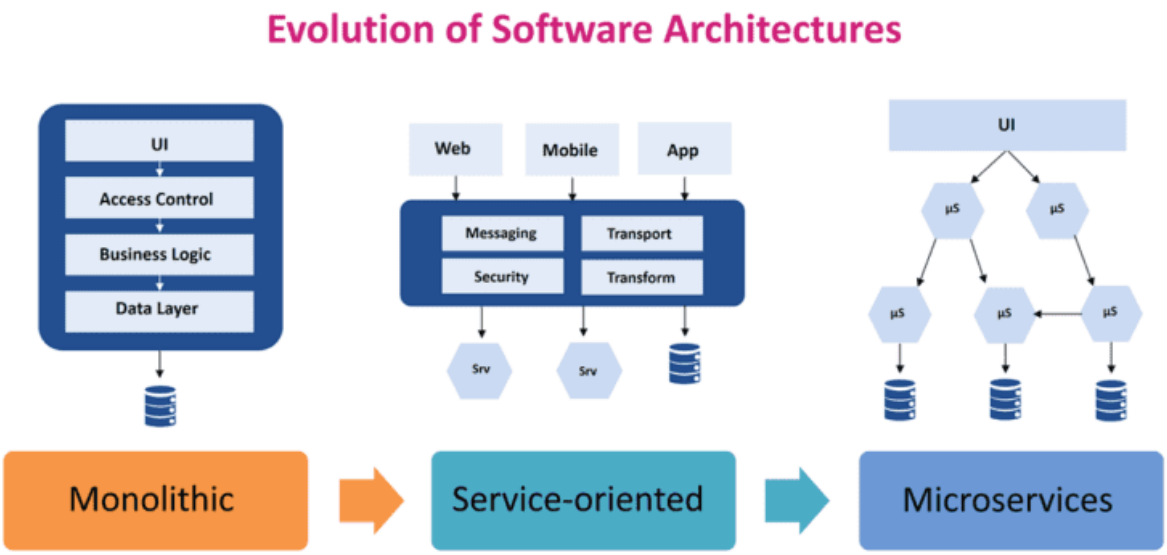
معماری نرم افزار چیست؟

منظور از معماری نرم افزار و یا Software Architecture 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.

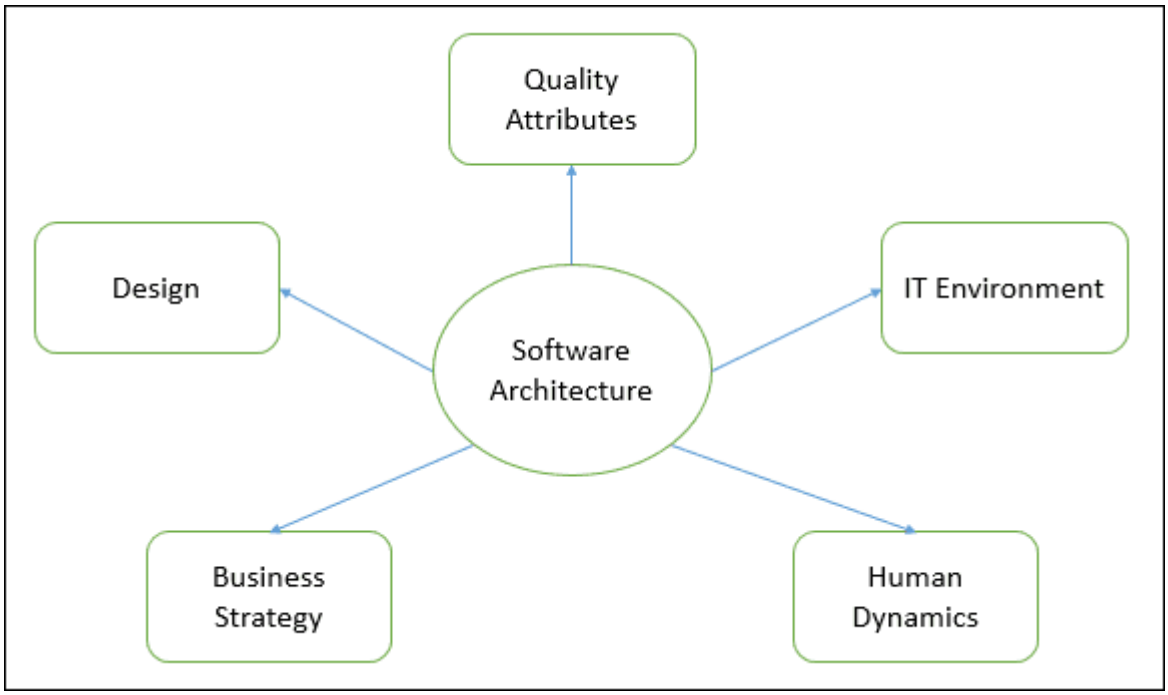


منظور از معماری نرم افزار؛ اغلب بررسی ساختار کلان تر و اساسی تر یک سیستم نرم‌افزاری و کار کردن با فرآیندهایی می باشد که با همکاری یکدیگر، وظایف یک نرم افزار را انجام می دهند. و منظور از طراحی نرم افزار و یا همان software design بررسی ساختارهای کوچکتر و ریزتر و همچنین بررسی طراحی داخلی یک فرآیند نرم افزاری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

مقدمه ای بر معماری نرم افزار و طراحی نرم افزار

معماری یک سیستم به اجزای تشکیل دهنده اصلی و کلان آن و البته ارتباط بین این ساختارها و تعاملات انجام شده بین آنها دلالت دارد. معماری نرم افزار و طراحی نرم افزار، شامل عوامل مشارکت کننده ای از قبیل؛ استراتژی کسب و کار و یا business strategy ویژگی های کیفی و یا quality attribute ها، مشارکت های انسانی و یا human dynamics ، طراحی و

یا design و محیط فناوری اطلاعات و یا IT environment می‌باشد. این موضوع به صورت واضح در تصویر زیر نشان داده شده است.



یک موضوع دیگر اینکه به راحتی می توان طراحی و معماری نرم افزار را به دو فاز منحصر به فرد، تفکیک کرد؛ معماری نرم افزار و طراحی نرم افزار. در معماری نرم افزار، تصمیمات غیر عملکردی و یا اصطلاحاً nonfunctional decision ها اتخاذ می‌گردند و از نیازمندی های عملکردی تفکیک می شوند. این در حالی است که در طراحی نرم افزار، نیازمندی های عملکردی در نظر گرفته می‌شوند. نیازمندی های عملکردی و یاfunctional requirements ، و در کنار آن نیازمندی های غیر عملکردی و یاNon-functional requirements ، دو موضوع بسیار مهم می باشند که در قسمت های بعدی در رابطه با آنها صحبت خواهیم کرد.

معماری نرم افزار

معماری به عنوان یک نقشه و یا یک الگو برای سیستم ایفای نقش می‌کند. با استفاده از معماری می‌توانیم یک دید کلی و یا اصطلاحاً abstraction را برای مدیریت پیچیدگی‌های سیستم ایجاد کنیم و یک ارتباط بسیار مفید و یک هماهنگی منحصربه‌فرد را بین اجزای تشکیل دهنده یک سیستم ایجاد نماییم.

معماری، یک راه حل ساختارمند را برای رسیدن به تمامی نیازمندی های فنی و عملکردی یک سیستم در دست قرار می دهد. علاوه بر این؛ با استفاده از معماری می توان ویژگی های کیفی معمول از قبیل؛ عملکرد و یا performanceو امنیت و یا security را تقویت کرد و یا بهبود داد.
علاوه بر این؛ معماری، شامل تصمیمات مهمی در رابطه با سازمان مربوط به توسعه نرم افزار است. هر کدام از این تصمیمات می‌توانند تاثیر چشمگیری بر روی کیفیت، قابلیت نگهداری، قابلیت عملکرد و موفقیت سراسری پروژه داشته باشند. برخی از این تصمیمات شامل موارد زیر هستند:

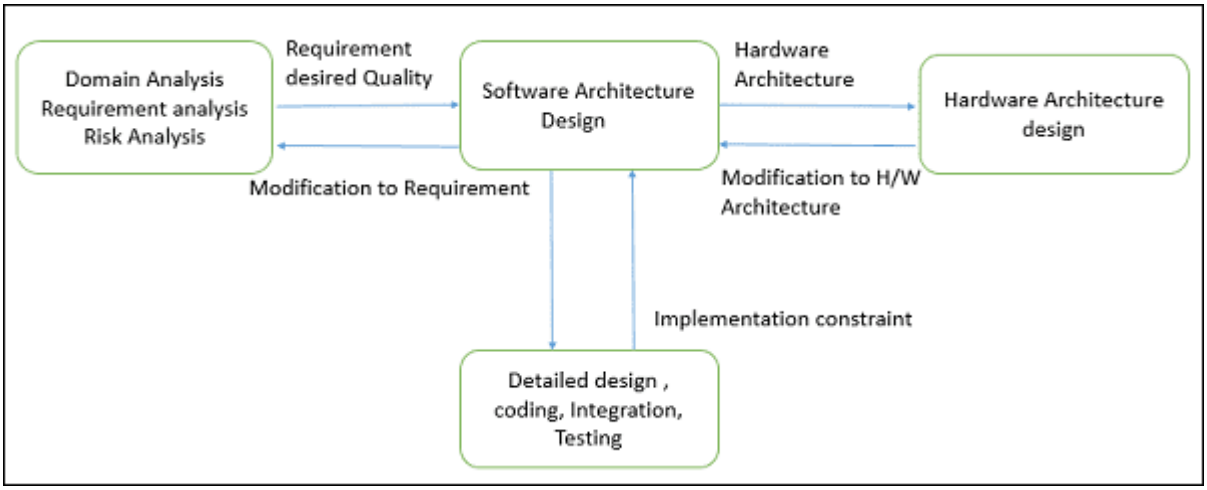
۱. انتخاب عناصر ساختاری و interface هایی که با استفاده از آنها سیستم به عملکرد خود جامه عمل می پوشاند.
۲. رفتار تعریف شده بین همکاری‌های شکل گرفته، مابین این عناصر.
۳. ترکیب این ساختارها و رفتارهای آنها به درون یک زیر سیستم بزرگتر.
٤. تصمیمات معماری منطبق بر اهداف کسب و کار.
٥. سبک های معماری و تصمیمات کاربردی در برابر آنها.

طراحی نرم افزار

طراحی نرم افزار باعث می شود که یک برنامه طراحی که در آن عناصر مربوط به یک سیستم، نحوه عملکرد آنها با یکدیگر و تعاملات بین آنها برای رسیدن به اهداف یک سیستم، توصیف شود. اهداف مربوط به داشتن یک برنامه طراحی و یا اصطلاحاً design plan، شامل موارد زیر می‌باشند:

۱. مذاکره کردن در رابطه با نیازمندی‌های سیستم و تنظیم کردن انتظارات مشتری، بازاریابی و مدیریت پرسنل.
۲. به عنوان یک نقشه برای فرآیند توسعه نرم افزار عمل می‌کند.
۳. راهنمایی برای وظایف پیاده سازی از قبیل؛ طراحی دقیق، کدنویسی، یکپارچه سازی و تست می باشد.

طراحی نرم‌افزار قبل از طراحی دقیق، کدنویسی، یکپارچه سازی و تست نرم‌افزار اتفاق می‌افتد و پس از تجزیه و تحلیل دامنه، تجزیه و تحلیل نیازمندی ها و تجزیه و تحلیل ریسک رخ می‌دهد. این تقدم و تأخر دقیقاً در تصویر زیر نشان داده شده اند.



اهداف معماری

هدف اصلی معماری، مشخص کردن نیازمندی هایی است که بر روی ساختار یک برنامه تاثیر می گذارند. یک معماری اصولی و دقیق، باعث کاهش ریسک های تجاری مربوط به ساختن یک نرم افزار فنی می‌شود و به عنوان یک پل ارتباطی، بین نیازمندی های فنی و نیازمندی های تجاری ایفای نقش می‌کند. برخی از دیگر اهداف مربوط به معماری، شامل موارد زیر هستند:

۱. نمایانگر ساختار سیستم می‌باشند اما جزئیات پیاده سازی را مخفی می کند.
۲. تمامی موارد کاری و یا use-case ها و سناریوها را جامه عمل می پوشانند.
۳. تلاش می‌کنند که به نیازمندی های مطرح شده توسط ذینفعان پاسخ دهند.
٤. هم نیازمندی های کیفی و هم نیازمندی های عملکردی را در نظر می گیرند.
٥. باعث کاهش هدف مالکیت و افزایش جایگاه بازار یک سازمان می شود.
٦. باعث افزایش سطح کیفی و عملکردی یک سیستم می گردد.
۷. باعث افزایش اعتماد خارجی به سازمان و یا سیستم می‌گردد.

