سوالات تستی

جواب سوالات تستی

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

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.

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.

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.

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.

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. از Component Wrapping عمدتاً در موقعیت‌های زیر استفاده می‌شود:

- **انطباق با معماری‌های جدید**: زمانی که نیاز است تا کامپوننت‌های موجود در یک معماری جدید بدون تغییر عمده کد اصلی قرار گیرند.
- **پنهان‌سازی پیچیدگی**: برای مخفی کردن پیچیدگی‌های داخلی یک کامپوننت و فراهم کردن یک رابط ساده‌تر برای استفاده‌کنندگان.
- **افزایش قابلیت استفاده مجدد**: امکان استفاده مجدد از کامپوننت‌های قدیمی در محیط‌های جدید با استفاده از رابط‌های جدید.

تکنیک‌های مورد استفاده در Component Wrapping شامل:

- Adapter Pattern: استفاده از یک واسط یا آداپتور برای تبدیل رابط یک کامپوننت به رابطی دیگر که مورد نیاز است.
- Facade Pattern: ایجاد یک واسط ساده برای یک سیستم پیچیده، برای ساده‌سازی دسترسی به آن.

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**: در مورد کلاس Fish که متد makeSound را استفاده نمی‌کند، این نشان‌دهنده نقض اصل Substitution Liskov (LSP) است زیرا Fish نمی‌تواند به درستی جایگزین Animal شود. همچنین، این ممکن است نقض اصل Open/Closed (OCP) باشد زیرا برای اضافه کردن Fish نیاز به تغییر در کلاس Animal است.

ب) **نقض اصل ISP**: کلاس رابط کاربری که مسئولیت‌های زیادی دارد، نقض اصل Segregation Interface (ISP) است. بهتر است که وظایف به کلاس‌های تخصصی‌تر تقسیم شوند تا هر کلاس فقط مسئولیت‌های مرتبط با خود را داشته باشد.

ج) **Cohesion پایین و نقض اصل DIP**: کلاس دسترسی به پایگاه داده با مسئولیت‌های گسترده و اجبار مصرف‌کنندگان برای پیاده‌سازی تمام ویژگی‌ها، نشان‌دهنده Cohesion پایین است. این همچنین ممکن است نقض اصل Inversion Dependency (DIP) باشد، زیرا کلاس به جای استفاده از انتزاعات، مستقیماً به جزئیات پیاده‌سازی وابسته است.

د) **Coupling بالا**: سیستم پردازش پرداخت که به شدت به یک درگاه پرداخت خاص وابسته است، نمونه‌ای از Coupling بالا است. این وابستگی موجب می‌شود که هر تغییری در درگاه پرداخت تأثیر مستقیمی بر سیستم داشته باشد.

سوال ۵

با در نظر گرفتن رویکرد چابک به سوالات زیر پاسخ دهید:

A. در اکثر پروژه‌های نرم‌افزاری پیش‌بینی موارد زیر سخت است:

- اینکه کدام نیازمندی‌های مشتری تغییر خواهند کرد و کدام نیازمندی‌ها ثابت خواهند بود؟
- اینکه به چه میزان طراحی پیش از پیاده‌سازی احتیاج داریم؟
- و چه مقدار زمان از نظر برنامه‌ریزی برای تحلیل و طراحی، پیاده‌سازی و تست محصول نیاز خواهد بود؟

فرآیندهای چابک چگونه در جهت رفع این شرایط‌های نیاز به پیش‌بینی پاسخ می‌دهند؟

.B

اگر برای سیستم‌های بزرگ و با عمر طولانی که توسط یک شرکت نرم‌افزاری برای مشتریان خارجی توسعه داده می‌شوند، از رویکرد چابک استفاده شود، چه مشکلاتی ممکن است بوجود آید؟. ۳ مورد از مشکلات ممکن را ذکر کنید.

.C

فکر می‌کنید مدل‌های چابک خود چه مشکلاتی داشته باشند؟ (حداقل ۴ مورد)

جواب سوال ۵

.A

در فرآیندهای چابک، به جای تلاش برای پیش‌بینی دقیق تمام نیازمندی‌ها و طراحی‌ها از ابتدا، تمرکز بر توسعه تدریجی و تکراری محصول است. این رویکرد اجازه می‌دهد تا تیم‌ها به سرعت و با انعطاف‌پذیری بالا به تغییرات نیازمندی‌ها واکنش نشان دهند. زمان‌بندی برای تحلیل، طراحی، پیاده‌سازی و تست نیز به صورت انعطاف‌پذیر در چرخه‌های کوتاه مدت مدیریت می‌شود.

فرآیندهای چابک، بر تعامل نزدیک با مشتری تأکید دارند. این امر به تیم توسعه نرم‌افزار کمک می‌کند تا نیازمندی‌های مشتری را به طور دقیق و کامل شناسایی کنند و تغییرات را به سرعت شناسایی و اعمال کنند.

