development teams, Northeastern University, Boston and the University of Tennessee, USA,2000.

[28] Jarvenpaa and Leidner, (1999) p.794 , Sirkka Jarvenpaa and Dorothy E. Leidner, Communication and Trust in Global Virtual Teams, Organization Science; Special Issue: Communication Processes for Virtual Organizations, Vol. 10, issue 6, 1999, p. 791-815.

[29] Powell, Piccoli and Ives (2004) p.11, Anne Powell, Gabriele Piccoli, and Blake Ives. Virtual teams: a review of current literature and directions for future research. The DATA BASE for Advances in Information Systems - Winter Vol. 35, issue 1, 2004.)

[30] Crampton, C. (2001) p.360, Catherine Cramton, The Mutual Knowledge Problem and its Consequences for Dispersed Collaboration, Organization Science, Vol. 12, issue 3, 2001, p. 346-371.

[31] McDonough, Kahn, Barczak (2000) p.119, Edward F. McDonough III, Kenneth B. Kahn, Gloria Barczak, An investigation of the use of global, virtual, and collocated new product development teams, Northeastern University, Boston and the University of Tennessee, USA,2000.

[32] Suchan and Hayzak (2001) p.179, Jim Suchan, Greg Hayzak, The Communication Characteristics of Virtual Teams: A Case Study, IEEE Transactions on Professional Communication, Vol. 44, issue 3, p. 174-186.

[33] Dan Pink, Drive: the surprising truth about what motivates us, Published 2009 by Riverhead Books in New York.

[34] Kruempel (2000) p. 191, Kari Kruempel, Making the Right (Interactive) Moves for Knowledge-Producing Tasks in Computer-Mediated Groups, IEEE transactions on professional communication, vol. 43, issue 2, 2000.

[35] Veinott, Olson, Olson, and Fu, (1999) p. 303, Elizabeth S. Veinott, Judith Olson, Gary M. Olson, and Xiaolan Fu. Video helps remote work: speakers who need to negotiate common ground benefit from seeing each other. Concurrent Engineering, vol 15 issue 2, 2007.

[36] Veinott, Olson, Olson, and Fu, (1999) p. 307, Elizabeth S. Veinott, Judith Olson, Gary M. Olson, and Xiaolan Fu. Video helps remote work: speakers who need to negotiate common ground benefit from seeing each other. Concurrent Engineering, vol 15 issue 2, 2007.

[37] Powell, Piccoli and Ives (2004) p.12, Anne Powell, Gabriele Piccoli, and Blake Ives. Virtual teams: a review of current literature and directions for future research. The DATA BASE for Advances in Information Systems - Winter Vol. 35, issue 1, 2004.)

[38] Maznevski and Chudoba (2000) p.489, Martha L. Maznevski, Katherine M. Chudoba, Bridging Space over Time: Global Virtual Team Dynamics and Effectiveness, Organization Science, Vol. 11, issue 5, 2000, p. 473-492.

[39] Powell, Piccoli and Ives (2004) p.11-12, Anne Powell, Gabriele Piccoli, and Blake Ives. Virtual teams: a review of current literature and directions for future research. The DATA BASE for Advances in Information Systems - Winter Vol. 35, issue 1, 2004.)

[40] Ramesh and Dennis (2002) p.219, Ramesh, Venkataraman and Dennis, Alan R., The object-oriented team: Lessons for virtual teams from global software development, System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference, p. 212- 221.

[41] Holland, Gaston and Gomes (2000), Sarah Holland, Kevin Gaston, Jorge Gomes, Critical success factors for cross-functional teamwork in new product development, International Journal of Management Reviews, vol. 2, issue 3, 2000, p.231-259

[42] Majchrzak et al. (2000) p.580-590, Ann Majchrzak, Ronald E. Rice, Arvind Malhotra, Nelson King, Sulin Ba, Technology Adaptation: The Case of a Computer-Supported Inter-Organizational Virtual Team, MIS Quarterly Vol. 24, issue 4, 2000, p. 569-600.

[43] Powell, Piccoli and Ives (2004) p.12-13, Anne Powell, Gabriele Piccoli, and Blake Ives. Virtual teams: a review of current literature and directions for future research. The DATA BASE for Advances in Information Systems - Winter Vol. 35, issue 1, 2004.)
[44] Powell, Piccoli and Ives (2004) p.13, Anne Powell, Gabriele Piccoli, and Blake Ives. Virtual teams: a review of current literature and directions for future research. The DATA BASE for Advances in Information Systems - Winter Vol. 35, issue 1, 2004.)
[45] Tan et .al (2000), Bernard Tan, Kwok-Kee Wei, Wayne Huang, Guet-Ngoh Ng, A Dialogue Technique to Enhance Electronic Communication in Virtual Teams, IEEE Transactions on Professional Communication, Vol. 43, issue 2, 2000, p. 153-165.
[46] Eveland and Bikson (1988) p.368, J. D. Eveland and T. K. Bikson. 1988. Work group structures and computer support: a field experiment. ACM Trans. Inf. Syst. Vol. 6, issue 4, 1988.
[47] Lind (1999) p.280, Mary R .Lind, The gender impact of temporary virtual work groups, Professional Communication, IEEE Transactions on professional communication, vol.42, issue 4, 1999, p.276-285.
[48] http://vizteams.com/
[49] http://workingremote.ly/statistics/
[50] Jury, Alister (2008). Leadership Effectiveness within Virtual Teams: Investigating Mediating and Moderating Mechanisms. PhD Thesis, School of Psychology, The University of Queensland

http://accura-marketing.com/pages/virtual-team.php==Further reading==

- Geographically Dispersed Teams (1999). Valerie Sessa et al. ISBN 1-882197-54-2
- Duarte, D.L., & Snyder, N.T. (2006). Mastering Virtual Teams (3rd ed.). San Francisco: Jossey-Bass. ISBN 0-7879-8280-6
- Hertel, G., Geister, S., & Konradt, U. (2005). Managing virtual teams: A review of current empirical research. Human Resource Management Review, 15, 69-95. ISSN: 1053-4822
- Lipnack, Jessica and Stamps, Jeffrey Virtual Teams. Wiley (2 edition - September 13, 2000) ISBN 0471388254 (http://www.amazon.com/dp/0471388254)
- Wiggins, B.E. (2009, July). Global teams and media selection. In Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications (pp. 705–710). Chesapeake, VA: AACE. Retrieved from http://www.editlib.org/p/31577.
- Konetes, G., & Wiggins, B.E. (2009, September). The effectiveness of virtual teams. In Proceedings of the Laurel Highlands Communications Conference (pp. 11–18). Indiana: Indiana University of Pennsylvania.
- Jarvis, Dana E. (2010) 7 Essentials for Managing Virtual Teams, University Readers, San Diego, CA.
- Carmel, E. and J.A. Espinosa. (2011) I'm Working While They're Sleeping: Time Zone Separation Challenges and Solutions, USA: Nedder Stream Press.
- Zofi, Y. (2011). A Manager's Guide to Virtual Teams (1st ed.). New York, NY: AMACOM. ISBN 0-8144-1659-4

External links

- Virtual Teams (http://www.teambuilding.za.bz/virtual-teams-creating-synergies-through-organizational-partnerships/)Wikipedia:Link rot (Robert Davison, ISWORLD)
- Virtual Teams Articles (http://www.virtualteamsblog.com)
- Virtual team statistics (http://workingremote.ly/statistics/)

Distributed development

A **distributed development** project is a research & development (R&D) project that is done across multiple business worksites or locations. It is a form of R&D where the project members may not see each other face to face, but they are all working collaboratively toward the outcome of the project. Often this is done through email, the Internet and other forms of quick long-distance communication.[1]

It is different from outsourcing because all of the organizations are working together on an equal level, instead of one organization subcontracting the work to another.

It also is similar to, but different from, a virtual team because there is a research element.

Characteristics of distributed development

Location

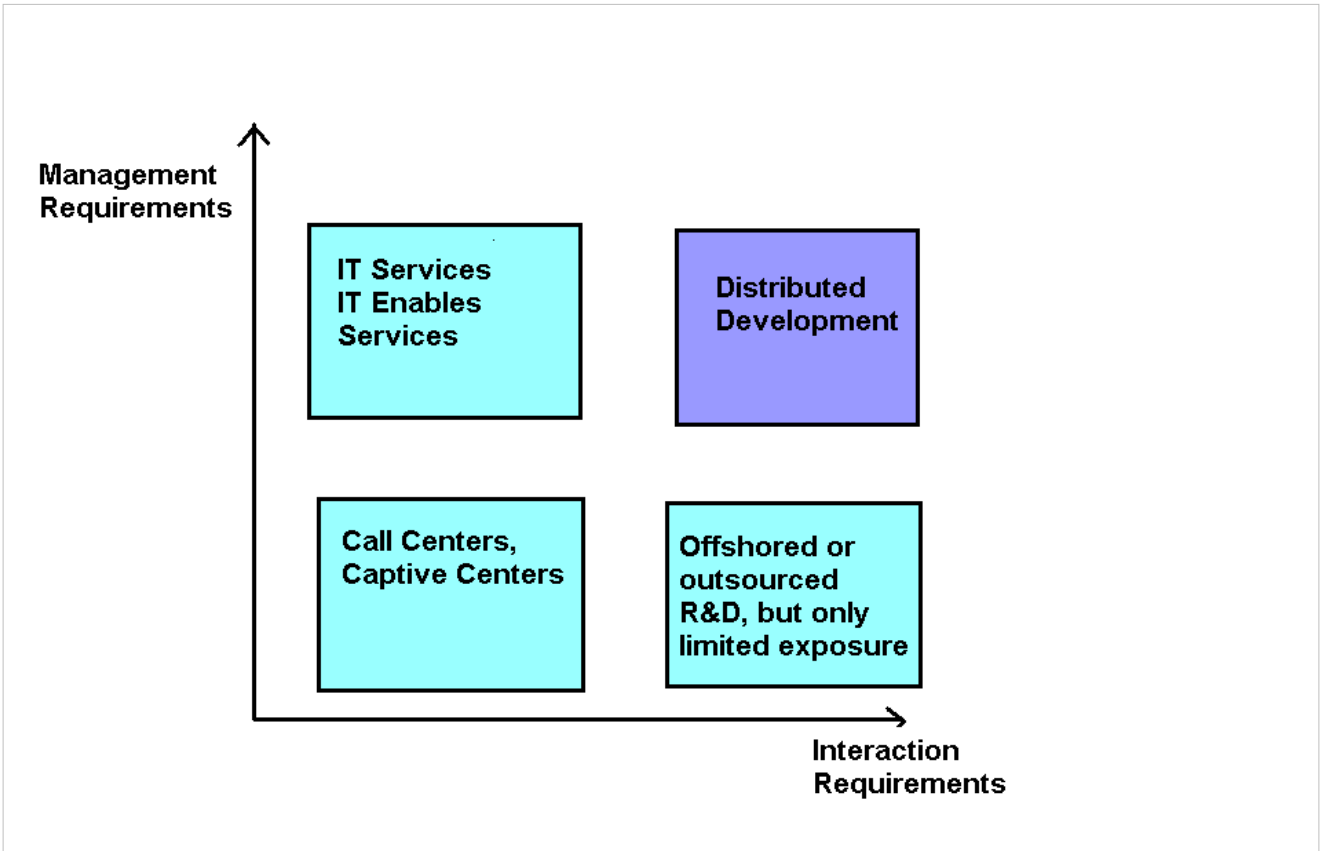
People are distributed across multiple locations and work on the same project or product. The reasons for the distributions do not matter, they might include the availability of resources in different locations, closeness to certain clusters, proximity to customers or cost advantages. Examples could be the production of an Airbus or Boeing aircraft - those are usually done in multiple locations (though by the same company) and assembled finally in one location.

Collaboration

People might specialize in a distributed development environment, but they actively collaborate to achieve the common goal. There must be a program lead or project manager somewhere in the project management mix. In a distributed environment, project members share ideas, information and resources. To get back to the Aircraft example; the Airbus engineers in Hamburg know exactly what their colleagues in Toulouse are doing and they know this well beyond just the interfaces of the pieces they do.

Responsibility and accountability

Everybody feels responsible for the achievement of the overall project goal. Nobody can succeed without everybody else being successful. This is also different from a typical outsourcing project, where every outsourced function just concentrates on (and gets measured against) the actual goals and tasks of that function. This mandatory set-up makes people think about what the "other side" thinks and makes them collaborate and help each other. Again, Airbus in Hamburg can never be successful if the aircraft does not take off. Even if they produce the best fuselage and wings the world has ever seen - the plane is only a success if it flies when assembled.



In summary, distributed development is one of the highest forms of collaboration in any engineering or scientific R&D environment. It is typically not a barrier to business success, but it can range from being somewhat of a burden to difficult to achieve, since it requires high management capabilities, an excellent communication environment, a politically free environment, a highly efficient infrastructure, a well-developed organization chart, and frequent interaction. Most important, management needs to believe in the set-up and put measures in place to reward compliance, as well as be very strict with those who do not comply. According to entrepreneur Mitch Kapor, many companies are doing distributed development successfully.

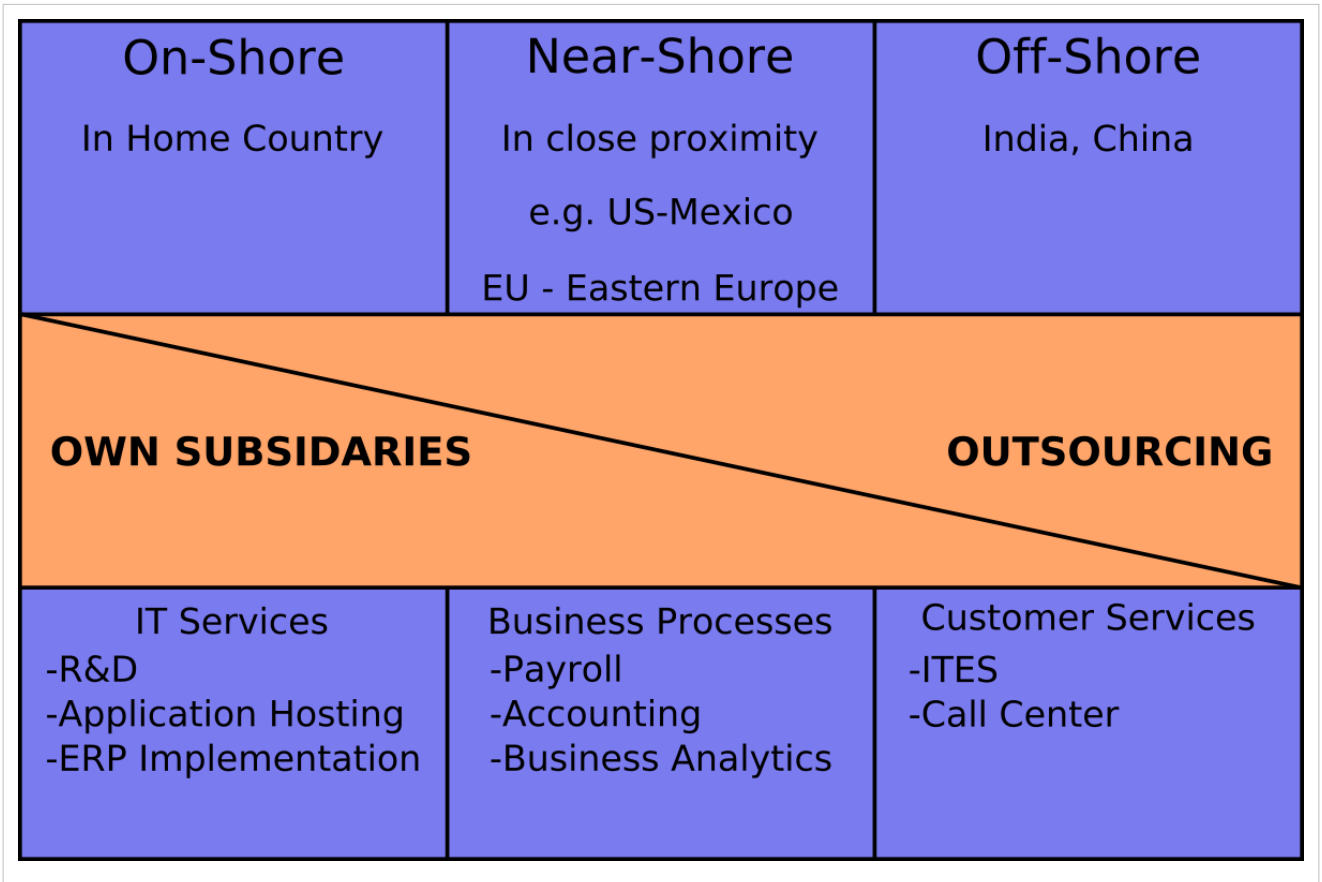
Success factors

There are three main success factors for a distributed development project:

1. Select and/or recruit good, strong, highly skilled people.
2. Spend some money for face-to-face meetings, especially at the beginning of each major project.[]
3. Build an organizational design that supports working in a distributed development, including the right incentive systems.

By doing these three actions, one may obtain advantages beyond pure outsourcing or offshoring, namely much higher motivated employees in all parts of the distributed network, higher retention and certainly one gains from the diversity of the network.

The image below tries to explain how the different pieces fit together. There is a distinction between on-shore, near-shore and off shore and all three can be done either outsources (means by another company) or by owns subsidiaries. In all cases there are different levels of work done outside of the own local boundaries, ranging from simple service to high-end R&D.



References

[1] Mitch Kapor, *How to Build a Successful Company* (http://ecorner.stanford.edu/authorMaterialInfo.html?mid=1901), lecture to Stanford University entrepreneurship students, 2008-01-16.

Redmine

Redmine

REDMINE flexible project management	
Developer(s)	Jean-Philippe Lang
Initial release	0.1.0 / June 25, 2006
Stable release	2.5.1 (March 29, 2014) [±] [1]
Development status	Active
Written in	Ruby on Rails
Operating system	Cross-platform
Type	Project management software, Bug tracking system
License	GNU General Public License v2
Website	www.redmine.org [2]

Redmine is a free and open source, web-based project management and bug-tracking tool. It includes a calendar and Gantt charts to aid visual representation of projects and their deadlines. It handles multiple projects. Redmine provides integrated project management features, issue tracking, and support for various version control systems.

The design of Redmine is significantly influenced by Trac, a software package with some similar features.

Redmine is written using the Ruby on Rails framework. It is cross-platform and cross-database.

Features

- Multiple projects
- Flexible role-based access control
- Flexible issue tracking system
- Gantt chart and calendar
- News, documents & files management
- Feeds & e-mail notifications.
- Per-project wiki
- Per-project forums
- Simple time tracking functionality
- Custom fields for issues, time-entries, projects and users
- SCM integration (SVN, CVS, Git, Mercurial, Bazaar and Darcs)
- Multiple LDAP authentication
- User self-registration
- Multiple languages
- Multiple databases
- Plugins
- REST API

Adoption

Redmine is reported to have more than 80 major installations worldwide. Among the users of Redmine are Gentoo's Summer of Code and Ruby.

Fork

Following concerns with the way the feedback and patches from the Redmine community were being handled a group of Redmine developers created a fork of the project in February 2011. The fork was initially named Bluemine, but changed to ChiliProject after.

Mobile application

MintRedmine, a free Android application, allows user to manage mintRedmine issues and can be used with Redmine server version 2.2 and higher.

Installers and software appliances

- Bitnami app library - installer and/or image (cloud/virtual machine)
- TurnKey Linux Virtual Appliance Library - image (cloud/virtual machine/ISO)

References

[1] http://en.wikipedia.org/w/index.php?title=Template:Latest_stable_software_release/Redmine&action=edit
[2] http://www.redmine.org/

Book references

- Lesyuk, Andriy (2013). *Mastering Redmine*. Packt Publishing. ISBN 978-1-849519-14-4.

External links

- Official website (http://www.redmine.org/)

Bug tracking system

A **bug tracking system** or **defect tracking system** is a software application that keeps track of reported software bugs in software development projects. It may be regarded as a type of issue tracking system.

Many bug tracking systems, such as those used by most open source software projects, allow end-users to enter bug reports directly. Other systems are used only internally in a company or organization doing software development. Typically bug tracking systems are integrated with other software project management applications.

A bug tracking system is usually a necessary component of a good software development infrastructure, and consistent use of a bug or issue tracking system is considered one of the "hallmarks of a good software team".

Components

A major component of a bug tracking system is a database that records facts about known bugs. Facts may include the time a bug was reported, its severity, the erroneous program behavior, and details on how to reproduce the bug; as well as the identity of the person who reported it and any programmers who may be working on fixing it.

Typical bug tracking systems support the concept of the life cycle for a bug which is tracked through status assigned to the bug. A bug tracking system should allow administrators to configure permissions based on status, move the bug to another status, or delete the bug. The system should also allow administrators to configure the bug statuses and to what extent a bug in a particular status can be moved. Some systems will e-mail interested parties, such as the submitter and assigned programmers, when new records are added or the status changes.

Usage

The main benefit of a bug-tracking system is to provide a clear centralized overview of development requests (including both bugs and improvements, the boundary is often fuzzy), and their state. The prioritized list of pending items (often called backlog) provides valuable input when defining the product road map, or maybe just "the next release".

In a corporate environment, a bug-tracking system may be used to generate reports on the productivity of programmers at fixing bugs. However, this may sometimes yield inaccurate results because different bugs may have different levels of severity and complexity. The severity of a bug may not be directly related to the complexity of fixing the bug. There may be different opinions among the managers and architects.

A *local bug tracker (LBT)* is usually a computer program used by a team of application support professionals (often a help desk) to keep track of issues communicated to software developers. Using an LBT allows support professionals to track bugs in their "own language" and not the "language of the developers." In addition, an LBT allows a team of support professionals to track specific information about users who have called to complain — this information may not always be needed in the actual development queue. Thus, there are two tracking systems when an LBT is in place.

Bug tracking systems as a part of integrated project management systems

Bug and issue tracking systems are often implemented as a part of integrated project management systems. This approach allows including bug tracking and fixing in a general product development process, fixing bugs in several product versions, automatic generation of a product knowledge base and release notes.

Distributed bug tracking

Some bug trackers are designed to be used with distributed revision control software. These distributed bug trackers allow bug reports to be conveniently read, added to the database or updated while a developer is offline. Fossil and Veracity both include distributed bug trackers.

Recently, commercial bug tracking systems have also begun to integrate with distributed version control. FogBugz, for example, enables this functionality via the source-control tool, Kiln.

Although wikis and bug tracking systems are conventionally viewed as distinct types of software, ikiwiki can also be used as a distributed bug tracker. It can manage documents and code as well, in an integrated distributed manner. However, its query functionality is not as advanced or as user-friendly as some other, non-distributed bug trackers such as Bugzilla. Similar statements can be made about org-mode, although it is not wiki software as such.