محدودیت ها

معماری نرم افزار در حال حاضر یک حوزه نوظهور و در حال توسعه و تکامل در مهندسی نرم افزار به حساب می آید و شامل محدودیت های زیر می باشد:

۱. نبود ابزارها و روش های استاندارد برای نشان دادن معماری.
۲. نبود روش های دقیق تجزیه و تحلیل برای پیش بینی کردن اینکه آیا یک معماری به پیاده‌سازی منجر خواهد شد و تمامی نیازمندی ها را پاسخ خواهد داد و یا خیر.
۳. نبود آگاهی از اهمیت طراحی معماری اساسی و توسعه نرم‌افزار.
٤. نبود درک مناسبی از نقش معمار نرم افزار و ارتباط نامناسب و ضعیف بین ذینفعان.
٥. نبوده درک مناسب از فرایند طراحی، تجربه طراحی و ارزیابی طراحی.

نقش یک معمار نرم افزار

در حوزه تولید محصولات نرم افزاری، نقش یک معمار نرم افزار، بسیار مهم و اساسی می‌باشد. یک معمار نرم افزار، یک solution و یا راه حل را ارائه می‌دهد که تیم فنی می‌تواند آن را طراحی و تولید کند و در سرتاسر برنامه و محصول نرم افزاری استفاده نماید. یک معمار نرم افزار باید در مواردی که در قسمت زیر آنها را بررسی خواهیم کرد، تخصص کافی داشته باشد:

مهارت کامل در طراحی

متخصصین طراحی سیستم‌های نرم‌افزاری، از متدهای متنوع و متعدد و راهکارهای متفاوتی از قبیل طراحی شی گرا و طراحی مبتنی بر رویدادها استفاده می کنند. معماران نرم افزار در این حوزه باید بتوانند

تیم های توسعه نرم‌افزار را رهبری کنند و تلاش کافی برای هماهنگی آن دسته از افراد، در پیشبرد طراحی سیستم نرم افزاری را داشته باشند .این گونه از افراد همچنین می بایست بتوانند، پیشنهادهای طراحی را بازبینی کنند و مصالحه ها و یا موازنه (trade off) های بین هر کدام را تجزیه و تحلیل نمایند.

تخصص کامل در مورد دامنه

یکی دیگر از حوزه هایی که یک معمار نرم افزار باید در آن تخصص کامل داشته باشد، دامنه و یا domainی هست که سیستم نرم‌افزاری قرار است در آن مورد استفاده قرار بگیرد.

معمار نرم افزار باید بر روی فرآیند کشف نیازمندی‌ها و تضمین کامل شدن و سازگاری پروژه، دقت کافی را داشته باشد.

علاوه بر این؛ تعریف domain model برای سیستم در حال توسعه، یکی دیگر از حوزه‌های تخصصی یک معمار نرم‌افزار است.

تخصص تکنولوژی

معمار نرم افزار باید بتواند درک دقیق و کاملی از تکنولوژی هایی که می‌توانند در پیاده سازی یک سیستم مورد استفاده قرار بگیرند را داشته باشد همچنین؛ این گونه از افراد باید بتوانند انتخاب صحیحی از پلتفرم ها، فریم ورک ها، زبان های برنامه نویسی، پایگاه های داده یا database ها و مواردی از این دست را داشته باشند.

تخصص در متدولوژی

متخصصین حوزه معماری نرم افزار باید بتوانند بر روی متدولوژی‌های توسعه نرم‌افزار که می‌توانند در فاز SDLCو همان software development lifecycle مورد استفاده قرار بگیرند، تخصص کاملی داشته باشند. این گونه از افراد باید بتوانند از روش های مناسب برای توسعه نرم افزار استفاده کنند و تمامی اعضای تیم نرم افزاری را به سمت این موارد سوق بدهند.

نقش پنهان معمار نرم‌افزار

یکی از نقشهای بسیار مهم و اغلب پنهان معمار نرم افزار، تسهیل کار فنی بین اعضای تیم نرم افزار و ایجاد اعتماد و ارتباط بین آنها است. متخصصین اطلاعات که دانش و تجارب خود را با اعضای تیم به اشتراک می گذارند، در واقع همین معماران نرم افزار هستند.

یکی دیگر از وظایف معماران نرم افزار، محافظت از اعضای تیم در برابر عوامل خارجی است که می‌تواند باعث از بین رفتن تمرکز آنها و از بین بردن ارزش کاری آنها بشود.

موارد مورد تحویل یک معمار

در این قسمت در رابطه با آنچه که یک معمار نرم افزار می بایست با حضور خود در پروژه رقم بزند و به ما تحویل دهد را بررسی خواهیم کرد.

اهداف عملکردی قابل اندازه گیری، کامل، شفاف و سازگار، اولین مورد از مواردی است که معمار نرم افزار باید آنها را بتواند تحویل بدهد.

مورد بعدی، یک توصیف عملکردی از سیستم با حداقل دو لایه از تجزیه می باشد.

علاوه بر این، معمار نرم افزار باید بتواند درکی از سیستم را برای ما ایجاد کند.
طراحی یک سیستم نرم‌افزاری با حداقل دو لایه از تجزیه و یا همان decomposition یکی دیگر از وظایف معمار نرم افزار است.
درک مناسبی از زمان بندی، ویژگی های عملکردی و همچنین پیاده سازی برنامه های عملیاتی، از دیگر وظایف یک معمار نرم افزار می باشد.
یک سند و یا فرآیند که ضمانت کند، تجزیه عملکردی در روند توسعه پروژه تحت نظر قرار می‌گیرد، از دیگر وظایف یک معمار نرم‌افزار است.

ویژگی های کیفی و اهمیت آنها در طراحی و معماری نرم افزار

کیفیت یک مقیاس برای سنجش سلامت و حالت یک محصول نرم افزاری و البته، فقدان و نبود مشکلات می باشد.

ویژگی های کیفی

ویژگی های یک سیستم را نشان می‌دهند که آن سیستم را از عملکرد خود تفکیک می کند. پیاده سازی کردن و لحاظ نمودن ویژگی های کیفی می تواند، تفاوت یک سیستم مناسب از یک سیستم غیر مناسب را نشان بدهد. ویژگی ها، عوامل کلی و سراسری هستند که بر روی رفتار یک نرم‌افزار در زمان اجرا شدن، طراحی سیستم و تجربه کاربری تاثیر می گذارند. ویژگی‌های کیفی در طراحی و معماری نرم افزار به دسته های مختلفی تقسیم بندی می شوند که در ادامه در رابطه با آنها صحبت خواهیم کرد.

ویژگی های کیفی ایستا

ویژگی های کیفی ایستا، بازتابی از ساختار یک سیستم و سازماندهی آن می باشند که به طور دقیق و مستقیم به معماری، طراحی و سورس کد مرتبط می شوند. این نوع از ویژگی های کیفی، از دید کاربر نهایی مخفی می باشند. اما بر روی هزینه توسعه و هزینه نگهداری تاثیر می گذارند. برخی از این ویژگی های کیفی شامل؛ ماژولار بودن، قابل تست بودن و قابل نگهداری بودن می باشند.

ویژگی های کیفی پویا

ویژگی های کیفی پویا و یاdynamic ، بازتابی از رفتار یک سیستم در زمان اجرا شدن می باشند. این نوع از ویژگی ها، به طور مستقیم به معماری یک سیستم، طراحی، سورس کد، پیکربندی، پارامتر های استقرار، محیط استقرار و پلتفرم مرتبط می شوند. این نوع از ویژگی ها توسط کاربر نهایی دیده می‌شوند و در زمان اجرا شدن برنامه وجود دارند. برخی از آنها شامل؛ بهره وری، محکم بودن، مقیاس پذیری و غیره می باشند.

سناریوهای کیفی

منظور از سناریو های کیفی، مشخص نمودن این است که چگونه می توان از یک fault یا عیب، اجتناب کرد و به یک failure یا خرابی نرسید. لطفاً دقت کنید که این دو واژه فنی یعنی fault و failure از لحاظ معنا هرچند در فارسی ممکن است شبیه به هم به نظر برسند. اما، دو مفهوم متفاوت را دارا هستند.

می‌توان سناریوهای کیفی را به شش قسمت مختلف تقسیم نمود که عبارتند از؛

۱. منبع و یا source : که در آن یک entity درونی و یا خارجی از قبیل؛ افراد، سخت افزارها، نرم افزارها یا زیرساخت های انسانی باعث ایجاد محرکی می شوند.
۲. محرک و یا stimulus : شرایطی که نیاز دارند به آنها رسیدگی شوند.
۳. محیط و یا environment : محرکی که در یک شرایط خاص به وجود می‌آید.
٤. مصنوع و یا artifact : یک سیستم کامل و یا بخشی از آن، از قبیل؛ پردازنده‌ها، کانال‌های ارتباطی، فضای ذخیره سازی، فرآیندها و غیره.
٥. پاسخ و یا response : عملیاتی که پس از به وجود آمدن یک محرک، از قبیل؛ کشف خطا، بازیابی از یک مشکل، غیرفعال کردن یک event source و غیره اتخاذ می شود.
٦. مقیاس پاسخ و یا response measure : می بایست پاسخ های اتفاق افتاده را اندازه‌گیری اندازه گیری کند تا اینکه بتوان نیازمندی‌ها را مورد تست قرار داد.

ویژگیهای کیفی معمول

در ادامه در رابطه با ویژگی های کیفی معمول که یک معمار نرم افزار باید به آنها دقت کند، صحبت خواهیم کرد.

دسته بندی	ویژگی کیفی	توصیف
Design Qualities یا ویژگی های طراحی	Conceptual Integrity	نمایانگر سازگاری و یا consistency و چسبندگی و یا coherence طراحی سراسری نرم افزار می باشد. این معیار شامل نحوه ارتباط و طراحی کامپوننت ها و ماژول ها با یکدیگر می باشد.
	Maintainability	توانمندی سیستم برای روبرو شدن با تغییرات با درجه مناسبی از آزادی و راحتی می باشد.

	Reusability	نمایانگر توانمندی کامپوننت ها و زیر سیستم ها برای مورد استفاده قرار گرفتن در دیگر برنامه ها می باشد.
Run-time Qualities یا ویژگی-های زمان اجرا	Interoperability	توانمندی یک سیستم یا سیستم های متفاوت به ارتباط برقرار کردن موفق با یکدیگر و ارسال و دریافت پیام ها با سیستم های خارجی که توسط تکنولوژی های مختلف نوشته شده اند.
	Manageability	نمایانگر این است که برای مدیران سیستم، مدیریت کردن نرم افزار چقدر ساده می باشد.
	Reliability	توانمندی سیستم برای این که بتواند برای مدتی طولانی به عملکرد خود ادامه دهد.
	Scalability	توانمندی اینکه یک سیستم بتواند افزایش بار را بدون تحت تاثیر قرار گرفتن کارایی سیستم مدیریت کند و در زمان های مورد نیاز بر اساس نیازمندی بزرگ و کوچک بشود.
	Security	توانمندی یک سیستم برای جلوگیری کردن از اتفاقات ناگوار و تصادفی که خارج از محدوده طراحی اتفاق می‌افتند.
	Performance	نمایانگر سطح پاسخ دهی بودن و یا responsiveباقی ماندن یک سیستم در مدت زمان مشخص است.
	Availability	نمایانگر مدت زمان اینکه یک سیستم چگونه می تواند به عملکرد خود ادامه بدهد. این معیار اغلب به صورت یک درصد بیان می شود.
System Qualitiesیا ویژگی های سیستم	Supportability	اینکه چگونه سیستم بتواند اطلاعاتی را برای حل و فصل کردن و شناختن معضلات و مشکلات گزارش بدهد.
	Testability	اینکه به چه صورت بتوانیم یک سیستم را بر اساس معیارهای تست، مورد ارزیابی قرار بدهیم.