فرآیندهای چابک، بر آزمایش مداوم تأکید دارند. این امر به تیم توسعه نرم‌افزار کمک می‌کند تا کیفیت محصول را در طول توسعه تضمین کنند و از تغییرات غیرمنتظره در نیازمندی‌های مشتری جلوگیری کنند.

فرآیندهای چابک، انعطاف‌پذیر هستند و اجازه می‌دهند تا تغییرات در نیازمندی‌ها، طراحی و برنامه‌ریزی پروژه به سرعت اعمال شوند. این امر به تیم توسعه نرم‌افزار کمک می‌کند تا به تغییرات محیطی و نیازهای مشتری پاسخ دهند.

فرض کنید یک شرکت نرم‌افزاری برای یک مشتری جدید، یک سیستم اتوماسیون فروش توسعه می‌دهد. مشتری نیازهای خود را به صورت کلی به تیم توسعه نرم‌افزار ارائه می‌دهد. تیم توسعه نرم‌افزار با تعامل نزدیک با مشتری، نیازهای مشتری را به صورت دقیق‌تر شناسایی می‌کند. سپس، محصول را در فواصل زمانی کوتاه تحویل می‌دهد تا مشتری بتواند آن را بررسی کند و تغییرات را درخواست کند.

در این مثال، فرآیندهای چابک به تیم توسعه نرم‌افزار کمک می‌کند تا به تغییرات در نیازمندی‌های مشتری پاسخ دهند. به عنوان مثال، اگر مشتری نیاز به اضافه کردن یک ویژگی جدید به سیستم را داشته باشد، تیم توسعه نرم‌افزار می‌تواند این ویژگی را در نسخه بعدی محصول اعمال کند.

در نتیجه فرآیندهای چابک، با توجه به چالش‌های پیش‌بینی در پروژه‌های نرم‌افزاری، از رویکردهای مختلفی استفاده می‌کنند. این رویکردها به تیم‌های توسعه نرم‌افزار کمک می‌کند تا تغییرات در نیازمندی‌های مشتری را به سرعت شناسایی و اعمال کنند و محصولی با کیفیت بالا تحویل دهند.

.B

برای سیستم‌های بزرگ و با عمر طولانی، استفاده از رویکرد چابک ممکن است مشکلاتی از قبیل:

- دشواری در مدیریت و هماهنگی بین تیم‌های بزرگ و پراکنده.
- چالش‌های مربوط به برقراری ارتباط مؤثر با مشتریان بین‌المللی.
- مسائل مربوط به مقیاس‌پذیری و ادغام مداوم تغییرات در یک سیستم بزرگ.

سیستم‌های بزرگ و با عمر طولانی، معمولاً توسط تیم‌های بزرگ و پراکنده توسعه داده می‌شوند. این تیم‌ها ممکن است در مکان‌های مختلف قرار داشته باشند و از فرهنگ‌ها و زبان‌های مختلف باشند. مدیریت و هماهنگی بین این تیم‌ها، در رویکرد چابک که بر تعامل نزدیک با مشتری تأکید دارد، می‌تواند دشوار باشد.

برای حل این مشکل، تیم‌های توسعه نرم‌افزار باید از ابزارها و تکنیک‌های ارتباطی و همکاری موثر استفاده کنند. همچنین، باید یک برنامه‌ریزی دقیق برای مدیریت و هماهنگی بین تیم‌ها داشته باشند.

اگر سیستم توسط یک شرکت نرم‌افزاری برای مشتریان خارجی توسعه داده شود، تیم توسعه نرم‌افزار باید بتواند با مشتریان بین‌المللی به طور موثر ارتباط برقرار کند. این امر می‌تواند چالش‌هایی را ایجاد کند، به خصوص اگر مشتریان از زبان‌ها و فرهنگ‌های مختلف باشند.

برای حل این مشکل، تیم توسعه نرم‌افزار باید مهارت‌های ارتباطی بین‌المللی داشته باشد. همچنین، باید از ابزارها و تکنیک‌های ارتباطی موثر استفاده کند.

سیستم‌های بزرگ و با عمر طولانی، معمولاً پیچیده هستند و نیاز به نگهداری و پشتیبانی مداوم دارند. در رویکرد چابک، تغییرات در سیستم به صورت مداوم اعمال می‌شوند. این امر می‌تواند چالش‌هایی را در زمینه مقیاس‌پذیری و ادغام تغییرات ایجاد کند.

برای حل این مشکل، تیم توسعه نرم‌افزار باید از ابزارها و تکنیک‌های مناسب برای مدیریت تغییرات استفاده کند. همچنین، باید یک برنامه‌ریزی دقیق برای مدیریت مقیاس‌پذیری سیستم داشته باشد.