Bug tracking and test management

While traditional test management tools such as HP Quality Center and IBM Rational Quality Manager come with their own bug tracking systems, other tools integrate with popular bug tracking systems.Wikipedia:Citation needed

References

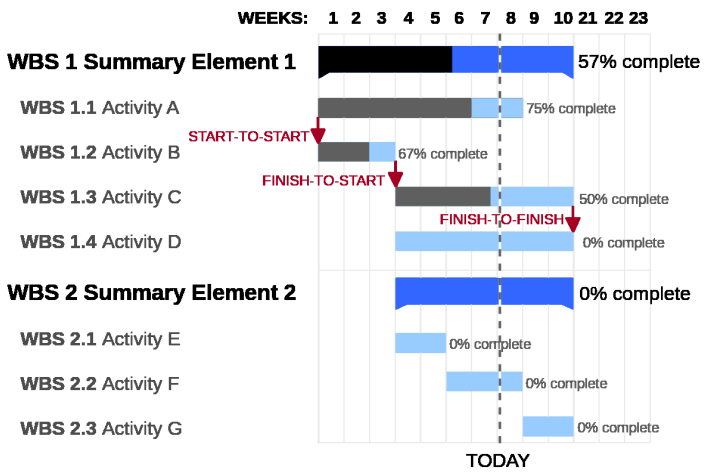
External links

- Bug Tracking Software (http://www.dmoz.org/Computers/Software/Configuration_Management/Bug_Tracking/) at DMOZ
- How to Report Bugs Effectively (http://www.chiark.greenend.org.uk/~sgtatham/bugs.html)
- List of distributed bug tracking software (http://dist-bugs.kitenet.net/software/)

Gantt chart

"Gantt" redirects here. For other uses, see Gantt (disambiguation).

A **Gantt chart** is a type of bar chart, developed by Henry Gantt in the 1910s, that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements comprise the work breakdown structure of the project. Modern Gantt charts also show the dependency (i.e. precedence network) relationships between activities. Gantt charts can be used to show current schedule status using percent-complete shadings and a vertical "TODAY" line as shown here.



A Gantt chart showing three kinds of schedule dependencies (in red) and percent complete indications.

Although now regarded as a common charting technique, Gantt charts were consideredWikipedia:Manual of Style/Words to watch#Unsupported attributions extremely revolutionary when first introduced. This chart is also used in information technology to represent data that has been collected.

Historical development

The first known tool of this type was developed in 1896 by Karol Adamiecki, who called it a *harmonogram*.[1] Adamiecki published his chart in 1931, however, only in Polish, which limited both its adoption and recognition of his authorship. The chart is named after Henry Gantt (1861–1919), who designed his chart around the years 1910–1915.[2][3]

One of the first major applications of Gantt charts was by the United States during World War I, at the instigation of General William Crozier.[4]

In the 1980s, personal computers allowed for widespread creation of complex and elaborate Gantt charts. The first desktop applications were intended mainly for project managers and project schedulers. With the advent of the internet and increased collaboration over networks at the end of the 1990s, Gantt charts became a common feature of web-based applications, including collaborative groupware.

Further applications

Gantt charts can be used for scheduling generic resources, so as well as their use in project management, they can also be used in scheduling production processes and employee rostering. In the latter context, they may also be known as *timebar schedules*.

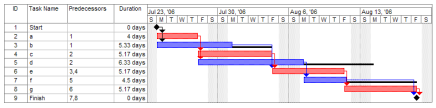
There are many computer applications supporting the use of Gantt charts for employee scheduling, for example, Ganttic. Gantt charts can be used to track shifts or tasks and also vacations or other types of out-of-office time.[5]

Example

In the following example there are seven tasks, labeled *A* through *G*. Some tasks can be done concurrently (*A* and *B*) while others cannot be done until their predecessor task is complete (*C* cannot begin until *A* is complete). Additionally, each task has three time estimates: the optimistic time estimate (*O*), the most likely or normal time estimate (*M*), and the pessimistic time estimate (*P*). The expected time (T_E) is estimated using the beta probability distribution for the time estimates, using the formula $(O + 4M + P) \div 6$.

Activity	Predecessor	Time estimates			Expected time
		Opt. (*O*)	Normal (*M*)	Pess. (*P*)	
A	—	2	4	6	4.00
B	—	3	5	9	5.33
C	*A*	4	5	7	5.17
D	*A*	4	6	10	6.33
E	*B*, *C*	4	5	7	5.17
F	*D*	3	4	8	4.50
G	*E*	3	5	8	5.17

Once this step is complete, one can draw a Gantt chart or a network diagram.



A Gantt chart created using Microsoft Project (MSP). Note (1) the critical path is in red, (2) the slack is the black lines connected to non-critical activities, (3) since Saturday and Sunday are not work days and are thus excluded from the schedule, some bars on the Gantt chart are longer if they cut through a weekend.

References

[1] http://connection.ebscohost.com/c/proceedings/17530521/harmonogram-karol-adamiecki

[2] H.L. Gantt, *Work, Wages and Profit*, published by *The Engineering Magazine*, New York, 1910; republished as *Work, Wages and Profits*, Easton, Pennsylvania, Hive Publishing Company, 1974, ISBN 0-87960-048-9.

[3] Peter W. G. Morris, *The Management of Projects*, Thomas Telford, 1994, ISBN 0-7277-2593-9, Google Print, p.18 (http://books.google.com/books?id=5ekyoWaeZ1UC&pg=PA18-IA7&dq=Adamiecki+Gantt&as_brr=3&sig=xe_RAipoqlvhnu0xLkIsxx-8OAQ)

[4] Wallace Clark and Henry Gantt (1922) *The Gantt chart, a working tool of management* (http://www.archive.org/details/ganttchartworkin00claruoft). New York, Ronald Press.

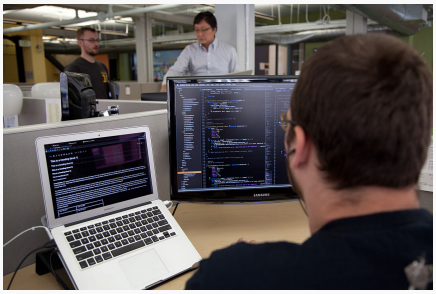
[5] http://ceiton.com/CMS/EN/scheduling/attendance.html#Out-of-Office

External links

- Long-running discussion (http://www.edwardtufte.com/bboard/q-and-a-fetch-msg?msg_id=000076&topic_id=1&topic=Ask E.T.) regarding limitations of the Gantt chart format, and alternatives, on Edward Tufte's website

Chapter 26: Agile App Development with Scrum in a Continuous Delivery Environment

Agile software development

Software development process

A software developer at work
Core activities
• Requirements • Specification • Architecture • Construction • Design • Testing • Debugging • Deployment • Maintenance
Methodologies
• Waterfall • Prototype model • Incremental • Iterative • V-Model • Spiral • Scrum • Cleanroom • RAD • DSDM • RUP • XP • Agile • Lean

• Dual Vee Model • TDD • FDD • DDD • MDD
Supporting disciplines
• Configuration management • Documentation • Quality assurance (SQA) • Project management • User experience
Tools
• Compiler • Debugger • Profiler • GUI designer • Modeling • IDE • Build automation
• v • t • e [1]

Agile software development is a group of software development methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes foreseen tight iterations throughout the development cycle.

The *Agile Manifesto* introduced the term in 2001. Since then, the *Agile Movement*, with all its values, principles, methods, practices, tools, champions and practitioners, philosophies and cultures, has significantly changed the landscape of the modern software engineering and commercial software development in the Internet era.Wikipedia:Citation needed

History

Predecessors

Incremental software development methods have been traced back to 1957. In 1974, a paper by E. A. Edmonds introduced an adaptive software development process. Concurrently and independently the same methods were developed and deployed by the New York Telephone Company's Systems Development Center under the direction of Dan Gielan. In the early 1970s, Tom Gilb started publishing the concepts of Evolutionary Project Management (EVO), which has evolved into Competitive Engineering.[1] During the mid to late 1970s Gielan lectured extensively throughout the U.S. on this methodology, its practices, and its benefits.

Martin Fowler, widely recognized as one of the key founders of the agile methods.

So-called *lightweight* agile software development methods evolved in the mid-1990s as a reaction against the *heavyweight* waterfall-oriented methods, which were characterized by their critics as being heavily regulated, regimented, micromanaged and overly incremental approaches to development.

Proponents of lightweight agile methods contend that they are returning to development practices that were present early in the history of software development.[]

Early implementations of agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development (1997), and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as agile methodologies, after the Agile Manifesto was published in 2001.

The Agile Manifesto

In February 2001, 17 software developers (see below) met at the Snowbird, Utah resort, to discuss lightweight development methods. They published the *Manifesto for Agile Software Development* to define the approach now known as agile software development. Some of the manifesto's authors formed the Agile Alliance, a non-profit organization that promotes software development according to the manifesto's values and principles.

Agile values

The Agile Manifesto reads, in its entirety, as follows:

> *We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
>
> **Individuals and interactions** over Processes and tools
>
> **Working software** over Comprehensive documentation
>
> **Customer collaboration** over Contract negotiation
>
> **Responding to change** over Following a plan
>
> *That is, while there is value in the items on the right, we value the items on the left more.*

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Ken Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brian Marick	

The meanings of the manifesto items on the left within the agile software development context are:

- Individuals and interactions – in agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
- Working software – working software will be more useful and welcome than just presenting documents to clients in meetings.
- Customer collaboration – requirements cannot be fully collected at the beginning of the software development cycle, therefore continuous customer or stakeholder involvement is very important.
- Responding to change – agile development is focused on quick responses to change and continuous development.[2]

Introducing the manifesto on behalf of the Agile Alliance, Jim Highsmith commented that the Agile movement was not opposed to methodology:

> The Agile movement is not anti-methodology, in fact, many of us want to restore credibility to the word methodology. We want to restore a balance. We embrace modeling, but not in order to file some diagram in a dusty corporate repository. We embrace documentation, but not hundreds of pages of never-maintained and rarely-used tomes. We plan, but recognize the limits of planning in a turbulent environment. Those who would brand proponents of XP or SCRUM or any of the other Agile Methodologies as "hackers" are ignorant of both the methodologies and the original definition of the term hacker.
>
> —Jim Highsmith, History: The Agile Manifesto

Agile principles

The Agile Manifesto is based on twelve principles:

1. Customer satisfaction by rapid delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Self-organizing teams
12. Regular adaptation to changing circumstances

Evolutions

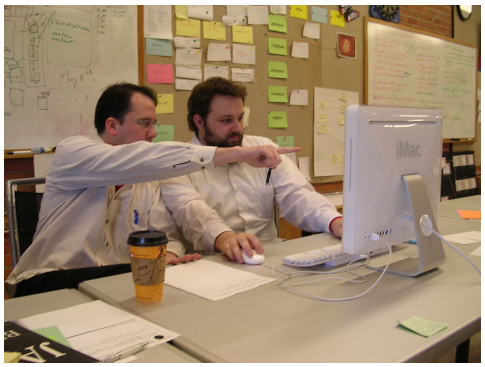
Later, Ken Schwaber with others founded the Scrum Alliance and created the Certified Scrum Master programs and its derivatives. Schwaber left the Scrum Alliance in the fall of 2009, and founded Scrum.org to further improve the quality and effectiveness of Scrum.

In 2005, a group headed by Alistair Cockburn and Jim Highsmith wrote an addendum of project management principles, the Declaration of Interdependence, to guide software project management according to agile development methods.

In 2009, a movement spearheaded by Robert C Martin wrote an extension of software development principles, the Software Craftsmanship Manifesto, to guide agile software development according to professional conduct and mastery.

Overview

There are many specific agile development methods. Most promote development, teamwork, collaboration, and process adaptability throughout the life-cycle of the project.



Pair programming, an agile development technique used by XP. Note information radiators in the background.

Iterative, incremental and evolutionary

Agile methods break tasks into small increments with minimal planning and do not directly involve long-term planning. Iterations are short time frames (timeboxes) that typically last from one to four weeks. Each iteration involves a cross-functional team working in all functions: planning, requirements analysis, design, coding, unit testing, and acceptance testing. At the end of the iteration a working product is demonstrated to stakeholders. This minimizes overall risk and allows the project to adapt to changes quickly. An iteration might not add enough functionality to warrant a market release, but the goal is to have an available release (with minimal bugs) at the end of each iteration. Multiple iterations might be required to release a product or new features.

Efficient and face-to-face communication

No matter what development disciplines are required, each agile team will contain a customer representative, e.g. *Product Owner* in Scrum. This person is appointed by stakeholders to act on their behalf and makes a personal commitment to being available for developers to answer mid-iteration questions. At the end of each iteration, stakeholders and the customer representative review progress and re-evaluate priorities with a view to optimizing the return on investment (ROI) and ensuring alignment with customer needs and company goals.

In agile software development, an *information radiator* is a (normally large) physical display located prominently in an office, where passers-by can see it. It presents an up-to-date summary of the status of a software project or other product. The name was coined by Alistair Cockburn, and described in his 2002 book *Agile Software Development*. A build light indicator may be used to inform a team about the current status of their project.

Very short feedback loop and adaptation cycle

A common characteristic of agile development are daily status meetings or "stand-ups", e.g. *Daily Scrum (Meeting)*. In a brief session, team members report to each other what they did the previous day, what they intend to do today, and what their roadblocks are.

Quality focus

Specific tools and techniques, such as continuous integration, automated unit testing, pair programming, test-driven development, design patterns, domain-driven design, code refactoring and other techniques are often used to improve quality and enhance project agility.

Philosophy

Compared to traditional software engineering, agile development is mainly targeted at complex systems and projects with dynamic, undeterministic and non-linear characteristics, where accurate estimates, stable plans and predictions are often hard to get in early stages, and big up-front designs and arrangements will probably cause a lot of waste, i.e. not economically sound. These basic arguments and precious industry experiences learned from years of successes and failures have helped shape Agile's favor of adaptive, iterative and evolutionary development.

Adaptive vs. Predictive

Development methods exist on a continuum from *adaptive* to *predictive*.[3] Agile methods lie on the *adaptive* side of this continuum. One key of adaptive development methods is a "Rolling Wave" approach to schedule planning, which identifies milestones but leaves flexibility in the path to reach them, and also allows for the milestones themselves to change. Adaptive methods focus on adapting quickly to changing realities. When the needs of a project change, an adaptive team changes as well. An adaptive team will have difficulty describing exactly what will happen in the future. The further away a date is, the more vague an adaptive method will be about what will happen on that date. An adaptive team cannot report exactly what tasks they will do next week, but only which features they plan for next month. When asked about a release six months from now, an adaptive team might be able to report only the mission statement for the release, or a statement of expected value vs. cost.

Predictive methods, in contrast, focus on analysing and planning the future in detail and cater for known risks. In the extremes, a predictive team can report exactly what features and tasks are planned for the entire length of the development process. Predictive methods rely on effective early phase analysis and if this goes very wrong, the project may have difficulty changing direction. Predictive teams will often institute a Change Control Board to ensure that only the most valuable changes are considered.

Risk analysis can be used to choose between adaptive (*agile* or *value-driven*) and predictive (*plan-driven*) methods. Barry Boehm and Richard Turner suggest that each side of the continuum has its own *home ground*, as follows:

Home grounds of different development methods

Agile methods	Plan-driven methods	Formal methods
Low criticality	High criticality	Extreme criticality
Senior developers	Junior developers(?)	Senior developers
Requirements change often	Requirements do not change often	Limited requirements, limited features see Wirth's law
Small number of developers	Large number of developers	Requirements that can be modeled
Culture that responds to change	Culture that demands order	Extreme quality

Iterative vs. Waterfall

One of the differences between agile and waterfall is that testing of the software is conducted at different stages during the software development lifecycle. In the Waterfall model, there is always a separate *testing phase* near the completion of an *implementation phase*. However, in Agile and especially Extreme programming, testing is usually done concurrently with coding, or at least, testing jobs start in early iterations.

Code vs. Documentation

In a letter to IEEE *Computer*, Steven Rakitin expressed cynicism about agile development, calling an article supporting agile software development "yet another attempt to undermine the discipline of software engineering" and translating "Working software over comprehensive documentation" as "We want to spend all our time coding. Remember, real programmers don't write documentation."

This is disputed by proponents of Agile software development, who state that developers should write documentation if that's the best way to achieve the relevant goals, but that there are often better ways to achieve those goals than writing static documentation. Scott Ambler states that documentation should be "Just Barely Good Enough" (JBGE), that too much or comprehensive documentation would usually cause waste, and developers rarely trust detailed documentation because it's usually out of sync with codes, while too little documentation may also cause problems for maintenance, communication, learning and knowledge sharing. Alistair Cockburn wrote of the *Crystal* method:

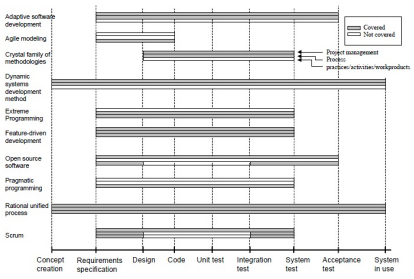
> Crystal considers development to be a series of co-operative games, and the provision of documentation is intended to be enough to help the next win at the next game. The work products for Crystal include use cases, risk list, iteration plan, core domain models, and design notes to inform on choices...however there are no templates for these documents and descriptions are necessarily vague, but the objective is clear, **just enough documentation** for the next game. I always tend to characterize this to my team as: what would you want to know if you joined the team tomorrow.
>
> —Alistair CockburnWikipedia:Verifiability

Agile methods

Well-known agile software development methods and/or process frameworks include:

- Adaptive Software Development (ASD)
- Agile Modeling
- Agile Unified Process (AUP)
- Crystal Methods (Crystal Clear)
- Disciplined Agile Delivery
- Dynamic Systems Development Method (DSDM)
- Extreme Programming (XP)
- Feature Driven Development (FDD)
- Lean software development
- Scrum
- Scrum-ban

The agile methods are focused on different aspects of the Software development life cycle. Some focus on the practices (e.g. XP, Pragmatic Programming, Agile Modeling), while others focus on managing the software projects (e.g. Scrum). Yet, there are approaches providing full coverage over the development life cycle (e.g. DSDM, IBM RUP), while most of them are suitable from the requirements specification phase on (FDD, for example). Thus, there is a clear difference between the various agile methods in this regard.



Software development life-cycle support

Agile practices

Agile development is supported by a bundle of concrete practices suggested by the agile methods, covering areas like requirements, design, modeling, coding, testing, project management, process, quality, etc. Some notable agile practices include:

- Acceptance test-driven development (ATDD)
- Agile Modeling
- Backlogs (Product and Sprint)
- Behavior-driven development (BDD)
- Cross-functional team
- Continuous integration (CI)
- Domain-driven design (DDD)
- Information radiators (Scrum board, Kanban board, Task board, Burndown chart)
- Iterative and incremental development (IID)

- Pair programming
- Planning poker
- Refactoring
- Scrum meetings (Sprint planning, Daily scrum, Sprint review and retrospective)
- Test-driven development (TDD)
- Agile testing
- Timeboxing
- Use case
- User story
- Story-driven modeling
- Velocity tracking

The Agile Alliance has provided a comprehensive online collection with a map guide to the applying agile practices.

Method tailoring

In the literature, different terms refer to the notion of method adaptation, including 'method tailoring', 'method fragment adaptation' and 'situational method engineering'. Method tailoring is defined as:

> A process or capability in which human agents determine a system development approach for a specific project situation through responsive changes in, and dynamic interplays between contexts, intentions, and method fragments.[4]

Potentially, almost all agile methods are suitable for method tailoring. Even the DSDM method is being used for this purpose and has been successfully tailored in a CMM context.[5] Situation-appropriateness can be considered as a distinguishing characteristic between agile methods and traditional software development methods, with the latter being relatively much more rigid and prescriptive. The practical implication is that agile methods allow project teams to adapt working practices according to the needs of individual projects. Practices are concrete activities and products that are part of a method framework. At a more extreme level, the philosophy behind the method, consisting of a number of principles, could be adapted (Aydin, 2004).

Extreme Programming (XP) makes the need for method adaptation explicit. One of the fundamental ideas of XP is that no one process fits every project, but rather that practices should be tailored to the needs of individual projects. Partial adoption of XP practices, as suggested by Beck, has been reported on several occasions.[] Mehdi Mirakhorli [6] proposes a tailoring practice that provides a sufficient road-map and guidelines for adapting all the practices. RDP Practice is designed for customizing XP. This practice, first proposed as a long research paper in the APSO workshop at the ICSE 2008 conference, is currently the only proposed and applicable method for customizing XP. Although it is specifically a solution for XP, this practice has the capability of extending to other methodologies. At first glance, this practice seems to be in the category of static method adaptation but experiences with RDP Practice says that it can be treated like dynamic method adaptation. The distinction between static method adaptation and dynamic method adaptation is subtle.[7]

Comparison with other methods

RAD

Agile methods have much in common with the Rapid Application Development techniques from the 1980/90s as espoused by James Martin and others.Wikipedia:Citation needed In addition to technology-focused methods, customer-and-design-centered methods, such as Visualization-Driven Rapid Prototyping developed by Brian Willison, work to engage customers and end users to facilitate agile software development.Wikipedia:Citation needed

CMMI

In 2008 the Software Engineering Institute (SEI) published the technical report "CMMI or Agile: Why Not Embrace Both" to make clear that the Capability Maturity Model Integration and Agile can co-exist. Modern CMMI-compatible development processes are also iterative. The CMMI Version 1.3 includes tips for implementing Agile and CMMI process improvement together.[8]

Measuring agility

While agility can be seen as a means to an end, a number of approaches have been proposed to quantify agility. *Agility Index Measurements* (AIM) score projects against a number of agility factors to achieve a total. The similarly named *Agility Measurement Index*, scores developments against five dimensions of a software project (duration, risk, novelty, effort, and interaction). Other techniques are based on measurable goals. Another study using fuzzy mathematics[9] has suggested that project velocity can be used as a metric of agility. There are agile self-assessments to determine whether a team is using agile practices (Nokia test, Karlskrona test, 42 points test).

While such approaches have been proposed to measure agility, the practical application of such metrics is still debated. There is agile software development ROI data available from the CSIAC ROI Dashboard.[10]

Experience and adoption

Surveys

One of the early studies reporting gains in quality, productivity, and business satisfaction by using Agile methods was a survey conducted by Shine Technologies from November 2002 to January 2003. A similar survey conducted in 2006 by Scott Ambler, the Practice Leader for Agile Development with IBM Rational's Methods Group reported similar benefits. Others claim that agile development methods are still too young to require extensive academic proof of their success.

Large-scale and distributed Agile

Large-scale agile software development remains an active research area.[11][12] Agile development has been widely seen as being more suitable for certain types of environment, including small teams of experts.:157 Positive reception towards Agile methods has been observed in Embedded domain across Europe in recent years.[13]

Some things that may negatively impact the success of an agile project are:

- Large-scale development efforts (>20 developers), though scaling strategies and evidence of some large projects[14] have been described.
- Distributed development efforts (non-colocated teams). Strategies have been described in *Bridging the Distance* and *Using an Agile Software Process with Offshore Development*.
- Forcing an agile process on a development team.
- Mission-critical systems where failure is not an option at any cost (e.g. software for avionics).

The early successes, challenges and limitations encountered in the adoption of agile methods in a large organization have been documented.

Agile offshore

In terms of outsourcing agile development, Michael Hackett, senior vice president of LogiGear Corporation has stated that "the offshore team ... should have expertise, experience, good communication skills, inter-cultural understanding, trust and understanding between members and groups and with each other."[15]

Criticism

Agile methodologies can be inefficient in large organizations and certain types of projects. Agile methods seem best for developmental and non-sequential projects. Many organizations believe that agile methodologies are too extreme and adopt a hybrid approach that mixes elements of agile and plan-driven approaches.

The term "agile" has also been criticized as being a management fad that simply describes existing good practices under new jargon, promotes a "one size fits all" mindset towards development strategies, and wrongly emphasizes method over results.

