

معماری مقیاس بزرگ

خانم دکتر سحر آدابی

علیرضا سلطانی نشان

۴ آبان ۱۴۰۳

فهرست مطالب

۲	۱ پیشگفتار
۲	۲ معرفی
۲	۱.۲ چه زمانی یک پروژه مقیاس بزرگ است؟
۳	۲.۲ یادآوری متدولوژی RUP
۳	۱.۲.۲ منظور از نظم در RUP
۳	۲.۲.۲ چهار فاز اصلی در RUP
۳	۳.۲.۲ منظور از فرسخ‌شمار چیست؟
۴	۴.۲.۲ محوریت بر روی نیازمندی‌ها
۴	۵.۲.۲ استفاده از برنامه‌نویسی OOP
۴	۳ معماری مقیاس بزرگ نرم‌افزار
۴	۱.۳ معماری نرم‌افزار چیست؟
۴	۲.۳ معمار نرم‌افزار کیست؟
۴	۳.۳ المان‌ها
۴	۴.۳ External Feasible Properties
۵	۵.۳ تفاوت کامپوننت و المان
۵	۶.۳ بخش‌هایی که معماری نرم‌افزار باید پوشش دهد
۵	۷.۳ قابلیت اطمینان یا Reliability
۵	۸.۳ قابلیت استفاده یا Useability
۵	۹.۳ ارائه سریع محصول یا Short time to market
۵	۱۰.۳ تفاوت معماری سازمانی و معماری نرم‌افزار
۶	۱۱.۳ جزئیات فعالیت‌های معماری نرم‌افزار
۶	۱.۱۱.۳ ساخت Business case برای سیستم
۶	۲.۱۱.۳ فهمیدن نیازمندی‌های پروژه
۶	۳.۱۱.۳ ایجاد یا انتخاب یک معماری نرم‌افزار
۶	۴.۱۱.۳ دنبال کردن معماری یا Communicating the architecture
۷	۵.۱۱.۳ بررسی و ارزیابی معماری

۶.۱۱.۳	پیاده‌سازی مبتنی بر معماری	۷
۷.۱۱.۳	اطمینان از انطباق نسبت به یک معماری	۷
۱۲.۳	چه چیزی یک معماری خوب را تحویل می‌دهد؟	۷
۱۳.۳	تفاوت Fault با Failure	۸
۱۴.۳	منابع همگن و ناهمگن	۹

مجوز

به فایل license همراه این برگه توجه کنید. این برگه تحت مجوز GPLv۳ منتشر شده است که اجازه نشر و استفاده (کد و خروجی/pdf) را رایگان می‌دهد.

۱ پیشگفتار

اگر درس مهندسی نیازمندی‌ها را خوانده باشید، احتمالاً در جریان آن هستید که برای تولید نرم‌افزار بخش‌های زیادی درگیر هستند اما در حالت کلی در درس پیشین دانستیم که در ابتدا بایستی نیازمندی‌های مشتری یا کارفرما را از محصول نرم‌افزار بدانیم، آن را بررسی و تحلیل کنیم، سند نیازمندی آن را آماده‌سازی کنیم و سپس به دنبال طراحی معماری آن برویم. در این درس به طراحی و پیاده‌سازی سند معماری مقیاس بزرگ یک محصول نرم‌افزاری می‌پردازیم تا فرایند تولید نرم‌افزار را به طور کامل طی کرده باشیم.

۲ معرفی

۱.۲ چه زمانی یک پروژه مقیاس بزرگ است؟

برای اینکه بتوانیم بگوییم که چه پروژه‌ای مقیاس بزرگ محسوب می‌شود، براساس دو استاندارد ایرانی و بین‌المللی می‌توان دو استاندارد را در اینجا مطرح کرد:

- استاندارد مقیاس بزرگ بودن پروژه از نظر دکتر شمس، آن است که پروژه بیشتر از ۶ ماه زمان پیاده‌سازی نیاز داشته باشد و تعداد درخواست‌های ارسالی به آن ۱۲ نفر به بالا باشد.
- استاندارد بین‌المللی مقیاس بزرگ بودن پروژه را زمان یک سال به بالا جهت پیاده‌سازی و تعداد درخواست‌ها را بین ۲۰ تا ۲۲ نفر تعیین می‌کند.

