بسم الله الرحمن الرحیم

تحلیل و طراحی سیستم‌ها

مدرس : مهدی مجد

تئوری سیستم‌ها (Systems theory)

تئوری سیستم‌ها یک چارچوب نظری گسترده است که به مطالعه و تحلیل سیستم‌های پیچیده و روابط بین اجزاء آن‌ها می‌پردازد. این تئوری در بسیاری از زمینه‌ها، از جمله مهندسی، مدیریت، زیست‌شناسی، اقتصاد و علوم اجتماعی کاربرد دارد. در ادامه به برخی از مفاهیم کلیدی تئوری سیستم‌ها می‌پردازیم.

تعریف سیستم: مجموعه ای از اجزا و رابطه ها است. (یک سری اجزا که باهم رابطه و فعالیت و همکاری دارند.)

در این تعریف هدف را به ناظر بیرونی که سیستم را به کار می‌گیرد، وابسته می‌کنیم. به عبارت دیگر ناظر بیرونی کاربرد (هدف) مناسب خود را در حیطه تعریف شده‌ی آن سیستم بکار می‌گیرد.

مثال : صندلی یک سیستم است که در آن مجموعه ای از اجزا با هم در ارتباط هستند و می‌توان هدف این سیستم وابسته به ناظر درنظر گرفت به عنوان نمونه اهدافی مثل نسشتن، استفاده به عنوان پایه ای برای نگه داشتن در اتاق و کلاس، چهارپایه، یک اثر هنری یا تاریخی و ... ذا می‌توان در نظر گرفت.

تعریفی دیگر از سیستم : مجموعه‌ای از اجزا که برای رسیدن به یک هدف، با هم همکاری فعالیت می‌کنند.

طبق تعریف اول می‌توان رابطه‌ی زیر را در نظر گرفت.

Entities + Relationships ⟶ System

در نتیجه می‌توانیم سیستم را که مجموعه‌ای از اجزا و رابطه‌ها در نظر گرفت را به صورت زیر تعریف کنیم.

$S = (E , R)$

به اجزاء، نام‌های مختلف دیگری را نیز می‌توان نسبت داد. همانند چیزی (thing) ، شئ (object) و ... .

رابطه را می‌توان به صورت یک دوتایی نمایش داد. $(x,y) \longrightarrow R$ که در اینجا x و y جزئی از سیستم هستند که با رابطه‌ی R با هم همکاری دارند.

در تئوری سیستم‌ها مفاهیم مختلفی وجود دارد که با چند مورد ضروری از آنها آشنا خواهیم شد.

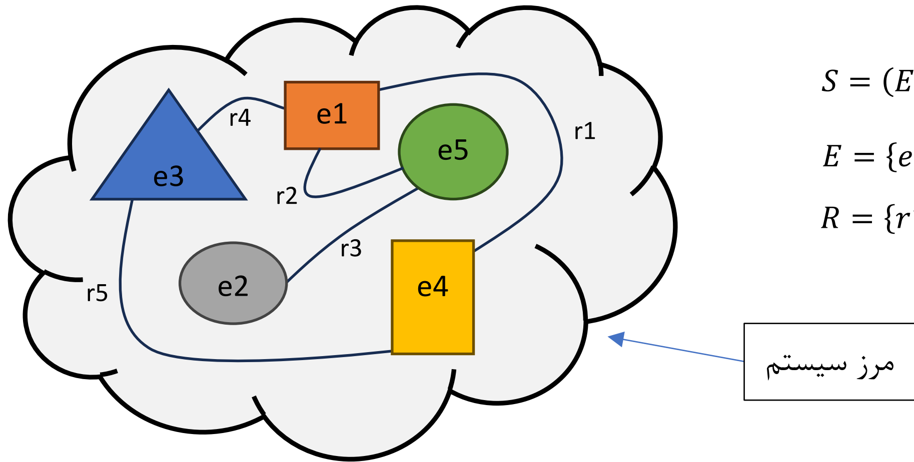
مرز سیستم (System Boundary) :

مرز سیستم خط فرضی است که سیستم را از محیط آن جدا می‌کند. در اینجا اجزاء درونی سیستم و نحوه‌ی ارتباط آنها باید مشخص شود.

$S = (E, R)$

$E = \{e1, e2, e3, e4, e5\}$

$R = \{r1, r2, r3, r4, r5\}$



ارزش کل سیستم (Total System Value) :

ارزش کلی سیستم مجموع ارزش فردی اجزاء و ارزش تعاملات و روابط بین آن‌ها است.

- *ارزش اجزاء (Component Value)* : ارزش هر یک از اجزاء مستقل سیستم که می‌تواند بر اساس معیارهای مختلفی مانند عملکرد، کارآیی و قابلیت نگهداری محاسبه شود.
- *ارزش رابطه‌ها (Interaction Value)* : ارزش تعاملات و روابط بین اجزاء که می‌تواند بر اساس معیارهای مانند هماهنگی، هم‌افزایی و کارآیی ارتباطات محاسبه شود.

ارزش کل سیستم > مجموع ارزش اجزاء سیستم

علت بیشتر شدن ارزش کل سیستم از مجموع ارزش اجزاء سیستم افزوده شدن ارزش سیستم بر اثر هم‌افزایی اجزاء سیستم می‌باشد.

هم‌افزایی (Synergy) :

هم‌افزایی به وضعیتی اشاره دارد که اثر کلی سیستم بیش از مجموع اثرات اجزاء آن است. همانند عملکرد هماهنگ اعضای بدن، همکاری موثر بین واحدهای مختلف کارخانه، تعاملات مثبت بین اجزاء نرم‌افزار.

زیرسیستم(Subsystem) :

در تئوری سیستم‌ها، یک زیرسیستم (Subsystem) بخشی از یک سیستم بزرگتر است که به عنوان یک سیستم مستقل نیز عمل می‌کند. زیرسیستم‌ها معمولاً دارای وظایف، اجزاء و روابط داخلی خاص خود هستند و با دیگر زیرسیستم‌ها و سیستم اصلی تعامل دارند.

مهمترین مزیت ایجاد زیر سیستم‌ها کاهش پیچیدگی است. با تقسیم سیستم به زیرسیستم‌های کوچکتر، پیچیدگی کلی سیستم کاهش می‌یابد و مدیریت آن آسان‌تر می‌شود.

چرخه حیات توسعه سیستم‌ها (Systems Development Life Cycle)

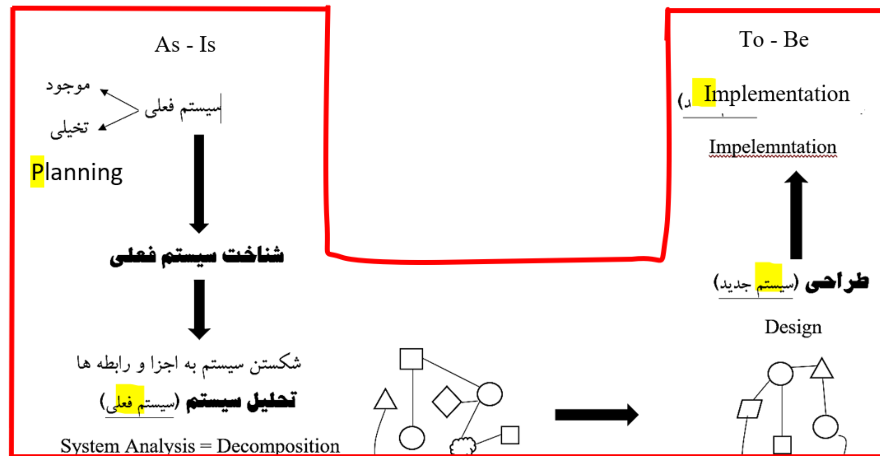
ساخت یک سیستم اطلاعاتی به نوعی شبیه ساخت یک خانه است. ابتدا، یک ایده اولیه شکل می‌گیرد. سپس، این ایده به یک طرح ساده تبدیل می‌شود که به مشتری نمایش داده شده و اصلاح می‌گردد تا زمانی که مشتری آن را تأیید کند. در مرحله بعد، نقشه‌های دقیق‌تری طراحی می‌شوند که جزئیات بیشتری از خانه را ارائه می‌دهند. در نهایت، خانه بر اساس این نقشه‌ها ساخته می‌شود، معمولاً با تغییراتی که مشتری درخواست می‌کند.

چرخه حیات توسعه سیستم‌ها (SDLC) شامل چهار مرحله اساسی است: برنامه‌ریزی، تحلیل، طراحی و پیاده‌سازی. پروژه‌های مختلف ممکن است بر جنبه‌های متفاوتی از SDLC تأکید کنند، اما همه پروژه‌ها عناصر این چهار مرحله را دارند. هر مرحله شامل یک سری مراحل است که به تکنیک‌هایی متکی هستند که محصولات نهایی (مدارک و فایل‌های خاص) را تولید می‌کنند.

به عنوان مثال، در درخواست پذیرش به دانشگاه، همه دانشجویان از مراحل مشابهی عبور می‌کنند: جمع‌آوری اطلاعات، درخواست و پذیرش. هر یک از این مراحل شامل قدم‌هایی است؛ برای مثال، جمع‌آوری اطلاعات شامل جستجوی مدارس، درخواست اطلاعات و مطالعه بروشورها. سپس دانشجویان از تکنیک‌هایی (مانند جستجوی اینترنتی) استفاده می‌کنند که می‌توانند به این قدم‌ها (مانند درخواست اطلاعات) اعمال شوند تا خروجی‌هایی (مانند ارزیابی جنبه‌های مختلف دانشگاه‌ها) ایجاد کنند

در حال حاضر، دو نکته مهم درباره SDLC وجود دارد که باید درک کنید. ابتدا باید یک حس کلی از مراحل و قدم‌هایی که پروژه‌های سیستم‌های اطلاعاتی از آن عبور می‌کنند و سپس برخی از تکنیک‌هایی که خروجی‌های خاصی تولید می‌کنند، داشته باشید. ثانیاً، درک این نکته مهم است که SDLC یک فرآیند تصحیح تدریجی است. خروجی‌های تولیدشده در مرحله تحلیل، ایده‌ای کلی از شکل سیستم جدید را ارائه می‌دهند. این خروجی‌ها به‌عنوان ورودی به مرحله طراحی استفاده می‌شوند، که سپس آن‌ها را تصحیح می‌کند تا مجموعه‌ای از خروجی‌ها

تولید کند که به‌طور دقیق‌تری توصیف می‌کند که سیستم چگونه ساخته خواهد شد. این خروجی‌ها نیز در مرحله پیاده‌سازی برای تولید سیستم واقعی مورد استفاده قرار می‌گیرند. هر مرحله بر کارهای قبلی بنا می‌شود و آن‌ها را گسترش می‌دهد.



برنامه‌ریزی (Planning)

مرحله برنامه‌ریزی فرآیند اساسی در درک این موضوع است که *چرا* یک *سیستم اطلاعاتی باید ساخته شود* و *تعیین چگونگی ساخت آن توسط تیم پروژه* (چرا و چگونه ساخته شود). این مرحله شامل دو قدم است:

۱. **شروع پروژه**: در این مرحله، ارزش تجاری سیستم برای سازمان شناسایی می‌شود: چگونه می‌تواند هزینه‌ها را کاهش دهد یا درآمدها را افزایش دهد؟ بیشتر ایده‌های مربوط به سیستم‌های جدید از خارج از حوزه سیستم‌های اطلاعاتی (مثلاً از بخش بازاریابی یا حسابداری) به‌صورت درخواست سیستم مطرح می‌شوند. یک درخواست سیستم خلاصه‌ای کوتاه از یک نیاز تجاری را ارائه می‌دهد و توضیح می‌دهد که چگونه سیستمی که از این نیاز حمایت می‌کند، ارزش تجاری ایجاد خواهد کرد. بخش سیستم‌های اطلاعاتی به همراه شخص یا بخشی که درخواست را ایجاد کرده است (که به آن حامی پروژه گفته می‌شود) برای انجام *تحلیل امکان‌پذیری* (feasibility) همکاری می‌کند. درخواست سیستم و تحلیل *امکان‌پذیری* به یک کمیته تأیید سیستم‌های اطلاعاتی (که گاهی اوقات کمیته راهبری نامیده می‌شود) ارائه می‌شوند که تصمیم می‌گیرد آیا پروژه باید انجام شود یا نه.

مدیریت پروژه: پس از تأیید پروژه، وارد مرحله مدیریت پروژه می‌شود. در این مرحله، مدیر پروژه یک برنامه کاری ایجاد می‌کند، اعضای پروژه را تعیین می‌کند و تکنیک‌هایی را به‌کار می‌گیرد تا به تیم پروژه کمک کند تا پروژه را در طول تمام مراحل SDLC کنترل و هدایت کند. خروجی مرحله مدیریت پروژه، یک برنامه پروژه است که توضیح می‌دهد تیم پروژه چگونه به توسعه سیستم خواهد پرداخت.

تحلیل (Analysis)

مرحله تحلیل به سوالات مربوط به اینکه *چه کسی از سیستم استفاده خواهد کرد*؟ *سیستم چه کاری انجام خواهد داد*؟ و *کجا و چه زمانی از آن استفاده خواهد شد*؟ پاسخ می‌دهد. در این مرحله، تیم پروژه به بررسی سیستم‌های فعلی می‌پردازد، فرصت‌های بهبود را شناسایی می‌کند و مفهومی برای سیستم جدید توسعه می‌دهد.

این مرحله شامل سه قدم است :

۱. **توسعه استراتژی تحلیل**: یک استراتژی تحلیل برای هدایت تلاش‌های تیم پروژه ایجاد می‌شود. این استراتژی معمولاً شامل تحلیل سیستم فعلی (که به آن سیستم موجود می‌گویند) و مشکلات آن و سپس راه‌هایی برای طراحی یک سیستم جدید (که به آن سیستم آینده می‌گویند) است.

۲. **جمع‌آوری نیازها**: مرحله بعدی جمع‌آوری نیازها است (مثلاً از طریق مصاحبه‌ها یا پرسشنامه‌ها). تحلیل این اطلاعات، به همراه ورودی از حامی پروژه (و دیگر افراد، منجر به توسعه مفهومی برای سیستم جدید می‌شود. این مفهوم به‌عنوان پایه‌ای برای توسعه مجموعه‌ای از مدل‌های تحلیل کسب‌وکار استفاده می‌شود که توصیف می‌کند اگر سیستم جدید توسعه یابد، کسب‌وکار چگونه عمل خواهد کرد.

۳. **تدوین پیشنهاد سیستم**: تحلیل‌ها، مفهوم سیستم و مدل‌ها در یک سند به‌نام پیشنهاد سیستم (system proposal) ترکیب می‌شوند که به حامی پروژه (کارفرما) و دیگر تصمیم‌گیرندگان کلیدی (مثلاً اعضای کمیته تأیید) ارائه می‌شود تا تصمیم بگیرند آیا پروژه باید به جلو برود یا خیر.

پیشنهاد سیستم اولین خروجی است که توضیح می‌دهد سیستم جدید باید چه نیازهای تجاری را برآورده کند. از آنجا که این مرحله واقعاً اولین قدم در طراحی سیستم جدید است، برخی کارشناسان معتقدند که استفاده از اصطلاح "تحلیل" برای نام‌گذاری این مرحله مناسب نیست و نام بهتری مانند "تحلیل و طراحی اولیه" پیشنهاد می‌دهند. با این حال، بیشتر سازمان‌ها همچنان از نام تحلیل برای این مرحله استفاده می‌کنند.

طراحی (Design)

در مرحله طراحی تصمیم گرفته می‌شود که سیستم از نظر سخت‌افزار، نرم‌افزار و زیرساخت شبکه؛ رابط کاربری، فرم‌ها و گزارش‌ها؛ و برنامه‌ها، پایگاه‌های داده و فایل‌های خاصی که نیاز خواهد بود، چگونه عمل خواهد کرد. اگرچه بیشتر تصمیمات استراتژیک درباره سیستم در توسعه مفهوم سیستم در مرحله تحلیل گرفته شده است، مراحل طراحی دقیقاً مشخص می‌کنند که سیستم چگونه عمل خواهد کرد. مرحله طراحی شامل چهار قدم است:

۱. **توسعه استراتژی طراحی**: ابتدا استراتژی طراحی توسعه داده می‌شود. این مرحله مشخص می‌کند که آیا سیستم توسط برنامه‌نویسان خود شرکت توسعه خواهد یافت، آیا سیستم به یک شرکت دیگر (معمولاً یک شرکت مشاوره) واگذار خواهد شد، یا اینکه شرکت یک بسته نرم‌افزاری موجود را خریداری خواهد کرد.

۲. **توسعه طراحی معماری**: این مرحله منجر به توسعه طراحی معماری پایه برای سیستم می‌شود که سخت‌افزار، نرم‌افزار و زیرساخت شبکه مورد استفاده را توصیف می‌کند. در بیشتر موارد، سیستم زیرساختی که در حال حاضر در سازمان وجود دارد را اضافه یا تغییر می‌دهد. طراحی رابط مشخص می‌کند که کاربران چگونه در سیستم حرکت خواهند کرد (مثلاً روش‌های ناوبری مانند منوها و دکمه‌های روی صفحه) و فرم‌ها و گزارش‌هایی که سیستم استفاده خواهد کرد.

۳. **توسعه مشخصات پایگاه داده و فایل**: این مرحله دقیقاً تعریف می‌کند که چه داده‌هایی ذخیره خواهند شد و کجا ذخیره خواهند شد.

۴. **توسعه طراحی برنامه**: تیم تحلیل طراحی برنامه را توسعه می‌دهد که برنامه‌هایی را که باید نوشته شوند و دقیقاً هر برنامه چه کاری انجام خواهد داد، تعریف می‌کند.

این مجموعه خروجی‌ها (طراحی معماری، طراحی رابط، مشخصات پایگاه داده و فایل، و طراحی برنامه) مشخصات سیستم است که به تیم برنامه‌نویسی برای پیاده‌سازی ارائه می‌شود. در پایان مرحله طراحی، تحلیل feasibility و برنامه پروژه دوباره بررسی و اصلاح می‌شوند و یک تصمیم دیگر توسط حامی پروژه و کمیته تأیید درباره اینکه آیا پروژه باید متوقف شود یا ادامه یابد، اتخاذ می‌شود.

پیاده‌سازی (Implementation)

آخرین مرحله در SDLC ، مرحله پیاده‌سازی است، که در آن سیستم واقعاً ساخته می‌شود. این مرحله معمولاً بیشترین توجه را جلب می‌کند، زیرا برای اکثر سیستم‌ها، طولانی‌ترین و پرهزینه‌ترین بخش از فرآیند توسعه است. این مرحله شامل سه قدم است:

۱. **ساخت سیستم**: اولین قدم، ساخت و آزمایش سیستم است تا اطمینان حاصل شود که به‌درستی عمل می‌کند. به‌دلیل هزینه‌های بالای ناشی از اشکالات، آزمایش یکی از مهم‌ترین مراحل در پیاده‌سازی است. بیشتر سازمان‌ها به آزمایش زمان و توجه بیشتری نسبت به نوشتن برنامه‌ها اختصاص می‌دهند.

۲. **نصب سیستم**: نصب فرایندی است که در آن سیستم قدیمی خاموش و سیستم جدید روشن می‌شود. یکی از مهم‌ترین جنبه‌های تبدیل، توسعه یک برنامه آموزشی برای آموزش کاربران در استفاده از سیستم جدید و کمک به مدیریت تغییرات ناشی از سیستم جدید است.

۳. **توسعه برنامه پشتیبانی**: تیم تحلیل یک برنامه پشتیبانی برای سیستم ایجاد می‌کند. این برنامه معمولاً شامل یک بازبینی رسمی یا غیررسمی پس از پیاده‌سازی و همچنین یک روش سیستماتیک برای شناسایی تغییرات عمده و جزئی مورد نیاز برای سیستم است.

مدل‌های فرایند توسعه‌ی سیستم

مدل‌های فرآیند توسعه نرم‌افزار چارچوب‌ها و روش‌هایی هستند که برای مدیریت و هدایت پروژه‌های نرم‌افزاری استفاده می‌شوند. این مدل‌ها به تیم‌های توسعه کمک می‌کنند تا پروژه‌ها را به شیوه‌ای ساختارمند و قابل مدیریت انجام دهند. بعبارتی دیگر نحوه و ترتیب انجام فعالیت و کارهای مورد نیاز جهت رسیدن از سیستم فعلی به سیستم جدید است. (مشخص کردن ترتیب انجام کارها است.)

۱- مدل آبشاری (Waterfall Model)

مدل آبشاری یکی از قدیمی‌ترین و ساده‌ترین مدل‌ها برای توسعه نرم‌افزار است. این مدل بر اساس یک ترتیب خطی و مرحله‌به‌مرحله کار می‌کند و هر مرحله پس از اتمام مرحله قبلی آغاز می‌شود. مراحل آن شامل برنامه‌ریزی، تحلیل، طراحی، پیاده‌سازی، آزمایش، و نگهداری است. در این مدل، امکان بازگشت به مراحل قبل محدود است.

این مدل برای سیستم های کوچک مناسب است.

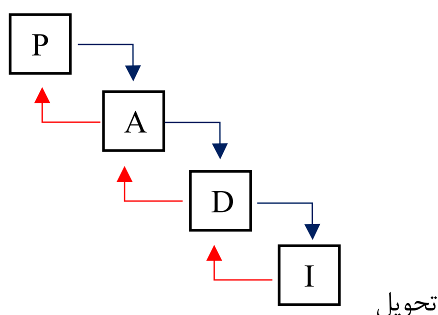
بدترین خطا ، خطایی است که در مرحله ی شناخت سیستم رخداده و به زودی نیز کشف نشده تا مراحل پایانی پیاده‌سازی نیز باقیمانده است. (خطاهای تحلیل نیازمندی‌ها)

مزایا:

- ساختار مشخص و قابل پیش‌بینی
- مناسب برای پروژه‌هایی با نیازهای ثابت

معایب:

- عدم انعطاف‌پذیری در تغییرات
- مناسب نبودن برای پروژه‌های پیچیده و بزرگ



شکل ... : مدل آبشاری - مسیر بازگشت یک استثناء برای زمانی است که خطای یا ضعفی کشف شود.

۲- **مدل توسعه موازی** (Parallel Development Model)

این مدل از نیاز به توسعه سریع و کاهش زمان تا بازار پدید آمده و به‌ویژه برای پروژه‌هایی مناسب است که بخش‌های مختلف نرم‌افزار به‌صورت موازی می‌توانند توسعه یابند. در مدل موازی، پروژه به چندین زیرپروژه یا بخش تقسیم می‌شود، و هر بخش یا زیرپروژه به صورت جداگانه و هم‌زمان توسعه می‌یابد. سپس این بخش‌ها با هم ادغام شده و به‌عنوان یک محصول نهایی مورد آزمایش و ارزیابی قرار می‌گیرند.

ویژگی‌ها و مزایای مدل موازی

۱. **کاهش زمان توسعه:** با تقسیم پروژه به بخش‌های کوچک‌تر و اجرای هم‌زمان آن‌ها، زمان کلی پروژه کاهش می‌یابد.

۲. **افزایش بهره‌وری:** تیم‌های مختلف می‌توانند به طور موازی و مستقل روی بخش‌های مختلف پروژه کار کنند.

۳. **تسهیل مدیریت تغییرات :**با کار روی بخش‌های کوچک‌تر، تغییرات و بازنگری‌ها را می‌توان سریع‌تر و با هزینه کمتری انجام داد.

معایب مدل موازی

۱. **افزایش پیچیدگی ادغام:** هر چه بخش‌های بیشتری به‌صورت هم‌زمان توسعه یابند، هماهنگی و ادغام آن‌ها سخت‌تر می‌شود.