Alistair Cockburn organized a celebration of the 10th anniversary of the Agile Manifesto in Snowbird, Utah on February 12, 2011, gathering some 30+ people who'd been involved at the original meeting and since. A list of about 20 elephants in the room ("undiscussable" agile topics/issues) were collected, including aspects: the alliances, failures and limitations of agile practices and context (possible causes: commercial interests, decontextualization, no obvious way to make progress based on failure, limited objective evidence, cognitive biases and reasoning fallacies), politics and culture. As Philippe Kruchten wrote in the end:

> *The agile movement is in some ways a bit like a teenager: very self-conscious, checking constantly its appearance in a mirror, accepting few criticisms, only interested in being with its peers, rejecting en bloc all wisdom from the past, just because it is from the past, adopting fads and new jargon, at times cocky and arrogant. But I have no doubts that it will mature further, become more open to the outside world, more reflective, and also therefore more effective.*

Applications Outside of Software Development

Agile methods have been extensively used for development of software products and some of them use certain characteristics of software, such as object technologies. However, these techniques can be applied to the development of non-software products, such as computers, motor vehicles, medical devices, food, and clothing; see Flexible product development.

Agile development paradigms can be used in other areas of life such as raising children. Its success in child development might be founded on some basic management principles; communication, adaptation and awareness. Bruce Feiler has shown that the basic Agile Development paradigms can be applied to household management and raising children. In his TED Talk, "Agile programming -- for your family" [16], these paradigms brought significant changes to his household environment, such as the kids doing dishes, taking out the trash, and decreasing his children's emotional outbreaks which inadvertently increased their emotional stability. In some ways, agile development is more than a bunch of software development rules; but it can be something more simple and broad, like a problem solving guide...

References

[1] http://www.gilb.com/Project-Management Evolutionary Project Management (EVO)

[2] Ambler, S.W. "Examining the Agile Manifesto" (http://www.ambysoft.com/essays/agileManifesto.html). Retrieved 6 April 2011.

[3] Appendix A, pages 165–194

[4] Aydin, M.N., Harmsen, F., Slooten, K. v., & Stagwee, R. A. (2004). An Agile Information Systems Development Method in use. *Turk J Elec Engin, 12(2),* 127-138

[5] Abrahamsson, P., Warsta, J., Siponen, M.T., & Ronkainen, J. (2003). New Directions on Agile Methods: A Comparative Analysis. *Proceedings of ICSE'03*, 244-254

[6] http://portal.acm.org/citation.cfm?id=1370143.1370149&coll=ACM&dl=ACM&CFID=69442744&CFTOKEN=96226775,

[7] Aydin, M.N., Harmsen, F., Slooten van K., & Stegwee, R.A. (2005). On the Adaptation of An Agile Information(Suren) Systems Development Method. *Journal of Database Management Special issue on Agile Analysis, Design, and Implementation, 16(4),* 20-24

[8] CMMI Product Team, ; CMMI for Development, Version 1.3 (CMU/SEI-2010-TR-033). Software Engineering Institute, Carnegie Mellon University, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm

[9] Kurian, Tisni (2006). Agility Metrics: A Quantitative Fuzzy Based Approach for Measuring Agility of a Software Process, *ISAM-Proceedings of International Conference on Agile Manufacturing'06(ICAM-2006)*, Norfolk, U.S.

[10] CSIAC ROI Dashboard (https://sw.thecsiac.com/databases/roi/) Retrieved 11 November 2011.

[11] Agile Processes Workshop II Managing Multiple Concurrent Agile Projects. Washington: OOPSLA 2002

[12] W. Scott Ambler (2006) Supersize Me (http://www.drdobbs.com/184415491) in Dr. Dobb's Journal, 15 February 2006.

[13] K Petersen's doctoral research in Sweden Implementing Lean and Agile Software Development in Industry (http://www.bth.se/tek/aps/kps.nsf/pages/phd-studies)

[14] Schaaf, R.J. (2007). Agility XL Systems and Software Technology Conference 2007 (http://www.sstc-online.org/Proceedings/2007/pdfs/RJS1722.pdf), Tampa, FL

[15] (http://www.logigear.com/in-the-news/973-agile.html) LogiGear, PC World Viet Nam, Jan 2011

[16] http://www.ted.com/talks/bruce_feiler_agile_programming_for_your_family.html

Further reading

- Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile Software Development Methods: Review and Analysis. (http://agile.vtt.fi/publications.html) *VTT Publications 478*.
- Cohen, D., Lindvall, M., & Costa, P. (2004). An introduction to agile methods. In *Advances in Computers* (pp. 1–66). New York: Elsevier Science.
- Dingsøyr, Torgeir, Dybå, Tore and Moe, Nils Brede (ed.): *Agile Software Development: Current Research and Future Directions* (http://www.amazon.co.uk/Agile-Software-Development-Research-Directions/dp/3642125743), Springer, Berlin Heidelberg, 2010.
- Fowler, Martin. *Is Design Dead?* (http://www.martinfowler.com/articles/designDead.html). Appeared in *Extreme Programming Explained*, G. Succi and M. Marchesi, ed., Addison-Wesley, Boston. 2001.
- Larman, Craig and Basili, Victor R. *Iterative and Incremental Development: A Brief History* IEEE Computer, June 2003 (http://www.highproductivity.org/r6047.pdf)
- Riehle, Dirk. *A Comparison of the Value Systems of Adaptive Software Development and Extreme Programming: How Methodologies May Learn From Each Other* (http://www.riehle.org/computer-science/research/2000/xp-2000.html). Appeared in *Extreme Programming Explained*, G. Succi and M. Marchesi, ed., Addison-Wesley, Boston. 2001.
- Rother, Mike (2009). *Toyota Kata*. McGraw-Hill. ISBN 0-07-163523-8.
- M. Stephens, D. Rosenberg. *Extreme Programming Refactored: The Case Against XP*. Apress L.P., Berkeley, California. 2003. (ISBN 1-59059-096-1)
- Shore, J., & Warden S. (2008). The Art of Agile Development. O'Reilly Media, Inc.
- Willison, Brian (2008). Iterative Milestone Engineering Model. New York, NY.
- Willison, Brian (2008). Visualization Driven Rapid Prototyping. Parsons Institute for Information Mapping.
- Wollmuth, Christine. "Risky Business" (http://requirements.seilevel.com/blog/2013/04/risky-business.html). 2013

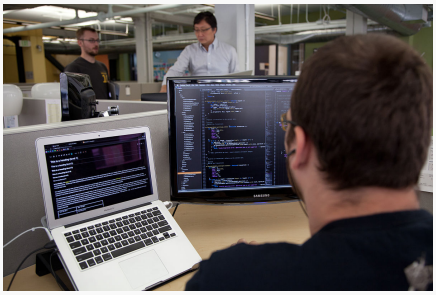
External links

- Agile (http://www.dmoz.org/Computers/Programming/Methodologies/Agile) at DMOZ
- What Is Agile Testing? (http://smartbear.com/products/qa-tools/what-is-agile-testing/)
- Two Ways to Build a Pyramid (http://www.informationweek.com/two-ways-to-build-a-pyramid/6507351), John Mayo-Smith (VP Of Technology At R/GA), October 22, 2001
- The New Methodology (http://martinfowler.com/articles/newMethodology.html) Martin Fowler's description of the background to agile methods
- Ten Authors of The Agile Manifesto Celebrate its Tenth Anniversary (http://www.pragprog.com/magazines/2011-02/agile--)
- Agile Manifesto (http://agilemanifesto.org/)
- Agile Alliance (http://www.agilealliance.org/)
- Scrum.org (http://scrum.org/)
- Scrum Alliance (http://scrumalliance.org/)

Scrum (software development)

This article is about a software development framework. For other uses, see Scrum.

Software development process



A software developer at work

Core activities
• Requirements • Specification • Architecture • Construction • Design • Testing • Debugging • Deployment • Maintenance
Methodologies
• Waterfall • Prototype model • Incremental • Iterative • V-Model • Spiral

• Scrum • Cleanroom • RAD • DSDM • RUP • XP • Agile • Lean • Dual Vee Model • TDD • FDD • DDD • MDD
Supporting disciplines
• Configuration management • Documentation • Quality assurance (SQA) • Project management • User experience
Tools
• Compiler • Debugger • Profiler • GUI designer • Modeling • IDE • Build automation
• v • t • e [1]

Scrum is an iterative and incremental agile software development framework for managing software projects and product or application development. It defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal". It challenges assumptions of the "traditional, sequential approach" to product development. Scrum enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members and daily face to face communication among all team members and disciplines in the project.

A key principle of Scrum is its recognition that during a project the customers can change their minds about what they want and need (often called requirements churn), and that unpredicted challenges cannot be easily addressed in a traditional predictive or planned manner. As such, Scrum adopts an empirical approach—accepting that the problem cannot be fully understood or defined, focusing instead on maximizing the team's ability to deliver quickly and respond to emerging requirements.

History

In rugby football, a scrum refers to the manner of restarting the game after a minor infraction.

Scrum was first defined as "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal" as opposed to a "traditional, sequential approach" in 1986 by Hirotaka Takeuchi and Ikujiro Nonaka in the "New New Product Development Game". Takeuchi and Nonaka later argued in "The Knowledge Creating Company" that it is a form of "organizational knowledge creation, [...] especially good at bringing about innovation continuously, incrementally and spirally".

The authors described a new approach to commercial product development that would increase speed and flexibility, based on case studies from manufacturing firms in the automotive, photocopier and printer industries. They called this the *holistic or rugby approach*, as the whole process is performed by one cross-functional team across multiple overlapping phases, where the team "tries to go the distance as a unit, passing the ball back and forth".

In the early 1990s, Ken Schwaber used what would become Scrum at his company, Advanced Development Methods, and Jeff Sutherland, with John Scumniotales and Jeff McKenna, developed a similar approach at Easel Corporation, and were the first to refer to it using the single word *Scrum*. In 1995, Sutherland and Schwaber jointly presented a paper describing the *Scrum methodology* at the Business Object Design and Implementation Workshop held as part of Object-Oriented Programming, Systems, Languages & Applications '95 (OOPSLA '95) in Austin, Texas, its first public presentation. Schwaber and Sutherland collaborated during the following years to merge the above writings, their experiences, and industry best practices into what is now known as Scrum.

In 2001, Schwaber worked with Mike Beedle to describe the method in the book *Agile Software Development with Scrum*. Its approach to planning and managing projects is to bring decision-making authority to the level of operation properties and certainties. Although the word is not an acronym, some companies implementing the process have been known to spell it with capital letters as SCRUM. This may be due to one of Ken Schwaber's early papers, which capitalized SCRUM in the title.

Later, Schwaber with others founded the Scrum Alliance and created the Certified Scrum Master programs and its derivatives. Ken left the Scrum Alliance in the fall of 2009, and founded Scrum.org to further improve the quality and effectiveness of Scrum.

Roles

There are three core roles and a range of ancillary roles. Core roles are often referred to as *pigs* and ancillary roles as *chickens* (after the story The Chicken and the Pig).

The core roles are those committed to the project in the Scrum process—they are the ones producing the product (objective of the project). They represent the *scrum team*. Although other roles may be encountered in real projects, Scrum does not define any roles other than those described below.

Product Owner

The Product Owner represents the stakeholders and is the voice of the customer. He or she is accountable for ensuring that the team delivers value to the business. The Product Owner writes (or has the team write) customer-centric items (typically user stories), ranks and prioritizes them, and adds them to the product backlog. Scrum teams should have one Product Owner, and while they may also be a member of the development team, this role should not be combined with that of the Scrum Master. In an enterprise environment, though, the Product Owner is often combined with the role of Project Manager as they have the best visibility regarding the scope of work (products).Wikipedia:Citation needed

Role of Product Owner in Defining and Communicating Product Requirements

Communication is a main function of the product owner. The ability to convey priorities and empathize with team members and stakeholders are vital to steer the project in the right direction. Product owners bridge the communication gap between the team and their stakeholders. As Figure 1 [1] shows, they serve as a proxy stakeholder to the development team and as a project team representative to the overall stakeholder community.[2]

As the face of the team to the stakeholders, the following are some of the communication tasks of the product owner to the team: - demonstrates the solution to key stakeholders who were not present in a normal iteration demo - announced releases - communicates team status - organizes milestone reviews - educates stakeholders in the development process - negotiates priorities, scope, funding, and schedule

Empathy is a key attribute for a product owner to have—the ability to put one's self in another's shoes. A product owner will be conversing with different stakeholders in the project— different people, with a variety of backgrounds, job roles, and objectives. A product owner needs to be able to see from these different points of view. To be effective, it would also be wise for a product owner to know the level of detail his audience needs from him. The development team would need thorough feedback and specifications so they build a product up to expectation, while an executive sponsor may just need summaries and cliff notes of progress. Providing more information than necessary may lose their interest and waste time. There is also significant evidence that face-to-face communication around a shared sketching environment is the most effective way to communicate information instead of documentation. A direct means of communication is then most preferred by seasoned agile product owners.[3]

A product owner's ability to communicate effectively is also enhanced by being skilled in techniques that identify stakeholder needs, negotiate priorities between stakeholder interests, and collaborate with developers to ensure effective implementation of requirements.

Development Team

The Development Team is responsible for delivering potentially shippable increments (PSIs) of product at the end of each Sprint (the Sprint Goal). A Team is made up of 3–9 individuals with cross-functional skills who do the actual work (analyse, design, develop, test, technical communication, document, etc.). The Development Team in Scrum is self-organizing, even though there may be some level of interface with project management offices (PMOs).

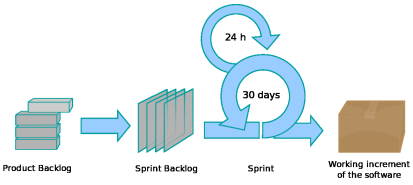
Scrum Master

Scrum is facilitated by a Scrum Master, who is accountable for removing impediments to the ability of the team to deliver the product goals and deliverables. The Scrum Master is not a traditional team lead or project manager, but acts as a buffer between the team and any distracting influences. The Scrum Master ensures that the Scrum process is used as intended. The Scrum Master is the enforcer of the rules of Scrum, often chairs key meetings, and challenges the team to improve. The role has also been referred to as a *servant-leader* to reinforce these dual perspectives.

The Scrum Master differs from a project manager in that the latter may have people management responsibilities unrelated to the role of Scrum Master. The Scrum Master role excludes any such additional people responsibilities. In fact, there is no role of project manager in Scrum at all, because none is needed. The traditional responsibilities of a project manager have been divided up and reassigned among the three Scrum roles, and mostly to the Development Team and the Product Owner, rather than to the Scrum Master. Practicing Scrum with the addition of a project manager indicates a fundamental misunderstanding of Scrum, and typically results in conflicting responsibilities, unclear authority, and sub-optimal results.

Sprint

A sprint (or iteration) is the basic unit of development in Scrum. The sprint is a "timeboxed" effort; that is, it is restricted to a specific duration. The duration is fixed in advance for each sprint and is normally between one week and one month, although two weeks is typical.



The Scrum process

Each sprint is started by a planning meeting, where the tasks for the sprint are identified and an estimated commitment for the sprint goal is made, and ended by a sprint review-and-retrospective meeting, where the progress is reviewed and lessons for the next sprint are identified.

Scrum emphasizes working product at the end of the Sprint that is really "done"; in the case of software, this means a system that is integrated, fully tested, end-user documented, and potentially shippable.

Events

Meetings

Sprint planning meeting

At the beginning of the sprint cycle (every 7–30 days), a "Sprint planning meeting" is held:

- Select what work is to be done
- Prepare the Sprint Backlog that details the time it will take to do that work, with the entire team
- Identify and communicate how much of the work is likely to be done during the current sprint
- Eight-hour time limit
  - (1st four hours) Entire team: dialog for prioritizing the Product Backlog
  - (2nd four hours) Development Team: hashing out a plan for the Sprint, resulting in the Sprint Backlog

Daily Scrum meeting

Each day during the sprint, a project team communication meeting occurs. This is called a *Daily Scrum (meeting)* and has specific guidelines:


A Daily Scrum meeting in the computing room. This choice of location lets the team start on time.

- All members of the development team come prepared with the updates for the meeting.
- The meeting starts precisely on time even if some development team members are missing.
- The meeting should happen at the same location and same time every day.
- The meeting length is set (timeboxed) to 15 minutes.
- All are welcome, but normally only the core roles speak.

During the meeting, each team member answers three questions:

- What have you done since yesterday?
- What are you planning to do today?
- Any impediments/stumbling blocks? Any impediment/stumbling block identified in this meeting is documented by the Scrum Master and worked towards resolution outside of this meeting. No detailed discussions shall happen in this meeting.

End meetings

At the end of a sprint cycle, two meetings are held: the "Sprint Review Meeting" and the "Sprint Retrospective".

At the Sprint Review Meeting:

- Review the work that was completed and the planned work that was not completed
- Present the completed work to the stakeholders (a.k.a. "the demo")
- Incomplete work cannot be demonstrated
- Four-hour time limit

At the Sprint Retrospective:

- All team members reflect on the past sprint
- Make continuous process improvements
- Two main questions are asked in the sprint retrospective: What went well during the sprint? What could be improved in the next sprint?
- Three-hour time limit
- This meeting is facilitated by the Scrum Master

Extensions

Backlog refinement (grooming)

Backlog refinement is the ongoing process of reviewing product backlog items and checking that they are appropriately prioritised and prepared in a way that makes them clear and executable for teams once they enter sprints via the sprint planning activity. Product backlog items may be broken into multiple smaller ones, acceptance criteria may be clarified, or new preparatory work such as clarification on client needs or technical spikes may be identified.

Backlog refinement is not a core Scrum practice but has been adopted as a way of managing the quality of backlog items entering a sprint.

Scrum of Scrums

The Scrum of Scrums (meeting) is a technique to scale Scrum up to large development groups (over a dozen people), which allows clusters of teams to discuss their work, focusing especially on areas of overlap and integration. Each daily scrum within a sub-team ends by designating one member as an "ambassador" to participate in a daily meeting with ambassadors from other teams, called the Scrum of Scrums. Depending on the context, ambassadors may be technical contributors, or each team's Scrum Master.

The agenda will be like a Daily Scrum, with the following four questions:Wikipedia:Citation needed

- What has your team done since we last met?
- What will your team do before we meet again?
- Is anything slowing your team down or getting in their way?
- Are you about to put something in another team's way?

Resolution of impediments is expected to focus on the challenges of coordination between the teams, and may entail agreeing to interfaces between teams, negotiating responsibility boundaries, etc. The Scrum of Scrums will track these working items via a backlog of its own, where each item contributes to improving between-team coordination.

As Jeff Sutherland commented,

> *Since I originally defined the Scrum of Scrums (Ken Schwaber was at IDX working with me), I can definitively say the Scrum of Scrums is not a "meta Scrum". The Scrum of Scrums as I have used it is responsible for delivering the working software of all teams to the Definition of Done at the end of the Sprint, or for releases during the sprint. PatientKeeper delivered to production four times per Sprint.*

> *Ancestry.com delivers to production 220 times per two week sprint. Hubspot delivers live software 100-300 times a day. The Scrum of Scrums Master is held accountable for making this work. So the Scrum of Scrums is an operational delivery mechanism.*

Artifacts

Product backlog

The *product backlog* is an ordered list of *requirements* that is maintained for a product. It consists of features, bug fixes, non-functional requirements, etc.—whatever needs to be done in order to successfully deliver a viable product. The *product backlog items* (PBIs) are ordered by the Product Owner based on considerations like risk, business value, dependencies, date needed, etc.

Items added to a backlog are commonly written in story format. The product backlog is *what* will be delivered, ordered into the sequence in which it should be delivered. It is open and editable by anyone, but the Product Owner is ultimately responsible for ordering the items on the backlog for the Development Team to choose.

The product backlog contains the Product Owner's assessment of business value and the Development Team's assessment of development effort, which are often, but not always, stated in story points using a rounded Fibonacci sequence. These estimates help the Product Owner to gauge the timeline and may influence ordering of backlog items; for example, if the "add spellcheck" and "add table support" features have the same business value, the Product Owner may schedule earlier delivery of the one with the lower development effort (because the ROI (Return on Investment) is higher) or the one with higher development effort (because it is more complex or riskier, and they want to retire that risk earlier).

The product backlog and the business value of each backlog item is the responsibility of the Product Owner. The size (i.e. estimated complexity or effort) of each backlog item is, however, determined by the Development Team, who contributes by sizing items, either in story points or in estimated hours.

There is a common misunderstanding that only user stories are allowed in a Product Backlog. By contrast, Scrum is neutral on requirement techniques. As the Scrum Primer states,

> *Product Backlog items are articulated in any way that is clear and sustainable. Contrary to popular misunderstanding, the Product Backlog does not contain "user stories"; it simply contains items. Those items can be expressed as user stories, use cases, or any other requirements approach that the group finds useful. But whatever the approach, most items should focus on delivering value to customers.*

Scrum advocates that the role of Product Owner be assigned. The Product Owner is responsible for maximizing the value of the product and the work of the Development Team. The Product Owner gathers input, takes feedback and is lobbied by many people, but it will ultimately make the call on what gets built. They are also solely responsible for the management of the backlog.

The product backlog is used to: • capture requests for modifying a product. This can include add new features, replacing old features, removing features and fixing issues; and • ensure the delivery team is given work which maximizes the business benefit to the owner of the product

Typically, the product owner and the SCRUM team come together and write down everything that needs to be prioritized and this becomes content for the first sprint, which is a block of time meant for focused work on selected items that can be accommodated within a timeframe. The SCRUM product backlog is permitted to evolve as new information surfaces about the product and its customers, and so new work are tackled for next sprints.

The following items typically comprise a scrum backlog: features, bugs, technical work, and knowledge acquisition. In the web development sphere, there is confusion as to the difference between a feature and a bug, technically a feature is “wanted”, while a bug is a feature that is “unintended” or “unwanted” (but may not be necessarily a defective thing). An example of technical work would be, “run virus check on all developers workstations”. An

example of knowledge acquisition could be a scrum backlog item about researching Wordpress plugin libraries and making a selection.

Managing the product backlog between product owner and scrum team

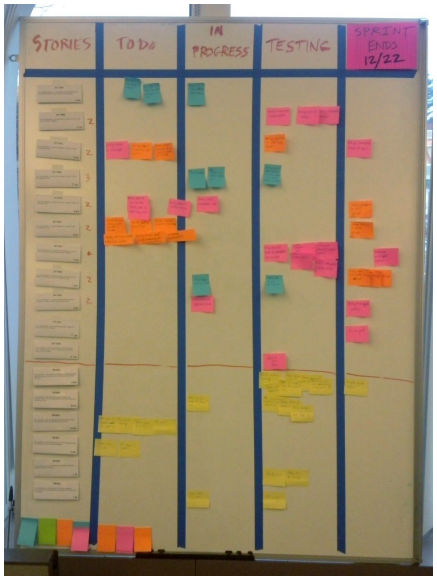
A backlog, in its simplest form, is merely a list of items to be worked on. Having well established rules about how work is added, removed and ordered helps the whole team make better decisions about how to change the product.

The product owner prioritizes which of the product backlog are most needed. The team then chooses which items can be completed in the coming sprint. On the SCRUM board, the team moves items from the product backlog to the sprint backlog, which is the list of items they will build. Conceptually, it is ideal for the team to only select what they think they can accomplish from the top of the list, but it is not unusual to see in practice that teams are able to take lower priority items from the list along with the top ones selected. This normally happens because there is time left within the sprint to accommodate more work. Items at the top of the backlog, the items that are going to be worked on first, should be broken down into stories that are suitable for the delivery team to work on. The further down the backlog goes, the less refined the items should be. As Schwaber and Beedle put it “The lower the priority, the less detail, until you can barely make out the backlog item.”

