سوالات تستی

جواب سوالات تستی

1. B
2. D
3. D
4. A
5. D

سوال ۱

چه تفاوتی بین الگوهای معماری با style های معماری وجود دارد؟ مختصرا شرح دهید.

جواب سوال ۱

الگوهای معماری (Architectural Patterns)

الگوهای معماری، راه‌حل‌های تجربه شده و آزموده‌شده‌ای هستند که برای حل مشکلات معماری خاص در طراحی نرم‌افزار به کار می‌روند. این الگوها شامل دستورالعمل‌ها و رهنمودهایی برای توزیع مسئولیت‌ها در بین اجزای نرم‌افزار هستند. مثال‌هایی از الگوهای معماری شامل معماری سه لایه (Three-Tier Architecture)، مدل‌ـ نما‌ـ کنترلر (MVC)، و میکروسرویس‌ها (Microservices) هستند.

Style های معماری (Architectural Styles)

Style های معماری، بیشتر بر روی اصول و مفاهیم کلی در طراحی سیستم‌های نرم‌افزاری تمرکز دارند و کمتر به جزئیات پیاده‌سازی می‌پردازند. آن‌ها چارچوب کلی‌تری برای درک و بیان ساختار یک سیستم فراهم می‌کنند. مثال‌هایی از Style های معماری شامل سرویس‌گرا (SOA)، رویداد محور (Event-Driven)، و منشوری (Layered) هستند.

تفاوت‌های کلیدی:

الف) *محدوده کاربرد*: الگوهای معماری اغلب با جزئیات بیشتری برای حل مشکلات مشخص در نرم‌افزار تعریف می‌شوند، در حالی که Style های معماری مفاهیم کلی‌تر و چارچوب‌های فکری را ارائه می‌دهند.

ب) *جزئیات و دستورالعمل‌ها*: الگوهای معماری معمولاً دستورالعمل‌های مشخص‌تری برای پیاده‌سازی دارند، در حالی که Style های معماری بیشتر به بیان اصول و مفاهیم پایه‌ای می‌پردازند.

ج) *انعطاف‌پذیری*: Style های معماری اغلب انعطاف‌پذیری بیشتری برای تطبیق با شرایط مختلف دارند، در حالی که الگوهای معماری ممکن است در شرایط خاصی محدودیت‌هایی داشته باشند.

د) *تمرکز بر کیفیت‌های سیستم*: Style های معماری تمرکز بیشتری بر کیفیت‌های کلی سیستم مانند قابلیت اطمینان، امنیت، و قابلیت استفاده دارند، در حالی که الگوهای معماری اغلب بر حل مسائل فنی و معماری تمرکز دارند.

ه) *تطبیق‌پذیری و مقیاس‌پذیری*: در حالی که الگوهای معماری ممکن است در پیاده‌سازی‌های خاص محدودیت‌هایی داشته باشند، Style های معماری اغلب اجازه می‌دهند که سیستم‌ها با تغییرات تکنولوژیکی یا نیازهای تجاری به راحتی تطبیق یابند.

سوال ۲

سناریویی را در نظر بگیرید که در آن کاربر در حال تعامل با یک برنامه موبایل جدید است که برای مدیریت امور مالی شخصی طراحی شده است. این برنامه به کاربران امکان میدهد هزینه‌ها را پیگیری کنند، بودجه را تنظیم کنند و گزارش‌های مالی را مشاهده کنند. با این حال، کاربران برخی از مشکلات را هنگام استفاده از برنامه گزارش کرده‌اند. بر اساس اصول طراحی Bruce Tognazzini ، مشخص کنید کدام اصل(ها) ممکن است در این سناریو نقض شده باشد و دلایل آن را بیان کنید.

مشکلات گزارش شده:

- برنامه اقدامات مربوطه را پیشنهاد نمی‌کند یا مراحل بعدی کاربر را پیش بینی نمی‌کند، مانند پیشنهاد تنظیم بودجه بر اساس الگوهای هزینه‌های گذشته.
- رابط برنامه با عملکردهای بیش از حد در صفحه اصلی به هم ریخته است، که تمرکز روی یک کار واحد مانند وارد کردن هزینه‌های روزانه را دشوار می‌کند.
- همینطور کاربران جدید گزارش کرده‌اند که درک نحوه پیمایش در برنامه و استفاده از ویژگی‌های آن مشکل دارند.
- ساختار Navigation گیج‌کننده است بطوریکه برخی از عملکردها که در زیر چندین لایه از منوها مدفون شده‌اند و یافتن آنها را سخت می‌کند.