۲. **نیاز به مدیریت قوی:** برای موفقیت این مدل، نیاز به مدیریت و برنامه‌ریزی دقیق وجود دارد تا زمان‌بندی بخش‌های مختلف با هم هماهنگ باشد.

۳. **ریسک افزایش تداخل:** از آنجا که تیم‌های مختلف هم‌زمان کار می‌کنند، ممکن است تغییرات در یک بخش تأثیراتی بر دیگر بخش‌ها داشته باشد.

کاربرد مدل موازی

مدل موازی بیشتر برای پروژه‌های بزرگ و پیچیده‌ای استفاده می‌شود که زمان در آن‌ها بسیار مهم است. این مدل به‌ویژه زمانی مفید است که بتوان سیستم را به‌صورت منطقی به بخش‌هایی با وابستگی کم تقسیم کرد.

نکته: برای استفاده از مدل موازی، باید آن سیستم مورد نظر قابلیت موازی شدن را داشته باشد. در بعضی از سیستم‌ها نمی‌توان از مدل موازی استفاده کرد زیرا آن سیستم‌ها قابلیت موازی شدن را ندارند.

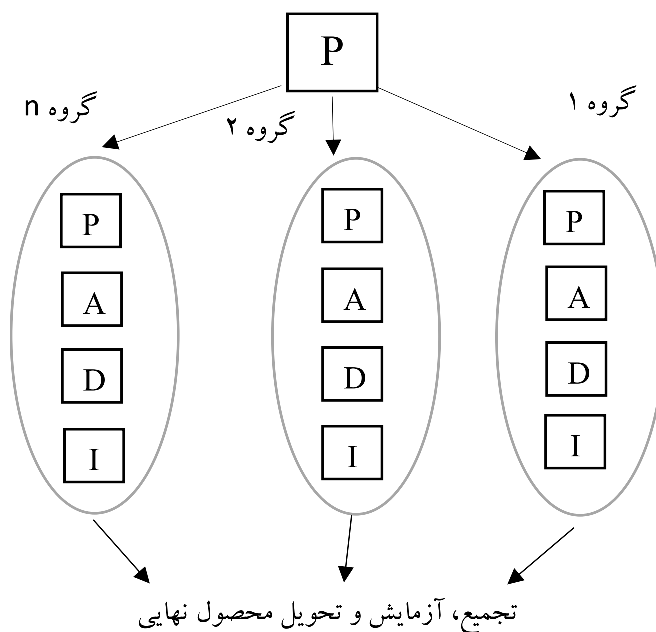


Figure ۱شکل ....

۳- مدل فازبندی شده (Phased Development Model)

یکی از مدل‌های فرآیند توسعه نرم‌افزار است که در آن نرم‌افزار به صورت مرحله‌به‌مرحله یا فازبندی‌شده توسعه پیدا می‌کند. در این مدل، به جای اینکه تمام نرم‌افزار یکجا تحویل داده شود، توسعه و تحویل سیستم در چندین فاز انجام می‌شود. هر فاز شامل بخشی از عملکرد یا ویژگی‌های سیستم است که پس از تکمیل، تحویل داده می‌شود و سپس فاز بعدی آغاز می‌شود.

ویژگی‌ها و مزایای مدل فازبندی شده

۱. **تحویل تدریجی :**این مدل اجازه می‌دهد تا سیستم به‌صورت تدریجی و در قالب نسخه‌های کوچک‌تر تحویل داده شود.

۲. **کاهش ریسک و هزینه تغییرات :**در هر فاز می‌توان بازخورد کاربران و نیازهای جدید را دریافت کرده و به‌روز رسانی کرد. این به کاهش ریسک و مدیریت تغییرات کمک می‌کند.

۳. **کاهش زمان تا بازار :**به دلیل تحویل تدریجی، می‌توان نسخه‌های قابل استفاده را زودتر به بازار ارائه داد و از مزایای تجاری آن بهره برد.

۴. **امکان بهبود مستمر :**هر فاز بهبودهای مورد نیاز و خواسته‌های جدید را در نسخه‌های بعدی لحاظ می‌کند.

معایب مدل فازبندی شده

۱. **نیاز به هماهنگی بالا :**برنامه‌ریزی و هماهنگی دقیق بین فازها ضروری است و ممکن است در مدیریت پروژه پیچیدگی ایجاد کند.

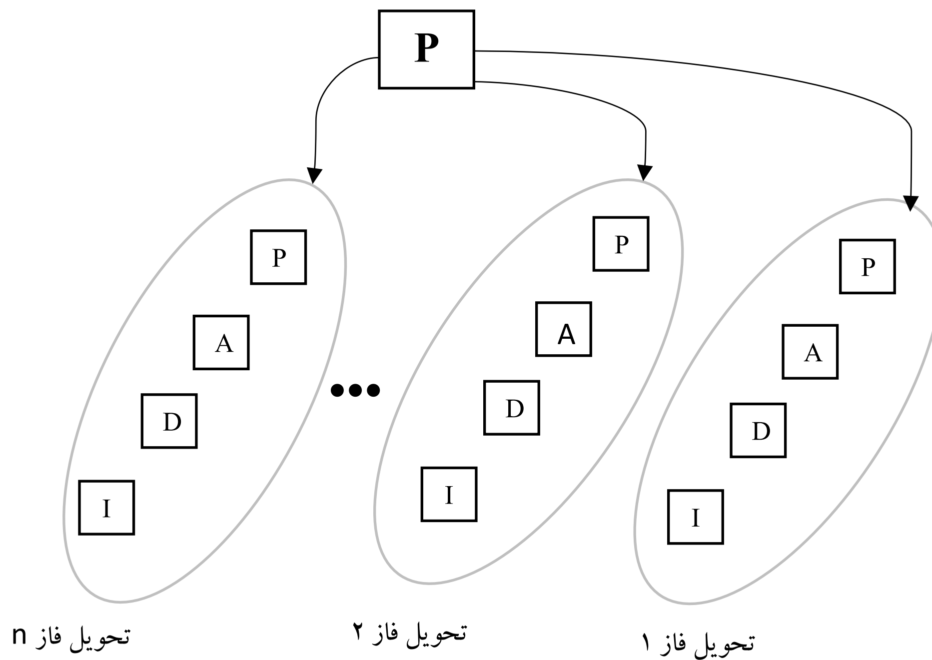
۲. **نیاز به طراحی ماژولار :**سیستم باید به گونه‌ای طراحی شود که بتواند به راحتی به فازهای کوچک‌تر تقسیم شده و قابلیت توسعه تدریجی را داشته باشد.

۳. **خطر تغییر در اهداف :**بازخوردها و نیازهای کاربران در طول توسعه ممکن است اهداف و اولویت‌های پروژه را تغییر دهد که می‌تواند باعث پیچیدگی‌های اضافی شود.

کاربرد مدل فازبندی شده

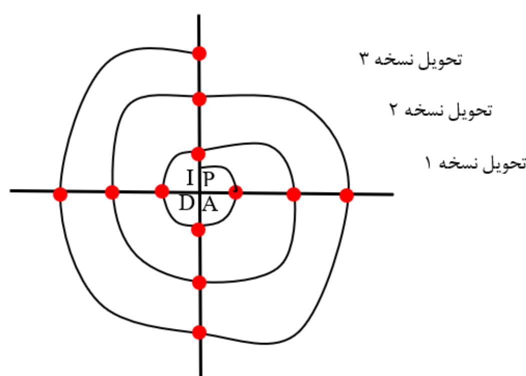
مدل فازبندی شده معمولاً در پروژه‌هایی به کار می‌رود که توسعه سریع، بازخوردهای مستمر، و تحویل تدریجی ارزش زیادی دارند. این مدل برای پروژه‌های پیچیده و سیستم‌هایی که نیاز به بروزرسانی‌های مستمر دارند مناسب است.

این مدل شباهت‌هایی به مدل‌های چابک نیز دارد، زیرا هر دو به بازخورد و تحویل تدریجی اهمیت می‌دهند، اما در مدل فازبندی شده، تمرکز بیشتری بر برنامه‌ریزی دقیق هر فاز و داشتن یک ساختار مرحله‌به‌مرحله برای توسعه است.



۴- مدل چرخشی- مارپیچ (Iterative-Spiral)

این مدل ترکیبی از ویژگی‌های *مدل آبشاری* و *مدل تکراری* است و بر شناسایی و مدیریت ریسک تمرکز دارد. در مدل مارپیچ، پروژه در قالب یک سری چرخه‌های تکراری یا مارپیچ توسعه داده می‌شود، که هر چرخه شامل مراحل کلیدی توسعه نرم‌افزار است.



این مدل را می‌توان برای مدریت نرم‌افزارهایی بکار برد که نیاز دارند با ارائه نسخه‌های جدید خطاهای موجود را برطرف کرده خدمات جدید مورد نظر را اضافه کنند. می‌توان گفت این مدل مشابه رشد انسان در دوران پس از تولد است.

۵- مدل چرخشی-افزایشی (Iterative-incremental) یا توسعه مبتنی بر نمونه‌سازی (Prototyping Model)

در این مدل، یک پروتوتایپ (نمونه اولیه) از نرم‌افزار ایجاد می‌شود تا نیازهای کاربر به درستی شناسایی شوند. پس از دریافت بازخورد، پروتوتایپ تکمیل و اصلاح می‌شود تا به محصول نهایی برسد.

در این مدل مقداری از شناخت سیستم، برای مثال درصدی از P را انجام می‌دهیم و دوباره درصدی از D، I را به ترتیب انجام می‌دهیم به این صورت نمونه‌های اولیه ساخته شده، به کارفرما تحویل می‌دهیم تا ذهنیت و شناخت بهتری از کار ایجاد شده، خطاها کمتر شود. همچنین به این شکل تلاش می‌شود یک دید و برداشت مشترک بین طرفین پروژه بوجود آید.

بخش‌های P، D، I، و A باید ۱۰۰ درصد کامل شوند تا محصول آماده تحویل شود.

	I 1	I 2	I 3	
P	٪۵	٪۱۰	...	تا ۱۰۰٪
A	٪۱۰	٪۱۰	...	
D	% ۱۰	٪۱۰	...	
I	٪۱۰	٪۱۰	...	

مزایا:

شناسایی بهتر نیازهای کاربر
کاهش ریسک‌های مرتبط با نیازهای مبهم

معایب:

زمان‌بر بودن برای ایجاد پروتوتایپ‌های متعدد
ممکن است به افزایش هزینه منجر شود

۶- مدل چابک(Agile Model)

مدل چابک (Agile) بر مبنای همکاری نزدیک، انعطاف‌پذیری و توسعه تکراری بنا شده است. این مدل از طریق چرخه‌های کوتاه‌مدت (معمولاً به نام اسپرینت) بخش‌هایی از محصول را توسعه داده و ارائه می‌دهد. متدولوژی‌های متعددی مانند اسکرام (Scrum) و کانبان (Kanban) زیرمجموعه‌های مدل چابک محسوب می‌شوند.

مزایا:

- انعطاف‌پذیری بالا
- ارائه مداوم و سریع محصول به کاربران

معایب:

- نیاز به مدیریت قوی و ماهر
- دشوار برای پروژه‌های بزرگ و پیچیده

۷- مدل توسعه سریع نرم‌افزار(RAD - Rapid Application Development)

این مدل با هدف تسریع فرآیند توسعه طراحی شده و بر اساس ساخت پروتوتایپ و توسعه سریع محصول استوار است. درRAD ، تیم‌ها با استفاده از ابزارهای آماده و تکرار سریع، محصولی اولیه را در اختیار کاربران قرار می‌دهند و بازخوردهای آن‌ها را در نسخه‌های بعدی اعمال می‌کنند.

مزایا:

- توسعه سریع و کاهش زمان
- مناسب برای پروژه‌هایی با نیازهای متغیر

معایب:

- مناسب نبودن برای پروژه‌های بزرگ
- نیاز به مشتریانی با مشارکت بالا

تفاوت مدل موازی با مدل فازبندی شده :

در مدل فازبندی شده، یک گروه تمام کارها را انجام میدهد ولی در مدل موازی کارها به صورت موازی بین چندین گروه تقسیم می‌شوند که زمان انجام پروژه سریع‌تر شود که کار هرکدام متفاوت است.

تفاوت مدل مارپیچی با مدل چرخشی :

در مدل چرخشی افزایشی، نمونه‌های اولیه‌ای که تحویل کارفرما داده می‌شود، در نسخه‌هایی از این مدل دور ریخته می‌شوند و فقط برای اینکه کارفرما یک شناخت و ذهنیتی از برداشت ما پیدا کرده، بازخورد گرفته شود تولید شده‌اند. ولی در مدل مارپیچی ، نمونه‌های تحویل شده نسخه‌های کاملی هستند، قابل استفاده می‌باشند و نصب و راه اندازی می‌شوند.

روش **Extreme Programming (XP)**

روش برنامه‌نویسی افراطی (XP) بر چهار ارزش اصلی بنا شده است :**ارتباطات، سادگی، بازخورد** و **شجاعت** . این ارزش‌ها پایه‌ای را برای توسعه‌دهندگان فراهم می‌کنند که با استفاده از آن‌ها سیستم‌های مختلفی را ایجاد می‌کنند.

اصول کلیدیXP

۱. **بازخورد سریع به کاربران نهایی**: توسعه‌دهندگان باید به‌طور مستمر بازخورد سریع را ارائه دهند.

۲. **پیروی از اصل سادگی**: توسعه‌دهندگان باید از سادگی در طراحی و کدنویسی پیروی کنند.

۳. **پذیرش تغییرات**: تغییرات جزئی برای رشد سیستم را بپذیرند و حتی آن‌ها را با آغوش باز بپذیرند.

٤. **تمرکز بر کیفیت**: کیفیت باید همیشه در اولویت باشد.

اصول عملیاتی:XP

۱. **تست مستمر**: کد هر روز تست می‌شود و در محیط تست یکپارچه قرار می‌گیرد.

۲. **کدنویسی دو نفری**: دو توسعه‌دهنده به‌صورت همزمان کد را می‌نویسند.

۳. **تعامل نزدیک با کاربران نهایی**: برای دریافت بازخورد و رفع مشکلات به‌صورت سریع.

فرآیندXP

۱. **شروع با داستان‌های کاربری**: توصیف نیازهای سیستم.

۲. **کدنویسی و تست ماژول‌های کوچک و ساده**: برای برآورده کردن نیازها.

۳. **دسترسی کاربران**: برای رفع سوالات و مشکلات به‌محض بروز آن‌ها.

مزایایXP

- بهبود ارتباطات بین برنامه‌نویسان و ذینفعان.
- توسعه تدریجی و تکاملی سیستم.
- مسئولیت مشترک برای هر مؤلفه نرم‌افزار.

- افزایش کیفیت محصول نهایی در هر تکرار.

محدودیت‌هایXP

- مناسب برای پروژه‌های کوچک با تیم‌های باانگیزه و هماهنگ.
- نیاز به انضباط زیاد.
- عدم مناسب بودن برای برنامه‌های بزرگ و حیاتی.
- کمبود مستندات تحلیل و طراحی.
- نیاز به ورودی‌های کاربر در محل.

اسکرام – Scrum

اسکرام اصطلاحی است که برای طرفداران راگبی بسیار آشناست. در راگبی، اسکرام برای شروع مجدد بازی استفاده می‌شود. به طور خلاصه، خالقان روش اسکرام معتقدند که بدون توجه به میزان برنامه‌ریزی، به محض آغاز توسعه نرم‌افزار، هرج و مرج ایجاد می‌شود و برنامه‌ها بی‌اثر می‌شوند. بهترین کاری که می‌توان انجام داد این است که به جایی که توپ نمادین راگبی خارج می‌شود واکنش نشان دهیم و سپس تا اسکرام بعدی با توپ پیش برویم. در مورد روش اسکرام، یک sprint (چرخه کاری) سی روز کاری طول می‌کشد. در پایان sprint ، یک سیستم به مشتری تحویل داده می‌شود.

در میان تمام رویکردهای توسعه سیستم، اسکرام در ظاهر پر هرج و مرج‌ترین است. برای کنترل برخی از این هرج و مرج ذاتی، توسعه اسکرام بر چندین عمل کلیدی تمرکز دارد. تیم‌ها خودسازماندهی شده و خودگردان هستند. برخلاف سایر رویکردها، تیم‌های اسکرام رهبر تیم مشخصی ندارند. در عوض، تیم‌ها به شیوه‌ای همزیستی سازماندهی می‌شوند و اهداف خود را برای هر sprint (چرخه) تعیین می‌کنند. هنگامی که یک sprint آغاز شد، تیم‌های اسکرام هیچ نیاز اضافی دیگری را در نظر نمی‌گیرند. هر نیاز جدیدی که شناسایی شود، به فهرست نیازهای باقی‌مانده برای بررسی اضافه می‌شود. در ابتدای هر روز کاری، یک جلسه اسکرام برگزار می‌شود. در پایان هر اسپریت، تیم نرم‌افزار را به مشتری نشان می‌دهد و بر اساس نتایج اسپریت، یک برنامه جدید برای اسپریت بعدی آغاز می‌شود.

جلسات اسکرام یکی از جنبه‌های جالب فرآیند توسعه اسکرام هستند. اعضای تیم در جلسات حضور دارند، اما هر کسی می‌تواند در آن شرکت کند. با این حال، با چند استثنا، فقط اعضای تیم اجازه صحبت دارند. یک استثنای مهم این است که مدیریت بازخوردی در مورد مرتبط بودن کار تیم با اهداف تجاری ارائه دهد. در این جلسه، همه اعضای تیم در دایره‌ای می‌ایستند و گزارش می‌دهند که روز گذشته چه کارهایی انجام داده‌اند، برنامه کاری خود برای امروز را اعلام می‌کنند و هر چیزی که پیشرفت کار روز گذشته را متوقف کرده است توضیح می‌دهند. برای پیشرفت مستمر، هر مانعی که شناسایی شود، در عرض یک ساعت حل می‌شود. از دیدگاه اسکرام، در این مرحله از توسعه "تصمیم بد" در مورد یک مانع بهتر است تا اینکه تصمیمی گرفته نشود. از آنجا که جلسات هر روز برگزار می‌شوند، یک تصمیم بد به راحتی قابل اصلاح است. لَرمان[1] پیشنهاد می‌کند که هر عضو تیم باید هر نیاز جدیدی که در طول اسپریت شناسایی شده و هر اطلاعاتی که یاد گرفته و می‌تواند برای سایر اعضای تیم مفید باشد را گزارش دهد.

یکی از انتقادات اصلی اسکرام، مانند سایر روش‌های چابک، این است که آیا اسکرام می‌تواند برای توسعه سیستم‌های بسیار بزرگ و حیاتی مفید و موثر باشد یا خیر. اندازه معمول تیم اسکرام بیش از هفت نفر نیست. تنها اصل سازمانی که توسط طرفداران اسکرام برای پاسخ به این انتقاد مطرح شده، سازماندهی یک «اسکرام از اسکرام‌ها» است. هر تیم هر روز جلسه دارد و پس از آن نماینده‌ای (نه رهبر) از هر تیم در جلسه اسکرام شرکت می‌کند. این روند تا تعیین پیشرفت کل سیستم ادامه می‌یابد. با توجه به تعداد تیم‌های درگیر، این رویکرد برای مدیریت یک پروژه بزرگ ممکن است مشکوک باشد. با این حال، مانند XP و سایر رویکردهای چابک، بسیاری از ایده‌ها و تکنیک‌های مرتبط با توسعه اسکرام در توسعه سیستم‌های شیءگرا مفید هستند، مانند تمرکز بر جلسات اسکرام، رویکرد تکاملی و افزایشی در شناسایی نیازها و رویکرد تکراری و تدریجی در توسعه سیستم.

انتخاب روش توسعه مناسب

با توجه به تعداد زیادی از روش‌های موجود، اولین چالشی که تحلیل‌گران با آن روبرو هستند، انتخاب روشی است که باید استفاده کنند. انتخاب یک روش ساده نیست، زیرا هیچ روشی همیشه بهترین نیست. (اگر چنین بود، ما آن را در همه جا استفاده می‌کردیم!) بسیاری از سازمان‌ها استانداردها و سیاست‌هایی برای راهنمایی در انتخاب روش دارند. شما متوجه خواهید شد که سازمان‌ها از داشتن یک روش "مورد تأیید" تا داشتن چند گزینه برای روش‌ها تا نداشتن هیچ سیاست رسمی، متفاوت هستند.

[1] C. Larman, *Agile & Iterative Development: A Manager's Guide* (Boston: Addison-Wesley, 2004).

نقش‌ها و مهارت‌های معمول تحلیل‌گران سیستم‌ها

با توجه به مراحل و گام‌های مختلفی که در طول چرخه حیات توسعه سیستم (SDLC) انجام می‌شود، مشخص است که تیم پروژه به مجموعه متنوعی از مهارت‌ها نیاز دارد. اعضای پروژه به عنوان عوامل تغییر عمل می‌کنند که راه‌هایی را برای بهبود یک سازمان شناسایی می‌کنند، یک سیستم اطلاعاتی برای پشتیبانی از آن ایجاد می‌کنند و دیگران را برای استفاده از سیستم آموزش داده و انگیزه می‌بخشند. درک اینکه چه چیزی باید تغییر کند و چگونه تغییر یابد—و متقاعد کردن دیگران به نیاز به تغییر—نیازمند مجموعه گسترده‌ای از مهارت‌ها است. این مهارت‌ها را می‌توان به شش دسته اصلی تقسیم کرد: *فنی*، *کسب‌وکار*، *تحلیلی*، *بین‌فردی*، *مدیریت* و *اخلاقی*. تحلیل‌گران باید مهارت‌های فنی لازم را برای درک محیط فنی موجود سازمان، فناوری‌هایی که سیستم جدید را تشکیل می‌دهند و نحوه ترکیب هر دو در یک راه‌حل فنی یکپارچه داشته باشند. مهارت‌های کسب‌وکاری نیز برای درک نحوه استفاده از فناوری اطلاعات در موقعیت‌های تجاری و اطمینان از اینکه فناوری اطلاعات ارزش واقعی برای کسب‌وکار ایجاد می‌کند، مورد نیاز است. تحلیل‌گران همواره در هر دو سطح پروژه و سازمان به عنوان حل‌کننده‌های مشکلات عمل کرده و مهارت‌های تحلیلی خود را به‌طور مرتب به کار می‌گیرند.

تحلیل‌گران اغلب نیاز دارند که به‌صورت یک‌به‌یک با کاربران و مدیران کسب‌وکار (که معمولاً تجربه کمی در زمینه فناوری دارند) و با برنامه‌نویسان (که معمولاً تخصص فنی بیشتری نسبت به تحلیل‌گر دارند) ارتباط برقرار کنند. آن‌ها باید قادر به ارائه در گروه‌های بزرگ و کوچک باشند و گزارش‌ها بنویسند. علاوه بر اینکه باید توانایی‌های قوی بین‌فردی داشته باشند، باید بتوانند افرادی که با آن‌ها کار می‌کنند را مدیریت کنند و همچنین فشار و ریسک‌های مرتبط با شرایط نامشخص را کنترل نمایند.

در نهایت، تحلیل‌گران باید به‌طور منصفانه، صادقانه و اخلاقی با سایر اعضای تیم پروژه، مدیران و کاربران سیستم برخورد کنند. آن‌ها اغلب با اطلاعات محرمانه یا اطلاعاتی که در صورت افشا ممکن است آسیب برساند (مانند نارضایتی بین کارکنان) مواجه هستند؛ بنابراین، حفظ اعتماد و اطمینان با همه افراد ضروری است.