As the team works through the backlog, it needs to be assumed that “changes in the world can happen”—the team can learn about new market opportunities to take advantage of, competitor threats that arise, and feedback from customers that can change the way the product was meant to work. All of these new ideas tend to trigger the team to adapt the backlog to incorporate new knowledge. This is part of the fundamental mindset of an agile team. The world changes, the backlog is never finished.[4]

Sprint backlog

The *sprint backlog* is the list of work the Development Team must address during the next sprint. The list is derived by selecting product backlog items from the top of the product backlog until the Development Team feels it has enough work to fill the sprint. This is done by the Development Team asking "Can we also do this?" and adding product backlog items to the sprint backlog. The Development Team should keep in mind its past performance assessing its capacity for the new sprint, and use this as a guide line of how much "effort" they can complete.



A Scrum task board

The product backlog items are broken down into tasks by the Development Team. Tasks on the sprint backlog are never assigned; rather, tasks are signed up for by the team members as needed according to the set priority and the Development Team member skills. This promotes self-organization of the Development Team, and developer buy-in.

The sprint backlog is the property of the Development Team, and all included estimates are provided by the Development Team. Often an accompanying *task board* is used to see and change the state of the tasks of the current sprint, like "to do", "in progress" and "done".

Once a Sprint Backlog is committed, no additional functionality can be added to the Sprint backlog except by the team. Once a Sprint has been delivered, the Product Backlog is analyzed and reprioritized if necessary, and the next set of functionality is selected for the next Sprint.

Increment

The *increment* (or *potentially shippable increment*, PSI) is the sum of all the Product Backlog items completed during a sprint and all previous sprints. At the end of a sprint, the Increment must be done according to the Scrum Team's criteria called Definition of Done (DoD). The increment must be in a usable condition regardless of whether the Product Owner decides to actually release it.

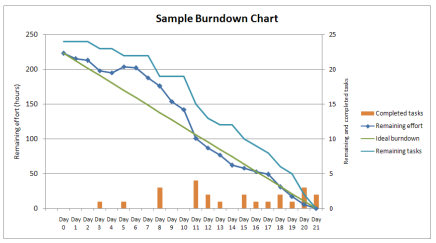
Burndown chart

Main article: burn down chart

The sprint burndown chart is a publicly displayed chart showing remaining work in the sprint backlog. Updated every day, it gives a simple view of the sprint progress. It also provides quick visualizations for reference.

There are also other types of burndown, for example the *release burndown chart* that shows the amount of work left to complete the target commitment for a Product Release (normally spanning through multiple iterations) and the *alternative release burndown chart*, which basically does the same, but clearly shows scope changes to Release Content, by resetting the baseline.

It should not be confused with an earned value chart.



A sample burn down chart for a completed iteration, showing remaining effort and tasks for each of the 21 work days of the 1-month iteration

Terminology

The following terms are often used in a Scrum process.

Scrum Team

: Product Owner, Scrum Master and Development Team

Product Owner

: The person responsible for maintaining the Product Backlog by representing the interests of the stakeholders, and ensuring the value of the work the Development Team does.

Scrum Master

: The person responsible for the Scrum process, making sure it is used correctly and maximizing its benefits.

Development Team

: A cross-functional group of people responsible for delivering potentially shippable increments of Product at the end of every Sprint.

Sprint burn down chart

: Daily progress for a Sprint over the sprint's length.

Release burn down chart

: Sprint level progress of completed product backlog items in the Product Backlog.

Product backlog (PBL)

: A prioritized list of high-level requirements.

Sprint backlog (SBL)

: A prioritized list of tasks to be completed during the sprint.

Sprint

A time period (typically 1–4 weeks) in which development occurs on a set of backlog items that the team has committed to. Also commonly referred to as a Time-box or iteration.

Spike

A time boxed period used to research a concept and/or create a simple prototype. Spikes can either be planned to take place in between sprints or, for larger teams, a spike might be accepted as one of many sprint delivery objectives. Spikes are often introduced before the delivery of large or complex product backlog items in order to secure budget, expand knowledge, and/or produce a proof of concept. The duration and objective(s) of a spike will be agreed between the Product Owner and Delivery Team before the start. Unlike sprint commitments, spikes may or may not deliver tangible, shippable, valuable functionality. For example, the objective of a spike might be to successfully reach a decision on a course of action. The spike is over when the time is up, not necessarily when the objective has been delivered.

Tracer Bullet

The tracer bullet is a spike with the current architecture, current technology set, current set of best practices which results in production quality code. It might just be a very narrow implementation of the functionality but is not throw away code. It is of production quality and the rest of the iterations can build on this code. The name has military origins as ammunition that makes the path of the weapon visible, allowing for corrections. Often these implementations are a 'quick shot' through all layers of an application, such as connecting a single form's input field to the back-end, to prove the layers will connect as expected.Wikipedia:Citation needed

Tasks

Work items added to the sprint backlog at the beginning of a sprint and broken down into hours. Each task should not exceed 12 hours (or two days), but it's common for teams to insist that a task take no more than a day to finish.Wikipedia:Citation needed

Definition of Done (DoD)

The exit-criteria to determine whether a product backlog item is complete. In many cases the DoD requires that all regression tests should be successful. The definition of "done" may vary from one Scrum team to another, but must be consistent within one team.[5]

Velocity

The total effort a team is capable of in a sprint. The number is derived by evaluating the work (typically in user story points) completed from the last sprint's backlog items. The collection of historical velocity data is a guideline for assisting the team in understanding how much work they can do in a future sprint.

Impediment

Anything that prevents a team member from performing work as efficiently as possible.

Sashimi

A term used to describe one or more user stories, indicating that they are thin slices of a product feature or capability.

Abnormal Termination

The Product Owner can cancel a Sprint if necessary. The Product Owner may do so with input from the team, Scrum Master or management. For instance, management may wish to cancel a sprint if external circumstances negate the value of the sprint goal. If a sprint is abnormally terminated, the next step is to conduct a new Sprint planning meeting, where the reason for the termination is reviewed.

ScrumBut

A ScrumBut (or Scrum But) is an exception to the "pure" Scrum methodology, where a team has changed the methodology to adapt it to their own needs.

Scrum-ban

Scrum-ban is a software production model based on Scrum and Kanban. Scrum-ban is especially suited for maintenance projects or (system) projects with frequent and unexpected work items or programming errors. In such cases the time-limited sprints of the Scrum model are of no appreciable use, but Scrum's daily meetings and other practices can be applied, depending on the team and the situation at hand. Visualization of the work stages and limitations for simultaneous unfinished work and defects are familiar from the Kanban model. Using these methods, the team's workflow is directed in a way that allows for minimum completion time for each work item or programming error, and on the other hand ensures each team member is constantly employed.

To illustrate each stage of work, teams working in the same space often use post-it notes or a large whiteboard. In the case of decentralized teams, stage-illustration software such as Assembla, ScrumWorks, TargetProcess, Rational Team Concert, Eylean board, OnTime, Kanban Tool or JIRA in combination with Jira Agile can be used to visualize each team's product backlog items, defects and tasks divided into separate phases.

In their simplest, the tasks are categorized into the work stages:

- Unstarted
- Ongoing
- Completed

If desired, though, the teams can add more stages of work (such as "defined", "designed", "tested" or "delivered"). These additional phases can be of assistance if a certain part of the work becomes a bottleneck and the limiting values of the unfinished work cannot be raised. A more specific task division also makes it possible for employees to specialize in a certain phase of work.

There are no set limiting values for unfinished work. Instead, each team has to define them individually by trial and error; a value too small results in workers standing idle for lack of work, whereas values too high tend to accumulate large amounts of unfinished work, which in turn hinders completion times. A rule of thumb worth bearing in mind is that no team member should have more than two simultaneous selected tasks, and that on the other hand not all team members should have two tasks simultaneously.

The major differences between Scrum and Kanban are derived from the fact that, in Scrum, work is divided into sprints that last a certain amount of time, whereas in Kanban the workflow is continuous. This is visible in work stage tables, which in Scrum are emptied after each sprint. In Kanban all tasks are marked on the same table. Scrum focuses on teams with multifaceted know-how, whereas Kanban makes specialized, functional teams possible.

Since Scrum-ban is such a new development model, there is not much reference material. Kanban, on the other hand, has been applied by Microsoft and Corbis.

Tooling

Like other agile methods, Scrum can be implemented through a wide range of tools.

Many companies use universal tools, such as spreadsheets to build and maintain artifacts such as the sprint backlog. There are also open-source and proprietary packages dedicated to management of products under the Scrum process. Other organizations implement Scrum without the use of any tools, and maintain their artifacts in hard-copy forms such as paper, whiteboards, and sticky notes.

Limitations

Hybridization of Scrum is common as Scrum does not cover the whole product development lifecycle; therefore, organizations find the need to add in additional processes to create a more comprehensive implementation. For example, at the start of the project, organizations commonly add process guidance on requirements gathering and prioritization, initial high-level design, and budget and schedule forecasting. However, the Scrum framework does not explicitly allow for extension points of such a kind; consequently, achieving a more comprehensive software life cycle requires extending the framework rather than instantiating it.

Assessments and certifications

There are two organizations that provide Scrum assessments and/or certifications (in order of founding).

Scrum Alliance

Scrum Alliance provides four certifications for Scrum practitioners: CSM (Certified ScrumMaster), CSPO (Certified Scrum Product Owner), CSD (Certified Scrum Developer), and CSP (Certified Scrum Professional). Courses are required for initial certifications of CSM, CSPO, and CSD.

Three more certifications are provided for Scrum trainers or coaches: CST (Certified Scrum Trainer), CSC (Certified Scrum Coach), and REP (Registered Education Provider). CSTs are CSPs who are also qualified Scrum teachers or instructors, while CSCs are CSPs who have helped one or more organizations effectively apply Scrum. REPs are organizations authorized to provide Scrum and Agile-related training and courses leading to the CSD credential.

Scrum.org

Scrum.org provides four families of assessments: free Open Assessments (the Scrum Open and the Developer Open), PSM Assessments (PSM I and PSM II), PSD Assessment (PSD I) and PSPO Assessments (PSPO I and PSPO II). Those who achieve a minimum passing score will receive certification, after taking PSM, PSD, or PSPO assessments.

Scrum.org offers the option of Professional Scrum Master (PSM) I and II and Professional Scrum Developer (PSD I) certifications to the public, based on the perception that certification should be available to all those who possess a particular level of knowledge—not only to those who have taken a class. The Professional Scrum Product Owner (PSPO) assessments currently require a course taking, and will be open to the public once their associated Bodies of Knowledge are formalized.

References

[1] http://agilemodeling.com/images/productOwner.jpg

[2] Pichler, Roman. Agile product management with Scrum : creating products that customers love. Upper Saddle River, NJ: Addison-Wesley, 2010.

[3] Cohn, Mike. Succeeding with agile : software development using Scrum. Upper Saddle River, NJ: Addison-Wesley, 2010.

[4] Pichler, Roman. Agile product management with Scrum : creating products that customers love. Upper Saddle River, NJ: Addison-Wesley, 2010.

[5] Ken Schwaber, *Agile Project Management with Scrum*, p.55

Further reading

- Jeff Sutherland; Ken Schwaber (2013). "The Scrum Guide" (https://www.scrum.org/Portals/0/Documents/Scrum Guides/2013/Scrum-Guide.pdf). Scrum.org. Retrieved July 2013.
- N.S. Janoff; L. Rising (2000). "The Scrum Software Development Process for Small Teams" (http://members.cox.net/risingl1/Articles/IEEEScrum.pdf). Retrieved March 15, 2007.

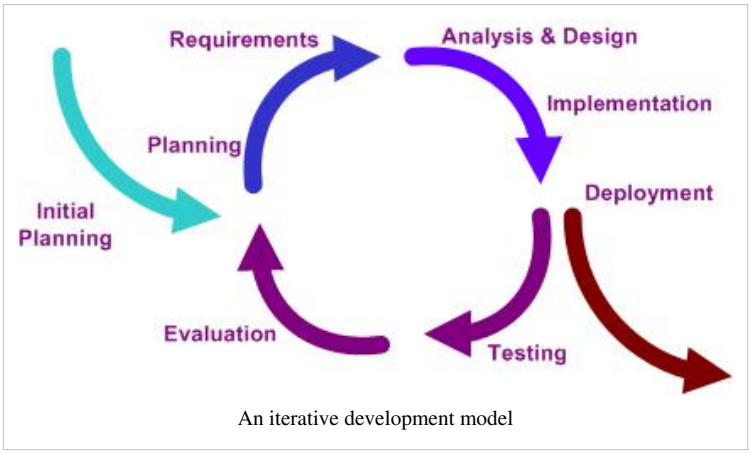
- Deemer, Pete; Benefield, Gabrielle; Larman, Craig; Vodde, Bas (2009). "The Scrum Primer" (http://scrumtraininginstitute.com/home/stream_download/scrumprimer). Retrieved June 1, 2009.
- Kniberg, Henrik. "Scrum and XP from the Trenches" (http://www.infoq.com/minibooks/scrum-xp-from-the-trenches). Retrieved August 9, 2010.
- Münch, Jürgen; Armbrust, Ove; Soto, Martín; Kowalczyk, Martin (2012). "Software Process Definition and Management" (http://www.springer.com/computer/swe/book/978-3-642-24290-8). Retrieved July 16, 2012.
- Ambler, Scott (2013). "Going Beyond Scrum: Disciplined Agile Delivery" (http://disciplinedagileconsortium.org/Resources/Documents/BeyondScrum.pdf). Retrieved February 4, 2014.

External links

- Scrum.org, the home of Scrum (http://scrum.org/)
- Scrum Alliance, non-profit (http://www.scrumalliance.org/)
- Agile Alliance's Scrum library (http://cf.agilealliance.org/articles/article_list.cfm?CategoryID=17)
- A Scrum Process Description (http://epf.eclipse.org/wikis/scrum/) by the Eclipse Process Framework (EPF) Project (http://www.eclipse.org/epf)
- BPMN process diagram of Scrum (http://www.ariscommunity.com/users/sstein/2010-08-09-bpm-view-scrum)

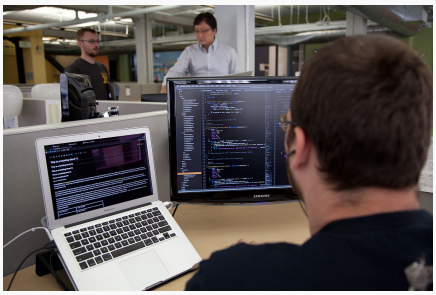
Iterative and incremental development

Iterative and Incremental development is any combination of both iterative design or iterative method and incremental build model for software development. The combination is of long standing and has been widely suggested for large development efforts. For example, the 1985 DOD-STD-2167[1] mentions (in section 4.1.2): "During software development, more than one iteration of the software development cycle may be in progress at the same time." and "This process may be described as an 'evolutionary acquisition' or 'incremental build' approach." The relationship between iterations and increments is determined by the overall software development methodology and software development process. The exact number and nature of the particular incremental builds and what is iterated will be specific to each individual development effort.



An iterative development model

Software development process



A software developer at work

Core activities

- Requirements
- Specification
- Architecture
- Construction
- Design
- Testing
- Debugging
- Deployment
- Maintenance

Methodologies

- Waterfall
- Prototype model
- Incremental
- Iterative
- V-Model
- Spiral
- Scrum
- Cleanroom
- RAD
- DSDM
- RUP
- XP
- Agile
- Lean
- Dual Vee Model
- TDD
- FDD
- DDD
- MDD

Supporting disciplines

- Configuration management
- Documentation
- Quality assurance (SQA)
- Project management
- User experience

Tools

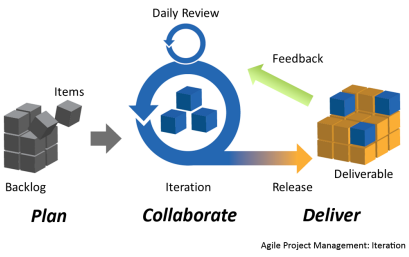
• Compiler • Debugger • Profiler • GUI designer • Modeling • IDE • Build automation	
	• v • t • e [1]

Iterative and incremental development are essential parts of the Modified waterfall models, Rational Unified Process, Extreme Programming and generally the various agile software development frameworks.

It follows a similar process to the plan-do-check-act cycle of business process improvement.

Overview

The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental), allowing software developers to take advantage of what was learned during development of earlier parts or versions of the system. Learning comes from both the development and use of the system, where possible key steps in the process start with a simple implementation of a subset of the software requirements and iteratively enhance the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added.

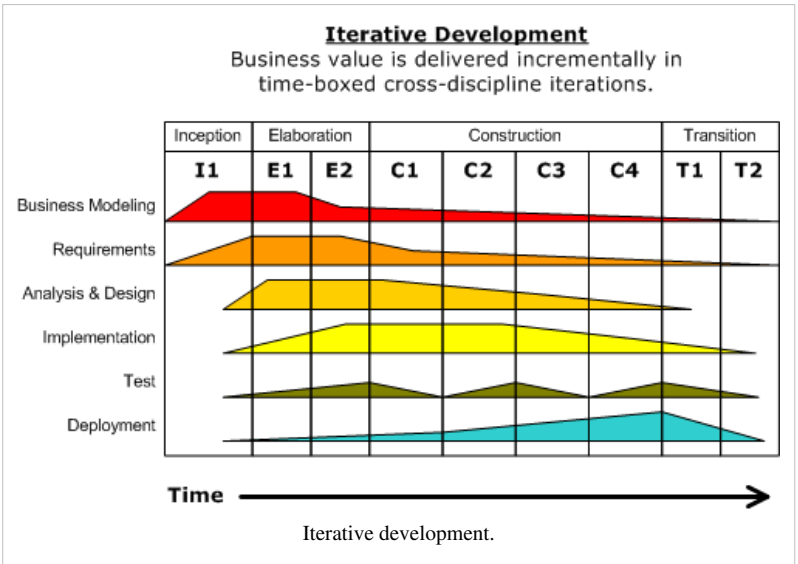


A simplified version of a typical iteration cycle in agile project management

The procedure itself consists of the initialization step, the iteration step, and the Project Control List. The initialization step creates a base version of the system. The goal for this initial implementation is to create a product to which the user can react. It should offer a sampling of the key aspects of the problem and provide a solution that is simple enough to understand and implement easily. To guide the iteration process, a project control list is created that contains a record of all tasks that need to be performed. It includes such items as new features to be implemented and areas of redesign of the existing solution. The control list is constantly being revised as a result of the analysis phase.

The iteration involves the redesign and implementation of iteration is to be simple, straightforward, and modular, supporting redesign at that stage or as a task added to the project control list. The level of design detail is not dictated by the iterative approach. In a light-weight iterative project the code may represent the major source of documentation of the system; however, in a critical iterative project a formal Software Design Document may be used. The analysis of an iteration is based upon user feedback, and the program analysis facilities available. It involves analysis of the structure, modularity, usability, reliability, efficiency, & achievement of goals. The project control list is modified in light of the analysis results.

Phases



Iterative development.

Incremental development slices the system functionality into increments (portions). In each increment, a slice of functionality is delivered through cross-discipline work, from the requirements to the deployment. The unified process groups increments/iterations into phases: inception, elaboration, construction, and transition.

- Inception identifies project scope, requirements (functional and non-functional) and risks at a high level but in enough detail that work can be estimated.
- Elaboration delivers a working architecture that mitigates the top risks and fulfills the non-functional requirements.
- Construction incrementally fills-in the architecture with production-ready code produced from analysis, design, implementation, and testing of the functional requirements.
- Transition delivers the system into the production operating environment.

Each of the phases may be divided into 1 or more iterations, which are usually time-boxed rather than feature-boxed. Architects and analysts work one iteration ahead of developers and testers to keep their work-product backlog full.

Usage

Many examples of early usage are provided in Craig Larman and Victor Basili's article "Iterative and Incremental Development: A Brief History",[2] with one of the earliest being NASA's 1960s Project Mercury.

Another is an "early and striking example of a major IID success is the very heart of NASA's space shuttle software—the primary avionics software system, which FSD built from 1977 to 1980. The team applied IID in a series of 17 iterations over 31 months, averaging around eight weeks per iteration. Their motivation for avoiding the waterfall life cycle was that the shuttle program's requirements changed during the software development process".

Some organizations, such as the US Department of Defense, have a preference for iterative methodologies, starting with MIL-STD-498 "clearly encouraging evolutionary acquisition and IID".

The current DoD Instruction 5000.2, released in 2000, states a clear preference for IID: "There are two approaches, evolutionary and single step [waterfall], to full capability. An evolutionary approach is preferred. … [In this] approach, the ultimate capability delivered to the user is divided into two or more blocks, with increasing increments of capability...software development shall follow an iterative spiral development process in which continually expanding software versions are based on learning from earlier development." it can also be done in phases .

Contrast with Waterfall development

Programming paradigms

- Action
- Agent-oriented
- Aspect-oriented
- Automata-based
- Concurrent computing
  - Relativistic programming
- Data-driven
- Declarative (contrast: Imperative)
  - Constraint
  - Dataflow
    - Flow-based
    - Cell-oriented (spreadsheets)
    - Reactive
  - Functional
    - Functional logic
  - Logic
    - Abductive logic
    - Answer set
    - Constraint logic
    - Functional logic
    - Inductive logic
- End-user programming
- Event-driven
  - Service-oriented
  - Time-driven
- Expression-oriented
- Feature-oriented
- Function-level (contrast: Value-level)
- Generic
- Imperative (contrast: Declarative)
  - Procedural
- Language-oriented
  - Natural language programming
  - Discipline-specific
  - Domain-specific
  - Grammar-oriented
    - Dialecting
  - Intentional
- Metaprogramming

- Automatic
- Reflective
  - Attribute-oriented
- Homoiconic
- Template
  - Policy-based
- Non-structured (contrast: Structured)
  - Array
- Nondeterministic
- Parallel computing
  - Process-oriented
- Point-free style
  - Concatenative
- Semantic
- Structured (contrast: Non-structured)
  - Block-structured
  - Modular (contrast: Monolithic)
  - Object-oriented (OOP)
    - By separation of concerns:
      - Aspect-oriented
      - Role-oriented
      - Subject-oriented
    - Class-based
    - Prototype-based
  - Recursive
- Value-level (contrast: Function-level)
- Probabilistic
- Concept
- v
- t
- e [3]

Waterfall development completes the project-wide work-products of each discipline in one step before moving on to the next discipline in the next step. Business value is delivered all at once, and only at the very end of the project. Backtracking is possible in an iterative approach.

Implementation guidelines

Guidelines that drive the implementation and analysis include:

- Any difficulty in design, coding and testing a modification should signal the need for redesign or re-coding.
- Modifications should fit easily into isolated and easy-to-find modules. If they do not, some redesign is possibly needed.
- Modifications to tables should be especially easy to make. If any table modification is not quickly and easily done, redesign is indicated.
- Modifications should become easier to make as the iterations progress. If they are not, there is a basic problem such as a design flaw or a proliferation of patches.
- Patches should normally be allowed to exist for only one or two iterations. Patches may be necessary to avoid redesigning during an implementation phase.
- The existing implementation should be analyzed frequently to determine how well it measures up to project goals.
- Program analysis facilities should be used whenever available to aid in the analysis of partial implementations.
- User reaction should be solicited and analyzed for indications of deficiencies in the current implementation.

