ESE SOFT

03 May 2024 05:06 AM

Published software should be free software. To make it free software, you need to release it under a free software license. We normally use the GNU General Public License (GNU GPL), specifying version 3 or any later version, but occasionally we use other free software licenses. We use only licenses that are compatible with the GNU GPL for GNU software.

Copyleft is a general method for making a program free software and requiring all modified and extended versions of the program to be free software as well.

The **GNU General Public License** (**GNU GPL** or simply **GPL**) is a series of widely used free software licenses, or copyleft, that guarantee end users the four freedoms to run, study, share, and modify the software.[7] The license was the first copyleft for general use, and was originally written by Richard Stallman, the founder of the Free Software Foundation (FSF), for the GNU Project. The license grants the recipients of a computer program the rights of the Free Software Definition.[8] The licenses in the GPL series are all copyleft licenses, which means that any derivative work must be distributed under the same or equivalent license terms. It is more restrictive than the Lesser General Public License, and even further distinct from the more widely-used permissive software licenses BSD, MIT, and Apache.

Historically, the GPL license family has been one of the most popular software licenses in the free and open-source software (FOSS) domain.[7][9][10][11][12] Prominent free software programs licensed under the

Stallman pioneered the concept of copyleft, which uses the principles of copyright law to preserve the right to use, modify, and distribute free software. He is the main author of free software licenses which describe those terms, most notably the GNU General Public License (GPL), the most widely used free software license.[7]

In 1989, he co-founded the League for Programming Freedom. Since the mid-1990s, Stallman has spent most of his time advocating for free software, as well as campaigning against software patents, digital rights management (which he refers to as digital *restrictions* management, calling the more common term misleading), and other legal and technical systems which he sees as taking away users' freedoms. This has included software license agreements, non-disclosure agreements, activation keys, dongles, copy restriction, proprietary formats, and binary executables without source code.

Debian Free Software Guidelines (DFSG)

1. **Free redistribution**
   The license of a Debian component may not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license may not require a royalty or other fee for such sale.
2. **Source code**
   The program must include source code, and must allow distribution in source code as well as compiled form.
3. **Derived works**
   The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
4. **Integrity of the author's source code**
   The license may restrict source-code from being distributed in modified form only if the license allows the distribution of patch files with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software. (This is a compromise. The Debian group encourages all authors not to restrict any files, source or binary, from being modified.)
5. **No discrimination against persons or groups**
   The license must not discriminate against any person or group of persons.
6. **No discrimination against fields of endeavor**
   The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.
7. **Distribution of license**
   The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
8. **License must not be specific to Debian**
   The rights attached to the program must not depend on the program's being part of a Debian system. If the program is extracted from Debian and used or distributed without Debian but otherwise within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the Debian system.
9. **License must not contaminate other software**
   The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be free software.
10. **Example licenses**
    The GNU General Public License, Berkeley Software Distribution, and Artistic licenses are examples of licenses that we consider free.

The concept of copyleft was described in Richard Stallman's *GNU Manifesto* in 1985, where he wrote:

> GNU is not in the public domain. Everyone will be permitted to modify and redistribute GNU, but no distributor will be allowed to restrict its further redistribution. That is to say, proprietary modifications will not be allowed. I want to make sure that all versions of GNU remain free.

Not invented here (**NIH**) is the tendency to avoid using or buying products, research, standards, or knowledge from external origins. It is usually adopted by social, corporate, or institutional cultures. Research illustrates a strong bias against ideas from the outside.[1]

The **Halloween documents** comprise a series of confidential Microsoft memoranda on potential strategies relating to free software, open-source software, and to Linux in particular, and a series of media responses to these memoranda. Both the leaked documents and the responses were published by open-source software advocate Eric S. Raymond in 1998.[1]

The documents are associated with Halloween because many of them were originally leaked close to October 31 in different years.

The first Halloween document, requested by senior vice-president Jim Allchin for the attention of senior vice-president Paul Maritz and written by Microsoft program manager Vinod Valloppillil, was leaked to Eric Raymond in October 1998, who immediately published an annotated version on his web site. The document contained references to a second memorandum specifically dealing with Linux, and that document, authored by Vinod Valloppillil and

An SLA is a documented agreement between a service provider and a customer that defines: (i) the level of service a customer should expect, while laying out the metrics by which service is measured, as well as (ii) remedies or penalties should agreed-upon service levels not be achieved. It is a critical component of any technology vendor contract.

Pesticide Paradox: It states that if the same tests are repeated over and over again, eventually, the same set of test cases will no longer identify any new bugs in the system. In plain English, this means that as you run your tests multiple times, they stop being effective in catching bugs. Moreover, your existing tests will not catch part of the new defects introduced into the system and will be released onto the field.

Almost 20 years ago Boris Beizer stated what became known as the Pesticide Paradox: *"Every method you use to prevent or find bugs leaves a residue of subtler bugs against which those methods are ineffectual."*

Mutation testing, also known as code mutation testing, is a form of white box testing in which testers change specific components of an application's source code to ensure a software test suite can detect the changes. Changes introduced to the software are intended to cause errors in the program. Mutation testing is designed to ensure the quality of a software testing tool, not the applications it analyzes.

