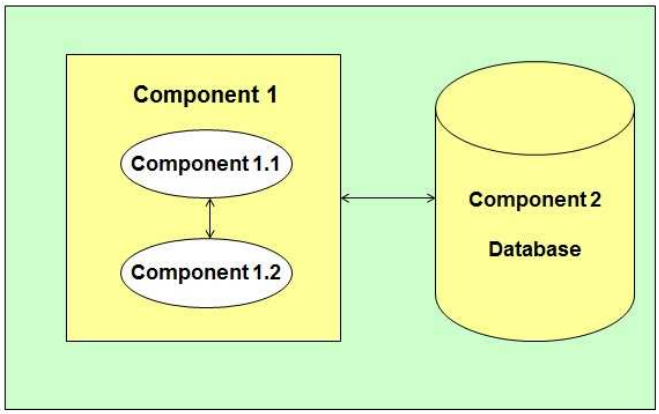
فصل ۱

در این اسلاید یک تعریف غیر رسمی از SA یا Software architect (مهندسی نرم افزار) ارائه شده است.

- اولین مرحله از توسعه راه حل است. یعنی همیشه اینطوری که یک سیستم نرم افزاری که می خواهیم طراحی کنیم همانند یک مسئله باید به آن نگاه کرد و راه حلی که برای آن ارائه می شود همان معماری آن است.
- یک ساختار کلی یا سطح بالایی از سیستم نرم افزاری است و دید سطح بالایی است که از سیستم نرم افزاری داریم
- معماری نرم افزار چیزی به جز معرفی یکسری component یا مولفه و connector های بین آنها نیست

اما این که این component ها و connectorها چه هستند در اسلاید بعدی مشخص است.

A simple example



این اسلاید یک مثال ساده است که فرض کنید شما یک سیستم اطلاعاتی خیلی ساده نوشتید که یک فرم ورود اطلاعات دارد و یک فرم خروج اطلاعات داریم که این فرم ورورد اطلاعات را مثلا به اسم component1.1 نوشته و فرم خروج اطلاعات را به اسم component 1.2. که ارتباط بین اینها باید معلوم شود که در معماری به چه صورت باید باشد مثلا component1.1 ارتباط اش اینجوری که با component2 اطلاعات وارد این component می کند و ذخیره می کند و component1.2 از آن اطلاعات می خواند.

در این اسلاید داریم که چرا معماری نرم افزار باید داشته باشیم :

- ❖ معماری نرم افزار برای فائق آمدن بر پیچدگی ها ارائه می شود. با استفاده از روش تقسیم و غلبه (Divide and Conquer) و تقسیم بندی وظایف از دو دید:
- ❖ از دید process یا فرآیند: یعنی ما فرآیند طراحی مان را به دو فاز تقسیم می کنیم:
  ۱. طراحی معماری
  ۲. طراحی جزئیات
- ❖ از دید product یا تولید: یعنی با استفاده از معماری نرم افزار یک سیستم مبتنی بر مولفه طراحی کنیم یعنی سیستم ما خروجی اش می شود یک مجموعه ای مولفه ها

  اینجوری مزیتی که دارد این است که هم پیاده سازی آن ساده است که در بحث process داریم چون هر مولفه ای که تعریف می شود در معماری یک تیم برنامه نویسی خاصی می توانیم برای طراحی آن بگذاریم و اینجوری کنترل اینکه هرکسی چه کاری انجام داده راحت تر است و خطایابی های آن آسان تر می شود. یعنی دقیقا همان بحث هایی که در برنامه نویسی ساخت یافته داریم که می گوییم چرا برنامه نویسی ساخت یافته خوب است این جا هم وجود دارد و از آن طرف توی محصولش هم شما مولفه وقتی تولید می کنید این مولفه usable است یعنی قابل استفاده مجدد است و می توانید در سیستم های بعدی استفاده کنید می توانید به صورت مجزا به فروش برسانید یا می توانید با استفاده از معماری یک خط تولید راه بیاندازید در واقع خط تولید را انداختن یعنی این که شما می توانید دقیقا خط تولید مولفه های آماده از دیگران یا از خود را به یکدیگر متصل کنیم تا یک سیستم جدید تولید کنیم.
- ❖ مهمترین دلیل دیگری که ما معماری نرم افزار را می خوانیم اطمینان (Assuring) از این که صفت های کیفی مورد نیاز در سیستم از همان ابتدا به طور کامل دیده شده است. nonfunctional requirement یعنی نیازمندی های غیر عملیاتی که یک اسم دیگری که اینجا برای آن بکار می بریم صفات کیفی است که به آن quality attribute ها نیز گفته می شود که شامل کارایی یا performance و modifiability یا

changeability یعنی قابلیت تغییر و security یا امنیت و testability یا قابلیت تست و usability یا قابلیت استفاده و غیره) و ما با استفاده از معماری است که می توانیم این صفات کیفی را ببینیم از اول پروسه تولید نرم افزار تا انتهای آن و حتی در فاز نگهداری

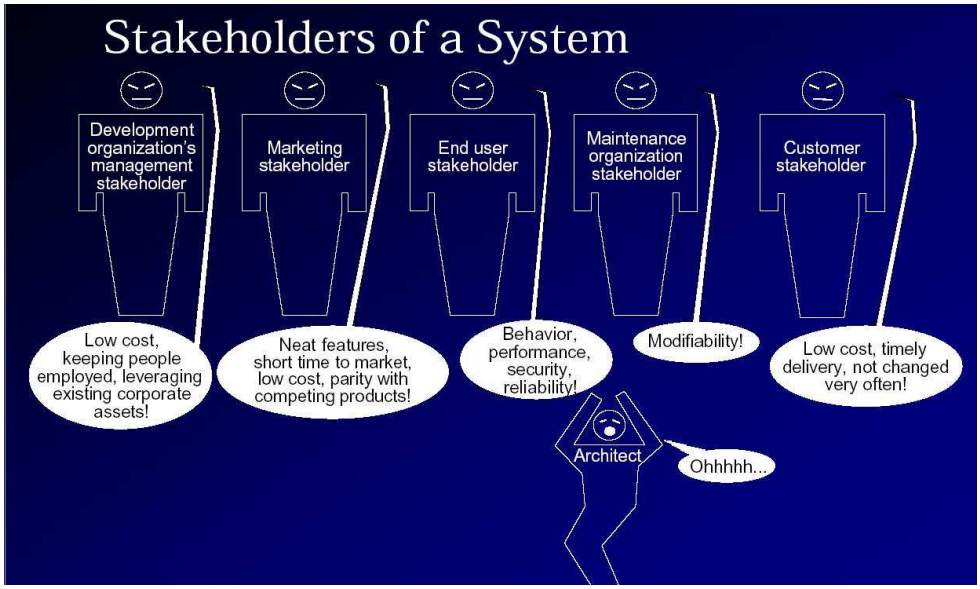
در این اسلاید ریشه های معماری نرم افزار بیان شده است:

- معماری نرم افزار شبیه معماری ساختمان است
- این ایده جدیدی نیست و از دهه شصت بوده و از دهه هفتاد به بعد این خیلی قوی تر شده است.
- حدود ده سال هست (البته چون کتاب مال سال ۲۰۰۳ است پس الان حدود بیست سال هست) که این دیگر خیلی توجه به آن شده است.

چرخه کسب و کار معماری:

منظورش این است که معماری روی چه چیزهایی و چه کسانی اثر میگذارد و برعکس چه کسانی و چه چیزهایی روی معماری اثر میگذارند.

چه چیزهایی روی معماری اثر میگذارند؟



۱-Stakeholder

۲-سازمان توسعه نرم افزار

۳-محیط های تکنیکی

۴- تجربه معمار

حال به تعریف هر یک از موارد فوق میپردازیم:

۱-Stakeholder:

گروه ذیعنفان –افرادی هستند که در تولید پروژه نرم افزار ها با آنها سر و کار داریم مثلا کارفرم.تمامی اعضاء تیم توسعه نرم افزار ، مدیران پروژه ، کارمندان آنها، کابر پایانی، مشتری، تسترها، نگهدارنده سیستم ، بازار و..

پس ۵ نوع استک هوادر داریم:

۱-استک هولدر مدیرتیم پروژه نرم افزاری

۲- استک هولدر بازار

۳- استک هولدر کاربر پایانی

۴- استک هولدر سازمان نگهداری و پشتیبانی

۵- استک هولدر مشتری

حال میخواهیم بدانیم هر یک از این استک هولدر روی معماری چه اثری میگذارند:

۱-استک هولدر مدیرتیم پروژه نرم افزاری

میگوید هزینه میخواهم پایین باشد ، از همین کارمند ها موجود که داریم با همین تبحرها و توانایی ها ی که دارند میخواهیم استفاده کنیم

۲- استک هولدر بازار

میگوید محصولی که تولید میشود باید بدوم ابهام و وخوش فهم باشد،زمانتحویل به بازار کوتاه باشد، هزینه ها پایین باشد ، با محصوالات رقابتی یک نواختی داشته باشد(متفاوت نباشد).

۳- استک هولدر کاربر پایانی

محصول خوش رفتار باشد یعنی به راحتی بتوان از آن استفاده کرد. ، محصول کارایی داشته باشد ، محصول جواب درست بدهد و بشود به پاسخ های سیستم اعتماد کرد، امنیت داشته باشد و اطلاعات را افشاء نکند

۴- استک هولدر سازمان نگهداری و پشتیبانی

بعد از استقرار سیستم ، پشتیبان وارد کار میشود و میخواهد که اگر ایرادی یا مشکلی بعد از مدت طولانی پیدا شود بتواند رفع کند ، و اگر کاربر نیاز جدیدی داشته باشد بتوان نیار کاربر را رفع کند.

۵- استک هولدر مشتری

اگر سیستم را برای یک نفر طراحی کنیم آنرا کارفرما است ولی اگر سیستم را طوری طراحی شود که به بقیه هم بتوان فروخت اصطلاحا افرادی بعدب مشتری میشوند.

مشتری هزینه پایین میخواهد، میخواهد به موقع سیستم تحویلش داده شود ، تغییرات زیادی سیستم نیاز نداشته باشد.

۲-سازمان توسعه نرم افزار :

نگرانی سازمان توسعه نرم افزار از دو دید است

۱-business Issues(شرایط کسب و کار):

-از نظر کسب و کار اول می آییم سرمایه گذاری میکنیم و بعدش زیر ساختارها را بررسی میکنیم : یعنی اول ببینیم که بازارش برای فروش وجود دارد امکاناتش را داریم یعنی اول امکان سنجی میکنیم و براورد میکنیم که آیا مقرون به صرفه است یا نه

-هزینه ها پایین باشد: برای اینکار باید از تخصص های موجود استفاده بکنیم

-سادگی پیاده سازی:بتواند ماژول ها یی که معماری معرفی میکند را بتواند به سادگی پیاده سازی نماید

۲-Organizational Issues(شرایط سازمانی):

-از ساختار های سازمانی جاری استفاده بکنیم

-از همین پرسنل که وجود دارند بهره برداری کنیم

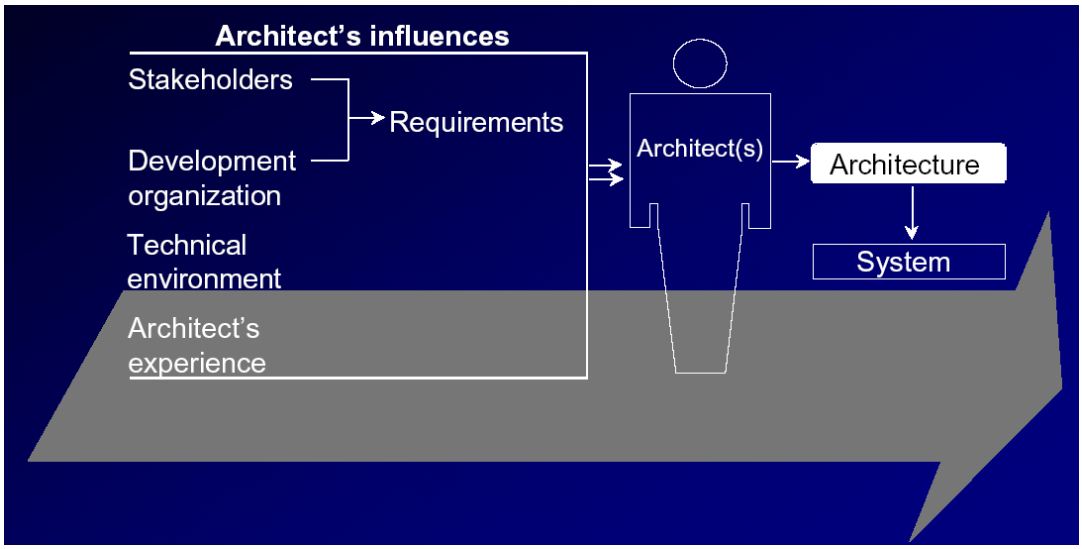
۳-محیط های تکنیکی:

-استفاده از سیستم های Middleware مثلا امروزه میخواهیم سیستم ها را وب بیس باشند بنابراین مجبوریم از یک سری سیستم های Middleware مثل جاوا و دات نت و.. استفاده کنیم.

-استفاده از تکنولوژی موجود : مثلا تکنولوژی های که در دسترس هستند باید از ابتدا مشخص شود سیستم می خواهد متمرکز باشد ی غیر متمرکز که همه اینها بستگی به هزینه Processor و هزینه ارتباطی و چیز های شبیه اینها دارد

۴-تجربه معمار :

ایا این معمار قبلا معماری طراحی کرده و به نتایج مثبت رسیده است یا نه اگر نتایج منفی رسیده باشد باید در طراحی های جدیدش از آن نتایج منفی پرهیز کند.



معماری روی چه چیز هایی اثر میگذارد؟

۱-سازمان توسعه نرم افزار

۲-نیازمندیهای مشتری

۳-تجربه معمار

۴- محیط های تکنیکی

۱-سازمان توسعه نرم افزار :

-ساختار سازمانی و منابع سازمانی تحت تاثیر قرار میگیرند . واحد های کاری که داریم در اطراف واحد های معماری سازماندهی میشوند یعنی اینکه معماری است که میگوید چه واحد کاری روی چه ماژولی روی چه مولفه های کار بکند و چه کاری انجام بدهد زمانبندی را معماری مشخص میکند بودجه را همینطور.

-از دید اهداف سازمانی ، معماری مشخص میکند خبرگی در ساختن یک نوع سیستم را

-موفقیت در بازار حاصل یک معماری خوب است

-ارزیابی یک بازار حاصل یک معماری خوب است

-معماری خودش سرمایه های خط تولید است

۲--نیازمندیهای مشتری:

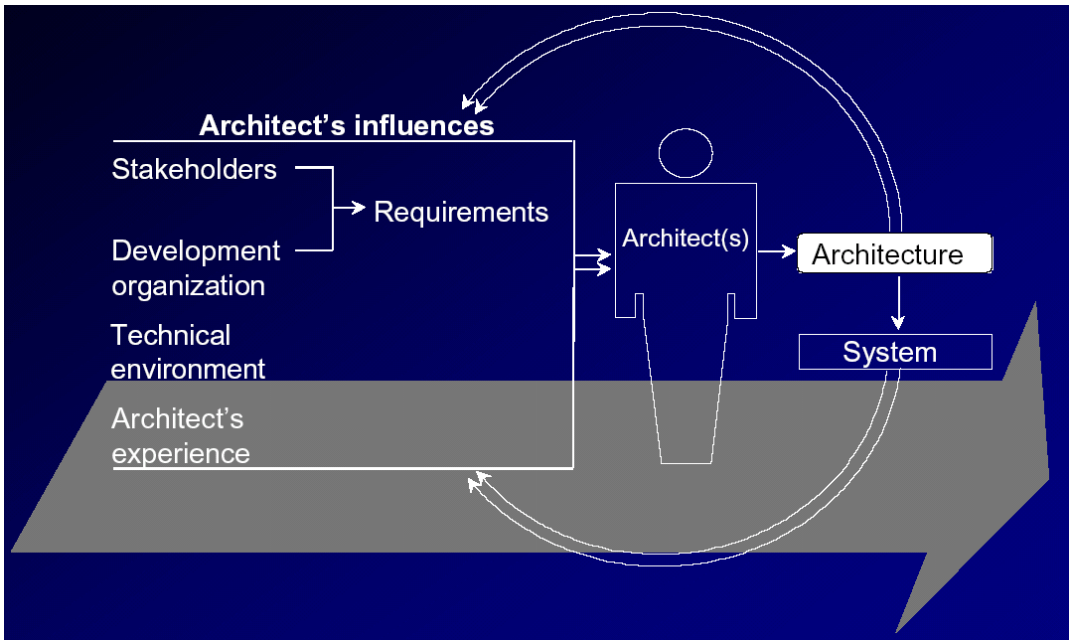
معماری یک دانش را برای مشتریها ایجاد میکند تا بتوانند نیاز های خاص را در سیستم های بعدی بخواهند و این معماری است که از آپ گرید کردن و انطباق پذیری پشتیبانی میکند

۳-تجربه معمار:

وقتی که یک تلاش برای ساختن سیستم انجام میشود تجربه معمار را افزایش میدهد .

۴- محیط های تکنیکی:

ممکن اسن بخاطرایجاد یک معماری ، یک محیط تکنیک جدیدی را بوجود آورده باشیم یا از روش های تکنین های جدیدی استفاده کرده باشیم برای اینکه دفعه اول است که این کار را انجام داده ایم و از این به بعد این عرف میشود و بقیه هم میتوانند از آن استفاده بکنند مثلا یک سیستمی طراحی کرده باشیم که بر اثر ، اثر انگشت بتوان تعیین هویت کرد این خودش یک تکنیک جدید است که ایجاد کرده ایم.



فرایندهای نرم افزار و چرخه کسب و کار معماری:

مراحلی که برای تولید یک نرم افزار است با توجه به چرخه کسب و کار معماری چیست در واقع چه فعالیت های باید انجام شود تا یک معماری نرم افزار طرا حی و ایجاد شود ، چه فعالیت هایی شامل میشود و در ایجاد یک معماری نرم افزار که بتوان بعد بر اساس آن معماری نرم افزار ، فاز طراحی را مشخص کنیم و پیاده سازی کنیم و بعد هم مدیریت کنیم.

فعالیت های معماری:

۱-باید یک کیس کسب و کار برای سیستم ایجاد کنیم

۲-باید نیازمندیها یمان را بفهمیم، استخراج کنیم، کشف کنیم

۳-باید یک معماری که از قبل وجود داشته را انتخاب کنیم و یا یک معماری را ایجاد کنیم

۴-باید معماری را مستند کنیم و معماری را ارائه دهیم(باید برای افرادی که باید بدانند توضیح دهیم)

۵-معماری را تحلیل و ارزیابی کنیم

۶-بایدسیستم را بر اساس معماری پیاده سازی کنیم

۷- اطمینان از پیروی از یک معماری

معمار نقشش از اولین لحظه ای که یک مشتری می آید سفارش سیستم میدهد تا آخرین لحظه که پشتیبانی تمام میشود پا بر جاست و همیشه هست و وجود دارد

۱-Creating the Business Case (ایجاد کیس کسب و کار)

این سوالات در کیس کسب و کار پاسخ داده میشود.