Notes

[1] DOD-STD-2167 Defense Systems Software Development (04 JUN 1985) (http://www.everyspec.com/DoD/DoD-STD/DOD-STD-2167_278/) on everyspec.com

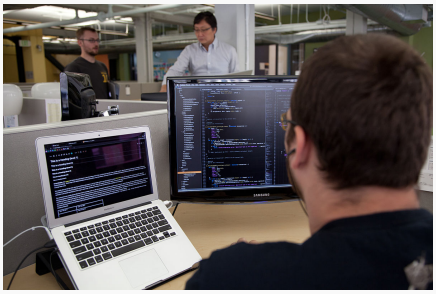
[2] Iterative and Incremental Development: A Brief History (http://doi.ieeecomputersociety.org/10.1109/MC.2003.1204375), Craig Larman and Victor Basili, IEEE Computer, June 2003

[3] http://en.wikipedia.org/w/index.php?title=Template:Programming_paradigms&action=edit

References

- Dr. Alistair Cockburn (May 2008). "Using Both Incremental and Iterative Development" (http://www.crosstalkonline.org/storage/issue-archives/2008/200805/200805-Cockburn.pdf). *STSC CrossTalk* (USAF Software Technology Support Center) **21** (5): 27–30. ISSN 2160-1593 (http://www.worldcat.org/issn/2160-1593). Retrieved 2011-07-20.
- Craig Larman, Victor R. Basili (June 2003). "Iterative and Incremental Development: A Brief History" (http://www.craiglarman.com/wiki/downloads/misc/history-of-iterative-larman-and-basili-ieee-computer.pdf). *IEEE Computer* (IEEE Computer Society) **36** (6): 47–56. doi: 10.1109/MC.2003.1204375 (http://dx.doi.org/10.1109/MC.2003.1204375). ISSN 0018-9162 (http://www.worldcat.org/issn/0018-9162). Retrieved 2009-01-10.

User story

Software development process

A software developer at work
Core activities
• Requirements • Specification • Architecture • Construction • Design • Testing • Debugging • Deployment • Maintenance
Methodologies
• Waterfall • Prototype model • Incremental • Iterative • V-Model • Spiral • Scrum • Cleanroom • RAD • DSDM • RUP • XP • Agile • Lean • Dual Vee Model • TDD • FDD • DDD • MDD
Supporting disciplines

• Configuration management • Documentation • Quality assurance (SQA) • Project management • User experience
Tools
• Compiler • Debugger • Profiler • GUI designer • Modeling • IDE • Build automation
• v • t • e [1]

In software development and product management, a **user story** is one or more sentences in the everyday or business language of the end user or user of a system that captures what a user does or needs to do as part of his or her job function. User stories are used with agile software development methodologies as the basis for defining the functions a business system must provide, and to facilitate requirements management. It captures the 'who', 'what' and 'why' of a requirement in a simple, concise way, often limited in detail by what can be hand-written on a small paper notecard.

History

User stories originated with Extreme Programming (XP), whose first written description in 1998 only claimed that customers defined project scope "with user stories, which are like use cases". Rather than offered as a distinct practice, they were described as one of the "game pieces" used in the planning game. However, most of the further literature thrust around all the ways arguing that user stories are "unlike" use cases, in trying to answer in a more practical manner "how requirements are handled" in XP and more generally Agile projects. This drives the emergence, over the years, of a more sophisticated account of user stories.

In 2001, Ron Jeffries proposed the well-known Three C's formula, i.e. *Card, Conversation, Confirmation*, to capture the components of a user story:

A *Card* (or often a Post-it note) is a physical token giving tangible and durable form to what would otherwise only be an abstraction;

A *Conversation* is taking place at different time and places during a project between the various stakeholders concerned by the given feature (customers, users, developers, testers, etc.), which is largely verbal but most often supplemented by documentation;

The *Confirmation*, the more formal the better, ensures that the objectives the conversation revolved around have been reached finally.

Creating user stories

User stories are written by or for business users or customers as a primary way to influence the functionality of the system being developed. User stories may also be written by developers to express non-functional requirements (security, performance, quality, etc.), though primarily it is the task of a product manager to ensure user stories are captured.

When the time comes for creating user stories, one of the developers gets together with a customer representative, e.g. a product manager (or product owner in Scrum), which has the responsibility for formulating the user stories. The developer may use a series of questions to get the customer representative going, such as asking about the desirability of some particular functionality, but must take care not to dominate the idea-creation process.

As the customer representative conceives a user story, it is written down on a note card (e.g. 3x5 inches or 8x13 cm) with a name and a brief description. If the developer and the customer representative find a user story deficient in some way (too large, complicated, or imprecise), it is rewritten until satisfactory - often using the INVEST guidelines. Commonly, user stories are not to be definite once they have been written down, since requirements tend to change throughout the development lifecycle, which agile processes handles by not carving them in stone upfront.

Format

A team at Connextra developed the traditional user-story template in 2001:[1]

> "As a <*role*>, I want <*goal/desire*> so that <*benefit*>"

Mike Cohn, a well-known author on user stories, regards the "so that" clause as optional:[2]

> "As a <*role*>, I want <*goal/desire*>"

Chris Matts suggested that "hunting the value" was the first step in successfully delivering software, and proposed this alternative as part of Feature Injection:[3]

> "In order to <*receive benefit*> as a <*role*>, I want <*goal/desire*>"

Another template based on the Five Ws specifies:

> "As <*who*> <*when*> <*where*>, I <*what*> because <*why*>."

Examples

Search for customers

> As a user, I want to search for my customers by their first and last names.

Modify schedules

> As a non-administrative user, I want to modify my own schedules but not the schedules of other users.

Run tests

> As a mobile application tester, I want to test my test cases and report results to my management.

Start application with last edit

> The application begins by bringing up the last document the user was working with.
>
> *Or*
>
> As a user, I want to start an application with the last edit.

Close application

> As a user closing the application, I want to be prompted to save if I have made any change in my data since the last save.
>
> *Or*

Upon closing the application, the user is prompted to save (when ANYTHING has changed in data since the last save!).

Or

As a user closing the application, I want to be prompted to save anything that has changed since the last save so that I can preserve useful work and discard erroneous work.

Enter expenses

The consultant will enter expenses on an expense form. The consultant will enter items on the form like expense type, description, amount, and any comments regarding the expense. At any time the consultant can do any of the following options:

(1) When the consultant has finished entering the expense, the consultant will “Submit”. If the expense is under fifty (<50), the expense will go directly to the system for processes.

(2) In the event the consultant has not finished entering the expense, the consultant may want to “Save for later”. The entered data should then be displayed on a list (queue) for the consultant with the status of “Incomplete”.

(3) In the event the consultant decides to clear the data and close the form, the consultant will “Cancel and exit”. The entered data will not be saved anywhere.

Usage

As a central part of many agile development methodologies, such as in XP's planning game, user stories define what has to be built in the software project. User stories are prioritized by the customer (or the product owner in Scrum) to indicate which are most important for the system and will be broken down into tasks and estimated by the developers.

When user stories are about to be implemented, the developers should have the possibility to talk to the customer about it. The short stories may be difficult to interpret, may require some background knowledge or the requirements may have changed since the story was written.

Every user story must at some point have one or more acceptance tests attached, allowing the developer to test when the user story is done and also allowing the customer to validate it. Without a precise formulation of the requirements, prolonged nonconstructive arguments may arise when the product is to be delivered.

Benefits

User stories offer a quick way of handling customer requirements without having to create formalized requirement documents and without performing administrative tasks related to maintaining them. A project will gather user stories in order to respond faster and with less overhead to rapidly changing real-world requirements.

XP and other agile methodologies favor face-to-face communication over comprehensive documentation and quick adaptation to change instead of fixation on the problem. User stories achieve this by:

- extreme brevity: they represent small chunks of business value which a programmer can implement in a period of days to weeks
- allowing a developer and the client representative to discuss requirements throughout the project lifetime
- needing very little maintenance.
- being considered only at the time of use
- allowing the breaking of projects into small increments
- suitability for projects which have volatile or poorly understood requirements: iterations of discovery drive the refinement process
- making it easier to estimate development effort

- maintaining a close customer contact
- requiring close customer contact throughout the project so that the most valued parts of the proposed software system get implemented

Limitations

Limitations of user stories include:

Scale-up problem

User stories written on small physical cards are hard to maintain, and difficult to scale to large projects with plenty of complex requirements.

Vague, informal and incomplete

User story cards are regarded as conversation starters. One of the main limitations of user story is that it is open for many interpretations, since a user story, especially the card text containing scripts which focuses on a user's functional goal, is simple but often vague. It does not state the details of user interactions with the system, while customers may have different understandings about the exact story from the team. As a result, it is hard to only use the user story (card) to act as a formal agreement between stakeholders. That is why it is suggested in Extreme Programming that a customer or end user representative is always a part of the team, so that they can communicate and clarify any doubt through direct face-to-face communication.

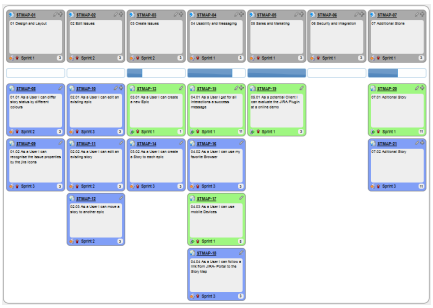
A user story is an *informal* statement of a requirement as long as its correspondence of acceptance testing procedures is lacking. Typically in Extreme Programming before a user story is to be implemented, appropriate acceptance tests or procedures must be written by the customer to ensure by testing the goals of the user story have been fulfilled. Some formalization finally happens when the developer accepts the user story and the acceptance procedures as a work-specific order.

Lack of non-functional requirements

The simplicity of a user story also causes another limitation. It usually has no performance or non-functional requirement details. When dealing with user stories which become the basis of user acceptance tests, often, only functional or business logic will be tested, while performance and non-functional tests (e.g. response time) are prone to be overlooked. To overcome this, developers or testers need to pay attention to specifications such as product quality attributes and system constraints when developing acceptance tests. These additional requirement documents have to be developed and maintained carefully as they will complement user stories.

Story map

A story map is a graphical, two-dimensional product backlog. At the top of the map are big user stories, which are sometimes "epics"Wikipedia:Please clarify as Mike Cohn describes them, and other times correspond to "themes"Wikipedia:Please clarify or "activities". These grouping units are created by orienting at the user's workflow or "the order you'd explain the behavior of the system". Vertically, below the epics, the actual story cards are allocated and ordered by priority. The first horizontal row is a "walking skeleton" and below that represents increasing sophistication.Wikipedia:Please clarify



A story map in action

In this way it becomes possible to describe even big systems without losing the big picture.

Comparing with use cases

A use case has been described as “a generalized description of a set of interactions between the system and one or more actors, where an actor is either a user or another system”.[4] While both user stories and use cases serve the purpose to capture user requirements in terms of interactions between the user and the system, there are several differences between them.

	User Stories	**Use Cases**
Similarities	• Generally formulated in users' everyday language. They should help the reader understand what the software should accomplish. • Must be accompanied by acceptance testing procedures (acceptance criteria) for clarification of behavior where ambiguous.	• Written in users' everyday business language, to facilitate stakeholder communications. • Must be accompanied and verifiable by test cases.
Differences	• XP stories (and similar things, often called features) break requirements into chunks for planning purposes. Stories are explicitly broken down until they can be estimated as part of XP's release planning process. • Provide a small-scale and easy-to-use presentation of information, with little detail, thus remaining open to interpretation, through conversations with on-site customers. • Usually written on small note cards. • Stories are usually more fine-grained because they have to be entirely buildable within an iteration (one or two weeks for XP).	• Use cases organize requirements to form a narrative of how users relate to and use a system. Hence they focus on user goals and how interacting with a system satisfies the goals. • Use case flows describe sequences of interactions, and may be worded in terms of a formal model. A use case is intended to provide sufficient detail for it to be understood on its own. • Usually delivered in a stand-alone document, and visualized by UML diagrams. • A small use case may correspond entirely to a story; however a story might be one or more scenarios in a use case, or one or more steps in a use case.

Kent Beck, Alistair Cockburn, Martin Fowler and others discussed this topic further on the c2.com wiki (the home of extreme programming).

References

[1] http://agilecoach.typepad.com/photos/connextra_user_story_2001/connextrastorycard.html
[2] http://www.mountaingoatsoftware.com/blog/advantages-of-the-as-a-user-i-want-user-story-template
[3] http://antonymarcano.com/blog/2011/03/fi_stories/
[4] Advantages of User Stories for Requirements (http://www.mountaingoatsoftware.com/articles/27-advantages-of-user-stories-for-requirements)

Further reading

- Daniel H. Steinberg, Daniel W. Palmer, *Extreme Software Engineering*, Pearson Education, Inc., ISBN 0-13-047381-2.
- Mike Cohn, *User Stories Applied*, 2004, Addison Wesley, ISBN 0-321-20568-5.
- Mike Cohn, *Agile Estimating and Planning*, 2006, Prentice Hall, ISBN 0-13-147941-5.
- Business Analyst Times http://www.batimes.com/articles/user-stories-and-use-cases-dont-use-both.html

Article Sources and Contributors

VirtualBox *Source*: http://en.wikipedia.org/w/index.php?oldid=605784108 *Contributors*: A:-)Brunuś, AVRS, Acdx, AddWittyNameHere, Aeluwas, Agentlame, Alan0098, Alisha.4m, AlistairMcMillan, Althepal, Andy Dingley, Aoidh, Archimaredes on a train, Arctrebortic, Arichnad, Artem-S-Tashkinov, Artkagor, Barnet.Herts, Baron1984, BenJWoodcroft, Biker Biker, Bmecoli, Bob Re-born, Boudhayan Gupta, Bradkittenbrink, Brianski, Bruce89, Bugaevc, CFeyecare, China Crisis, Chris Ssk, Chris the speller, Ciaran H, Codename Lisa, CommonsDelinker, Confusionball, Cpiral, Craig Macomber, Crazycomputers, Curmudgeonry, Cyzor, DGaryGrady, DOSGuy, DalekClock, Daminstant, DanielVonEhren, Darkmorpher, David Chisnall, Dcorry, Demonkoryu, Denniss, DivineOmega, Dnstest, DoomMaster, Doomonyou, Doubledose 2, Dougher, DrSeehas, DragonLord, E rulez, EagleOne, EdgeOfEpsilon, El aprendelenguas, ElectronicsEnthusiast, Epbr123, EqualRights, EvilHom3r, EwokiWiki, Favonian, FleetCommand, Frap, Free Software Knight, GSK, Galactic Dominator, GandalfDaGraay, GarrettJeanes, Ghepeu, Greenrd, Greg L, Gronau, Gsdefender2, Guyjohnston, Götz, HPSCHD, Halsteadk, Hekerui, Helix84, Hidro, Hildebrand us, Hugo 87, Ian Vaughan, Ikar.us, Inservibile, Instigator uk, IronGargoyle, Iusta, Ivanusto, JCDenton2052, JGXenite, JLaTondre, Jamieostrich, Jandalhandler, Jarble, Jasper Deng, Jdthood, Jeff G., Jengelh, Jerebin, JeremyWJ, Jirijanata, Julesd, JulianYap, KAMiKAZOW, Kiore, Kipaihoski, Kl4m, Klingoncowboy4, Koman90, Kvng, Lester, Liangsuilong, LifeInTheDarkNight, Limulus, Little Professor, Lkundrak, Loopakoopa, Lopifalko, Lordgabor, M4gnum0n, MadNav, Majeru, Mark Renier, MarkusHagenlocher, Mars fenix, Masoris, Matthew V Ball, Med, Medovina, Melnakeeb, Merbenz, Michaelkpate, Mindeye, Minikola, Mnemo, Mojo Hand, Momposi, Mortense, Mprove, Musan, Mwtoews, Mxxcon, Myworld.lostin, NapoliRoma, Nbrouard, Neilh79, Nil Einne, NotinREALITY, O, Ohnoitsjamie, OmegaElheats, One Of Seven Billion, OnionBulb, Ost316, Palosirkka, PentiumPro, Petru Dimitriu, Phatom87, PhilipO, Phobos11, Piandcompany, Piculo, Pinecar, Purrete, Quadunit404, Raysonho, Reisio, Rgb9000, Rgiltrap, Rich Farmbrough, Rig0, Rmsuperstar99, RoestVrijStaal, Rōnin, SF007, SHARD, Samvtran, SasakiSumire, Sbiki, Sbmeirow, Schapel, Scorpus57, ScotXW, Sdornan, Seashorewiki, Seliopou, Seryo93, Setu, Sfiga, Silas S. Brown, Simple Bob, Snarespenguin, Solbu, Someone's Moving Castle, Starionwolf, Stephen Gilbert, SteveLetwin, Superhappyfuntime, Tabledhote, Tarabo, Tatsh, Technologov, Teknomancer, TerryE, Tesssla, TheChampionMan1234, TheFuzzball, TheRingess, Theone256, Thewikipedian, Thisara.d.m, Thumperward, TimTay, Toehead2001, TranslucentCloud, Trasz, TwoTwoHello, UU, Ujoimro, UncleDouggie, Uzume, Vadreb, Veliath, W Nowicki, Wikievil666, Wikimike2007, Winterst, Woohookitty, Xiaq, Xojo, Xpclient, Ysangkok, Ytrewq17, Z killemall, Zero Thrust, Zippy, Zirconscot, Zundark, 326 anonymous edits

Vagrant (software) *Source*: http://en.wikipedia.org/w/index.php?oldid=608021531 *Contributors*: Benjaminoakes, Bunnyhop11, Cuadraman, Dsimic, Fco.plj, Frap, Jdthood, Keltroth, Klimov, NicatronTg, PabloCastellano, Raude, Ruby Murray, ScotXW, Tedickey, Wilbysuffolk, 14 anonymous edits

Puppet (software) *Source*: http://en.wikipedia.org/w/index.php?oldid=607704609 *Contributors*: Ajaxfiore, Ale jrb, Arto B, Benjaminoakes, Bovineone, Brandorr, Carlp101, Chatterboxer, Compfreak7, DanielPharos, David Gerard, DavidDouthitt, Demiurge1000, Demonkoryu, Diomidis Spinellis, Dsimic, Elassint, Frap, Gacq, HighKing, Hulten, Hunnur, Jeremyb, Joannaspark, Jonik, Joy, Kai Burghardt, Kennethbarber, Kgunnam, Klokie, Larstobi, Le Deluge, MarcelButucea, Martarius, Martin Dluhos, Matthaus.litteken, Mike.lifeguard, Moreati, MosesMendoza, NoridelZeus, Puppetlabs, Ranamalo, Raude, Raysonho, Schneelocke, ScotXW, SncBlue, Stahnma, Stephan Leeds, SteveLoughran, Stwalkerster, Thomei08, Thumperward, Thyrsus, Trtrmitya, Widefox, Zundark, 58 anonymous edits

Software performance testing *Source*: http://en.wikipedia.org/w/index.php?oldid=604770784 *Contributors*: AMbroodEY, Abhasingh.02, AbsolutDan, Aeusoes1, Aisteco, Alex Vinokur, Andreas Kaufmann, Andy Dingley, Apodelko, Argyriou, Armadillo-eleven, Bbryson, Bourgeoisspy, Brian.a.wilson, Burakseren, CaroleHenson, Cit helper, Ckoenigsberg, Coroberti, D6, David Johnson, Davidschmelzer, Deicool, Delete12, Dhiraj1984, Dwvisser, Dzmzh, Edepriest, Eitanklein75, Filadifei, Freek Verkerk, Ghewgill, Gnowor, Grotendeels Onschadelijk, GururajOaksys, Gururajs, Hagoth, HenryJames141, Hooperbloob, Hu12, Ianmolynz, Iulus Ascanius, J.delanoy, JaGa, Jdlow1, Jeremy Visser, Jewbacca, Jim1138, Jncraton, Jvkiet, KAtremer, Kbustin00, Keepwish, Ken g6, KnowledgeOfSelf, Lauramocanita, M.boli, M4gnum0n, MER-C, Maimai009, Makesalive, Matt Crypto, Matthew Stannard, MelbourneStar, Michig, MrOllie, Mrmatiko, Msadler, Muhandes, Mywikicontribs, Nono64, Notinasnaid, Noveltywh, Ocaasi, Oliver Lineham, Optakeover, Pinecar, Pnieloud, Pratheepraj, R'n'B, Ravialluru, Raysecurity, Rjwilmsi, Robert Merkel, Ronbarak, Ronz, Rsbarber, Rstens, Rwalker, SchreiberBike, Sebastian.Dietrich, Sfgiants1995, ShelfSkewed, Shimser, Shirtwaist, Shoeofdeath, SimonP, Smalljim, Softlogica, Solstan, SunSw0rd, Swtechwr, Timgurto, Veinor, Versageek, Vrenator, Wahab80, Walter Görlitz, Weregerbil, Widr, Wilsonmar, Wizzard, Wktsugue, Woohookitty, Wselph, 346 anonymous edits

Black-box testing *Source*: http://en.wikipedia.org/w/index.php?oldid=607624827 *Contributors*: A bit iffy, A'bad group, AKGhetto, Aervanath, Ag2402, AndreniW, Andrewpmk, Ash, Asparagus, Avi260192, Benito78, Betterusername, Blake-, Bluebloodpole, CWY2190, Caesura, Canterbury Tail, Chris Pickett, Chrys, Clarkcj12, ClementSeveillac, Cnwilliams, Colinky, Courcelles, DRogers, DanDoughty, Daveydweeb, Deb, Discospinster, DividedByNegativeZero, Docboat, Donner60, DylanW, Ebde, Electiontechnology, Epim, Erkan Yilmaz, ErkinBatu, Fluzwup, Frap, Gayathri nambiar, Geeoharee, Ginsuloft, Happy Attack Dog, Haymaker, Hooperbloob, Hu12, Hugh.glaser, Ian Pitchford, Ileshko, Incognito668, Isnow, Jack Greenmaven, Jamesx12345, Jarble, JimVC3, Jmabel, Jondel, Karl Naylor, Kgf0, Khym Chanur, Kuru, LOL, Lahiru k, Lambchop, Liao, Mark.murphy, Mathieu, Michael Hardy, Michig, Mpilaeten, Mr Minchin, MrOllie, NEUrOO, NawlinWiki, NickW557, Nitinqai, Notinasnaid, Nschoot, OlEnglish, Otheus, PAS, PerformanceTester, Picaroon, Pinecar, Poor Yorick, Pradameinhoff, PupidoggCS, Radiojon, Retiono Virginian, Rich Farmbrough, Rstens, Rsutherland, Rwwww, S.K., Sergei, Shadowjams, Shijaz, Sietec, Solar Police, Solde, Subversive.sound, SuperMidget, Tedickey, TheyCallMeHeartbreaker, Thumperward, Tobias Bergemann, Toddst1, UnitedStatesian, WJBscribe, Walter Görlitz, Widr, Xaosflux, Zephyrjs, 258 anonymous edits

Functional testing *Source*: http://en.wikipedia.org/w/index.php?oldid=607855655 *Contributors*: Abdull, Anonymous monkey, AvicAWB, Ben b, Bgwhite, Chris Pickett, Culix, DRogers, Decrease789, Ego White Tray, GTBacchus, Gareth Griffith-Jones, Gilliam, Hemantrathii, Henri662, Inna Z, Jamesx12345, Javamen, Jesse V., Khalid hassani, Ontist, Randhirreddy, Rp, SiPlus, Softwareqa, Tentinator, Toorbit, Wotan, Yintan, 61 anonymous edits

Smoke testing (software) *Source*: http://en.wikipedia.org/w/index.php?oldid=602548167 *Contributors*: Op47, Qwertyus, Walter Görlitz, 1 anonymous edits

