Code reuse

Code reuse, also called **software reuse**, is the use of existing software, or software knowledge, to build new software,[1] following the reusability principles.

Contents

Overview

Ad hoc code reuse has been practiced from the earliest days of programming. Programmers have always reused sections of code, templates, functions, and procedures. Software reuse as a recognized area of study in software engineering, however, dates only from 1968 when Douglas McIlroy of Bell Laboratories proposed basing the software industry on reusable components.

Code reuse aims to save time and resources and reduce redundancy by taking advantage of assets that have already been created in some form within the software product development process.[2] The key idea in reuse is that parts of a computer program written at one time can be or should be used in the construction of other programs written at a later time.

Code reuse may imply the creation of a separately maintained version of the reusable assets. While code is the most common resource selected for reuse, other assets generated during the development cycle may offer opportunities for reuse: software components, test suites, designs, documentation, and so on.[3]

The software library is a good example of code reuse. Programmers may decide to create internal abstractions so that certain parts of their program can be reused, or may create custom libraries for their own use. Some characteristics that make software more easily reusable are modularity, loose coupling, high cohesion, information hiding and separation of concerns.

For newly written code to use a piece of existing code, some kind of interface, or means of communication, must be defined. These commonly include a "call" or use of a subroutine, object, class, or prototype. In organizations, such practices are formalized and standardized by domain engineering, also known as software product line engineering.

The general practice of using a prior version of an extant program as a starting point for the next version, is also a form of code reuse.

Some so-called code "reuse" involves simply copying some or all of the code from an existing program into a new one. While organizations can realize time to market benefits for a new product with this approach, they can subsequently be saddled with many of the same code duplication problems caused by cut and paste programming.

Many researchers have worked to make reuse faster, easier, more systematic, and an integral part of the normal process of programming. These are some of the main goals behind the invention of object-oriented programming, which became one of the most common forms of formalized reuse. A somewhat later invention is generic programming.

Another, newer means is to use software "generators", programs which can create new programs of a certain type, based on a set of parameters that users choose. Fields of study about such systems are generative programming and metaprogramming.

Types of reuse

Concerning motivation and driving factors, reuse can be:

- Opportunistic – While getting ready to begin a project, the team realizes that there are existing components that they can reuse.
- Planned – A team strategically designs components so that they'll be reusable in future projects.

Reuse can be categorized further:

- Internal reuse – A team reuses its own components. This may be a business decision, since the team may want to control a component critical to the project.
- External reuse – A team may choose to license a third-party component. Licensing a third-party component typically costs the team 1 to 20 percent of what it would cost to develop internally.[4] The team must also consider the time it takes to find, learn and integrate the component.

Concerning form or structure of reuse, code can be:[5]

- Referenced – The client code contains a reference to reused code, and thus they have distinct life cycles and can have distinct versions.
- Forked – The client code contains a local or private copy of the reused code, and thus they share a single life cycle and a single version.

Fork-reuse is often discouraged because it's a form of code duplication, which requires that every bug is corrected in each copy, and enhancements made to reused code need to be manually merged in every copy or they become out-of-date. However, fork-reuse can have benefits such as isolation, flexibility to change the reused code, easier packaging, deployment and version management.[5]

Systematic

Systematic software reuse is a strategy for increasing productivity and improving quality of the software industry. Although it is simple in concept, successful software reuse implementation is difficult in practice. A reason put forward for this is the dependence of software reuse on the context in which it is implemented. Some problematic issues that needs to be addressed related to systematic software reuse are:[6]

- a clear and well-defined product vision is an essential foundation to an SPL.
- an evolutionary implementation strategy would be a more pragmatic strategy for the company.
- there exist a need for continuous management support and leadership to ensure success.
- an appropriate organisational structure is needed to support SPL engineering.
- the change of mindset from a project-centric company to a product-oriented company is essential.

Examples

Software libraries

A very common example of code reuse is the technique of using a software library. Many common operations, such as converting information among different well-known formats, accessing external storage, interfacing with external programs, or manipulating information (numbers, words, names, locations, dates, etc.) in common ways, are needed by many different programs. Authors of new programs can use the code in a software library to perform these tasks, instead of "re-inventing the wheel", by writing fully new

code directly in a program to perform an operation. Library implementations often have the benefit of being well-tested, and covering unusual or arcane cases. Disadvantages include the inability to tweak details which may affect performance or the desired output, and the time and cost of acquiring, learning, and configuring the library.[7]

Design patterns

A design pattern is a general solution to a recurring problem. Design patterns are more conceptual than tangible and can be modified to fit the exact need. However, abstract classes and interfaces can be reused to implement certain patterns.