User Qualitiesیا ویژگی های کاربر	Usability	اینکه چگونه بتوان نرم افزار تولید شده را بر اساس نیازمندی های کاربر و مصرف کننده ایجاد کرد تا بتواند مورد استفاده قرار بگیرد.
Architecture Qualityیا ویژگی های معماری	Correctness	اینکه یک نرم افزار چگونه نیازمندی های سیستم را جواب می دهد.
Non-runtime Quality یا ویژگی های غیر زمان اجرا	Portability	اینکه یک سیستم چگونه می‌تواند در محیط-های نرم افزاری و کامپیوتر مختلف اجرا شود.
	Integrality	اینکه چگونه می توان قسمت های مختلف سیستم که توسط افراد مختلف توسعه داده شدند را در کنار یکدیگر مورد استفاده قرار داد.
	Modifiability	اینکه چگونه می توان سیستم نرم‌افزاری را بر اساس تغییراتی که رخ می دهند، تغییر داد.
Business quality attributes یا ویژگی های کسب و کار	Cost and schedule	هزینه سیستم با توجه به زمان، بازاریابی، طول عمر پروژه و سطح استفاده کردن از آن.
	Marketability	کاربرد سیستم با توجه به رقابت های بازار.

اصول کلیدی در معماری نرم افزار

در این قسمت قصد داریم در رابطه با مهمترین اصول کلیدی در معماری و طراحی نرم‌افزار صحبت کنیم. معماری نرم افزار در واقع سازماندهی یک سیستم است که در آن یک سیستم از اجزای تشکیل دهنده مختلفی که هر کدام قرار است وظایف خاصی را ایفا کنند، تشکیل شده است.

سبک معماری و یاarchitectural style

منظور از سبک معماری و یا architectural style که تحت عنوان الگوی معماری و یا architectural pattern نیز شناخته می شود، مجموعه ای از اصول است که یک محصول نرم افزاری را تشکیل می‌دهند. سبک معماری مشخص کننده یک چهارچوب مجرد و انتزاعی است که در آن مجموعه ای از سیستم ها با الگوهای خاصی از لحاظ معماری در کنار هم سازماندهی می‌شوند. وظایف سبک معماری به قرار زیر می باشند:

۱. یک مجموعه از اجزای تشکیل دهنده و ارتباطات بین آنها را تعریف می کند.
۲. باعث می شود که سیستم به قطعات کوچکتر تقسیم گردد و قابلیت استفاده مجدد از هرکدام از قسمت های مختلف را فراهم می کند.

۳. مشخص کننده یک روش خاص برای پیکربندی مجموعه ای از اجزای تشکیل دهنده) ماژول هایی که interface های خوش تعریف، قابل استفاده مجدد و قابل جایگزینی (بوده و البته، مشخص کننده اتصالات و ارتباطات بین این اجزای تشکیل دهنده نیز می باشد.

سیستم نرم افزاری که بر اساس سبک های معماری مختلف نوشته می شود، می‌تواند نسبت به دیگر محصولات نرم افزاری، موفقیت بیشتری را کسب کند. هر سبک معماری شامل موارد زیر می باشد:

۱. مجموعه ای از انواع component ها که یک عملیات خاصی را در سیستم اجرا می کنند.
۲. مجموعه ای از اتصالات و یا ارتباطات، از قبیل؛ subroutine call ها و remote procedure callها و data streamها وsocket ها که باعث برقراری ارتباط، هماهنگی و همکاری بین component های مختلف می‌شوند.
۳. قیود معنایی و یا semantic constraint ها که مشخص می‌کنند چگونه component ها برای شکل دهی سیستم نهایی با یکدیگر یکپارچه می‌شوند.
٤. یک چیدمان توپولوژیکی از component ها که مشخص کننده روابط فی مابین آنها در زمان اجرای برنامه است.

طرح های معماری معمول

در ادامه در رابطه با سبک های معماری معمول و ویژگی های خاص هر کدام با یکدیگر، صحبت خواهیم کرد.

دسته بندی	سبک های معماری	شرح
Communication	Message bus	مشخص کننده‌ی استفاده از سیستم‌های نرم‌افزاری برای ارسال و دریافت پیام ها و شکل‌دهی ارتباطات بین componentها می باشد.
	Service–Oriented Architecture (SOA)	مشخص کننده نرم افزارهای می باشند که برخی عملیات را در قالب service و یا خدماتی با استفاده از قراردادها و پیام هایی در اختیار دیگر برنامه‌ها قرار می‌دهند.
Deployment	Client/server	سیستم نرم‌افزاری را به دو برنامه به نام client و server می شکند که در آن client درخواست‌هایی را به server ارسال می‌کند.

	3-tier or N-tier	عملیات مربوط به برنامه را به قسمت ها و لایه های مختلف می شکند که هر کدام از آنها می‌توانند بر روی سیستم های فیزیکی مختلفی قرار بگیرند.
Domain	Domain Driven Design	بر اساس مدل سازی business domain و یا دامنه تجاری و تعریف کردن business object ها، بر اساس entityها عمل می کند.
Structure	Component Based	برنامه را به قسمت های کوچک تر و قابل استفاده مجدد می شکند که هر کدام برای ارتباط بین یکدیگر، دارای interface های می‌باشند.
	Layered	قسمت های مختلف برنامه را به لایه‌های مختلف می شکند.
	Object oriented	بر اساس تقسیم کردن مسئولیت های یک برنامه و یا سیستم به object های مختلف است که هر کدام شامل داده و رفتارهای مربوط به آن object می باشند.

انواع معماری ها

به طور کلی در طراحی و معماری نرم‌افزار، چهار نوع معماری از نقطه نظر یک سازمان (کسب و کار) و یا enterprise وجود دارد که اصطلاحاً به آنها enterprise architecture می گوییم. در رابطه با هر کدام از آنها در قسمت زیر صحبت خواهیم کرد.

۱. معماری تجاری و یا business architecture : مشخص کننده راهبردهای تجاری، مدیریتی، سازماندهی و فرایندهای تجاری اصلی در یک کسب و کار است و بر روی تجزیه و تحلیل و طراحی فرایندهای تجاری تمرکز می‌کند.

۲. معماری نرم افزار و یا application software architecture: به عنوان یک نقشه و یا راهنما برای سیستم‌های نرم‌افزاری عمل می‌کند و ارتباطات و تعاملات بین فرایندهای تجاری یک سازمان را مدل می‌کند.

۳. معماری اطلاعاتی و یا information architecture : مشخص کننده دارایی های اطلاعاتی فیزیکی و منطقی و همچنین منابع مدیریت داده می‌باشند.

٤. معماری فناوری اطلاعات و یا IT architecture : مشخص کننده اجزای تشکیل دهنده نرم افزاری و سخت افزاری می باشد که سیستم نهایی مربوط به سازمان مورد نظر را تشکیل می دهند.

فرآیندهای طراحی معماری و یاarchitecture design process

منظور از فرآیند طراحی معماری، تمرکز بر روی تجزیه کردن یک سیستم به اجزای تشکیل دهنده و یا component های متفاوت و همچنین تعاملات بین آنها برای برقراری نیازمندی‌های functional و non-functional می باشد. ورودی اصلی طراحی معماری نرم افزار، موارد زیر می باشند:

۱. نیازمندی های تولید شده توسط مراحل تجزیه و تحلیل

۲. معماری سخت افزاری (معمار نرم‌افزار به نوبه خود، نیازمندی‌های معمار سیستم که مسئولیت پیکربندی معماری سخت افزاری را دارد، نیز فراهم می‌کند.)

نتیجه و یا خروجی فرایند طراحی معماری، یک توصیف معماری و یا architectural description می باشد. یک فرایند طراحی معماری ساده از مراحل طراحی زیر تشکیل گردیده است:

درک مسئله

اولین و مهمترین مرحله درک مسئله است. چراکه این مرحله می‌تواند شدیداً بر روی کیفیت طراحی مراحل بعدی تاثیر بگذارد. بدون داشتن درک مناسبی از مسئله پیش رو، نمی‌توان یک solution و یا راه حل موثر را برای مسئله ایجاد کرد. بسیاری از پروژه های نرم افزاری و محصولات، به این دلیل شکست می خورند که نمی توانند یک مسئله تجاری معتبر و یا اصطلاحاً business problem را حل و فصل کنند و یا بازگشت سرمایه قابل مشاهده ای را ایجاد نمی کنند.

مشخص کردن عناصر طراحی و ارتباطات بین آنها

۱. در این مرحله می بایست که یک شالوده و اساس برای طراحی کردن و تعریف کردن مرزها و همچنین context مربوط به سیستم، ایجاد کنید. منظور ازcontext ، فضایی است که سیستم نرم‌افزاری نهایی قرار است در آن اجرا بشود.

۲. تجزیه کردن یک سیستم به component های اصلی آن بر اساس نیازمندی های functional و یا عملکردی در این قسمت اتفاق می‌افتد. تجزیه کردن می‌تواند با استفاده از یک ماتریس ساختار طراحی و یا اصطلاحاً design structure matrix اتفاق بیفتد. این ماتریس نشان دهنده وابستگی های بین عناصر طراحی، بدون مشخص کردن دانه بندی مربوط به این عناصر می باشند.

۳. در این مرحله اولین اعتبارسنجی معماری انجام می‌شود و این موضوع با توصیف تعدادی از نمونه های سیستم اتفاق می افتد. علاوه بر این؛ نام دیگری که بر این مرحله می‌گذارند functionality based architectural design و یا طراحی معماری مبتنی بر عملکرد می باشد.

ارزیابی کردن طراحی معماری

۱. هر ویژگی کیفی که در قسمت های قبلی مشخص شده است و حتی در رابطه با آنها نیز صحبت کردیم، به صورت تخمینی در این مرحله ارزیابی می شود. این کار برای به دست آوردن یک معیار کیفی و کمی اتفاق می افتد.

۲. در این مرحله، معماری طراحی شده را برای تطابق آن با ویژگی ها و نیازمندی های کیفی معماری می سنجیم.

۳. اگر تمامی ویژگی های کیفی بر اساس استانداردهای نیازمندی‌ها، ارزیابی گردند، سپس فرآیند طراحی معماری به پایان می‌رسد.

٤. اگر این اتفاق نیافتد، وارد فاز سوم از طراحی معماری نرم افزار می شویم که به آن architecture transformation و یا تغییر شکل معماری می گویند. اگر معیارهای کیفی به نیازمندی‌های مشخص شده پاسخ مناسبی ندهند، می بایست که یک طراحی جدید ایجاد شود.

تغییر شکل طراحی معماری

در این مرحله؛ می بایست که طراحی معماری مشخص شده، کمی تغییر کند. این مرحله پس از ارزیابی یک طراحی معماری اتفاق می افتد. به عبارت دیگر؛ طراحی معماری مورد نظر می بایست تا زمانی که به نیازمندی‌های کیفی پاسخ بدهد تغییر کند. در این مرحله راه حل های طراحی مختلفی برای بهبود ویژگی های کیفی نرم افزار و در عین حال حفظ کردن domain functionality و یا عملکرد دامنه نرم افزار، انتخاب می گردد.

نحوه تغییر دادن یک طراحی با استفاده از اضافه کردن عملگرهای طراحی، سبک های مختلف و یا حتی الگوه های متنوع می باشد. برای تغییر شکل، یک طراحی از قبل موجود را انتخاب کرده و عملگرهای طراحی متفاوتی از قبیل؛ تجزیه و یا decomposition، تکثیر و یاreplication ، فشرده سازی و یاcompression ، مجرد سازی و یا abstraction و به اشتراک گذاری منابع و resource sharing را بر روی آن انجام می دهیم.

پس از انجام این کار، طراحی مورد نظر باری دیگر ارزیابی می شود و اگر نیاز باشد مراحلی که ذکر شد دوباره انجام می گردند. این کار تا زمانی اتفاق می افتد تا شرایط طراحی مورد نظر، از نقطه نظر ارزیابی مساعد و مناسب گردند. تغییر شکل ها، اغلب ویژگی های کیفی متعددی را بهبود می بخشند و از طرفی؛ ممکن است بر روی برخی از ویژگی‌های دیگر تاثیر منفی داشته باشد.

مهمترین اصول معماری صحیح نرم افزار

در این قسمت در رابطه مهمترین اصول طراحی معماری موفق محصولات نرم افزاری صحبت خواهیم کرد. مواردی که در قسمت زیر بررسی خواهند شد، جزء مهمترین اصولی هستند که در پیاده سازی یک طراحی موفق باید به آنها دقت کرد.

ساختن برای تغییر کردن به جای ساختن برای باقی ماندن

همه ما می‌دانیم که سیستم‌های نرم‌افزاری در طول عمر خود دچار تغییراتی در نیازمندی‌ها و البته بروز چالش‌های جدید می‌شوند. به همین دلیل؛ می بایست سیستم های نرم افزاری از انعطاف پذیری بالایی برخوردار باشند تا بتوانند به این تغییر در نیازمندی ها به خوبی پاسخ بدهند.