-هزینه تولید چقدر است

-بازار های هدف مان چه هست

-زمان تحویل به بازار چقدر است(کوتاه مدت –بلند مدت) محصولی که داریم را میخواهیم سریع به بازار تحویل دهیم بحث رقابت مطرح است اگر محصول در یک بازه زمانی معینی وارد بازار نشود ارزشی برای اریه اش دیگر ندارد میخواهیم سریع جمع اش کنیم و هدف کسب و کار هم کوتاه مدت است یعنی این کسب و کار بازه ای است مثلا موقع انتخاب که میشود بحث پورتال و اس ام اس و.. داغ میشود و کاندید ها سفارش میدهند که چنین نرم افزار هایی را برایشان ساخته شود یا زمان المپیک و ... اینها بازار های کوتاه مدت هستند برای تولید این جور نرم افزار ها باید سریع کار انجام شود.
Long Term Business : منظور این است که این سیستم قرار نیست خیلی سریع آماده شود قرار است اولا عمر طولانی داشته باشد باید ماندگاری داشته باشد پس بنابراین کشف نیازمندیها و پروسه تحویل ، طراحی آن هم خیلی طول میکشد مثلا مثل سیستم مالیات کشور.
** مثال های برای سوالات خیلی مهم است و نمره هم دارد
-آیا سیستمی که طراحی می کنیم نیاز است که با سیستم های دیگر در ارتباط باشد
-آیا سیستم هایی که وجود دارند که سیستم ما میخواهد با آنها کار کند کار با آنها محدودیتی وجود دارد
۲-Understanding the Requirements (درک نیازمندیها)
نیازمندیها در تحلیل Object Oriented به دو صورت نشان میدهیم:۱-سناریو ۲-use Cases
پس نیازمندیها دو دسته هستند :۱- نیازمندیهای عملیاتی(Use Cases) ۲- نیازمندیهای غیر عملیاتی(Senarios)
پس با نوشتن سناریو ها و Use cases ها نیازمندیها را لیست میکنیم.
-با استفاده از Prototype:
Prototype تولید می کنیم یعنی نمونه اولیه بسازیم با ساختن نمونه های اولیه که یک سری مدل های هستند که با استفاده از آنها میتوانیم رفتار سیستم را نشان دهیم که چطوری کار میکند یا یوزر اینترفیس ها را بسازیم ونشان کاربر دهیم که چه کار هایی میکند و تحلیل کنیم منابعی که استفاده می کنیم با استفاده از Prototype میتوانیم به همه اینها برسیم مثلا اولین کاری که میکنیم فرمهایمان را طراحی می کنیم نیازی هم نیست کلید هایش کار کند فقط ظاهر برنامه که این منو ها را دارد و این قسمت ها را دارد مثلا در بحث ساختمان Prototype میشود ماکت ۳ بعدی ساختمان
۳-Communicating the Architecture (ارایه معماری)
باید به تمام کسانی که قرار است از این معماری استفاده بکنند و بر اساس آن کار را جلو ببرند معماری را اریه بدهیم پس برای اینکه معماری بتوانند بعنوان ستون فقرات طراحی پروژه موثر باشد باید به سادگی و بطور واضع و راحت بدون هیچ ابهامی برای تمامی Stakeholder ها شرح داده شود توسعه دهندگان باید کار هایی که انجام میدهند را درک کنند تستر ها باید ساختارو وظایف خودشان را درک کنند مدیران باید درک کنند که زمانبدی هایشان باید دقیقا بر معماری دلالت کند و ....
پس همه استک هلدر ها شریک هستند.
۴-Analyzing or Evaluating the Architecture (تحلیل و ارزیابی معماری)
-در هر پرسه طراحی ما چندین کاندید طراحی داریم که بعضی از آنها را باید سریعا رد کرد و بعضی دیگر را باید بحث کرد که کدام انتخاب شود
پس یکی از چالش ها ی بزرگ این است که انتخاب کنیم بهترین معماری را از بین معماری های که وجود دارد و با هم رقابت دارند
-ارزیابی یک معماری برای اینکه بفهمیم که کیفیت ها را پشتیبانی میکند و برای اینکه بتوانیم اطمینان بکنیم از اینکه نیازمندیهای استک هلدر ها را فراهم میکند ضروری است
۵-Implementing Based on the Architecture (پیاده سازی براساس معماری)
-در این مرحله گفته شده که ما مطمئن بشویم این توسعه دهندگان به ساختار و پروتکل های ارتباطی در نظر گرفته شده در معماری وفادار هستندیعنی دارند بر همان اسا کار انجام میدهند.
برای رسیدن به چنین چیزی باید یک معماری داشته باشیم که سریع و واضع باشد و به خوبی ارتباط برقرار کند و اولین مرحله در این راه اطمینان از درست بودن معماری است
-داشتن یک زیر ساخت با یک محیط فعال , به توسعه دهندگان در ایجاد و ساخت معماری کمک میکند.
اگر بتوان چنین محیطی را ایجاد بکنیم که بتوانیم کنترل کنیم که توسعه دهندگان بر اساس همچنین معماری جلو برود خیلی بهتر است
۶-Ensuring Conformance to an Architecture(اطمینان از انطباق از یک معماری)
وقتی که معماری ساخته و استفاده شد وارد فاز نگهداری میشویم یعنی بعد از اینکه مطمین شدیم که پیاده سازی بر اساس معماری است حالا می خواهیم مطمئن شویم که پشتیبانی هم بر اساس معماری انجام میشود یا نه.
ما به یک هوشیاری دائمی نیاز داریم تا اطمینان بکنیم که معماری و ارائه های که از معماری داشتیم در طول این فاز هم وفادار هستند و همچنین با اینکه کار در این زمینه یک کمی هنوز کامل نیست ولی فعالیت های زیادی در سال های اخیر انجام شده یعنی متد ها و روش هایی که این اطمینان را در فاز نگهداری و پشتیبان به ما بدهند که همچنان معماری ارائه شده با معماری اولیه و واقعی همچنان مطابق با همدیگر جلو میروند و کار میکند
معماری خوب چه هست ؟

-ما وقتی که میخواهیم یک معماری ارزیابی کنیم را باید بگوئیم که چقدر این معماری با نیازمندی و اهداف ما تطبیق دارد و یا اینکه ندارد .یعنی چند درصد از نیازمندیها را پوشش داده و چند درصد را پوشش نداده پس به همین دلیل چیزی بعنوان اینکه یک معمار ذاتا خوب هست یا بد وجود ندارد مثلا یک معماری ساختمان را در نظر بگیرید نمی آئیم بگویم که معماری این ساختمان بد است میتوانیم بگوئیم که معماری این ساختمان طوری است که خیلی از موارد را ندیده است , به خیلی از مسئل توجه نکرده آینده نگری قرضا نشده است.

-ما میتوانیم در این راستا معماری مان را ارزیابی بکنیم و یکی از بزرگترین هزینه های که میتوانیم صرفه جویی کنیم همین ارزیابی معماری است با ارزیابی کردن معماری ما میتوانیم مشکلاتی که در آینده خواهیم داشت را نداشته باشیم.

اگر یک معماری را ارزیابی کنیم و مطمئن شویم که همه نیازمندیهای ما را پوشش داده است.

قواعد سر انگشتی برای خوب و بد بودن معماری:

بر اساس این نکات و قواعد میتوانیم بفهمیم که یک معماری چقدر توانسته اهداف مورد نظر را پوشش بدهد:

۱-توصیه ها درباره فرایند(Process recommendations):

توصیه هایی فرایندی برای اینکه بتوانیم به یک معماری خوب برسیم:

-بهتر است که یک معماری حاصل خروجی یک معمار باشد یعنی اینکه یک معماری داشته باشیم و یا یک گروهی از معمارها را داشته با شیم که یک نفر رئیس شان هست یعنی تصمیم گیرنده نهایی یک نفر است.

-معمار یک سیستم معماری باید نیازمندیهای Functional سیستم را در قالب یک لیست اولویت بندی شده از صفات کیفی که قرار است معماری آنها را به ما بروسند تهیه بکند و داشته باشد(یعنی معمار باید نیازمندیهای Functional را بصورت لیست شده که اولویت بندی شده ارائه بدهد نیازمندیهای Functional هم همون صفات کیفی هستند و صفات کیفی هم یک چیزهایی مثل کارایی,قابلیت دسترسی , قابلیت تغییر,امنیت و.. هستند)

-معماری ما باید خوب مشتند شده باشند و حداقل یک دید ایستا و یک دید پویا باید برایش ارائه شده باشد که بشود با استفاده از آن این معماری را به همه استک هلدر ها با کمترین زحمت و تلاش فهماند .

-معماری باید بین اتک هلدر ها سیستم بچرخد آنهایی که فعالیتشان باید بر اساس معماری ایجاد شود یعنی بین سازمان توسعه نرم افزار , استک هلدر بازر , مشتری, کابر پایانی دست بدست شود یعنی باید برای همه توضیح داده شود.

-معماری باید بصورت کمی (قابل اندازه گیری)تحلیل شود یعنی مثلا یک عددهایی بدهد یعنی اگر بگوید که اگر از این معماری جلو بروید این مقدار سود میدهد توان عملیاتی اینقدر میشود بازدهی اینقدر میشود. بتوان یک معیار قابل محاسبه و بیان و کاربردی را ارئه کند.

-معماری باید امکان توسعه تکاملی را بدهد در توسعه تکاملی یک نسخه اولیه ساخته میشود و این نسخه اولیه کامل و کامل تر میشود تا نسخه نهایی صادر شود.

دقیقا برای معماری هم همین بحث را داریم وقتی که میخواهیم پیاده سازی بر اساس معیار انجام پذیرد اول یک اسکلت از سیستم می سازیم. اول اساس سیستم که چجوری هست پایه اش ساخته میشود و بعد بر اساس نیازمندیهایی که کشف کردیک کامل و کامل تر میشود.

-در معماری باید منابع رقابتی مشخص شود و راه حل های واضحی برای بکار بردن و چرخ انداختن و حفظ این منابع بیان شود.

این منابع و حوزه هایی که چالش وجود دارد بحث و درگیری وجود دارد نحوه استفاده اش باید در معماری دیده شود و مشخص شود

مثلا اگر نگرانی ما از استفاده از شبکه است در معماری باید مشخص شده باشد که یک دستوالعمل هایی به تیم توسعه بدهیم تا طوری سیستم را طراحی و پیاده سازی کنند که بتوانند پایین ترین ترافیک شبکه را ایجاد بکند یعنی طوری کدنویسی ها را انجام بدهند مثلا تصمیم بگیرند که کد ها را سرور ساید کندد و یا کلاینت ساید تا بتوان ترافیک را پائین بیاورند.

۲-توصیه ها درباره محصول(:Product Recommendations)

-معماری باید ماژول های خوش تعریفی ایجاد کند که در آنها وظایف Functional ی که به آن ماژولها اختصاص داده شده دقیقا بر اساس اصول مخفی سازی اطلاعات و جداسازی نگرانیهای Concern ها اعمال شده باشد.یعنی هر کس فقط اطلاعاتی که مورد نیازش است را خبر داشته باشد همه اطلاعات برای همه فاش نشده باشد .چون برای ارتباط ۲ ماژول لازم نیست که دقیق بدانند که داخل هر یک چه خبر است فقط باید یک سری اطلاعاتی را خبر داشته باشد که نیاز است برای برقراری ارتباطشان.

-هر ماژولی باید یک واسط خوش تعریفی داشته باشد که جنبه های قابل تغییر را مخفی کند تا برای نرم افزار هایی که از آن استفاده میکنند مخفی باشد.

-صفات کیفی که باید به آنها برسیم باید از طریق یک سری تاکتیک های معماری خوش تعریفی به آنها دست بیابیم

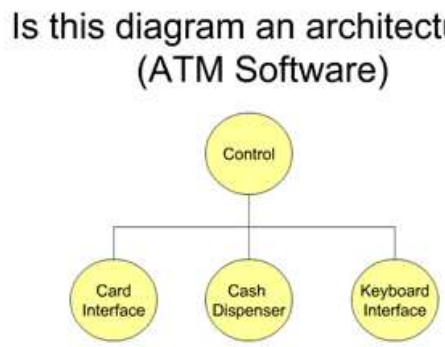
-معماری هیچ وقت نباید به یک ماژول خاص از ابزارات و محصوالات تجاری وابسته باشد

-اگر وابسته باشد هزینه بر است چون اگر آن ابزار تغییر بکند کلا سیستم ما هم تغییر پیدا میکند مثلا به یک سخت افزار خاص وابستگی داشته باشد آن وابستگی باید تعریف شده باشد تا یک حدی باشد که اگر مدلش عوض شد بتواند با کمترین هزینه و اثر دوباره سیستم بتواند خودش را راه اندازی کند.

-باید کاری کنیم که ماژول ها که داده تولید میکنند از ماژول هایی که داده مصرف میکنند جدا باشند اینها نکته های تولیدی است که نا را به MOdifablity میرساند اگر جدا باشند اگر ماژول تغییر کند کمترین اثر را روی ماژول بعدی که داده را مصرف میکند خواهد گذاشت
-اگر سیستمی که داریم طراحی میکنیم قرار است پردازش موازی انجام بدهد یک سری پرسه یا تسک های تعریف کرده باشد که اینها براحتی بتوانند روی Processor های مختلف اجرا شوند یعنی از همان اول باید معماری توزیع شده را دیده باشیم موازی که میگوید باید پردازش توزیع شده را در معماری بتوانیم ببینیم.
-معمار باید یک تعداد کمی از الگو ها ارتباطی ساده را مشخصه بندی کند.

فصل دوم

در این فصل تعریفهای رسمی از معماری نرم افزار رو خواهیم داشت.



این اسلاید یک معماری خیلی ساده از یک ATM رو نشون میده. سوالی که مطرح است اینه؟ آیا این دیاگرام یک معماری است؟ معماری یکسری کامپوننت و کانکتورهای بین آنهاست. در این تصویر چهار کامپوننت و ارتباط بین آنها نشان داده شده است. حالا سوالاتی که مطرح میشود:

- ماهیت این المنتها چیست؟ (پروسس، کلاس، شی، ماژول، تابع و ...)
- وظایف این المنتها چیست؟
- نحوه اتصال بین آنها چیست؟
- اهمیت طرح و جانمایی المنتها چقدر است؟
- عملیات زمان اجرای این سیستم به چه صورتی است؟

تعریف معماری رو اینجا میبینیم.
طراحی نرم افزار برای یک برنامه یا سیستم محاسباتی، ساختار یا ساختارهایی از سیستم هستند که عناصر نرم افزار، خصوصیات قابل مشاهد این المنتها و ارتباط بین آنها را نمایش می‌دهند.
خصوصیات قابل مشاهد از بیرون برای یک المنت یا کامپونت به شرح زیر است:
خصوصیات و فرضیاتی برای المنت که توسط المنت‌های دیگر قابل مشاهد است.

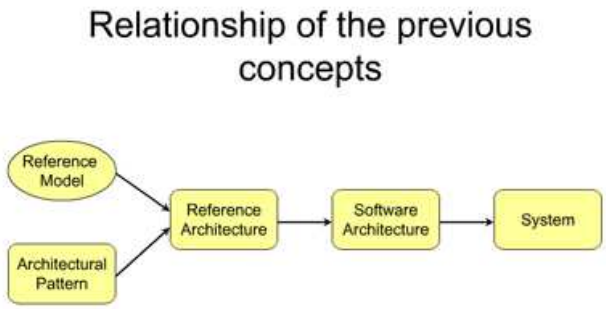
- این المنت چه سرویس‌هایی را ارائه می‌دهد و واسطه‌های قابل دسترسی به این المنت چیست.
- کارایی این المنت چقدر است.
- روشهای مقابله با خطا و شکست المنت چگونه است.
- منابع به اشتراک گذاشته شده توسط المنت چیست؟

معماری نرم افزار به صورت ذاتی این ویژگیها را به صورت انتزاعی بیان می‌کند. کاری ندارد که چگونه تولید می‌شود فقط نحوه استفاده از آن را بیان می‌کند.
هر سیستم فقط یک معماری دارد.
یک معماری می‌تواند دارای چند ساختار باشد.
ارتباط بین اجزا در معماری به صورت انتزاعی تعریف می‌شود.
تعریف ارائه شده برای معماری در مورد خوب یا بد بودن معماری صحبت نمی کند. بلکه متدهای ارزیابی طراحی نرم افزار این مورد را بررسی خواهند کرد.

کدام موضوعات در حوزه معماری هستند؟ موضوعات حوزه معماری موضوعاتی هستند که در سطح انتزاعی طراحی نرم افزار برای ما مهم هستند. هر موضوع در سطح معماری هست اگر مربوط به موارد داخلی المنتها نباشد.

- کارایی بخشی از معماری است که به عنوان یک صفت کیفی معماری مطرح می‌شود.
- رفتار یک المنت در حدی که رابطه آن با سایر المنت‌ها را برای طراحی نمایش دهد در حوزه معماری است.

به عنوان مثال یک ماژول کار مرتب سازی را انجام می‌دهد. اینکه با کدام الگوریتم پیاده شود معماری نیست اما اینکه بر اساس ورودی‌ها با چه سرعتی و کارایی می‌تواند کار را انجام دهد این معماری است.



ما میخواهیم این دیگارام رو بررسی کنیم. که معماری سیستم رو توضیح می‌دهد. ما یک رفرنس مرجع ایجاد می‌کنیم. سپس با انتخاب یک پترن معماری و نگاشت و ترکیب این دو باهم به یک رفرنس طراحی دست پیدا می‌کنیم. با بررسی بیشتر و ایجاد تغییرات در رفرنس طراحی به یک معماری نرم افزار می‌رسیم و از روی آن سیستم طراحی می‌شود. این دیاگرام نشان می‌دهد که چگونه به معماری نرم افزار می‌رسیم. حال قسمت‌های مختلف این دیاگرم بررسی می‌شود.

ما اصطلاحی به نام پترن معماری داریم که به آن استایل هم گفته می‌شود. یک پترن معماری توصیفی از عناصر و نوع ارتباط آنهاست به اضافه مجموعه ای از محدودیت‌ها در هنگام استفاده از آنها. پترن معماری تعریفی مشابه معماری نرم افزار دارد ولی در اصل پایه و پیش نیاز معماری است. یعنی برای ایجاد یک معماری نرم افزار ابتدا نیاز به انتخاب یک پترن معماری است.

- به عنوان مثال: client-server، layered، data-centered، share data، object oriented

به عنوان مثال وقتی از client-server صحبت می‌شود دو نوع المنت یکی به عنوان server و دیگیری به عنوان client مد نظر است که باهم ارتباط دارند و client درخواست سرویس کرده و server ارائه دهنده سرویس است.

- موضوعات بحث نشده در پترن:
  - تعداد المنت‌ها و ارتباط میان آنها در طول اجرا
  - رفتار المنت‌ها در طول اجرا
  - پیکربندی و پیاده سازی

پترن‌ها محدودیت‌هایی را روی معماری تعریف می‌کنند ولی خودشان معماری نیست بلکه انتزاعی هستند برای دسته ای از معماری ها. به عنوان مثال وقتی پترن client-server انتخاب می‌شود نوع پیاده سازی و محدودیت‌های لازم برای معاماری تعریف شده و مد نظر قرار می‌گیرد.

یکی از مهمترین جنبه‌های کاربردی که پترن‌ها ارائه می‌دهند این است که هر کدام برای یکی از صفات کیفی مناسب هستند.

چرا یک معمار یک پترن را انتخاب می‌کند تصادفی نیست و بر اساس میزان اهمیت رسیدن به صفات کیفی است.

بعضی از پترن‌ها راه حل‌هایی را برای بهبود کارایی معرفی می‌کنند. بعضی از پترن‌ها ما را به درجه بالایی از امنیت می‌رسانند. به نوعی که هر پترن یک ویژگی را بیشتر توسعه و گسترش داده و در مقابل ویژگی‌های دیگر ممکن است کمتر شوند. به عنوان مثال پترن لایه بندی کارایی را بالا برده در مقابل ممکن است امنیت سیستم را پایین بیاورد.

انتخاب یک پترن معماری اغلب اولین و اصلی ترین تصمیمی است که معمار انجام می‌دهد.

مدل مرجعی تقسیمی عملیات‌های قابل انجام به اضافه جریان داده بین قطعات است. یا مدل مرجع تقسیم بندی وظایف و عملیاتی است که باید انجام شود و جریان داده‌هایی که باید وجود داشته باشد تا بتوان عملیات را انجام داد.

- یک تقسیم بندی استاندارد از یک مساله شناخته شده با چندین قسمت.
- کامپایلر و DBMS معروفترین مثال برای رفرنس مدل هستند. در اینها یک سری وظایف و مسئولیت تعیین می‌شود و روی ماژول‌های نرم افزاری تمرکز نمی کند. مثل مواردی که DBMS برای تعریف جداول بررسی می‌کند.
- رفرنس مدل یک معماری محسوب نمی شود.

از پترن معماری می‌شود فهمید چه المنت‌هایی داریم و از رفرنس مدل می‌شود وظایف را شناخت.

یک معماری مرجع یک مدل مرجع است که نگاشت شده روی المنت‌های نرم افزار و جریان دادهای میان المنت ها.

- المنت‌ها وظایف و functionalityهایی که در رفرنس مدل تعریف شده اند رو انجام می‌دهند.
- معماری مرجع معماری نهایی نیست ولی خیلی هم از آن دور نیست
- map کردن رفرنس مدل به پترن لزوما یک به یک نیست. یک عملیات می‌تواند توسط چندین المنت انجام شود یا یک المنت می‌تواند در چندین عملیات شرکت داشته باشد.

چرا بحث معماری مهم است؟

- معماری پیچیدگی‌ها را مدیریت و هندل می‌کند.
- با استفاده از معماری گفتگویی بین استک هولدرها ایجاد میکنیم
  - نیازمندی و نگرانی ها
  - زمان
  - بودجه
- تصمیمات اولیه طراحی
  - محدودیت‌های پیاده سازی و پیاده سازها
  - ساختار سازمانی
  - پیش بینی صفات کیفی مد نظر برای سیستم
  - امکان تصمیم گیری در خصوص مدیریت تغییرات
  - کمک به تولید نسخه اولیه (کاهش ریسک)
- معماری یک مدل قابل انتقال و قابل استفاده مجدد تولید میکند
  - ایجاد خط تولید نرم افزار
  - ایجاد نرم افزار بر پایه مولفه‌ها که تولید می‌شوند و به هم متصل شده و یک سیستم را ایجاد می‌کنند و همه این مولفه‌ها را می‌توان در پروژه‌های دیگر استفاده کرد.
  - مستند کردن معماری در ساختار قالب ایجاد ساختار به صورت اتوماتیک
  - پایه ای برای آموزش
- یک مدل زمان اجرا که خود سازگار بوده و هم قابل پیکربندی است.

معایب و مشکلات معماری

دیدن تغییرات در معماری در همه موارد قابل دستیابی و رسیدن نیست و باید دید در کدام سطح تغییرات اعمال می‌شود:

- محلی (در یک کامپوننت): تغییر در یک کامپوننت ممکن است در کامپوننت‌های مرتبط به این کامپوننت نیز تاثیر بگذارد.
- محلی نباشد (در چندین کامپوننت): اگر تغییرات دامنه اش غیر محلی است باید تمهیدات و عوامل موثر آن مد نظر باشد.
- تغییرات معماری
  - تغییرات در سطح کل معماری بسیار سخت و غیر قابل انجام است.
  - با استفاده از یک معماری ممکن است به بعضی از صفتهای کیفی برسیم و به برخی از صفتها نیز نرسیم.

مقایسه معماری نرم افزار و معماری سیستم

- معماری نرم افزار جزئی از معماری سیستم است. معماری سیستم هم نرم افزار را می‌بیند و هم سختت افزار را.

معماری سخت افزار از سیستم‌هایی که موجود است باید استفاده کند ولی در معماری نرم افزار می‌توان سیستم‌های نرم افزاری جدید ایجاد کرده و با استفاده از آنها سیستم را طراحی کرد.