Load testing *Source*: http://en.wikipedia.org/w/index.php?oldid=607561236 *Contributors*: 5nizza, Abarkth99, AbsolutDan, AnonymousDDoS, Archdog99, AreYouFreakingKidding, ArrowmanCoder, BD2412, Bbryson, Beland, Belmond, Bernard2, Bgwhite, Bkranson, BluePyth, CanadianLinuxUser, Christian Paulsen, Crossdader, Ctcdiddy, Czei, DanielaSZTBM, DanielaSzt1, Danykurian, Daonguyen95, Derby-ridgeback, Dhiraj1984, El Tonerino, Emumt, Ettrig, Faught, Ff1959, Frontaal, Gadaloo, Gail, Gaius Cornelius, Gbegic, Gene Nygaard, Gilliam, Gordon McKeown, Gururajs, Hooperbloob, Hu12, Icairns, In.Che., Informationh0b0, Itsyousuf, JHunterJ, JaGa, Jo.witte, Joe knepley, Jpg, Jpo, Jruuska, Ken g6, LinguistAtLarge, Loadfocus, Loadtracer, Lotje, M4gnum0n, MER-C, Magioladitis, Manzee, MarijnN72, Mean as custard, Merrill77, Michig, Mikerowan007, Misterlump, NameIsRon, Nettiehu, Nimowy, Nurg, PerformanceTester, Philip2001, Photodeus, Pinecar, Pushtotest, Radagast83, Ravialluru, Rgraf, Rjwilmsi, Rklawton, Rlonn, Rlsheehan, Robert.maclean, Ronwarshawsky, Rstens, S.K., Scoops, ScottMasonPrice, Shadowjams, Shadriner, Sharmaprakher, Shashi1212, Shilpagpt, Shinhan, SireenOMari, Smith02885, SpigotMap, Swtechwr, Testgeek, Theopolisme, Tusharpandya, Veinor, VernoWhitney, Wahab80, Walter Görlitz, Whitejay251, Wilsonmar, Woohookitty, Wrp103, Yossin, Zje80, 212 anonymous edits

Scenario testing *Source*: http://en.wikipedia.org/w/index.php?oldid=603273715 *Contributors*: Abdull, Alai, Bobo192, Brandon, Cekli829, Chris Pickett, Cindamuse, Epim, Hu12, JaGa, Karbinski, Kingpin13, Kku, Kuru, Muon, Nimmalik77, Pas007, Pinecar, Ronz, Rp, Sainianu088, Shepard, Smtchahal, Surfer43, Tikiwont, Walter Görlitz, Yintan, தென்காசி சுப்பிரமணியன், 38 anonymous edits

Test plan *Source*: http://en.wikipedia.org/w/index.php?oldid=605071406 *Contributors*: -Ril-, Aaronbrick, Aecis, Alynna Kasmira, AndrewStellman, AuburnPilot, Bashnya25, Charles Matthews, Craigwb, Dave6, Downsize43, Drable, E Wing, Foobaz, Freek Verkerk, Grantmidnight, Hennessey, Patrick, Hongooi, Icbkr, Ismarc, Jaganathcfs, Jason Quinn, Jeff3000, Jgorse, Jlao04, Kbpkumar, Ken tabor, Ketiltrout, Kindx, Kitdaddio, LogoX, M4gnum0n, MarkSweep, Matthew Stannard, Mellissa.mcconnell, Michig, Mk*, Moonbeachx, NHSavage, NSR, Niceguyedc, OllieFury, Omicronpersei8, OndraK, Oriwall, Padma vgp, Pedro, Pine, Pinecar, RJFJR, RL0919, Randhirreddy, Rlsheehan, Roshanoinam, Rror, SWAdair, Schmiteye, Scope creep, Shadowjams, SimonP, Stephenb, Tgeairn, The Thing That Should Not Be, Theopolisme, Thunderwing, Thv, Uncle Dick, Wacko, Waggers, Walter Görlitz, Widr, Yparedes, 371 anonymous edits

Apache JMeter *Source*: http://en.wikipedia.org/w/index.php?oldid=589320847 *Contributors*: Cam809, Ciphergoth, Dionysostom, Doug Bell, Ehn, Faisal.akeel, Falcon8765, Funsocaltiger, Grendelkhan, KAtremer, Kbdank71, Kelly Martin, Kingstonclee, Kuru, Lesser Cartographies, M4gnum0n, MKrivosheev, Materialscientist, Milamberspace, Mortense, Mtmoore321, Mumblingmutant, Pinecar, Pmouawad, RedWolf, Roadmr, SimonP, StuartMoncrieff, SunSw0rd, SvGeloven, TZ master, Tiddly Tom, Undera, Walter Görlitz, Woohookitty, Yutsi, 46 anonymous edits

Regression testing *Source*: http://en.wikipedia.org/w/index.php?oldid=608587592 *Contributors*: 7, A.amitkumar, Abarkth99, Abdull, Abhinavvaid, Ahsan.nabi.khan, Alan ffm, AliveFreeHappy, Amire80, Andrew Eisenberg, Anorthup, Antonielly, Baccyak4H, Benefactor123, Boongoman, Brenda Kenyon, Cabalamat, Carlos.l.sanchez, Cdunn2001, Chris Pickett, Chris the speller, Christian75, Cnwilliams, DRogers, Dacian.epure, Deb, Dee Jay Randall, Designatevoid, Doug.hoffman, Eewild, Elsendero, Emj, Enti342, EricEnfermero, Estyler, Forlornturtle, G0gogcsc300, Gregbard, Hadal, Hector224, Henri662, Herve272, HongPong, Hooperbloob, Iiiren, Jacob grace, Jarble, Jwoodger, Kamarou, Kesla, Kmincey, L Kensington, Labalius, LandruBek, Luckydrink1, MER-C, Marijn, Mariotto2009, Materialscientist, Matthew Stannard, Maxwellb, Menzogna, Michaelas10, Michig, MickeyWiki, Mike Rosoft, MikeLynch, Msillil, NameIsRon, Neilc, Neurolysis, Noq, Parvuselephantus, Philipchiappini, Pinecar, Qatutor, Qfissler, Ravialluru, Robert Merkel, Rsavenkov, Ryans.ryu, S3000, SchreyP, Scoops, Snarius, Softzen, Spock of Vulcan, SqueakBox, Srittau, Strait, Svick, Swtechwr, Throwaway85, Thv, Tobias Bergemann, Tobias Hoevekamp, Toon05, Urhixidur, Vsync, Walter Görlitz, Will Beback Auto, Wlievens, Zhenqinli, Zvn, 223 anonymous edits

Web browser engine *Source*: http://en.wikipedia.org/w/index.php?oldid=601023571 *Contributors*: 16@r, A3e6u9, AlistairMcMillan, Anárion, AwamerT, Aydee, Barsamin, Bismar007, Bitbit, Bokthorian varakreivi Juhana kolmastoista nuorempi, Bomazi, Butch566, Camilo Sanchez, Che090572, Comp.arch, Crimsonmargarine, D'Agosta, Dadu, Daniel.Cardenas, EEMIV, Ed g2s, Fenring, Ferrer.jorge, Frap, GHULBEDDINHEKMATIAR, Geniac, Georgekuttykurian, Guru-45, Gyrobo, Gywst, Hervegirod, Hm2k, Hooperbloob, Hydrox, Jeraphine Gryphon, Joliv, Jor, Julesd, K.lee, Karada, Kgyt, Khlo, Klausness, LinguistAtLarge, LittleBenW, MK8, Mabdul, Melah Hashamaim, Miernik, Mindmatrix, Minghong, Mjb, Nbarth, Ncpdupage, Nekohan, Nickg,

Ozmigor, RedWolf, Sami1989, Samuell, Tarquin, Tbhotch, Tokei-so, Toussaint, UU, UberMan5000, Ultramandk, User 7777, Vaganyik, Valodzka, Voidvector, Vystrix Nexoth, Wiki-dude, Winterst, Wschlitz, WulfTheSaxon, Yugsdrawkcabeht, Zef, İncelemeelemani, 69 anonymous edits

Comparison of web browser engines *Source*: http://en.wikipedia.org/w/index.php?oldid=606312658 *Contributors*: 100110100, 1wolfblake, AVRS, Adam Nohejl, Aditya.m4, Akhristov, AlistairMcMillan, Andy Dingley, Ant, Antidrugue, Anárion, Ashertg, Barefootguru, Barsamin, Bobblewik, Chris the speller, ChristTrekker, Comp.arch, Connor Behan, CyberSkull, D'Agosta, Darc, David Latapie, Dhraakellian, Donhalcon, DrSeehas, Dreamertan, Duckbill, Ed g2s, El T, Fenring, Ftiercel, Georgij Michaliutin, GoldenTorc, GreenReaper, Greenrd, GreyWanderer, GreyWyvern, Guaka, Gudeldar, Gyrferret, Gyrobo, Götz, Hawaiian717, Hm2k, Howcome, Ice Ardor, JJay, Jarble, Jayflux, Jeff schiller, Jerryobject, Karnesky, KellyGAllen, LEW21, LittleBenW, Logixoul, Lpetrazickis, Mabdul, Mail6543210, Maxim Masiutin, Michal Jurosz, Mindmatrix, Minghong, Mms, Msulyaev, Nekohan, Nickshanks, ONjA, Odo1982, Paul1337, Progers1618, Qutezuce, Reisio, Sanao, Schapel, Shishkander, Sk4nk, Skedaddle, Skybon, Thorenn, Tobych, Toussaint, Tungpham42, Twpol, UU, Widefox, WulfTheSaxon, Zlogic, パンダ, 171 anonymous edits

Selenium (software) *Source*: http://en.wikipedia.org/w/index.php?oldid=608899137 *Contributors*: Alokgrazitti, Andreas Kaufmann, Apetrovskiy, Arnold xml, Baijum81, BenFrantzDale, Bharathkumar ng, Brian Kendig, C. A. Russell, Capitalismojo, Capricorn42, Chibimagic, Chkno, Cmcmahon, DanielPharos, Darshana.jayasinghe, Dennis Bratland, Devourer09, Diego Moya, Eraserhead1, Ettrig, Excirial, Faizan, Falcor84, Fraggle81, Frankcohen, Frecklefoot, G0gogcsc300, G5reynol, Geniac, Gilliam, Ginsuloft, Hu12, Ixfd64, Jason12345, Jeremy Reeder, Jerome Kelly, Joachimvda, John Yesberg, Jonik, Khalid hassani, Kissedsmiley, Kskchsagar, Leesplankje, Llywrch, Magioladitis, MarinaMichaels, Martin Hampl, MartinMatuš, Materialscientist, Matnatlak4, Mayur, Michael wowro, Mmmarilyn, Modify, Mysdaao, NickW557, Noid, Pas007, PaulHammant, Pecaperopeli, Pinecar, Psubbarao4u, Rajkumarmadineni, Reatlas, Renesis, Retired username, Rich Farmbrough, Rob Burbidge, Ronz, RossPatterson, Runnerweb, Sambub, Sean sunlis, SirGeek CSP, Sleske, Softtest123, Solarra, Spii, SqueakBox, StuartMoncrieff, SunSw0rd, Tannermyoung, Tattoo360-digital, Tushar.murudkar, Twonex, Tyrol5, Ugog Nizdast, Walter Görlitz, Weirdy, Widr, Wikfr, WikiSlasher, Wuzur, Xompanthy, Yintan, Zven, 266 anonymous edits

Build automation *Source*: http://en.wikipedia.org/w/index.php?oldid=603747231 *Contributors*: Age Happens, Alexamies, Alfredo ougaowen, Allens, Andreas Toth, Andy Dingley, Angusgr, Antonielly, Belma06, Berland, Cander0000, CharlesC, Coffee and TV, Conskeptical, Cybercobra, DC, Daniel.Cardenas, David Gerard, Dheeman, Dsavalia, Eddiehu, Ejrrjs, Electrobins, Erechtheus, Frau K, Geofflane, Getsw, GopiTaylor, Isilanes, Jamelan, JnRouvignac, John Vandenberg, JordoCo, Josh Parris, Joy, Kevin.lee, Kvng, Mac, Mark Foskey, Mathias-S, Mistercupcake, Mpeylo, MrOllie, Mreftel, Nullinfinity667, Olilo, Olinga, On5deu, RoestVrijStaal, RuudAlthuizen, Rwwww, Salamurai, ScastlePM, SchreiberBike, SimonTrew, South Bay, Stemonitis, Stevage, Tedickey, X7q, Yurez, 75 anonymous edits

Revision control *Source*: http://en.wikipedia.org/w/index.php?oldid=608729493 *Contributors*: .digamma, 1-is-blue, 16@r, 16x9, A.M., Acdx, Aitias, Aldie, Alvatov, Ancheta Wis, Andy Dingley, Antonielly, ArneBab, Arpit88dawda, Astronautics, Bartosz, Baudway, Berny, Bevo, Bgpaulus, Bjmiller, Blackeagle, BobBagwill, Boobtimelive, Breakpoint, Brockert, Btwied, Burris, Bursar42, Burschik, Cander0000, CanisRufus, Centrx, Certes, CesarB, CharlesC, CitizenDAK, Cjcollier, Clkao, Coasterlover1994, Collabi, Cquan, Crossmr, Cyde, CyrilB, DARTH SIDIOUS 2, DRAGON Elemental, Danielx, David Crawshaw, Deagle AP, Degenera, Deuchlincoln35, Dgw, Dionye, Dorftrottel, Echion2, Edward, Edward Z. Yang, Eep², Egfrank, Ego White Tray, Eloquence, Elwikipedista, Erik9, EvanCarroll, Ewlyahoocom, Felixdakat, Flammifer, Forage, Forderud, Fractal3, Frau K, Frecklefoot, Fryed-peach, Gamefreak2413, Greenrd, Grimboy, Gsmgm, Gurch, Harmil, Hellboy1975, Heron, Hhhjjjkkk, Hydrargyrum, IanClatworthy, ImperfectlyInformed, Imroy, Iridescent, Iulianu, IvanLanin, Ixfd64, J-Wiki, Jan1nad, Janakan86, Jarble, Jblack, Jni, Jonabbey, Jonkerz, Jose.mira, JoshRyan, Joshbosh, Kate, KennethJ, KentCL, Kevins, Khym Chanur, Knarrff, Koavf, Krischik, Leandrogfcdutra, LeeHunter, Lichen0426, Lotje, Lquilter, Luna Santin, Madduck, Mark viking, MarkBrooks, Martinkunev, Matt.forestpath, Memogron, MicahElliott, Michael Allan, Mild Bill Hiccup, Mimosinnet, Minghong, Mitch Ames, Moa3333, Molibdeno, Moxfyre, Msikma, Mu Mind, MyNameIsJackson, Naradapuri, Nashev, Naught101, NawlinWiki, Nbarth, Netsnipe, Nickg, Nigelj, Nikai, O.Koslowski, ObfuscatePenguin, Ohnoitsjamie, Olivier Debre, Pascal666, PatrikFuhrmann, Peruvianllama, PeteVerdon, Peter.vk, Petra.hegarty, Phil Boswell, PhilipMW, Phoe6, PiRSquared17, Piet Delport, Pkchan, Pne, Pretend2bsmart, Project2501a, Qaswebmaster, Qxz, R'n'B, Rachana09, Rashack, RedWolf, Rich Farmbrough, Righteye, Rjd0060, Rjwilmsi, Rob Hooft, Robert Illes, Robert Merkel, Robth, Rodii, RossPatterson, Ruud Koot, Sabri76, Salgueiro, Sam Korn, Seitz, Shenme, SickTwist, Sietse Snel, Sjgcit, Skoczko, Skyezx, Slakr, Snarpel, Soumyasch, Spalding, SpeedyGonsales, Stevietheman, Sun Creator, Sunny256, Synthetik, Talgalili, TechnoGuyRob, Technobadger, Tedickey, Tejas81, Teratornis, Tevildo, The Aviv, The very model of a minor general, Thejoshwolfe, Thiagomacieira, ThomasOwens, Three-quarter-ten, Thv, Timkccb, Timneu22, Tobias Bergemann, Tom Morris, Tommyjb, TruthinQuest, Tue18nov, Tusjain, Tyler McHenry, Tyler Oderkirk, UU, Uršul, Vanished user g454XxNpUVWvxzlr, Vcwizard, Wavelength, Wesley, Weyrick, Wikipelli, Winterst, Wmahan, X7q, YUL89YYZ, Yage, Yaronf, Yevrah342, Youandme, Zachcloud, Zhxjojo, Ziounclesi, 371 anonymous edits

Continuous integration *Source*: http://en.wikipedia.org/w/index.php?oldid=607851664 *Contributors*: Abdull, Acharnock, Adam messinger, Adamleggett, Aidinnz, AlastairIrvine, AlexMorozov, AliveFreeHappy, AllenDowney, Alxgrepe, Andrea105, Andy Dingley, AndyGavin, Anthony Gaudino, Aponar Kestrel, Aron.gombas, Art Carlson, Athought, Badon, Beland, Bertport, Bryan.dollery, Bryankennedy, BuSchu, Cander0000, CanisRufus, Chandresa, Chris Howard, ChrisGualtieri, Chymb, CiaranG, Codename Lisa, Collinjc, CouchSurfer222, Cybercobra, DBooth, Daltenty, Damiens.rf, Danarmak, Daniel.Cardenas, Dariusz wozniak, DarkseidX, Deccico, Denis.dallaire, Dewritech, Drm mills, Duplicity, Dwi Secundus, Edward Reponte, Elizaveta Revyakina, ElodieAndco, ErkinBatu, Erncrts, Etech144, Etoombs, Ettrig, Fabrictramp, Ffangs, FrankTobia, Ganesan Janarthanam, Gary, Gaudol, Ghettoblaster, Gilliam, Gioto, Grover cleveland, Haakon, Hanno Wendt, Henk Langeveld, Hilgerdenaar, Honeplus, Hu12, Hutcher, Huygensvector, Icairns, Imeshev, Irishguy, JBsupreme, JHunterJ, Jacek Kendysz, Jack Merridew, Jacob Robertson, Jamesx12345, Jarble, Jbellwiki, Jchyip, Jeffhudsonbush, Jerryobject, Jgrahamc, Jjdawson7, Jm34harvey, Joeblakesley, JohnMcDonnell, Jonik, Josh Parris, Jpo, Julesd, Jwarnier, Jyankus, Kbrd, Kem254, Khalid hassani, Kingart2, Kku, Kslotte, Leandrod, Lewis, Lgfcd, Liahocho, M4gnum0n, Mahbubur-r-aaman, MandyOwens, Markbassett, Maszanchi, Matsumuraseito, Mcsee, Melimic, Meng6, Mhinterseher, Michaelmalak, Michig, Mmeijeri, Mmenal, Modbear, Morlach01, Mrhericus, Nallen20, Nathan.f77, Nimowy, Nlfiedler, OBeLx, Omnipaedista, Open-collar, PTSE, Pbb, Pedro.m.gama, Pmcollins, Proactivity, Rbalacha, Renato Primavera, Rich Farmbrough, RickBeton, Rickyp, RobCheng, Robinshine, Ronjouch, RossPatterson, Rursus, Régis Décamps, SRyll, Scottmackay, Seajay, SensuiShinobu1234, SethosInOffice, Smountcastle, Softzen, Soltysh, Staceyeschneider, Stemcd, Stephanakib, Stevage, Steveluo, Steven Walling, TJRC, Technobadger, Tedickey, The Wild Falcon, Themshow, Thumperward, TotoBaggins, Tracyragan, Tracyragan10, Uniyal680, Vidiecan, Vidya Raghavan, Vishnu.datla, Vwoowv, WRK, Wadewilliams, Wafulz, Walter Görlitz, Waveclaw, Whywhenwhohow, Wickity, Widefox, William Avery, Winterst, Xfeldman, Yaegor, Ymalik, Yschimke, Zazpot, Zhaofeng Li, Zorgon7, Ï¿½, 332 anonymous edits

Software metric *Source*: http://en.wikipedia.org/w/index.php?oldid=564653834 *Contributors*: Addshore, Adri Timp, Aivosto, Alan McBeth, AliveFreeHappy, Andreas Kaufmann, Andrewpmk, Andypandy.UK, Atul1612, BillGosset, Blaisorblade, BoD, Bobatwiki, Calor, ChristianEdwardGruber, Colin Marquardt, Colonies Chris, Conan, Crawdad1960, Csanjay, Cybercobra, DSParillo, DamienPo, DavidCary, Dekart, Docdrum, Edward, Ehajiyev, Esap, Evil saltine, Franklin90210, Freddy.mallet, Fredrik, Freejason, Gbolton, Gecko06, GenezypKapen, Glapu, Gronky, GuySh, Hu12, Int19h, Intray, Ipsign, IvanLanin, Jamelan, Je alexander, Jeff.foster, Jgrahn, Kdakin, Khalid hassani, Lance.black, Levin, Ligulem, Lucian.voinea, Madir, Martinig, MaxHund, Mbvlist, Mdd, Mheusser, Michael Hardy, Microtony, MrOllie, Nczempin, NewSkool, Nicolapedia, Nposs, Nzeemin, OMouse, Oege, Okivekas, On5deu, Pezra, Phil Boswell, Pm master, Pnm, Pomoxis, Project2501a, Ptrb, Rahulchic, Rozek19, Rp, Saif always, Sashakir, Shinigami7531, Shnout, Slayman, Softwrite, Sozin, SpikeTorontoRCP, Stewartadcock, Swtechwr, Taibah U, Tannin, Tedickey, That Guy, From That Show!, Thenickdude, Van der Hoorn, Vaucouleur, Wafulz, Walter Görlitz, Weregerbil, Wulvengar, Yamla, 165 anonymous edits

Code coverage *Source*: http://en.wikipedia.org/w/index.php?oldid=604912810 *Contributors*: 194.237.150.xxx, A5b, Abdull, Abednigo, Abhinavvaid, Ad88110, Agasta, Aislingdonnelly, Aitias, Aivosto, AliveFreeHappy, Alksub, Allen Moore, Alonergan76, Altenmann, Andreas Kaufmann, Andresmlinar, Andy Dingley, Anorthup, Argonesce, Attilios, Auteurs, Beetstra, BenFrantzDale, Billie usagi, Billinghurst, Bingbangbong, BlackMamba, Blacklily, Blaxthos, Centic, Chester Markel, Cindamuse, Conversion script, Coombes358, Coveragemeter, DagErlingSmørgrav, Damian Yerrick, Derek farn, Didgeedoo, Digantorama, Dr ecksk, Dwheeler, Ebelular, Erkan Yilmaz, Ethically Yours, Faulknerck2, FredCassidy, Gaudol, Ghettoblaster, Gibber blot, Greensburger, HaeB, Henri662, Hertzsprung, Hob Gadling, Hooperbloob, Hqb, Hunghuuhoang, Ianb1469, Infofred, Ixat totep, JASpencer, JGMalcolm, JJMax, Jamelan, JavaTenor, Jdpipe, Jerryobject, Jkeen, Johannes Simon, JorisvS, Jtheires, Julias.shaw, JustAnotherJoe, Kdakin, Ken Gallager, Kku, Kurykh, LDRA, LouScheffer, M4gnum0n, MER-C, Materialscientist, Mati22081979, Matt Crypto, MehrdadAfshari, Mhaghighat, Millerlyte87, Miracleworker5263, Mittgaurav, Mj1000, MrOllie, MywikiaccountSA, Nat hillary, NawlinWiki, NickHodges, Nigelj, Nin1975, Nintendude64, Nixeagle, Ntalamai, Parasoft-pl, Penumbra2000, Phatom87, Picapica, Pinecar, Pratyya Ghosh, Prosfilaes, Ptrb, QARon, Quamrana, Quinntaylor, Quux, RedWolf, Roadbiker53, Rob amos, Robert Merkel, Rpapo, RuggeroB, Rwwww, Scubamunki, Sdesalas, Sebastian.Dietrich, Sferik, SimonKagstrom, Smharr4, Snow78124, Snoyes, Stoilkov, Suruena, Taibah U, Technoparkcorp, Test-tools, Testcocoon, Thargor Orlando, Thumperward, Tiagofassoni, TutterMouse, U2perkunas, Vasywriter, Veralift, Walter Görlitz, Walterkelly-dms, WimdeValk, Witten rules, Wlievens, Wmwmurray, X746e, 268 anonymous edits