کاهش ریسک ها و مدلسازی برای تجزیه و تحلیل

با استفاده از ابزارهای طراحی و سیستم های مدلسازی از قبیل UML می‌توانیم تصمیمات طراحی و نیازمندی ها را هرچه بهتر درک کنید. تاثیرِ تصمیم گیری ها می توانند با استفاده از این ابزار ها، اندازه گیری و تجزیه و تحلیل بشود. نکته مهم در استفاده کردن از این ابزار ها این است که می‌بایست نگاه کاملاً کاربردی داشته باشید. به عبارت دیگر؛ همواره به این نکته فکر کنید که هدف این ابزارها کاهش دادن پیچیدگی است و نه اینکه خود آنها به عنوان یک پیچیدگی به سیستم اضافه گردند.

استفاده کردن از مدل ها و ابزارهای بصری به عنوان روشی برای ارتباط و همکاری

یکی از موضوعات مهم در ایجاد یک معماری خوب، ارتباط و انتقال موثر طراحی، تصمیمات و تغییراتی است که مرتباً بر روی نیازمندی ها رخ می دهند. با استفاده از مدل ها و view ها و دیگر ابزارهای بصری، می توان معماری را و طراحی آن را به ذینفعان پروژه منتقل کرد. این موضوع باعث می شود که ارتباط موثر و پایداری بین تیم توسعه نرم افزار و ذینفعان پروژه اتفاق بیافتد.

می بایست که تصمیمات کلیدی مهندسی نرم افزار را شناخته و آنها را تشخیص بدهیم. علاوه بر این؛ تصمیماتی که احتمال اشتباه و خطا در آنها هستند را مشخص کنیم. در ابتدای کار همواره تمرکز باید بر این باشد که تصمیمات طراحی و معماری، صحیح اتخاذ و پیاده سازی گردند. این موضوع باعث می شود که طراحی پیاده سازی شده از انعطاف پذیری بالاتری برخوردار باشد و تغییرات باعث شکستن برنامه نشود.

استفاده از روش‌های تکراری (iterative) و افزایشی (incremental)

با استفاده از این روش ها در ابتدا یک شالوده اصلی و کلی را برای معماری در نظر گرفته و سپس، به مرور با استفاده از تکنیک‌های مختلفی، این معماری بهبود و تکامل خواهد یافت. در این روند، تست کردن طراحی و معماری نرم افزار بسیار ضروری و مهم می باشد. اگر به صورت تکراری و افزایشی جزئیات مورد نظر، در فازهای متعدد، به طراحی مورد نظر اضافه گردند، تصویر نهایی جذاب تر و البته از انعطاف پذیری بالاتری برخوردار خواهد بود.
علاوه بر این اصول که در رابطه با مهم ترین قضایای مربوط به معماری نرم افزار صحبت می کنند، مهم ترین اصول طراحی نرم افزارهای موفق را نیز باید در نظر گرفت. این موضوع را در قسمت بعدی بررسی خواهیم کرد.

مهمترین اصول طراحی نرم افزار های موفق

در این قسمت می خواهیم مهمترین اصول مربوط به طراحی نرم افزارهای موفق را بررسی کنیم. با در نظر گرفتن و توجه به این اصول، می توانید به سادگی، هزینه را کاهش داده و قدرت نگهداری و یا همان maintenance نرم افزار را افزایش بدهید. قابلیت های دیگر از قبیل؛ قابلیت گسترش پذیری و یا همان extendibility و قابلیت استفاده کردن و یاusability ، با استفاده از این اصول، افزایش پیدا خواهند کرد.

اصلseparation of concerns

بر اساس این اصل بسیار مهم، اجزای تشکیل دهنده و یا component های یک سیستم نرم افزاری، می بایست به قسمت‌های کوچک تر تقسیم شده و هیچ کدام از آنها با یکدیگر همپوشانی نداشته باشند. این موضوع باعث افزایش cohesion و کاهش coupling می‌شود. برای به دست آوردنseparation of concerns ، روش های مختلفی وجود دارند. یکی از این روش ها، استفاده کردن مناسب از interface ها می باشد. با استفاده از اصلseparation of concerns ، وابستگی درونی اجزای تشکیل دهنده و یا component های یک سیستم به حداقل رسیده و این موضوع باعث افزایش قابلیت نگهداری و یا همان maintenance نرم افزار خواهد شد.

اصل تک وظیفه ای(single responsibility principle)

بر اساس اصلsingle responsibility principle ، هر ماژول و یا کلاس و یا حتی متد از یک سیستم نرم‌افزاری، می‌بایست فقط و فقط یک وظیفه خاص را داشته باشد و آن وظیفه را به خوبی انجام بدهد. این موضوع باعث می‌شود که درک سیستم نرم‌افزاری برای برنامه نویسان به مراتب ساده تر گردد. علاوه بر این؛ با استفاده از این اصل، به راحتی، integration و یا یکپارچه سازی یک نرم افزار با دیگر اجزای تشکیل دهنده آن، به خوبی انجام می‌گردد. در رابطه با این اصل که اولین اصل از اصول پنجگانه SOLID می باشد.

اصلprinciple of least knowledge

بر اساس اصلprinciple of least knowledge ، هر component و یا object خاصی می بایست حداقل دانش را در رابطه با جزئیات درونی دیگر component ها داشته باشد. این اصل نیز هدف خود را بر کاهش وابستگی درونی اجزاهای تشکیل دهنده نرم افزار گذارده است و به این ترتیب قابلیت نگهداری برنامه را افزایش می‌دهد.

کاهش دادن طراحی‌ها در ابتدای کار

یک موضوع بسیار مهم در طراحی نرم افزارهای موفق این است که در ابتدای کار نباید طراحی های بزرگ را انجام داد. چراکه نیازمندی‌ها در این زمان کاملاً مشخص نشده و امکان تغییر یافتن آنها زیاد است. به هر حال؛ اگر احتمال تغییر یافتن نیازمندی ها زیاد می باشد، بهتر است که در ابتدای کار، سرمایه‌گذاری زیادی بر روی طراحی کلی سیستم قرار نگیرد.

عدم تکرار کردن عملکرد و یاfunctionality

یک موضوع بسیار مهم دیگر، در طراحی نرم افزارهای موفق، عدم تکرار یک functionality و یا ویژگی در برنامه است. به عبارت دیگر؛ هیچگاه نباید یک تکه کد و یا یک class دارای کدهای تکراری باشد. احتمالاً می‌دانید که تکرار کدها و یا اصطلاحاً code duplication، جزء مهمترین code smell ها و یا بوی بدِ کُد به حساب می‌آید. تکراری بودن کد ها باعث می‌شود که ایجاد تغییرات در یک نرم افزار به سختی انجام گردد. علاوه بر این؛ شفافیت برنامه کاهش یافته و احتمال به وجود آمدن ناسازگاری و یا اصطلاحاً inconsistency افزایش می‌یابد

از composition به جای inheritance استفاده کنید و به این ترتیب کدها را مجدداً مورد استفاده قرار بدهید

یک از موضوعات بسیار مهم در طراحی یک نرم افزار موفق، استفاده از اصول شی گرایی می‌باشد. یکی از اصول شی گرایی، وراثت و یا inheritance است. متاسفانه وراثت، باعث ایجاد dependency و یا وابستگی بین کلاس های فرزند و پدر می‌شود و به همین ترتیب، استفاده کردن از کلاس‌های فرزند را دچار اختلال می کند. این در حالی است که تکنیک compositionو یا ترکیب می‌تواند سطح بسیار مناسبی از آزادی در استفاده از کلاس های موجود در سلسله مراتب وراثت را به ما بدهد. برای پیاده سازی composition و یا ترکیب، از interface ها استفاده می کنیم

مشخص کردن component ها و گروه بندی کردن آنها در لایه های منطقی

موضوع دیگر این است که می بایست اجزای تشکیل دهنده و یا component های مربوط به یک قسمت از برنامه را بر اساس نیازمندی ها تشخیص بدهید. پس از انجام این کار؛ این component های مرتبط با یکدیگر باید در یک لایه منطقی گروه بندی شوند. این موضوع باعث می‌شود تا درک ساختار کلی یک نرم افزار در سطوح بالاتر راحت تر اتفاق بیافتد. البته دقت کنید که نمی‌بایست component هایی که نوع آنها متفاوت است و یا در رابطه با functionality های متفاوت پیاده سازی شده اند را در یک لایه یکسان قرار داد.

تعریف کردن پروتکل ارتباطی بین لایه ها

اگر component ها به لایه های مختلف تقسیم گردند، می بایست بتوانند با یکدیگر ارتباط برقرار کنند. به منظور پیاده سازی قابلیت ارتباط با یکدیگر، componentها و لایه ها می بایست پروتکل های خاصی را داشته باشند. این موضوع نیز، یکی دیگر از اصول مهم طراحی موفق نرم افزار ها می باشد.

تعریف کردن قالب داده برای یک لایه

همانطور که تا به اینجای کار گفتیم، لایه های مختلف و component های بین آنها می بایست بتوانند با یکدیگر ارتباط برقرار کنند. به منظور ارسال داده ها بین لایه های مختلف، یک قالب داده ای و یا date format نیاز خواهد بود. لطفاً دقت کنید که نمی بایست date format ها را طوری با هم ترکیب کنید که برنامه دچار اختلال گردد. پیاده سازی date formatها باید به گونه‌ای باشد که استفاده کردن از آنها به سادگی انجام پذیر باشد و علاوه بر این؛ قابلیت گسترش و نگهداری بالایی داشته باشند. می‌بایست برای یک لایه، یک data format یکسان لحاظ بگردد تا component های مختلف بتوانند

داده های مورد نظر خود را رمزگشایی و یا رمزنگاری کنند و به این ترتیب، سربار پردازش کار با data format ها را کاهش دهند.

componentهای system service می بایست abstract باشند

یک موضوع مهم دیگر در طراحی نرم افزارهای موفق این است که کدهای مربوط به مسائلی از قبیل؛ امنیت و ارتباط و حتی لاگ کردن و پروفایل کردن و پیکر بندی داده ها، می بایست به صورت abstract در component های جداگانه تعریف بگردند. این گونه از کدها که اصطلاحاً تحت عنوان cross-cutting concern نیز شناخته می شوند، نمی بایست که با business logicبرنامه ترکیب بگردند. این کار باعث می شود که قابلیت گسترش طراحی و نگهداری آن، افزایش پیدا کند.

تلاش به طراحی exception ها و مکانیسمexception handling

تعریف کردن exception های مربوط به یک domain از قبل از طراحی، کمک می‌کند که component ها بتوانند خطاها و شرایط نامطلوب را به شکل مناسبی مدیریت کنند. سیستم مدیریت استثنا، می بایست در سرتاسر نرم افزار به یک شکل پیاده سازی گردد.

رسم و رسومات نامگذاری

یک موضوع بسیار مهم دیگر در طراحی نرم افزارهای موفق، استفاده از رسم و رسومات نامگذاری عناصر مختلف نرم افزار می باشد. رسم و رسومات نامگذاری که تحت عنوان naming convention از آنها یاد می شود، برای یک زبان و یا یک نرم افزار می بایست از پیش تعریف شوند. این روش باعث می شود که یک مدل سازگار و یک شکل در اختیار برنامه نویسان قرار بگیرد، تا بتوانند نرم‌افزار را هر چه راحت تر درک کنند. این موضوع علاوه بر این می‌تواند به برنامه نویسان کمک کنند تا کدهای دیگر برنامه نویسان را ارزیابی کنند و از این طریق قابلیت نگهداری و یا maintainability سیستم افزایش می یابد.

مدل های معماری نرم افزار

معماری نرم افزار شامل ساختار سطح بالای یک سیستم نرم‌افزاری انتزاعی و یا abstract می باشد که با استفاده از تکنیک های مختلفی از قبیل؛ تجزیه و یا decomposition و ترکیب و یا composition و البته استفاده کردن از سبک های معماری مختلف و ویژگی‌های کیفیتی و یاquality attribute ها تشکیل می گردد. یک معماری نرم افزار و طراحی آن می بایست با مهمترین نیازمندی های عملکردی و کارایی سیستم تطابق داشته باشد و علاوه بر این نیازمندی های غیر عملکردی و یا non-functional از قبیل؛ قابلیت reliability و یا اعتمادپذیری، قابلیت scalability و یا مقیاس پذیری، قابلیت portabilityو یا جابجایی پذیری و قابلیت availability و یا در دسترس بودن را برقرار کند.

یک معماری نرم افزار باید گروهی از component ها را برای خود تعریف کرده، ارتباطات و تعاملات بین هر کدام را مشخص کند و پیکربندی های مختلف مربوط به استقرار و یا deployment این component ها را تنظیم نمایند.