ساختارهای معماری و دیدهای معماری

در ساختار ساختمان سازی و معماری ساختمان، یک سری طرح‌های اولیه را داریم:

- نقشه
- نماهای مختلف از ساختمان
- برق کشی
- لوله کشی

در معماری نرم افزار هم همین است. چندین ساختار و دید از سیستم در اختیار ما قرار می‌گیرد.

هر کدام از این دیدها یک تک موجودیت را مشخص می‌کنند. یعنی همون طور که چندین ساختار در یک معماری نرم افزار داریم، چندین دید نیز در سیستم و معماری آن وجود خواهد داشت. تمرکز روی هر ساختار یک دید به شما می‌دهد. به عنوان مثال برقکار ساختمان که ساختار برق را می‌سازد تنها دید برق ساختمان را می‌بیند. در نرم افزا هم همین است. Viewها دیدهایی هستن که استک هولدرها از سیستم دارند.

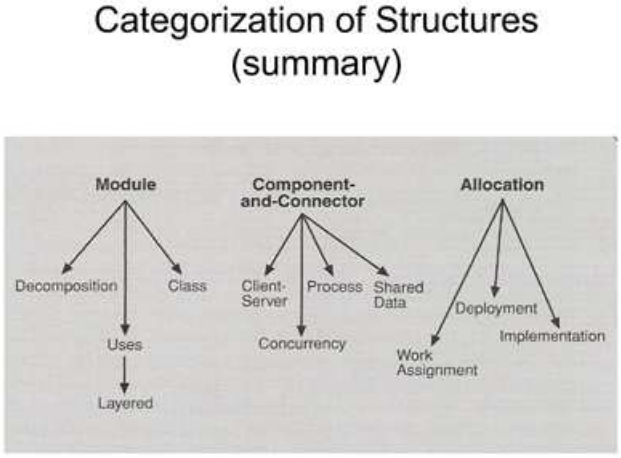
ساختار و دید

ساختار یک مجموعه از عناصر به هم پیوسته و ارتباط بین آنها است. که برای هر ساختار ویژگی‌های زیر وجود دارد.

- انواع المنت‌های ساختار
- ارتباطات بین المنت ها
- محدودیت‌های لغوی روی ساختارها
- معانی دیاگرامهایی که برای ساختار وجود دارد
- اصول و راهنماهای در مورد ساختار
- هدف این ساختار و مورد استفاده ساختار

view یک نمایش یا ارائه مجدد از معماری نرم افزار است بر اساس یک ساختار که توسط معمار نوشته شده و توسط استک هولدرها خوانده می‌شود. یک نمونه از ساختار است. یعنی بر اساس اینکه با کدام استک هولدر کار داریم باید view مربوط به آن ارائه شود.

معماری نرم افزار توسط چندین view مستند می‌شود.



سه دسته ساختار (Structure) داریم

۱. Module
۲. Component and Connector
۳. Allocation

ساختار دسته اول سیستم را به صورت Code Base می‌بینند. دسته دوم ساختار اجرایی سیستم را می‌بینند. دسته سوم ارتباط بین سیستم و محیطی که قرار است سیستم پیاده سازی شود را میبینند. هر کدام از این دسته خود به چندین زیر دسته تقسیم می‌شوند که بررسی خواهد شد.

Viewهای موجود در سیستم نیز به چهار دسته اصلی و یک مورد فرعی تقسیم می‌شود که توضیح داده خواهد شد

یاد آوری :

تعریف معماری نرم افزار (رسمی و اصلی) : معماری نرم افزار ، یک برنامه یا سیستم محاسباتی ، تشکیل شده از یک یا چند ساختار از سیستم است ، که این ساختارها از عناصر نرم افزاری و ویژگی های قابل رویت این عناصر از بیرون و ارتباط بین عناصر تشکیل شده است .

تمام بحثهای در اینجا اشاره دارد به این که معماری خودش یک ساختار است ، طبق روال این کتاب ابتدا با ساختارها آشنا می شویم و مراحلی را طی می کنیم تا به معماری میرسیم ، در واقع الگو های معماری (Architectur Pattern) سیستم را کشف می کنیم و ارتباط بین انها و محدودیت های آنها را مشخص میکنیم و براساس مدل مرجع (Refrence Model) تقسیم وظایف میکنیم و با نگاشت مدل مرجع برروی الگوهای معماری به معماری مرجع میرسیم و با بهبود آن معماری نرم افزار نهایی را تولید می کنیم و بعد به سیستم نهایی میرسیم .

با توجه به این که معماری نرم افزار یک یا چند ساختار می باشد در اینجا به تعریف ساختار اشاره میکنیم:

Structures and Views (cont)

Structure : ساختار یک مجموعه بهم پیوسته از عناصر و ارتباط بین آنها میباشد . برای هر یک از این ساختار ها ما می توانیم این جزییات را مشخص کنیم :

_ انواع عناصر

_ انواع روابط

_ مجموعه ای از محدودیت نحوی (محدودیت های معنایی برروی عناصر)
_ معناشناسی نمودار (مفاهیم چینش و نمودار)
_ منطق، اصول، و دستورالعمل (منطق و اصول حاکم براین ساختار)
_ برای چه اهداف آن مفید است

ساختار های اصلی که ما داریم عبارتند از :

۱. Module Structure
۲. Component and Connector Structures
۳. Allocation Structures

مثل یک ساختمان که چندین قسمت داره مثل : نمای بیرونی – نمای داخلی – فنداسیون- پی –دکور داخلی - برق کشی – لوله کشی و ..... معماری هم از یکسری ساختار تشکیل شده است .

View : وقتی از بیرون یک Stackholder (ذینفع) به معماری ما نگاه میکند به یک ساختار خاص از سیستم نگاه می کند و همه stackholder ها به همه ساختارها کار ندارند ، ار یک دید خاص به سیستم نگاه میکنند ، دیدی که انها دارند یک view است .

طبق تعریف در کتاب View یک نمایش مجدد از معماری نرم افزار براساس یک ساختار که توسط معمار نوشته شده و توسط Stackholder خوانده میشود ، پس یعنی View به ازای یک Stackholder خاص ایجاد می شود ویک روش نمایش معماری است ، در واقع آخر سر که میخواهید معماری تان را بر روی کاغذ بیاورید ، براساس View ها می کشید ، پس یک معماری نرم افزار براساس یک سری Viewمستند شده است.

مثلا در ساختمان ، یک لوله کش (Stackholder) وقتی به ساختمان نگاه میکند ، از معماری ساختمان سیستم لوله کشی ساختمان را میبیند و ....

ما ۵ نوع View داریم که همه Stackholder ها را پوشش می دهد و ۵ تا Stackholder داریم :

۱. سازمان توسعه نرم افزار
۲. بازار
۳. مشتری
۴. کاربر نهایی
۵. پشتیبانی سیستم

همه view ها یک به یک با هم Match نمی شوند ، ما با همه view ها کار داریم ولی به همه جزییات کار نداریم ، مثلا ما به عنوان کاربر نهایی از ساختمان استفاده می کنیم ودر ساختمان زندگی می کنیم ما از همه عناصر ساختمان استفاده می کنیم ولی با همه جزییات انها کاری نداریم ،یعنی هر Stackholder به دید خود با view خاص کار دارد .

Categorization of Structures

در این قسمت به بررسی ساختارهای اصلی می پردازیم .

Orthogonal: این طبقه بندی ساختارهای ، Module Structure و Component and Connector Structures و Allocation Structures متعامد هم هستند ، یعنی این سه دسته با هم نقاط مشترک دارند ، یعنی این گونه نیست که یک ساختار فقط Module باشد و با ساختار Component and Connector کاری نداشته باشد ، این سه ساختار با هم همپوشانی دارند و در طول هم هستند .

Module Structures . 1

در این ساختار المنت های ما ماژول ها هستند که واحد های پیاده سازی میباشند و ارتباط بین آنها ، اهداف ، انواعشان و نحوه ای چینش بین آنها مطرح است .

ماژول ها شیوه مبتنی بر Code را برای یک سیستم نمایش می دهد و ساختاری که یک برنامه نویس باید ان را ببیند ، در نتیجه برای Stackholder های تیم توسعه نرم افزار یک View وجود دارد. (Code base)

فعالیتهای و کارهای که باید سیستم انجام بدهد مثلا براساس یک ورودی یک خروجی بدهد را به یک ماژول اختصاص می دهد و هر ماژول مسولیت های کاربردی مختص به خودش را انجام می دهد .

در این ساختار تاکید کمتری در مورد چگونگی نتیچه کار نرم افزار در زمان اجرا وجود دارد .

ساختار ماژول به ما اجازه می دهد که به سوالات زیر پاسخ بدهیم :

مسئولیت عملکردی اولیه اختصاص یافته به هر یک از ماژول ها چیست؟ هر ماژول چه کاری انجام میدهد ؟

کدام عناصر های نرم افزاری دیگری مجاز به استفاده از یک ماژول هستند ؟

نرم افزار های دیگری که بتوانند از این مازول استفاده کنند ، کدامند ؟

ارتباط بین ماژول های با یک دیگر چه صورت است ؟ از طریق روابط تعمیم و یا تخصیص (به عنوان مثال، ارث بری) – بحثهای شی گرایی – از پدر به پسر specialization – از پسر به پدر generalization

2 . Component and Connector Structures مولفه – اتصال

در این ساختار المنت های ما مولفه های زمان اجرا هستند که واحد های منطقی از محاسبات و کانکتورها (اتصالات – وسایل ارتباطی بین مولفه ها هستند) میباشند . در اینجا تعامل و فعالیتهای زمان اجرا را می بینند . (Run Time)

ساختار مولفه – اتصال کمک میکند که به سوالات زیر پاسخ بدهیم :

اجزای اصلی اجرای در سیستم چه چیزهای هستند و تعامل (تبادل داده) بین آنها چگونه است ؟

مخزن داده های که به اشتراک گذاشته اند کدامند ؟

بخش های از سیستم که کپی ازشون ایجاد شده کدامند ؟

پیشرفت دادها و جریان داده ها از طریق سیستم چگونه است ؟

بخش های از سیستم که به صورت موازی اجرا می شوند ، کدامند ؟

چگونه ساختار سیستم در زمان اجرا می تواند تغییر کند ؟

پس هر چیزی در مورد زمان اجرا ، جریان داده و تعامل بین مولفه ها را در ساختار مولفه – اتصال داریم.

3 . Allocation Structures تخصیص

ساختار تخصیص ارتباط بین المنت های نرم افزاری و المنت هایی که در محیط پیاده سازی خارجی ما وجود دارد را نشان میدهد .

ارتباط بین سیستم و محیط پیاده سازی ، یعنی افراد و سخت افزار محیط پیاده سازی .(افراد - Hradware)

پاسخ به این قبیل پرسش ها :

چه المنتی روی کدام پروسسر یا پردازنده ما اجرا می شود ؟

فایل های که برروی هر المنت هستند در طول فازهای توسعه ، تست در کجا ها ذخیره می شوند ؟ چه داده ای روی hard / Data storage/... در طول فازهای توسعه ، تست سیستم ، ذخیره می شود ؟

چه المنت های نرم افزاری را به چه تیم های نرم افزاری اختصاص می دهیم و قراره چه کارهای بر روی آن انجام دهند ؟

در طراحی معماری این سه ساختار که براساس آنها سه تصمیم گیری ایجاد می شود شامل بخش زیر می باشند :

سیستم چگونه براساس واحد های کد (ماژول ها) ساختار بندی شده است ؟

سیستم چگونه براساس رقتار مجموعه ای از عناصر در زمان اجرا (اجزا) و تعاملات بین این عناصر (اتصالات) ساختار بندی شده است ؟

سیستم چگونه در ارتباط با بخش های غیر نرم افزاری (به عنوان مثال، پردازنده، سیستم های فایل، شبکه، تیم توسعه، و غیره) ساختار بندی شده است ؟

۱. Module Structure

- Elements عناصر : ماژول (واحد های پیاده سازی) . ماژول ها یک شیوه Code Base از سیستم را در نظر میگیرند .
- Specifies ویژگی ها – مشخصات :
  - فعالیتهای فاکشنالی که این ماژول ها استفاده می کند .
  - المنت های دیگه که اجازه دارند از این ماژول استفاده کنند .
  - مشخص کردن ارتباط بین ماژول ها که روابط تعمیم و تخصیص
- فعالیتهای زمان اجرا در این view برای ما کم اهمیت است .

۱. Decomposition Structure 1

ساختار تجزیه مهم ترین ساختار معماری نرم افزار است ، یعنی معماری نرم افزار حتی در ساده ترین سیستم حتما دارای ساختار تجزیه است .

Elements عناصر : ماژول با ساختار سلسله مراتبی ، یعنی هر ماژول به چه ماژولهای دیگری متصل است .

Relations ارتباطات : ماژول زیر مجموعه ماژول دیگر است ، یا اینکه اطلاعات را باهم به اشتراک میگذارند .

Functional Example این ماژول در چه حوزه ای به درد ما می خورد یا در چه حوزهای کاربرد دارد :

قابلیت تغییر در سیستم ، تضمین اینکه تغییرات ، حداقل اثر را روی ماژول داشته باشد ، اندازه گیری قابلیت تغییر در سیستم با استفاده از ساختار تجزیه امکان پذیر است ، اینکه ما بر طبق این ساختار چگونه مازول ها را چیدیم که در صورت ایجاد تغییر ، حداقل این تغییر وجود داشته باشد .

اغلب یه عنوان پایه ای برای سازمان پروژه توسعه (طراحان و برنامه نویسان) استفاده می شود و در ساختار های مستند سازی ، طرح های تست و مجتمع سازی سیستم مشارکت دارد و استفاده می شود .

مهم ترین سند یا مستند یا ساختار معماری ، ساختار تجزیه است .

۲. Uses Structure 1

Elements عناصر : ماژول ها، روش ها Procedure، و یا منابع ای که مازول ها روی ان با هم تعامل دارند یا از طریق آنها باهم ارتباط دارند ، واسط های بین ماژول ها

Relations ارتباطات : uses نشان میدهد چه ماژولی از چه ماژول دیگری استفاده می کند مثل خط تولید یعنی مرحله یک (ماژول اول) به درستی انجام شود تا مازول دوم بتواند از ان استفاده کند و کارش را انجام دهد (مثل دستگاه ATM : ماژول اول صحت رمز را تایید می کند و در صورت تایید ماژول اول ، ماژول دوم اطلاعات حساب مثل مانده حساب را در اختیار کاربر می گذارد).

Functional Example این ماژول در چه حوزه ای به درد ما می خورد یا در چه حوزهای کاربرد دارد :

اجازه می دهد تا توسعه تدریجی / افزایشی داشته باشیم .

۳. Layered Structure 1

Layered structure یک زیر مجموعه از ساختار Uses است .

Elements عناصر : لایه ها ، هر لایه یک مجموعه بهم پیوسته از قابلیت ها (فانکشنالیتی های)مرتبط به هم هستند .

Relations ارتباطات : هر لایه یک سطح انتزاع برای لایه قبلی و بعدی ایجاد می کند و لایه n میتواند از سرویس های لایه n-1 استفاده کند .هر ماژول زیر مجموعه یک ماژول دیگر است یا ارث بری و یا فقط برای استفاده بهم متصل شده اند ، اینکه برخی مازول ها هم سطح هم هستند یا اینکه یک ماژول باید تولید شود تا مازول دیگر استفاده کند یا نولید بشود .

Functional Example این ماژول در چه حوزه ای به درد ما می خورد یا در چه حوزهای کاربرد دارد :

لایه ها اغلب برای انتزاع طراحی می شوند که ویژگی های پیاده سازی را از لایه های زیرین پنهان کنند که همان قابلیت Portability است ، وقتی می خواهیم میزان Portability یعنی قابل حمل بودن سیستم را نشان بدهیم ، باید ساختار لایه ای را نشان بدهیم و اینکه بگیم به چه سخت افزاری و چه سیستم عاملی نیاز دارد .

۴. Class Structure 1

تحلیل شی گرایی

Elements عناصر : Class کلاسها

Relations ارتباطات : ارتباط بین کلاسها (ارث بری inherits from / یا یک شی از آن است is an instance of) ارث بری و نمونه سازی

Functional Example این ماژول در چه حوزه ای به درد ما می خورد یا در چه حوزهای کاربرد دارد :

هر وقت راجع به reuse بودن سیستم حرف بزنیم باید از ساختار کلاس حرف بزنیم (بحث شی گرایی) ، بحث کردن و دلیل آوردن برای اینکه سیستم قابلیت استفاده مجدد دارد و عینا می توان در سیستم دیگر از آن استفاده کرد و بهره برد ، ساختار کلاس به ما اجازه میدهد که Function ها را به صورت افزایشی اضافه کنیم .

۲. Component and Connector Structures

- Elements عناصر : مولفه های زمان اجرا (واحد های منطقی محاسباتی) و اتصالات (ارتباط سایر مولفه ها) / Connector مثل مسیر های شبکه ای یا Bus های که بین مولفه ها وجود دارد و Component مثل Process , DataStore , Replicate
- Specifies ویژگی ها – مشخصات :
  - مولفه های اجرای اصلی و چگونگی تعامل انها با هم .
  - مخزن داده های اصلی که بین مولفه ها به اشتراک گذاشته شده است .
  - کدام قسمت های از سیستم کپی شده است در واقع برای رسیدن به برخی مسایل مثل بالابردن دسترسی پذیری ،مقیاس پذیری ، دسترسی پذیری ، یک نسخه کپی از سیستم ایجاد می کنیم .
  - جریان داده در سرتاسر سیستم به چه صورت انجام می پذیرد .
  - چه بخش های از سیستم میتوانند به صورت موازی اجرا شوند .
  - ساختار سیستم چگونه می تواند در زمان اجرا تغییر کند .

Process Structure 2.1

Elements عناصر : processes or threads فرایندها یا نخ ها

Relations ارتباطات : attachment ارتباط بین فرایندها (که اجازه ارتباط برقرار کردن بین دو فرایند ، فرایندها به صورت همزمان اجرا شوند یا همگام سازی شوند یا یک فرایند باعث kill کردن یا خروج دیگری شود) (اجازه دادن – همگام سازی – عملیات ممانعت سازی)

Functional Example این ماژول در چه حوزه ای به درد ما می خورد یا در چه حوزهای کاربرد دارد :

مهندسی اجرای سیستم (تعداد فرایندهای که میتواند اجرا کند) – کارایی سیستم (فرایندهای که در واحد زمان با موفقیت انجام میشوند) _ در دسترس بودن (فرایندهایی که در واحد زمان با موفقیت انجام شده اند)

Shared Data or Repository Structure 2.2

Elements عناصر : مخزن ذخیره سازی ، تولید کنندگان داده ، مصرف کنندگان داده

Relations ارتباطات : جریان داده

Functional Example این ماژول در چه حوزه ای به درد ما می خورد یا در چه حوزهای کاربرد دارد :
اطمینان از عملکرد خوب و جامعیت داده ها.

Client-Server Structure 2.3

Elements عناصر : کلاینت و سرور
Relations ارتباطات : ارتباط بین کلاینت و سرور از طریق توافق بین انها است که به آن پروتکل می گویند و زیر ساخت تبادل پیام
Functional Example این ماژول در چه حوزه ای به درد ما می خورد یا در چه حوزهای کاربرد دارد :
جدا سازی نگرانی ها (پشتیبانی از تغییر پذیری)
توازن بار (پشتیبانی از کارایی در زمان اجرا) – اگر بار روی یک سرور زیاد باشد یعنی تعداد درخواستها بالا باشد ،در این حالت یم سرور موازی فعال می کنند یا لز مهاجرت کد یا داده استفاده میکنند و درخواست کلاینتها به سرور دیگر انتقال میابد (معماری نرم افزار پیشنهاد شده برای توازن بار - Grid- Sensor Vanet -Cloud)

Concurrency 2.4

همزمانی .این ساختار جزء و کانکتور اجازه می دهد تا فعالیت ها به صورت همروند انجام شوند و وقتی فعالیت ها همروند اجرا می شوند با هم تداخل دارند ، مثلا بر سر منابع اشتراکی با هم رقابت میکنند که مشکل ناحیه بحرانی ایجاد میکنند و راه حل ان از طریق سمافور میباشد .

3. Allocation Structures

- ارتباط بین نرم افزار و عناصر نرم افزار در یک یا چند محیط خارجی که قرار نرم افزار روی ان ایجاد یا اجرا بشود .
- Specifies ویژگی ها – مشخصات :
  - پردازنده های که هر عنصر نرم افزاری را اجرا میکند .
  - فایل های ذخیره شده در هر کدام از المنت های نرم افزار در طول فاز توسعه.
  - انتصاب قسمت های مختلف نرم افزار به تیم توسعه نرم افزار .

Deployment Structure 3.1

نشان می دهد چگونه نرم افزار به سخت افزار انتصاب داده شده است .
Elements عناصر : نرم افزار ، سخت افزار
Relations ارتباطات : جریان داده
Functional Example این ماژول در چه حوزه ای به درد ما می خورد یا در چه حوزهای کاربرد دارد :
اطمینان از عملکرد خوب و جامعیت داده ها.

