What Is Code Reuse? Code Reuse Best Practices

CODING BEST PRACTICES | STATIC ANALYSIS

By Richard Bellairs

Code reuse can solve the software growth problems associated with both business and technical challenges. Here we explain what is code reuse and how to reuse code effectively.

Read along or jump ahead to the section that interests you the most:

How to Ensure Reusability of Code in a Project?

Code reuse is a great goal.

In an ideal environment, a developer would be able to access stable and trusted code libraries. They'd then be able to reuse code from those libraries as building blocks within their application.

So, you can reuse code when it can be:

- Easily extended and adapted for the new application.
- Ported to different hardware if needed.
- Shown to be free from defects or problems that affect the reliability, safety, or security of the new application.

But the environment is not always ideal. And code doesn't always fulfill these requirements. So, reusing code often sounds much easier than it is.

When Is Code Reuse Not Possible?

Code reuse often proves to be difficult.

In practice, developers often end up rebuilding

When Is Code Reuse Not Possible?

Code reuse often proves to be difficult.

In practice, developers often end up rebuilding software from scratch. Or they can only reuse a small fraction of existing code in new projects.

Why Software Reuse Is Difficult

Software reuse is difficult. This is especially true for organizations with a large number of product components and geographically distributed development teams.

Here are three reasons software reuse is difficult.

Organization and Scale

As the number of projects and developers increases, it becomes harder to reuse software. It's a challenge to effectively communicate the details and requirements for code reuse. And it's difficult to provide adequate guidance and feedback on the reuse of code.

Administration

As the number of projects and developers grows, it's difficult to share libraries of reusable

feedback on the reuse of code.

Administration

As the number of projects and developers grows, it's difficult to share libraries of reusable code. It's a challenge to catalog, archive, and retrieve reusable assets on a global basis.

Platforms such as GitHub can make this easier. But it still takes time and effort to establish a usable and scalable code repository.

Politics and Psychology

At an organizational level, office politics can be a barrier to software reuse. As business units strive for autonomy — or compete amongst themselves — they may try to block reuse of their assets by other units.

At the individual level, developers may view code reuse as stifling their creativity — or as the organization lacking confidence in their technical abilities. Such perceptions lead some developers to resist efforts to increase code reuse.

Code Reuse Challenges

There are both operational and technical challenges with reusing code.

Code Reuse Challenges

There are both operational and technical challenges with reusing code.

Code Reuse Operational Challenges

Creating reusable code requires an investment of time and resources during development. Truly reusable code can be reused in new ways that differ substantially from the code's original design intent.

More Time

To support planned reuse, development teams need to spend additional time writing documentation for their code. And they need to test it more thoroughly than code slated for just a single project.

This is a challenge for developers with tight deadlines. Some fail because they are overly ambitious. They invest too much in upfront design efforts. Others fail due to poor planning, lack of design flexibility, or lack of funding.

More Resources

Creating reusable code requires project managers to plan additional resources upfront.

Design Patterns

In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Uses of Design Patterns

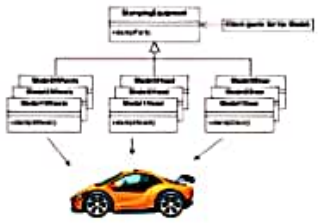
Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.

Often, people only understand how to apply certain software design techniques to certain problems. These techniques are difficult to apply to a broader range of problems. Design patterns provide general solutions, documented in a format that doesn't require

Creational design patterns

These design patterns are all about class instantiation. This pattern can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.

- **Abstract Factory**
  Creates an instance of several families of classes

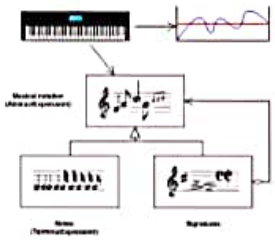


- **Builder**
  Separates object construction from its representation
- **Factory Method**
  Creates an instance of several derived classes
- **Object Pool**
  Avoid expensive acquisition and release of resources by recycling objects that are no longer in use
- **Prototype**
  A fully initialized instance to be copied or cloned

Behavioral design patterns

These design patterns are all about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.

- **Chain of responsibility**
  A way of passing a request between a chain of objects



- **Command**
  Encapsulate a command request as an object
- **Interpreter**
  A way to include language elements in a program
- **Iterator**
  Sequentially access the elements of a collection
- **Mediator**
  Defines simplified communication between classes
- **Memento**
  Capture and restore an object's internal state
- **Null Object**
  Designed to act as a default value of an object