ابتدایی ترین فاز معماری یک محصول نرم‌افزاری مقیاس بزرگ، طراحی و بررسی و آنالیز سناریوهای آن است. سند معماری نرم‌افزار به مجموعه‌ای از سناریوهایی گفته می‌شود که در ازای هر کدام یک راه‌حل مناسب مطرح می‌شود. یکی از نیازمندی‌های بررسی معماری نرم‌افزار مقیاس بزرگ استفاده از متدولوژی RUP^۱ می‌باشد. دلیل اصلی آن این است که می‌توان تمام فرایندهای آن را به همراه Artifactها شخصی‌سازی کرد. در معماری نرم‌افزار می‌توانیم مشخص کنیم که چه اجزایی داریم و این اجزا چگونه با یکدیگر در ارتباط هستند و شامل چه قیدهایی می‌شود. در حقیقت در سند معماری نرم‌افزار نمود خارجی المان‌ها را مطرح می‌کنیم. نحوه در کنار هم چیدن سرویس‌ها را مطرح می‌کنیم اما هیچ وقت در مورد جزئیات اینکه برای مثال از چه الگوریتم‌هایی استفاده می‌کنیم، صحبت نمی‌شود. در این سند علاوه بر نیازهای جاری، در مورد نیازهای آتی نیز صحبت می‌شود که در آینده چقدر باید نرم‌افزار قابلیت گسترش Expandability داشته باشد.

^۱Rational Unified Process

۲.۲ یادآوری متدولوژی RUP

این متدولوژی به عنوان یک متدولوژی توسعه نرم افزار اجایل، به دلیل قابل تکرار بودنش در نظر گرفته شده است. این روش مهندسی نرم افزار از یک سیستم انعطاف پذیر و سازگار در فرایند توسعه نرم افزار استفاده می کند که در برگیرنده انجام تنظیمات و تکرار دوره های مهندسی نرم افزار است تا زمانی که محصول به نیازمندی های مطرح شده و اهداف برسد [۲].

۱.۲.۲ منظور از نظم در RUP

منظور از نظم در حقیقت نمودهایی می باشد که در فرایند توسعه نرم افزار مورد استفاده قرار می گیرد، در حقیقت نظم، مدل سازی حرفه ای را نشان می دهد. این نظم ها به ما کمک می کنند که چه زمانی چه Activity هایی را باید به چه میزان در چه بازه هایی انجام دهیم و خروجی مورد نظر ما چیست؟

برای مثال در فرایند تحلیل نیازمندی پروژه، نظم نیازمندی، خروجی فازهای آن است که به شکل مدل های Usecase diagram و سند معماری نرم افزار کشیده و نوشته شده است.

۲.۲.۲ چهار فاز اصلی در RUP

۱. فاز آغاز یا Inception: در این فاز تمام نیازمندی ها جمع آوری می شود و مقیاس پروژه در آن بدست می آید.

۲. فاز توسعه یا Elaboration: طراحی سیستم و تحلیل دقیق تر نیازمندی ها صورت می گیرد.

(آ) استفاده از مدل سازی ها و کشیدن دیاگرام ها

(ب) کشیدن مدل usecase: کاربرد بزرگی برای مشتری (کارفرما) و طراح دارد و برای هر دو طرف قابل فهم می باشد. در این نوع نمودار افعال و نیازمندی های functional مطرح می شود. انتظارات در مورد سیستم در اینجا مورد بحث قرار می گیرند.

- اینکه کاربرد بتواند زیر ۲ ثانیه احراز هویت شود مربوط به نیازمندی های non-functional می باشد.
- شامل دو سند می شود:

– سند Usecase که انتظارات سیستم را مشخص می کند.

– سند معماری که function و non-functional را در بر می گیرد.

(ج) طراحی Class diagram

(د) طراحی Sequence diagram

۳. فاز ساخت یا Construction: در این فاز کد نویسی و ارزیابی کدهای نوشته شده صورت می گیرد.

۴. فاز استقرار یا Deployment: در این فاز نرم افزار آماده شده است و در بستری مناسب به کاربران نهایی^۲ ارائه می شود که نیازمند آموزش های لازم می باشد.