البته، استفاده از رویکرد چابک برای سیستم‌های بزرگ و با عمر طولانی، مزایای زیادی نیز دارد. به عنوان مثال، چابکی می‌تواند به تیم‌های توسعه نرم‌افزار کمک کند تا به تغییرات نیازهای مشتری به سرعت پاسخ دهند.

در نهایت، انتخاب رویکرد مناسب برای توسعه یک سیستم نرم‌افزاری، به عوامل مختلفی مانند اندازه سیستم، پیچیدگی سیستم، و الزامات مشتری بستگی دارد.

.C

مدل‌های چابک، به دلیل انعطاف‌پذیری و تمرکز بر تحویل ارزش به مشتری، مزایای زیادی نسبت به مدل‌های سنتی دارند. با این حال، این مدل‌ها نیز می‌توانند مشکلاتی داشته باشند. در ادامه، چهار مورد از مشکلات احتمالی مدل‌های چابک را بررسی می‌کنیم:

مدل‌های چابک خود می‌توانند مشکلاتی داشته باشند از جمله:

- نیاز به ارتباط و همکاری مداوم و نزدیک با مشتری.

  یکی از اصول کلیدی مدل‌های چابک، ارتباط و همکاری مداوم و نزدیک با مشتری است. این امر به تیم توسعه نرم‌افزار کمک می‌کند تا نیازهای مشتری را به طور دقیق و کامل شناسایی کنند و تغییرات را به سرعت شناسایی و اعمال کنند.

  با این حال، این امر می‌تواند چالش‌هایی را برای تیم توسعه نرم‌افزار ایجاد کند. به عنوان مثال، ممکن است مشتری در دسترس نباشد یا زمان کافی برای همکاری با تیم توسعه نرم‌افزار را نداشته باشد. همچنین، ممکن است مشتری نیازهای خود را به طور دقیق نداند و تغییرات مکرر در نیازهای خود ایجاد کند.

- دشواری در پیش‌بینی هزینه‌ها و زمان‌بندی‌های بلندمدت.

  یکی دیگر از مشکلات احتمالی مدل‌های چابک، دشواری در پیش‌بینی هزینه‌ها و زمان‌بندی‌های بلندمدت است. این امر به دلیل انعطاف‌پذیری بالای مدل‌های چابک و امکان تغییرات مکرر در الزامات پروژه است.

  در مدل‌های سنتی، معمولاً یک برنامه‌ریزی دقیق در ابتدای پروژه انجام می‌شود و هزینه‌ها و زمان‌بندی‌های بلندمدت بر اساس این برنامه‌ریزی تخمین زده می‌شوند. با این حال، در مدل‌های چابک، این برنامه‌ریزی معمولاً در طول پروژه انجام می‌شود و هزینه‌ها و زمان‌بندی‌های بلندمدت به صورت مداوم به روز می‌شوند.

- خطر انحراف از اهداف اصلی در صورت عدم وجود برنامه‌ریزی دقیق.

  در مدل‌های چابک، بر تحویل ارزش به مشتری در فواصل زمانی کوتاه تأکید می‌شود. این امر می‌تواند منجر به انحراف از اهداف اصلی پروژه شود، به خصوص اگر برنامه‌ریزی دقیقی برای پروژه انجام نشود.

برای جلوگیری از این مشکل، تیم توسعه نرم‌افزار باید اهداف اصلی پروژه را به خوبی شناسایی کند و یک برنامه‌ریزی دقیق برای پروژه داشته باشد. همچنین، باید در طول پروژه، اهداف اصلی پروژه را به طور مداوم بررسی کند و در صورت لزوم، تغییرات لازم را اعمال کند.

- ممکن است کیفیت کد در اثر تغییرات مکرر کاهش یابد.

یکی از ویژگی‌های مدل‌های چابک، تغییرات مکرر در محصول است. این امر می‌تواند منجر به کاهش کیفیت کد شود، به خصوص اگر تیم توسعه نرم‌افزار از روش‌های صحیح مدیریت تغییرات استفاده نکند.

برای جلوگیری از این مشکل، تیم توسعه نرم‌افزار باید از روش‌های صحیح مدیریت تغییرات استفاده کند. همچنین، باید کیفیت کد را در طول پروژه به طور مداوم بررسی کند و در صورت لزوم، اقدامات لازم را برای بهبود کیفیت کد انجام دهد.

چند مورد دیگر نیز هستند مانند « تمرکز زیادی روی اهداف کوتاه مدت» و «داکیومنتیشن نه‌چندان کافی و مناسب» و «نه‌چندان مناسب برای پروژه‌های خیلی مهم و حساس» که آن‌ها نیز قابل بررسی هستند.