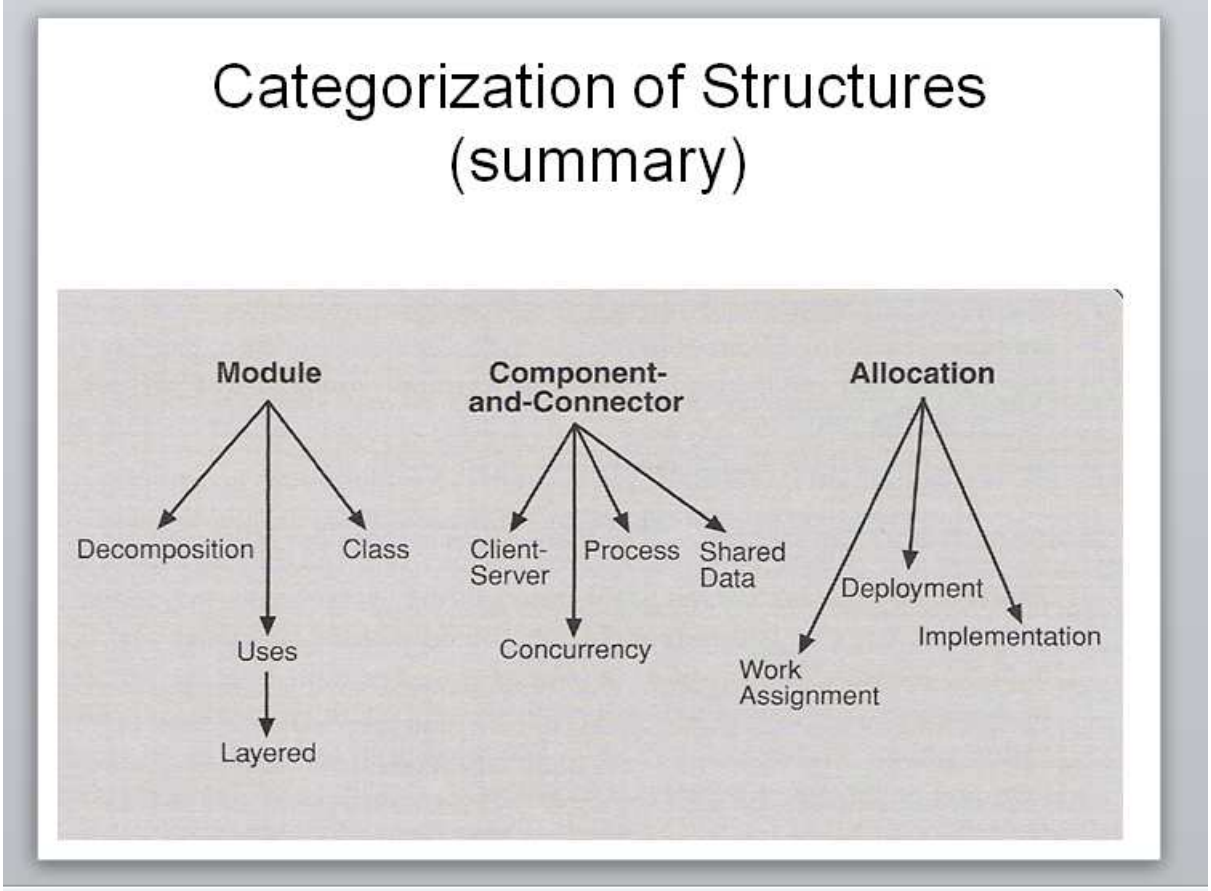
Implementation Structure 3.2

نشان می دهد چگونه المنت های نرم افزاری که معمولا ماژول ها هستند در محیط های توسعه سیستم ، یکپارچگی سیستم (همه قسمت های سیستم را به هم وصل می کند) ، مدیریت پیکر بندی سیستم ، به system File ، map می شوند .
Elements عناصر : هر واحد منطقی (به عنوان مثال : یک فایل – یک تیکه کد – یک تیکه برنامه – یک بلوک کنترل فرایند - Table)
Relations ارتباطات : محل ذخیره سازی
Functional Example این ماژول در چه حوزه ای به درد ما می خورد یا در چه حوزهای کاربرد دارد :
در حوزه مدیریت فعالیت های توسعه و فرآیند ساخت از این ساختار استفاده می شود .

Work Assignment Structure 3.3

انتصاب مسولیت پیاده سازی و یکپارچه سازی ماژول ها به تیم های توسعه مناسب (چه کسی قراره چه کاری را انجام دهد) .
Elements عناصر : هر واحد منطقی
Relations ارتباطات : انتصاب داده
Functional Example این ماژول در چه حوزه ای به درد ما می خورد یا در چه حوزهای کاربرد دارد :
معمار با استفاده از این ساختار می تواند خبرگی مورد نیاز در هر تیم را تعیین کند و به جای اینکه یک مازولی که همه دارن ازش استفاده می کنند را هر تیم استفاده کننده جدا گانه آن را تولید کند ، با ابزاری که این ساختار در اختیار معمار قرار ، نقاط مشترکش را می گیرد و یک تیم ان را پیاده سازی میکند و بقیه از ان استفاده میکنند .
در واقع ما می توانیم با استفاده از این ساختار ، عاملیت های رایج را پیدا کنیم و به یک تیم منقرد ارجاع بدهیم ، به جای اینکه هر تیمی خودش جداگانه آن را تولید کند .
در این دو ساختار work Assignment و Implementation ، عناصر ما هر واحد منطقی هستند .

Categorization of Structures(summary)



Notes نکات :

- هر ساختاری برای خودش به درستی استفاده می شود ولی همه ساختارها در همه پروژه ها استفاده نمی شوند یعنی اینگونه نیست که در یک سیستم نرم افزاری ، همه ساختارها را معرفی کنیم یا همه ساختارها را داشته باشیم ، در واقع نسبت به نیاز سیستم نرم افزاری ما به بعضی از ساختارها احتیاج داریم ، بعضی اوقات سیستم را با یک تک ساختار انجام میدهیم . مثل ساختار تجزیه
- ساختار ها مستقل نیستند و باید با هم در نظر گرفته شوند .

— به عنوان مثال ارتباط مازول با مولفه – اتصال می تواند many to many باشد (یعنی یک مازول از ساختار مازول می تواند یک یا چند کامپوننت را از ساختار مولفه – اتصال پیاده سازی کند و یا برعکس ، یعنی می توانید در ساختار Decomposition Structure یک ماژول معرفی کنید که این مازول چندتا پروسس را از ساختار مولفه – اتصال اجرا کند یا یک پروسس از ساختار مولفه – اتصال اجرا کنید که چندتا ماژول از ساختار مازول را درگیر کند .)

— برخی از ساختارها می توانند در برخی از سیستم ها مشابه باشند .

— برخی از ساختار ها می توانند ترکیبی باشند (برای مثال همه ساختارهای مولفه – اتصال ممکن ترکیب شده باشند درون یک ساختار تکی) مثلا ما یک سیستم داریم و یک ساختار برای آن معرفی کرده ایم با و یک ساختار داریم که هر المنتی از آن خودش برای خودش یک ساختار دارد ، مثلا یک ساختار مولفه – اتصال معرفی کرده ایم ، خود این ساختار مولفه – اتصال ترکیبی از یک المنت از نوع پروسس و یک المنت دیگر از نوع همزمانی است یا در یک ساختار Decomposition یک ماژول معرفی میکنیم که هر یک از المنت های ان خودش یک ساختار کلاینت – سرور براش معرفی شده است .

Relating Structures to Each Other ساختارها بهم مرتبط هستند :

اگر چه ساختارها ، دیدگاه های (پرسپکتیوهای) متفاوتی از سیستم به ما میدهند اما از همدیگر مستقل نیستند ، المنت های یک ساختار می توانند به المنت های ساختار دیگر متصل باشند و باید در مورد دلایش بحث و گفتگو شود .

به عنوان مثال، یک ماژول در یک ساختار تجزیه ممکن است به یک یا بخشی از یک یا چند، قطعه (کامپوننت)در یک ساختار مولفه – اتصال در زمان اجرا نگاشت (map) شده باشد .

به طور کلی، نگاشت many to many چند به چند است.

4 + 1 View Model of Architecture

(یعنی ۴ view اصلی و یک view فرعی)
در ابتدای جلسه گفتیم که view یک انتزاع (instance) نمونه از معماری است ، تمایش مجددی از معماری که توسط معمار نوشته می شود و توسط stackholder ها خوانده می شود .

Logical منطقی : اشیاء و کلاس ها، عناصر که "انتزاعی کلید" دارند ، اشیاء و یا کلاس در Object Oriented هستند.
(نمایش ماژول module view – وقتی از دید ساختار ماژول به معماری نگاه کنیم میشود دید منطقی)

Process فرآیند : آدرس بندی همزمانی و توزیع قابلیت عملیات ها به صورت همزمان / همروند که در سیستم انجام می شود .
(نمایش مولفه – اتصال component and connector view – وقتی از دید ساختار مولفه – اتصال به معماری نگاه کنیم میشود دید فرآیندی)

Development توسعه : سازمان ها از ماژول های نرم افزاری، کتابخانه ها، زیر سیستم های، و واحد های توسعه را نشان می دهد.
(نمایش انتصاب allocation view – وقتی از دید ساختار انتصاب به معماری نگاه کنیم میشود دید توسعه ای)

Physical فیزیکی : المنت ها چگونه به cpu ها و node ها ی(گره ها ی) ارتباطی متصل شده اند . در اینجا هم دید allocation view داریم و هم دید deployment view

(نمایش انتصاب و استقرار deployment & allocation view – وقتی از دید ساختار انتصاب و استقرار به معماری نگاه کنیم میشود دید فیزیکی)

Scenarios سناریوها : (use Case دیاگرام ها) این view ها خودشان برای نمایش به Stackholder ها ی مختلف ارایه داده میشود .

نهایتا با استفاده از این ۴ view ، معماری معرفی میشود .

فصل چهارم

Quality Attributes

یادآوری: تعاریف- چرخه ی کسب و کار را خواندیم

در فصل دوم که خیلی مهم بود یک تعریف رسمی از معماری را داشتیم- ساختار یا ساختارهایی است یا سیستم محاسباتی که از یک سری المان تشکیل شده است. خصوصیات قابل دید این المان ها (المنت ها) و ارتباطات بین آن ها می باشد.

سه دسته استراکچر داریم

ماژول استراکچر Allocation structure ,component & connect structure

از هر نمونه چند مثال زدیم و در انتها view ها را نام بردیم و گفتیم که یک نمونه از استراکچر بهش می گن view یا دیدی که stack holder های خاص نسبت به استراکچر معماری یا معماری دارند براساس نیاز خودشون، view ها را تشکیل می دهند که ۱ + ۴ view را خواندیم، یکی از این view هایی را که خواندیم آخرین view سناریو بود که ما در این جلسه می خواهیم درمورد اون صحبت کنیم کلاً اسم فصل جدید ما (فصل چهارم) Quality Attribute یا صفات کیفی است. از اول جلسه ای که داشتیم تا الان مدام گفتم که معماری مهم ترین هدفش این که ما رو به صفات کیفی برسونه حالا می خواهیم ببینیم این صفات کیفی چی هست و چه جوری می شد اونها رو نمایش داد. و بیانشون کرد.

طبقه بندی کلاسیکی که در مورد نیازمندی ها وجود داره به این صورت است که نیازمندی ها را به دو دسته Functionality یا non Functional یا غیر عملیاتی تبدیل می کند. قبلاً هم این رو گفته بودم که نیازمندی های ما یا عملیاتی هستند [فصل سوم case study هست که درس نمیدیم] یا غیر عملیاتی هستند اگر خاطرتون باشد ما می گفتیم نیازمندی های Functional همون خواسته های عملکردی مربوط به خود سیستم است. سرویس هایی که خود سیستم باید ارائه دهد. پاسخ هایی است که سیستم باید به ورودی ها بدهد. اون کارهایی است که کاربر در سند خواسته هایش می گوید و نیاز هست به آن ارائه شود و یا در کل (لُپ مطلب) کارهایی است که سیستم باید انجام دهد. و به خاطر انجام دادن آن ها قرار است سیستم ایجاد شود. نیازمندی هایی که ما همیشه در بحث های مهندسی نرم افزار یک و دو رو می دیدیم و می خواستیم اون ها را داشته باشیم. و بهشون برسیم و کشفشون کنیم با این روش های مشاهده مصاحبه، پرسش نامه و چیزهای دیگه، اما نیازمندی های غیرعملیاتی که اسمش را صفات کیفی هم می توانیم بزاریم مستقیماً به عملکردهای تحویل داده شده سیستم، بر نمی گردد (مربوط نمی شود) دقیقاً اون چیزهایی نیست که باید سیستم تحویل بدهد، مستقیماً، بلکه به گونه ای از محدودیت ها گفته می شود که توی سرویس ها یا عملکردهایی که سیستم باید ارائه بدهد اعمال می شود این یک تعریف جدیدی بود که تا حالا گفته نشده بود. مثلاً سیستم پرسنلی را در نظر بگیرید از نیازمندی های عملکردی این است که از مشخصات پرسنلی منو بتوان حقوق را ثبت کند محاسبه کند وضعیت حضور و غیاب رو بتواند حساب کند و کلیه مسائل مالی ، که مربوط به پرسنل می شود را محاسبه کند یا یک اتوماسیون اداری را در نظر بگیرید تمام مکاتبات را بتواند انجام بدهد.ولی این ها نیازمندی های Functional آن می باشد.

اما نیازمندی های non Functional چیه؟

نیازمندی های non Functional یعنی این که من می خواهم مستقیم روی عملکرد هایی که تأثیرگذار نیست (مربوط نمیشه)، بلکه محدودیت روشه صحبت کنیم، یعنی این که می خواهم سیستم من امنیت داشته باشه یعنی یک محدودیت دسترسی را تعریف کنیم، سیستم من می خواهم performance داشته باشد یعنی این که بتونی به سرعت به event هایی که وارد سیستم می شود چه پاسخی بدهی در کوتاه ترین مدت. سیستم من باید Availability داشته باشد یعنی اگر که این سیستم من دچار مشکل شد یا خرابی براش به وجود آمد مشکلش حل بشه یعنی این که قابلیت تغییر داشته باشد قابلیت استفاده داشته باشد حالا ما شش صفت کیفی برای سیستم داریم چند تا صفت کیفی هم برای بازار داریم و چند صفت کیفی هم برای خود معماری داریم. که توی این فصل راجع به این ها صحبت می کنیم.

خوب پس با این تعریف از Quality Attribute ها و Functional Requirement هایی که داریم فصل مان را شروع می کنیم.

مطلبی که این جا بیان کرده گفته که در بحث های مهندسی نرم افزار سابق این طوری بود که ما اول می آمدیم یک نرم افزار را می ساختیم که نیازمندی های Functional را برای ما ایجاد بکنه و برسونه سپس سعی می کردیم که بیام اضافه کنیم یا تزریق بکنیم نیازمندی های غیر Functional مان رو، این ایده منجر شده به فقدان منابع و نهایتاً کیفیت ضعیف یعنی این کار جواب نمی دهد. بلکه چه راهی درسته؟

راه درست این که ما باید طراحی بکنیم کیفیت رو از همان ابتدا، از همان ابتدا یعنی چه؟ یعنی از همان سطح معماری، چی رو مثال بزنم آقای بابایی؟ Functional و non Functional و Function Reotu ها نیازمندی هایی که سیستم باید داشته باشد، عملکردهایی است که سیستم باید انجام

بدهد. نیازمندی هایی کاربر می شود همون چیزهایی که سیستم به دلیل رسیدن به آن به وجود می آید. این می شود نیازمندی های Functional. این همان سند خواسته های کاربر می شود. که باید تهیه بکنیم.

اما non Functional Requirement ها محدودیت هایی هستند که روی این سرویس ها و عملکردها که از سیستم می خواهیم تعریفشون می کنیم. مثل امنیت، قابلیت تغییر، قابلیت استفاده، قابلیت تست، کارایی و مانند این ها. پس با این صحبتی که این جا داریم به این نتیجه می رسیم که ما باید نیازمندی های کیفی خود را همان اول ببینیم یعنی در همان سطح معماری که شما به فکر ایجاد ماژول های سیستم هستید.

Component هایی هستند که نیازمندی های Functional یا Responsibility هایی که اون هفته با هم می خوندیم را فراهم بکنه و برای ما در واقع security فراهم کنند، همون لحظه باید نیازمندی های کیفی رو هم ببیند اما چگونه این کار انجام می شود را در این درس با هم می خوانیم.

قبل از این که شروع بکنه می آید و نیازمندی های کیفی را اسم ببره و یک روش برای نمایش نیازمندی های کیفی که همان سناریوها هستند ارائه بدهد.

یک سری مقدمات را اول ذکر می کند و پس از یک پیش زمینه ای می رود سراغ آن، مثل این اسلاید که گفته Functionality و معماری، گفته فلان صفات های کیفی و function های ما orthogonal هست یعنی مُتِعامِد هستند یعنی در طول هم دیگر هستند و یعنی هیچی صفت کیفی قابل دسترسی نیست با هر سطح مطلوبی از Functionality و ّFunctionality هم ممکنه قابل دسترسی نباشد ، اما ّFunctionality می تواند قابل دسترسی باشد به هر روش یا راهی که می تواند معماری نباشد.

صفت کیفی در هر سطح مطلوبی از ّFunctionality نمی شه به آن رسید یعنی چه؟ یعنی صفات کیفی شما محدود می شود به Function هاتون، بعضی وقت ها یکسری عملکردهایی از سیستم می خواهد که اون عملکرد مانع این می شود که به یک Functionality خاصی برسیم یعنی مثلاً این طوری که اگر من بخواهم Availability بالایی داشته باشم، دسترسی پذیری بالایی داشته باشم خوب باید بحث های Recovery را داشته باشم ولی اگر که توی بحث هایی که مثلاً Functionality من مثلاً به گونه ای باشد که اجازه Recovery را به من ندهد یعنی همچین چیزی برایش فراهم نشده باشد من نمی تونم به اون صفت کیفی ام برسم.

یا حالا البته شاید خیلی مثالم واضح نبود. اجازه بدید جلوتر می بینید. بعد از طرف دیگه درسته که ما می گیم Functional ها و non Functional ها متعامد همدیگر هستند یعنی هر دو را با هم داریم و به آن می رسیم یعنی در طول هم هستند ولی بعضی وقتا ّFunctional اجازه نمی دهد به صفت کیفی خاصی برسیم خود صفات کیفی هم، با هم در تضادند (این یک بحث جدا است) و مطلب سوم این که function ها هم این جوری نیست که بگیم در معماری به این function هامون می رسیم بلکه بعضی از function ها رو در بحث معماری می بینیم و بهشون می رسیم و بعضی از function ها رو ما در سطح طراحی بهشون میرسیم و پیاده سازی ، یعنی اون جمع آخری که این جا داریم می بینیم گفته معماری یک وسیله ای برای رسیدن به صفت کیفی با استفاده از ساختاربندی کردنه function ها به درون المنت ها، این نتیجه گیری که دارد می کند پس اینجا معماری را یک ابزرای معرفی می کند که صفات کیفی ما رو برامون فراهم بکنه چه جوری با ساختاربندی کردن خاص function ها درون المنت های همون ساختارها.

حالا این بحث را بازش می کنم. ببینید در اسلاید بعدی این جا اگر نگاه کنید گفته صفات کیفی باید در سرتاسر Design ، پیاده سازی و توسعه سیستم و استقرار سیستم (تولید سیستم) در نظر گرفته شود. پس هم در فاز Design که معماری هم جزء آن است و هم در فاز پیاده سازی و هم در فاز استقرار باید صفات کیفی را ببینیم رسیدن به صفات کیفی در همه فازها است . مطلب بعدی که گفته این که Quality ها یا همان صفات کیفی ما هم جنبه های معماری دارند و هم جنبه های غیرمعماری. پس این هم باید کشفشون بکنیم که وقتی بحث از یک صفت کیفی می کنیم ، در سطح معماری مون به کدوم جنبه اون صفت کار داریم، چون این صفت جنبه غیرمعماری هم دارد برای مثال قابلیت تغییر که این جا مثال زده (modifiability) در مورد این که چگونه function ها تقسیم بشن تا بتونیم قابلیت تغییر را داشته باشیم این یک دید یا جنبه ی معماری گونه است اما درمورد این که چگونه تکنیک های کد نویسی را داشته باشیم درون یک ماژول تا بتونه قابلیت تغییر رو برای ما فراهم بکنه این یک دید غیرمعماری است. یعنی در سطح معماری به قابلیت تغییر یا modifiability می خواهید برسید وقتی از دید معماری می خواهید به اون صفات کیفی برسید بعد به همین ترتیب مثال زده

مثلاً usability جنبه ی غیرمعماریش چیه جنبه معماریش چیه؟

گفته جنبه غیرمعماریش شامل این که usability یک واسط کاربر واضح داشته باشیم و به سادگی بشود ازش استفاده کرد این که مثلاً چینش صفحه ات قابل فهم باشد این که نوع فونت هات و نحوه ی نوشتنت بزرگ باشه واضح و خوانا باشد این ها هم جنبه های غیرمعماری است یعنی شما در فاز طراحی یا در زمان کدنویسی این رو می توانید اعمالش کنید از این فاز کدنویسی و طراحی به usability (قابلیت دسترسی) می رسید ولی این که در بحث usability این که یک سیستم برای کاربرش قابلیت کنسل کردن عملیات را فراهم بکنه قابلیت undo کردن رو فراهم بکنه یا قابلیت استفاده مجدد از داده ای که قبلاً وارد کرده رو فراهم بکنه این جنبه معماری گونه قابلیت استفاده است (usability) .

ببینید شما یک سیستمی دارید که به شما اجازه undo کردن یا Redo کردن را می دهد یعنی در معماریش چنین چیزی دیده شده است. undo کردن یا Redo کردن خیلی کار سختیه یعنی در واقع قبلاً در بحث معماری دیده شده است.