یک معماری نرم افزار را می توان به روش های مختلف تعریف کرد، که سه مورد از مهمترین آنها شامل موارد زیر می باشند:

۱. **استفاده از UML و یا Unified Modeling Language** : احتمالاً می‌دانید که UML یک زبان مدلسازی برای نرم افزار های شی گرا می باشد که در روند طراحی و مدلسازی نرم افزار مورد استفاده قرار می گیرد.
۲. **روش Architecture View Model و یا ۱+۴ :view model** مدل روش Architecture View Modelنمایانگر نیازمندی های عملکردی و غیر عملکردی یک سیستم نرم افزاری است. منظور از نیازمندی های عملکردی و یاfunctional ، نیازمندی هایی هستند که جزئی از نیازمندی های مطرح شده توسط کاربر می باشند و نیازمندی های غیر عملکردی و یاnon-functional ، شامل مواردی از قبیل؛ کارایی و قابلیت نگهداری می باشند.
۳. **روش ADL و یا :Architecture Description Language** منظور از روشADL ، روشی برای تعریف کردن رسمی و معنایی یک سیستم نرم‌افزاری است.

زبان مدلسازیUML

واژه UML مخفف Unified Modeling Language و یا زبان مدلسازی یک شکل می باشد. زبانUML ، یک زبان بصری است که برای ایجاد کردن نقشه‌های سیستم‌های نرم‌افزاری مورد استفاده قرار می‌گیرند. در ابتدا UML توسط Object Management Groupو یا OMG مطرح شد. استاندارد UML 1.0 توسط OMG در ژانویه سال ۱۹۷۰ پیشنهاد شد. زبان مدلسازی UML به عنوان یک استاندارد برای مستند سازی طراحی و تجزیه و تحلیل نیازمندی های نرم افزار که اساس توسعه نرم‌افزار می‌باشد، بنا نهاده شده است.
زبان مدلسازی UML می تواند به عنوان یک زبان مدلسازی بصری همه منظوره برای ساخت، تعریف و مستند سازی سیستم‌های نرم افزاری به صورت بصری مورد استفاده قرار بگیرد. هرچند که زبان UML اغلب برای مدل کردن سیستم های نرم افزاری مورد استفاده قرار می‌گیرد، اما محدود به این قضیه نیست. از زبان UML می‌توان برای مدل سازی سیستم های غیر نرم افزاری نیز استفاده کرد.

در زبان مدلسازی UML عناصری وجود دارند که شبیه به component ها و یا اجزای تشکیل دهنده یک سیستم نرم افزاری، به طرق مختلف با یکدیگر در ارتباط هستند تا مدلسازی کامل UML که تحت عنوان نمودار و یا diagram نیز شناخته می‌شود کامل بگردد. به همین دلیل درک نمودارهای مختلف UML برای پیاده سازی دانش مربوطه در یک سیستم دنیای واقعی بسیار مهم است. به طور کلی؛ diagram ها و یا نمودارهای UML را به دو دسته با نام‌های structural diagram و یا نمودار های ساختاری و behavioral diagram و یا نمودار های رفتاری تقسیم بندی می کنند.

نمودارهای ساختاری و یا structural diagram ها

نمودارهای ساختاری اغلب نمایانگر جنبه‌های ایستا و غیرقابل تغییر یک سیستم می باشند. این جنبه های ایستا، نمایانگر بخش‌هایی از نمودار هستند که ساختار کلی یک سیستم نرم‌افزاری پایدار را نشان می‌دهند. این قسمت های ایستا، در یک سیستم نرم‌افزاری، با استفاده از کلاس ها، interfaceها، اشیاء، componentها و گره ها نشان داده می شوند. نمودارهای ساختاری را به دسته های زیر تقسیم بندی می کنیم:

۱. **نمودارهای class diagram** : این نمودار نمایانگر خاصیت شی گرایی و یا object orientation یک سیستم است و نحوه ارتباط کلاس‌های مختلف در یک سیستم نرم‌افزاری با یکدیگر را نشان می‌دهند.
۲. **نمودارهای object diagram** : نمایانگر مجموعه از object ها و ارتباط بین آنها در زمان اجرا و یا runtimeبرنامه و همچنین یک view و یا دید ایستا از سیستم است.
۳. **نمودارهای component diagram** : نمایانگر تمامی component ها یا اجزای تشکیل دهنده یک سیستم، ارتباط بین آنها، تعامل و همچنین interface یک سیستم است.
٤. **نمودارهای deployment diagram** : نمایانگر مجموعه‌ای از گره‌ها و یا node ها و ارتباط بین آنها است. این node ها در واقع موجودیت‌های فیزیکی هستند که component های نرم افزاری بر روی آنها deploy و یا مستقر خواهند شد.
۵. **نمودارهای package diagram** : نمایانگر ساختار package و سازماندهی یک نرم‌افزار است. این نمودار شامل کلاس های درون یک package و همچنین package های درون دیگر package می شود.
٦. **نمودارهای composite structure** : نمایانگر ساختار درونی component ها با لحاظ class ها، interfaceها و component های دیگر می باشد.

نمودارهای رفتاری و یا behavioral diagram ها

نمودارهای رفتاری و یا behavioral diagram ها، به طور کلی؛ جنبه های پویا و یا dynamic یک سیستم را بررسی می‌کنند. منظور از جنبه‌های پویا، آن دسته از قسمت های از یک سیستم می باشد که در حال تغییر کردن و یا حرکت کردن هستند. نمودارهای رفتاری در زبان مدلسازی UML به موارد زیر تقسیم می‌شوند:

۱. **نمودار use case** : نمایانگر رابطه بین functionality ها و internal controllers ها و یا external controllers ها می باشند. این کنترلر ها را اغلب Actor و یا عامل می‌نامند.
۲. **نمودار activity** : نمایانگر جریان کنترل سیستم است و شامل activity ها و لینک ها و یا ارتباطات بین آنها می باشند. جریان می‌تواند ترتیبی و یاsequential ، هم روند و یا concurrentو شاخه شده و یا branched باشد.
۳. **نمودار state machine و یا state chart** : نمایانگر تغییرات حالت سیستم بر اساس رویدادهایی می باشد که در سیستم رخ می دهند. این نمودار تغییرات کلاس‌ها، interface ها و دیگر موارد را نشان می‌دهند و برای بصری کردن تعاملات و واکنش های یک سیستم به عوامل داخلی و خارجی مورد استفاده قرار می گیرد.
٤. **نمودار sequence** : نمایانگر دنباله ای از فراخوان ها درون سیستم و انجام functionalityهای خاص است.
۵. **نمودار interaction overview** : ترکیب نمودارهای activity و sequence به منظور فراهم کردن یک دید کلی از جریان کنترل و یا control flow مربوط به سیستم و البته فرآیند تجاری را با این نمودار نشان می دهند.

٦. **نمودار communication** : شبیه به نمودار sequence diagram عمل می کند با این تفاوت که تمرکزش بر روی نقش هر object است. هر ارتباط و یا communication با یک sequence order و البته، تعداد آنها به علاوه message ها و یا پیام های رد و بدل شده تشکیل می‌گردد.
٧. **نمودار time sequenced** : نمایانگر تغییراتی است که message ها در state و conditionو event ها ایجاد کرده اند.

بررسی Architecture View Model در معماری نرم افزار

منظور از یک مدل، یک توصیف ساده شده و پایه‌ای و البته کامل از معماری نرم‌افزار است که از view های مختلف به یک نرم افزار نگاه می‌کند. منظور ازview ، یک منظر و یا دید است. یکview ، در واقع یک نمایش از یک سیستم به طور سراسری، از یک نقطه نظر خاص و البته موارد و concern های مرتبط با هم است. از یک view برای توصیف سیستم، از نقطه نظرهای مختلفی که ذینفعان شبیه به کاربران نهایی، برنامه نویسان، مدیران پروژه و tester ها به نرم‌افزار نگاه می‌کنند، استفاده می‌شود.

بررسی روش view model۱+۴

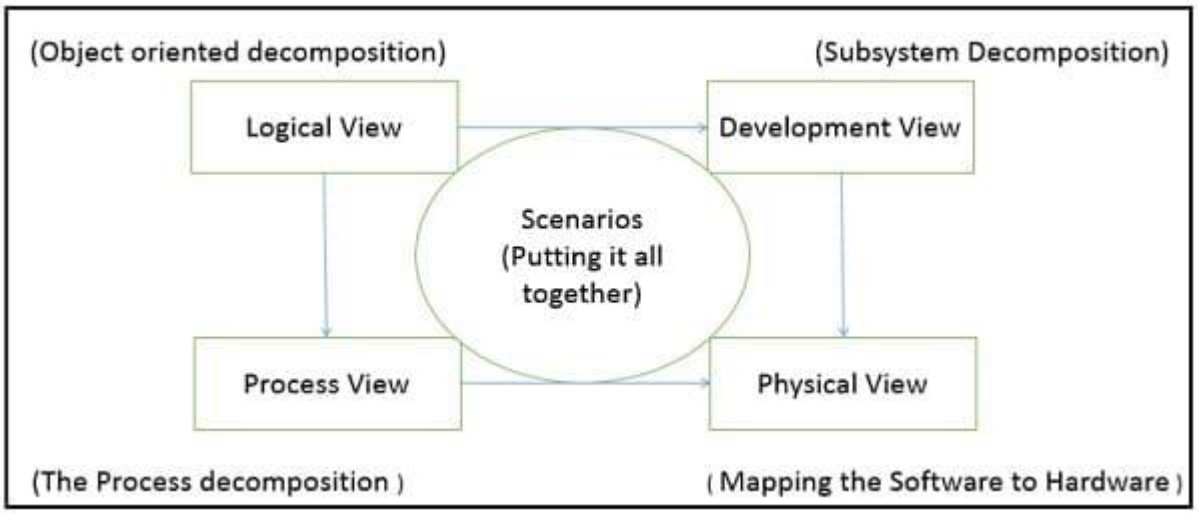
روش ۱+۴ view model توسط Philippe Kruchten طراحی شد و به منظور توصیف معماری یک سیستم نرم افزاری بر اساس به کار بستن چندین view همروند و موازی مورد استفاده قرار می گیرد. خاصیت اصلی این روش، همین چند view بودن مدل آن است که می‌تواند ویژگی‌ها و concern های مختلف یک نرم‌افزار را بررسی کند. علاوه بر این؛ این روش می‌تواند مستندات طراحی سیستم های نرم افزاری را استاندارد کرده و درک طراحی مورد نظر را برای ذینفعان آسان کند.

روش ۱+۴ view model یک روش architecture verification و یا اصطلاحاً ارزیابی معماری است که به منظور بررسی دقیق و مستند سازی طراحی و معماری یک نرم‌افزار از جنبه‌های مختلف، مورد استفاده قرار می گیرد. به طور کلی؛ در این روش چهار view به قرار زیر تعریف می شوند:

١. **logical viewو یا :conceptual view** نمایانگر یک object model و یا مدل شی گرا از طراحی است.
٢. **:process view**نمایانگر activity ها و فعالیت های سیستم و البته جنبه های concurrencyو synchronization سیستم می باشد.
٣. **:physical view**نمایانگر نگاشت نرم‌افزار به سخت افزار مورد نظر و بازتاب جنبه های توزیع شده ی نرم‌افزار است.
٤. **:development view**نمایانگر سازماندهی ایستا و یا ساختار نرم‌افزار در محیط توسعه می باشد.

علاوه بر چهار view تعریف شده؛ می‌توان این روش را با اضافه شدن یک view دیگر با عنوان scenario view و یا use case viewبرای کاربران نهایی و یا مشتریان توسعه داد. این view با ۴ مورد view قبلی به خوبی ترکیب شده و به منظور

به تصویر کشیدن معماری، مورد استفاده قرار می گیرد. به همین دلیل است که به این روش ۴+۱ view model می‌گویند. تصویری که در قسمت زیر مشاهده می کنید، این روش معماری را با استفاده از ۵ view به نمایش گذاشته است.



چرا به این روش ۴+۱ می‌گویند و نه روش ۵

اگر بخواهید بدانید که چرا نام این مدل، ۴+۱ view model گذاشته شده است و نه ۵ view باید بگویم که view آخر که تحت عنوان use case view اضافه شد، اهمیت ویژه‌ای دارد. چراکه جزئیات نیازمندی یک سیستم نرم‌افزاری را از سطح بالا بررسی می کند. این در حالی است که دیگر view ها، این نیازمندی ها را از نقطه نظرهای متفاوتی مورد بررسی قرار می دهند. زمانی که هر ۴ view کامل بشود، طراحی و معماری نهایی شکل می گیرد. البته؛ یک نوع موضوع بسیار مهم این است که با وجود ۴ view اول به صورت کامل، عملاً view آخر بیهوده خواهد بود. با این وجود نکته بسیار مهم این است که هیچکدام از چهار view قبلی، بدون view آخر کامل نخواهند شد.