Mutation testing is typically used to conduct unit tests. The goal is to ensure a software test can detect code that isn't properly tested or hidden defects that other testing methods don't catch. Changes called mutations can be implemented by modifying an existing line of code. For example, a statement could be deleted or duplicated, true or false expressions can be changed or other variables can be altered. Code with the mutations is then tested and compared to the original code.

Richard Lipton proposed the mutation testing in 1971 for the first time.

Functional Testing:

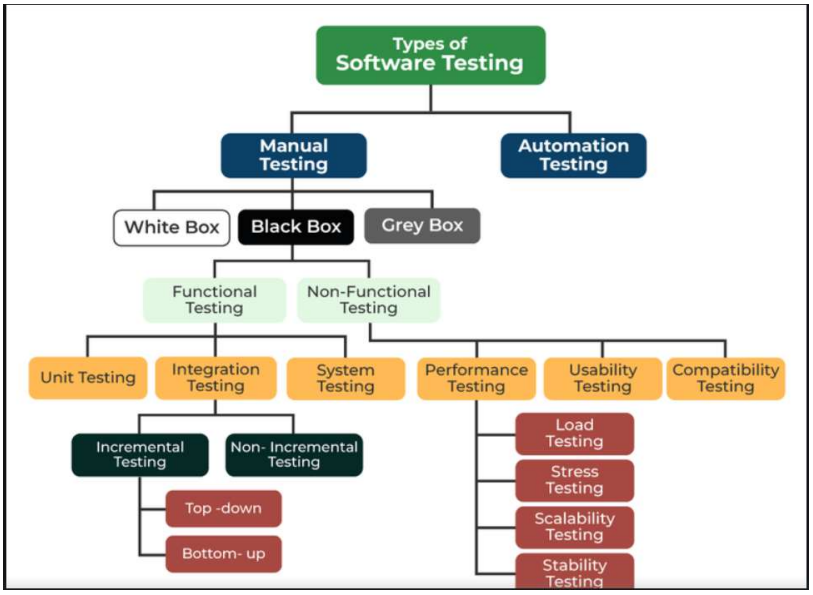
Functional testing is defined as a type of testing that verifies that each function of the software application works in conformance with the requirement and specification. This testing is not concerned with the source code of the application. Each functionality of the software application is tested by providing appropriate test input, expecting the output, and comparing the actual output with the expected output. This testing focuses on checking the user interface, APIs, database, security, client or server application, and functionality of the Application Under Test. Functional testing can be manual or automated. It determines the system's software functional requirements.

Regression Testing:

Regression Testing is the process of testing the modified parts of the code and the parts that might get affected due to the modifications to ensure that no new errors have been introduced in the software after the modifications have been made. Regression means the return of something and in the software field, it refers to the return of a bug. It ensures that the newly added code is compatible with the existing code. In other words, a new software update has no impact on the functionality of the software. This is carried out after a system maintenance operation and upgrades.

Software testing can be divided into two steps:

1. **Verification:** It refers to the set of tasks that ensure that the software correctly implements a specific function. It means "Are we building the product right?".
2. **Validation:** It refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements. It means "Are we building the right product?".



1. Functional testing: It is a type of software testing that validates the software systems against the functional requirements. It is performed to check whether the application is working as per the software's functional requirements or not. Various types of functional testing are Unit testing, Integration testing, System testing, Smoke testing, and so on.
2. Non-functional testing: It is a type of software testing that checks the application for non-functional requirements like performance, scalability, portability, stress, etc. Various types of non-functional testing are Performance testing, Stress testing, Usability Testing, and so on.
3. Maintenance testing**:** It is the process of changing, modifying, and updating the software to keep up with the customer's needs. It involves regression testing that verifies that recent changes to the code have not adversely affected other previously working parts of the software.

. Unit testing: It a level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed.
. Integration testing: It is a level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.
. System testing: It is a level of the software testing process where a complete, integrated system/software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.
. Acceptance testing: It is a level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

Software Development Life Cycle.

White box testing

1. Statement Coverage

In this technique, the aim is to traverse all statements at least once. Hence, each line of code is tested. In the case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, it helps in pointing out faulty code.

2. Branch Coverage:

In this technique, test cases are designed so that each branch from all decision points is traversed at least once. In a flowchart, all edges must be traversed at least once.

Acceptance testing is software testing that evaluates whether a system meets its business and user requirements. It is the final testing stage before a system is released to production.

The main purpose of acceptance testing is to verify that a system:

- Meets all of its functional and non-functional requirements
- Easy to use and navigate
- Reliable and functions as expected
- Secure and comply with all applicable regulations

This testing is performed by end users or an expected group of users. This ensures that the system is tested from the perspective of the people using it daily.

Regression Testing is a type of testing in the software development cycle that runs after every change to ensure that the change introduces no unintended breaks. Regression testing addresses a common issue that developers face — the emergence of old bugs with the introduction of new changes.

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.