که می شه فرض کنید خیلی ساده یک لاگی در یک سیستم مدام گرفته می شد در ازای یک check point های خاصی که اگر که ثبت می شه وضعیت سیستم که اگر undo کردید شما بتونید برگردید به آخرین check point که وجود داشته باشد یا این که اگر بخواهیم، undo کردن کار خیلی سختیه ، اگر که فرض کنیم یک تراکنش انجام دادی شما باید با undo کردن بتونی یک تراکنش رو کنسل کنی حتی یک تراکنشو که ما می گویم تراکنش فعالیتی است که اگر با موفقیت به اتمام برسد قابل تغییر نیست هیچ وقت اثرش از بین نمی رود یا وقتی Redo می کنیم مجدداً یک کاریو برمی گردونی یعنی این که کاری رو که شما کردین و کنسل کردیم با این که شما نخواستینش ولی یک جوری ثبتش کرده و نگه داشته، یا Reuse کردن دیتا یعنی این که داده های را که وارد کردیم یک تاریخچه از آن نگه داریم و حفظش کرده حالا می تونه دوباره اجازه استفاده را بدهد این ها همه مسائل قابل استفاده است. تسهیلاتی است که شما برای کاربر ایجاد می کنید.

این ها از جنبه ی چیست؟ از جنبه معماری، یعنی این ها در سطح معماری دیده شده است که این کارها را می شود انجام داد. ولی این که چینش صفحه چه قدر قشنگ باشه و فونت هاش چگونه باشه این ها در فاز طراحی است یا پیاده سازی بعد این برای usability است

برای performanceمثال زده گفته ببینید این که چه قدر ارتباط نیاز هست بین component های مختلف بحث performance راجع به هزینه و زمان صحبت می کند در واقع performance در مورد زمان که داره صحبت می کند

مثلاً شما دربحث های سیستم عامل در performance می بینید کلی روش زمانبندی وجود دارد و برای این که توی درس سیستم عامل برای این که بتونید performance استفاده از CPU را بالا ببرید مثلاً – یا performance استفاده از حافظه اصلی را ببرید بالا.

پس performance در مورد زمان صحبت می کند که من بتونم event یا یک رویداد که حالا تعریفش رو تو همین فصل مرور می کنیم وارد سیستم که می شود من بتونم بهش در حداقل زمان ممکن پاسخ بدهم.

حالا ببینیم جنبه های معماری performance چیست و جنبه های غیر معماری آن چیست؟ پس این که می گوید چه قدر ارتباطات نیاز هست بدون مؤلفه ها ارتباط باید وجود داشته باشد این معماری ، این که چه function هایی انتصاب پیدا کرده، یا allocate شده به هر مؤلفه به چه میزان است ،این Functionality های معماری گونه اش است.

و این که منابع چگونه به اشتراک گذاشته شدند بین مولفه های مختلف این بار یک بحث معماریه چون اگر بحث اشتراک از منابع وجود داشته باشد پس انحصار متقابل در کنارش می آید ،انحصار متقابل هم وقتی میاد در زمانبندی شما تأثیر می گذارد. و نهایتاً روی performance تأثیر می گذارد. پس این خودش روی performance تأثیر می گذارد. این ها جنبه معماری است.

حالا چه چیز performance معماری نیست؟ انتخاب الگوریتم برای پیاده سازی functionality انتخاب شده، معماری نیست این که چگونه این الگوریتم ها را کدنویسی می کنیم هم معماری نیست. یادتون هست من برای performance مثال می زدم من این ماژول را دارم این ماژول می تواند برای من مرتب سازی انجام بدهد در مدت زمان فلان یا پیچیدگی (n) یا order(n) خوب یعنی من فقط می گویم این چه Functionality انجام بدهد درسته ؟

این قسمت معماریه ولی این که بگویم از چه الگوریتم مرتب سازی استفاده بکن تا بتونی performance را فراهم بکنی این می شود جنبه غیرمعماری ، این که این الگوریتم با چه زبانی پیاده سازی بکن که این حلقه For آن را از چه روشی بره که فرض کنید که حداقل تعداد تکرار رو داشته باشد یا حداقل زمان را بگیرد این ها می شود جنبه ی غیرمعماری.

خوب پس در کل الان میگه توی پیامی که توی این بخش می دهد دو جنبه است. دو تا پیغام می دهد یکی این که معماری بحرانیه برای محقق کردن بسیاری از صفاتی کیفی که سیستم می خواهد و این صفات کیفی باید طراحی باشند و ارزیابی شوند در سطح معماری- یکی از نتایجی که از صحبتامون به دست می یاد و نکته ای دومش اینه که معماری، خودی خود نمی تواند به ثبات کیفی برسد. بلکه اون یک فوندانسیون یا پایه ای را فراهم می کند. برای دستیابی به این صفات که این پایه هم محقق نمی شه. اگر با ریز جزئیات بهش توجه نشه.

پس معماری در واقع یک فوندانسیون فراهم می کند که بتوانیم ثبات کیفی را بهش برسیم. این جا اومده صفات کیفی رو که یکسری نکته است را بیان کرده.

معماری و کیفیت ها... صفات کیفی مستقل از همدیگه نیستن. و به تنهایی قابل دسترسی نیستند. یعنی به صورت ایزوله قابل دسترسی نیستند. بلکه صفات کیفی که ما داریم. یک وابستگی مثبت بعضیاشون دارن. بعضیاشونم با هم وابستگی منفی دارن. یعنی یا اثر مثبت رو همدیگه دارن یا منفی.

مثبت مثلاً modifiability و Buildability این ها با هم اثر مثبت دارند یعنی ما هر چه بیش تر با modifiability برسیم به Buildability هم می رسیم یا هر چه بیش تر Buildability فراهم کنیم بیش تر modifiability فراهم می کنیم ببین بعضی از کتاب ها قابلیت ساخت خودشو جزئی از modifiability می بینند. بعضی از کتاب ها هم نه اون بو به صورت جداگانه اصلاً معرفی می کنند

به عنوان یک صفت کیفی غیر اصلی یکمی جلوتر اسلاید بعدیش فکر کنیم دسته بندی از صفاتش کیفی سیستم رو داریم Buidability هم جزء صفات کیفی سیستم هست ولی اصلی هاش نیستند در واقع فرعی اند قابلیت ساخت یعنی این که سیستم خوش ساخت باشد یعنی شما توی معماری ماژول هایی رو تعریف کردید که این ماژول ها را به راحتی و مستقل از هم دیگر و حتی با زبان های مختلف برنامه نویسی می شه ایجادشون کرد. این می شود قابلیت ساخت modifiability که دقیقاً این رو میگه، تا اونجایی که می تونید از هم دیگه مستقل تعریفشون بکنید تا هر وقت خواستید بتونید

این ماژول ها رو جدا کنید عوض کنید تغییر بدید یا اضافه کنید پس رسیدن به modifiability ، Buidability را هم برامون فراهم می کند. اینو بهش می گن همبستگی مثبت (positive correlation)

اما همبستگی منفی یا (conflict)یعنی چی (در تضادند) یعنی رسیدن به یکیشون باعث فدا شدن یکی دیگر می شود مثلاً قابلیت اطمینان متضاد امنیت است یعنی هر چی می خواین Reliability رو بالا ببری Security می آید پایین هر چی بخوای (Availability Reliability) را بالا ببری security را می آورد پایین

Reliability تعریف اسمیش این که در یک پریود منظمی از زمان به سیستم که مراجعه می کنید x% درست کار کند. اما این دقیقاً مقابل security است. یعنی شما می خواهید همیشه درست کارکردن سیستم رو داشته باشید اما از طرفی دیگر می خواهیم که اگر ،کاربری هست که Authenticate نشده یعنی تعیین هویت نشده و یا Authorization نشده یعنی حدس می زنه که سیستم کاربر قانونی نیست، و سیستم پاسخ درست ندهد یعنی برای داشتن امنیت مجبوریم مواردی را کنترل نماییم که ممکن است در بعضی مواقع کاربر قانونی را غیرقانونی بشناسد و جواب درست را نمی دهد.امنیت Reliability را پایین می آورد.

یا مثلاً امنیت با performance مخالف و با تمام صفات کیفی دیگر در تضاد است. یعنی کنترل همون شخص که ببینید قانونی است یا نه و این که سطح دسترسیش چه قدره، زمان میبرد، کار اضافه غیر از functionality است توی سیستم پس این performance رو آورده پایین پس صفات کیفی با هم دیگر در تضاد هستند. خود Availability می گوید Backup داشته باشیم و وجود Buckup خودش security را پایین می آورد. یعنی دوباره باید ماژول کنترلی بزاریم روی اون نسخه Backup که وجود دارد.

ما سه دسته کلی صفات کیفی داریم،

صفات کیفی سیستم که شش تا اصلی داره و قابلیت دسترسی قابلیت تغییر، کارایی، امنیت، قابلیت تست و قابلیت استفاده

یک سری صفات کیفی هم داریم صفات کیفی کسب و کار ماست. که روی معماری اثر می گذارند مثل زمان تحویل به بازار - یعنی بیزینس مستقیم مربوط می شوند به موارد بازار-

سوم صفات کیفی معماری: که خود معماری نشان می دهد . در واقع معیارهای ارزیابی معماری را نشان می دهد. یعنی این که توی چند تا پارامتر معرفی می کنه که براساس آن نشان می دهد که یک معماری کیفیت دارد. ۲ یا ۳ اسلاید برای Business ها داریم. یکی دو تا هم برای Architecture ها داریم بقیه همه صفات کیفی سیستم خواهد بود.

صفات کیفی سیستم: system Quality Attributes

از سال ۱۹۷۰ بحث بوده سر این که این ها چی ها هستند چه جورین و دسته بندی آن چگونه است. یکسری تعاریف مختلفی تا حالا منتشر شده در مورد صفات کیفی سیستم اما توی این کتاب، از دید یک معمار سه مشکل اصلی روی تعریف هایی که برای صفات کیفی بیان شده است وجود دارد. پس در این جا یک تعریفی را ارائه می دهیم که همه این مشکلات را حل نماید. که این روش تعریف صفات کیفی را اسمش رو می ذاریم سناریو، حالا ببینیم مشکلات چی ها بوده اند.

۱- تعریف فراهم شده برای صفات کیفی operational نیستند. (عملیاتی نیستند)

عملیاتی نیستند یعنی چی؟ با مثال بیان کرده است گفته شما وقتی می گویید یک سیستم قابلیت تغییر دارد این مفهوم واضحی ندارد. چه قدر قابلیت تغییر دارد یعنی چی قابلیت تغییر دارد؟ یعنی به صورت عملیاتی بیان نشده است این که درصد برایش مشخص نشده است یعنی می توانم بگویم تغییرات local داشته باشد یا non local داشته باشد در کجاها local و در کجاها non local هستند و این تغییرات در زمانی باید اتفاق بیافتند و مطابق با اون چه کسی می تواند انجام دهد آن را؟ مثلاً تغییر در فاز طراحی باید باشد یا در فاز پیاده سازی یا حتی در زمان Runtime یا اجرا است.

آیا کاربر نهایی End user هم می تواند تغییر دهد یا خیر؟ و صرفاً گفتن این که یک سیستم قابلیت تغییر داشته باشد Operational نیست.

مثل این که بگوییم این آدم خیلی آدم خوبیه. یعنی چی آدم خوبیه؟ در چه زمینه خوبه؟ کار؟ تخصص ؟ روابط دوستی؟

مشکل دوم این که تمرکز این بحث ها روی این که اون صفت کیفی به یک جنبه خاصی تعلق دارد. مثلاً این که اگر گفته یک failure روش رخ بدهد آیا یک جنبه availably است؟ یا یک جنبه security یا یک جنبه usability است؟ یعنی این که این ها را فقط روی یک جنبه نگاه می کنند. همه را با هم هیچ وقت ندیده، اثر همه را با هم ندیده- یعنی وقتی falt رخ می دهد. falt یعنی شکست.

یک falt یا یک نقص وقتی رخ می دهد. بررسی نشه و بهش رسیدگی نشود تبدیل می شود به failure. حالا اگر یک failure رخ می دهد آیا فقط Availability را تحت تأثیر قرار می دهد؟ یا نه بر روی security هم اثر می گذارد؟ یا usability هم اثر می گذارد همه این ها failure را برای خودشان می بینند.

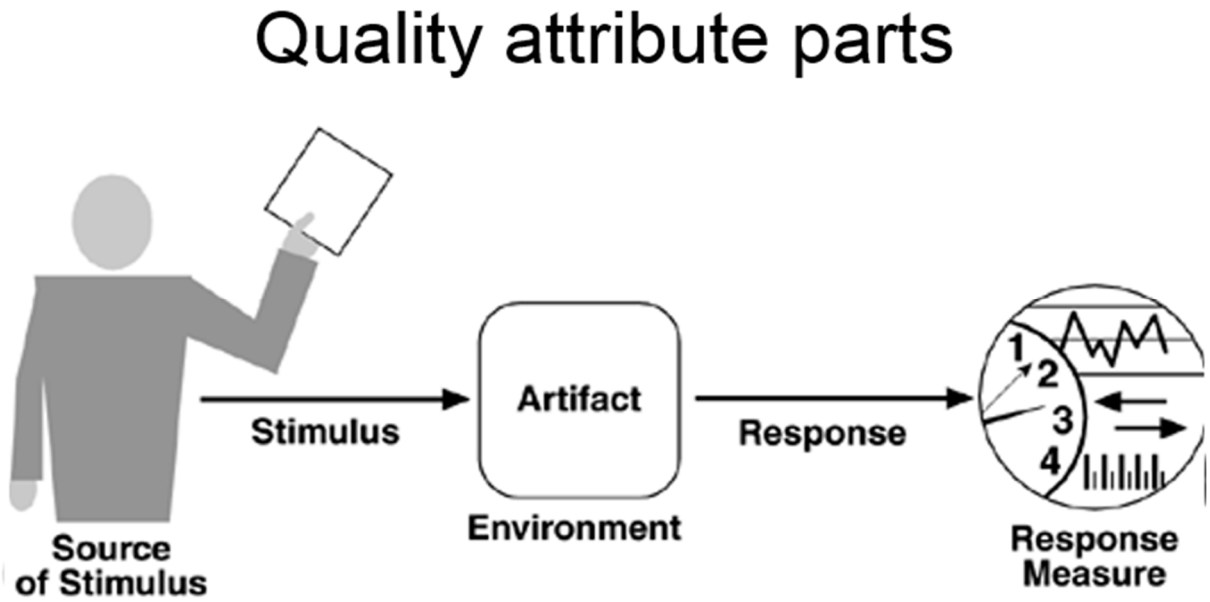
روشی که می خواهیم صفت کیفی را تعریف کنیم باید طوری باشد که توی اون قشنگ درک بشه که این واقعه یا failure که رخ داده است چه صفات دیگری هم در آن درگیر هست.

مشکل سوم: هر اجتماعی از این صفات کیفی vocabulary خاص خود شدن رو دارند یعنی فرهنگ لغات خودشون رو دارند. مثلاً می گویند در performance ها event تعریف می کنیم performance ها همیشه با events ها سر و کار دارد، security با Attack ها سر و کار دارد. Availability با failure ها سر و کار دارد و usability هم با user Input سر و کار دارد در حالی که همه این ها دارند به یک موجودیت اشاره می کنند.

یعنی مثلاً ببینید یک شخصی یک دستوری را وارد می کند. از دید قابلیت استفاده این یک ورودی از طرف کاربر (user Input) ، از دید Availability این درخواست ممکن است منجر به failure بشود. از دید security ممکن است یک حمله باشد و از دید performance این یک event است که باید بهش پاسخ داده شود. یعنی در واقع هم روی یک موجودیت کار می کنند. اما هر کدام برای خودشان یک اسمی بر روی آن گذاشته اند. پس یعنی همه آن ها به صورت واقعی اشاره می کنند به یک رخداد مشترک اما توصیف می شود در مقولات متفاوت.

خوب باید یک مترادف سازی در vocabulary ها ایجاد شود. که بگه این Attack معادل با همون event است حال چگونه این مشکلات را رفع نمایم؟ راه حل دو مشکل اول این که عملیاتی نیستند و این نگرانی که این صفات overlap دارند یعنی در حوزه های مختلف هم پوشانی دارند این که ما سناریوهای صفات کیفی رو به عنوان ابزاری برای مشخصه بندی کردن صفات کیفی معرفی کنیم. پس سناریوها از این جهت و برای این دو مشکل اول، می آید و سناریو تعریف می کند که حالا جلوتر می گویم که سناریو چی هست؟ اما راه حل مشکل سوم چی هست بیاییم یک توصیف مختصری از هر صفت فراهم بکنیم با توجه به نگرانی هایی که اون صفت ایجاد می کند و اون توصیف رو به عنوان یک پایه برای کل صفات کیفی استفاده کنیم. یعنی یک فرهنگ لغات مشترک بسازیم و مترادف ها را بنویسیم. پس برای مشکلات اول و دوم قرار است به ازای هر صفت کیفی سناروییی تعریف کنیم تا تعریفش operated شود و دیگر اینکه حوزه ی کاریش به طور واضح مشخص شود.

تعریف سناریوهای صفات کیفی: یک سناریوی صفت کیفی یک نیازمندی خاص صفت کیفی است که شامل ۶ قسمت است. ۶ پارت دارد به صورت گرافیکی یک صفت کیفی رو به این شکل معرفی می کنیم.



سناریو در واقع داستانیه که در بحث های فیلم نامه نویسی اولین بار به گوشتون خورده. روالی است که تعریف می شود که یک ماجرا رو تعریف می کنه با ریز جزئیات. تمام شرایط محیطی نور رو تعریف می کنه. محل قرار گیری بازیگران، صحنه رو مشخص می کنه. فعالیت یا رویدادی را هم که باید انجام شود را نیز توصیف می کند و صحبت هایی که باید رد و بدل شود، این می شود سناریو. مثلاً در بحث های شبکه یا شبیه سازی می گویند که من می خوام n دونه node هر کدام در یک نقطه مشخصی چیده شده باشد بعد با یک الگوی خاصی حرکت بکند و بعد این قدر packet در ثانیه ارسال شود. هر کدام n تا فایل داشته باشند یعنی یک داستان رو تعریف می کنیم برای شبکه توی یک مدت زمان معین. این جا هم همین جوریه. این جا هم یک داستانی را برای یک سیستم نرم افزاری تعریف می کنیم اما از یک جنبه خاص. از جنبه performance یا modifiability یا usability و یا ... که چند تا قسمت توی اون باید تعریف بکنیم. اول باید بگویم یک اتفاقی داره می اُفته. یک تحریک برای سیستم در حال وقوع است. اسم این تحریک stimulus است.

یک نفری یک تحریکی را انجام می دهد. پس یک منبع تحریک یک تحریک را انجام می دهد. سیستم وقتی تحریک می شود یک شرایط محیطی دارد و یک شرایط خاص سیستم دارد. یک Environment دارد، یک محیطی داره سیستم. تحریک وقتی وارد سیستم می شود. این تحریک بر یک جایی از سیستم اثر می گذارد. که به آن Artifact می گویند Artifact می شود محصول مصنوعی تحریک، پس تحریک یک Artifact تولید می کند در اون سیستم، شما به عنوان معمار قراره به اون تحریک پاسخ بدهید و این پاسخ رو باید به گونه ای نمایش بدین. پس خود پاسخ می شود Response و نحوه ی نمایش پاسخ می شود Response Measure، پس بنابراین ۶ قسمت داریم برای سناریو.