در عکس زیر، جزئیات بیشتری از ۴+۱ view model در دسترس است.

	Logical	Process	Development	Physical	Scenario
Description	Shows the component (Object) of system as well as their interaction	Shows the processes / Workflow rules of system and how those processes communicate, focuses on dynamic view of system	Gives building block views of system and describe static organization of the system modules	Shows the installation, configuration and deployment of software application	Shows the design is complete by performing validation and illustration
Viewer / Stake holder	End-User, Analysts and Designer	Integrators & developers	Programmer and software project managers	System engineer, operators, system administrators and system installers	All the views of their views and evaluators
Consider	Functional requirements	Non Functional Requirements	Software Module organization (Software management reuse, constraint of tools)	Nonfunctional requirement regarding to underlying hardware	System Consistency and validity
UML – Diagram	Class, State, Object, sequence, Communication Diagram	Activity Diagram	Component, Package diagram	Deployment diagram	Use case diagram

بررسی Architecture Description Languages در معماری نرم افزار

زبان هایADL ، در واقع زبان‌هایی هستند که حاوی syntax و semantic مناسب برای تعریف کردن یک معماری نرم افزار هستند. منظور ازsemantic ، جنبه های معنایی و منظور ازsyntax ، جنبه های املایی و دستوری است. در واقع یک زبانADL ، یک زبان استانداردسازی است که می‌تواند قابلیت‌های یک سیستم نرم افزاری را از نقطه نظر ادراکی و یا conceptualبه سادگی مدلسازی کند و معماری را از بُعد پیاده سازی سیستم تفکیک نماید.

زبانهای ADL می بایست از component های معماری پشتیبانی کرده و ارتباطات، interfaceها و پیکربندی های معماری که در واقع اجزای تشکیل دهنده اصلی معماری به حساب می آیند را توصیف کند. زبان های ADL یک روش بیان معماری می باشند که توانمندی تجزیه کردن component ها، ترکیب نمودن آنها با یکدیگر و همچنین تعریف نمودن interface های آنها را دارد.

یک زبان ADL یک زبان استانداردسازی رسمی است که ویژگی‌های یک نرم‌افزار از قبیل؛ فرآیندها، threadها، داده، زیر برنامه ها و همچنین component های سخت افزاری از قبیل؛ پردازنده ها، دستگاه‌ها، خطوط ارتباطی و حافظه را توصیف می‌کند. دسته بندی کردن و یا تمییز دادن زبان های ADL از زبان های برنامه نویسی یا زبان های مدل سازی، کار ساده ای نیست. اما به طور کلی؛ نیازمندی های یک زبان که آن را در دسته زبان های ADL قرار می دهد شامل موارد زیر می باشند:

۱. باید برای برقراری ارتباط و انتقال مفاهیم معماری به تمامی ذینفعان پروژه مناسب باشد.
۲. می بایست برای وظایفی از قبیل؛ ایجاد کردن، پالایش کردن و اعتبارسنجی کردن معماری مناسب باشد.
۳. می بایست بنایی را برای پیاده سازی های بعدی ایجاد کند و امکان اضافه کردن اطلاعات به تعاریف ADL را به منظور فعال نمودن سیستم نهایی بدهد.

٤. می بایست بتواند سبک های معماری معمول را به سادگی ایجاد کند.
٥. می‌بایست بتواند قابلیت‌های تجزیه و تحلیل ای داشته باشد و پیاده سازی نمونه های اولیه را به سادگی انجام دهد.

رویکرد شی گرا (object oriented) در معماری نرم افزار

رویکرد شیء گرایی در معماری نرم افزار، مفاهیم اولیه خود را از یک روش جدید برنامه نویسی اتخاذ کرد که در آن علاقه زیادی به طراحی و تجزیه و تحلیل شی گرا بود. شی گرایی که پیش از این ممکن است به صورت OO از آن یاد کنیم، یک رویکرد طراحی و تجزیه و تحلیل بود که پیش از آن توسط زبان های برنامه نویسی مختلف اتخاذ شد.

اولین زبان برنامه نویسی شی گرا زبان simula بود که در سال ۱۹۶۰ توسط محققین مرکز دانشگاهی نروژ ایجاد شد. در سال ۱۹۷۰ بود که آقای Alan Kay و گروه تحقیقاتی آن، یک سیستم کامپیوتری شخصی به نام Dynabook را ایجاد کرد و اولین زبان برنامه نویسی شی گرا به نام OOPL را ایجاد نمود. پس از آن در دهه ۱۹۸۰ آقای Grady Booch اولین مقاله رسمی خود را با عنوان object oriented design و یا طراحی شی گرا منتشر کرد که تمرکز آن بر روی پیاده سازی مفاهیم و طراحی شی گرا در زبان برنامه نویسی Ada بود. در نسخه های بعدی این مقاله، آقای Grady Booch ایده ها و مفاهیم شی گرایی را بر روی دیگر زبان های برنامه نویسی مطرح نمود و نهایتاً در دهه ۱۹۹۰، Coadروش های شی گرایی و همچنین ویژگی های رفتاری این رویکرد را مطرح کرد.
علاوه بر روش های شی گرایی دیگر روش ها از قبیل؛ object modeling technique و یا تکنیک های مدل سازی شی گرا توسط James Rum Baugh و همچنین مهندسی نرم افزار شی گرا و یا object oriented software engineeringتوسط Ivar Jacobson هم مطرح شدند.

معرفی رویکرد شی گرا

رویکرد شی گرا و یا object oriented ، یک متدولوژی و روش بسیار مهم برای توسعه نرم‌افزارهای موفق می باشد. بسیاری از سبک های معماری و الگوهایی که برای مهندسی و طراحی نرم‌افزار مورد استفاده قرار می‌گیرند، بر اساس همین رویکرد بنا نهاده شده اند. برخی از آنها شامل؛ pipe و filter ها، الگوهای date repository و سبک های معماری مبتنی بر component می‌باشند. در ادامه در رابطه با مهمترین مفاهیم برنامه نویسی شی گرا، صحبت خواهیم کرد.

بررسی شیء و یاobject

در طراحی و رویکرد شی گرا، منظور از یک object و یا شیء، یک عنصر در دنیای واقعی است که می‌تواند در یک فضای شیء گرا نیز، برخی از ویژگی ها و مفاهیم ادراکی آن شیء در دنیای واقعی را داشته باشد. برخی از ویژگی های مهم اشیاء و یا objectها در رویکرد شی گرایی شامل موارد زیر می باشند:

١. هر object و یا شیء دارای یک هویت و یا شناسه که تحت عنوان identity نیز از آن یاد می‌شود، می باشد. این identity وظیفه آن را دارد که یک object را در یک سیستم کامل از دیگر object ها متمایز و آن را بشناسد.

۲. حالت و یا state که نمایانگر ویژگی ها و خصیصه های یک object و علاوه بر این، مقادیر مربوط به هرکدام از آن property ها و ویژگی ها می باشد.
۳. رفتار و یا behavior که شامل فعالیت هایی است که یک object می‌تواند انجام بدهد تا بر روی state و یا مجموعه property های خود تغییر ایجاد کند.

به طور کلی؛ object ها می‌توانند بر اساس نیاز یک application و یا سیستم نرم‌افزاری، مدل سازی شوند. یک object می تواند یک موجودیت فیزیکی، شبیه به یک مشتری، یک ماشین و یا غیره باشد و یا یک موجودیت ادراکی غیرملموس شبیه به یک پروژه، پردازش و یا غیره باشد.

بررسی یک کلاس

در رویکرد شی گرایی، یک class نمایانگر مجموعه ای از object ها می باشد که شامل ویژگی‌ها و یا property های یکسان و البته behavior ها و رفتارهای مشترک هستند. در واقع یکclass ، نقشه و یا تعریف کلی یک سری object را ایفا می‌کند که این object ها از آن class ساخته می شوند. به روال ایجاد کردن یک object از یکclass ، اصطلاحاً روال نمونه سازی و یا instantiation می گویند. به همین دلیل است که یک object را یک نمونه از یک کلاس و یا اصطلاحاً یک instance از آن class می نامند. اجزای تشکیل دهنده یک class شامل موارد زیر می باشند:

۱. مجموعه‌ای از ویژگی‌ها و یا property های object هایی که قرار است از این class نمونه‌سازی شوند. به طور کلی object های مختلفی که از یک class ساخته می‌شوند، در مقادیری که در هر کدام از این property ها قرار می‌دهند، تفاوت هایی با یکدیگر دارند. اغلب به این property ها و یا مکان هایی که در آنها داده قرار می‌گیرند، attributeو یا class date نیز می گویند.
۲. علاوه بر attribute ها، یک کلاس حاوی مجموعه ای از رفتارها و یا عملیاتی است که در قالب متد ها و یا operation در کلاس تعریف می شوند و می توانند بر روی داده های کلاس عمل کنند. به این نوع از رفتارها اصطلاحاً function و یا تابع و یا method می گویند.

بررسی یک مثال

برای اینکه تمایز و تفاوت دو مفهوم class و شیء را بهتر فرا بگیرید، بگذارید مثالی را بررسی کنیم. فرض کنید که یک کلاس با نام circle تعریف کرده ایم که نمایانگر یک شیء هندسی، یعنی همان دایره، در فضای دو بعدی است. حال با خود فکر کنید که یک class با نام circle می‌تواند دارای چه ویژگی ها و یا attribute هایی باشد. برخی از آنها را در قسمت زیر قرار داده‌ایم:

۱. مختصات مرکز دایره، مختصات مرکز دایره در دستگاه دکارتی، شعاع دایره و مواردی از این دست

علاوه بر این می توانیم بر روی این دایره عملیاتی نیز تعریف کنیم که شامل موارد زیر می باشند:

۱. متد findArea برای محاسبه کردن محیط دایره

۲. متد findCircumference برای محاسبه مساحت دایره
۳. متد scale برای تغییر اندازه دادن شعاع دایره

مفاهیم مربوط به برنامه نویسی و طراحی شی گرا

کپسوله سازی و یاencapsulation

یکی از مهمترین مفاهیم مربوط به برنامه نویسی و طراحی شی گرا، مفهوم encapsulation و یا کپسول سازی است. منظور از کپسوله سازی، فرآیند انتصاب دادن ویژگی ها و یا همان attribute ها و رفتارها و یا همان متدهای یک class به یکدیگر است. با استفاده از encapsulation جزئیات درونی یک کلاس می‌تواند از فضای بیرونی نیز محافظت و البته مخفی گردد. این موضوع باعث می‌شود که عناصر موجود در یک کلاس بتوانند فقط از طریق یک واسط و یا interface مشخص شده، توسط دیگر کلاس ها مورد دسترسی قرار بگیرند. این interface را خود کلاس مورد نظر ایجاد کرده و به دیگر کلاسهای اجازه می‌دهد که از طریق این interface با آن در ارتباط باشند.

چندریختی و یاpolymorphism

یکی دیگر از مفاهیم مربوط به طراحی و برنامه نویسی شی گرا، مفهوم چندریختی و یا polymorphism است که بر اساس آن یک کلاس و یا object ساخته شده از آن کلاس می تواند شکل های مختلفی را به خود بگیرد. در رویکرد شی گرایی مفهومpolymorphism ، استفاده کردن از عملیات مختلف در روش‌های متنوع است و اینکه در هر کدام از این شرایط چه اتفاقی می‌افتد، بستگی به آن object دارد که عملیات مورد نظر بر روی آن انجام می شود. مفهوم polymorphism اجازه می دهد که object ها و ساختارهای درونی متفاوتی را داشته باشند و یک interface خارجی یکسان را دارا باشند. مفهوم polymorphismمی‌تواند در پیاده سازی وراثت و یا inheritance نیز بسیار موثر عمل کند.

برای درک هرچه بهتر مفهوم polymorphism بگذارید مثالی را بررسی کنیم. فرض کنید که دو کلاس با نام‌های Circle و Square داریم که هر کدام از آنها یک متد به نام findArea را تعریف کردند. هرچند که نام و اهداف این متد ها، در این دو کلاس یکسان است، اما جزئیات پیاده سازی درونی آنها، برای مثال روند محاسبه کردن مساحت برای این دو کلاس متفاوت است. زمانی که یک object از کلاس Circle متد findArea را صدا می‌زند، عملیات محاسبه کردن مساحت برای یک دایره بدون هیچگونه مشکلی و برخوردی با متد findArea در کلاس Square انجام می‌پذیرد. علاوه بر این؛ برای تعریف کردن یک سیستم، می بایست که ویژگی‌های ایستا که همان ویژگی‌های منطقی نیز هستند، نسبت به ویژگی های پویا و یا ویژگی های رفتاری فراهم گردند. ویژگی های پویا نمایانگر ارتباط بین object های مختلف یک class است که تحت عنوان message passingنیز از آن یاد می شود. همچنین، ویژگی های ایستا، نمایانگر ارتباط بین کلاس ها از قبیل؛ aggregationو association و inheritance است.