Stress Testing is a software testing technique that determines the robustness of software by testing beyond the limits of normal operation. Stress testing is particularly important for critical software but is used for all types of software. Stress testing emphasizes robustness, availability, and error handling under a heavy load rather than what is correct behavior under normal situations.

Purpose of Stress Testing

- **Analyze the behavior of the application after failure:** The purpose of stress testing is to analyze the behavior of the application after failure and the software should display the appropriate error messages while it is under extreme conditions.
- **System recovers after failure:** Stress testing aims to make sure that there are plans for recovering the system to the working state so that the system recovers after failure.
- **Uncover Hardware issues:** Stress testing helps to uncover hardware issues and data corruption issues.
- **Uncover Security Weakness:** Stress testing helps to uncover the security vulnerabilities that may enter into the system during the constant peak load and compromise the system.
- **Ensures data integrity:** Stress testing helps to determine the application's data integrity throughout the extreme load, which means that the data should be in a dependable state even after a failure.

Objectives of Load Testing

1. **Evaluation of Scalability:** Assess the system's ability to handle growing user and transaction demands. Find the point at which the system begins to function badly.
2. **Planning for Capacity:** Describe the system's ability to accommodate anticipated future increases in the number of users, transactions and volume of data. Making well-informed decisions regarding infrastructure upgrades is made easier by this.
3. **Determine bottlenecks:** Identify and localize bottlenecks in the application or infrastructure's performance. Finding the places where the system's performance can suffer under load is part of this.
4. **Analysis of Response Time:** For crucial transactions and user interactions, track and evaluate response times. Make that the system responds to changes in load with reasonable response times.
5. **Finding Memory Leaks**: Find and fix memory leaks that may eventually cause a decline in performance. Make sure the programme doesn't use up too many resources when it's running.

What is Greenfield Software Development?

Greenfield software development refers to developing a system for a totally new environment and requires development from a clean slate – no legacy code around. It is an approach used when you're starting fresh and with no restrictions or dependencies.

A pure Greenfield project is quite rare these days, you frequently end up interacting or updating some amount of existing code or enabling integrations. Some examples of Greenfield software development include: building a website or app from scratch, setting up a new data center, or even implementing a new rules engine.

What is Brownfield Software Development?

Brownfield software development refers to the development and deployment of a new software system in the presence of existing or legacy software systems. Brownfield application development usually happens when you want to develop or improve upon an existing application, and compels you to work with previously created code.

Therefore, any new software architecture must consider and coexist with systems already in place – so as to enhance existing functionality or capability. Examples of Brownfield development include: adding a new module to an existing enterprise system, integrating a new feature to software that was developed earlier, or upgrading code to enhance the functionality of an app.

Netscape Navigator is a discontinued proprietary web browser, and the original browser of the Netscape line, from versions 1 to 4.08, and 9.x. It was the flagship product of the Netscape Communications Corp and was the dominant web browser in terms of usage share in the 1990s, but by around 2003 its user base had all but disappeared.[2] This was partly because the Netscape Corporation (later purchased by AOL) did not sustain Netscape Navigator's technical innovation in the late 1990s.[3]

The business demise of Netscape was a central premise of Microsoft's antitrust trial, wherein the Court ruled that Microsoft's bundling of Internet Explorer with the Windows operating system was a monopolistic and illegal business practice. The decision came too late for Netscape, however, as Internet Explorer had by then become the dominant web browser in Windows.

Enterprise resource planning (ERP) is a software system that helps you run your entire business, supporting automation and processes in finance, human resources, manufacturing, supply chain, services, procurement, and more.

Ken thompson = org unix
Donald Knuth = creating tex
Andrew Tanenbaum releases Minix, a version of UNIX
Larry Wall creates Perl (Practical Extraction and Report Language)
. Eric Allmann, a student at UC Berkely develops a program that routes messages between computers over ARPANET. It later evolves into Sendmail.

The **Advanced Research Projects Agency Network** (ARPANET) was the first wide-area packet-switched network with distributed control and one of the first computer networks to implement the TCP/IP protocol suite. Both...

Creator of internet =
Vinton Cerf and Bob Kahn

Tim Berners-Lee invented the World Wide Web while working at CERN in 1989

Linus Benedict Torvalds (/ˈliːnəs ˈtɔːrvɔːldz/ *LEE-nəs TOR-vawldz*,[a] Finland Swedish: [ˈliːnʉs ˈtuːrvalds] ⓘ; born 28 December 1969) is a Finnish-American software engineer who is the creator and lead developer of the Linux kernel. He also created the distributed version control system Git.

The term Waterfall was first coined by Winston W. Royce in 1970

The first handheld calculator was a 1967 prototype called Cal Tech, whose development was led by Jack Kilby at Texas Instruments

Electronic Numerical Integrator And Computer-eniac-first computer

Cookies were created in 1994 by Lou Montulli, a web browser programmer at Netscape Communications.

The term 'Internet of Things' was coined in 1999 by the computer scientist Kevin Ashton. While working at Procter & Gamble

Definition. The term big data has been in use since the 1990s, with some giving credit to John Mashey for popularizing the term.