اولین قسمت، منبع تحریک: یک موجودیت می باشد که این Entity می تونه یک فرد باشد می تونه یک سیستم کامپیوتری باشد. یا می تونه هر بازیگر دیگری باشد که تحریک رو تولید می کند.
قسمت دوم stimulus است یک شرایطی است که باید در نظر گرفته شود وقتی که وارد سیستم می شود. پس یک condition است یک موقعیت یا وضعیتی است که ایجاد می شود. که در این زمان باید سیستم بررسی شود. تحریکی که رخ می دهد.
سومیش محیط Environment: تحریک در یک شرایط خاصی رخ می دهد که سیستم می تونه درشرایط over Load باشه یا در زمان اجرا باشد یا می تواند هر شرایط دیگری داشته باشد. در واقع شرایطی که سیستم در آن قرار دارد و تحریک رخ می دهد را به آن می گویند Environment. مثلاً در حالت فعال پرکار، overlade، معمولی Normal یا در حالت degraded – یعنی با کم ترین امکاناتش سیستم بالا امده است. مثلاً safe mode که برای سیستم عامل وجود داره یا هر شرایط دیگه ای این می شود Artifact
Environment – چی؟
قسمت چهارم: Artifact برخی از اثرات در واقع تحریک شدن که می تونه کل سیستم باشه یا قسمتی از اون، یعنی اون جاهایی که تحریک روی اون اثر می گذارد. ممکنه همه سیستم باشد یا ممکن است بخشی از سیستم باشد. [Artifact محصول تحریک است وقتی تحریک انجام می شود چه تغییری را ایجاد می کند و کجای سیستم خراب می شود.]
بخش پنجم Response: پاسخ فعالیتی است که باید در نظر گرفته شود بعد از رسیدن stimulus (تحریک). یعنی تحریک=
[مثلاً اگر یک جای امنیتی باشد تحریک یک حمله است، حمله رخ داد. اثر این حمله می تواند یک فایلی فاش شده خونده شده- دستکاری شده این فایل خونده شده می شود Artifact اون حمله می شود stimulus– شخصی که حمله را انجام داده- می شود منبع تحریک و اون زمانی که سیستم حمله براش رخ داده که نرمال کار خود را انجام می داده، می شود environment حالا سیستم باید پاسخ بدهد، پاسخی که سیستم می دهد (فعالیتی که در مقابل تحریک انجام می دهد)را به آن میگوییم Response
حالا Response measure یعنی چه؟ هنگامی که پاسخ صادر می شود باید این پاسخ قابل اندازه گیری باشد- به شیوه های مختلفی که نیاز داریم بتونیم مشخص کنیم. این رو بهش می گن Response- Measure– یعنی پاسخ خود را باید به یک گونه ای نمایش بدید مثلاً باید فرض کنید همون حمله رو و من باید پیغام بدم شما یک کاربرغیر مجاز هستید- شما امکان مشاهده این فایل را ندارید این پیغام می شود یک measure چون اگر نتوانید پیغام را نمایش بدهید واضح نیست. باید تمام جنبه های اون سناریوتون واضح باشد، بعضی از stimulus ها یک artifact دارند و بعضی ها هم امکان دارد چند Artifact داشته باشد.
درواقع روش های سناریونویسی را یاد می گیرید و سپس می توانید برای سیستم های دیگر شرایط خاص را بنویسید.
در کل ما دو نوع ساریو داریم یکسری سناریوهای جنرال general و یک سری سناریوهای concrete، سناریوهای جنرال در حالت کلی یعنی همه ی. یعنی برای صفات کیفی خاص تمامی منبع های تحریکی که می تواند وجود داشته باشد. تمام محصولاتی که می تواند داشته باشد تمام شرایطی که سیستم می تواند داشته باشد تمام پاسخ هایی که توانید بدهید. تمام measure هایی که برای پاسخ ها می توانید بدهید.
سناریوی کلی- سناریوی concrete چیه؟ سناریوی concrete برای یک سیستم خاصه و یک شرایط خاص یعنی مثاله. مثال برای جنرال بخواهیم بزنیم اسمشو می ذاریم concrete سناریو. حالا برای نوشتن سناریو باید جنرال سناریو یاد بگیریم و بعد برای این که بفهمیم و بتوانیم استفاده کنیم مثال خاص می زنیم تا خودمون بتونیم اون مدلی بتونیم بنویسیم. یعنی از این جا به بعد ابتدا صفت کیفی را می گویم. جنرال سناریو شو بیان میکنم . سپس concrete سناریو بیان می شود.
Quality Attribute Scenario in practice
برای 6 صفت هایی که مهمه می آییم و سناریوها را معرفی کنیم.
۲ هدف
۱- مشخص کردن مفاهیم استفاده شده برای اجتماع این صفت ها
۲- روشی برای فراهم کردن سناریوهای جنرال را یاد می گیریم

Availability: قابلیت دسترسی- نگرانی آن در مورد failure ها و عواقب آن می باشد یک failure سیستم رخ می دهد هنگامی که سیستم یک مدت طولانی به سرویس های هماهنگش پاسخ نمی دهد. با توجه به سرویس هایی که برای آن مشخص شده پاسخ نمی دهد اگر برای یک مدت زمانی به سرویس ها با آن موردی که باید پاسخ بدهد پاسخ ندهد می گوییم سیستم دچار failure شده است. یک تفاوت باید قائل بشویم بین fault و failure. وقتی یک fault رخ می دهد دچار failure می شویم یعنی وقتی که fault رخ بدهد و اون fault , correct نشود اصلاح نشد یا masked نشود سیستم را دچار failure می کند. مثال تقسیم بر صفر یک fault است نتیجش بستن کل برنامه است خارج شدن از برنامه است که failure را نشان می دهد. یا ورودی اشتباه وارد شده بعد نمی شه روش محاسبه انجام بشه این یک fault است. و این دچار پاسخ ندادن به یک مؤلفه است این می شود failure.

Fault در واقع ریشه ایجاد failure است. تا fault ی نباشد failure ی نیست و نکته دیگر این که یک failure قابل مشاهده است توسط کاربر سیستم اما fault قابل مشاهده نیست. یعنی شما وقتی مشکلی در سیستم ببینید دسترسی پذیری را برایتان کم کرده یا از بین برده در واقع دارید failure را می بینید اما fault رو نمی بینید. شاید fault , correct mask شده باشد. Fault توسط end user دیده نیمی شود ولی مدیر سیستم آن را می بیند.

منبع تحریک (source of stimulus) منبع تحریک یا داخلی اند یا خارجی هستند که failure یا fault را نمایش می دهد.

در مثال ما یک پیام ناشناخته ای از یک سیستم خارجی وارد سیستم شده است پس منبع تحریکش یک سیستم دیگر بوده است پس بنابراین external می شود. و پس تحریک برای Availability می شود یک fault.

Fault ها کلاً چهار دسته اند، یک دسته omission یعنی چی؟ وقتی یک component ، fail می کنه در پاسخ دادن به یک ورودی، پس fault ی از دسته omission می شود.ورودی می آید ولی پاسخی به آن داده نمی شود این ر می گویند fault هایی از جنس omission

بعد اگر این که این fault (omission) مدام تکرار بشه اسمش می شود crash پس crash می شود وقتی که مؤلفه ای مدام متحمل خطای omission شود می شودcrash .

timing یعنی چی؟ یعنی یک مؤلفه ای پاسخ بدهد خیلی زود یا خیلی دیر- بعضی وقتا یک دستوری را اجرا می کنید و سیستم خیلی زود جواب می دهد، مثلاً زود فایلی را attach کند احساس می کنید خطایی رخ داده است در واقع باید سر وقت انجام شود. یا وقتی که خیلی دیر انجام شود. پس این هم خطای timing می شود. اما خطای Response یک مؤلفه ای پاسخ می دهد جواب اشتباه را. یعنی جواب اشتباه می دهد و اگر جواب اشتباه بدهد می شود Response اگر جواب ندهد می شود omission اگر تعداد جواب ندادن ها شد تکرار شود می شود crash اگر خیلی دیر یا خیلی زود جواب بدهد می شود timing

Artifact: خطایی که رخ می دهد روی چی اثر می گذارد. گفته این Artifact مشخص می کند منابعی که مورد نیازند به شدت Available باشند دسترس پذیر باشند مثل یک processor مثل مسیر ارتباطی مثلاً یک فرآیند یا مخزن ذخیره سازی کمی unavailable می شود. کسی دیگر استرس پذیر نیست؟ پس اثر stimulus کجا است؟ شرایط یا Environment چی هست؟ وضعیت سیستم وقتی که fault یا failure رخ می دهد. می تواند اثر بذارد روی پاسخ مطلوب سیستم برای مثال اگر سیستم درحال حاضر برخی از fault ها را تحمل بکند و اون درحال عملکردن در مد نرمال باشد فرق دارد تا این که اون در حال دیگری باشد. مطلوب این گونه است که سیستم را shot down بکند. بستگی دارد که سیستم چه شرایطی داشته باشد کلاً در بحث های Availability یا توی حالت نرمال سیستم که fault برایش رخ می دهد یا توی حالت degraded هست یا حالت overload خیلی سرش شلوغه- degraded با حداقل منابع راه افتاده نرمال هم که نرماله

پاسخ ها حالا چی هستند؟ پاسخ هایی که ما می دهیم به fault یا failure در واقع به آن stimulus ی که داریم یک تعدادی از عکس العمل های ممکن برای این که سیستم می دهد به failure ها به این ها می شود Response های ما که مثلاً چیه؟ شامل این مواردی می تواند باشد لاگ بکنیم failure را فقط ثبت کنیم.

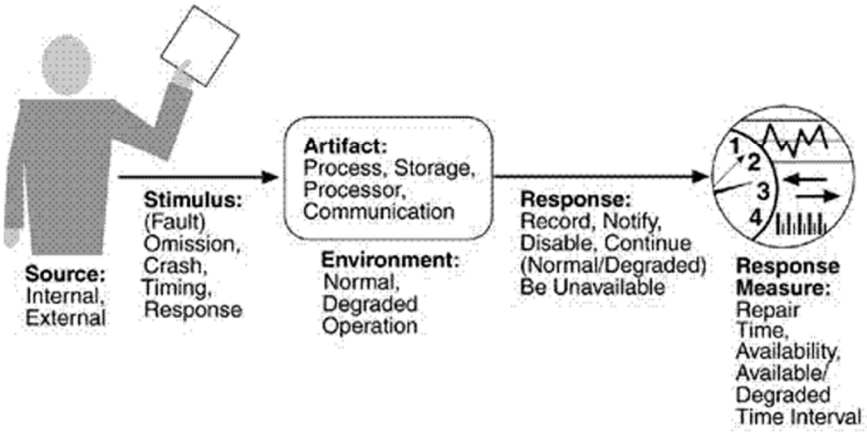
Fault یعنی خطا failure یعنی شکست، error هم تازه داریم.

Error هم می شود اخطار، خوب پس کارهایی که ما انجام می دهیم یا این که فقط ثبت می کنیم یا این که اطلاع می دهیم به یک کاربر خاصی، این failure را یا به سیستم های دیگر این اطلاع را می دهیم یا این که سوئیچ می کنیم به حالت کاهش یافته به مُد کاهش یافته برای این که بتونیم با حداقل ظرفیت به کارمون ادامه بدهیم یا این که او سیستم خارجیو که stimulus را به وجود آورده خاموش می کنیم. یا این که در طول تعمیرمون unavailable می شویم. یعنی غیرقابل دسترس می شویم این ها همه پاسخ های ممکن است که بکار برده می شود.

حالا چگونه این پاسخ ها را نمایش بدهیم. مثلاً در حدی availability را نمایش دهیم یا می توانیم زمانی را برای reapaire شدن در نظر بگیریم مثلاً یک statusbar می آوریم و نشان می دهد که یک کاری را انجام می دهد. و measure دیگری که داریم این که زمانی که سیستم باید Available باشد رو نشون بدیم. یا این که مدت زمان Available را نشان دهیم.

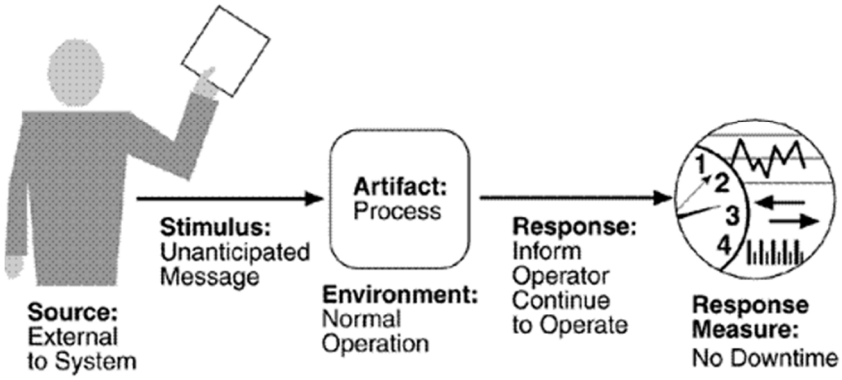
مدت زمانی که باید Available باشد و چه زمان هایی سیستم باید Available باشد. مثلاً برای سیستم ثبت نام باید حتماً در پایان ترم و شروع ترم بعدی سیستم Available باشد. این زمان هاست ولی این که چه مدت زمانی سیستم Available باشد مثلاً سیستم می تواند ۵ ساعت را Available باشد این منظورش هست میزان پاسخ مبنا است (داشبورد ماشین) ماشین یک سیستم است که به تمام کارهایی که داخلش رخ می دهد چه ناشی از کاری است که ما انجام دادیم چه کارهای داخلی خودش پاسخ می دهد و این پاسخ را بگونه ای برای شما نمایش می دهد. چگونه نمایش می دهد از همان داشبورد می توانید ببینید. حالا وقتی Response می دهیم به یک failure باید به یک روشی اعلام بکنم مثلاً این قدر زمان می برد تا ریکاور شود – این قدر زمان طول می کشد تا unavailable بمونم- در این زمان ها حتماً available هستم. پس این ها همه می شوند Response measure

Example: Availability General Scenario



منبع تحریک داخلی یا خارجی است stimulus یا fault

Example: Availability Concrete Scenario



مثال خاص concrete : از خارج سیستم یک درخواست غیر منتظره وارد سیستم شده است. این پیغام وقتی وارد شده باشد در حالت normal بوده- این پیغام برروی فرآیندهای سیستم اثر گذاشته. پاسخی که ما دادیم اطلاع دادیم به operator که به کار خودت ادامه بده و به این پیغام توجهی نکن. چه جوری نشون دادیم و گفتیم بر اثر این failure ما هیچ downtime نداریم. و بعداً می توانیم unanticipated را هم جزء fault های سیستم بذاریم.

Table 4.1. Availability General Scenario Generation	
Portion of Scenario	Possible Values
Source	Internal to the system; external to the system
Stimulus	Fault: omission, crash, timing, response
Artifact	System's processors, communication channels, persistent storage, processes
Environment	Normal operation; degraded mode (i.e., fewer features, a fall back solution)
Response	System should detect event and do one or more of the following: record it notify appropriate parties, including the user and other systems disable sources of events that cause fault or failure according to defined rules be unavailable for a prespecified interval, where interval depends on criticality of system continue to operate in normal or degraded mode
Response Measure	Time interval when the system must be available Availability time Time interval in which system can be in degraded mode Repair time

چه طوری سؤال می آید از این صفات کیفی؟ شما باید این جدول را یاد داشته باشید تا بتوانید از روی آن مثال بنویسید یکی دو تا سه تا، که اگر یک سیستم نوعی را براتون معرفی کردیم نگیم برای این سیستم چند تا سناریوی Availability یک سناریوی modifiability و یک ... بنویسیم ، بتوانیم از روی آن سناریو را بنویسیم- باید بلد باشید که concrete سناریو بنویسید- چه جوری؟ با توجه به این جدولی که از جنرال سناریوها می آید می توانید concerte سناریو را بسازید.

صفت کیفی modifiability یا قابلیت تغییر در مورد هزینه تغییر صحبت می کند و که دو جنبه خودش داره ، یکی این که چه چیزی تغییر می کند که همان Artifact می باشد و یک تغییر می تواند رخ بدهد تو هر جنبه از سیستم که معمول ترش function های محاسباتی سیستم، یا platform که سیستم بر روی آن وجود داره (سخت افزار، میان افزار، نرم افزار) یا این که محیطی که بر روی آن عمل می کند یعنی سیستم هایی که با هم interoperate دارند همکاری می کنند.

یا پروتکل های ارتباطی که برای برقراری ارتباط با بقیه قسمت ها نیاز دارد تغییر بکنند. پس یکی از جنبه های مهم یک پیش درآمد دار می گه قبل از این که جنرال سناریو بگوید که منظور از تغییر یعنی چی؟ تغییر در مورد هزینه تغییر داره صحبت می کنه و ۲ تا نگرانی دارد اول این که چه چیزی داره تغییر می کنه و یک functional داره تغییر می کند یا platform باید عوض بشه پس سناریوها بستگی به این دارد که چه چیزی باید تغییر کند و دوم این که چه کسی می خواهد تغییر را انجام بدهد.

یا چه موقعی تغییر می خواهد انجام شود یا چه کسی می خواهد تغییر را انجام دهد هر روش یک مفهوم دارد. اغلب در گذشته این گونه رایج بوده است- تغییر فقط روی source که می توانسته رخ دهد- کسی تغییر را انجام می دهد در چه زمانی- در فاز پیاده سازیDevelop،در زمان طراحی Designer، در زمان اجرا enduser← انجام می دهد

مسئله دارد وقتی یک تغییری را انجام می دهیم در هر زمان و توسط هر کسی اول باید تغییر انجام شود بعد test شود بعد Deploy شود. یعنی اول که تغییر کند بعد تست کنیم بعد استقرار دهیم سپس هر تغییری ۳ مرحله دارد ۱ انجام تغییر ۲ ثبت تغییر و ۳ Deploy کردنش

این که می گوییم تغییر زمان بره – زمانی است که ۳ تا کار انجام می شود. قابلیت تغییر یعنی این مراحل را بتونیم این ۳ مرحله رو انجام بدهم و با حداقل زمان بتوانیم این کار رو انجام بدیم (زمان و هزینه) و گرنه زمان و هزینه اگر زیاد باشد اصلاً شاید از انجام تغییر صرف نظر کنیم.

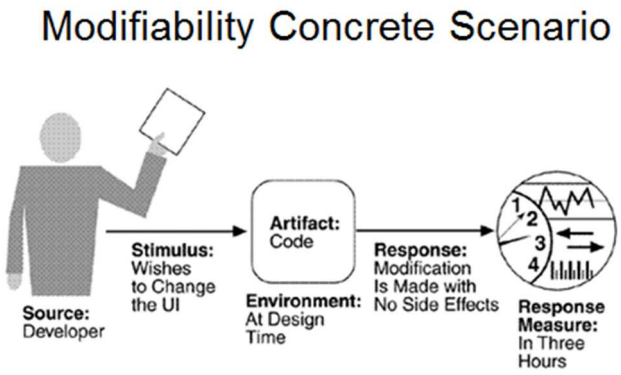
چه زمانی یک تغییر می تواند انجام شود. معادل با این که چه کسی می تواند آن تغییر رو انجام دهد یعنی هر جاش در مورد یک چیزی داره صحبت می کند.و بعدیش گفته که تغییر توسط مدیر سیستم Develop یا end user می تواند انجام شود.

مثال: یک developer – می خواهد یک تغییر بدهد رابط کاربر رو که این تغییر می تواند در زمان طراحی و حداقل ۳ ساعت طول می کشه تغییر انجام شود تست بشه و هیچ اثری هم رفتار سیستم نخواهد گذاشت، منبع تحریک (Developer) است، چی را می خواهد تغییر بدهد، Artifact چیه؟ (stimulus – (user Interface) تغییر رابط کاربری، چی را تغییر می دهیم کد را تغییر می دهیم سیستم در چه Environ است (زمان طراحی) پاسخ به چه صورتی است؟ پاسخ به صورت: بدون هیچ اثر جانبی تغییر را انجام می دهیم. Measure به چه صورت است؟ در کمتر از ۳ ساعت این تغییر انجام شد یا به این صورت که پاسخ این که تغییر انجام می شود چه طوری نمایش می دهیم بیان می کنیم که کمتر از ۳ ساعت بدون هیچ اثر جانبی این تغییر رو انجام بدم.

نیازمندیها را به دو دسته تقسیم می کنیم .

نیازمندیهای functional و نیازمندیهای nonfunctional .

نیازمندیهای functional کارهای اصلی که سیستم باید انجام بدهد را شامل می شود (نیازمندیهای کاربر) اما گفته شده که غیر از آن یک سری نیازمندیهای دیگر برای سیستم کامپیوتری وجود دارد که مستقیما جزء نیازمندیهای اصلی نیست اما روی آنها تاثیر گذار است که به آنها صفات کیفی گفته می شود.



صفات کیفی به سه دسته تقسیم می شود . ۱- نیازمندیهای کیفی سیستم، ۲- نیازمندیهای کیفی بازار ، ۳- نیازمندیهای کیفی معماری

نیازمندیهای کیفی سیستم به چند دسته تقسیم شدند . ۱- Modifiability ، ۲- Use ability، ۳- Performance ، ۴- Test ability ،

۵- Security ، ۶- Availability

این نیازمندیهی بعنوان نیازمندیهای اصلی سیستم هستند. که از بین آنهایک روش معرفی شده که همانطوریکه نیازمندیهای کاربر که برای آنها use case رسم میشد.برای معماری نرم افزار جهت نیازمندیهای کیفی نیز سناریو نوشته می شود . یک سناریو از ۶ قسمت تشکیل شده بود طبق شکل همیشه یک سناریو از یک محرک شروع می شود که به آن Source گفته می شود. یعنی منبع تحریک که می تواند یک انسان باشد می تواند یک سیستم کامپیوتری باشد که یک تحریکی را انجام می دهد که به آن Stimulus گفته می شود.

تحریک یک وضعیتی است که برای یک سیستم شرایطی را ایجاد می کند، این تحریک روی یک قسمتی از سیستم اثر می گذارد که به آن Artifact گفته می شود ، و وقتیکه به سیستم تحریک وارد می شود در یک موقعیتی قرار می گیرد که به آن موقعیت Environment گفته می شود ما باید به آن تحریک یا اثر پاسخی بدهیم که به آن Response گفته می شود که این پاسخ باید به نحوی نمایش داده شود که به آن response measure گفته می شود که این ۶ قسمت یک سناریویی را برایمان معرفی می کند. که این سناریو صفت کیفی مد نظر ما چه چیزی است و به چه اندازه می باشد بیان می کند. صفت کیفی بیان شده به همراه Response measure ، تبدیل به یک کمی تا بصورت عدد برایمان نمایش دهد.

سناریوها به دو دسته تقسیم می شوند ، یک دسته سناریوهای حالت کلی داریم. حالت کلی سیستم بیان می شود بعنی از این ۶ گزنه موجود تمام حالتهایی که برای ۶ تا گزینه وجود داشته باشد را بیان می کند که با چینش های مختلفی روبرو هستیم که با مقدار دهی می توانیم به یک حالت ویژه برسیم.

مثلا در اسلاید مورد نظر Modifiability Concrete Senario یعنی سناریوی خاصی برای قابلیت تغییر که اعلام می کند یک Developer می خواهد User Interface را تغییر دهد، این تغییر روی کد اثر گذاشته و تغییر در زمان Design انجام می شود و پاسخ ما اینست که ما می خواهیم این تغییر انجام شود. بدون هیچ اثر جانبی روی بقیه ماژولها و کدهای دیگر و از طریق یک معیار مثلا این تغییر کمتر از ۳ ساعت انجام شود در کل برای هر کدام از صفات کیفی یک سناریوی خاص خودش وجود دارد مثلا در مورد modifiability که صحبت می شود باید بحث اصلی آنرا مشخص کنیم ، بحث اصلی modifiability اینست که تغییر در سیستم را در چه حدی می توانیم ببینیم ، چه تغییر هایی را پیش بینی کردیم و این تغییرات همیشه با هزینه و زمان سرو کار دارند، چه تغییری ، چه کسی ، در چه زمانی می تواند انجام دهد و این تغییر چقدر زمان می برد و بسته به زمانی که خواهد برد، هزینه خواهد داشت ، تغییر در چه حالتهایی انجام می شود یا اینکه بخواهیم یک functionality را اضافه یا حذف کنیم یا تغییر دهیم که خود باعث می شود جند تا از ماژولها عوض شوند،این تغییرات باید کدشان عوض شود یا در فاز طراحی ، طراحی عوض شود ، بعد از اینکه تغییر انجام شد این تغییر باید تست شود ، اگر درست بود باید روی سیستم قرار گیرد. تغییر ۳ فاز دارد ۱- develop ، ۲- test ، ۳- deploy ، در کل modifiability با این موضوع سرو کار دارد.