انتقال پیام و یاmessage passing

در طراحی شی گرا، هر برنامه ای ممکن است بخواهد با object های مختلف ارتباط برقرار کند و از این طریق به پیشبرد برنامه کمک کند .object های درون یک سیستم ممکن است بخواهند با استفاده از مکانیزم‌های مختلفی از قبیل؛ message

passingو یا انتقال پیام با یکدیگر در ارتباط باشند. فرض کنید که دو object با نام‌های obj1 و obj2 در یک سیستم تعریف شدند. به این ترتیب؛ object اول یعنی همان obj1 می‌تواند یک پیام را به object دوم یعنی obj2 ارسال کند و از این طریق، یکی از متدهای آن را اجرا نماید.

ترکیب و یا composition در مقابل اجتماع و یاaggregation

دو رابطه دیگر که می توانند بین کلاسها برقرار شوند و با استفاده از آنها، objectهای ساخته شده از این کلاس ها با یکدیگر در ارتباط باشند، ارتباط aggregation و composition است. این دو ارتباط اجازه می‌دهند که یک کلاس به صورت مستقیم در بدنه یک کلاس دیگر تعریف بگردد. بطور کلی رابطه aggregation را به صورت بخشی از یا part-of و یا دارد یا has-a تعریف می‌کنند. با استفاده از این دو رابطه می‌توان، از طریق یک کلاس و اجزای دیگر آن، که کلاس ها و object های دیگر هستند، رفت. یک aggregate object نیز در واقع یک object است که از یک و یا چند object دیگر تشکیل گردیده است.

رابطه انجمنی و یاassociation

یک نوع دیگر از رابطه بین object ها و یا کلاس ها، رابطه انجمنی و یا association است که با استفاده از آن می توان گروهی از object ها و یا کلاس هایی که رفتارهای یکسان و یا ساختارهای مشابه دارند را با یکدیگر گروه بندی کرد. در واقع رابطه association نمایانگر رابطه بین object ها و یا کلاس هایی است که تا حدودی مشابه هم هستند. این ارتباط با استفاده از یک پیوند و یا link اتفاق می افتد و از یک پیوند تحت عنوان یک نمونه از یک association یاد می شود. یک ماهیت دیگر برای این پیوند، درجه و یا degree است که نمایانگر تعداد کلاس هایی است که در یک ارتباط و پیوند شرکت داده می شوند. یک درجه و یا ارتباط می تواند تکین، دودویی و یا سه تایی باشد.

۱. روابط تکین، objectهای مربوط به یک کلاس یکسان را با یکدیگر مرتبط می کنند.
۲. رابطه های دودویی، objectهای مربوط به دو کلاس مختلف را با یکدیگر مرتبط می کنند.
۳. رابطه های سه تایی، سه object و یا تعداد بیشتری از object های مربوط به کلاس های مختلف را به یکدیگر مرتبط می کنند.

وراثت و یاinheritance

یکی دیگر از رابطه های بین object ها، رابطه وراثت است. با استفاده از این رابطه، کلاس های جدید می‌توانند بر اساس کلاس های از قبل موجود ساخته شوند و قابلیت‌های آنها را گسترده تر و یا حتی پالایش کنند. کلاس‌های از قبل موجود را اصطلاحاً base classو یا parent class و یا super class می نامیم و کلاس‌های که از آنها ساخته می‌شوند را اصطلاحاً derived classو یا child class و یا sub class می نامیم.

یک sub class می تواند ویژگی ها و یا همان attribute ها و متدها و یا همان رفتارهای super class را به ارث ببرد. علاوه‌براین، sub classمورد نظر می‌تواند attribute ها و متدهای خاص خود را نیز اضافه کند و یا حتی آن دسته از attribute ها و متدهای به ارث برده شده را تغییر بدهد. رابطه inheritance و یا وراثت، یک رابطه‌ی is-a و یا هست را مشخص می کند.

تجزیه و تحلیل شی گراobject-oriented analysis

در فاز تجزیه و تحلیل شی گرا از توسعه یک نرم افزار، نیازمندی های سیستم مشخص شده، کلاس‌های آنها تعریف گردیده و ارتباط بین کلاسها مشخص و معین می گردند. هدف تجزیه و تحلیل شی گرا، درک application domainو یا حوزه کاری و فعالیتی یک نرم افزار و البته نیازمندی های خاص یک سیستم است. نتیجه این کار مشخص شدن نیازمندی ها و تجزیه و تحلیل ابتدایی ساختار منطقی و قابلیت امکان پذیری سیستم می باشند.

سه مورد از تکنیک های تجزیه و تحلیل که در این قسمت مورد استفاده قرار می‌گیرند شامل؛ object modeling و dynamic modeling و functional modeling هستند که در ادامه در رابطه با آنها صحبت خواهیم کرد.

بررسی مدل سازی شی گرا و یا object modeling

منظور از تکنیکobject modeling ، توسعه دادن ساختار ایستای یک سیستم نرم‌افزاری از نقطه‌نظر object ها می باشد. این تکنیک می تواند object ها و class های مربوطه را مشخص کرده و آنها را در رابطه های معینی گروه بندی کند. علاوه بر این؛ با استفاده از این تکنیک می‌توانیم attribute ها و operation های خاص هر کلاس را نیز مشخص کنیم. فرایند object modelingو یا مدل سازی شی گرا، می تواند شامل مراحل زیر باشد:

۱. مشخص کردن object ها و گروه بندی کردن آنها در کلاس ها.
۲. مشخص کردن ارتباط بین کلاس ها.
۳. ایجاد کردن یک نمودار مدل شی‌گرا برای کاربر.
٤. تعریف کردن attribute های مربوط به object ها.
٥. تعریف کردن operation هایی که باید بر روی هر کدام از class ها انجام شود.

مدل سازی پویا و یاdynamic modeling

پس از اینکه رفتار ایستای مربوط به یک سیستم در قسمت object modeling تجزیه و تحلیل گردید، می بایست رفتار سیستم با در نظر گرفتن زمان و همچنین تغییرات خارجی مورد بررسی قرار بگیرد. به این تکنیک dynamic modeling و یا مدل سازی پویا می‌گویند.

dynamic modeling به عنوان روشی برای توصیف کردن اینکه یک object چگونه به رویدادها و دیگر اتفاقاتی که در سیستم رخ می‌دهند پاسخ می‌دهد، تعریف می گردد. فرآیند dynamic modeling را می توان در مراحل زیر انجام داد:

۱. مشخص کردن state و یا حالت هر کدام از object ها.
۲. مشخص کردن event ها و رویدادها و البته تجزیه و تحلیل کردن عملیاتی که در پاسخ به آنها باید انجام شود.

۳. ایجاد کردن یک نمودار dynamic modeling که از نمودارهای state transition diagramتشکیل شده است.
٤. مشخص کردن state هرکدام از object ها با استفاده از attribute های آنها.
٥. اعتبارسنجی کردن نمودارهای state transition و یا انتقال حالت هرکدام از object ها

مدلسازی تابعی و یاfunctional modeling

تکنیک آخری که در تجزیه و تحلیل شی گرا مورد استفاده قرار می‌گیرد، functional modelling است. منظور از یک functional modeling، مشخص کردن فرآیندهایی است که در یک object اتفاق می‌افتند و البته تغییراتی که بر روی داده های مربوط به هر کدام از آن object ها رخ می دهد. با استفاده از این تکنیک می‌توانیم معانی هرکدام از عملیاتی که بر روی یک object رخ می دهند را مشخص کنیم. یک functional model نمایانگر data flow diagram و یا نمودار جریان داده هر کدام از object ها می باشد. فرآیند functional modeling می‌تواند از مراحل زیر تشکیل بگردد:

۱. مشخص کردن تمامی ورودی ها و خروجی ها
۲. ساختن نمودار جریان داده و یا data flow diagram با استفاده از functional dependencyها
۳. مشخص کردن هدف هر کدام از function ها
٤. مشخص کردن قیود و یا constraint ها
٥. مشخص کردن قیود بهینه سازی

طراحی شی گرا(Object-Oriented Design)

در فاز طراحی شی گرا و یا همانObject-Oriented Design ، یک مدل ادراکی و یا conceptual model توسعه داده شده و به یک مدل شی گرا با استفاده از طراحی شی گرا تقدیم می گردد. در این فاز، مفاهیم مستقل از تکنولوژی که در فاز تجزیه و تحلیل شی گرا ایجاد گردیدند به پیاده‌سازی کلاس‌ها، قیود، interfaceها و دیگر عناصر موجود در یک مدل شی گرا تبدیل می‌گردند. در واقع هدف از فاز طراحی شی گرا، توسعه دادن ساختار معماری یک سیستم است. مراحل مختلف طراحی شی گرا شامل عناصر زیر می‌باشند:

۱. تعریف کردن context و یا فضای کاری یک سیستم
۲. طراحی کردن معماری سیستم
۳. مشخص کردن object های درون سیستم
٤. ساختن design model ها و یا مدل های طراحی
٥. مشخص کردن object interface ها و یا واسط های اشیا

در فاز طراحی شی گرا می توانیم مراحل مختلفی را نیز داشته باشیم که به طور کلی تحت مراحل conceptual design و یا طراحی ادراکی و detailed design و یا طراحی جزئی تنظیم می‌گردند.

طراحی ادراکی و یا conceptual design

در این قسمت تمامی کلاس ها مشخص می گردند و سیستم بر اساس آنها ایجاد می‌شود. علاوه بر این؛ وظایف هر کدام از کلاس ها نیز مشخص و تعیین می‌گردند. نمودار کلاس و یا class diagram برای مشخص کردن ارتباط بین کلاس ها مورد استفاده قرار می‌گیرد و البته نمودار تعامل و یا interaction diagram برای نشان دادن جریان event ها مورد استفاده قرار می‌گیرد. از قسمت طراحی ادراکی، تحت عنوان طراحی سطح بالا و یا high-level design نیز یاد می شود.

طراحی جزئی و یا detailed design

در این مرحله attribute ها و operation های مربوط به هر کلاس مشخص و به آن انتصاب داده می‌شود. این موضوع بر اساس نمودار تعامل و یا interaction diagram اتفاق می‌افتد. علاوه بر این؛ state machine diagram توسعه داده شده و با استفاده از آن، جزئیات مربوط به طراحی هر کدام از کلاس ها به طور مفصل تری مشخص می گردد. به این قسمت low-level designو یا طراحی سطح پایین نیز می گویند.

اصل decouple کردن و یا principle of decoupling

یکی از موضوعات مهم در طراحی شی گرای موفق، طراحی کردن یک سیستم با استفاده از class های مستقل از یکدیگر است. به عبارت دیگر، کلاس ها باید طوری طراحی بشوند که ایجاد تغییر در یک کلاس، باعث ایجاد تغییرات آبشاری در دیگر کلاس‌ها نشود. اگر قرار باشد که این در هم تنیدگی و یا اصطلاحاً tight coupling از بین برود، می‌بایست با استفاده از interface ها و دیگر تکنیک های شی گرا، طراحی سیستم انجام بپذیرد

برقراری چسبندگی و یا cohesion

یکی دیگر از اصول بسیار مهم در طراحی شی گرا های موفق این است که کلاس ها همواره باید در سطح مناسبی از چسبندگی قرار داشته باشند. به عبارت دیگر؛ یک کلاس باید مجموعه‌ای از attribute ها و function های مرتبط با یکدیگر باشد. فقدان cohesion به این معنی است که یک کلاس، شامل function های غیر مرتبط است. این موضوع باعث می‌شود که ساختار کلی یک سیستم نرم‌افزاری پیچیده شود و مدیریت کردن، گسترش دادن، نگهداری کردن و ایجاد تغییر در آن به مراتب دشوار بگردد.

اصل باز و بسته بودند و یا open-closed principle

بر اساس این اصل، یک سیستم باید برای پیاده‌سازی نیازمندی های جدید باز باشد و برای تغییر ایجاد کردن در پیاده سازی های از قبل موجود بسته باشد. به عبارت دیگر باید به سادگی بتوان در یک سیستم کدهای جدید را ایجاد کرد و تغییر کدهای از قبل نوشته شده دشوار باشد. این اصل، یکی از اصول پنج گانه SOLID است.