Technical debt *Source*: http://en.wikipedia.org/w/index.php?oldid=606769868 *Contributors*: ASZ 87, Abcdefghi76543, Adavidb, Ahunt, AliveFreeHappy, Andreas Kaufmann, Apparition11, Atomice, Chealer, Chris the speller, ChrisGualtieri, D.dunlop, Doctorhawkes, Dr. Zed, Edward, Frecklefoot, Garionh, Goflow6206, Gregpass, HelloAnnyong, Jakuzem, Jameboy, JamesQMurphy, JnRouvignac, Jtison, Karaletourneau, Kareeser, Kostmo, Mandarax, Melizg, MrOllie, NSCoder, Oh Snap, Pauli133, Paulish, PhilipR, Phord, PoqVaUSA, Ptrb, Ringbang, Stefanvenken, Steven Walling, TooTechy, Vchelaru, Vocaro, Vralinitis, WigginsD, WillWare, ZeroOne, 44 anonymous edits

Jenkins (software) *Source*: http://en.wikipedia.org/w/index.php?oldid=608672395 *Contributors*: ABehrens, Anders Feder, Beltranrubo, Brossow, Corti, DVdm, David Gerard, Dprevite, Drm mills, Faqc, Floating Boat, Forderud, Frap, FrenchIsAwesome, Friederbluemle, Gang65, GauravJShah, Gioto, GoingBatty, GregorB, Henrik, Hermzz, HrEkstedt, Ikks, Jarry1250, Jeff Song, Jminne, Jwetherb, KennethJ, Krinkle, LMB, Lone boatman, Macrakis, Michael miceli, Mobimation, Mogism, Nczempin, Nvaracalli, Ohnoitsjamie, Oubiwann, Pombredanne, QuimGil, Rdeknijf, Rmburkhead, Rmrwiki, Ronabop, Scolebourne, Sjh, Solarix, Squawk80, Talion86, The Wild Falcon, ThurnerRupert, TomeWyrm, Tweisbach, Urkle0, Wickorama, Winterheart, Winterst, Yutsi, Zdouglas, 97 anonymous edits

Apache Ant *Source*: http://en.wikipedia.org/w/index.php?oldid=607766863 *Contributors*: Abelson, Akimabu, Alainr345, AlanUS, Alexandre.tp, Andkore, Andy Dingley, Anna Frodesiak, AutumnSnow, Ayush3292, Beetstra, Benefros, Bhasinnitish, Bonadea, Boshomi, C.Oezbek, CaribDigita, Chowbok, Ckielstra, Cmccormick8, Cpflames, David Gerard, Deineka, Discospinster, Dmccreary, Doopokko, Doug Bell, Edward, Elwikipedista, Emre D., Eritikass, Evildeathmath, Faisal.akeel, Ffangs, Firetechguru, Fragglet, Frakturfreund, Fredrik Östman, Gauleng, Golbez, Grs1969, HHLee, Haakon, Hanavy, Hao2lian, Hede2000, Hervegirod, Holon67, Ideoplex, Ilyathemuromets, Isilanes, JLaTondre, Jay, Jgrahn, Jim1138, Jm34harvey, Joeblakesley, JoelDick, John

of Reading, Jonik, Josephw, Julguinet, Juliancolton, Kbdank71, KellyCoinGuy, Kesla, Kevin.lee, Kiranmova, Knuckles, Laudak, Lkt1126, M4design, M4gnum0n, Mahanga, Mamling, Marudubshinki, Mason Simon, Materialscientist, Matt Crypto, Matthewdingley, Matthewfarwell, Melonkelon, MrOllie, Mzajac, NOnash61, Nealmcb, Nixdorf, Od Mishehu, On5deu, Oolong, Opptic, Opticyclic, Peak, Peterl, R27182818, Rodamaker, Rodrigob, S3000, SF007, Sam jervis, Secretlondon, Securiger, Senpai71, SimonP, Skybluecos, Softwaresavant, SolAce7x, Stevage, SteveLoughran, Strebe, Sun Creator, TJRC, Tdmalone, Telso, The Hokkaido Crow, Thumperward, Tony Sidaway, Trygvis, Tumbarumba, Ulrich.b, Ute in DC, Virgiltrasca, Wangi, Weblum, Wickorama, William Avery, Wlievens, Wourriezer, Xduseko, Yozh, Zhenqinli, Zigger, Zootm, 168 anonymous edits

Apache Maven *Source*: http://en.wikipedia.org/w/index.php?oldid=608409311 *Contributors*: Aitter, Alainr345, Alex mayorga, Alexamies, Andkore, ArikTheRed, Aschauer, Ashlux, B3t, Bindulbhowmik, Bkonrad, Bobsmack, Bogdan.iosif, Bonadea, Boshomi, Carbidfischer, Cedar101, Chris the speller, Codebrain, Confluente, Cybercobra, Cyberroadie, Darshana.jayasinghe, Demonkoryu, Doug Bell, Dougher, EasyTarget, Ed Hamburger, ErikFranzK42, Esfand30, Evolve75, Exaton, Faisal.akeel, Forage, Frap, Ftiercel, Groovecoder, Grs1969, HandyAndE, Harm.frielink, Haruth, Heldeen, Hervegirod, Hkong, Ianare, Iweinf, JJuran, JLaTondre, Jandalhandler, Jfrog.media, Jimg, Josephw, Jottinger, Juhko, Jvoegele, Jérôme, Kbdank71, Kesla, KnightRider, Lenin1991, Leonard Davidson, M4gnum0n, Matonen, Medovina, MickScott, Minuteminder, Miremare, Mortense, NOnash61, Necromantiarian, Niteowlneils, Number29, Ohnoitsjamie, On5deu, Opticyclic, Paulusbenedictus, Peterl, Prap19, Programr, Quoth, Raysonho, Robert Illes, RobertDGloverJr, Rocketrod1960, Rootmoose, Rubyuser, Scribionics, Shukka, Silroquen, SimonP, Softwaresavant, SteveLoughran, Strebe, SunKing2, Sweikart, Tentinator, Thumperward, ThurnerRupert, Tsherratt, UKoch, Ultrahob, Uztnus, Vonfraginoff, Weregerbil, West81, Wickorama, WikHead, Willeeuwis, Winterst, Wålberg, Yadavjpr, Yozh, Zhenqinli, Zootm, Zzuuzz, 268 anonymous edits

Convention over configuration *Source*: http://en.wikipedia.org/w/index.php?oldid=606521795 *Contributors*: Ahsanity, AlanUS, Ananthakanaga, Andreas Kaufmann, Arjant, Atzbert, Beland, Belovedfreak, Bomazi, CarlosMunozRodriguez, ChrisGualtieri, ChrisMiddleton, Cybercobra, DNewhall, Dariush.hasanpoor, Diego Moya, DmitriBichko, Doggum, Donkiely, Ethaniel, Gary, GeekyGoon, Goncalo.borrega, Holon67, Hrobeers, JForget, JLaTondre, Jamelan, JaroslavTulach, JohnDoe0007, Josephgrossberg, KevinTeague, Mikermcneil, Mivsek, Mnb20, MrOllie, Msurguy, Mwasheim, New old adam, Ortzinator, Pwroberts, Qleem, R'n'B, Ripounet, Schoetty, Sda005, Sebastian.Dietrich, ShelfSkewed, Shtriter, Sikon, Stefanbente, Thumperward, Tigrisek, Uzume, Vikas.sasidharan, Wanderingstan, Wifelette, WikiLaurent, Yeng-Wang-Yeh, 用心阁, 66 anonymous edits

SonarQube *Source*: http://en.wikipedia.org/w/index.php?oldid=607741206 *Contributors*: Ashik, Blacksnow666, ChrisGualtieri, D3acon, David.racodon, Ganncamp, Gilliam, Henri662, JnRouvignac, Lekthor, Mandrikov, Mercer66, Nirav124u, Nux, Olo55w, Pmerson, Ppapapetrou, Spekulatius2410, Squids and Chips, SzpakEng, Thesquaregroot, ThurnerRupert, Tnyholm, 26 anonymous edits

Minification (programming) *Source*: http://en.wikipedia.org/w/index.php?oldid=605429994 *Contributors*: A Quest For Knowledge, Abelonreg, Alejandro.myc, Alexius08, Andreas Kaufmann, Andrewsuth, Austincheney, Bcosca, Blazar, Brambleclawx, Buweichiu, Chappy84, Chenluois, Chris the speller, Clamum, Corpx, Czaries, DigitalOverload, Donkiely, Dreftymac, Ekerazha, Finlay McWalter, Frap, Globex, Guido71, Harmonic.cipher, Hazardousgaming, Isaac Rabinovitch, Jaehyunkim, Jeepday, Kellystephens301, Kirillosipov, Lastbuncher, Liangent, MC10, Marcus256, Markusbradley, Martijn faassen, Masonaxcte, Mindmatrix, Motine, Mperdeck, NSK Nikolaos S. Karastathis, Nneonneo, Ohconfucius, Pictonpanther, Rfl, Rwalker, Tanthalas39, Tedickey, Thumperward, Timdream, Tobias Bergemann, Welsh, Wonderfl, Wonko, XP1, 83 anonymous edits

Google Closure Tools *Source*: http://en.wikipedia.org/w/index.php?oldid=603666460 *Contributors*: AliveFreeHappy, Danim, Dsbonev, Ginsuloft, Jfmantis, John of Reading, Johnson Cheung, MartinThoma, Mortense, Nsteinberg, Pramod pramd7, Thomas Linder Puls, Thorwald, Trichuris trichiura, Xenotaur, 21 anonymous edits

JSHint *Source*: http://en.wikipedia.org/w/index.php?oldid=603711742 *Contributors*: Antonkovalyov, Ber, FallingGravity, Gotofritz, HelloAnnyong, Ironholds, Jeraphine Gryphon, Krinkle, Northamerica1000, Phette23, Rezonansowy, Scottywong, UltimateSupreme, 5 anonymous edits

Virtual team *Source*: http://en.wikipedia.org/w/index.php?oldid=607900145 *Contributors*: AManWithNoPlan, AbsolutDan, AcademicBusinessResearch, Ahudson6456, Aleebrahim.Nader, Anupbhandari, Auntof6, Bemiller422, Benjaminibe1, Bgwhite, Bill in Alexandria, Bmassey, Bonadea, Chowbok, Christopherhoon, ChuckBiggs2, CierraKristen, Cmarti75, Compo, Csouthers, Dancter, Deaccura39, Deed89, DeknMike, DiceWiki, Discospinster, Djarv4, Download, Drgaines, Drpickem, Edward, Epicgenius, Eustress, Excarmel, Excirial, Filing Flunky, Fland155, G1076, GDallimore, GHe, Ghertel, Golgofrinchian, Gonzo fan2007, Guroadrunner, Hongjone, Jcravens42, Jessicalipnack, Jlipnack, Jojalozzo, Karciakol, Kelly Martin, Khalid hassani, Klind-kbaldwin, Krtek3, Kuru, Lizpresson, Lricci, Mack2, Martijnvanrooij, McGeddon, Mhjackson, MikeCapone, Mkburton, MrOllie, Mrm7171, Mydogategodshat, Mykdavies, N.Aleebrahim, NJITalumni, Nvolve, O.Koslowski, Penbat, RJBurkhart, Rdavout, Revinfotech10, Rjwilmsi, Ronz, RossMJS, Sadatkhanoom, Sajjadpedia, Sbugs, Serpent's Choice, Sherry Happel, Shirulashem, Simonwilloughby, Spitfire, Steepsy, TheJJJunk, Thevirtualteam, Tnelmes-Myoung, Tnxman307, Torped, VCSonline, Vesala, Virtualteambuilders, Wintermutee, Wlodzimierz, 145 anonymous edits

Distributed development *Source*: http://en.wikipedia.org/w/index.php?oldid=551061224 *Contributors*: Eustress, Folajimi, GoingBatty, IjonTichyIjonTichy, Jojalozzo, L Kensington, MrChupon, 4 anonymous edits

Redmine *Source*: http://en.wikipedia.org/w/index.php?oldid=607567173 *Contributors*: 16x9, AllenJB, Astronaut, Barisiriz, Calabe1992, Cbuckley, Charybdisz, Correctivist, Cpc464, DBigXray, Davidjcmorris, Dedmonds, Denysonique, Diza, Editorfun, Enoushka, Frap, Giraffedata, GregorB, Haakon, Hanavy, Hatter87, Hugo-cs, Ihcoyc, JedMeister, Johan D'Hondt, John Vandenberg, Karl Brodowsky, KindDragon, Kkubova, Korath, Mdkoch84, Michaelmcandrew, Michigangold, Mischa The Evil, Mohit28sharma, Mortense, MrOllie, Noq, Oberiko, Oneiros, Openproject-foundation, Paulhey, Pdurbin, Pelerin2, Pharm, Pinethicket, Planiogmbh, Recena, Remember the dot, Rich Farmbrough, Rkumar81, Sabri76, Sagarwal1981, Sbeyer, Speck-Made, Tct13, TextMech, TheCuriousGnome, TheFlowerKing, Walter Görlitz, Welsh, Yaron K., Yk4ever, Zoombody, Zvukupavshixvek, 63 anonymous edits

Bug tracking system *Source*: http://en.wikipedia.org/w/index.php?oldid=605110864 *Contributors*: AlannY, AliveFreeHappy, Alpha0, Alternamke, Andreas Kaufmann, AngeloCoppola, Anoopkp06, Beetstra, Bet Bass, Bgibbs2, BigNate37, Bonadea, Bruce89, Bugaware, C xong, CUTKD, Caerwine, Carlkoppel, Chilblane, Clappingsimon, Cometstyles, Ctrager, Cwolfsheep, Dawnseeker2000, Discospinster, Drhipp, DuLithgow, Edward, Eelvex, Elendal, Ellissound, Epim, Erkan Yilmaz, Ffangs, Filadifei, Fleminra, Fubar Obfusco, Gmarinp, Goa103, Greenrd, Greg Tyler, Hooperbloob, Hooverbag, Hornsofthebull, IO Device, Informup, Iri.nobody, Jacktruman, Jamesnoe, Jbolden1517, Jedimike, Jesse V., JnRouvignac, Johnuniq, JonHarder, Jorunn, Jpbowen, JulianMummery, Jumper32, JustinH, Karamba10, Karnesky, Krishami, Kuru, Libragopi, Luk, MarkyGoldstein, Martin Blazek, Matt Schwartz, Maximkr, Michigangold, Mortense, MrOllie, Nandak89, Nikdo, Notinasnaid, Nzeemin, O.sharov, Ohnoitsjamie, Ojw, P.Y.Python, Pafcu, Pharaoh of the Wizards, PhilipO, Piano non troppo, Pinecar, Pinethicket, Postdlf, Prabin60, Radagast83, Ratemonth, Redrocket, Roneny1, Rupl, S.K., Sam1064, Sarah, Skittleys, Softy, SvGeloven, Taeshadow, Tedickey, TheParanoidOne, Thumperward, Travis99, Truthbro, U2fanboi, Vrenator, Walter Görlitz, Yuuki Mayuki, Yym, Zumbo, 203 anonymous edits

Gantt chart *Source*: http://en.wikipedia.org/w/index.php?oldid=608686611 *Contributors*: AbigailAbernathy, Achalmeena, Agent007bond, Alan Liefting, Alansohn, Alex-golder, Alexandre Ilha, AliceHeere, Andreviens, AndriuZ, Anna Lincoln, Anon126, Archer3, Arthena, Atropus32, Bandeditor, Beatrice06, Big Bird, BillFlis, Biswalchinmayranjan, Black Butterfly, Bollyjeff, Bonkling, Bruno Unna, Buki ben Yogli, C i d, Can't sleep, clown will eat me, Capricorn42, Cavejohnson2, Cdh1001, Celuici, Chandoo.d, Chartex, Chemical Engineer, Chris857, Chrisirwin, Chuunen Baka, Clust4a, Courcelles, Crazycomputers, CycleTimeChart, DARTH SIDIOUS 2, DBigXray, Dan D. Ric, Darko.veberic, Dead3y3, Den fjättrade ankan, Denisarona, Dhavelin, Don Gis, Download, Drawn Some, Dream out loud, Eclipsed, Economist332, Ehn, Ehrenkater, Ellywa, Elm-39, Emrekenci, Enriketuned, Epharos, Etz Haim, Excirial, Fallschirmjäger, Flewis, FlyingToaster, Fulton123, Gaius Cornelius, Garrybooker, GentlemanGhost, George100, Georgewilliamherbert, Gilliam, Gimmetrow, Ginsuloft, Gogo Dodo, GraemeL, Graham87, Graibeard, Gramlin, Grant beest, Grotte, Gwernol, Haham hanuka, Helfrich c, Hephaestos, Heron, HeronOfAlex, Hon-3s-T, Horstvonludwig, Hu12, Hulagutten, Hydrargyrum, Ijon, Ioanapv, IronGargoyle, Isarra (HG), Itai, J.delanoy, JLK7476, Jackfork, Janus999, Jeff3000, Jeffreywherrmann, Jeltz, JidGom, Jimmy Dimmy, Jmlk17, Johnenew, Jos.jong, Jpbowen, Karnesky, Katieh5584, Kayau, Kenmckinley, Khaled hosny, Kine, Kingboyk, Kmccoy, Koranjem, Kotasik, KrakatoaKatie, Kuru, Kzafer, LGinDC, Lawsonstu, Leesly, Ligulem, Lilmissdaughty, Lizzlizzlizz, LogicalDash, Mac Davis, Macbookair3140, Magioladitis, MagnaMopus, Mahjong33, Malleus Fatuorum, Manfi, Manifredtellew, Manop, Mapperboy, MarkS, Materialscientist, Mato, Maurreen, Mdd, Mecanismo, Mecjmr, Melcombe, Michael Hardy, Milkrut, Mindmapmad, Mkoval, Montblanc2000, Mydogategodshat, Myleslong, Nascar1996, Naudefj, Naughtyphil, Ndyguy, Nevron, Nihil novi, NinjaOnFire58, Nixdorf, Nmanos1986, Octahedron80, OlEnglish, Overfloater, P.deshmukh09, PMDrive1061, PatrickWeaver, Peachris, Pere Serafi, Peteazza, Pgcysneiros, Pinethicket, Piotrus, Pm master, Pol098, PookeyMaster, Possums, Poulami100, Preslethe, Proofreader77, Psykocyber, Quasihuman, Quebec99, Raindy, Reach Out to the Truth, RedWolf, Redfirefly, Rememberme123, Renderpeterson, Requestion, Ronhjones, Ronz, Rrburke, Ryan Postlethwaite, Ryan032, S.K., SHCarter, SNIyer12, ST47, Saffordblack, Salvio giuliano, Sbisolo, Serbianboy, Sergio.ballestrero, Seriema, SimonP, Skybum, Smtchahal, Snigbrook, SpaceFlight89, Spangineer, SpeedyGonsales, Steveswei, Stevoabc, SusanB99, Susurrus, Svetovid, Tanetris, Tanár, TastyPoutine, Tbone55, Teenage wikian, Telecineguy, Thane, The Thing That Should Not Be, Theopolisme, Thorpe, Tobias Bergemann, Tomzi, Train78, Travis99, TrbleClef, Tregoweth, Trialsanderrors, Trivialist, Triwbe, Trustcontroller, U2fanboi, UNV, Ubiquity, Uucp, VMS Mosaic, Va7sdf, Velella, Vipinhari, Wapcaplet, Wavefront3, Whpq, WikHead, Wiki13, Wikipelli, Xftan, Yamamoto Ichiro, Yaris678, Yettie0711, Yintan, Yuripobrasil, Zachlipton, Çalıştay, Σ, ć, 653 anonymous edits

Agile software development *Source*: http://en.wikipedia.org/w/index.php?oldid=609094755 *Contributors*: 1exec1, 4johnny, A Quest For Knowledge, A Train, ABV, AGK, Aamahdys, Aardvark92, Aaronbrick, Abmmw, Abukhader, Adbge, Adblast, Aditya2k, Aeuoah, Agauthier007, Agile blog, Agiledev, Agilista, Aguanno, Ahart7, Alansohn, Alexf, Aliaghatabar, Alienus, AliveFreeHappy, Allan McInnes, Amehrabian, Amit Singh, Ancheta Wis, Ancientphoenixians, AndrasSzilard, Andreas Kaufmann, Andrew-King, Andrewman327, Android79, Anonymous Dissident, Ans, Antoniadjames, Apjordan, Apparition11, Artbrock, Ash, Aternity, Atethnekos, Auderobinson, AutumnSnow, Azuendorf, Badgettrg, BaldPark, BartVB, Bdasgupta@gmail.com, Beetstra, Begoodenough, Beland, BenLinders, Benjamin.booth, Benzirpi, BertBril, Bevo, Bf2k, BillyPreset, Bisswajit, Blaisorblade, Bmcmichael, Boing! said Zebedee, Bosef1, Boson, Bovineone, Brendanoconnor, Brick Thrower, Brighterorange, Brusulaf, Bunchofgrapes, BvwRedux, CL, Capricorn42, Carewolf, CatWikiEdit, Catgut, Cawel, Ceyockey, Charlessin, Chicken Wing, Chmod007, Chonglongchoo, Chris the speller, ChrisCork, ChrisGualtieri, ChristianEdwardGruber, Chunkylover199, Cinderella314, Ciotog, Cliffu, Cochiloco, Cognizance, CommonsDelinker, Compfreak7, Cpm, CrimsonBlue, Cruccone, Crysb, Ctobola, Cybercobra, Czarkoff, D'n, DRAGON BOOSTER, Daen, Damitchellvbcoach, Damon Poole, Dandv, Daniel.Cardenas, Danielklein, Danr7, DarkseidX, Davelane103, David Biddulph, David Newton, David Waters, David.alex.lamb, Davidjcmorris, Davidparker9, Davidwhittle, Dbenson,