محبوبیت استفاده از متدولوژی RUP به خاطر آن است که کاملاً به صورت جامع سیستم را در بر می گیرد.

۳.۲.۲ منظور از فرسخ شمار چیست؟

فرسخ شمار یا Milestone در هر کدام از فازها مشخص می شود که در حقیقت در مورد تعیین یک بازه زمانی مشخص صحبت می کند. در آن می توانیم ببینیم که در فازهای قبلی چه کارهایی بایستی انجام می شده، آیا آن ها را انجام داده ایم و اگر انجام نداده ایم یا مشکلی در آن وجود دارد آن فاز را تکرار می کنیم تا به انتهای آن برسیم که به نحوی تسک یا وظیفه را ببندیم.

^۲ End users

نکات

- ساده‌ترین سند در میان این ۴ فاز، سند استقرار می‌باشد.
- معماری مقیاس‌پذیر (بزرگ) یک پروژه نرم‌افزار دو بُعد پویا و ثابت دارد.
- در مورد ارزیابی کارایی و آزمون نرم‌افزار گفتنی است که هر توسعه‌دهنده مسئول Quality control بخش خودش است.
- تکرارها n تا هستند مدیر پروژه یا طراح سیستم باید به ما تعداد تکرارها را به صورت تقریبی بگوید.

۴.۲.۲ محوریت بر روی نیازمندی‌ها

متدولوژی RUP تاکید زیادی روی شناسایی و مدیریت نیازمندی‌ها را دارد و به تیم‌ها کمک می‌کند تا نیازمندی‌های کلیدی پروژه را به خوبی درک و پیاده‌سازی کنند.

۵.۲.۲ استفاده از برنامه‌نویسی OOP

این متدولوژی به طور گسترده از ۴ اصل شی‌ءگرایی استفاده می‌کند و به توسعه‌دهندگان اجازه می‌دهد که کدهای قابل استفاده مجدد و مدیریت فاکتورهای انعطاف پذیری را ایجاد کنند.

۳ معماری مقیاس بزرگ نرم‌افزار

۱.۳ معماری نرم‌افزار چیست؟

معماری نرم‌افزار یک تعریف واحد ندارد. معماری نرم‌افزار یک برنامه یا یک سیستم محاسباتی می‌باشد. یک ساختار یا مجموعه ساختارهایی است از سیستم مورد نظر ما که متشکل از المان‌های کامپیوتری است و نمود خارجی یک چیز (المان) می‌باشد و ارتباطات بین آن‌ها را در بر می‌گیرد. هیچ‌گاه نمی‌توان نرم‌افزاری نوشت که معماری نرم‌افزار نداشته باشد. برای مثال از معماری MVC در نرم‌افزار خود استفاده کرده‌ایم. نرم‌افزاری وجود ندارد که معماری نداشته باشد. اگر بگوییم نرم‌افزاری معماری ندارد در حقیقت علم معماری به کار گرفته شده را نمی‌دانیم که آن را بی‌معماری می‌نامیم. برای مثال معماری کلاینت سرور که براساس نیازمندی‌های نرم‌افزاری بیان می‌شود که چه بخش‌هایی سمت سرور باشد چه بخش‌هایی سمت کلاینت.

۲.۳ معمار نرم‌افزار کیست؟

معمار نرم‌افزار شخصی مدبر است که تجربه تخصصی آن در حوزه‌ای مشخص بیشتر از ۱۰ سال است که تسلط کافی در آن سیستم مشخص دارد و از ابتدا تا انتهای پروژه با فرایند توسعه و توسعه‌دهندگان همراه است.

۳.۳ المان‌ها

بخش‌های یک سیستم نرم‌افزاری را گویند برای مثال یک نرم‌افزار واحد مانیتورینگ، واحد زمان‌بندی، واحد بررسی درخواست‌ها و غیره را دارد.

۴.۳ External Feasible Properties

آن بخش چه وظیفه‌ای را باید انجام دهد و آن بخش آن وظیفه را در حال انجام است یا خیر؟ جزئیات مربوط به المان‌های درگیر در بخش معماری در External Feasible Properties مطرح نمی‌شود.