علاوه بر این شش مجموعه مهارتی کلی، تحلیل‌گران نیاز به مهارت‌های خاص بسیاری دارند که با نقش‌های آن‌ها در پروژه مرتبط است. در روزهای اولیه توسعه سیستم‌ها، بیشتر سازمان‌ها انتظار داشتند که یک فرد، یعنی تحلیل‌گر، تمام مهارت‌های خاص مورد نیاز برای انجام یک پروژه توسعه سیستم را داشته باشد. برخی از سازمان‌های کوچک هنوز انتظار دارند که یک نفر چندین نقش را ایفا کند، اما با پیچیده‌تر شدن سازمان‌ها و فناوری‌ها، بیشتر سازمان‌های بزرگ اکنون تیم‌های پروژه‌ای تشکیل می‌دهند که شامل چندین فرد با مسئولیت‌های مشخص است. سازمان‌های مختلف نقش‌ها را به شیوه‌های مختلفی تقسیم‌بندی می‌کنند. اکثر تیم‌های فناوری اطلاعات شامل

افراد دیگری از جمله برنامه‌نویسان که برنامه‌های سیستم را می‌نویسند و نویسندگان فنی که صفحات کمک و مستندات دیگر (مانند راهنماهای کاربران و راهنماهای سیستم) را آماده می‌کنند، هستند.

تحلیل‌گر کسب‌وکار

یک تحلیل‌گر کسب‌وکار بر مسائل تجاری مرتبط با سیستم تمرکز دارد. این مسائل شامل شناسایی ارزش تجاری است که سیستم ایجاد می‌کند، ارائه ایده‌ها و پیشنهادات برای بهبود فرآیندهای کسب‌وکار و طراحی فرآیندها و سیاست‌های جدید همراه با تحلیل‌گر سیستم است. این فرد به احتمال زیاد تجربه کسب‌وکار و نوعی آموزش حرفه‌ای دارد. او نماینده منافع حامی پروژه و کاربران نهایی سیستم است و در فازهای برنامه‌ریزی و طراحی کمک می‌کند، اما عمدتاً در فاز تحلیل فعالیت دارد.

تحلیل‌گر سیستم‌ها

یک تحلیل‌گر سیستم‌ها بر مسائل فناوری اطلاعات مرتبط با سیستم تمرکز دارد. این شخص ایده‌ها و پیشنهاداتی برای بهبود فرآیندهای کسب‌وکار از طریق فناوری اطلاعات ارائه می‌دهد، با کمک تحلیل‌گر کسب‌وکار فرآیندهای جدید کسب‌وکار را طراحی می‌کند، سیستم اطلاعاتی جدید را طراحی کرده و اطمینان حاصل می‌کند که تمامی استانداردهای فناوری اطلاعات رعایت می‌شوند. احتمالاً یک تحلیل‌گر سیستم‌ها آموزش و تجربه قابل‌توجهی در زمینه تحلیل و طراحی، برنامه‌نویسی و حتی حوزه‌های کسب‌وکار دارد. او نماینده منافع بخش فناوری اطلاعات است و در تمام پروژه کار می‌کند، اما شاید کمتر در فاز پیاده‌سازی درگیر باشد.

تحلیل‌گر زیرساخت

تحلیل‌گر زیرساخت بر مسائل فنی مرتبط با نحوه تعامل سیستم با زیرساخت‌های فنی سازمان (مانند سخت‌افزار، نرم‌افزار، شبکه‌ها و پایگاه‌های داده) تمرکز دارد. وظایف تحلیل‌گر زیرساخت شامل اطمینان از انطباق سیستم اطلاعاتی جدید با استانداردهای سازمان و شناسایی تغییرات زیرساختی مورد نیاز برای پشتیبانی از سیستم است. این فرد احتمالاً تجربه و آموزش قابل‌توجهی در زمینه شبکه، مدیریت پایگاه داده و محصولات سخت‌افزاری و نرم‌افزاری دارد و نماینده منافع سازمان و گروه فناوری اطلاعات است که در نهایت باید سیستم جدید را پس از نصب مدیریت و پشتیبانی کند. تحلیل‌گر زیرساخت در طول پروژه کار می‌کند، اما شاید کمتر در فازهای برنامه‌ریزی و تحلیل درگیر باشد.

تحلیل‌گر مدیریت تغییرات

تحلیل‌گر مدیریت تغییرات بر مسائل انسانی و مدیریتی مرتبط با نصب سیستم تمرکز دارد. نقش‌های این فرد شامل اطمینان از مستندات و پشتیبانی کافی برای کاربران، ارائه آموزش به کاربران درباره سیستم جدید و توسعه استراتژی‌هایی برای غلبه بر مقاومت در برابر تغییر است. این فرد باید آموزش و تجربه قابل‌توجهی در زمینه رفتار سازمانی و به‌ویژه مدیریت تغییرات داشته باشد. او نماینده منافع حامی پروژه و کاربران است که سیستم برای آن‌ها طراحی می‌شود. یک تحلیل‌گر مدیریت تغییرات عمدتاً در فاز پیاده‌سازی فعالیت دارد، اما از فازهای تحلیل و طراحی زمینه را برای تغییر آماده می‌کند.

مدیر پروژه

مدیر پروژه مسئولیت اطمینان از اتمام پروژه در زمان مقرر و با بودجه معین و تحویل سیستم با تمامی مزایای مورد نظر حامی پروژه را بر عهده دارد. نقش مدیر پروژه شامل مدیریت اعضای تیم، توسعه برنامه پروژه، تخصیص منابع و ارتباط اولیه با افرادی است که در خارج از تیم سوالاتی درباره پروژه دارند. این فرد باید تجربه زیادی در مدیریت پروژه داشته، سال‌ها به‌عنوان تحلیل‌گر سیستم‌ها فعالیت کرده باشد. او نماینده منافع بخش فناوری اطلاعات و حامی پروژه است و در تمامی فازهای پروژه به‌شدت فعالیت می‌کند.

انتخاب و بکارگیری ابزارهای توسعه نرم‌افزار (Software Development Tools)

انتخاب و بکارگیری ابزارهای توسعه نرم‌افزار یکی از مراحل مهم و حیاتی در فرآیند تحلیل و طراحی نرم‌افزار است. این ابزارها به توسعه‌دهندگان کمک می‌کنند تا با کیفیت بالاتر، کارایی بیشتر و زمان کمتر نرم‌افزارها را توسعه دهند. در ادامه به توضیح برخی از این ابزارها و اهمیت انتخاب آن‌ها می‌پردازیم.

۱. ابزارهای مدیریت پروژه

ابزارهایی مانندJira ، Trelloو Asana برای مدیریت پروژه، پیگیری وظایف و همکاری تیمی استفاده می‌شوند . این ابزارها به توسعه‌دهندگان کمک می‌کنند تا وظایف را بهتر سازماندهی کنند، پیشرفت پروژه را پیگیری کنند و ارتباطات تیمی را بهبود بخشند.

۲. ابزارهای کنترل نسخه

ابزارهایی مانندGit ، GitHub، GitLabو Bitbucket برای مدیریت نسخه‌ها و کد منبع استفاده می‌شوند. این ابزارها امکان پیگیری تغییرات کد، همکاری موثر بین تیم‌ها و بازگردانی به نسخه‌های قبلی در صورت نیاز را فراهم می‌کنند.

۳. بزارهای طراحی و مدل‌سازی

ابزارهایی مانندMicrosoft Visio ، UMLetو Enterprise Architect برای طراحی و مدل‌سازی سیستم‌ها استفاده می‌شوند. این ابزارها به توسعه‌دهندگان کمک می‌کنند تا ساختار و رفتار سیستم‌ها را به‌صورت بصری و دقیق مدل‌سازی کنند و بهبود ارتباطات و مستندسازی را فراهم آورند.

۴. محیط‌های توسعه یکپارچه(IDE)

ابزارهایی مانندVisual Studio ، IntelliJ IDEA، Eclipseو PyCharm برای توسعه کد و پیاده‌سازی استفاده می‌شوند. IDEها امکاناتی مانند تکمیل خودکار کد، اشکال‌زدایی، تست و یکپارچه‌سازی ابزارهای دیگر را فراهم می‌کنند و توسعه‌دهندگان را در فرآیند کدنویسی یاری می‌دهند.

۵. ابزارهای تست و یکپارچه‌سازی

ابزارهایی مانندSelenium ، JUnit، Jenkinsو Travis CI برای تست و یکپارچه‌سازی مداوم استفاده می‌شوند. این ابزارها امکان تست مداوم کد و اطمینان از صحت عملکرد سیستم را فراهم می‌کنند و به شناسایی و رفع باگ‌ها کمک می‌کنند.

۶.ابزارهای مدیریت پایگاه داده

ابزارهایی مانندMySQL Workbench ، pgAdminو SQL Server Management Studio برای مدیریت و طراحی پایگاه داده‌ها استفاده می‌شوند. این ابزارها به توسعه‌دهندگان کمک می‌کنند تا پایگاه داده‌ها را طراحی، پیاده‌سازی و مدیریت کنند و ارتباطات داده‌ای سیستم را بهبود بخشند.

۷. ابزارهای نظارت و عیب‌یابی

ابزارهایی مانندNew Relic ، Datadogو Splunk برای نظارت بر عملکرد سیستم و عیب‌یابی استفاده می‌شوند. این ابزارها به توسعه‌دهندگان امکان می‌دهند تا عملکرد سیستم را نظارت کنند، مشکلات عملکردی را شناسایی و برطرف کنند و به بهبود کارایی سیستم کمک کنند.

زبان‌های مدلسازی(Modeling Languages)

زبان‌های مدلسازی ابزارهایی هستند که برای **طراحی، مستندسازی، تحلیل، و نمایش بصری سیستم‌ها** در زمینه‌های مختلف، به‌ویژه **مهندسی نرم‌افزار، سیستم‌های اطلاعاتی و معماری سیستم‌ها**، به کار می‌روند. این زبان‌ها با استفاده از نمادها و قواعد خاص به معماران و تحلیل‌گران کمک می‌کنند تا جنبه‌های مختلف یک سیستم را بهتر درک کرده و آن را به دیگران منتقل کنند.

یک زبان مدل‌سازی می‌تواند بصورت گرافیکی یا متنی باشد.

- زبان‌های مدل‌سازی گرافیکی از روش نمودار با نمادهای نام‌گذاری شده استفاده می‌کنند که بیانگر مفاهیم و خطوطی است که نمادها را به هم متصل می‌کند و روابط را نشان می‌دهد و همچنین علامت‌های مختلف گرافیکی دیگر را برای نمایش محدودیت‌ها نشان می‌دهد.
- زبان‌های مدل‌سازی متنی ممکن است از کلمان کلیدی استاندارد شده همراه با پارامترها یا اصطلاحات و عبارات زبان طبیعی برای ایجاد عبارات قابل تفسیر کامپیوتری استفاده کنند.

انواع زبان های مدل سازی

دسته‌بندی زبان‌های مدلسازی

۱. **زبان‌های مدلسازی ساختاری**(Structural Modeling Languages) :

این زبان‌ها ساختار یک سیستم را نمایش می‌دهند، مانند اجزا، موجودیت‌ها، و روابط بین آن‌ها.

مثال‌ها:

- **UML (Unified Modeling Language):** برای نمایش ساختار و رفتار سیستم‌های نرم‌افزاری.
- **ERD (Entity-Relationship Diagram):** برای مدل‌سازی پایگاه‌های داده.
- **نمودار کلاسی** (Class Diagram) برای نمایش کلاس‌ها و روابط بین آن‌ها در نرم‌افزار.

۲. **زبان‌های مدلسازی رفتاری**(Behavioral Modeling Languages) :

این زبان‌ها رفتار و فرآیندهای دینامیک یک سیستم را نمایش می‌دهند.

مثال‌ها:

- **نمودار ماشین حالت**(State Machine Diagram) برای نمایش حالات مختلف یک سیستم و انتقالات بین آن‌ها.
- **نمودار فعالیت** (Activity Diagram)**:** برای نمایش جریان فعالیت‌ها.
- **نمودار توالی** (Sequence Diagram): برای نشان دادن ترتیب تعاملات بین اشیا.

۳. **زبان‌های مدلسازی معماری** (Architectural Modeling Languages)

این زبان‌ها برای نمایش **معماری سطح بالا**ی یک سیستم استفاده می‌شوند.

مثال‌ها:

- **ArchiMate** برای مدلسازی معماری سازمانی.
- (Systems Modeling Language) **SysML**: برای مدلسازی سیستم‌های پیچیده.
- **BPMN (**Business Process Model and Notation**):** برای مدلسازی فرآیندهای کسب‌وکار.

٤. **زبان‌های مدلسازی دامنه خاص (**Domain-Specific Modeling Languages - DSL) **:**

زبان‌هایی که برای یک دامنه خاص طراحی شده‌اند.

مثال‌ها:

- MATLAB Simulink : برای مدلسازی سیستم‌های کنترلی.
- **Verilogیا VHDL** : برای مدلسازی سخت‌افزار.

UML یا زبان مدل‌سازی یکپارچه (Unified Modeling Language)

UML یک زبان مدل‌سازی استاندارد است که برای مشخص کردن، تجسم، ساخت و مستندسازی اجزاء سیستم‌های نرم‌افزاری استفاده می‌شود. UML به توسعه‌دهندگان و تحلیل‌گران کمک می‌کند تا ساختار و رفتار سیستم‌های نرم‌افزاری را به صورت گرافیکی نمایش دهند و ارتباطات بین اجزاء مختلف سیستم را درک کنند.

اجزای UML :

UML شامل چندین نوع دیاگرام است که هر یک جنبه‌های مختلفی از سیستم را مدل‌سازی می‌کنند. در ادامه به مهم‌ترین دیاگرام‌های UML و کاربرد آن‌ها اشاره می‌شود:

۱. دیاگرام‌های ساختاری(Structural Diagrams)

- **دیاگرام کلاس‌ها (Class Diagram)** : نمایش اجزاء کلاس‌ها، روابط بین آن‌ها و ساختار کلاس‌ها و اشیاء.
- **دیاگرام اجزاء(Component Diagram)** : نمایش اجزاء فیزیکی سیستم و نحوه تعامل آن‌ها.
- **دیاگرام بسته‌ها(Package Diagram)** : نمایش گروه‌بندی و سازماندهی بسته‌ها و ماژول‌ها.
- **دیاگرام اشیاء (Object Diagram)** نمایش نمونه‌های کلاس‌ها در یک لحظه خاص.
- **دیاگرام استقرار (Deployment Diagram)** : نمایش نحوه استقرار اجزاء نرم‌افزاری بر روی سخت‌افزار.

۲. دیاگرام‌های رفتاری(Behavioral Diagrams)

- **دیاگرام موارد استفاده (Use Case Diagram)** نمایش تعاملات بین کاربران و سیستم و عملکردهای سیستم.
- **دیاگرام توالی (Sequence Diagram)** نمایش ترتیب پیام‌ها و تعاملات بین اجزاء سیستم در زمان.
- **دیاگرام فعالیت (Activity Diagram)** نمایش جریان کار و فرآیندها در سیستم.
- **دیاگرام وضعیت (State Diagram)** نمایش وضعیت‌های مختلف یک شیء و تغییرات آن‌ها.
- **دیاگرام تعامل (Communication Diagram)** نمایش تعاملات بین اجزاء سیستم از دیدگاه ساختاری.
- **دیاگرام زمان‌بندی (Timing Diagram)** نمایش زمان‌بندی تعاملات و وضعیت‌ها در طول زمان.

۳. دیاگرام‌های ترکیبی(Composite Diagrams)

- **دیاگرام ترکیب ساختاری (Composite Structure Diagram)** نمایش ساختار داخلی کلاس‌ها و اجزاء و نحوه ترکیب آن‌ها.

- **دیاگرام نمودار نموداری** (Interaction Overview Diagram) ترکیبی از دیاگرام‌های فعالیت و توالی برای نمایش جریان کلی سیستم.

مزایای استفاده ازUML

۱. **بصری‌سازی :**کمک به بصری‌سازی ساختار و رفتار سیستم به صورت گرافیکی.

۲. **مستندسازی :**ارائه مستندات جامع و قابل فهم از سیستم.

۳. **ارتباط :**بهبود ارتباطات بین اعضای تیم توسعه و تحلیل‌گران.

۴. **پشتیبانی از فرآیند توسعه :**پشتیبانی از فرآیندهای تحلیل، طراحی و پیاده‌سازی سیستم.

انواع دیدگاه ها در تحلیل و طراحی سیستم ها:

- **کدنویسی و رفع خطا Code & Fix**
- **داده محور Data Centered**
- **Functional / Procedural**
- **شی گرا Object Oriented**

کدنویسی و رفع خطا (Code and Fix) به یک روش یا سبک توسعه نرم‌افزار ابتدایی و غیررسمی اشاره دارد که بیشتر در اوایل تاریخچه توسعه نرم‌افزار و در پروژه‌های کوچک یا توسط برنامه‌نویسان تازه‌کار استفاده می‌شد. در این روش، برنامه‌نویسان به‌جای دنبال کردن یک فرآیند ساختاریافته، مستقیماً شروع به نوشتن کد می‌کنند و سپس مشکلات یا خطاهای آن را در مراحل بعدی شناسایی و برطرف می‌کنند. این روش از لحاظ ساختاری بسیار ساده است اما معایب بسیاری دارد.

ویژگی‌های دوران کدنویسی و رفع خطا (Code and Fix)

۱. **شروع بدون برنامه‌ریزی:**

- برنامه‌نویسی بدون طراحی اولیه یا برنامه‌ریزی جامع.
- توسعه با تمرکز بر نوشتن کد بدون تحلیل کافی از نیازها.

۲. **تمرکز بر کد اولیه:**

- برنامه‌نویس مستقیماً کدنویسی را آغاز می‌کند و سپس مشکلات کد را رفع می‌کند.

۳. **رفع مشکلات در حین یا بعد از توسعه:**

- اشکالات و تغییرات در مراحل بعدی بررسی و رفع می‌شوند.

٤. **عدم مستندات رسمی:**

- مستندات برنامه به ندرت ایجاد می‌شوند یا اگر ایجاد شوند، بعد از اتمام پروژه اضافه می‌شوند.

رویکرد ***مبتنی بر داده یا Data-Centered***

در تحلیل، طراحی، و برنامه‌نویسی به معنای متمرکز شدن بر داده‌ها و ساختارهای مرتبط با آن‌ها در فرایند توسعه سیستم‌ها است. این رویکرد با تمرکز بر چگونگی ذخیره‌سازی، مدیریت، و جریان داده‌ها در سیستم، یکی از اساسی‌ترین جنبه‌ها در توسعه نرم‌افزار و زیرساخت‌های فناوری اطلاعات است.

در این نگاه، تحلیل و طراحی سیستم‌ها حول محور **داده‌ها** و تعاملات آن‌ها انجام می‌شود، نه صرفاً فرایندها یا عملکردها. هدف اصلی این رویکرد، اطمینان از طراحی بهینه پایگاه داده و تضمین کیفیت و سازگاری داده‌ها در سیستم است.

۱. **تحلیل:Data-Centered**

در این مرحله، تحلیل‌گر سیستم بیشتر به **شناسایی داده‌ها**، روابط بین داده‌ها، و چگونگی جریان آن‌ها در سیستم توجه می‌کند.

ابزارهای رایج:

- **مدل‌های داده‌ای منطقی و فیزیکی** (Logical & Physical Data Models)
- **نمودار موجودیت-رابطه**(ERD) برای طراحی ساختار پایگاه داده.

۲. **طراحی**Data-Centered :

طراحی سیستم به گونه‌ای انجام می‌شود که ساختار پایگاه داده بهینه باشد و عملیات ذخیره‌سازی و بازیابی داده‌ها کارآمد صورت گیرد.

اهداف اصلی طراحی:

- یکپارچگی داده‌ها
- دسترسی سریع
- کاهش افزونگی داده‌ها

۳. **برنامه‌نویسیData-Centered :**

در برنامه‌نویسی، دستورات و عملکردها اغلب بر اساس نحوه دسترسی و استفاده از داده‌ها نوشته می‌شوند.

استفاده از زبان‌هایی مانند SQL برای مدیریت داده‌ها و تعامل با پایگاه داده.

پارادایم Functional-Structural در تحلیل و طراحی سیستم‌ها یک رویکرد ترکیبی است که عناصر *تحلیل تابع‌گرا (Functional Analysis)* و *تحلیل ساخت‌گرا (Structural Analysis)* را با هم ادغام می‌کند. این پارادایم تلاش می‌کند تا از مزایای هر دو روش بهره ببرد و یک دیدگاه جامع برای طراحی سیستم‌ها ارائه دهد. این پارادایم شامل دو بخش اصلی است:

۱. **تحلیل و طراحی تابع‌گرا (Functional)** :

- تمرکز بر شناسایی وظایف (Functions) و رفتارهایی که سیستم باید انجام دهد.
- تعریف ورودی‌ها و خروجی‌های سیستم.
- تعیین نحوه جریان داده‌ها در سیستم.

۲. **تحلیل و طراحی ساخت‌گرا** (Structural) :

- تمرکز بر ساختار فیزیکی یا منطقی سیستم و اجزای آن.
- شناسایی ماژول‌ها، اجزای اصلی، و روابط میان آن‌ها.
- تعریف سلسله‌مراتب یا ارتباطات درون سیستم.

رویکرد شی‌گرا (Object-Oriented)

رویکرد شی‌گرا در تحلیل و طراحی یکی از مهم‌ترین و پرکاربردترین روش‌ها در مهندسی نرم‌افزار است. این رویکرد بر اساس مفاهیم **اشیا (**Objects**)** و **کلاس‌ها (**Classes**)** بنا شده است و به دلیل انعطاف‌پذیری و قابلیت باز استفاده (Reusability) محبوبیت زیادی دارد.

پارادایم شی‌گرایی یکی از رویکردهای اصلی در **برنامه‌نویسی** و **تحلیل و طراحی نرم‌افزار** است که سیستم‌ها را بر اساس اشیا (Objects) مدل‌سازی می‌کند. این پارادایم بر اساس **تفکر دنیای واقعی** شکل گرفته و هدف آن ساده‌سازی فرآیند توسعه نرم‌افزار از طریق تقسیم سیستم به اجزای کوچک‌تر (اشیا) است که هر کدام مسئولیت‌ها و رفتارهای خاص خود را دارند. در این رویکرد، سیستم به مجموعه‌ای از اشیا تقسیم می‌شود که هر شی نشان‌دهنده یک موجودیت واقعی یا مفهومی در سیستم است. این اشیا می‌توانند داده‌ها (ویژگی‌ها) و رفتارها (عملیات یا متدها) را با هم ترکیب کنند.

اصول شئ‌گرایی

پارادایم شئ‌گرایی در هرجایی که از اجزا مستقل از هم تشکیل شده، می‌تواند شکل بگیرد.

چهار اصل اساسی شئ گرایی

۱- **انتزاع یا تجرید** (Abstraction)

درمورد یک موضوع کلی صحبت می‌کند ، بدون اینکه به جزئیات اشاره کند. (جزییات را نادیده می‌گیریم.)

خصوصیات نگاه انتزاعی:

۱- ایجاد نگاه کلی به مسئله (موضوع)   نگاه ما را به موضوع ساده تر میکند.

۲- انتخاب جزئیات مهم و کنار گذاشتن یا پنهان کردن بقیه جزئیات (توجه به بخش های مهم تر).

۳- نگاه ساده به موضوع برای قابل فهم شدن آن.

۲- **کپسوله سازی** (Encapsulation) **:** مخفی سازی یا پنهان سازی / بسته بندی / محافظت

در برنامه‌نویسی شیءگرا(OOP) ، مفهوم **Encapsulation** یا **پنهان‌سازی** یکی از اصول اساسی است که به سازمان‌دهی و مدیریت کد کمک می‌کند. پنهان‌سازی به معنای مخفی کردن جزئیات داخلی یک شیء و نشان دادن فقط موارد ضروری به کاربران است. این اصل به حفاظت از داده‌ها و جلوگیری از دسترسی مستقیم و تغییر ناخواسته داده‌های داخلی کمک می‌کند.