جواب سوال ۲

بر اساس اصول طراحی Bruce Tognazzini، چندین اصل ممکن است در این سناریو نقض شده باشند:

الف) **قابلیت پیش‌بینی (Predictability)**: برنامه باید توانایی پیش‌بینی نیازهای کاربر و ارائه پیشنهادات مفید را داشته باشد. نبود این ویژگی در برنامه نشان‌دهنده نقض این اصل است. برای مثال، عدم پیشنهاد تنظیم بودجه بر اساس الگوهای هزینه‌های گذشته می‌تواند به ناکارآمدی در استفاده از برنامه منجر شود.

ب) **سادگی رابط کاربری (Simplicity)**: رابط کاربری باید ساده و قابل فهم باشد. ازدحام عملکردها در صفحه اصلی موجب پیچیدگی و دشواری در استفاده می‌شود که نقض این اصل محسوب می‌شود. یک رابط کاربری بیش از حد پیچیده می‌تواند کاربران را سردرگم کند و تجربه کاربری را کاهش دهد.

ج) **قابلیت پیمایش (Navigability)**: کاربران باید بتوانند به راحتی در برنامه حرکت کنند و به ویژگی‌های مختلف دسترسی داشته باشند. گیج‌کننده بودن ساختار Navigation نشان‌دهنده نقض این اصل است. وقتی کاربران نمی‌توانند به آسانی وظایف مورد نظر خود را در برنامه پیدا کنند، احتمال اینکه از استفاده از برنامه دست بکشند، افزایش می‌یابد.

د) **وضوح و شفافیت (Clarity)**: کاربران باید به راحتی بتوانند از عملکرد و نحوه استفاده هر قسمت از برنامه آگاه شوند. نبود شفافیت در نحوه پیمایش و استفاده از ویژگی‌ها، همچون مواردی که در ساختار Navigation یا عملکردهای پنهان در منوها مشاهده می‌شود، نشان‌دهنده نقض این اصل است.

ه) **پاسخگویی (Responsiveness)**: برنامه باید به نیازهای کاربران به شکلی سریع و مؤثر پاسخ دهد. در صورتی که برنامه نتواند به سرعت و به طور مناسب به ورودی‌ها و درخواست‌های کاربران پاسخ دهد، این اصل نقض شده است.

سوال ۳

چه زمانی از Component Wrapping استفاده می‌کنیم؟ تکنیک‌های مورد استفاده در آن را مختصر توضیح دهید.

جواب سوال ۳

Component Wrapping یک تکنیک در مهندسی نرم‌افزار است که برای ادغام کامپوننت‌های موجود در یک سیستم جدید یا برای افزایش سازگاری بین کامپوننت‌های مختلف استفاده می‌شود. کاربردها و تکنیک‌های مورد استفاده در Component Wrapping عبارتند از:

- **انطباق با معماری‌های جدید**: برای ادغام کامپوننت‌های قدیمی در معماری‌های جدید بدون نیاز به بازنویسی کد.
- **پنهان‌سازی پیچیدگی**: مخفی کردن جزئیات داخلی کامپوننت و ارائه رابط کاربری ساده‌تر.
- **افزایش قابلیت استفاده مجدد**: استفاده مجدد از کامپوننت‌های موجود در محیط‌های مختلف با اندکی یا بدون تغییر در کد اصلی.
- **تسهیل تعامل بین کامپوننت‌ها**: تسهیل ارتباط بین کامپوننت‌هایی که ممکن است در ابتدا برای کار با یکدیگر طراحی نشده باشند.