۵.۳ تفاوت کامپوننت و المان

وقتی در مورد کامپوننت می‌گوییم در حقیقت چیزی است که می‌خواهیم آن را پیاده‌سازی کنیم. المان کامپوننتی است که قسمت اجرایی را برای آن در نظر نگرفته‌ایم.

۶.۳ بخش‌هایی که معماری نرم‌افزار باید پوشش دهد

۱. المان‌هایی که در سیستم قرار است استفاده شود را مشخص می‌کند.
 ۲. نمود خارجی المان‌های سیستم مورد نظر را مشخص می‌کند و تعیین می‌کند هر المانی در سیستم چه وظیفه‌ای را انجام دهد.
 ۳. ارتباطات بین المان‌ها را به روشنی مشخص می‌کند.
- در کنار تمامی موارد بالا، بخش‌هایی که یک معماری نرم‌افزار پوشش نمی‌دهد شامل ذات المان‌ها می‌باشد.

۷.۳ قابلیت اطمینان یا Reliability

یک سیستمی که در زمان مشخص درست کار کند به شرطی که در زمان $T - x$ درست کار کرده باشد.

۸.۳ قابلیت استفاده یا Useability

سیستمی که کارآمد باشد برای آن دسته از افرادی که سیستم را حاضر و آماده کرده‌ایم. به گونه‌ای که با ظاهر مناسب کار کردن با آن نیز آسان باشد.

۹.۳ ارائه سریع محصول یا Short time to market

قابلیت یا Feature مجموعه‌ای از توابع نرم‌افزاری است که وقتی در بازار ارائه می‌شود، واقعاً کار می‌کند.

نکات

- تا آنجایی که می‌شود هزینه‌ها را باید کاهش بدهیم و بیشترین هزینه‌ها را ما در بخش توسعه نرم‌افزار خواهیم داشت. همیشه باید کارمندان را مشغول توسعه نگهداریم تا باعث از دست رفتن هزینه‌ها نشود.
- مشکل معمار آن است که حجم زیادی از نیازمندی‌های نرم‌افزاری را در حال بررسی است که نسبت به هم در تضاد هستند.
- بخشی از وظایف اصلی معماری نرم‌افزار مشخص کردن استک‌های نرم‌افزاری می‌باشد. بخش دیگری از آن این است که بررسی کند این موارد توسط تیم اجرا و استفاده می‌شود یا خیر (Follow up)

۱۰.۳ تفاوت معماری سازمانی و معماری نرم‌افزار

معماری سازمانی و معماری نرم‌افزار با یکدیگر متفاوت است. در معماری سازمانی، چارت سازمانی آن مشخص‌کننده محدوده و کلیت هر قسمت آن سازمان می‌باشد.

۱۱.۳ جزئیات فعالیت‌های معماری نرم‌افزار

۱.۱۱.۳ ساخت Business case برای سیستم

یادآوری Business plan

بیزینس پلن سندی است که چهارچوب سیستم ما را در بر می‌گیرد و آن را نسبت به سیستم مشابه مقایسه می‌کند و آن را به چالش می‌کشد. نسبت به هر چالشی که در سیستم ما وجود دارد بایستی پاسخی مطرح شده باشد. سند Business case سندی از جنس مالی است که در آن برآورد هزینه و توجیهات اقتصادی بیان شده است. در این سند، درگیر بودن نیروها بررسی می‌شود و در نهایت توجیهات اقتصادی را از نظر کاهش هزینه‌ها مطرح می‌کند. برای مثال ممکن است بیشتر اوقات بهره‌وری سیستم را افزایش داده باشیم که نسبت به این افزایش باید توجیهی وجود داشته باشد. از ابتدا تا انتهای پروژه دائماً در حال تخمین قیمت پروژه به عنوان معمار نرم‌افزار هستیم که در آن هزینه‌ای را مطرح می‌کنیم که بررسی کرده‌ایم در توسعه نرم‌افزار نیاز خواهد شد. در این حین اگر هزینه توسعه بیشتر شود باعث ضرر ما و اگر کمتر شود باعث سود ما خواهد شد. پس به همین دلیل سعی می‌کنیم سند Business case را از ابتدا تا انتهای فرایند پروژه توسعه دهیم و توجیه اقتصادی به روزی در آن داشته باشیم که موجب ضرر از سمت پیمانکار نشود.