ویژگی‌های Encapsulation

1. **پنهان‌سازی داده‌ها**

- **خصوصی‌سازی** (Private) : متغیرها و داده‌های داخلی یک کلاس به صورت خصوصی تعریف می‌شوند تا از دسترسی مستقیم از خارج از کلاس جلوگیری شود.
- **واسط‌های عمومی** (Public Interfaces) : متدهای عمومی برای دسترسی به داده‌های داخلی و تغییر آن‌ها ایجاد می‌شوند.
- **متدهای دسترسی Getter و Setter**
- **Getter** : متدهایی که برای خواندن مقادیر متغیرهای خصوصی استفاده می‌شوند.
- **Setter :** متدهایی که برای تغییر مقادیر متغیرهای خصوصی استفاده می‌شوند.

با کپسوله کردن اطلاعات ، دسترسی به آنها را محدود میکنیم و جلوی بسیاری از خطاها را می گیریم.

اجزا را میتوان به عنوان یک کپسول در نظر بگیریم و اجزا در کپسول ها بسته بندی می‌شوند و رابطه‌های تعریف شده برقرار ارتباط بین اجزا را فراهم می‌کنند. هرکلاسی باید بتواند اجزاء و رابطه‌های داخلی کلاس (شناسه و رفتارها) را از دید کلاس های دیگر مخفی کند.

این اصول به عنوان یک قالب الزامی نیستند و ما به عنوان یک ایده در تحلیل و طراحی از آنها استفاده می کنیم.

در برنامه‌نویسی ساختیافته ایجاد و بکارگیری متغیرهای سراسری با مفهوم کپسوله‌سازی در تضاد می‌باشد.

۳– **پیمانه‌بندی** (Modularity)**:**

پیمانه‌بندی در تحلیل و طراحی شیءگرا به معنای تقسیم یک سیستم بزرگ به واحدهای کوچکتر و مستقل به نام ماژول‌ها است. هر ماژول وظیفه یا مجموعه‌ای از وظایف مشخص و خاصی را انجام می‌دهد و به صورت مستقل قابل تغییر و نگهداری است. ماژولاریتی به طراحان و برنامه‌نویسان امکان می‌دهد تا سیستم‌های پیچیده را به بخش‌های کوچکتر و قابل مدیریت تقسیم کنند و از این طریق کار با کدها و توسعه نرم‌افزار را ساده‌تر کنند.

در پیمانه بندی باید به شکلی عمل کنیم که ماژول ها بیشترین انسجام (همبستگی) داخلی (Cohesion) داشته باشند و ارتباط بین ماژول ها (Coupling) حداقل شود (حداقل شدن به معنی صفر شدن بلکه ارتباط تعریف شده‌ی حداقلی است).

۴– **ایجاد سلسله مراتب** (Hierarchy) :

یکی از راه های کاهش و مدیریت پیچیدگی ایجاد سلسله مراتب است. سلسله مراتب را به دو صورت می توانیم ببینیم.

۱- جزیی از (is part of) – تجمیع (Aggregation) اجزاء باعث تشکیل سیستم بزرگتر خواهد شد.

۲- نوعی از (is kind of) – وراثت (Inheritance) باعث به ارث بردن ویژگی‌های انواع بالاتر به انواع پایین‌تر خواهد شد.

سیستم‌های تحلیل و طراحی شی‌گرا(OOSAD)

رویکردهای شی‌گرا در توسعه سیستم‌های اطلاعاتی، از نظر فنی، می‌توانند از هر یک از روش‌های سنتی استفاده کنند. با این حال، رویکردهای شی‌گرا بیشتر با توسعه فازی RAD یا روش چابک مرتبط هستند. تفاوت اصلی بین یک رویکرد سنتی مانند طراحی ساخت‌یافته و یک رویکرد شی‌گرا در نحوه تجزیه مسئله است. در رویکردهای سنتی، فرآیند تجزیه مسئله یا به‌صورت فرآیند محور یا داده محور انجام می‌شود. با این حال، فرآیندها و داده‌ها آن‌قدر به هم نزدیک هستند که انتخاب یکی از آنها به‌عنوان تمرکز اصلی دشوار است. بر اساس این عدم هماهنگی با دنیای واقعی، روش‌های جدید شی‌گرا پدید آمده‌اند که از توالی مبتنی بر RAD در فازهای SDLC استفاده می‌کنند، اما تلاش دارند تا با تمرکز بر تجزیه مسائل بر اساس اشیایی که هم داده‌ها و هم فرآیندها را در خود جای داده‌اند، تأکید بین فرآیند و داده را متعادل سازند.

براساس گفته‌های سازندگان زبان مدل‌سازی یکپارچه(UML) ، یعنی گری بوچ، ایوار جاکوبسون، و جیمز رامبا، هر رویکرد مدرن شیءگرا در توسعه سیستم‌های اطلاعاتی باید مبتنی بر مورد کاربرد (use-case driven)، متمرکز بر معماری (architecture-centric)، و به صورت تکراری و افزایشی باشد.

متمرکز بر معماری

هر رویکرد مدرن به تحلیل و طراحی سیستم باید متمرکز بر معماری باشد. متمرکز بر معماری به این معناست که معماری نرم‌افزاری زیرین در مشخصات سیستم در حال توسعه، راهنمای مشخصات، ساخت و مستندسازی سیستم است. روش‌های تحلیل و طراحی سیستم‌های شیءگرای مدرن باید از حداقل سه دیدگاه معماری مجزا اما مرتبط پشتیبانی کنند: عملکردی، ایستا و پویا. دیدگاه عملکردی یا خارجی، رفتار سیستم را از دید کاربر توصیف می‌کند. دیدگاه ساختاری یا ایستا، سیستم را از نظر ویژگی‌ها، روش‌ها، کلاس‌ها و روابط توصیف می‌کند. دیدگاه رفتاری یا پویا، رفتار سیستم را از نظر پیام‌های رد و بدل شده بین اشیا و تغییرات حالت درون یک شیء توضیح می‌دهد.

تکراری و افزایشی

روش‌های مدرن تحلیل و طراحی سیستم‌های شی‌گرا بر توسعه تکراری و افزایشی تأکید دارند که طی پروژه به‌طور مستمر مورد آزمایش و بهبود قرار می‌گیرد. این روش بدین معناست که تحلیل‌گران سیستم با ایجاد تدریجی سه دیدگاه معماری، درک خود را از مسئله کاربر افزایش می‌دهند. تحلیل‌گر سیستم با همکاری کاربر یک نمای عملکردی از سیستم تحت بررسی ایجاد می‌کند. سپس، تحلیل‌گر تلاش می‌کند یک نمای ساختاری از سیستم در

حال تکامل ایجاد کند. با استفاده از این نمای ساختاری، تحلیل‌گر قابلیت‌های سیستم را بر ساختار در حال توسعه توزیع می‌کند تا نمای رفتاری سیستم را به دست آورد.

در این فرآیند، تحلیل‌گر در حین توسعه سه دیدگاه معماری سیستم در حال تکامل، به‌طور مداوم به آن‌ها بازمی‌گردد و میان دیدگاه‌ها نیز تعامل برقرار می‌کند. به این ترتیب، با درک بهتر نمای ساختاری و رفتاری، تحلیل‌گر متوجه الزامات یا اشتباهات احتمالی در نمای عملکردی می‌شود. این بازخورد می‌تواند منجر به اعمال تغییرات در نمای ساختاری و رفتاری شود.

هر سه نمای معماری سیستم به هم پیوسته و وابسته به یکدیگر هستند (نگاه کنید به شکل ۱۴-۱). با تکمیل هر تکرار و افزودن جزئیات بیشتر، نمای کامل‌تری از الزامات واقعی عملکردی کاربر کشف می‌شود.

مزایای تحلیل و طراحی سیستم‌های شی‌گرا

مفاهیم رویکرد شی‌گرا به تحلیل‌گران کمک می‌کند تا سیستم‌های پیچیده را به ماژول‌های کوچک‌تر و قابل مدیریت تقسیم کنند، روی هر ماژول به‌طور جداگانه کار کنند و به‌راحتی این ماژول‌ها را کنار هم قرار دهند تا سیستم اطلاعاتی کاملی ایجاد کنند. این ماژول‌بندی، توسعه سیستم‌ها را آسان‌تر کرده، همکاری اعضای تیم پروژه را بهبود می‌بخشد و امکان برقراری ارتباط بهتر با کاربران را فراهم می‌کند؛ کاربرانی که باید نیازهای خود را اعلام و تأیید کنند که سیستم تا چه اندازه نیازهای آن‌ها را در طول فرآیند توسعه برآورده می‌کند. با ماژولار کردن توسعه سیستم، تیم پروژه در واقع بخش‌های قابل استفاده مجددی ایجاد می‌کند که می‌توانند در پروژه‌های دیگر به کار گرفته شوند یا به عنوان نقطه شروع پروژه‌های جدید استفاده شوند. در نهایت، این امر می‌تواند موجب صرفه‌جویی در زمان شود زیرا پروژه‌های جدید نیازی به شروع کاملاً از ابتدا ندارند.

فرآیند یکپارچه

فرآیند یکپارچه (Unified Process) یک روش‌شناسی خاص است که زمان و نحوه استفاده از تکنیک‌های مختلف زبان مدل‌سازی یکپارچه (UML) را برای تحلیل و طراحی شی‌گرا تعیین می‌کند. از مهم‌ترین سازندگان این فرآیند می‌توان به گریدی بوچ، ایوار جیکوبسن، و جیمز رامباف اشاره کرد. در حالی که UML پشتیبانی ساختاری برای توسعه ساختار و رفتار یک سیستم اطلاعاتی ارائه می‌دهد، فرآیند یکپارچه پشتیبانی رفتاری را فراهم می‌کند. این فرآیند مبتنی بر مورد کاربرد(UC)، معماری‌محور و به‌صورت تکراری و افزایشی است.

علاوه بر این، فرآیند یکپارچه یک فرآیند توسعه سیستم دوبعدی است که از مجموعه‌ای از فازها و جریان‌های کاری تشکیل شده است. فازها شامل شروع(inception) ، بسط(elaboration) ، ساخت(construction) ، و انتقال (transition) هستند. جریان‌های کاری نیز شامل مدل‌سازی کسب‌وکار، نیازمندی‌ها، تحلیل، طراحی، پیاده‌سازی،

آزمایش، استقرار، مدیریت پیکربندی و تغییرات، مدیریت پروژه، و محیط است. شکل ۱-۱۵ فرآیند یکپارچه را نشان می‌دهد.

فازها

فازهای فرآیند یکپارچه به تحلیل‌گران کمک می‌کنند تا سیستم‌های اطلاعاتی را به‌صورت *تکراری* و *افزایشی* توسعه دهند. این فازها نحوه تکامل یک سیستم اطلاعاتی را در طول زمان توصیف می‌کنند. بسته به فاز توسعه‌ای که سیستم در حال تکامل در آن قرار دارد، میزان فعالیت‌ها در جریان‌های کاری متفاوت است. منحنی موجود در شکل ۱-۱۵ که به هر جریان کاری اختصاص دارد، میزان تقریبی فعالیت‌هایی را که در هر فاز خاص انجام می‌شود نشان می‌دهد. به‌عنوان مثال، فاز شروع (inception) عمدتاً شامل جریان‌های کاری مدل‌سازی کسب‌وکار و نیازمندی‌ها است و تقریباً جریان‌های کاری آزمون و استقرار را نادیده می‌گیرد.

هر فاز شامل مجموعه‌ای از تکرارها است و هر تکرار از جریان‌های کاری مختلفی برای ایجاد نسخه‌ای افزایشی از سیستم در حال تکامل استفاده می‌کند. با پیشروی سیستم در فازها، بهبود می‌یابد و کامل‌تر می‌شود. هر فاز دارای اهداف خاص، تمرکز بر فعالیت‌های جریان‌های کاری و تحویل‌های افزایشی است. فازهای مختلف به شرح زیر است:

فاز شروع(Inception)

از بسیاری جهات، فاز شروع بسیار شبیه به فاز برنامه‌ریزی در رویکرد سنتی چرخه حیات توسعه سیستم (SDLC) است. در این فاز، یک مورد تجاری (business case) برای سیستم پیشنهادی تهیه می‌شود. این مورد شامل تحلیل امکان‌سنجی است که باید به سؤالاتی مانند موارد زیر پاسخ دهد:

- آیا توانایی فنی برای ساخت آن را داریم؟ (امکان‌سنجی فنی)
- اگر آن را بسازیم، آیا برای کسب‌وکار ارزش ایجاد می‌کند؟ (امکان‌سنجی اقتصادی)
- اگر آن را بسازیم، آیا در سازمان مورد استفاده قرار خواهد گرفت؟ (امکان‌سنجی سازمانی)

برای پاسخ به این سؤالات، تیم توسعه بیشتر بر روی جریان‌های کاری مدل‌سازی کسب‌وکار، نیازمندی‌ها و تحلیل تمرکز می‌کند. در برخی موارد، بسته به پیچیدگی‌های فنی که ممکن است در طول توسعه سیستم رخ دهد، یک نمونه اولیه غیرقابل استفاده (throwaway prototype) ایجاد می‌شود. این بدان معناست که جریان‌های کاری طراحی، پیاده‌سازی و آزمون نیز ممکن است در این فاز درگیر باشند. جریان‌های کاری مدیریت پروژه و پشتیبانی محیط برای این فاز بسیار حائز اهمیت هستند. اصلی‌ترین خروجی‌های فاز شروع شامل سند چشم‌انداز است که محدوده پروژه را مشخص می‌کند؛ نیازمندی‌ها و

محدودیت‌های اصلی را شناسایی می‌کند؛ یک برنامه اولیه پروژه را تنظیم می‌کند و امکان‌سنجی و ریسک‌های مرتبط با پروژه، انتخاب محیط مناسب برای توسعه سیستم و برخی جنبه‌های کلاس‌های حوزه مسئله که در حال پیاده‌سازی و آزمون هستند را توصیف می‌کند.

- **فاز توسعه (Elaboration)**

هنگامی که از تحلیل و طراحی سیستم‌های شی‌گرا صحبت می‌کنیم، فعالیت‌های مرتبط با فاز بسط در فرآیند یکپارچه بسیار مهم هستند. جریان‌های کاری تحلیل و طراحی در این فاز بیشترین تمرکز را دارند. در فاز بسط، توسعه سند چشم‌انداز ادامه می‌یابد که شامل نهایی‌سازی مورد تجاری، بازنگری ارزیابی ریسک و تکمیل برنامه پروژه به جزئیاتی کافی است تا ذی‌نفعان بتوانند برای ساخت سیستم نهایی واقعی توافق کنند. این فاز به گردآوری نیازمندی‌ها، ساخت مدل‌های ساختاری و رفتاری UML از حوزه مسئله، و جزئیات چگونگی انطباق مدل‌های حوزه مسئله با معماری سیستم در حال توسعه می‌پردازد.

در این فاز، توسعه‌دهندگان به تمام جریان‌های کاری به جز جریان کاری مهندسی استقرار مشغول هستند. همان‌طور که توسعه‌دهندگان جریان‌های کاری را به‌طور تکراری دنبال می‌کنند، اهمیت مدیریت پیکربندی و تغییرات آشکار می‌شود. همچنین، ابزارهای توسعه که در فاز شروع به‌دست آمده‌اند، در موفقیت پروژه در این فاز بسیار اهمیت دارند. خروجی‌های اصلی این فاز شامل نمودارهای ساختار و رفتار UML و یک نسخه اجرایی پایه از سیستم اطلاعاتی در حال تکامل است. این نسخه پایه به‌عنوان مبنایی برای تمامی تکرارهای بعدی عمل می‌کند. با فراهم کردن یک مبنای مستحکم در این مرحله، توسعه‌دهندگان، پایه‌ای برای تکمیل سیستم در فازهای ساخت و انتقال دارند.

- **فاز ساخت (Construction)**

فاز ساخت به شدت بر روی برنامه‌نویسی سیستم اطلاعاتی در حال تکامل تمرکز دارد. این فاز عمدتاً مربوط به جریان کاری پیاده‌سازی است، با این حال، جریان کاری نیازمندی‌ها و جریان‌های کاری تحلیل و طراحی نیز در این فاز درگیر هستند. در این فاز است که نیازمندی‌های مفقود شده شناسایی می‌شوند و مدل‌های تحلیل و طراحی به‌طور نهایی تکمیل می‌شوند. معمولاً در این فاز تکرارهایی از جریان‌های کاری وجود دارد و در آخرین تکرار، جریان کاری استقرار به صورت فعال آغاز می‌شود. جریان کاری مدیریت پیکربندی و تغییرات، به همراه فعالیت‌های کنترل نسخه، در فاز ساخت بسیار اهمیت پیدا می‌کند. گاهی اوقات، یک تکرار باید به نسخه قبلی برگردد. بدون کنترل نسخه‌های مناسب، بازگشت به نسخه قبلی (پیاده‌سازی افزایشی) سیستم تقریباً غیرممکن است. اصلی‌ترین خروجی این فاز یک پیاده‌سازی از سیستم است که می‌تواند برای آزمایش بتا و پذیرش آماده شود.

- **فاز انتقال**(Transition)

فاز انتقال، مشابه با فاز ساخت، به جنبه‌هایی می‌پردازد که معمولاً با فاز پیاده‌سازی در رویکرد چرخه حیات توسعه سیستم سنتی (SDLC) مرتبط هستند. تمرکز اصلی این فاز بر روی جریان‌های کاری آزمون و استقرار است. به‌طور کلی، جریان‌های کاری مدل‌سازی کسب‌وکار، نیازمندی‌ها و تحلیل باید در تکرارهای قبلی سیستم اطلاعاتی در حال تکامل تکمیل شده باشند. همچنین، جریان کاری آزمون نیز باید در فازهای قبلی سیستم در حال تکامل اجرا شده باشد. بسته به نتایج جریان کاری آزمون، ممکن است برخی فعالیت‌های بازطراحی و برنامه‌نویسی در جریان‌های کاری طراحی و پیاده‌سازی لازم باشد، اما این تغییرات باید در این مرحله حداقلی باشند. از دید مدیریتی، مدیریت پروژه، مدیریت پیکربندی و تغییرات، و محیط درگیر هستند. برخی از فعالیت‌های این فاز شامل آزمایش بتا و پذیرش، تنظیم طراحی و پیاده‌سازی، آموزش کاربران و انتشار محصول نهایی روی یک پلتفرم تولیدی است.
خروجی اصلی این فاز سیستم اطلاعاتی اجرایی واقعی است. سایر خروجی‌ها شامل راهنماهای کاربری، یک برنامه پشتیبانی از کاربران و یک برنامه برای ارتقاء سیستم اطلاعاتی در آینده هستند.

جریان‌های کاری(Workflows)

جریان‌های کاری به وظایف یا فعالیت‌هایی اشاره دارند که یک توسعه‌دهنده برای تکامل یک سیستم اطلاعاتی در طول زمان انجام می‌دهد. جریان‌های کاری در فرآیند یکپارچه به دو دسته کلی تقسیم می‌شوند: مهندسی و پشتیبانی.

- **جریان‌های کاری مهندسی**(Engineering Workflows)

جریان‌های کاری مهندسی شامل مدل‌سازی کسب‌وکار، نیازمندی‌ها، تحلیل، طراحی، پیاده‌سازی، آزمون و استقرار هستند. این جریان‌های کاری به فعالیت‌هایی می‌پردازند که محصول فنی (یعنی سیستم اطلاعاتی) را تولید می‌کنند.

- **جریان کاری مدل‌سازی کسب‌وکار**(Business Modeling Workflow)

جریان کاری مدل‌سازی کسب‌وکار به شناسایی مشکلات و پروژه‌های بالقوه درون سازمان کاربر کمک می‌کند. این جریان کاری به مدیریت در درک دامنه پروژه‌هایی که می‌توانند بهره‌وری و کارایی سازمان کاربر را بهبود بخشند، کمک می‌کند. هدف اصلی مدل‌سازی کسب‌وکار این است که هر دو سازمان توسعه‌دهنده و کاربر درک کنند که سیستم اطلاعاتی در حال توسعه در کجای فرآیندهای کسب‌وکار سازمان کاربر قرار می‌گیرد و چگونه با آنها سازگار است. این جریان کاری عمدتاً در فاز شروع اجرا می‌شود تا اطمینان حاصل شود که سیستم‌های اطلاعاتی را توسعه می‌دهیم که از نظر کسب‌وکار منطقی هستند.

فعالیت‌های این جریان کاری عمدتاً با فاز برنامه‌ریزی در SDLC سنتی مرتبط هستند، اما تکنیک‌های جمع‌آوری نیازمندی‌ها و مدل‌سازی مورد کاربرد و فرآیندهای کسب‌وکار نیز به ما کمک می‌کنند تا وضعیت کسب‌وکار را بهتر درک کنیم.

- **جریان کاری نیازمندی‌ها** (Requirements Workflow)

در فرآیند یکپارچه، جریان کاری نیازمندی‌ها شامل شناسایی نیازمندی‌های کاربردی و غیرکاربردی است. به طور معمول، نیازمندی‌ها از ذینفعان پروژه جمع‌آوری می‌شوند، مانند کاربران نهایی، مدیران سازمان کاربر و حتی مشتریان. این جریان کاری بیشتر در فازهای آغازین و توسعه استفاده می‌شود. نیازمندی‌های شناسایی شده برای توسعه سند چشم‌انداز و موارد کاربردی که در طول فرآیند توسعه استفاده می‌شوند، بسیار مفید هستند. نیازمندی‌های اضافی معمولاً در طول فرآیند توسعه کشف می‌شوند، و تنها در فاز انتقال است که تعداد کمی از نیازمندی‌های جدید یا حتی هیچ‌یک شناسایی می‌شود.

- **جریان کاری تحلیل**(Analysis Workflow)

جریان کاری تحلیل به طور عمده به ایجاد یک مدل تحلیلی از حوزه مشکل می‌پردازد. در فرآیند یکپارچه، تحلیلگر شروع به طراحی معماری مرتبط با حوزه مشکل می‌کند و با استفاده از UML ، نمودارهای ساختاری و رفتاری را ایجاد می‌کند که توصیف‌کننده کلاس‌های حوزه مشکل و تعاملات آنها هستند. هدف اصلی جریان کاری تحلیل این است که هر دو سازمان توسعه‌دهنده و کاربر، مشکل و دامنه آن را به خوبی درک کنند بدون آنکه وارد تحلیل بیش از حد شوند. در صورت عدم دقت، تحلیلگران ممکن است دچار "فلج تحلیلی" شوند، یعنی پروژه به حدی درگیر تحلیل می‌شود که هیچگاه به طراحی و پیاده‌سازی نمی‌رسد. هدف دوم این جریان کاری، شناسایی کلاس‌های قابل استفاده مجدد برای کتابخانه‌های کلاس است؛ با استفاده‌ی مجدد از کلاس‌های از پیش تعریف شده، تحلیلگر می‌تواند از ایجاد مجدد کلاس‌ها هنگام طراحی نمودارهای ساختاری و رفتاری جلوگیری کند. این جریان کاری بیشتر با فاز تفضیل مرتبط است، اما همانند جریان کاری نیازمندی‌ها، احتمال دارد تحلیل‌های اضافی در طول فرآیند توسعه نیاز شود.

