Refactoring در واقع یک فرآیند بازیابی کد هست بدون اینکه رفتار کد تغییر کند. هدف این کار ارتقا ویژگی های nonfunctional نرم افزار هست. این جز عملیات های اصلی ساختن یک محصول هست برای اینکه maintainability را بالا ببریم به این منظور که بتوانیم در فاز های مختلف توسعه نرم افزار محصول را توسعه بدهیم. یک کد خوب قابل فهم هست شناخته شده است. این عمل عملیات را تغییر نمیدهد. باید مرتب تست انجام بدهیم تا بفهمیم کد صحیح هست یا نه. هدف این است که کد ما خوانا باشد، نامگذاری ها در آن واضح و ثابت باشند همچنین کد تکراری در برنامه نداشته باشیم. هیچ مقادیری نباید در سیستم به صورت hard code باشند یعنی نباید به صورت مستقیم به متغیر ها مقادیر ثابت بدهیم و باید قابل configurable یا پویا باشند. یکی از اهداف refactoring، code smell هست که خودش لزوما به معنا مشکل داشتن کد نیست بلکه نشانه یک مشکل عمیق تر هست که باعث میشود maintaibability از دست برود چرا که باعث میشود خوانایی کد سخت تر شود.

مارتین فولر به 3 نکته در کتاب refactoring خودش اشاره میکند

1. یه آدم ضعیف میتواند کدی بنویسد که کامپیوتر میتواند متوجه شود ولی یک برنامه نویس خوب کدی مینویسد که انسان ها میتوانند متوجه شوند. یعنی اینکه باید کد هایی بنویسیم به صورت maintainable باشند و کد ها را بتوان تغییر داد زیرا که برنامه ها توسعه پیدا میکنند و به فیچر هایی جدید نیاز هست و نیاز های جدید ظهور میکند پس برای اینکه مسیر برنامه را بتوانیم تغییر بدهیم کد ما باید maintable باشد. پس باید کدی بنویسیم که انسان بفهمد تا بتوانیم تغییرش بدهیم.
2. 3 بار تلاش کن بعد refactor کن یعنی اینکه اگه کد تکراری ظاهر کرد بار اول کاری نکن فقط کد زدن را ادامه بده و وقتی برای بازیابی نزار برای بار دوم وقتی فهمیدی کد تکراری وجود دارد باز فقط تو ذهنش نگهدار ولی توسعه را ادامه بده اما در دفعه سوم اگه باز هم اگه کد تکراری ظهور کرد دیگه پشت سرش نزار و سعی کن حل بکنی و آسان بشود برای خواندن و تغییر دادن.
3. وقتی میخواهی چیزی را تغییر بدهی اول تغییر رو آسان کن بعد تغییرات رو انجام بده. چون چیزی که الان نیاز به تغییر دارد احتمالا در آینده هم نیاز به تغییر دارد.

الگوی بازآفرینی تابع استخراج در C++ تکنیکی است که برای بهبود خوانایی، قابلیت نگهداری و قابلیت استفاده مجدد کد از طریق تجزیه توابع بزرگ یا پیچیده به توابع کوچکتر و قابل مدیریت تر استفاده می شود. این شامل شناسایی یک بلوک کد در یک تابع است که یک کار مجزا را انجام می دهد، سپس آن بلوک را به یک تابع جدید با یک نام توصیفی منتقل می کند.

Singleton:

الگوی Singleton معمولاً در سیستم‌های گزارش‌گیری و اتصالات پایگاه داده به چند دلیل مهم استفاده می‌شود:

مدیریت منابع:

سیستم‌های گزارش‌گیری و اتصالات پایگاه‌داده اغلب شامل منابعی هستند که ایجاد یا نگهداری آنها پرهزینه است، مانند دسته‌های فایل یا اتصالات شبکه.

با استفاده از Singleton، اطمینان حاصل می کنید که تنها یک نمونه از این اشیاء پرمصرف منابع وجود دارد، که باعث کاهش سربار و بهبود عملکرد می شود.

ثبات:

برای ثبت گزارش، داشتن یک نقطه دسترسی منفرد و ثابت برای نوشتن گزارش‌ها بسیار مهم است. این تضمین می کند که ورودی های گزارش به درستی مرتب و قالب بندی شده اند.

در اتصالات پایگاه داده، یک Singleton می تواند یک استخر اتصال را مدیریت کند، از دسترسی ثابت به پایگاه داده اطمینان حاصل کند و از مسائلی مانند نشت اتصال جلوگیری کند.

دسترسی جهانی:

عملیات ورود به سیستم و پایگاه داده اغلب در سراسر یک برنامه مورد نیاز است.

الگوی Singleton یک نقطه دسترسی جهانی را فراهم می‌کند و ثبت پیام‌ها یا انجام عملیات پایگاه داده را از هر قسمتی از کد بدون عبور از لاگر یا اشیاء اتصال آسان می‌کند.

مدیریت دولتی:

لاگرها اغلب نیاز به حفظ وضعیت دارند (به عنوان مثال، سطح گزارش فعلی، مقصد خروجی).

اتصالات پایگاه داده ممکن است نیاز به مدیریت استخرهای اتصال یا وضعیت تراکنش داشته باشند.

یک Singleton می تواند به طور موثر این حالت مشترک را در سراسر برنامه مدیریت کند.

کنترل پیکربندی:

هر دو سیستم گزارش و اتصالات پایگاه داده اغلب به پیکربندی نیاز دارند (به عنوان مثال، مسیرهای فایل گزارش، اعتبار پایگاه داده).

استفاده از Singleton به شما امکان می دهد این پیکربندی را متمرکز کنید و مدیریت و به روز رسانی آن را آسان تر می کند.

Observer

الگوی Observer به چندین دلیل کلیدی به طور گسترده در سیستم های مدیریت رویداد و چارچوب های UI استفاده می شود:

جداسازی اجزا:

این اجازه می دهد تا بین اشیایی که رویدادها (موضوعات) را ایجاد می کنند و اشیایی که به این رویدادها پاسخ می دهند (ناظرها) جدا شود.

این جداسازی سیستم را ماژولارتر کرده و نگهداری آن را آسان‌تر می‌کند.

روابط پویا:

مشاهده‌گرها را می‌توان در زمان اجرا اضافه یا حذف کرد، که امکان ایجاد روابط انعطاف‌پذیر و پویا بین اجزا را فراهم می‌کند.

ارتباط پخش:

این ارتباط کارآمد یک به چند را امکان پذیر می کند، جایی که تغییر در یک شی می تواند باعث به روز رسانی در چندین شیء وابسته شود.

سازگاری در به‌روزرسانی‌های رابط کاربری:

اطمینان حاصل می کند که تمام بخش های رابط کاربری به طور مداوم در صورت تغییر مدل داده های اساسی به روز می شوند.

برنامه نویسی رویداد محور:

معماری‌های رویداد محور را تسهیل می‌کند، که برای اکثر چارچوب‌های UI مدرن اساسی هستند.