General senario :

انواع source ها که چه کسانی می توانند نغییر را انجام دهند ، end user ، developer ، مدیر سیستم کسانی هستند که می توانند تغییر را انجام دهند. Developer در فاز طراحی و پیاده سازی ، end user در زمان اجرا و مدیر سیستم در زمان اجرا و پیکر بندی می توانند تغییرات را انجام دهند. بر اساس اینکه source ها چه کسانی هستند Environment ها نیز بر همان اساس هستند. وضعیت و موقعیتی که در سیستم وجود دارد زمان اجرا یا کامپایل و یا ساخت و یا در زمان طراحی مواقعی هستند که تحریک وارد می شود .

Table 4.2. Modifiability General Scenario Generation	
Portion of Scenario	Possible Values
Source	End user, developer, system administrator
Stimulus	Wishes to add/delete/modify vary functionality, quality attribute, capacity
Artifact	System user interface, platform,environment; system that interoperates with target system
Environment	At runtime, compile time, build time, design time
Response	Locates places in architecture to be modified; makes modification without affecting other functionality; tests modification; deploys modification
Response Measure	Cost in terms of number of elements affected, effort, money; extent to which this affects other functions or quality attributes

Stimulus چیست؟ یعنی چه تغییری قراره انجام دهیم؟ بخواهیم اضافه ، حذف و یا تغییر بدهیم دسته ای از صفات کیفی یا functionality ها و یا ظرفیت سیستم را تغییر دهیم.Artifact چه خواهد شد یعنی تغییر انجام شده روی چه چیزی اثر خواهد گذاشت در واقع روی همان چیزی که تغییر داده شده اثر می گذارد. معمولا end user در زمان اجرا چه چیزی را تغییر می دهد ، رابط گرافیکی را تغییر می دهد یا اینکه پلتفرم تغییر می کند، پلتفرم می تواند هم سیستم عامل هم سخت افزار و هم نرم افزار را شامل شود. یا محیط تحت تاثیر تغییر قرار می گیرد یا سیستم هایی که قرار شده با سیستم تغییر داده شده ما در ارتباط باشند نیز تغییر می کنند.

Response : باید جاهایی را که باید قراره تغییرات انجام شود در معماری خود تعیین کرده باشیم یعنی باید از قبل تغییرات را پیش بینی بکنیم و باید تغییرات بدون هیچگونه اثری روی functionality دیگری انجام شود (یعنی باید کمترین اثر را داشته باشد) این تغییرات باید تست و سپس این تغییرات باید deploy یا مستقر شوند.

Response Measure : چطور این پاسخ ها را نشان دهیم. بر اساس هزینه تعداد عناصری که اثر را پذیرفتن ، نشان دهیم ، میزان تلاش و یا هزینه پولی که صرف تغییرات شده را نشان دهیم. بحث measure اینست که چقدر زمان گرفته می شود ، چقدر اثر جانبی دارد ، چقدر باید هزینه دهیم.

Performance : سومین صفت کیفی می باشد. که در مورد آن صحبت می کنیم.

در کارایی با رویدادها سرو کار داریم اگر یک رویدادی وارد سیستم شود که این رویداد چه چیزی می باشد ، می تواند یک وقفه یا یک پیام یا یک درخواست کاربر یا یک گذر زمان باشد. هر روخدادی که اتفاق می افتد سیستم باید به آن جواب بدهد یعنی کاری که از سیستم خواستیم، وقتی که یک رویدادی رخ دهد سیستم چقدر سریع می تواند جواب دهد ، یعنی متوسط زمان انتظار برای آن پاسخ چقدر است.

یک سناریوی کارایی باید درخواستی از سرویسی که از سیستم گرفته است آغاز شود و برای اینکه به این درخواست پاسخ داده شود باید مقداری از منابع سیستم مصرف شود وقتیکه رویداد وارد سیستم می شود و سیستم باید به آن پاسخ دهد، سیستم همزمان در حال پاسخ دادن به سرویسهای دیگر باشد، ممکن است باعث اختلال در زمان پاسخ به درخواست اولیه شود. الگوی ورود رویداد به سیستم مهم بوده و زمان پاسخ گویی آنها اثر گذار می باشد الگوی ورودی رویدادها به سیستم می تواند در فواصل متناوبی از زمان باشد می تواند گهگاهی رخ دهد بصورت تصادفی یا اتفاقی و یا میتواند نه متناوب و نه تصادفی باشد بر اساس یک تابع توزیع نمایی رویدادها وارد سیستم می شوند.

پاسخ سیستم به تحریک می تواند مشخصه بندی شود و در قالب زمان انتظار بیان شود (بین زمان ورود یک تحریک و پاسخ سیستم به آن تحریک latency گفته می شود)

پاسخ سیستم می تواند در مقوله توان عملیاتی بیان شود (تعداد تراکنشهای سیستم که در یک ثانیه می توانند پردازش شوند)

یا اینکه اصلا پاسخی داده نشود، تعداد رویدادهایی که پردازش نشدند بخاطر اینکه سیستم مشغول بوده یا اینکه در مقوله مقدار داده هایی که گم شدند بیان شود.

سناریو کلی کارایی

کاربرها می خواهند ۱۰۰۰ تراکنش را در هر دقیقه بصورت stochastic انجام بدهند در حالی که سیستم عملیاتش در حالت نرمال کار می کند، این تراکنش ها باید با میانگین تاخیر ۲ ثانیه پردازش شوند.

منبع تحریک کاربران هستند و خود تحریک تراکنش ها می باشند ۱۰۰۰ تراکنش که با الگوی stochastic (اتفاقی) وارد سیستم شدند روی چه چیزی اثر می گذارند ، بر روی سیستم اثر می گذارد ، پاسخ مان اینست که به تراکنش ها پاسخ می دهیم اینکه این عملیات ها تاخیرشان کمتر از ۲ ثانیه خواهد بود

جدول ۶ قسمت اصلی کارایی را بیان می کند.

Source : یکی از تعداد منبع های مستقل هستند که می توانند از داخل سیستم باشد.

Stimulus : رویدادهایی هستند که یا بصورت متناوب و یا بصورت تصادفی و یا بصورت اتفاقی وارد سیستم می شوند یعنی رویداد و الگوی وارد شدن به این رویداد ، چون رویدادها بصورت دسته ای وارد می شوند.

Artifact : همیشه سیستم می باشد.

Environment : محیط سیستم یا درحالت نرمال و یا در حالت Overload می باشد.

Response : یا تحریک پردازش می شود و یا اینکه سطح سرویس را تغییر می دهیم.

Response measure : یا بصورت Latency معرفی می شود و یا با Deadline یعنی اینکه حداقل تا این زمان وجود دارد تا پاسخی داده شود معرفی می شود و یا بصورت Throughput معرفی می شود و یا بصورت jitter که نوسانی نموداری معرفی می شود ، یا بصورت نرخ مفقودی یا بصورت مقدار دادههایی که از دست داده ایم معرفی می شود.

Table 4.3. Performance General Scenario Generation

Portion of Scenario	Possible Values
Source	One of a number of independent sources, possibly from within system
Stimulus	Periodic events arrive; sporadic events arrive; stochastic events arrive
Artifact	System
Environment	Normal mode; overload mode
Response	Processes stimuli; changes level of service
Response Measure	Latency, deadline, throughput, jitter, miss rate, data loss

صفت Usability (قابلیت استفاده)

نگرانی و بحث قابلیت استفاده در مورد چگونگی استفاده از سیستم برای کاربران تا بتوانند task های مطلوب خود را کامل بکنند و همچنین در مورد اینکه سیستم چه پشتیبانی را برای کاربر فراهم می کند.

کاربر چه رضایتی از سیستم بدست می آورد یعنی اینکه سیستم چه چیزهایی را برای کاربر فراهم می کند

پشتیبانی هایی که یک سیستم محاسباتی و کامپیوتری که می تواند برای کاربر فراهم بکند در چند دسته مشخص می شود.

انواع پشتیبانی ها

۱- آموزش ویژگیهای سیستم یعنی یک سیستم چقدر می تواند به کاربرش کمک کند تا انجام دادن task ها را یاد بگیرد و راحت می تواند کارکردن با سیستم را آموزش ببیند.

۲- چقدر کاربر می تواند بصورت کارا از سیستم استفاده کند ، یعنی اینکه چگونه بیشترین استفاده و بهره برداری از سیستم را داشته باشد.

۳- اثر خطاها را به حداقل برساند یعنی اینکه سیستم چقدر می تواند اثر خطاهای یک کاربر را به حداقل برساند مثلا وقتیکه بخواهیم روی یک فیلدی که فرم ورود اطلاعات داریم تاریخ را بگیریم اگر فرمت آنرا از قبل مشخص کرده باشیم یا اینکه بخواهیم استان را انتخاب کنیم ار کمبو باکس استفاده می کنیم که باعث کم شدن اثر خطا کاربر می گردد ، به کاربران امکان ورود اطلاعات اشتباه داده نشود.

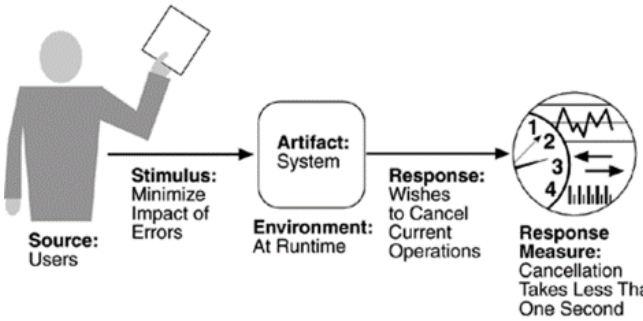
۴- چقدر سیستم خودش را با نیازهای کاربر هماهنگ کند مثلا اینکه بتواند چیدمان صفحه بدست خود سیستم قرار گیرد اینکه فونت ها را بتواند بزرگ یا کوچک کند یا اینکه بتواند خود سیستم نوار ابزار را مرتب کند و ...

۵- افزایش اطمینان و رضایتمندی ، چقدر سیستم می تواند به کاربر این اطمینان را بدهد که کار را به درستی انجام می دهد مثلا سیستم به کاربر پیغام دهد اینکه عملیات با موفقیت انجام شد و ...

همیشه مبدأ تحریک در سناریوهای usability ، کاربران نهایی هستند. تحریک همان نیازمندیهایی هستند که کاربران دارند (از ۵ موردی که در اسلاید قبل گفته شده بعنوان نیازمندیهای کاربر محسوب شده و بعنوان تحریک در نظر گرفته می شود)

تحریک به این صورت در نظر گرفته می شود که یک کاربر بخواهد از سیستم بصورت کارا استفاده کند و یا اینکه بخواهد استفاده از سیستم را یاد بگیرد اثر خطاها را بخواهد کم کند ، و یا اینکه بخواهد سیستم را با خودش هماهنگ بکند و یا اینکه بخواهد احساس راحتی با سیستم داشته باشد.
Artifact در usability خود سیستم می باشد.
محیط یعنی اینکه سیستم در چه شرایطی قرار دارد ، در حالت usability محیط سیستم در حالت پیکربندی می باشد.
تنظیمات اولیه انجام شده و اجرا شروع می شود. (تنظیماتی انجام شده و سیستم طبق این تنظیمات اجرا می شود.)
سیستم باید نیازهای کاربر را از قبل پیش بینی کند ، مثلا سیستم قابلیت Redo ، Undo و Cancel را داشته باشد.
مدت زمانیکه یک task انجام می شود ، تعداد خطاها ، تعداد مسائل حل شده ، میزان رضایتمندی کاربران ، میزان دانشی که کاربر بدست آورده ، نرخ موفقیت عملیات نسبت به کل عملیات ، زمان یا داده از دست رفته وقتیکه یک خطا رخ می دهد. هر کدام از این پارامترها بیانگر Response Measure هستند.

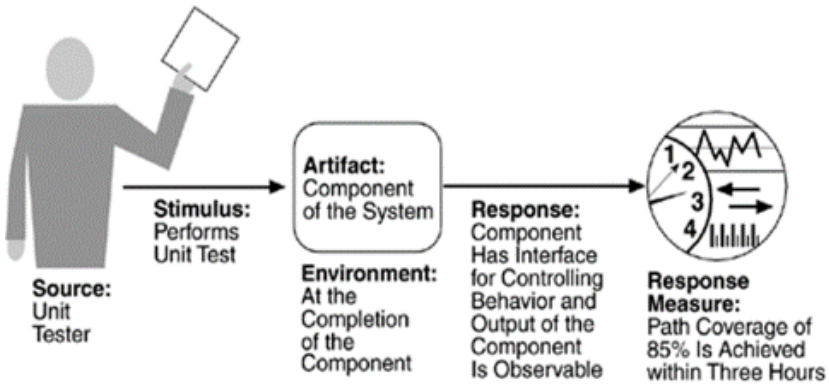
مثال خاص : یک کاربر می خواهد اثر خطاها را به حداقل برساند روی کل سیستم اثر خواهد گذاشت و پاسخی که بخواهیم بدهیم می توانیم Cancel کردن عملیات جاری را فراهم کنیم و لغو کردن کار به مدت ۱ ثانیه زمان می برد.



صفت قابلیت تست یا Testability
قابلیت تست نرم افزار به سادگی تست و خطاهای سرتاسر طول تست را نشان می دهد ، اشاره می کند.
۴۰ درصد از هزینه تولید سیستمهای مهندسی صرف زمان تست می شود.
بزرگترین هزینه تولید سیستمها صرف تست کردن آنهاست.
قابلیت تست در معماری در نظر گرفته می شود یعنی اگر در معماری جاهایی را برای تست دیده باشیم باعث کاهش هزینه یا همان کاهش زمان خواهیم بود و سود زیادی خواهد داشت.

سناریوی کلی قابلیت تست
مبدأ تحریک
تست کنندگان بصورت واحد به واحد تست را انجام می دهند.
تست یکپارچه یا جامعیت یعنی اینکه چند قسمت که تست شدند بهم وصل شوند و بصورت درست و صحیح کارایی داشته باشند.
تست سیستم یا کلاینت یهنی اینکه کل سیستم را بخواهیم تست کنیم.
کسی که درخواست می کند و از سیستم استفاده می کند نیز تست انجام می دهد.
تست توسط توسعه دهندگان نیز انجام می شود و حتی می توان از یک گروه خارج از پروژه نیز جهت تست استفاده کرد.
تحریک ها برای تست کردن یک milestone هستند ، که در پروسه توسعه نشان داده می شوند
(milestoneیعنی اینکه مراحلی به دنبال هم باید انجام شوند تا به یک نتیجه ای برسند)
تستی که انجام شده می تواند مراحل تحلیل و یا طراحی را بصورت increment کامل کند.
کد یا کلاس را تست می کنیم ، و یا اینکه integration را کامل می نیم و یا اینکه کل سیستم را تست می گیریم.
Artifact : قسمتی از کد ، طراحی ، یا کل سیستم می تواند تست باشد
Environment : بستگی به اینکه چه چیزی را تست کنیم همان تعیین کننده است محیط سیستم وقتیکه تست انجام می شود ، می باشد. مثلا در زمان طراحی و یا در زمان توسعه و یا موقع کامپایل و یا موقع استقرار سیستم می توانیم تست را انجام دهیم.
Response : از آنجایی که قابلیت تست با قابلیت مشاهده و قابلیت کنترل در ارتباط می باشد پاسخ های مطلوب برای سیستم اینست که بتوانیم تست های مورد نظرمان که انجام می دهیم کنترل کنیم و هر پاسخی که روی تست انجام شد قابل مشاهده باشد.
تست که انجام می شود بصورت مرحله به مرحله کنترل صورت می گیرد.
درصد دستوراتی که با موفقیت در برخی از تستها اجرا شدند. طول بلندترین زنجیره تست (یعنی معیاری که سختی کارانه ای تست را نشان می دهد) .تخمین زدن احتمال پیدا کردن faultهای اضافی.
(fault های اضافی همان کشف باگ های سیستم می باشد)

Sample testability scenario



مثال : یک واحد تست کننده ای ، می خواهد یک مولفه ای را تست کند ، مولفه سیستم قرار است تست شود ، سیستم در چه وضعیتی می باشد ، سیستم در حال کامل کردن مولفه ها می باشد (یعنی در فاز پیاده سازی می باشد) ، پاسخ نیز بدین صورت می باشد که مولفه رابطی برای کنترل کردن رفتار داشته و خروجی های مولفه قابل مشاهده می باشد ، پاسخ بدین صورت نمایش داده می شود که ۸۵ درصد مسیر تست در سه ساعت پوشش داده شده است.

صفت کیفی امنیت یا security

امنیت معیاری از توانایی سیستم برای مقابله با استفاده های غیر مجاز بوده در حالی که برای کاربرهای قانونی خود سرویسهای خود را فراهم می کند.
امنیت همیشه با حمله سرو کار دارد ، هر تلاشی که برای شکستن امنیت بکار گرفته شود یک حمله نامیده می شود که می تواند به شکلهای مختلفی باشد. مثلا می تواند به شکل درخواست غیر مجاز برای تغییر داده و یا دسترسی به دیتا و یا سرویس باشد و یا اینکه یک سرویس را برای یک کاربر قانونی لغو کند (همیشه حملات یا قصد استفاده و یا تغییر دارند)

انواع پشتیبانی هایی که در امنیت برای سیستم صورت می پذیرد.

- Nonrepudiation
- Confidentiality
- Integrity
- Assurance
- Availability
- And auditing

Nonrepudiation (عدم انکار)

یک تراکنش (دسترسی یا تغییر دادن سرویس) توسط هر قسمتی که آن را انجام داده غیر قابل انکار می باشد.
یعنی اینکه نمی توان سفارشی را که در اینترنت داده شده را انکار کنید.

Confidentiality (قابلیت اطمینان)

ویژگی است که داده ها و سرویسها را از دسترسی غیر مجاز محافظت می کند و معروفترین و مهمترین ویژگی امنیتی که تعریف می شوند .مثلا هکرها نمی توانند مالیات بر درآمد یک سیستم دولتی دسترسی داشته باشند.

Integrity (یکپارچگی)

ویژگی است که داده و یا سرویس همانطور که مد نظر می باشد، تحویل داده میشود. مثلا نمرات دانشجویان پس از ثبت استاد قابل تغییر توسط دانشجو نمی باشد.

Assurance (ضمانت)

ویژگی است که بخشهایی که در یک تراکنش (دسترسی و یا تغییر دادن سرویس) شرکت دارند. همان کسانی هستند که باید باشند. مثلا هنگامیکه مشتری شماره رمز اعتباری کارت خود را در اینترنت برای تاجر وارد می کند آن مشتری همان شخصی می باشد که تاجر تصور می کند.

Availability (دسترس پذیری)

ویژگی است که سیستم برای کاربران قانونی در دسترس باشد حمله هایی از نوع DOS (حمله هایی که اجازه سفارش دادن به سرور را نمی دهند) باشد را جلوگیری می کند . مثلا بخواهیم کتابی را سفارش دهیم ولی بدلیل شلوغ بودن کار سرور این دسترسی برای شما بوجود نمی آید.

Auditing (دنبال کردن یا رد گیری کردن)

ویژگی است که سیستم تمام فعالیتها را در یک سطحی به اندازه کافی تا اگر خرابی رخ داده بتواند بازسازی کند را دنبال می کند. مثلا اگر مقداری پول از حساب خود به حساب دیگر منتقل کنیم سیستم ردگیری لازم را انجام می دهد تا اگر حمله یا حادثه ای رخ داد بتواند اطلاعات را به حالت قبل برگرداند.

- Table 4.4. Security General Scenario Generation
- Portion of Scenario Possible Values
- Source Individual or system that is
- correctly identified
- identified incorrectly, of unknown identity
- who is internal/external,
- authorized/not authorized with
- access to limited resources, vast resources
- Stimulus Tries to display data,
- change/delete data,
- access system services,
- reduce availability to system services
- Artifact System services; data within System
- Environment Either online or offline, connected
- or disconnected, firewalled or open

جدول کلی سناریو امنیت

Source : هر سیستم مستقلی که بصورت درست شناسایی شده و یا ناشناخته باشد و یا شخص داخلی یا خارجی که تعیین هویت شده و یا نشده که بخواهد به منابع محدود یا گسترده دسترسی داشته باشد را دربر دارد (هر شخصی یا هر سیستمی می توانند مبدا تحریک باشند)

Stimulus : خود تحریک می تواند تغییر ، حذف و یا نمایش سرویس ها باشد و یا کاهش دسترس پذیری می توانند خود تحریک باشند و یکی از مهمترین و معروفترین حملات ، حملات DOS می باشند که دسترس پذیری را تحت تاثیر قرار می دهند.

Artifact : بر روی چه چیزی تاثیر گذار هستند ، باید مشخص کرد که به کجا حمله می شود یا به سرویس سیستم و یا به داده درون سیستم حمله می شود.

Environment : سیستم در چه وضعیتی قرار دارد ، معمولا حملات از طریق شبکه اتفاق می افتد که یا سیستم به شبکه وصل بوده ویا خیر یا Offline و یا Online و یا با firewall یا بصورت باز ، پس وضعیت سیستم به این حالتها می تواند باشد.