جریان کاری طراحی(Design Workflow)

جریان کاری طراحی، مدل تحلیل را به شکلی که برای پیاده‌سازی سیستم مناسب باشد تبدیل می‌کند. در حالی که جریان کاری تحلیل بر درک حوزه مشکل تمرکز دارد، جریان کاری طراحی به توسعه راه‌حلی می‌پردازد که در محیط خاصی اجرا شود. اساساً، جریان کاری طراحی، توصیف سیستم در حال تکامل را با اضافه کردن کلاس‌هایی که به محیط سیستم مرتبط هستند، به مدل تحلیل بهبود می‌دهد. این جریان کاری شامل فعالیت‌هایی مانند طراحی دقیق کلاس‌های حوزه مشکل، بهینه‌سازی سیستم اطلاعاتی در حال تکامل، طراحی پایگاه داده، طراحی رابط کاربری، و طراحی معماری فیزیکی است. جریان کاری طراحی عمدتاً با فازهای تفضیل و ساخت فرآیند یکپارچه مرتبط است.

جریان کاری طراحی (Design Workflow)

جریان کاری طراحی، مدل تحلیل را به مدلی تبدیل می‌کند که قابل پیاده‌سازی باشد: مدل طراحی. در حالی که جریان کاری تحلیل بر درک حوزه مشکل تمرکز داشت، جریان کاری طراحی به توسعه راه‌حلی می‌پردازد که در محیطی خاص اجرا شود. به طور کلی، جریان کاری طراحی با اضافه کردن کلاس‌هایی که به محیط سیستم مربوط می‌شوند، توصیف سیستم در حال تکامل را بهبود می‌بخشد. این جریان شامل فعالیت‌هایی مانند طراحی دقیق کلاس‌های حوزه مشکل، بهینه‌سازی سیستم اطلاعاتی در حال تکامل، طراحی پایگاه داده، طراحی رابط کاربری، و طراحی معماری فیزیکی است. این جریان کاری بیشتر با فازهای تفضیل و ساخت فرآیند یکپارچه مرتبط است.

جریان کاری پیاده‌سازی(Implementation Workflow)

هدف اصلی جریان کاری پیاده‌سازی، ایجاد یک راه‌حل اجرایی بر اساس مدل طراحی (یعنی برنامه‌نویسی) است. این شامل نوشتن کلاس‌های جدید و همچنین استفاده از کلاس‌های قابل استفاده مجدد از کتابخانه‌های کلاس اجرایی در راه‌حل در حال تکامل است. همانند هر فعالیت برنامه‌نویسی دیگری، کلاس‌های جدید و تعاملات آن‌ها با کلاس‌های قابل استفاده مجدد نیز باید تست شوند. در صورتی که گروه‌های مختلفی پیاده‌سازی سیستم اطلاعاتی را انجام دهند، افراد پیاده‌ساز باید ماژول‌های جداگانه و آزمایش‌شده را نیز یکپارچه کنند تا یک نسخه اجرایی از سیستم ایجاد شود. این جریان کاری عمدتاً با فازهای تفضیل و ساخت مرتبط است.

جریان کاری تست (Testing Workflow)

هدف اصلی جریان کاری تست، افزایش کیفیت سیستم در حال تکامل است. تست فراتر از تست‌های ساده واحدی است که در جریان پیاده‌سازی انجام می‌شود. این جریان شامل تست یکپارچگی تمام ماژول‌ها برای پیاده‌سازی سیستم، تست پذیرش کاربر، و تست آلفای واقعی نرم‌افزار نیز می‌شود. در واقع، تست باید در طول توسعه سیستم

انجام شود؛ تست مدل‌های تحلیل و طراحی در طول فازهای تفضیل و ساخت صورت می‌گیرد، در حالی که تست پیاده‌سازی عمدتاً در فاز ساخت و تا حدی در فاز انتقال انجام می‌شود. به طور کلی، در پایان هر تکرار در طول توسعه سیستم اطلاعاتی، نوعی تست باید انجام شود.

جریان کاری استقرار (Deployment Workflow)

جریان کاری استقرار بیشتر با فاز انتقال فرآیند یکپارچه مرتبط است. این جریان شامل فعالیت‌هایی مانند بسته‌بندی نرم‌افزار، توزیع، نصب، و تست بتا است. هنگام استقرار سیستم جدید در یک سازمان، ممکن است توسعه‌دهندگان نیاز داشته باشند داده‌های فعلی را تبدیل کرده، نرم‌افزار جدید را با نرم‌افزار موجود یکپارچه کنند و کاربران نهایی را برای استفاده از سیستم جدید آموزش دهند.

جریان‌های کاری پشتیبانی (Supporting Workflows)

جریان‌های کاری پشتیبانی شامل مدیریت پروژه، مدیریت پیکربندی و تغییر، و محیط است. این جریان‌ها بر جنبه‌های مدیریتی توسعه سیستم‌های اطلاعاتی تمرکز دارند.

جریان کاری مدیریت پروژه (Project Management Workflow)

در حالی که جریان‌های کاری دیگر در فرآیند یکپارچه به طور فنی در هر چهار فاز فعال هستند، جریان کاری مدیریت پروژه تنها جریان کاری واقعی میان فازهاست. فرآیند توسعه از توسعه افزایشی و تکراری پشتیبانی می‌کند، بنابراین سیستم‌های اطلاعاتی معمولاً به مرور زمان رشد یا تکامل پیدا می‌کنند. در پایان هر تکرار، یک نسخه افزایشی جدید از سیستم برای تحویل آماده می‌شود. به دلیل پیچیدگی مدل دو بعدی فرآیند یکپارچه (جریان‌های کاری و فازها)، جریان کاری مدیریت پروژه اهمیت زیادی دارد. فعالیت‌های این جریان شامل شناسایی و مدیریت ریسک‌ها، مدیریت محدوده، تخمین زمان لازم برای تکمیل هر تکرار و کل پروژه، تخمین هزینه تکرارهای جداگانه و کل پروژه، و پیگیری پیشرفت کار به سمت نسخه نهایی سیستم اطلاعاتی در حال تکامل است.

جریان‌کاری پیکربندی و مدیریت تغییرات

هدف اصلی جریان‌کاری پیکربندی و مدیریت تغییرات، پیگیری وضعیت سیستم در حال توسعه است. به طور خلاصه، سیستم اطلاعاتی در حال توسعه شامل مجموعه‌ای از مصنوعات (مانند نمودارها، کد منبع و فایل‌های اجرایی) است. این مصنوعات در طول فرآیند توسعه به‌طور مرتب تغییر می‌کنند. با توجه به میزان کار و هزینه‌ای که در توسعه این مصنوعات صرف می‌شود، باید این مصنوعات به‌عنوان دارایی‌های گران‌بها مدیریت شوند؛ به این معنی که باید کنترل‌های دسترسی برای محافظت از این مصنوعات در برابر سرقت یا تخریب در نظر گرفته شود.

از آنجا که این مصنوعات به طور مداوم و پیوسته تغییر می‌کنند، استفاده از مکانیزم‌های مناسب کنترل نسخه ضروری است. علاوه بر این، مقدار زیادی اطلاعات مدیریتی از پروژه مانند نویسنده، زمان و مکان هر تغییر باید ثبت شود. جریان‌کاری مدیریت پیکربندی و تغییرات بیشتر با فازهای ساخت و انتقال مرتبط است.

جریان‌کاری محیط

در طول توسعه یک سیستم اطلاعاتی، تیم توسعه به استفاده از ابزارها و فرآیندهای مختلف نیاز دارد. جریان‌کاری محیط به این نیازها پاسخ می‌دهد. به‌عنوان مثال، ممکن است استفاده از یک ابزار CASE برای پشتیبانی از توسعه یک سیستم اطلاعاتی شی‌گرا از طریق UML لازم باشد. ابزارهای دیگر شامل محیط‌های برنامه‌نویسی، ابزارهای مدیریت پروژه و ابزارهای مدیریت پیکربندی هستند. جریان‌کاری محیط شامل تهیه و نصب این ابزارها می‌شود. با اینکه این جریان‌کاری در تمام فازهای فرآیند یکپارچه می‌تواند فعال باشد، اما به‌طور عمده در فاز آغازی اهمیت بیشتری دارد.

گسترش‌های فرآیند یکپارچه

با اینکه فرآیند یکپارچه بسیار بزرگ و پیچیده است، نویسندگان بسیاری به نقاط ضعف کلیدی آن اشاره کرده‌اند. اولاً، فرآیند یکپارچه به مسائل مربوط به تأمین نیرو، بودجه‌بندی یا مدیریت قراردادها نمی‌پردازد. این فعالیت‌ها به‌طور صریح از فرآیند یکپارچه کنار گذاشته شده‌اند. دوماً، فرآیند یکپارچه به مسائلی مانند نگهداری، عملیات یا پشتیبانی از محصول پس از تحویل توجهی ندارد؛ بنابراین، این فرآیند یک فرآیند کامل نرم‌افزاری نیست، بلکه فقط یک فرآیند توسعه است. سوماً، فرآیند یکپارچه به مسائل بین‌پروژه‌ای یا مابین پروژه‌ها نمی‌پردازد. با توجه به اهمیت استفاده مجدد در توسعه سیستم‌های شی‌گرا و این واقعیت که در بسیاری از سازمان‌ها کارمندان به‌طور هم‌زمان در چندین پروژه مشغول به کار هستند، نادیده گرفتن مسائل بین‌پروژه‌ای یک نقص عمده محسوب می‌شود.

برای پرداختن به این نواقص، «امبلر» و «کانستانتین» پیشنهاد کرده‌اند که یک فاز تولید و دو جریان‌کاری اضافه شود: جریان‌کاری عملیات و پشتیبانی و جریان‌کاری مدیریت زیرساخت (شکل ۱-۱۶ را ببینید). علاوه بر این جریان‌کاری‌های جدید، جریان‌کاری‌های آزمایش، انتقال و محیط اصلاح شده و جریان‌کاری‌های مدیریت پروژه و مدیریت پیکربندی و تغییرات به فاز تولید نیز گسترش یافته‌اند. این گسترش‌ها بر اساس فرآیندهای جایگزین نرم‌افزار شی‌گرا مانند فرآیند OPEN (Object-oriented Process, Environment, and Notation) و فرآیند نرم‌افزار شی‌گرا انجام شده‌اند.

مرحله تولید (Production Phase)

مرحله تولید عمدتاً به مسائلی مرتبط با محصول نرم‌افزاری پس از استقرار موفق آن می‌پردازد. این مرحله بر موضوعاتی مانند به‌روزرسانی، نگهداری و بهره‌برداری از نرم‌افزار تمرکز دارد. برخلاف مراحل قبلی، در این مرحله تکرارها و تحویل‌های تدریجی وجود ندارد. اگر قرار باشد نسخه جدیدی از نرم‌افزار توسعه یابد، توسعه‌دهندگان باید مراحل اول تا چهارم را مجدداً طی کنند. با توجه به فعالیت‌هایی که در این مرحله انجام می‌شود، هیچ جریان کاری مهندسی خاصی مرتبط نیست. جریان‌های کاری پشتیبان که در این مرحله فعال هستند شامل مدیریت پیکربندی و تغییرات، مدیریت پروژه، عملیات و پشتیبانی جدید، و مدیریت زیرساخت‌ها می‌باشند.

جریان کار عملیات و پشتیبانی (Operations and Support Workflow)

جریان کار عملیات و پشتیبانی، همان‌طور که ممکن است حدس بزنید، به مسائل مرتبط با پشتیبانی از نسخه فعلی نرم‌افزار و بهره‌برداری روزانه از آن می‌پردازد. فعالیت‌ها شامل ایجاد برنامه‌هایی برای عملیات و پشتیبانی از محصول نرم‌افزاری پس از استقرار، تهیه مستندات آموزشی و کاربری، اعمال رویه‌های پشتیبان‌گیری ضروری، نظارت و بهینه‌سازی عملکرد نرم‌افزار و انجام تعمیر و نگهداری اصلاحی بر روی نرم‌افزار است. این جریان کار در مرحله ساخت فعال می‌شود؛ سطح فعالیت آن در طول مرحله انتقال و در نهایت مرحله تولید افزایش می‌یابد. فعالیت این جریان کار زمانی متوقف می‌شود که نسخه فعلی نرم‌افزار با نسخه جدیدی جایگزین شود. بسیاری از توسعه‌دهندگان به اشتباه فکر می‌کنند که پس از تحویل نرم‌افزار به مشتری، کار آن‌ها به پایان رسیده است. در بیشتر موارد، کار پشتیبانی از محصول نرم‌افزاری بسیار پرهزینه‌تر و زمان‌برتر از توسعه اولیه است. در این مرحله، کار توسعه‌دهنده ممکن است تازه شروع شود.

جریان کار مدیریت زیرساخت (Infrastructure Management Workflow)

هدف اصلی جریان کار مدیریت زیرساخت، پشتیبانی از توسعه زیرساخت لازم برای توسعه سیستم‌های شیءگرا است. فعالیت‌هایی مانند توسعه و اصلاح کتابخانه‌ها، استانداردها و مدل‌های سازمانی بسیار مهم هستند. هنگامی که توسعه و نگهداری یک مدل معماری دامنه مسئله فراتر از محدوده‌ی یک پروژه می‌رود و استفاده مجدد قرار است رخ دهد، جریان کار مدیریت زیرساخت ضروری است. مجموعه‌ای دیگر از فعالیت‌های میان‌پروژه‌ای بسیار مهم، بهبود فرآیند توسعه نرم‌افزار است. از آنجا که فعالیت‌های این جریان کار تمایل به تأثیرگذاری بر پروژه‌های زیادی دارند و فرآیند یکپارچه تنها بر یک پروژه خاص تمرکز دارد، فرآیند یکپارچه تمایل به نادیده‌گیری این فعالیت‌ها دارد (یعنی این فعالیت‌ها فراتر از محدوده و هدف فرآیند یکپارچه هستند).

تغییرات و توسعه‌های جریان کارهای موجود (Existing Workflow Modifications and Extensions)

علاوه بر جریان‌های کاری که به منظور رفع کمبودهای موجود در فرآیند یکپارچه اضافه شدند، جریان‌های کاری موجود نیز باید به مرحله تولید گسترش داده یا اصلاح شوند. این جریان‌های کاری شامل تست، استقرار، محیط، مدیریت پروژه و مدیریت پیکربندی و تغییرات هستند.

جریان کار تست (Test Workflo*w*)

برای توسعه سیستم‌های اطلاعاتی با کیفیت بالا، باید روی هر تحویلی، از جمله تحویل‌هایی که در مرحله آغازین ایجاد می‌شوند، آزمایش انجام شود. در غیر این صورت، سیستم‌هایی با کیفیت پایین به مشتری تحویل داده خواهند شد.

جریان کار استقرار (Deployment Workflo*w*)

سیستم‌های قدیمی در بیشتر شرکت‌ها وجود دارند و این سیستم‌ها دارای پایگاه‌های داده‌ای هستند که باید برای تعامل با سیستم‌های جدید تبدیل شوند. به دلیل پیچیدگی استقرار سیستم‌های جدید، این تبدیل نیاز به برنامه‌ریزی دقیق دارد. بنابراین، فعالیت‌های جریان کار استقرار باید از مرحله آغازین شروع شود و نباید تا پایان مرحله ساخت به تأخیر بیفتد، همان‌طور که توسط فرآیند یکپارچه پیشنهاد شده است.

جریان کار محیط (Environment Workflow)

جریان کار محیط نیاز به اصلاح دارد تا فعالیت‌های مرتبط با راه‌اندازی محیط‌های عملیات و تولید را در بر بگیرد. کار واقعی انجام شده مشابه کار مربوط به راه‌اندازی محیط توسعه است که در طول مرحله آغازین انجام شد. در این مورد، کار اضافی در طول مرحله انتقال انجام می‌شود.

مدلسازی کسب و کار (Business Modeling)

مدلسازی کسب و کار (Business Modeling) فرآیندی است که به منظور درک، تحلیل و بهبود فرآیندها و ساختارهای کسب و کار انجام می‌شود. این فرآیند به مدیران و تحلیل‌گران کمک می‌کند تا دیدگاه جامعی نسبت به عملکرد سازمان به دست آورند و بتوانند تصمیمات استراتژیک بهتری اتخاذ کنند.

اهداف مدلسازی کسب و کار

1. **درک دقیق کسب و کار** :کمک به تحلیل و درک بهتر فرآیندها و ساختارهای موجود.

۲. **شناسایی مشکلات و فرصت‌ها** :شناسایی نقاط ضعف و قوت، فرصت‌ها و تهدیدها در کسب و کار.

۳. **بهبود و بازطراحی فرآیندها** :ارائه راهکارهایی برای بهبود و بازطراحی فرآیندهای کسب و کار.

٤. **تسهیل ارتباطات و همکاری** :ایجاد یک زبان مشترک برای تمامی ذینفعان به منظور تسهیل ارتباطات و همکاری.

اجزاء اصلی سند مدلسازی کسب و کار

۱) مقدمه و چکیده

مقدمه بخشی است که اطلاعات اولیه و زمینه‌ای را درباره موضوع ارائه می‌دهد. این بخش شامل زمینه‌سازی موضوع، بیان مسئله، اهمیت و ضرورت موضوع، و اهداف تحقیق است.

چکیده خلاصه‌ای جامع و مختصر از محتوای سند است که به خوانندگان کمک می‌کند تا به سرعت از اهداف، روش‌ها، نتایج و نتیجه‌گیری‌های اصلی آگاه شوند. این بخش باید کوتاه، جامع، و شامل کلیدواژه‌های مرتبط باشد.

۲) اهداف سازمانی

اهداف می‌توانند به دو دسته کوتاه‌مدت و بلندمدت تقسیم شوند:

- **کوتاه‌مدت** :اهدافی که قرار است در بازه‌های زمانی کوتاه مانند یک سال محقق شوند.
- **بلندمدت** :اهداف استراتژیکی که در بازه‌های زمانی طولانی‌تر به آن‌ها دست یافته شود.

۳) واحدهای فیزیکی و واحدهای سازمانی

واحدهای فیزیکی به مکان‌ها و تأسیساتی گفته می‌شود که فعالیت‌ها و فرآیندهای سازمان در آنجا انجام می‌شود. هر واحد فیزیکی ممکن است شامل تعدادی واحد سازمانی نیز باشد.

فرض کنید یک شرکت تولیدی قصد دارد یک سیستم مدیریت تولید Manufacturing Execution Systemیا (MES) پیاده‌سازی کند. واحدهای فیزیکی شامل کارخانه‌ها و انبارها هستند. تحلیل‌گران سیستم باید مکان‌ها و فرآیندهای کاری در این واحدها را شناسایی و مستندسازی کنند. سپس سیستم به گونه‌ای طراحی می‌شود که تمامی فرآیندهای تولیدی، نگهداری و موجودی را در واحدهای فیزیکی پوشش دهد.

واحدهای فیزیکی در تحلیل و طراحی سیستم‌های اطلاعاتی نقش کلیدی دارند و به تضمین انطباق سیستم با نیازهای عملیاتی و جغرافیایی سازمان کمک می‌کنند.

واحدهای سازمانی به گروه‌های مشخصی از کارکنان یا بخش‌ها در یک سازمان اشاره دارد که وظایف و مسئولیت‌های خاصی را بر عهده دارند. هر واحد سازمانی معمولاً دارای ساختار سلسله‌مراتبی خاص خود و فرآیندهای داخلی است.

فرض کنید سازمانی قصد دارد یک سیستم مدیریت منابع انسانی (HRMS) پیاده‌سازی کند. واحدهای سازمانی مانند منابع انسانی، مالی، و فناوری اطلاعات هر کدام نیازمندی‌ها و الزامات خاص خود را دارند. تحلیل‌گران سیستم باید با هر یک از این واحدها تعامل کنند تا نیازهای آن‌ها را شناسایی و مستند کنند. سپس سیستم به گونه‌ای طراحی می‌شود که بتواند تمامی فرآیندها و نیازمندی‌های این واحدها را پوشش دهد.

واحدهای سازمانی نقش کلیدی در تحلیل و طراحی سیستم‌های اطلاعاتی دارند و تضمین می‌کنند که سیستم نهایی به درستی با نیازها و فرآیندهای سازمان هماهنگ است.

۴) نقش سازمانی(Organizational Role)

نقش سازمانی به مجموعه‌ای از وظایف، مسئولیت‌ها و انتظاراتی گفته می‌شود که از یک فرد یا یک تیم در یک سازمان انتظار می‌رود. نقش‌ها معمولاً بر اساس فرآیندهای کسب و کار و اهداف سازمانی تعریف می‌شوند

۵) فرآیند سازمانی

فرآیند سازمانی به توالی منطقی از وظایف و فعالیت‌های سازمان گفته می‌شود که با همکاری و هماهنگی افراد و واحدهای مختلف سازمان برای تحقق اهداف خاص صورت می‌گیرد. این فرآیندها معمولاً شامل ورودی‌ها، خروجی‌ها، منابع و کنترل‌های مختلف هستند.(خلاصه : مجموعه‌ای مرتب از فعالیت‌های سازمانی)

مثال از یک فرآیند سازمانی

فرض کنید یک شرکت تولیدی فرآیندی برای مدیریت سفارشات مشتری دارد. این فرآیند ممکن است شامل مراحل زیر باشد:

۱. **دریافت سفارش مشتری** :ثبت اطلاعات سفارش و تأیید آن.

۲. **بررسی موجودی** :چک کردن موجودی انبار برای اطمینان از تأمین کالا.

۳. **تولید کالا** :اگر کالا موجود نباشد، دستور تولید صادر می‌شود.

۴. **بسته‌بندی و حمل** :بسته‌بندی کالا و ارسال آن به مشتری.

۵. **پیگیری و پشتیبانی** :پیگیری وضعیت ارسال و ارائه خدمات پس از فروش.

برای مستندسازی فرآیندهای سازمانی اجزاء زیر مورد نیاز می‌باشند.

- کد فرآیند : باید در سرتاسر پروژه یکتا (Unique) باشد.
- نام فرآیند : نامی که مربوط به هدف فرآیند باشد.
- شرح مختصر : توضیحی مختصر از فرآیند سازمانی مورد نظر در حد ۴ تا ۶ خط
- سناریوی فرآیند : شرح ترتیب فعالیت‌هایی که فرآیند را شکل می‌دهند.
- رخدادها (event) – اتفاقاتی که باعث خواهند شد فرآیند دست خوش تغییراتی شود که در سناریوی فرآیند قابل ذکر نیستند.
- منابع Resources
- فرهنگ لغت (Glossary) : ارائه‌ی تعریف دقیق و شفاف از اصطلاحات و مفاهیم کلیدی
- فرم‌ها : برای فرم موارد زیر را باید ارئه نماییم.
  - کد فرم
  - نام فرم
  - شکل فرم
  - فیلدهای روی فرم
  - بررسی های روی فرم

۶) چکیده خلاصه‌ای جامع و مختصر از محتوای سند است که به خوانندگان کمک می‌کند تا به سرعت از اهداف، روش‌ها، نتایج و نتیجه‌گیری‌های اصلی آگاه شوند. این بخش باید کوتاه، جامع، و شامل کلیدواژه‌های مرتبط باشد.