معماری جریان داده(data flow architecture)

در معماری جریان داده و یا data flow architecture تمامی یک سیستم نرم افزاری به عنوان دنباله ای از تغییرات بر روی مجموعه ای از داده های ورودی پشت سر هم دیده می‌شوند. که در آن؛ داده ها و عملیات اتفاق افتاده بر روی داده ها مستقل از یکدیگر هستند. در این رویکرد؛ در ابتدا، داده به درون سیستم وارد شده و از module ها و component های مختلفی که در سیستم وجود دارند، یکی بعد از دیگری عبور می کند، تا زمانی که به مقصد نهایی خود برسد. این مقصد نهایی می تواند یک مخزن ذخیره سازی و یا data store باشد.

اتصالات و ارتباطات مابین module ها و component ها در این سیستم، به روش های مختلفی از قبیل؛ I/O stream ها و I/O buffer ها و pipe ها، پیاده سازی می گردند. داده ها در سیستم می‌توانند بر اساس توپولوژی های مختلف گراف ها به حرکت در بیایند. برخی از این توپولوژی ها شامل؛ توپولوژی های مبتنی بر cycle و یا چرخه، توپولوژی های مبتنی بر خطوط صاف و یا linear که بدون چرخه هستند و یا حتی ساختارهای شبیه درخت ها می باشند.

هدف اصلی این رویکرد؛ به دست آوردن کیفیت های قابلیت استفاده مجدد و همچنین قابلیت تغییر پذیری می باشد. این رویکرد برای آن دسته از سیستم های نرم افزاری مفید است که شامل مجموعه‌ای خوش تعریف از تغییرات مستقل بر روی داده ها و یا محاسبات ترتیبی بر روی داده های ورودی می باشند. این داده های ورودی نهایتاً به عنوان خروجی از سیستم خارج شده و تحویل نرم افزارهای دیگری از قبیل نرم افزارهای پردازش داده های تجاری و یا حتی compiler ها می‌شوند. به طور کلی؛ سه روش برای پیاده سازی کردن دنباله عملیات اجرایی بر روی داده ها و ما بین module ها وجود دارند که عبارتند از؛

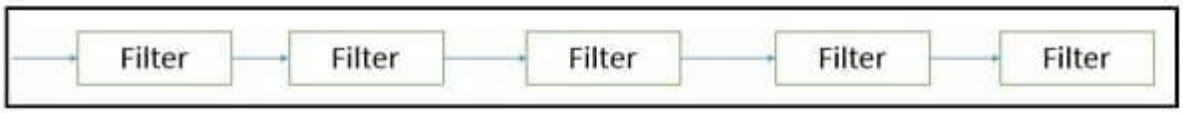
۱. سیستمbatch sequential
۲. سیستم pipe and filter که با نام non-sequential pipeline mode نیز شناخته می شود
۳. سیستمprocess control

سیستمbatch sequential

در سیستم batch sequential که یک مدل پردازش داده سنتی است، یک زیر سیستم تغییر داده، فقط زمانی می‌تواند فرآیند خود را آغاز کند که زیر سیستم قبلی به صورت کامل عملیات خود را تکمیل کرده باشد.

۱. در این سیستم، جریان داده ها، یک مجموعه و یا دسته که معنی کلمه batch می باشد را که از داده ها تشکیل شده است، از یک زیر سیستم به سیستم دیگر حمل می‌کند.
۲. ارتباطات بین module ها بر اساس فایل های موقت میانی شکل می گیرد که می‌توانند توسط زیر سیستم های بعدی حذف گردند.
۳. از این روش برای آن دسته از نرم‌افزارهای استفاده می‌شود که در آن ها داده ها به صورت دستی و یا batch مورد استفاده قرار گرفته و هر زیر سیستم، فایل های ورودی مرتبط با خود را خوانده و در فایل های خروجی داده هایی را می نویسند.

٤. کاربرد معمول این معماری شامل؛ نرم افزارهای پردازش داده های تجاری در حوزه‌های مختلف از قبیل؛ بانکداری و صدور صورتحساب می‌باشد. تصویر زیر مثالی از این موضوع را نشان می‌دهد.



در ادامه در رابطه با مزایا و معایب این سیستم صحبت خواهیم کرد.

مزایای سیستم batch sequential

١. مهمترین مزیت این سیستم، فراهم کردن تفکیک ساده و مناسب مابین زیر سیستم ها می باشد.
٢. علاوه بر این؛ هر زیر سیستم می‌تواند به عنوان یک برنامه کاملاً مستقل، بر روی مجموعه ای از داده های ورودی کار کند و داده های خروجی خاص خود را تولید نماید.

معایب روش batch sequential

١. یکی از مهمترین معایب این روش ایجاد کردن latency و یا تاخیر در ایجاد شدن خروجی کار و همچنین بهره بری پایین می باشد.
٢. علاوه بر این؛ انجام عملیات همروندی و یا concurrency ، در این نوع از سیستم ها کمی دشوار است.
٣. گذشته از تمامی این موارد؛ کنترل از بیرون و یا اصطلاحاً external control برای پیاده‌سازی این گونه از سیستم ها مورد نیاز است.

معماری pipe و filter یا pipe and filter architecture

در این قسمت در رابطه با معماری pipe و filter و یا pipe and filter architecture صحبت خواهیم کرد. در این رویکرد؛ تمرکز بر روی تغییرات افزایشی و مرحله‌به‌مرحله داده هاست که توسط component های پشت سر هم اتفاق می‌افتد. در این رویکرد؛ جریان داده‌ها توسط خود داده ها اتفاق می‌افتد و تمامی یک سیستم به component های کوچکتر تجزیه می شود. این component ها شامل؛ data source ها و filter ها و pipe ها data sink ها می باشد. ضمناً به دلیل فنی بودن این واژگان از ترجمه کردن آنها جلوگیری کرده‌ایم.

علاوه بر این؛ ارتباط و اتصال بین ماژول های مختلف، مبتنی بر data stream ها می باشند. در واقع data stream ها bufferهایی هستند که به صورت first in first out کار می‌کنند و می‌توانند از stream هایی از نوع byte و characterو یا هر نوع داده ای دیگری باشند. ویژگی اصلی این معماری، اجرا شدن عملیات به صورت موازی و همچنین افزایشی می‌باشد.

فیلتر (filter) چیست؟

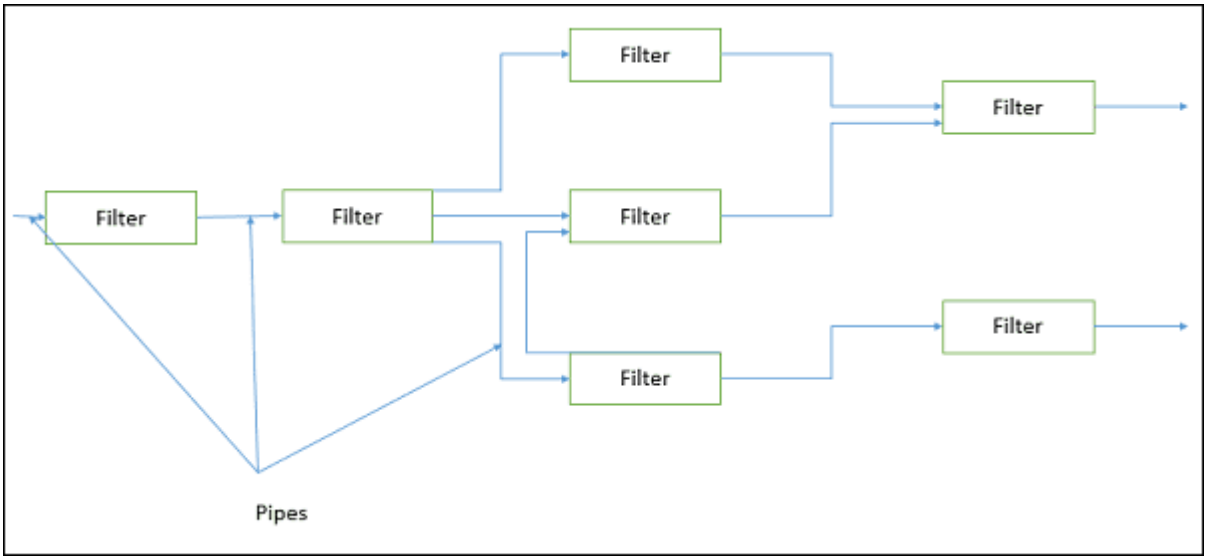
در این سبک معماری، ماهیت filter مطرح می شود که یک تغییر دهنده data stream ها است که به صورت مستقل کار می کند. وظیفه یک فیلتر، تغییر دادن داده های ورودی، پردازش کردن آنها و سپس نوشتن آنها به درون data stream خروجی می باشد. این کار با استفاده از یک pipe اتفاق می‌افتد تا فیلتر بعدی بتواند داده ها را دریافت کرده و آنها را پردازش کند. فیلترها به صورت incremental و یا افزایشی عمل می‌کنند و این بدان معناست که به محض ورود داده ها به درون سیستم، عملیات پردازش اتفاق می‌افتد. به طور کلی دو نوع فیلتر با نام‌های active filter و passive filter وجود دارند. در ادامه به بررسی هر کدام از این دو مورد خواهیم پرداخت.

بررسی active filter ها

یک active filter اجازه می دهد تا pipe های متصل، داده‌ها را خوانده و سپس آنها را به درون خروجی بنویسند. فیلترهای فعال و یا active filter ها با passive pipe ها کار میکنند که وظیفه آنها فراهم کردن مکانیزم هایی برای خواندن و نوشتن داده ها می باشد. این موضوع در pipe های مربوط به UNIX مورد استفاده قرار می گیرد.

بررسیpassive filter

یک passive filter اجازه می دهد که pipe های متصل، داده ها را نوشته و همچنین آنها را بخوانند .passive filter ها با active pipe ها کار می کنند که وظیفه آنها خواندن داده ها از درون یک فیلتر و نوشتن آن ها به درون فیلتر بعدی می باشد. اینگونه از فیلترها می بایست مکانیسم‌های خواندن و نوشتن را نیز فراهم کنند. این موضوع در تصویر زیر نشان داده شده است.



مزیت های سیستمpipe and filter architecture

۱. یکی از مهم‌ترین مزیت‌های مربوط به این، سیستم امکان کار کردن به صورت موازی و ایجاد بهره‌وری بسیار بالا برای حجم داده های در حال پردازش سنگین می باشد.
۲. قابلیت استفاده مجدد و ساده تر شدن سطح نگهداری و یا maintenance سیستم نیز از دیگر مزیت‌های این معماری به حساب می‌آید.
۳. اضافه کردن قابلیت تغییر پذیری و همچنین coupling کم مابین فیلترها از دیگر مزیت های این سیستم به حساب می آیند.
٤. این سیستم با سادگی خاص خود می‌تواند تفکیک مناسبی بین فیلترها و pipe ها ایجاد کند.
٥. انعطاف پذیری این سیستم و پشتیبانی کردن از اجرای موازی و ترتیبی نیز از دیگر مزیت های آن است.

معایب سیستمpipe and filter architecture

۱. علیرغم مزیت های بی شمار این سیستم تعداد معایب آن نیز کم نیست. یکی از مهم‌ترین آنها عدم مناسب بودن این سیستم برای تعاملات پویا است.
۲. علاوه بر این؛ برای تغییرات بر روی فرمت ASCII نیاز به تنظیمات اضافی وجود دارد.
۳. سربار انجام دادن تغییرات بر روی داده ها، مابین فیلترها، نیز از دیگر عیب های این سیستم است.
٤. این سیستم روشی را برای فیلترها فراهم نمی‌کند تا بتوانند به صورت همکارانه اقدام به حل و فصل کردن یک مسئله کنند.
٥. مورد آخر اینکه پیکربندی کردن این معماری به صورت پویا کار ساده ای نیست.

خط لوله (pipe) چیست؟

خط لوله و یا pipe ها در واقع stateless و یا بدون حالت هستند و وظیفه اصلی آنها انتقال stream هایی از نوع characterو یا داده های binary مابین دو فیلتر می باشد .pipe ها می توانند یک data stream را از یک فیلتر به یک فیلتر دیگر منتقل کنند. با استفاده از pipe ها، اطلاعات مختلفی منتقل می گردند. اما به هیچ وجه هیچ گونه اطلاعاتی در رابطه با state مربوط به فیلتر ها جابجا نمی گردد.