تکنیک‌های مورد استفاده در Component Wrapping شامل:

- **Adapter Pattern**: استفاده از یک آداپتور برای تبدیل رابط یک کامپوننت به رابطی دیگر که مناسب سیستم جدید است.
- **Facade Pattern**: ایجاد یک واسط ساده برای دسترسی به یک سیستم پیچیده، که می‌تواند چندین کامپوننت پیچیده را پنهان کند.
- **Proxy Pattern**: استفاده از یک کامپوننت نماینده برای کنترل دسترسی به کامپوننت اصلی، مفید برای کنترل دسترسی یا افزودن عملکردهای اضافی.
- **Decorator Pattern**: اضافه کردن رفتار جدید به یک کامپوننت موجود بدون تغییر در کد اصلی آن کامپوننت.

استفاده از این تکنیک‌ها در Component Wrapping به توسعه‌دهندگان اجازه می‌دهد تا از مزایای کامپوننت‌های موجود بهره‌مند شوند و در عین حال انعطاف‌پذیری لازم برای ادغام با سیستم‌های جدید را داشته باشند.

سوال ۴

سناریوهای زیر را در نظر بگیرید. هر سناریو ممکن است اصول طراحی نرم‌افزار شامل اصل OCP (Open/Closed Principle) ، اصل LSP (Liskov Substitution Principle) ، اصل DIP (Dependency Inversion Principle) ، اصل ISP (Interface Segregation Principle) را نقض کرده باشند و یا دارای Coupling بالا یا Cohesion پایین باشند. بررسی نمایید هر یک از این سناریوها چه مشکلی دارند و چرا؟

الف) یک کلاس Animal متدی به نام makeSound دارد. کلاس Dog و کلاس Cat هر دو از Animal به ارث می‌برند. نوع جدیدی از حیوانات به نام Fish اضافه می‌شود، ولی متد makeSound برای آن استفاده نمی‌شود.

ب) یک کلاس رابط کاربری مسئول مدیریت ورودی‌های ماوس، ورودی‌های صفحه کلید، رندر کردن گرافیک و مدیریت پیام‌های شبکه است.

ج) کلاس دسترسی به پایگاه داده یک سیستم نرم‌افزاری از مصرف‌کنندگان می‌خواهد که مدیریت تراکنش، مدیریت اتصال و مدیریت خطا را پیاده‌سازی کنند، حتی اگر فقط به اجرای یک عملیات خواندن ساده نیاز داشته باشند.

د) یک سیستم پردازش پرداخت به یک درگاه پرداخت خاص وابسته است و هر تغییری که در درگاه پرداخت ایجاد شود، تأثیر مستقیمی بر سیستم پردازش پرداخت خواهد داشت.

جواب سوال ۴

در این سناریوها، مشکلات زیر ممکن است وجود داشته باشند:

الف) **نقض اصل LSP و اصل OCP**: اضافه کردن کلاس Fish که متد makeSound را استفاده نمی‌کند، می‌تواند نقض اصل Liskov Substitution Principle (LSP) باشد زیرا کلاس Fish نمی‌تواند به درستی جایگزینی برای کلاس Animal باشد. همچنین، ممکن است این امر نقض اصل Open/Closed Principle (OCP) باشد، زیرا افزودن کلاس جدید نیازمند تغییر در کلاس پایه است.

ب) **نقض اصل ISP**: کلاس رابط کاربری با مسئولیت‌های متعدد، مانند مدیریت ورودی‌های ماوس و صفحه کلید، رندر کردن گرافیک و مدیریت پیام‌های شبکه، نقض اصل Interface Segregation Principle (ISP) است. این کلاس باید به کلاس‌های کوچکتر با مسئولیت‌های محدودتر تقسیم شود تا هر کلاس تنها مسئولیت‌های مرتبط با خود را بر عهده گیرد.

ج) **Cohesion پایین و نقض اصل DIP**: کلاس دسترسی به پایگاه داده که از مصرف‌کنندگان می‌خواهد مدیریت تراکنش، اتصال و خطا را پیاده‌سازی کنند، نشان‌دهنده Cohesion پایین است. این ممکن است نقض اصل Dependency Inversion Principle (DIP) باشد، زیرا کلاس به جای استفاده از انتزاعات، به جزئیات پیاده‌سازی وابسته است.