روش‌های بیان سناریوی فرآیند

۱- شبه کد (Pseudocode)

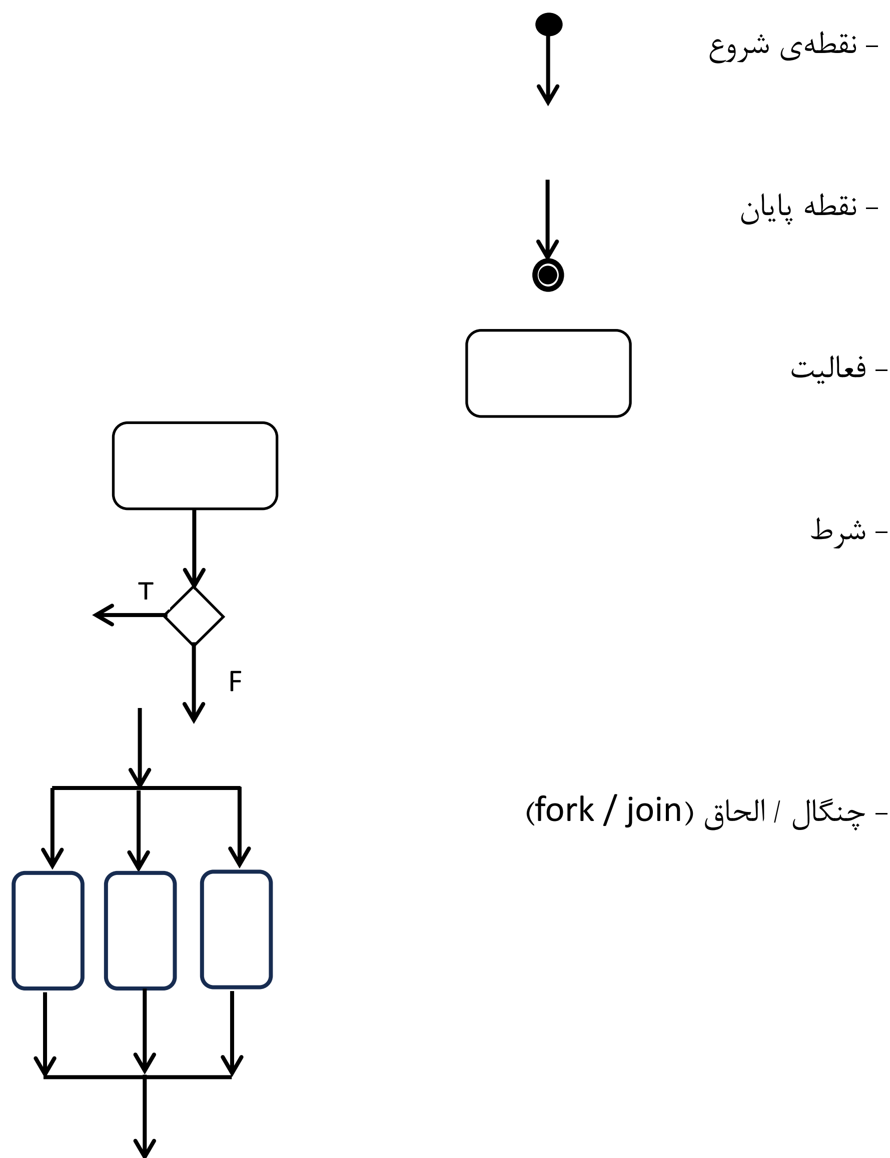
شبه کد می‌تواند به منظور بیان و مستندسازی سناریوهای فرآیند و فرآیندهای کسب‌وکار استفاده شود. شبه کد به زبان ساده و قابل فهم برای انسان نوشته می‌شود و کمک می‌کند تا منطق و جریان کار به صورت واضح و بدون پیچیدگی‌های زبانی یا دستوری خاصی نمایش داده شود. این روش بسیار مفید است، به‌ویژه زمانی که نیاز به توضیح دقیق منطق و مراحل یک فرآیند داریم.

مثال: شبه کد فرآیند دریافت سفارش مشتری

```
// شروع فرآیند دریافت سفارش مشتری
اگر مشتری سفارش جدیدی ثبت کند
    بررسی موجودی کالا
    اگر موجودی کافی باشد
        تأیید سفارش و ارسال به انبار
        بسته‌بندی کالا
        ارسال کالا به مشتری
    وگرنه
        اطلاع رسانی به مشتری در مورد عدم موجودی
پایان اگر
پایان فرآیند دریافت سفارش مشتری
```

۲- نمودار فعالیت (Activity Diagram)

نمودار فعالیت برای بیان و مستندسازی سناریوی فرآیندهای کسب‌وکار استفاده می‌شوند. این نمودارها یکی از ابزارهای مهم در زبان مدلسازی یکپارچه (UML) هستند و به توضیح دقیق جریان کار و فعالیت‌ها در یک فرآیند کمک می‌کنند. نمودار فعالیت روابط میان فعالیت‌ها، ترتیب اجرا و وابستگی‌های آن‌ها را به تصویر می‌کشد. اجزاء نمودار فعالیت شامل موارد زیر می‌باشد.



۳- خط شنا (Swim Lane)

نمودار **Swimlane** یا خط‌شنا، یکی از انواع نمودارهای فعالیت در زبان مدلسازی یکپارچه (UML) است که برای نمایش فرآیندها و تقسیم وظایف میان نقش‌ها یا واحدهای مختلف در یک سازمان استفاده می‌شود. این

نمودار به تحلیل‌گران کمک می‌کند تا ببینند که چگونه مسئولیت‌ها در طول فرآیند توزیع شده و چگونه نقش‌ها با یکدیگر تعامل دارند.

شروع	پست‌سازمانی i	پست‌سازمانی j	پست‌سازمانی k	....	......

جمع‌آوری نیازمندی‌ها Requirement Gathering

جمع‌آوری نیازمندی‌ها یکی از مراحل حیاتی در فرآیند توسعه نرم‌افزار و پروژه‌های سیستم‌های اطلاعاتی است. این مرحله به شناخت دقیق نیازها و خواسته‌های کاربران و مشتریان کمک می‌کند و پایه‌ای برای طراحی و توسعه صحیح سیستم‌ها فراهم می‌آورد.

مراحل و فنون جمع‌آوری نیازمندی‌ها:

۱. **مصاحبه‌ها (Interviews) :**

- برگزاری جلسات مصاحبه با ذینفعان و کاربران کلیدی برای درک نیازهایشان.
- پرسیدن سوالات مشخص و باز برای کشف نیازهای عملیاتی و غیر عملیاتی.

۲. **کارگاه‌ها (Workshops) :**

- برگزاری جلسات گروهی با حضور کاربران، تحلیل‌گران و توسعه‌دهندگان برای تبادل ایده‌ها و نیازمندی‌ها.
- استفاده از تکنیک‌های مانند Brainstorming برای ایجاد خلاقیت و استخراج نیازها.

۳. **پرسشنامه‌ها و نظرسنجی‌ها(Questionnaires and Surveys) :**

- تهیه و توزیع پرسشنامه‌ها برای جمع‌آوری اطلاعات از تعداد زیادی از کاربران.
- تحلیل داده‌های جمع‌آوری شده برای استخراج نیازمندی‌ها.

۴. **مشاهده مستقیم(Observation) :**

- مشاهده فرآیندها و عملکرد کاربران در محیط کاری واقعی.
- یادداشت‌برداری از رفتارها و تعاملات کاربران با سیستم‌های موجود.

۵. **مطالعه مستندات(Document Analysis) :**

- بررسی مستندات و مدارک موجود مانند راهنماها، گزارش‌ها و استانداردها.
- استخراج اطلاعات مفید و نیازمندی‌های مستند شده.

۶. **نمونه‌سازی(Prototyping) :**

- ایجاد نمونه‌های اولیه از سیستم برای دریافت بازخورد کاربران.
- اصلاح و بهبود نمونه‌ها بر اساس بازخوردهای دریافت شده.

اهداف و مزایای جمع‌آوری نیازمندی‌ها:

۱. **شفافیت و وضوح نیازها:**

کمک به تیم توسعه برای فهم دقیق نیازمندی‌ها و جلوگیری از تفسیرهای نادرست.

۲. **کاهش ریسک‌ها:**

شناسایی و مدیریت ریسک‌های مرتبط با نیازمندی‌ها در مراحل ابتدایی پروژه.

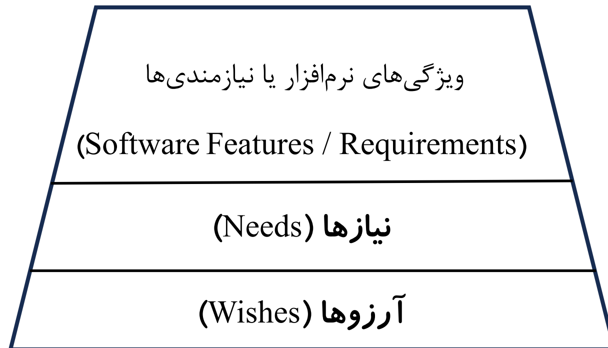
۳. **بهبود کیفیت محصول نهایی:**

اطمینان از این که سیستم نهایی مطابق با نیازهای کاربران و مشتریان است.

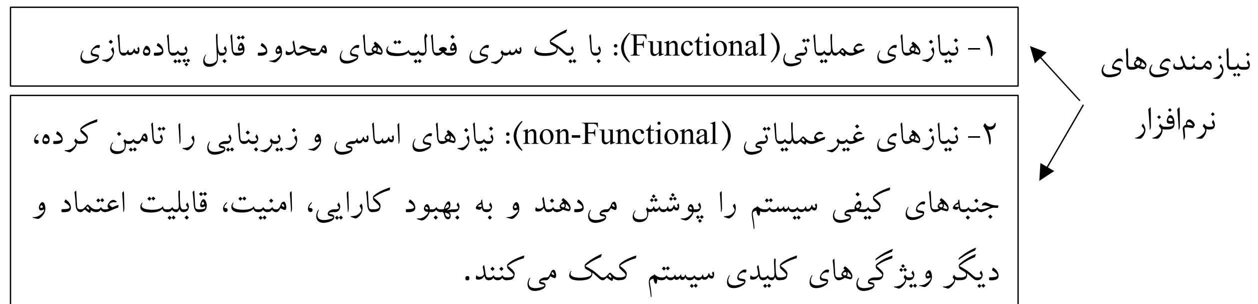
۴. **تسهیل ارتباطات:**

ایجاد زبان مشترک بین کاربران، تحلیل‌گران و توسعه‌دهندگان.

جمع‌آوری نیازمندی‌ها به طور کلی فرآیندی مداوم است و در طول چرخه عمر پروژه ممکن است نیازمندی‌ها تغییر کنند یا تکمیل شوند. استفاده از ابزارها و تکنیک‌های مختلف به تیم‌های پروژه کمک می‌کند تا نیازمندی‌ها را به بهترین شکل ممکن جمع‌آوری و مدیریت کنند.



در مصاحبه‌ها با انواع خواسته‌های مختلف روبه‌رو هستیم که آنها را تحت عنوان آروزها می‌توانیم نام‌گذاری کنیم. مجموعه‌ی این خواسته‌ها در کنار هم سیستم نرم‌افزاری مناسب مورد نیاز را تشکیل نخواهد داد. تحلیلگر باید تلاش کند با استدلال و توافق با صاحبان سیستم مجموعه‌ی نیازهای واقعی را درست تشخیص دهد. اما در بین این نیازها به دلایل مختلفی بخش‌هایی را می‌توانیم پیاده سازی کنیم که با مطالعات امکان‌سنجی قابلیت تولید داشته باشند. این مجموعه‌ی قابل پیاده‌سازی را ویژگی‌های نرم‌افزار یا نیازمندی‌های نرم‌افزار می‌نامیم. این نیازمندی‌ها را به دو دسته می‌توان تقسیم نمود.



ویژگی‌های اصلی نیازهای غیرعملیاتی:

۱. **کارایی یا کارآمدی (Performance) :**

سیستم باید در زمان معقولی پاسخ‌دهی کند و توانایی مدیریت بار کاری مورد نظر را داشته باشد. برای مثال زمان پاسخ‌دهی در هنگام انجام یک تراکنش کمتر از یک حد معینی باشد. البته باید توجه داشته که بخشی از نیازمندی‌های مربوط به کارایی به صورت غیرعملیاتی فراهم می‌شود.

۲. **قابلیت دسترسی (Accessibility)**

این ویژگی به توانایی سیستم برای دسترس‌پذیر بودن توسط کاربران مورد نظر در زمان و مکان مورد نیاز اشاره دارد.

۳. **قابلیت حمل (Portability)**

قابلیت حمل به توانایی سیستم یا نرم‌افزار برای انتقال از یک محیط یا سکو (Platform) به محیط یا سکوی دیگری اشاره دارد بدون اینکه عملکرد یا کیفیت آن دچار افت شود.

۴. **امنیت**(Security)

سیستم باید از داده‌ها و اطلاعات محافظت کند و دسترسی غیرمجاز را جلوگیری کند. به عنوان مثال همه‌ی تراکنش‌های مالی باید از طریق ارتباطات رمزگذاری‌شده انجام شوند.

۵. **قابلیت نگهداری**(Maintainability)

سیستم باید به راحتی قابل نگهداری و به‌روزرسانی باشد. برای مثال کد منبع سیستم باید مستند و قابل فهم باشد تا توسعه‌دهندگان بتوانند به راحتی آن را تغییر دهند.

۶. **قابلیت استفاده**(Usability)

سیستم باید به گونه‌ای طراحی شود که کاربران به راحتی بتوانند از آن استفاده کنند. برای مثال رابط کاربری باید ساده و کاربرپسند باشد و کاربران جدید بتوانند به راحتی با سیستم کار کنند.

۷. **قابلیت مقیاس‌پذیری**(Scalability)

سیستم باید قابلیت افزایش و کاهش بار کاری را داشته باشد بدون اینکه عملکردش کاهش یابد. برای مثال سیستم باید بتواند به راحتی تعداد کاربران را از ۱۰۰۰ به ۱۰۰۰۰ کاربر ارتقا دهد.

مدلسازی دامنه (Domain Modeling)

مدلسازی دامنه یکی از مراحل مهم در توسعه سیستم‌های نرم‌افزاری است که به تحلیلگران و توسعه‌دهندگان کمک می‌کند تا مفاهیم و روابط کلیدی در یک دامنه خاص را شناسایی و مدل‌سازی کنند. این فرآیند به درک بهتر از نیازمندی‌ها، ساختار و رفتار سیستم کمک می‌کند و پایه‌ای برای طراحی و پیاده‌سازی سیستم ارائه می‌دهد.

ویژگی‌های اصلی مدلسازی دامنه:

۱. **شناسایی موجودیت‌ها و مفاهیم کلیدی:**

موجودیت‌ها (Entities) نشان‌دهنده اشیاء یا مفاهیمی هستند که در دامنه مورد نظر وجود دارند و اهمیت دارند.

مثال: در یک سیستم کتابخانه، موجودیت‌های کلیدی می‌توانند کتاب، عضو، و امانت باشند.

۲. **تعریف ویژگی‌ها و صفات**(Attributes)

هر موجودیت دارای ویژگی‌هایی است که آن را توصیف می‌کنند.

مثال: موجودیت کتاب می‌تواند دارای ویژگی‌هایی مانند عنوان، نویسنده، و تاریخ انتشار باشد.

۳. **تعیین روابط**(Relationships)

روابط بین موجودیت‌ها نحوه تعامل و وابستگی آن‌ها را نشان می‌دهد.

مثال: یک رابطه می‌تواند بین عضو و امانت وجود داشته باشد که نشان می‌دهد یک عضو می‌تواند کتابی را امانت بگیرد.

۴. **نمودارهای دامنه:**

نمودارهای دامنه به صورت تصویری مفاهیم و روابط بین موجودیت‌ها را نمایش می‌دهند.

این نمودارها می‌توانند شامل نمودار کلاس‌ها (Class Diagrams) یا نمودارهای موجودیت-رابطه (ERD) باشند.

مراحل مدلسازی دامنه:

۱. **جمع‌آوری نیازمندی‌ها:**

مصاحبه با ذینفعان و کاربران برای درک نیازمندی‌های دامنه.

مستندسازی نیازمندی‌ها و مفاهیم کلیدی.

۲. **تحلیل و شناسایی موجودیت‌ها:**

تحلیل نیازمندی‌ها برای شناسایی موجودیت‌های اصلی و ویژگی‌های آن‌ها.

ایجاد فهرستی از موجودیت‌ها و ویژگی‌های مربوطه.

۳. **تعریف روابط:**

تعیین روابط بین موجودیت‌ها و نوع این روابط (یک به یک، یک به چند، چند به چند).

ترسیم نمودارهای دامنه برای نمایش روابط.

۴. **بررسی و اعتبارسنجی مدل:**

بازبینی مدل دامنه با ذینفعان و کاربران برای اطمینان از صحت و کامل بودن آن.

اعمال تغییرات و بهبودهای لازم بر اساس بازخوردها.

مزایای مدلسازی دامنه:

۱. **بهبود درک و ارتباطات:**

کمک به تحلیلگران، توسعه‌دهندگان و ذینفعان برای درک بهتر از سیستم و نیازمندی‌ها.

۲. **پایه‌ریزی صحیح طراحی:**

ایجاد پایه‌ای محکم برای طراحی و پیاده‌سازی سیستم.

۳. **کاهش ابهامات و خطاها:**

شفاف‌سازی نیازمندی‌ها و مفاهیم کلیدی که به کاهش ابهامات و خطاها کمک می‌کند.

در مدلسازی دامنه به دنبال تشخیص آن هستیم که چه چیزی درون سیستم است و چه چیزی بیرون و همچنین نحوه‌ی ارتباط درون و بیرون چگونه است؟

تشخیص اجزاء درونی سیستم شامل شناسایی و تعریف مؤلفه‌ها و عناصر مختلف سیستم است که به تحقق اهداف و نیازمندی‌های آن کمک می‌کنند.

فرض کنید می‌خواهیم اجزاء درونی یک *سیستم مدیریت فروشگاه اینترنتی* را تشخیص دهیم. در این سیستم، اجزاء مختلف ممکن است شامل موارد زیر باشند:

۱. **ماژول ثبت‌نام و احراز هویت کاربر:**

وظیفه: ثبت‌نام کاربران جدید و احراز هویت آن‌ها.

اجزاء: فرم ثبت‌نام، سیستم احراز هویت، بانک اطلاعاتی کاربران.

۲. **ماژول مدیریت محصولات:**

وظیفه: مدیریت و نمایش محصولات فروشگاه.

اجزاء: پایگاه داده محصولات، سیستم مدیریت محتوا، رابط کاربری نمایش محصولات.

۳. **ماژول سبد خرید و پرداخت:**

وظیفه: مدیریت سبد خرید کاربران و انجام تراکنش‌های پرداخت.

اجزاء: سبد خرید، سیستم پرداخت آنلاین، بانک اطلاعاتی تراکنش‌ها.

۴. **ماژول پشتیبانی و خدمات مشتری:**

وظیفه: ارائه خدمات پشتیبانی به مشتریان و پاسخ به سوالات آن‌ها.

اجزاء: سیستم تیکتینگ، پایگاه دانش، چت آنلاین.

تشخیص اجزاء بیرونی (External Components) سیستم نیز یک مرحله حیاتی در تحلیل و طراحی سیستم‌های اطلاعاتی است. اجزاء بیرونی به عناصری اشاره دارند که خارج از مرزهای سیستم هستند، اما با آنها تعامل دارند و بر عملکرد سیستم تأثیر می‌گذارند. شناسایی این اجزاء به تحلیلگران کمک می‌کند تا وابستگی‌ها و تعاملات خارجی را به خوبی درک کنند و برنامه‌های مناسب برای یکپارچه‌سازی و ارتباط با آن‌ها تدوین کنند.

فرض کنید می‌خواهیم اجزاء بیرونی یک *سیستم مدیریت فروشگاه اینترنتی* را تشخیص دهیم. در این سیستم، اجزاء بیرونی ممکن است شامل موارد زیر باشند:

۱. **درگاه پرداخت آنلاین:**

وظیفه: پردازش تراکنش‌های پرداخت کاربران.

رابط API :درگاه پرداخت برای ارسال و دریافت اطلاعات تراکنش.

۲. **سیستم ارسال و پست:**

وظیفه: مدیریت ارسال و تحویل سفارش‌ها به مشتریان.

رابط: ارتباط با سیستم پستی یا شرکت‌های حمل و نقل برای ارسال اطلاعات سفارش‌ها و پیگیری تحویل.

۳. **بانک اطلاعاتی کاربری:**

وظیفه: مدیریت اطلاعات کاربران و احراز هویت.

رابط: تبادل اطلاعات کاربری با سیستم‌های دیگر برای احراز هویت.

۴. **سیستم‌های تبلیغاتی و بازاریابی:**

وظیفه: مدیریت و اجرای کمپین‌های تبلیغاتی و بازاریابی.

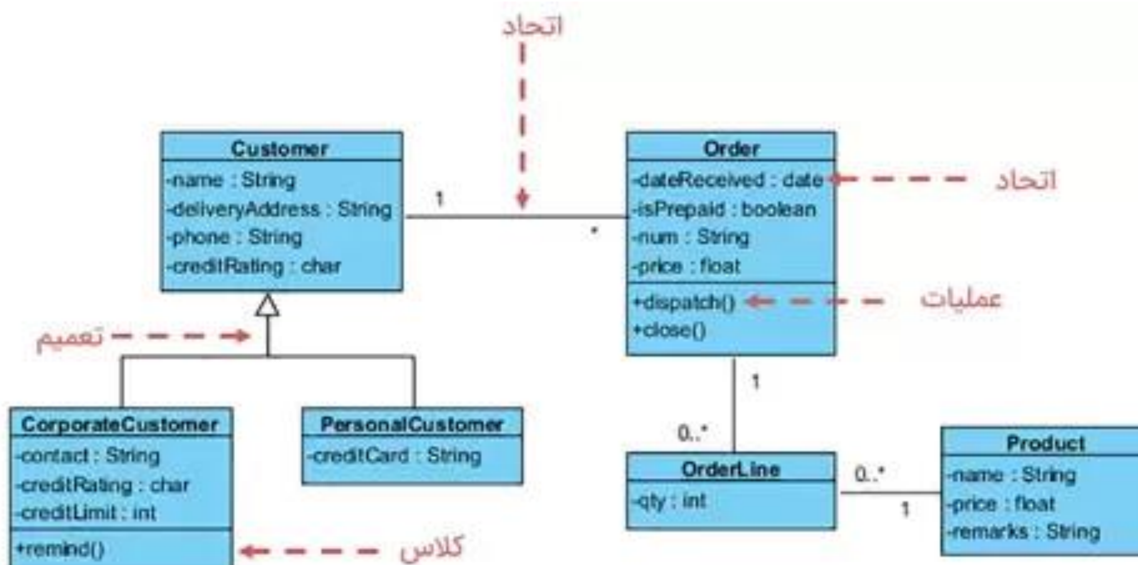
رابط: ارسال و دریافت داده‌های مرتبط با کمپین‌های تبلیغاتی و بازاریابی.

نمودار کلاس (Class Diagram)

نمودار کلاس در UML یک نمودار ایستا است و در واقع این نمودار دید ایستایی یک برنامه را نشان می‌دهد. Class Diagram برای تجسم، توصیف و یا مستند سازی و در نهایت ساخت کد اجرایی یک سیستم و یا نرم افزار کاربرد دارد. همچنین Class Diagram محدودیت های تحمیل شده بر یک سیستم را شرح میدهد. نمودارهای کلاس به طور گسترده ای در مدل سازی سیستم های شی گرا مورد استفاده قرار می گیرند زیرا آنها تنها نمودارهای UML هستند که می توانند مستقیماً با زبان های شی گرا نگاشت شوند. نمودار کلاس (Class Diagram)در UML مجموعه ای از کلاس ها ، رابط ها ، انجمن ها ، همکاری ها و محدودیت ها را نشان می‌دهد و به طور کلی یک نمودار ساختاری است.

دراین نمودار سه نوع رابطه اساسی مهم وجود دارد.

- **اتحاد (Association)** : نماینده روابطی بین وهله‌های انواع است (یک شخص که برای یک شرکت کار می‌کند، یک شرکت چند اداره دارد).
- **وراثت (Inheritance)** : بدیهی‌ترین افزودنی به نمودارهای ER برای استفاده در شیئ‌گرایی است. تناظر بی‌واسطه‌ای با وراثت در طراحی شیئ‌گرایی دارد.
- **تجمیع (Aggregation)** : شکلی از ترکیب‌بندی شیئ در طراحی شیئ‌گرا محسوب می‌شود.