Deathphoenix, Deb, Demiane, Demonkoryu, Derekrogerson, Designatevoid, Dethomas, Devon Fyson, Dhruvinatgmail, DiiCinta, Dingsoyr, Dionyziz, Dirk Riehle, Dlazerka, Dogcow, Donblair27, Dordal, DouglasGreen, Download, Dparsonsnz, Dprust, Draicone, Eagerterrier, Ed Poor, Edward, EdwardH, Ejvyas, Elf, Elfoles, Elminstersage, Emvee, Enderandpeter, Ennui93, Epeters1, Eric B. and Rakim, Eric Le Bigot, Ericholmstrom, Erpingham, Ettrig, Euryalus, EvaIntland, Exir Kamalabadi, Explicit, Ezani, Famarsha, Fastily, FatalError, Feigling, Finell, First.thesecond, FlashSheridan, Fongamanda, FrancoGG, Frecklefoot, Fred Bradstadt, Freek Verkerk, Friday, Furrykef, Futuregood, G5reynol, GADFLY46, Gary a mason, Gazpacho, GeorgeBills, Ggiavelli, Gilliam, Glennmcgrath2007, Gogo Dodo, GoingBatty, Graham Jones, Graham leach, Graytay, Gronky, Gruffi, Grutness, Gurubrahma, H10130, HalJor, Halmir, Halsteadk, Happysailor, Harizotoh9, Harry, Harry4000, Hede2000, Heirpixel, HelloAnnyong, Hessamnia, Hmains, Hooperbloob, Hu, Hu12, Hzhbcl, Ianneub, IjonTichyIjonTichy, Ileshko, Ipsign, J.delanoy, JHP, JPalonus, Jack Greenmaven, Jafet, JakobVoss, JamesMoose, JamesShore, Janeve.george, Jarble, Jchyip, Jdpipe, Jean-Frédéric, Jeffrd10, Jeffreydca, Jerzy, Jethro B, Jferman, Jim1138, Jjdawson7, Jmabel, Jmilunsky, Jmlive, Johnolgierd, Johnromeovetis, Jojalozzo, Jon.strickler, Jonadin93, Jonas Blind Höna, Jonathan Drain, Jonkpa, Josh Parris, JoshRyan, Joshb, Joshkuo, Jovial Air, Joy2541, Jpbowen, Jtowler, Julesd, Juliancolton, Junnytony, Jusdafax, Jóna Þórunn, KGasso, Kazvorpal, Kbdank71, Kbh3rd, Kelovy, Kenyon, KerryBuckley, Ketiltrout, KevinBrydon, Kevinevans, Khalid hassani, Khouston1, Klazorik, Klemen Kocjancic, Knutux, Koavf, Kosmocentric, Kswaters, Kundu.anupam, Kuru, Kvng, KylieTastic, Laterality, Latha P Nair, Laughing Man, LazyFan, Lefty, Leonus, Leopedia2, LesLein, Leszek Jańczuk, Liao, Lightmouse, Ligulem, Little green rosetta, Littlesal, Liujiang, LizardWizard, Ljhenshall, Lolawrites, Lorezsky, Lsisson, Lugia2453, Lumberjake, Lumos3, MER-C, Magnuschr, Mahbubur-r-aaman, Mange01, Mangst, Mario777Zelda, Mark Arsten, Markwaldin, Martin Hampl, Martinig, MartinsB, Martinvl, Mathmo, Matiasp, MaxGuernseyIII, MaxSem, Maycrying, Mberteig, Mblumber, Mbroooks, Mcsee, Mdd, Mdubakov, Merenta, Mgualtieri, Michael Hardy, Michael Slone, Michig, Mike Field, Mindmatrix, Mmainguy, Mmpubs, Moe Epsilon, Moliang, Morendil, Mortense, Moulding, Mount Flatten, Mr Stephen, MrOllie, Msh210, Mubashar Shahzad, Mymallandnews, NathanLee, Natl1, NawlinWiki, Nearly Human, Neufusion, Nhajratw, Nima1024, Niteowlneils, Niteshnema, Nixeagle, Nnivi, Obiomap1, Ocaasi, Oicumayberight, Okipage8p, Okivekas, OlEnglish, Ontarioboy, Orangeumbrella, PCock, PJamshidi, PabloStraub, Papeschr, Parth pratim, Pastore Italy, Patrickegan, Patsw, Pauli133, Paulralph, Peashy, Penneys79, Peqdahl66, Petri Krohn, Pgr94, Phanishashank, Phansen, Philtro, Pinar, Pm master, Pmsyyz, Pointillist, Porao, Possum, Postdlf, Pratheepraj, Presidentelect00, Prestonpdx, Prunesqualer, PseudoSudo, Ptrb, Pundit, QNo, Qollin, Qst, RA0808, RG2, RHaworth, Raand, Ramsyam, Raspalchima, Rawler, Rddill, Rebroad, Renffeh, Rentaferret, Rhododendrites, Riana, Rich Farmbrough, Rich257, Richwil, Rickjpelleg, RiyaKeralore, Rnickel, Robert.ohaver, RobertoComoEsta, Rockpocket, Rodney Boyd, Rogermw, Ron Richard, Rothley, RoyHanney, Rrelf, Rubicon, Rursus, Rushbugled13, Rwalker, SQGibbon, SamJohnston, Sampi, Samwilson, Sardanaphalus, Sarefo, ScottWAmbler, Sean Whitton, Sean.hoyland, Seanbreeden, Seaphoto, SebastianBreier, Semperf, Sgroupace, Shepard, Shinmawa, Shirik, Shoessss, Siliconglen, Silvonen, Sir Nicholas de Mimsy-Porpington, Skarebo, Smalljim, Softzen, SolBasic, SouthLake, SpaceFlight89, Specter01010, Spellmaster, Srayhan, Sri2001, Srinicenthala, Srleffler, Stein korsveien, SteveCrook, SteveLoughran, Stevegallery, Stickee, Strategy architecture, Strazhce, Stumps, Sulfis, Susan Akers, Swguy, Swillcox, Swtechwr, TJK4114, Tagishsimon, TakuyaMurata, Tarinth, Tarjei Knapstad, Techdoer, Teckmx5, Teknobo, Tempshill, Tentinator, Tgwizard, The Banner, The Thing That Should Not Be, The wub, TheCoffee, TheJJJunk, Theboymusic, Thejoshwolfe, Themfromspace, Themshow, Thingg, Thinkpod, Thiseye, Tiago simoes, Ticaro, Tide rolls, Timwi, Tisni, Tlazar, Tobias Bergemann, Tomb, Tomcat0815, Tommy2010, Tony Sidaway, Trappist the monk, Tsilb, U2fanboi, UdayanBanerjee, Umofomia, Uncle Milty, UncleDouggie, Urrameu, Vald, Van der Hoorn, Vanhorn, Vanished user kijsdion3i4jf, Vasemwant, Vitalij zad, Vreemt, Vyyjpkmi, WDavis1911, WTF-tech, Walter Görlitz, Widefox, Widr, Wiki3 1415, William Avery, William Pietri, Windowsvista2007, Wiretse, Wjbean, Xyad, Yahya Abdal-Aziz, Yath, YoavShapira, ZAD-Man, Zalgo, Zane McFate, Zenofile, Zigger, 1448 anonymous edits

Scrum (software development) *Source*: http://en.wikipedia.org/w/index.php?oldid=609030603 *Contributors*: 4johnny, AGK, Aardvark92, Abhi3385, AbominableBob, Acontrib, Adi4094, Afternoon, Agauthier007, Agile blog, Agilista, Ajm661023, Alansohn, Alex43223, Ali Osmany, Allecher, Alphachimp, Alphamage, Alshall, Amalthea, Amanieux, Andre's Possee, Andreas Kaufmann, Andrewman327, Andy Dingley, Andywarr, Anilgundogdu, Annasw, AntiMeth, Antialias, Anupam, Aoineko, Apparition11, Arcandam, Archipelago88, Arichnad, Avalon, Avec, Axd, Az944racer, Baa, Bates.matt38, Bdiscoe, Beland, Belindatee, Bernburgerin, Bernd in Japan, BertBril, Beverfar, Bgwhite, Bliong, Blueguybur, Bob Stein - VisiBone, Bombe, Boomshadow, Bownyboy, BradBradleySecond, Brainix, Breadtk, Brevan, BrianWren, Brianski, Brick Thrower, Buddhikaeport, C.deepthi, Can't sleep, clown will eat me, CapitalR, Carlton.northern, Cat Parade, Cayuga, Cdedaj, Cerrol, Charlessin, Cheny com, Chi11ken, Chris.beatty, ChrisSims, Chrisc666, Chrisjj3, Christian75, ChuckOp, Chunkylover199, Clarince63, Clayoquot, Closedmouth, Cmccormick8, Codeengage, CometGuru, CommonsDelinker, ComputerGeezer, ConchangoMaster, Corbett.n.michael, Corprew, Costlysoap, Craig Stuntz, Craigwbrown, Crasshopper, Csshaastry, Cthart, Cumeo89, Cybercobra, D(r)ead End, Dandv, Daniel.Cardenas, Daniel347x, Danielbutcher, Dannydohrmann, Darkwind, DavidShepherdson, Davidjcmorris, Deb, Debhart, Demonkoryu, Dereckson, Derek.munneke, Derekhuether, DevelCuy, Dfg13, Diego Moya, Discospinster, DixonD, Diya batti, Dogaroon, Dopetimes, Doroli, Dougher, Dougluce, Dpcoka, DrBeard, Drunken Pirate, Dspectar, Dude4456dude, Dwellings, E Wing, Ebacky, Echofloripa, Edddie, Edward, Eirik.midttun, Elcidia, Elecnix, Eliyahu S, EngineerScotty, Entropy, Ericclack, Esben Krag Hansen, Ettrig, Evan.leonard, EweC, Exemplar sententia, Faller, FeralDruid, Ficell, Firien, Fongamanda, Frank Ferenz, Fred Bradstadt, Fred Waltzer, FreeRangeFrog, Freedomlinux, Ftiercel, Futurepower(R), GadgetSteve, Garde, Garethcollinson, Gaurav, Gegonut, Gekritzl, Gmathur2k, Gmjohnston, Goodhandyman, Guillom, Guppie, Gwern, Gwernol, Hajatvrc, Hardik260, Heaths, Heron, Hhiroshi, Historyfiend2000, Hondo77, Hooperbloob, Humu, Huygens 25, Hwsd, Hyad, Hydrogen Iodide, I8abug, Iani, Ikhnaton2, Ikluft, Immunize, Iner22, Influent1, Ishi Gustaedr, Itairaz, JaGa, Jack in the box, Jamelan, Janeve.george, January2009, Jared Hunt, Jaysweet, Jbondeson, Jdpipe, Jed meyers, Jeff.Mortimer, Jeff.lasovski, Jeffnailen, Jesse Laanti, Jferman, Jhertel, Jivecat, Jlederer2, Jlgrock, Jmilunsky, Jmlive, Jmundo, Johannes.braunias, John of Reading, Johnchapman4, Jonducrou, Jordo ex, Jorfer, Joshuaprewitt, Jpfoerster, Jprg1966, Jpverdoorn, Julesd, Juraj.Kubelka, Jvstein, Jzylstra, Jérôme, KF9129, Kazvorpal, Kenimaru, Kevin sh, Kirrages, Kku, Kmouly, Koustubh123, Kragelund, Kristjan Wager, Ktrueman, Kvng, Kwertii, LOL, Lakeworks, Lampak, Larryaustin10, Lastorset, Lerkey, LeslieValentine, Linusdalin, Littlesal, LokiClock, Longpat, Loreleine1, Lorezsky, Lotje, Lprichar, Lrzepecki, Luc Legay, Lwoodyiii, Lyra Matin, MLetterle, Madhavan.elango, Magioladitis, Manolo w, Marcoshz, Mark Arsten, Markbassett, Markmansouraustralia, MarmotteNZ, Martinig, Maschreu, Materialscientist, Mathiastck, Mattdm, Matthew0028, Mckaysalisbury, Mdd, Mdubakov, Me and, Mesolimbo, Methossant, Mgalvao, Michael.schmidt.62, Michael98101, Michaelpgoggin, Milan.jaros, Miranda, Mistercupcake, Mjb, MontrealKCD, Moriori, Mortense, Mottzo, MrOllie, Mrh30, Mutant, Mvaraujo, Nadia.ndr, Nakon, Nathan Johnson, NickVeys, Nmehta6, Nuiloa, Obonicus, Ohiostandard, Oicumayberight, Olivierhory, OpenFuture, Oskilian, Ostraaten, PDX Aaron, PRRfan, PanderMusubi, Patricidio, PavelCurtis, Peterhundermark, Peterstev, Petritis, Pgan002, Phanibca, Philip Trueman, Polidari, Psiphiorg, Pueywei, Quasarblue, RJaguar3, Radagast83, Rajesh212e63, Ramesh.perla, Razaulhaq.akif, Reach Out to the Truth, Reconsider the static, Reindeeer, Reo46, Rhundhausen, Rich Farmbrough, Rich257, Ricky stern, Rignac, Riki, Rinkle gorge, Riordanmr, Rklawton, Rlliii, Robert The Rebuilder, Robert.d.macdonald, RobinDaniel, Robjenssen, Rogerborg, Rsrikanth05, Rubyuser, Runtime, Rushbugled13, S M Woodall, SAE1962, SDC, SMcCandlish, Saforcer, Safro Elizabeth, Salon Essahj, Sardanaphalus, Satsel, Scbomber, ScottWAmbler, Scottwilleke, Scrum2001, Scrumedge, Scrumireland, Scrummaster, Scrumone, Sean D Martin, Secarrie, Sesoo222, Shadowcat231, Shinmawa, Shubhi choudhary, Simhedges, Sire TRM, Sirk390, Sky Diva, Skyrail, Sleepingbull, SlubGlub, Smaines, Softzen, Songrit, Soopagroove, Stephanvaningen, Stephenb, Stephiee01, Stevage, Strategy architecture, Sumanta 21, Taz00, Tech vaibhav, Techwizization, Teckmx5, Teh pageboy, Tellyaddict, TextMech, Th4n3r, The Thing That Should Not Be, The Wild Falcon, ThomasOwens, Thorwald, Thumperward, Ticaro, Tiddly Tom, Tijfo098, Tim Chambers, Timbertstc, Timneu22, Timwi, Tiuks, Tjarrett, Tnxman307, Tobias Bergemann, Tobych, Torin the Chosen, Trevorjfitz, TriSimon, Trusilver, Tryevenharder, Tsemii, Tumma72, U235, Ularre, Unkei, Untruth, Vacation9, Van der Hoorn, Vanished user kijsdion3i4jf, Vikram.auradkar, Vipinhari, Vthavo, Wainstead, Walter Görlitz, Wclock, Wdfarmer, Wegra, Wikiloop, William Avery, Willowrising, Winterstein, Wizmo, Wperdue, Xanalaska, Xexeo, Zodiakos, Zombiedeity, Zorakoid, Zycron, 1126 anonymous edits

Iterative and incremental development *Source*: http://en.wikipedia.org/w/index.php?oldid=608606260 *Contributors*: AGK, Aamahdys, Aaronbrick, Adeio, Al guy, Alex.ryazantsev, Alistair Cockburn, Alopez6, Ancheta Wis, Andy Dingley, Antandrus, Axd, Bachrach44, Basawala, Basilero, Blues-harp, Cbuckley, Ceyockey, Cmdrjameson, Cmichael, Conan, Darkwind, David.alex.lamb, Desmay, Diannaa, Domokato, Dutchguilder, Edward, Epbr123, Eric B. and Rakim, Fibula, Flewis, Flyer22, Fongamanda, Frequencydip, GFLewis, George Rodney Maruri Game, GeorgeBills, Gesslein, Glenn4pr, GraemeLeggett, Guroadrunner, Gwickwire, Hobart, Holnet, Jack Greenmaven, Jamelan, Jamesx12345, Jerryobject, JimClark, Jj137, Jonesey95, Jtmoon, Kbdank71, KnightRider, Liquixis, Ludde23, M2Ys4U, MER-C, Mafutrct, Marek69, Mark Renier, Markbassett, Markyleong, MaxHund, Mboverload, Mdd, Merenta, Michig, Mikaey, Morendil, Moritz, MrOllie, MyOwnLittlWorld, Natajack, Nawat, Nigosh, Niteowlneils, Objectivesea, Offenbach, Oicumayberight, PMG, Parth pratim, Paul August, Pearle, Polymerbringer, Psequeirag, Radagast83, Rchandra, RedWolf, Rich Farmbrough, Rjwilmsi, Romaine, Sardanaphalus, Sephiroth BCR, Sergiy Kheylyk, Sgryphon, Shanemcd, Smtchahal, SoftwarePM, Srice13, Srushe, Steven Walling, Stijn Vermeeren, Suruena, TakuyaMurata, Thompson.matthew, Tobias Bergemann, Tslocum, Vejvančický, Vrenator, Westerhoff, Whoda, Zhenqinli, 227 anonymous edits

User story *Source*: http://en.wikipedia.org/w/index.php?oldid=607277086 *Contributors*: Anca, Andreas Kaufmann, Andy Dingley, BradBradleySecond, Brick Thrower, Byr8n, Ccyber5, ChasRMartin, Chris Howard, Chunkylover199, Cybercobra, DRogers, Daserver, Davidjcmorris, De728631, Deadbeef, Diego Moya, Dteran1, Fgenolini, Flemingkr, FlorianBauer79, Fridek, GuySh, Hz.tiang, Invenio, Jareha, Jcardazzi, Jdavidb, Jhertel, Jinxypreston, Josh Petitt, Juanpaco, Julesd, Katieh5584, Khalid hassani, Kopf1988, Kpdyer, LOL, Lunivore, M4bwav, M4gnum0n, MandePier, Manuerwin, McGeddon, Meng6, MrOllie, Mithrandir, Ohnoitsjamie, Olau, PSzalapski, Raghunathan.george, Redaktor, Rennerocha, RichardF, Rixs, Ronz, Rrburke, Rvs 2000, SJP, Sloscal1, Softzen, Subversive.sound, Sureshsoft, TCrossland, Ttg2, TutterMouse, Winterst, Wizmo, Xionbox, Ysangkok, Zyravi, 流星依旧, 134 anonymous edits

Image Sources, Licenses and Contributors

File:Virtualbox logo.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Virtualbox_logo.png *License*: unknown *Contributors*: Codename Lisa, Foroa, Magog the Ogre

File:VirtualBox screenshot.png *Source*: http://en.wikipedia.org/w/index.php?title=File:VirtualBox_screenshot.png *License*: GNU General Public License *Contributors*: Kubuntu 11.04: VirtualBox 4.1.8: Windows 7: This screenshot:

File:VirtualBox logo 64px.png *Source*: http://en.wikipedia.org/w/index.php?title=File:VirtualBox_logo_64px.png *License*: unknown *Contributors*: innotek GmbH

File:Ubuntu1310 LiveCD with VirtualBox on Ubuntu1310.ogv *Source*: http://en.wikipedia.org/w/index.php?title=File:Ubuntu1310_LiveCD_with_VirtualBox_on_Ubuntu1310.ogv *License*: unknown *Contributors*: User:OnionBulb

File:Vagrant.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Vagrant.png *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: User:Fco.plj

File:Vagrantup.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:Vagrantup.jpg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: User:Fco.plj

File:140228puppetrunExampleManuallyInvokedPackageUpdate.png *Source*: http://en.wikipedia.org/w/index.php?title=File:140228puppetrunExampleManuallyInvokedPackageUpdate.png *License*: unknown *Contributors*: Puppet Labs

File:Expanded relationships.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Expanded_relationships.svg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: User:MarcelButucea

File:Blackbox.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Blackbox.svg *License*: Public Domain *Contributors*: Original uploader was Frap at en.wikipedia

File:Web-browser usage on Wikimedia.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Web-browser_usage_on_Wikimedia.svg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: User:Joliv

File:Usage share of web browsers (Source StatCounter).svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Usage_share_of_web_browsers_(Source_StatCounter).svg *License*: Creative Commons Attribution 3.0 *Contributors*: User:Daniel.Cardenas, User:Litehacker, User:arichnad

file:Coding Shots Annual Plan high res-5.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:Coding_Shots_Annual_Plan_high_res-5.jpg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: User:Matthew (WMF)

File:Revision controlled project visualization-2010-24-02.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Revision_controlled_project_visualization-2010-24-02.svg *License*: Creative Commons Attribution-ShareAlike 3.0 Unported *Contributors*: Revision_controlled_project_visualization.svg: *Subversion_project_visualization.svg: Traced by User:Stannered, original by en:User:Sami Kerola derivative work: Moxfyre (talk) derivative work: Echion2 (talk)

File:Jenkins logo with title.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Jenkins_logo_with_title.svg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: Jarry1250

File:Apache-Ant-logo.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Apache-Ant-logo.svg *License*: unknown *Contributors*: The Apache Ant Project Team, designed by Nick King

File:Maven logo.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Maven_logo.svg *License*: Attribution *Contributors*: Apache Software Foundation

File:Maven CoC.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Maven_CoC.svg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: 用心阁

File:Sonarqube-48x200.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Sonarqube-48x200.png *License*: Creative Commons Attribution 3.0 *Contributors*: David.racodon

File:Sonarqube-nemo-dashboard.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Sonarqube-nemo-dashboard.png *License*: Creative Commons Attribution 3.0 *Contributors*: David.racodon

File:Diagram of the focus of virtual team research (Powell, Piccoli and Ives, 2004, p.8).png *Source*: http://en.wikipedia.org/w/index.php?title=File:Diagram_of_the_focus_of_virtual_team_research_(Powell,_Piccoli_and_Ives,_2004,_p.8).png *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: Vesala

Image:Distributed Dev Graphic3.gif *Source*: http://en.wikipedia.org/w/index.php?title=File:Distributed_Dev_Graphic3.gif *License*: Public Domain *Contributors*: Clas

Image:Outsourcing-Offshoring.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Outsourcing-Offshoring.svg *License*: Public Domain *Contributors*: TyIzaeL

Image:Redmine logo.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Redmine_logo.svg *License*: Creative Commons Attribution-Sharealike 2.5 *Contributors*: Martin Herr released the original artwork under CC-BY-SA 2.5

File:GanttChartAnatomy.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:GanttChartAnatomy.svg *License*: Public Domain *Contributors*: GanttChartAnatomy.png: Original uploader was Garrybooker at en.wikipedia Later versions were uploaded by Abdull at en.wikipedia. derivative work: Malyszkz (talk)

Image:pert example gantt chart.gif *Source*: http://en.wikipedia.org/w/index.php?title=File:Pert_example_gantt_chart.gif *License*: GNU Free Documentation License *Contributors*: Dbsheajr, FSII, Jni, Sagsaw, Sfan00 IMG, Vanished 1850, 8 anonymous edits

File:Martin Fowler (2008).jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:Martin_Fowler_(2008).jpg *License*: Creative Commons Attribution 2.0 *Contributors*: Ade Oshineye

Image:Pair programming 1.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:Pair_programming_1.jpg *License*: Creative Commons Attribution 2.0 *Contributors*: Lisamarie Babik

File:SoftwareDevelopmentLifeCycle.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:SoftwareDevelopmentLifeCycle.jpg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J.

File:Scrum process.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Scrum_process.svg *License*: unknown *Contributors*: Bruce1ee, KTo288, Lakeworks, Mdd, Sebastian Wallroth, 2 anonymous edits

File:Daily sprint meeting.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:Daily_sprint_meeting.jpg *License*: Creative Commons Attribution-Sharealike 2.0 *Contributors*: FlickreviewR, Ftiercel

File:Scrum task board.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:Scrum_task_board.jpg *License*: Creative Commons Attribution 2.0 *Contributors*: FlickreviewR, Ftiercel, Mattes, Ww2censor, 1 anonymous edits

File:SampleBurndownChart.png *Source*: http://en.wikipedia.org/w/index.php?title=File:SampleBurndownChart.png *License*: Public Domain *Contributors*: Pablo Straub

Image:Iterative development model V2.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:Iterative_development_model_V2.jpg *License*: Public Domain *Contributors*: w:User:Westerhoff

File:Agile Project Management by Planbox.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Agile_Project_Management_by_Planbox.png *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: User:Fongamanda

Image:Development-iterative.gif *Source*: http://en.wikipedia.org/w/index.php?title=File:Development-iterative.gif *License*: Public Domain *Contributors*: Dutchguilder

File:User Story Map in Action.png *Source*: http://en.wikipedia.org/w/index.php?title=File:User_Story_Map_in_Action.png *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: FlorianBauer79

License