د) **Coupling بالا**: وابستگی سیستم پردازش پرداخت به یک درگاه پرداخت خاص، نمونه‌ای از Coupling بالا است. این وابستگی باعث می‌شود که هر تغییری در درگاه پرداخت، تأثیر مستقیمی بر سیستم پردازش پرداخت داشته باشد، که انعطاف‌پذیری سیستم را محدود می‌کند و به تغییرات احتمالی در آینده حساس می‌شود.

به این ترتیب، می‌توان دریافت که هر یک از این سناریوها نمونه‌هایی از چالش‌های متداول در طراحی نرم‌افزار هستند که اهمیت توجه به اصول مهندسی نرم‌افزار را نشان می‌دهند. این تحلیل به توسعه‌دهندگان کمک می‌کند تا از این اشتباهات در پروژه‌های آینده اجتناب کرده و به ساخت سیستم‌هایی با کیفیت بالاتر و قابلیت نگهداری بهتر بپردازند.

سوال ۵

در طراحی Pattern-Based ، زمانی که تعداد design pattern هایی که می‌خواهید از بین آن‌ها انتخاب کنید زیاد می‌شود، مرتب‌سازی به یک ضرورت تبدیل می‌شود. چه روشی برای این مرتب‌سازی و انتخاب پیشنهاد می‌کنید؟ شکل کلی روش خود را توضیح دهید.

جواب سوال ۵

برای مرتب‌سازی و انتخاب در میان تعداد زیادی از design patternها در طراحی Pattern-Based، روش زیر پیشنهاد می‌شود:

الف) **تعیین نیازها و محدودیت‌ها**: ابتدا، نیازهای دقیق پروژه و هرگونه محدودیت مربوط به آن (مانند زمان، منابع، و محدودیت‌های تکنولوژیک) را شناسایی کنید.

ب) **دسته‌بندی الگوها**: الگوها را بر اساس دسته‌بندی‌هایی مانند ساختاری، رفتاری، و سازمانی تقسیم‌بندی کنید.

ج) **ارزیابی مطابقت الگوها**: برای هر دسته، الگوهایی که بیشترین مطابقت را با نیازهای پروژه دارند را ارزیابی کنید.

د) **تحلیل ترکیب‌پذیری**: بررسی کنید که چگونه الگوهای انتخاب شده می‌توانند به طور مؤثر با یکدیگر ترکیب شوند تا از تداخل کمتر و همکاری بیشتر بین آن‌ها اطمینان حاصل شود.

ه) **مدل‌سازی و ارزیابی**: پیاده‌سازی مدل اولیه با استفاده از الگوهای انتخاب شده و ارزیابی عملکرد آن در محیط‌های آزمایشی.

و) **افزایش تکراری**: بر اساس بازخورد و تحلیل‌ها، انتخاب الگوها را بهبود بخشیده و به صورت تکراری فرآیند را بهبود دهید.

علاوه بر این، می‌توانیم چند نکته مهم دیگر را نیز در نظر بگیریم:

- **بررسی تعارضات**: ارزیابی اینکه آیا الگوهای انتخاب شده تعارضی با یکدیگر دارند یا خیر. در صورت وجود تعارض، باید تصمیم‌گیری شود که کدام الگو برای پروژه مناسب‌تر است.
- **انعطاف‌پذیری و مقیاس‌پذیری**: توجه داشته باشید که الگوهای انتخابی باید انعطاف‌پذیر و مقیاس‌پذیر باشند تا بتوانند با تغییرات احتمالی در پروژه همگام شوند.
- **ارزیابی مستندات و جامعه کاربری**: بررسی میزان پشتیبانی و مستندات موجود برای هر الگو و همچنین تجربیات جامعه کاربری می‌تواند در انتخاب الگوهای مناسب کمک کننده باشد.

این رویکرد جامع به تیم‌های توسعه کمک می‌کند تا الگوهای مناسب را با در نظر گرفتن تمام جوانب و محدودیت‌های پروژه انتخاب کنند، و در نتیجه به ساخت نرم‌افزاری مؤثر و کارآمد بیانجامد. اجازه می‌دهد تا راه‌حل‌های مؤثر و کارآمدی بیابند.