کاربردClass Diagram

- نمودار کلاس برای مدل سازی نمای استاتیک (static view) یک سیستم استفاده می شود.
- نمای ایستا واژگان سیستم را توصیف می کند.
- نمودار کلاس به عنوان پایه و اساس نمودارهای اجزا (component) و استقرار (deployment) در نظر گرفته می شود.
- نمودار کلاس به وضوح نگاشت با زبان های شی گرا مانند Java ، C++ و ... را نشان می دهد،نمودار کلاس به طور کلی برای ساخت برنامه های کاربردی استفاده می شود.

اجزای یک کلاس

درUML ، یک کلاس در قالب یک مستطیل نمایش داده می شود که به سه قسمت تقسیم شده است، بالاترین قسمت مربوط به نام کلاس است. در قسمت میانی لیست صفات (attributes) کلاس آورده می شوند و در بخش پایینی نیز عملیات مربوط به آن لیست می شوند. در ادامه این سه قسمت را به طور مختصر شرح خواهیم داد:

۱- نام کلاس

در هنگام کشیدن نمودار باید توجه داشته باشید که بخش اول (نام کلاس) اجباری است و دو بخش دیگر می توانند بصورت اختیاری پر شوند. این دو بخش عموما در مواقعی که کامپوننت های سطح بالا را نشان می دهیم پر نمی گردند. چرا که تنها در این سطح ارتباطات بین انواع باید شخص گردد نه جزئیات آنها.

Flight ← نام
flightNumber : Integer departureTime : Date flightDuration : Minutes
delayFlight (numberOfMinutes : int) : Date getArrivalTime ( ) : Date

۲- لیست صفات یک کلاس

در بخش میانی یک کلاس که مربوط به نمایش صفات کلاس می باشد، هر صفت در یک خط مجزا آورده می‌شود. این بخش اختیاری است اما در صورتی که قصد نوشتن آن را دارید،دقت لازم داشته باشید.هر خط باید در قالب زیر نوشته شود:

flightNumber : Integer

departureTime : date

flightDuration : minute

در نمودارهای تجاری این صفات به گونه ای نوشته میشود که خوانندگی به راحتی بتوانند آن را درک کنند. اما در تعریف نوع صفات در نمودار کلاسی که قرار است از آن برای تولید کد استفاده شود، باید حتما از انواع داده معتبر در زبان های برنامه نویسی و یا انواعی از مدل هایی که بعدا قرار است در سیستم پیاده سازی شوند، استفاده گردد.

Flight
flightNumber : Integer departureTime : Date flightDuration : Minutes
delayFlight (numberOfMinutes : int) : Date getArrivalTime ( ) : Date ← عملیات

۳-لیست عملیات کلاس

عملیات های مربوط به کلاس ها در قسمت سوم نمودارها است و البته این بخش اختیاری است. برای مستند سازی عملیات از شیوه نشانه گذاری زیر استفاده می کنیم:

name(parameter list) : type of value returned

در هنگام مستند سازی عملیات مربوط به تابع می توان از علائمی استفاده نمود مبنی بر اینکه پارامتر ورودی یا خروجی است. این علائم اختیاری "in" و "out" می باشند. بطور معمول از این علائم در UML استفاده نمی شود مگر آنکه زبان برنامه نویسی ما زبانی قدیمی مانند Fortran باشد که در این صورت این اطلاعات می تواند بسیار مفید باشد. از آنجایی که در زبان های C++ و Java پارامتر های یک تابع از نوع ورودی هستند، بسیاری از افراد در UML از آوردن علائم in و out صرف نظر می کنند. البته در صورت تمایل می توانید از آنها استفاده کنید.

مورد کاربرد (Use Case)

مورد کاربرد (UC) ابزاری برای تعریف تعاملات مورد نیاز کاربر در سیستم است، در واقع مجموعه اقداماتی است که مرحله به مرحله تعاملات بین کاربر و سیستم را برای رسیدن به یک هدف خاص (که همان کامل شدن مورد یا کیس است) تعریف می کند. کیس را می توان یک کار در نظر گرفت که باید تکمیل شود.

در توسعه نرم افزار Use Cases یا مجموعه ای ازCase ها (کیس ها) نوشته می شود. در تصویر زیر هر کدام از بیضی ها یک کیس را نشان می دهد و کاربری که با سیستم از طریق این کیس ها ارتباط برقرار می کند.

چه کسانی از مستندات "Use Case" استفاده می‌کنند؟

مستندات Use Case یک نمای کامل از مسیرهای مختلف تعاملات کاربر با سیستم برای رسیدن به هدف ارائه می دهد. هرچه اسناد بهتری نوشته شود، شناسایی نیازمندی های یک سیستم نرم افزاری بسیار ساده تر خواهد شد.

توسعه دهندگان نرم افزار، آزمایش کنندگان نرم افزار و همچنین سایر گروه های ذی نفع می توانند از این مستندات استفاده کنند.

موارد استفاده از مستندات:Use Case

- توسعه دهندگان برای پیاده سازی کد و طراحی آن
- آزمایش کنندگان برای ایجاد کیس های آزمایشی
- ذی نفعان تجاری برای درک نیازهای نرم افزار

کنشگرها(Actors)

- **اکتورها یا کنشگرها شروع کننده یک فعالیت در سیستم مورد نظر هستند**-شما در ابتدا باید کنشگرهای مرتبط با کسب و کار مد نظرتان را نامگذاری کنید. به عنوان مثال اگر یوزکیس شما با یک سازمان خارجی تعامل دارد بهتر است به جای استفاده از نام آن سازمان برای یوزکیس، عملکرد آن سازمان را به عنوان اسم به کار ببرید. برای مثال Airline Company بهتر از PanAir است.
- **انواع اکتورها عبارت اند از:**

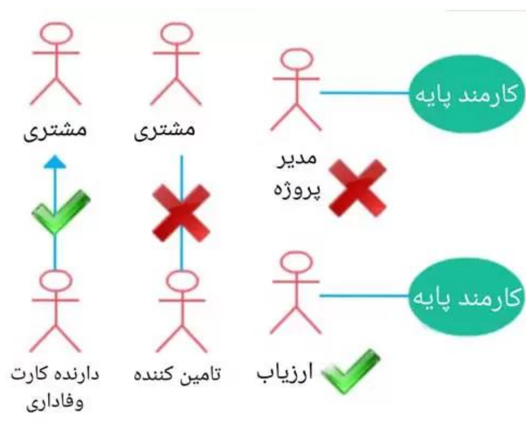
۱) افراد – اشخاص

۲) سیستم نرم افزاری دیگر که به صورت برخط درخواست ارسال می‌کند / بخشی دیگر از نرم‌افزار خودمان

۳) زمان (مثلا پشتیبان گری یا ریکاوری و سیو خودکار اطلاعات) – کارهایی که بدون دخالت هیچ فردی در یک دوره‌ی تناوب انچام می‌شوند.

- **اکتور یا کنشگرهای اولیه (Primary Actors) باید در سمت چپ نمودار شما قرار بگیرند** –این کار باعث می شود تا به سرعت بتوانید نقش های مهم و کلیدی موجود در سیستم را برجسته کنید.
- **نقش کنشگرها (نه موقعیت هایشان) را عنوان کنید** – به عنوان مثال، در یک هتل هم مدیر دفتر و هم مدیر شیفت می توانند کار رزرو کردن را انجام بدهند. بنابراین با استفاده از نامی مثل "مأمور رزرو" باید نقش هر کنشگر در سیستم را مشخص نمایید.
- **سیستم های خارجی کنشگر هستند**-اگر یوزکیس شما ارسال ایمیل است و با نرم افزار مدیریت ایمیل تعامل دارید، این نرم افزار یک کنشگر برای یوزکیس خاص شما محسوب می شود.

- **کنشگرها با یکدیگر تعامل ندارند** – در صورتی که کنشگرهای یک سیستم با هم تعامل دارند، باید یک یوزکیس دیاگرام جدید ایجاد کنید که سیستم ارائه شده در یوزکیس دیاگرام قبلی را به عنوان یک کنشگر نشان بدهد.
- **کنشگرهای ارث بری شده** (inheriting actors) **را زیر کنشگرهای والد** (parent actors) **قرار بدهید** - این کار برای خوانایی بیشتر و برجسته کردن سریع موارد کاربرد خاص برای هر کنشگر انجام می شود.

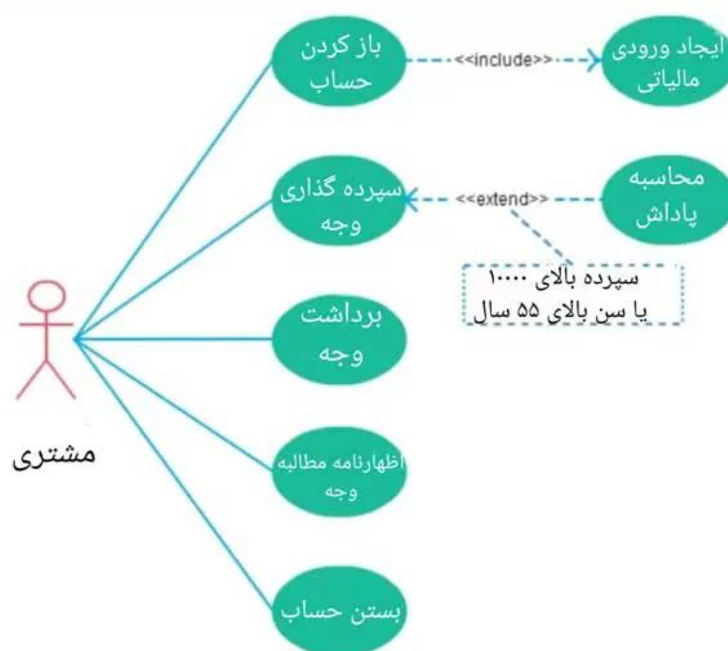


برخی از مواردی که باید هنگام ایجاد کنشگر در یوزکیس در نظر گرفت.

یوزکیس ها(Use Cases)

- مجموعه‌ای مرتب و پایان‌پذیر از فعالیت‌های نرم‌افزاری که به صورت یکپارچه (یک‌تکه) هستند.
- **نام یوزکیس ها با یک فعل شروع می شود** -یک یوزکیس بیان کننده عمل یک مدل یا سیستم است، بنابراین اسم آن باید با یک فعل شروع شود.
- **نام یوزکیس را توصیفی انتخاب کنید** -توصیفی کردن اسم یوزکیس باعث می شود افرادی که به نمودار شما نگاه می کنند اطلاعات بیشتری از آن دریافت کنند. به عنوان مثال نام "چاپ فاکتور" بهتر از "چاپ" است.

- **ترتیب منطقی یوزکیس ها را برجسته کنید** -برای مثال، اگر در حال تجزیه و تحلیل اطلاعات برای یک مشتری بانک هستید، یوزکیس های معمول شما شامل باز کردن حساب، سپرده گذاری و برداشت می شود. موقع ترتیب بندی این موارد، آنها را به شکلی معقول و منطقی بچینید.
- **یوزکیس های اینکلود یا شامل شده** (included use cases) **را در سمت راست یوزکیس قرار بدهید** -این کار برای بهبود خوانایی و افزایش وضوح انجام می شود.
- **یوزکیس های ارث بری شده** (inheriting use case) **را زیر یوزکیس های منبع یا والد** (parent use case) **قرار بدهید** -این کار هم برای افزایش خوانایی و فهم نمودار شما انجام می شود.



هنگام ترسیم یوزکیس باید به چند نکته توجه کرد:

منابع استخراج موردهای کاربرد (UC)

۱) فرآیندهای سازمانی (فعالیت های انجام شده در یک سازمان)

۲) نیازمندی‌ها

۳)رفتار Actorها

۴) کلاس های Domain Modeling

۵) توابع – CRUD چهار عملیات اصلی است که بر روی داده‌ها انجام می‌شود: ایجاد(Create) ، خواندن(Read) ، به‌روزرسانی(Update) ، و حذف .(Delete)

۶) گزارش های نرم افزاری

۷) فرم های نرم افزاری

ارتباطات(relationships)

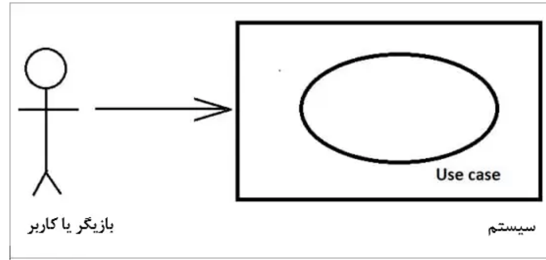
- هنگام استفاده از <<extend>>، پیکان یا فلش به یوزکیس مبنا اشاره می کند.
- <<extend>>می تواند شرایط بسط اختیاری داشته باشد.
- <<extend>>و <<include>>هر دو به صورت فلش های نقطه چین دار نشان داده می شوند.
- رابطه اکتور و یوزکیس فلش ها را نشان نمی دهد.

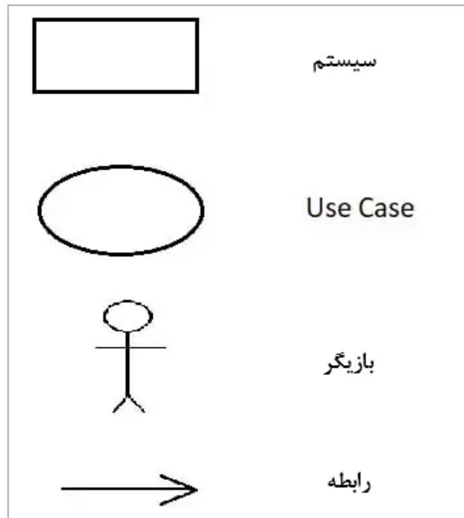
سیستم ها یا پکیج ها(Systems / Packages)

- از سیستم ها یا پکیج ها خیلی کم و فقط در موارد ضروری استفاده کنید.
- برای اشیاء (objects) نام های معنی دار و توصیفی انتخاب کنید.

نمودارUse Case

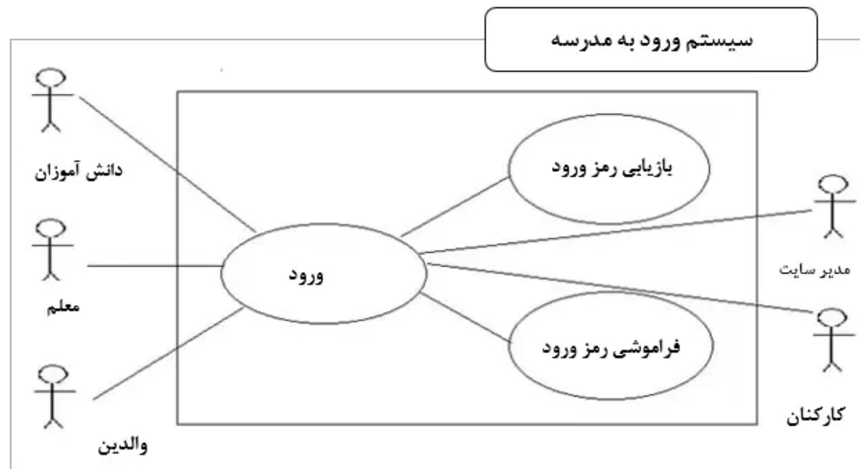
نمودار Use Case یک نمایش تصویری از اقدامات کاربر(ها) در سیستم است. اگر یوزکیس بازیگران زیادی داشته باشد، نمودار تصویری یک ابزار عالی در این زمینه فراهم می کند که درک یوزکیس را بسیار آسان می کند. در نمودار سطح بالا جزئیات زیادی وجود ندارد و ایده های پیچیده را به روش کاملاً ساده نشان می دهد.





در نمودار بالا شکل های مستطیل، بیضی، فلش و فرد به ترتیب سیستم، Use Case، رابطه و بازیگر یا کاربر را نشان می دهند. یک نمودار سیستم یا نرم افزار را نشان می دهد، سازمان یا افرادی که با سیستم تعامل دارند و همچنین جریان اصلی اینکه " سیستم چه کاری انجام می دهد؟" را نشان می دهد.

مثال : نمودار Use case ورود به سیستم



- این نمودار Use caseمثال قبل برای "ورود به سیستم" است. در اینجا بیشتر از یک بازیگر داریم و همه آنها خارج از سیستم قرار می گیرند. دانش آموزان، معلم و والدین به عنوان بازیگران اصلی یا اولیه در نظر گرفته شده اند و به همین دلیل همه آنها در سمت چپ مستطیل قرار می گیرند.

- کارکنان و مدیر سایت به عنوان بازیگران ثانویه در نظر گرفته شده اند و در سمت راست مستطیل قرار می گیرند. بازیگران می توانند به سیستم وارد شوند بنابراین بازیگران و کیس با رابط به هم وصل می شوند.
- سایر عملکردهای موجود در سیستم مانند بازیابی رمز ورود و فراموشی رمز ورود. مرتبط به کیس ورود به سیستم می باشند و به همین دلیل به کیس وصل شده اند.

چند نکته در مورد تفاوت فرآیند سازمانی و موردکاربرد (Use Case) :

۱) "فرآیند سازمانی" در دنیای سیستم موجود (کسب وکار مورد نظر) است ولی **"موردکاربرد"** در دنیای نرم‌افزاری.

۲) "فرآیند سازمانی" به صورت یکپارچه است ولی **"موردکاربرد"** به طور معمول به صورت یکپارچه.

۳) یک **"موردکاربرد"** بخشی از یک فرآیند سازمانی است، به عبارتی دیگر یک فرآیند سازمانی معمولاً (نه همیشه) به چند **"موردکاربرد"** تبدیل می‌شود.

بررسی موردکاربرد (UC) از دیدگاه پیچیدگی

از دیدگاه پیچیدگی UC ها را می‌توان به ۲ دسته پیچیده و ساده، تقسیم کرد. این تقسیم‌بندی به تحلیلگران و توسعه‌دهندگان کمک می‌کند تا اولویت‌بندی بهتری داشته باشند و منابع را به طور مؤثرتر مدیریت کنند. UCهای پیچیده، UCهایی هستند که شامل تعاملات و فرآیندهای کلیدی و حیاتی سیستم هستند و نیازمند تحلیل و طراحی دقیق‌تری می‌باشند. برای تشخیص پیچیدگی به معیارهای زیر می‌توان توجه کرد.

معیارهای مرتبط با توسعه و پیاده‌سازی:

1. **تعداد خط‌های کد مورد نیاز:**

- هرچه تعداد خط‌های کد مورد نیاز برای پیاده‌سازی یوزکیس بیشتر باشد، پیچیدگی آن بالاتر است.
- مثال: یوزکیس‌های پیچیده‌تر نیاز به کدهای بیشتری برای پوشش تمامی سناریوها و شرایط دارند.

۲. **پیچیدگی الگوریتم:**

- یوزکیس‌هایی که نیاز به الگوریتم‌های پیچیده دارند، پیچیدگی بالاتری دارند.
- مثال: الگوریتم‌های مرتب‌سازی یا پردازش داده‌های بزرگ.

معیارهای مرتبط با تحلیل و طراحی:

۳. **تازگی و نو بودن:**

- یوزکیس‌های جدید و نوآورانه که قبلاً تجربه نشده‌اند، معمولاً پیچیدگی بیشتری دارند.
- مثال: یوزکیسی که برای یک فناوری یا فرآیند کاملاً جدید طراحی شده باشد.

٤. **پیچیدگی محاسباتی:**

- یوزکیس‌هایی که شامل محاسبات پیچیده و تحلیل‌های عددی هستند، پیچیدگی بیشتری دارند.
- مثال: محاسبات مالی یا علمی پیچیده.

٥. **پیچیدگی فرم:**

- یوزکیس‌هایی که شامل فرم‌ها و رابط‌های کاربری پیچیده و چندمرحله‌ای هستند، پیچیدگی بیشتری دارند.
- مثال: فرم‌های چندگانه با داده‌های وابسته و تعاملات پیچیده.

٦. **پیچیدگی فرآیندهای تجاری(Business Process Complexity) :**

- یوزکیس‌هایی که شامل فرآیندهای تجاری پیچیده و طولانی هستند، پیچیدگی بیشتری دارند.
- مثال: فرآیندهای چندمرحله‌ای کسب‌وکار با تداخلات متعدد.

معیارهای مرتبط با تعاملات و وابستگی‌ها:

۷. **تعداد بازیگران بیشتر:**

- یوزکیس‌هایی که با تعداد بیشتری از بازیگران تعامل دارند، پیچیده‌تر هستند.
- مثال: یوزکیس‌هایی که نیاز به هماهنگی بین چندین کاربر یا سیستم دارند.

۸. **روابط پیچیده:**

- یوزکیس‌هایی که شامل روابط و وابستگی‌های متعددی بین موجودیت‌ها هستند، پیچیدگی بیشتری دارند.
- مثال: یوزکیس‌های با وابستگی‌های شدید بین موجودیت‌ها و داده‌ها.

۹. **سناریوهای متعدد:**

- یوزکیس‌هایی که دارای سناریوهای اصلی و فرعی زیادی هستند، پیچیدگی بیشتری دارند.
- مثال: یوزکیسی با چندین جریان جایگزین و استثنا.

۱۰. **نیازهای غیرعملیاتی مهم:**

- یوزکیس‌هایی که نیازهای غیرعملیاتی سخت‌گیرانه‌تری مانند امنیت، عملکرد و قابلیت اعتماد دارند، پیچیده‌تر هستند.
- مثال: یوزکیس‌های با نیازمندی‌های امنیتی بالا یا زمان پاسخ‌دهی کوتاه.

برای مستندسازی UCهای ساده اجزاء زیر را ارائه خواهیم کرد.

۱. نام UC
۲. کد UC
۳. توضیحات (Note) : شرح روند اجرا
۴. فرم ها (Form) : شامل نام فرم، کد فرم، شکل ظاهری فرم، فیلدها، اعتبارسنجی‌های قابل استفاده
۵. لیست اکتورها (Actor)
۶. با کدام کلاس Domain Model چه ارتباطی برقرار می‌کند.

اما برای UCهای پیچیده علاوه بر اجزایی که برای UC ساده معرفی شدند، موارد زیر نیز باید بکار گرفته شوند.

۷. پیش‌شرط‌ها (Preconditions):

شرایطی که باید قبل از آغاز یوزکیس برآورده شوند. مثال: کاربر باید به سیستم وارد شده باشد، پیش‌شرط‌های امانت کتاب و ...

۸. پس‌شرط‌ها (Postconditions):

شرایطی که پس از اتمام یوزکیس باید برآورده شوند. همچنین کارهایی که با محقق شدن شرایطی باید به اجرا برسند. برای مثال، کاربر باید با موفقیت ثبت‌نام شده باشد.

۹. سناریوی UC

در اینجا دیدگاه ما نرم‌افزاری است. جریان اجرای UC با یک جریان اصلی (پایه - Basic) و جریان‌های جایگزین (Alternative)

مثال عملی

فرض کنید می‌خواهیم یوزکیس "امانت گرفتن کتاب" را مستندسازی کنیم:

۱. **نام یوزکیس :**امانت گرفتن کتاب

۲. **شناسه یوزکیس:** UC-003

۳. **بازیگران(Actor):** کاربر، سیستم کتابخانه

۴. **شرح مختصر:** این یوزکیس فرآیند امانت گرفتن یک کتاب توسط کاربر را شرح می‌دهد.