Frameworks

Developers generally reuse large pieces of software via third-party applications and frameworks. Though frameworks are usually domain-specific and applicable only to families of applications.

Higher-order function

In functional programming higher-order functions can be used in many cases where design patterns or frameworks were formerly used.

Retrocomputing

Retrocomputing encompasses reuse of code, simply because retro programs are being run on older computers, or emulators for them.

Computer security

In computer security code-reuse is employed as a software exploit method.[8] When an attacker is not able to directly input code to modify the control flow of a program, for example in presence of code injection defenses such as W⊕X, he or she can redirect the control flow to code sequences existing in memory.

Examples of code-reuse attacks are return-to-libc attack, return-oriented programming, and jump-oriented programming.[8][9]

Components

A component, in an object-oriented extent, represents a set of collaborative classes (or only one class) and its interfaces. The interfaces are responsible for enabling the replacement of components. Reusable components can also be isolated and synchronized between SCM repositories using component source code management technologies (CSCM (https://github.com/team bit/bit)).

Outside computers

The whole concept of "code reuse" can also encompass engineering applications outside software. For instance, parametric modeling in computer-aided design allows for creating reusable designs. Standardization results in creation of interoperable parts that can be then reused in many contexts.

Criticism

Code reuse results in dependency on the component being reused. Rob Pike opined that "A little copying is better than a little dependency". When he joined Google, the company was putting heavy emphasis on code reuse. He believes that Google's codebase still suffers from results of that former policy in terms of compilation speed and maintainability.[10]

See also

- Don't repeat yourself

- ICSR
- Inheritance
- Language binding
- Not invented here (antonym)
- Polymorphism
- Procedural programming
- Reinventing the wheel (antonym)
- Reusability
- Reuse metrics
- Single source of truth
- Software framework
- Virtual inheritance

References

1. Frakes, W.B.; Kyo Kang (July 2005). "Software Reuse Research: Status and Future" (https://pdfs.semanticscholar.org/660a/9612cdc5baf70c82fa1db291ded7bef39546.pdf) (PDF). *IEEE Transactions on Software Engineering*. **31** (7): 529–536. doi:10.1109/TSE.2005.85 (https://doi.org/10.1109%2FTSE.2005.85).
2. Lombard Hill Group. "What Is Software Reuse?" (http://lombardhill.com/what_reuse.htm). *lombardhill.com*. Lombard Hill Group. Retrieved 22 October 2014.
3. Lombard Hill Group. "What Is Software Reuse?" (http://lombardhill.com/what_reuse.htm). Retrieved 22 October 2014.
4. McConnell, Steve (1996). *Rapid Development: Taming Wild Software Schedules*. ISBN 978-1-55615-900-8.
5. Colombo, F. (2011). "It's not just reuse" (http://sharednow.blogspot.com/2011/05/its-not-just-reuse.html). *SharedNow.blogspot*.
6. Champman, M.; Van der Merwe, Alta (2008). "Contemplating Systematic Software Reuse in a Small Project-centric Company" (http://portal.acm.org/citation.cfm?id=1456662). *Proceeding SAICSIT '08 Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*. doi:10.1145/1456659.1456662 (https://doi.org/10.1145%2F1456659.1456662). ISBN 978-1-60558-286-3.
7. "Code reuse" (https://web.archive.org/web/20110710143019/http://docforge.com/wiki/Code_reuse). *DocForge*. Archived from the original (http://docforge.com/wiki/Code_reuse) on 2011-07-10. Retrieved 27 January 2019.
8. Bletsch, Tyler (2011). "Code-reuse Attacks: New Frontiers and Defenses" (http://dl.acm.org/citation.cfm?id=2338075). North Carolina State University. ISBN 978-1-124-75297-6.
9. Bletsch, Tyler; Jiang, Xuxian; Freeh, Vince W; Liang, Zhenkai (2011). "Jump-oriented programming: a new class of code-reuse attack" (https://web.comp.nus.edu.sg/~liangzk/papers/asiaccs11.pdf) (PDF). *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*. ACM. pp. 30–40. doi:10.1145/1966913.1966919 (https://doi.org/10.1145%2F1966913.1966919). ISBN 978-1-4503-0564-8.
10. The Go Programming Language (2015-12-01), *Go Proverbs – Rob Pike – Gopherfest – November 18, 2015* (https://www.youtube.com/watch?v=PAAkCSZUG1c), retrieved 26 February 2016

External links

- ReNews – an information site about software reuse and domain engineering (http://wfrakes.wordpress.com)
- Software Reuse Tips Article (http://www.infoq.com/articles/vijay-narayanan-software-reuse)