نقطه سر به سر یا Break event point

وضعیتی است که در آن هزینه‌هایی که ما برآورد کرده‌ایم در بازه زمانی مشخص، برابر با صفر شده است. برای مثال اگر مبلغ ۵۰۰ میلیون تومان را به عنوان هزینه نرم‌افزار محاسبه کرده باشیم و طی یک سال دیگر سازمان دقیقاً همان مبلغ را بدون کاهش یا افزایش پرداخت می‌کند این وضعیت نقطه سر به سر خواهد شد. زمان نقطه سر به سر معمولاً در بازه ۲ تا ۵ سال تعریف می‌شود که در واقعیت زمان خیلی طولانی برای تخمین هزینه توسعه نرم‌افزار محسوب می‌شود.

۲.۱۱.۳ فهمیدن نیازمندی‌های پروژه

نیازمندی‌ها تنها متغیر ثابت هستند. همانطور که پیش‌تر اشاره شد non-functional ها در قالب سند معماری نرم‌افزار دیده می‌شوند. همچنین اگر نیازمندی functional وجود نداشته باشد non-function ها معنایی ندارند. برای مثال می‌گوییم که نرم‌افزارمان بایستی امن باشد، یعنی Usecase diagram داریم که ازای آن نیازمندی non-function تعریف کرده‌ایم. نکته مهم آن است که مهمار نرم‌افزار تنها نیازمندی‌های جاری را در نظر نمی‌گیرد بلکه نیازمندی‌های آتی را هم در سند پیشبینی می‌کند تا بتواند یک جریان را کامل کند و به نوعی Proof of concept داشته باشد.

۳.۱۱.۳ ایجاد یا انتخاب یک معماری نرم‌افزار

معمار نرم‌افزار معمولاً یا یک معماری را انتخاب می‌کند یا آن را از نو می‌سازد. ممکن است در ترزهای آکادمیک معماری جدیدی را ایجاد کرده باشیم زیرا ممکن است صورت مسئله‌های جدیدی پدید آمده باشند یا نیازمندی‌های non-function جدیدی یافت شده باشد. در حالت کلی معماری‌ها را یا باید بهینه‌سازی کنیم یا اجرا و در پروژه نرم‌افزاری پیاده کنیم.

۴.۱۱.۳ دنبال کردن معماری یا Communicating the architecture

چیزی که در معماری نرم‌افزار بیان شده است را دنبال کنیم و بررسی کنیم که تمام آن فرایندهای مهندسی توسعه نرم‌افزار را توسعه‌دهنده‌ای دنبال می‌کند یا خیر. بیان این مسئله طرف معمار و اجرای آن در طرفی دیگر است. خیلی با بخش Implementing based on the architecture ارتباط دارد.

۵.۱۱.۳ بررسی و ارزیابی معماری

نکته مهم آن است که در حوزه معماری نرم افزار چیزی به نام شبیه سازی نداریم چرا که قدم ابتدایی هر پروژه ای تعیین معماری آن است. اینکه از ما سوال شود که معماری نرم افزارتان را با چه شبیه سازی اجرا کردین یا با چه فاکتورهای شبیه سازی آن را ارزیابی کرده اید، بحث کاملاً غلطی می باشد. مقایسه بین الگوریتم ها امکان پذیر می باشد زیرا داریم شبیه سازی ورکلودها را انجام می دهیم. در این خصوص روشی را داریم به نام ATAM که ارزیابی معماری نرم افزار را می توان از طریق آن انجام داد. یک استاندارد مشابه با ISO می ماند که نیازمندی های non-functional را در نظر می گیرد. یک تیم مستقل با آن همراه است که بررسی کند ما از راه درستی استفاده کرده ایم یا خیر. در نهایت به ما نشان می دهد که نسبت به آن سناریو می توانیم این معماری را پوشش دهیم یا خیر. برای بررسی و ارزیابی معماری راه های مختلفی وجود دارد:

۱. مدل سازی صوری، فرمال و رسمی: برای اینکه ثابت کنیم چیزی که داریم می نویسم قابل تایید است و درست می باشد. به دلیل آن که ریاضی هستند می توانند مفید و رویکردی مناسب باشند.