۵. **جریان اصلی:**

   ۱. کاربر به سیستم وارد می‌شود.
   ۲. کاربر کتاب مورد نظر خود را جستجو می‌کند.
   ۳. کاربر کتاب را برای امانت انتخاب می‌کند.
   ۴. سیستم کتاب را برای کاربر رزرو می‌کند.
   ۵. سیستم پیام تأیید امانت را نمایش می‌دهد.

۶. **جریان‌های فرعی:**

- **کتاب موجود نیست:**

۱. سیستم پیام عدم موجودی کتاب را نمایش می‌دهد.

۲. کاربر کتاب دیگری را جستجو می‌کند.

۷. **پیش‌شرط‌ها:** کاربر باید وارد سیستم شده باشد.

۸. **پس‌شرط‌ها:** کتاب باید برای کاربر رزرو شده باشد.

۹. **نیازمندی‌های خاص:** عملیات امانت گرفتن باید در کمتر از ۵ ثانیه انجام شود.

۱۰. **روابط:** این یوزکیس شامل یوزکیس "ورود به سیستم" است.

۱۱. **نمودار یوزکیس:** نمودار نشان‌دهنده تعاملات کاربر و سیستم برای امانت گرفتن کتاب.

معماری نرم افزار (Software Architecture)

منظور از معماری نرم افزار سازماندهی کردن سازه های مختلف نرم افزاری بر اساس اصول و قواعدی است که به رشد و طول عمر یک سیستم نرم‌افزاری کمک کند. هر سازه نرم‌افزاری از عناصر مختلفی تشکیل شده است که اغلب با یکدیگر رابطه دارند و در روند این ارتباط رسالت کامل یک سیستم نرم‌افزاری رقم می خورد. منظور از معماری نرم افزار همان معماری ای هست که برای ساختن یک ساختمان نیز به آن نیاز داریم. به عبارت دیگر، معماری نرم افزار شبیه به یک نقشه عمل می‌کند که متخصصین نرم‌افزار برای پیاده سازی کردن نیازمندی های مربوط به پروژه از آن استفاده کرده و نرم‌افزار را بر اساس آن پیاده سازی می کنند. انتخاب های مربوط به تکنولوژی های مختلف نرم افزاری و نحوه پیاده سازی کردن آنها با حداقل هزینه بخشی از اتفاقی است که در روند معماری نرم‌افزار رخ می دهد. استفاده کردن از سیستم های مختلف به منظور پیاده‌سازی کردن هرچه بهتر نیازمندی‌ها جزء وظایفی است که می بایست در روند معماری نرم افزار پیاده سازی بشود.

به شکل خلاصه: معماری نرم‌افزار، ساختار کلی یک سیستم نرم‌افزاری است که شامل اجزاء نرم‌افزار، روابط و تعاملات بین این اجزاء، و اصول و الگوهای طراحی است که برای ایجاد این ساختار به کار رفته‌اند. هدف اصلی معماری نرم‌افزار، تضمین این است که سیستم به طور کارآمد، قابل نگهداری و مقیاس‌پذیر ساخته شود و نیازمندی‌های عملیاتی و غیر عملیاتی را برآورده کند.

تفاوت معماری نرم‌افزار و طراحی نرم‌افزار : معماری نرم افزار؛ اغلب بررسی ساختار کلان‌تر و اساسی تر یک سیستم نرم‌افزاری و کار کردن با فرآیندهایی می باشد که با همکاری یکدیگر، وظایف یک نرم افزار را انجام می دهند. و منظور از طراحی نرم افزار و یا همان software design بررسی ساختارهای کوچکتر و ریزتر و همچنین بررسی طراحی داخلی یک فرآیند نرم افزاری تک می باشد.

عناصر کلیدی معماری نرم‌افزار

۱. **اجزاء** (Components)

واحدهای اصلی سیستم که وظایف خاصی را انجام می‌دهند. مثال: ماژول‌ها، سرویس‌ها، کلاس‌ها.

۲. **اتصالات** (Connectors)

نحوه تعامل و ارتباط بین اجزاء سیستم. مثال: پروتکل‌های ارتباطی، واسط‌های برنامه‌نویسی.(APIs)

۳. **پیکربندی** (Configuration)

ترتیب و چیدمان اجزاء و اتصالات در سیستم.مثال: چگونگی قرارگیری ماژول‌ها و نحوه ارتباط آن‌ها با یکدیگر.

اهداف معماری نرم‌افزار

۱. **مدیریت پیچیدگی:** کاهش پیچیدگی سیستم از طریق تجزیه آن به اجزاء کوچکتر و مدیریت‌پذیرتر.

۲. **قابلیت نگهداری و توسعه:** ایجاد ساختاری که به راحتی قابل نگهداری و ارتقاء باشد.

۳. **عملکرد و کارایی:** بهینه‌سازی ساختار سیستم برای دستیابی به عملکرد بالا و کارایی بهتر.

٤. **مقیاس‌پذیری:** توانایی سیستم برای مقیاس‌پذیری به میزان بار کاری و درخواست‌های بیشتر.

٥. **امنیت:** تضمین حفاظت از داده‌ها و جلوگیری از دسترسی‌های غیرمجاز.

٦. **انعطاف‌پذیری:** امکان پاسخگویی به تغییرات سریع و ناپیوسته در نیازمندی‌ها و شرایط محیطی.

مثال عملی از معماری نرم‌افزار

فرض کنید در حال طراحی یک سیستم مدیریت کتابخانه هستیم. معماری این سیستم ممکن است شامل اجزاء زیر باشد.

- **ماژول کاربران:** مدیریت ثبت‌نام و ورود کاربران.
- **ماژول کتاب‌ها:** مدیریت اطلاعات کتاب‌ها و جستجو.
- **ماژول امانت:** مدیریت فرآیند امانت دادن و بازگرداندن کتاب‌ها.
- **واسط‌های ارتباطیAPI**های لازم برای ارتباط بین ماژول‌ها و سیستم‌های خارجی.

گروه معماری

گروه معماری به تیمی از متخصصان اشاره دارد که وظیفه دارند ساختار کلی سیستم نرم‌افزاری را طراحی کنند، تصمیمات فنی کلیدی را اتخاذ کنند و اطمینان دهند که سیستم نهایی نیازهای کاربران و ذینفعان را به درستی برآورده می‌کند. این گروه شامل نقش‌های مختلفی است که هر یک مسئولیت‌ها و وظایف خاصی در فرآیند طراحی و پیاده‌سازی سیستم دارند.

وظایف گروه معماری نرم‌افزار

در فرآیند تحلیل و طراحی نرم‌افزار، گروه معماری نرم‌افزار وظایف کلیدی و حیاتی را بر عهده دارند که تضمین می‌کند سیستم به درستی طراحی و پیاده‌سازی شود. این وظایف شامل موارد زیر است:

۱. **تعریف و مستندسازی معماری**

ایجاد و مستندسازی ساختار کلی سیستم، شامل اجزاء، اتصالات و پیکربندی‌ها.

اقدامات: تهیه مستندات معماری که توصیف کننده اجزاء سیستم، نحوه تعامل آن‌ها و اصول و الگوهای طراحی هستند.

۲. انتخاب الگوهای معماری

انتخاب الگوهای معماری مناسب که به نیازمندی‌های سیستم پاسخ دهند.

اقدامات: بررسی و ارزیابی الگوهای معماری مختلف مانند معماری لایه‌ای، میکروسرویس‌ها، سرویس‌گرا و رویدادمحور، و انتخاب بهترین گزینه.

۳. تصمیم‌گیری‌های فنی

اتخاذ تصمیمات فنی کلیدی درباره فناوری‌ها، ابزارها و چارچوب‌های مورد استفاده در سیستم.

اقدامات: ارزیابی و انتخاب فناوری‌ها و ابزارهایی که با نیازهای سیستم و اهداف معماری همخوانی دارند.

۴. مدیریت پیچیدگی

تجزیه و تحلیل سیستم‌های پیچیده به اجزاء کوچکتر و مدیریت‌پذیرتر.

اقدامات :تعریف اجزاء مستقل و مدیریت ارتباطات بین آن‌ها به منظور کاهش پیچیدگی کلی سیستم.

۵. ایجاد نمونه‌های اولیه(Prototypes)

توسعه نمونه‌های اولیه برای آزمایش مفاهیم و ایده‌های معماری.

اقدامات: ایجاد نمونه‌های اولیه از اجزاء کلیدی سیستم و ارزیابی کارایی و قابلیت‌های آن‌ها.

۶. مدیریت نیازمندی‌های غیرعملیاتی:

تضمین اینکه نیازمندی‌های غیرعملیاتی مانند امنیت، عملکرد، مقیاس‌پذیری و قابلیت اعتماد در طراحی معماری لحاظ شوند.

اقدامات :شناسایی و مستندسازی نیازمندی‌های غیرعملیاتی و اتخاذ تدابیر مناسب برای برآورده‌کردن آن‌ها.

۷. راهنمایی تیم توسعه:

ارائه راهنمایی و پشتیبانی به تیم توسعه برای اطمینان از پیاده‌سازی صحیح معماری.

اقدامات :همکاری نزدیک با تیم توسعه، ارائه دستورالعمل‌ها و استانداردها و نظارت بر پیشرفت کار.

۸. **بررسی و ارزیابی معماری:**

ارزیابی معماری‌های پیشنهادی و تأیید اینکه معماری نهایی نیازمندی‌های سیستم را برآورده می‌کند.

اقدامات :انجام جلسات بررسی معماری، شناسایی نقاط ضعف و فرصت‌های بهبود و تأیید معماری نهایی.

۹. **مستندسازی و ارتباطات**

تضمین اینکه تمامی جزئیات معماری به درستی مستند شده و بین تمامی ذینفعان به اشتراک گذاشته شود.

اقدامات :تهیه و به‌روزرسانی مستندات معماری و برگزاری جلسات و کارگاه‌های آموزشی برای توضیح معماری به تیم‌های مختلف.

الگوی معماری نرم‌افزار(Software Architecture Pattern)

الگوی معماری نرم‌افزار یک الگوی طراحی کلی و تکرارپذیر است که ساختار سیستم نرم‌افزاری را تعریف می‌کند. این الگوها به توسعه‌دهندگان کمک می‌کنند تا سیستم‌های پیچیده را به اجزای کوچکتر و مدیریت‌پذیرتر تقسیم کنند و بهترین روش‌های طراحی را به کار بگیرند.

الگوهای معماری نرم‌افزار به چارچوب‌ها و ساختارهایی اشاره دارد که برای طراحی و سازماندهی اجزاء سیستم‌های نرم‌افزاری استفاده می‌شوند. این الگوها به توسعه‌دهندگان کمک می‌کنند تا سیستم‌های قابل نگهداری، مقیاس‌پذیر و کارآمد ایجاد کنند. در ادامه به معرفی برخی از الگوهای معماری معروف می‌پردازیم:

۱. معماری لایه‌ای(Layered Architecture)

سیستم به لایه‌های مختلفی تقسیم می‌شود که هر لایه وظایف خاصی را انجام می‌دهد و با لایه‌های دیگر تعامل دارد. **مثال :**لایه‌های نمایش، منطق کسب‌وکار، دسترسی به داده‌ها و پایگاه داده.

۲. معماری میکروسرویس‌ها(Microservices Architecture)

سیستم به سرویس‌های کوچک و مستقل تقسیم می‌شود که هر سرویس وظایف خاصی را انجام می‌دهد و به طور مستقل قابل توسعه و نگهداری است. **مثال :**سرویس‌های مربوط به کاربر، سفارش، پرداخت.

۳. معماری سرویس‌گرا(Service-Oriented Architecture - SOA)

استفاده از سرویس‌های مستقل که از طریق پروتکل‌های استاندارد با یکدیگر تعامل دارند. **مثال :**سرویس‌های وب مبتنی بر SOAP یا REST که برای تعامل بین سیستم‌های مختلف استفاده می‌شوند.

۴. معماری رویداد محور(Event-Driven Architecture)

اجزاء سیستم از طریق رویدادها با یکدیگر تعامل دارند و تغییرات را از طریق رویدادها انتقال می‌دهند. **مثال :** استفاده از پیام‌ها و صف‌ها برای ارسال و دریافت رویدادها در یک سیستم مدیریت سفارشات.

۵. معماری لوله و فیلتر(Pipes and Filters Architecture)

سیستم به مجموعه‌ای از اجزاء پردازشی (فیلترها) تقسیم می‌شود که داده‌ها را پردازش می‌کنند و از طریق کانال‌ها (لوله‌ها) به یکدیگر متصل می‌شوند. **مثال :**پردازش داده‌های ورودی از طریق چندین مرحله فیلتر برای تجزیه و تحلیل و تغییر شکل داده‌ها.

۶. معماری ناحیه‌ای(Modular Architecture)

سیستم به ماژول‌های مستقل و قابل ترکیب تقسیم می‌شود که هر ماژول وظایف خاصی را انجام می‌دهد. **مثال :** ماژول‌های مربوط به کاربران، مدیریت محتوا، پرداخت و غیره در یک سیستم مدیریت محتوا.**(CMS)**

۷. معماری مبتنی بر لایه‌های ارجاع(Reference Layers Architecture)

استفاده از لایه‌های ارجاعی به منظور ساده‌سازی دسترسی به اجزاء مختلف سیستم و کاهش وابستگی‌ها. **مثال :** استفاده از لایه‌های ارجاعی برای مدیریت دسترسی به سرویس‌ها و اجزاء مختلف در یک سیستم پیچیده.

۸. معماری کلاینت-سرور(Client-Server Architecture)

سیستم به دو بخش کلاینت و سرور تقسیم می‌شود که کلاینت‌ها درخواست‌هایی را به سرور ارسال می‌کنند و سرور آن‌ها را پردازش و پاسخ می‌دهد. **مثال :**وب‌اپلیکیشن‌های معمولی که کلاینت از طریق مرورگر وب با سرور تعامل دارد.

پیمانه‌بندی یا ماژولاریتی (Modularity)

یکی از اصول اساسی در معماری نرم‌افزار است که به معنای تقسیم سیستم به اجزاء کوچک‌تر و مستقل‌تر است. این اجزاء یا ماژول‌ها هر کدام وظایف خاصی را انجام می‌دهند و به طور مستقل توسعه، نگهداری و بهبود می‌یابند. ماژولاریتی کمک می‌کند که سیستم نرم‌افزاری بهتر سازماندهی شود و مدیریت آن آسان‌تر گردد. در ادامه به بررسی جزئیات و اهمیت ماژولاریتی در معماری نرم‌افزار می‌پردازیم:

اصول ماژولاریتی:

۱. **انسجام**(Cohesion)

هر ماژول باید وظایف مرتبط و متجانسی را انجام دهد. انسجام بالا نشان‌دهنده کیفیت خوب طراحی ماژولار است، زیرا تمام عملکردهای یک ماژول به یکدیگر مرتبط هستند.

۲. **کاهش وابستگی**(Low Coupling)

ماژول‌ها باید تا حد امکان از یکدیگر مستقل باشند و وابستگی کمی بین آن‌ها وجود داشته باشد. کاهش وابستگی بین ماژول‌ها باعث افزایش انعطاف‌پذیری و نگهداری بهتر می‌شود.

۳. **رابط‌های مشخص**(Well-defined Interfaces)

هر ماژول باید دارای رابط‌های مشخصی باشد که نحوه تعامل آن با سایر ماژول‌ها را تعریف کند. این رابط‌ها باید پایدار و به خوبی مستند شده باشند.

سیستم‌های ماژولار معمولاً بهتر مدیریت می‌شوند. دلایل متعددی برای این امر وجود دارد که به تفصیل توضیح می‌دهم:

۱. کاهش پیچیدگی

با تقسیم سیستم به ماژول‌های کوچکتر، پیچیدگی کلی سیستم کاهش می‌یابد. هر ماژول به صورت مستقل وظایف خاصی را انجام می‌دهد. توسعه‌دهندگان می‌توانند به جای مدیریت یک سیستم پیچیده، به مدیریت و نگهداری ماژول‌های کوچکتر و ساده‌تر بپردازند.

۲. قابلیت نگهداری و به‌روزرسانی

ماژول‌های مستقل به راحتی می‌توانند به‌روزرسانی و نگهداری شوند بدون اینکه نیاز به تغییرات گسترده در سایر بخش‌های سیستم باشد. این امر باعث کاهش هزینه‌ها و زمان لازم برای نگهداری و به‌روزرسانی سیستم می‌شود.

۳. استفاده مجدد از کد

ماژول‌های مستقل می‌توانند در پروژه‌ها و سیستم‌های دیگر نیز مورد استفاده قرار گیرند که باعث صرفه‌جویی در زمان و هزینه توسعه می‌شود. توسعه‌دهندگان می‌توانند از ماژول‌های موجود در پروژه‌های جدید استفاده کنند و نیازی به نوشتن کد جدید نداشته باشند.

۴. توسعه همزمان

امکان توسعه همزمان ماژول‌های مختلف توسط تیم‌های مختلف وجود دارد که باعث تسریع در فرآیند توسعه نرم‌افزار می‌شود. تیم‌های توسعه می‌توانند به صورت موازی روی ماژول‌های مختلف کار کنند بدون اینکه منتظر تکمیل بخش‌های دیگر سیستم باشند.

۵. انعطاف‌پذیری

با ماژولار بودن سیستم، تغییرات و بهبودها به راحتی در یک ماژول اعمال می‌شوند بدون اینکه به سایر بخش‌های سیستم تأثیر بگذارد. این امر باعث افزایش انعطاف‌پذیری سیستم و کاهش ریسک تغییرات می‌شود.

۶. تست و اعتبارسنجی آسان‌تر

ماژول‌های مستقل به صورت جداگانه تست و اعتبارسنجی می‌شوند که باعث افزایش دقت و کیفیت تست‌ها می‌شود. توسعه‌دهندگان می‌توانند تست‌های خودکار و دستی را بر روی هر ماژول به صورت جداگانه اجرا کنند و از صحت عملکرد هر بخش اطمینان حاصل کنند.

۷. قابلیت تغییرپذیری(Changeability)

توانایی سیستم در پذیرفتن تغییرات با کمترین تلاش ممکن. امکان اعمال تغییرات سریع و بدون نیاز به بازنویسی گسترده کد، افزایش انعطاف‌پذیری و کاهش زمان و هزینه نگهداری.

۸.تعویض‌پذیری(Replaceability)

توانایی جایگزینی یک ماژول با یک ماژول دیگر با عملکرد مشابه بدون نیاز به تغییرات گسترده در سایر بخش‌های سیستم . کاهش ریسک، بهبود مستمر، و افزایش انعطاف‌پذیری.

برتراند مایر، یکی از پیشگامان مهندسی نرم‌افزار، نظرات مهمی در مورد ماژولاریتی ارائه داده است. او معتقد است که ماژولاریتی یکی از اصول کلیدی در طراحی و توسعه نرم‌افزار است که به بهبود کیفیت و قابلیت نگهداری سیستم‌ها کمک می‌کند. مایر در کتاب خود، "Object-Oriented Software Construction"، پنج معیار اصلی برای ماژولاریتی را معرفی کرده است:

معیارهای ماژولاریتی مایر

۱.قابلیت تجزیه(Decomposability)

سیستم باید به اجزاء کوچکتر و مستقل تقسیم شود. این ماژول‌ها باید به گونه‌ای طراحی شوند که هر کدام وظایف خاصی را انجام دهند.

مزیت: تجزیه سیستم به اجزاء کوچکتر باعث کاهش پیچیدگی و بهبود قابلیت نگهداری و فهم سیستم می‌شود.

۲. قابلیت ترکیب(Composability)

اجزاء یا ماژول‌های سیستم باید به گونه‌ای طراحی شوند که بتوانند به صورت مجدد ترکیب شوند و سیستم‌های جدید را ایجاد کنند.

مزیت: این ویژگی باعث افزایش انعطاف‌پذیری و قابلیت استفاده مجدد از کدها می‌شود، زیرا می‌توان اجزاء موجود را به راحتی در پروژه‌های دیگر مورد استفاده قرار داد.

قابلیت فهم(Understandability)

طراحی سیستم باید به گونه‌ای باشد که اجزاء مختلف آن به راحتی قابل فهم و درک باشند.

مزیت :این ویژگی به توسعه‌دهندگان کمک می‌کند تا به راحتی کدها را بخوانند، درک کنند و تغییرات لازم را اعمال کنند، که به بهبود کیفیت و نگهداری سیستم منجر می‌شود.

تداوم:(Continuity)

سیستم باید به گونه‌ای طراحی شود که تغییرات در یک ماژول به حداقل تأثیر ممکن بر سایر ماژول‌ها داشته باشد.

مزیت :این ویژگی باعث افزایش ثبات و قابلیت نگهداری سیستم می‌شود، زیرا تغییرات محلی و محدود به یک ماژول تأثیر کمتری بر کل سیستم خواهد داشت.

حفاظت(Protection)

سیستم باید به گونه‌ای طراحی شود که اجزاء مختلف از تأثیرات منفی تغییرات در سایر اجزاء محافظت شوند.

مزیت :حفاظت از ماژول‌ها در برابر تأثیرات منفی باعث افزایش قابلیت اعتماد و پایداری سیستم می‌شود.

پنج قانون ماژولاریتی Meyer

برتراند مایر برای رسیدن به پنج معیار فوق، **پنج اصل اساسی** برای طراحی ماژولار سیستم‌های نرم‌افزاری معرفی کرده است که به بهبود کیفیت، نگهداری و قابلیت تغییر سیستم کمک می‌کنند. این اصول شامل نگاشت مستقیم، رابط‌های کم، رابط‌های کوچک، رابط‌های صریح و پنهان‌سازی اطلاعات هستند. در ادامه هر یک از این اصول توضیح داده شده است:

۱. نگاشت مستقیم (Direct Mapping)

طراحی سیستم باید به گونه‌ای باشد که ماژول‌ها مستقیماً وظایف و نیازمندی‌های مشخصی را انجام دهند.

این اصل باعث می‌شود که توسعه‌دهندگان بتوانند به راحتی تغییرات و نیازمندی‌های جدید را در سیستم اعمال کنند و همچنین فهم و نگهداری سیستم را تسهیل می‌کند.

۲. رابط‌های کم (Few Interfaces)

تعداد رابط‌های بین ماژول‌ها باید تا حد ممکن کم باشد.

کاهش تعداد رابط‌ها باعث کاهش پیچیدگی و افزایش قابلیت نگهداری و انعطاف‌پذیری سیستم می‌شود. این اصل همچنین از وابستگی‌های زیاد بین ماژول‌ها جلوگیری می‌کند.

۳. رابط‌های کوچک (Small Interfaces)

رابط‌های بین ماژول‌ها باید کوچک و ساده باشند و فقط شامل اطلاعات ضروری باشند.

این اصل باعث می‌شود که ماژول‌ها به طور مستقل و با کمترین وابستگی به یکدیگر عمل کنند و تعاملات بین ماژول‌ها ساده‌تر و کم‌خطاتر شوند.

۴. رابط‌های صریح (Explicit Interfaces)

تمامی رابط‌های بین ماژول‌ها باید به صورت صریح و مشخص تعریف شوند.

این اصل کمک می‌کند تا تمامی ارتباطات بین ماژول‌ها به خوبی مستند و قابل درک باشند. این امر به توسعه‌دهندگان اجازه می‌دهد که به راحتی کدها را نگهداری و تغییرات لازم را اعمال کنند.

۵. پنهان‌سازی اطلاعات (Information Hiding)

جزئیات پیاده‌سازی داخلی ماژول‌ها باید مخفی باشد و فقط رابط‌های عمومی آن‌ها قابل دسترسی باشد.

پنهان‌سازی اطلاعات باعث کاهش وابستگی‌ها و افزایش انعطاف‌پذیری می‌شود. تغییرات داخلی یک ماژول تأثیر کمی بر سایر ماژول‌ها خواهد داشت که این امر به بهبود نگهداری و توسعه سیستم کمک می‌کند.