۲. استفاده از ADL: Architecture Definition Language

۳. استفاده از Prototype ها: اگر یک سیستم کوچک درست کنیم که به صورت صحیح کار کند می تواند نشان دهنده آن باشد که سیستم اگر بزرگتر شود هم صحیح کار خواهد کرد.

۴. ۸۰ درصد نیازمندی ها و دامنه های مسئله با نرم افزاری که در حال کار است که با این معماری انطباق دارد پس قطعاً معماری با کارهای آینده ما نیز منطبق خواهد بود.

نکته

دامنه مسئله مثل دامنه سیستم های مالی، دامنه سیستم های آموزشی و غیره یک تعریف مسئله مخصوص و واحد دارد که در سیستم های گوناگون مشابه یکدیگر هستند و دقیقاً نیازمندی های آن ها نیز مشابه هستند. اگر یک سیستم مشابه را پیدا کردیم، معماری انتخابی ما می تواند با سیستم مشابه نیز کار کند.

۶.۱۱.۳ پیاده سازی مبتنی بر معماری

بررسی اینکه آیا تمام فرایندهایی که در فازهای توسعه نرم افزار شروع می شود با بیانات و نقشه راه معمار نرم افزار مطابقت دارد یا خیر؟

۷.۱۱.۳ اطمینان از انطباق نسبت به یک معماری

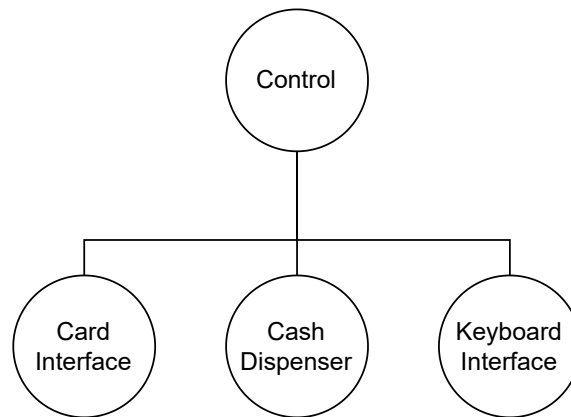
در این بخش بررسی می کنیم که فرایندها تماماً تابعی از معماری هستند یا خیر؟ در انتهای کار، تمام اسناد را در کنار هم قرار می دهیم و به تطبیق اسناد با سند معماری می پردازیم. برای مثال تمام Usecase diagram ها مطابق با سند معماری Usecase ها بوده اند؟ طبق استاندارد طراحی شده اند؟ اگر هر کدام مطابقت نداشت بایستی سریعاً اصلاح شود تا از ایجاد هزینه های آینده جلوگیری به عمل آورد. ما بایستی مطمئن باشیم که تمام اسنادی که به معمار نرم افزار تحویل می دهیم مطابق با سند معماری باشد که شامل چندین بخش خواهد شد.

۱۲.۳ چه چیزی یک معماری خوب را تحویل می دهد؟

هیچ چیز ذاتاً خوب یا بد نیست بلکه وابستگی بسیار زیادی به کاربرد آن در سیستم مورد نظر دارد. معماری کنترل دمای یک نیروگاه تولید برق را نمی توان در کنترل دمای خانه استفاده کرد چون استفاده نادرست و نا به جایی بوده است. ذاتاً نمی توانیم بگوییم که کدام معماری خوب است کدام معماری بد بلکه انتخاب ما نسبت به سیستم خوب و بد دارد.

معماری قابل ارزیابی است پس باید براساس اهدافی که داریم فرایند ارزیابی را انجام دهیم تا در نهایت ببینیم که این ارزیابی چقدر می تواند معماری را با اهداف ما Match کند. برای مثال ممکن نیست که یک سامانه ای که امنیت ندارد را الهام بگیریم برای سامانه ای که یکی از اهداف اصلی آن امنیت است.

نمودار زیر را از معماری سطح بالای ATM را در نظر بگیرید:



شکل ۱: نمایش معماری سطح بالا دستگاه ATM

شکل شماره ۱ ساختار کلی از دستگاه ATM را نمایش می‌دهد. درست است که در مطالب بالاتر گفتیم که معماری یک چکیده از ساختار سیستم می‌باشد اما در این نمودار ارتباطات بین المان‌ها کاملاً همراه با ابهام می‌باشد و تمام معماری دستگاه ATM را پوشش نمی‌دهد. در این نمودار ذات المان‌ها مشخص نیست. برای مثال مشخص نیست که واحد Card interface ممکن است هم کارت را دریافت کند هم اعتبار کارت ورودی را بسنجد؟ پس می‌توان گفت وظیفه المان Card interface در این نمودار اصلاً مشخص نیست و در این نمودار دقیقاً ماهیت ارتباطات بین المان‌ها مشخص نیست. همچنین لایه‌بندی در این نمودار به صورت واضح کشیده نشده است. در لایه‌بندی همواره منطق وجود دارد مانند لایه‌بندی استاندارد OSI که جانمایی المان‌ها در این نمودار کامل کشیده نشده است. برای مثال بحث امنیت و کارایی و تعامل‌پذیری اصلاً وجود ندارد. لایه‌بندی یک معماری تماماً توسط معمار نرم‌افزار مشخص خواهد شد.

۱۳.۳ تفاوت Fault با Failure

Fault یعنی یک نقضی بالقوه در سیستم وجود دارد تا زمانی که سیستم بدون مشکل کار کند آن نقض خودش را نمایان نمی‌کند اما در شرایط خاصی ممکن است برنامه به این Falut برخورد کند و بالفعل موجب از کار افتادن نرم‌افزار شود. در حقیقت بعد از برخورد نرم‌افزار با Fault پدیده‌ای به نام Failure رخ می‌دهد. در حقیقت Fault در نرم‌افزار وجود داشته است که در شرایط خاص با ورودی خاص کاربر برنامه با شکست یا Failure رو به رو می‌شود و باعث کار نکردن درست نرم‌افزار خواهد شد.

همواره Fault از سمت طراحی نرم‌افزار همراه خواهد بود زیرا بایستی در اسناد طراحی به آن نگاه مهندسی شود. زمانی که پیاده‌سازی می‌شود اگر در شرایط آزمون بتوانیم آن Fault را شناسایی کنیم و آن‌ها را رفع کنیم سیستم را از Failureهای آینده نجات خواهیم داد. نکته: Fault موجب Failure می‌شود.

اگر ما یک حافظه USB داشته باشیم و در هنگام انتقال اطلاعات ناگهانی فلش را از سیستم خارج کنیم، در حقیقت نه Fault نه Failure و نه Error رخ داده است. بلکه سیستم توسط عامل خارجی دستکاری شده است. اگر Fault در سیستم داشته باشیم بایستی توانایی هندل کردن آن را در نرم‌افزار داشته باشیم.

نکات

- معماری نرم‌افزار المان‌های نرم‌افزاری را مشخص می‌کند.
- Mapping تسک‌ها به منبع مشخص تعریف زمانبندی است.
- تسک‌ها در حقیقت کارهایی هستند که در سیستم تعریف می‌شوند.

- ورکفلو مجموعه‌ای از تسک‌های وابسته به هم می‌باشد.
- Bag of tasks عموماً وابستگی ندارند.
- Multiple workflow scheduling به معنای زمانبندی چند ورکفلو می‌باشد.
- همیشه یک زمانبندی بهینه نداریم بلکه باید در شرایط مناسب از زمانبندی مناسبی استفاده کنیم.
- در هنگام مهندسی نرم‌افزار باید تا آنجایی که می‌شود نرخ موفقیت یک تسک را بالا ببریم.

۱۴.۳ منابع همگن و ناهمگن

- منابع همگن به منابعی گفته می‌شود که همه المان‌ها در آن دقیقاً یک چیز هستند یا به عبارتی همه منابع به صورت Clone از یکدیگر هستند. برای مثال همه از یک سخت‌افزار استفاده می‌کنیم بدون هیچ تفاوتی.
- منابع ناهمگن به متفاوت بودن سیستم‌ها نسبت به یکدیگر اشاره دارد.