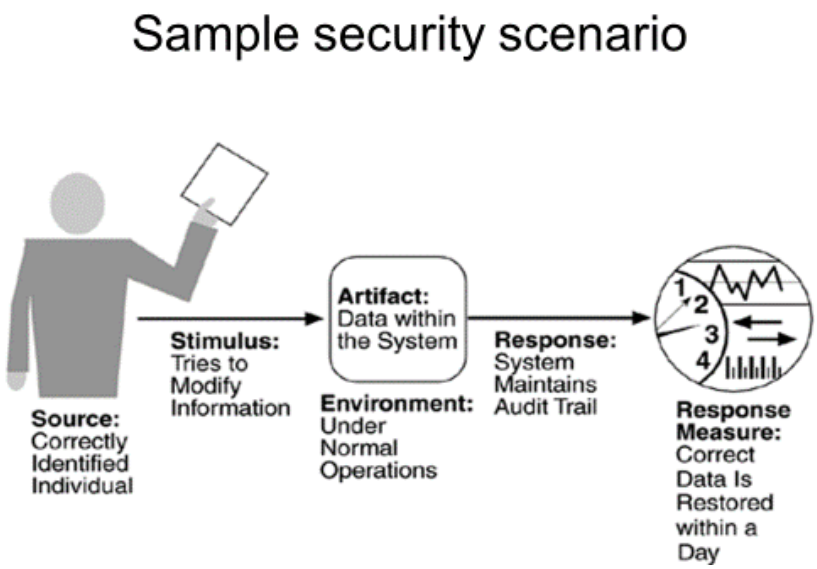
- **Response** Authenticates user;
- hides identity of the user;
- blocks access to data and/or services

- allows access to data and/or services;
- grants or withdraws permission to
- access data and/or services;
- records access/modifications or
- attempts to access/modify data/services
- by identity; stores data in an unreadable
- format; recognizes an unexplainable
- high demand for services, and informs
- a user or another system, and restricts
- availability of services

Response : پاسخها بر مبنای نوع حمله های که صورت می گیرد داده می شود و تعیین هویت کاربران باید مشخص شود و یا شناسایی و مشخصات کاربر مخفی شود و دسترسی به داده و یا سرویس را بلاک و یا سطح دسترسی را مشخص کنیم ، دسترسی و تلاش برای دسترسی ها و تغییرات را ثبت کنیم ، داده ها با فرمت غیر قابل خواندن ذخیره شوند ، دستوراتی که غیر قابل توجیح هستند را شناسایی کنیم ، رفتارهای غیر منتظره را شناسایی کنیم.

Response Measure : زمان ، منابع تلاشی که صورت گرفته را نمایش دهیم ، احتمال شناسایی حمله افرادی که مسئول حمله بودند و یا دیتاهایی که مورد حمله قرار گرفته و قابل تغییر هستند را نمایش دهیم.

درصد دسترس پذیر بودن سیستم بعد از حمله را مشخص کرده و نمایش دهیم ، حمله از نوع DOS را کشف کنیم.



نمونه سناریو امنیت

یک شخصی که بدرستی تعیین هویت شده می خواهد اطلاعات را تغییر دهد، پس داده های درون سیستم تغییر می کند ، لذا سیستم کار نرمال خود را انجام میدهد پاسخ سیستم بدین صورت می باشد چون کاربر مجاز است پس اجازه می دهد تا کارش را انجام دهد، سیستم همه کارهایی را که کاربر انجام داده است را ثبت می کند سپس داده ها در مدت یک روز اصلاح می شوند.

نکاتی بر روی سناریوها
نقش سناریوهای خاص برای نیازمندیهای صفات کیفی مشابه نقش usecase برای نیازمندیهای functional می باشد. یک مجموعه ای از سناریوهای خاص می توانند برای صفات کیفی مورد نیاز سیستم مورد استفاده قرار گیرد
یکی از کاربردهای سناریوهای کلی اینست که به ذینفعان این فرصت داده می شود که خودشان سناریو تولید کنند

صفات کیفی سیستم ۶ مورد می باشد

- قابلیت دسترسی که خودش قابلیت اطمینان دارد
- قابلیت تغییر که خودش شامل قابلیت جابجایی ، قابلیت استفاده مجدد و قابلیت مقیاس پذیری داراست
- قابلیت کارایی
- قابلیت امنیت
- قابلیت تست
- قابلیت استفاده که خودش شامل قابلیت سازگاری با خود و قابلیت سازگاری با کاربر می باشد.

یادآوری از کل جلسات گذشته: در درس مهندسی نرم افزار پیشرفته ما اومدیم روی بحث معماری نرم افزار تمرکز کردیم و در فصل اول در مورد مباحث کلی معماری نرم افزار و جایگاه معماری که معماری چه هست؟؟ و چرخه کسب و کار را گفتیم در فصل دوم راجع به یک تعریف رسمی از معماری نرم افزار گفتیم و ساختارهای معماری را گفتیم و و دسته بندی کردیم در سه دسته کلی و هرکدام انواع اش را گفتیم و در فصل چهارم گفتیم که یکی از اهداف اصلی معماری نرم افزار رسیدن سیستم به صفات کیفی یا همان نیازمندی های غیر functional است پس باید بر روی آنها تمرکز کنیم و گفتیم که نیازهای ما به دودسته تقسیم می شوند یکی نیازمندی های functionalو نیازمندی های nonfunctional که نیازمندی های functional همان سرویس ها و کارهایی که سیستم قرار است انجام دهد و نیاز داریم که انجام دهد و همان نیازهای کاربری است همان سند خواست های کاربر است ولی نیازمندی های nonfunctional به صورت مستقیم نیازمندی های کاربری ما نیست و سرویس هایی که سیستم ما قرار است ارائه بدهد نیست بلکه محدودیت هایی هستند که برای این خدمات و سرویس ها ما تعریف می کنیم که شش تا از مهمترین های آنها را بیان کردیم مثل دسترسی پذیری یعنی سیستم می خواهیم دسترسی پذیری داشته باشد مثل قابلیت تغییر مثل امنیت و مثل قابلیت تست و قابلیت استفاده و گفتیم که اینها نیازمندی های کیفی می شوند و گفتیم که نیازمندی های کیفی یا صفت کیفی که برای سیستم داریم را چگونه تعریف بکنیم چون کیفی است تعریف آنها سخت است که یکی از راه های تعریف آنها ایجاد سناریو است یعنی برای هر نیامندی کیفی یک سناریو بیان کنیم که این سناریو از شش قسمت تشکیل شده بود که یکی منبع تحریک، خود محرک، محیطی که سیستم در آن قرار گرفته است، artifact، پاسخ و یک روشی برای ارائه نمایش این پاسخ را داشت. و کل فصل چهار راجع به سناریوهای کلی که می شود برای صفات کیفی بیان کرد را داشتیم و به ازای هر کدام از این صفات کیفی یک جدول داشتیم که در این جدول کلیه شش قسمت از این صفات کیفی را مشخص می کردیم که در ادامه در مورد این صحبت خواهیم کرد که غیر از صفات کیفی که برای سیستم داریم یک دسته صفات کیفی است که برای کسب و کار داریم و یک دسته صفات کیفی هست که برای خود معماری داریم.

Business Qualities اینها یک دسته دیگری از صفات کیفی هستند که برای کسب کار هستند در واقع برای صفات کیفی محصول نهایی است که سیستم ما تولید کرده است این صفات که در اسلاید فوق نام برده در دو اسلاید بعدی هرکدام را به اندازه دو تا سه خط توضیح داده است. این صفات کیفی کسب و کار عبارتند از:

- زمان تحویل به بازار
- سود و هزینه
- طول و عمر پیش بینی شده برای سیستم
- بازار هدف
- زمان بندی Rollout
- یکپارچگی سیستم های قدیمی

- ✓ زمان تحویل به بازار: اگر که یک فرصت کوتاهی باشد برای اینکه محصول را به بازار ارائه دهیم زمان توسعه و تحویل سیستم خیلی مهم می شود. قبلا هم صحبت کرده بودیم که بعضی از محصولات deadline دارند یعنی زمان ارائه آنها خیلی مهم است و اگر در آن زمان این را وارد بازار نکنیم و ارائه ندهیم و به فروش نرسانیم دیگر ارزشی نخواهند داشت زیرا هم رقابت زیاد است و هم اینکه کلا الان برای آن مشتری وجود

دارد مثل خیلی از نرم افزارها که در زمانهای خاصی ارائه می شود مثلا برنامه هایی که در زمان انتخابات یا جام جهانی ارائه می شود. و این خودش منجر می شود به اینکه آیا تصمیم بگیرید که از المنت هایی که می خواهید در معماری بگنجانید خریداری کنید یا از همین هایی که وجود دارند و قبلا خودتان تولید کرده اید استفاده کنید. یا اینکه از اول آنها را بنویسد که بسته به زمان می توان این تصمیم را گرفت.

- ✓ سود و هزینه: که مسلما هر تولید محصولی یک سودی و یک هزینه ای دارد که هرچه میزان سودی که داشتیم نسبت به هزینه ای که کردیم بیشتر باشد یا در کل درآمد نسبت به هزینه بیشتر باشد این کیفیت سیستم را از نظر کسب و کار بهتر می کند و این برمی گردد به معماری که یعنی معماری های مختلف هزینه های توسعه ای مختلف تولید می کنند.
- ✓ طول عمر پیش بینی شده برای سیستم: درواقع چه پیش بینی برای سیستم شده است و اینکه سیستم بخواهد طول عمر زیاد داشته باشد و برای مدت طولانی کار بکند که در حال حاضر سیستم هایی وجود دارد که در همان اولین لحظات حضورشان در بازار می تواند که از بین برود و سیستم های نرم افزاری شدیدا تحت تاثیر پلت فرمهایی است که وجود دارند بنابراین طول عمر خیلی مهم است که ببینیم برای چند سال می خواهیم این سیستم سرپا باشد و کار بکند و اگر بخواهیم سیستم طول عمر طولانی داشته باشد باید بدانیم چه صفات کیفی برای سیستم مهم است در واقع صفاتی مثل قابلیت تغییر یا مقیاس پذیری یا قابلیت حمل خیلی مهم است که قابلیت تغییر یعنی داخل مجموعه را بتوان به راحتی تغییر داد یعنی فقط مثلا داخل یه ماژول را تغییر دهیم یا حذف بکنیم یا اضافه بکنیم و مقیاس پذیری می تواند از نظر جغرافیایی باشد یا از نظر کاربر باشد یا از نظر مدیریتی هم می تواند باشد که این بحث مقیاس پذیری بیشتر در سیستم های توزیع شده است و در سیستم های متمرکز بیشتر از نظر افزایش کاربران مبحث مقیاس پذیری مطرح می باشد.
- ✓ قابلیت حمل: یعنی سیستم بتواند از روی یک پلت فرم بر روی یک پلت فرم دیگه حرکت بکند و منظور از پلت فرم در واقع سیستم عامل به اضافه سخت افزار می باشد. پس یعنی سیستم نرم افزاری که تولید کردیم بتواند بر روی هر سیستمی و هر نرم افزاری بتواند کار خودش را ادامه دهد.
- ✓ اگر می خواهید به طول عمر سیستم بپردازید و این صفات برای شما مهم باشد مثلا صفتی مثل قابلیت انتقال نیاز دارد که زیر ساخت بهتری بسازید بهترین کاری که می توانید انجام دهید این است که برای سیستم خود از pattern هایی مثل layer استفاده کنید آن زمان شما لایه سیستم عامل را جدا کرده اید پس یه لایه میان افزار می توان به آن اضافه کرد مثل لایه های فریمورک تا بتوان این شفاف سازی را برای آنها انجام داد که اگر قرار باشد سیستم عامل در آن تاثیر نداشته باشد بتوان از آن استفاده کرد که این می شود زیر ساخت اضافه کردن که این باعث می شود زمان ارائه به بازار را تحت تاثیر قرار دهد که باید این مسئله را مد نظر داشت چون تولید سخت تر می شود پس صفت کیفی طول عمر بیشتر متضاد است با صفت کیفی زمان تحویل به بازار.
- ✓ بازار هدف: یعنی که ما کجا می خواهیم سیستم را بفروشیم و پتانسیل بازار چقدر است یا کجاها به فروش می رسد که بعضی از اوقات این بحث جغرافیایی می شود بعضی از اوقات بحث سیاسی می شود بعضی اوقات اجتماعی می شود که همش را می شود همان بحث جغرافیایی در نظر گرفت مثلا یه سیستمی داریم که به درد یه قاره خاص می خورد اما بعضی اوقات یک سیستم همه منظوره ارائه شده است که برای بازار های بزرگتری در نظر گرفته می شود که این نرم افزارها از جمله نرم افزارهایی است که خودشان برای تولید به کار می رود مثل نرم افزار visual studio این نرم افزارها یه جورهایی general purposeهستند که یعنی نرم افزارهایی که وظیفه تولید و کد نویسی را برعهده دارند. و اینها همه منظوره به حساب می آیند .
- ✓ چیزی که اندازه بازار هدف را محدود می کند پلت فرم است که قرار است سیستم با همه مجموعه ویژگی هایی که دارد روی آن پلت فرم اجرا شود که این هم جز صفات کیفی کسب و کار می شود.
- ✓ Rollout schedule: یعنی یک محصولی را معرفی کردیم برای یک عملکرد اصلی حال می خواهیم آن را برای موارد مشابه هم استفاده کنیم که برای اینکار مجبوریم سیستم صفات کیفی قابلیت انعطاف، قابلیت شخصی سازی داشته باشد. به عنوان مثال یک نرم افزاری تهیه شده برای شعب بانک ملی و نیاز هست که برای شعب دیگر هم قابل استفاده باشد بنابراین اگر این امکان برای سیستم در نظر گرفته شود که بتوان از آن در سایر شعب نیز استفاده کرد می گوئیم که این سیستم وقتی در شعب دیگه هم استفاده می شود مثل این می ماند که این نرم افزار Rollup شده است. در کل برای این صفت کیفی یک اصطلاحی بکار می برند و اصطلاحا می گویند سیستم باید قابلیت انبساط و انقباط داشته باشد. یا مثالی دیگر این که مثلا سیستم درآمد شهرداری را بخواهیم در شهرهای مثل اصفهان و تهران و کاشان به کار ببریم که با یک دیگر متفاوت هستند در واقع با نهایت امکاناتی که می تواند وجود داشته باشد این سیستم را عرضه می کنیم ولی همه این مولفه ها برای شهرداری کاشان مورد نیاز نیست و شهرداری کاشان برای همه این مولفه ها هزینه پرداخت نمی کند بنابراین ما باید متناسب با نیاز شهرداری کاشان سیستم را Rollup کنیم و سپس به شهرداری کاشان تحویل دهیم و حتی می توانیم این مولفه ها را به صورت جداگانه قیمت گذاری کنیم یا این را هم در نظر بگیرید یک شهر ساحلی با یک شهر غیر ساحلی درآمدهایشان با هم فرق دارد.

- ✓ یکپارچه سازی با سیستم های قدیمی: اگر که سیستم شما می خواهد با سیستم های موجود کار کند باید یکسری مکانیزم های یکپارچه سازی مناسب برای آنها طراحی کنیم یعنی همان سیستم شهرداری که طراحی می کنیم باید بتواند با سیستم های موجود شهرداری کار کند که به این کار به اصطلاح integrate کردن گویند.

به این صفاتی که گفته شد اگر دقت شود محصول نهایی که می خواهیم عرضه بکنیم کیفت اش مشخص می شود که چگونه است.

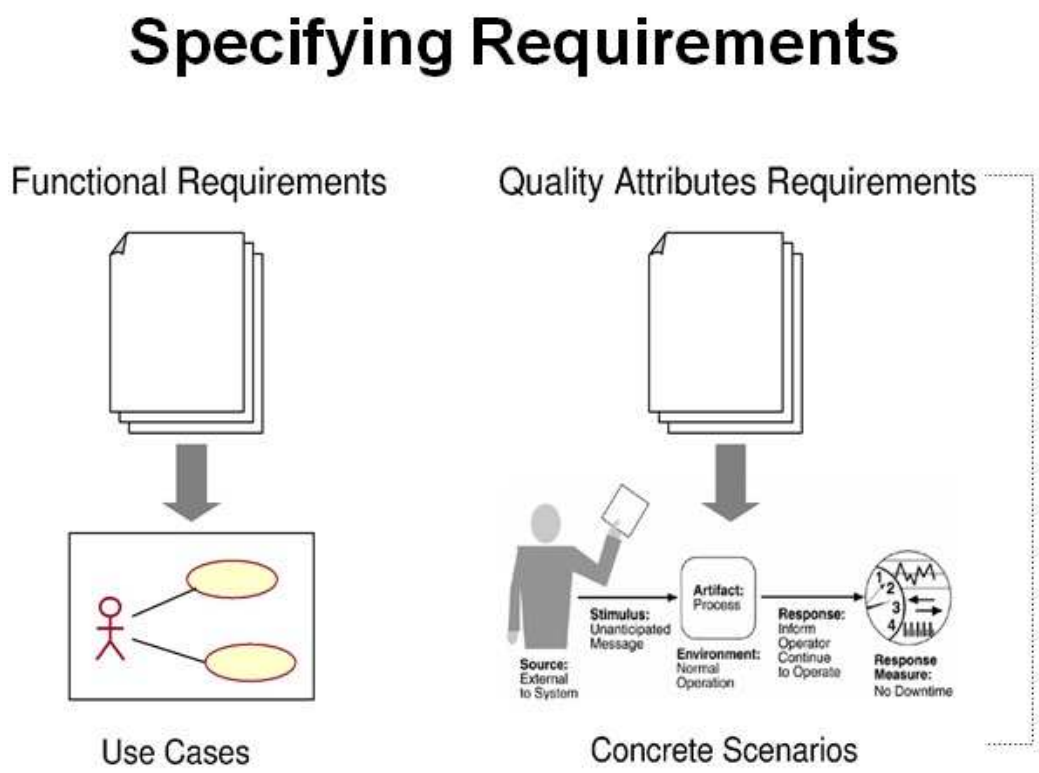
صفات کیفی معماری

صفات کیفی معماری یعنی ما می خواهیم ببینم کلا معماری که طراحی کردیم چگونه است؟! ما قرار نیست بگیم کلا یک معماری خوب هست یا نه بلکه چیزی که مد نظر ماست این است که یه معماری بیشتر ما را به اهدافمان می رساند یا کمتر. سه صفت وجود دارد:

- یکپارچگی مفهومی
- صحت و تمامیت
- قابلیت ساخت

این سه صفت توضیحات کاملش در کتاب هست ولی کلا برای ارزیابی معماری دو روش ATAM و CBAM که در فصل های آینده به آنها خواهیم پرداخت.

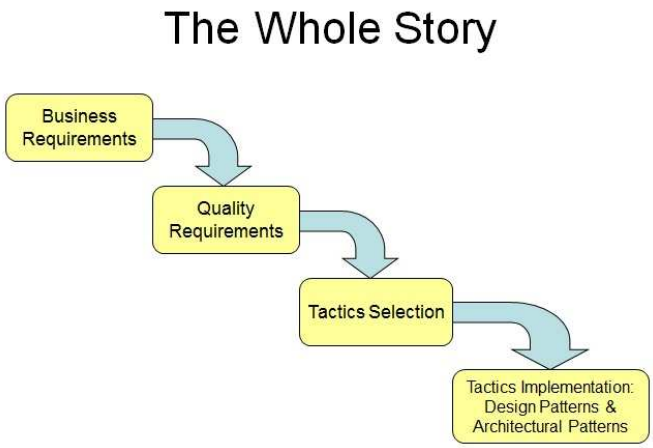
- ✓ یکپارچگی مفهمومی: یعنی اینکه بهتر است سیستم ویژگی های غیر نرمال مشخص را حذف کرده باشد و بهبود داده باشد به جای اینکه منعکس کرده باشد یک ویژگی هایی از طراحی را که شامل ایده های بسیار خوبی هستند که مستقل از همدیگر هستند وغیر منطبق با همدیگر هستند. یعنی به جای یک معماری که در آن پر از ایده های طراحی است که با هم همنواختی ندارند و مستقل از همدیگر هستند این اصلا خوب نیست به جای آن یک معماری ارائه کنید و همه چیز متناسب با آن باشد و این یعنی یکپارچگی مفهومی.
- ✓ صحت و کامل بودن:یعنی همه نیازمندی های سیستم را در معماری به درستی دیده باشیم، محدودیت های زمان اجرا را دیده باشیم، محدودیت های منابع را دیده باشیم، و سپس یک شیوه ارزیابی رسمی را طی کنیم تا از کامل بودن صحت معماری مطمئن باشیم که آن شیوه همان ATAM و CBAM می باشد.
- ✓ قابلیت ساخت: وقتی ما معماری را طراحی کرده ایم که یکپارچه است و صحت دارد و کامل است باید قابلیت ساخت نیز داشته باشد. یعنی اینکه تیم های توسعه ای باشند که بتوانند آن را پیاده سازی کنند و یا ابزارهایی برای پیاده سازی وجود داشته باشند به عنوان مثال یک فرمی را داریم که ایجاد این فرم باید کنترلهای خاصی داشته باشد که یک سری ابزارهایی وجود داشته باشد که بتوان این کنترلها را ساخت. در قابلیت ساخت زمان و هزینه نیز مهم است. قابلیت ساخت منظورش این است که در هزینه و زمانی که تعیین شده است باید بتوان اینها را ساخت.



در اسلاید فوق فقط گفته ببینید که ما نیازمندی های کیفی را با usecase بیان می کنیم و صفات کیفی را با سناریو مستند می کنیم.

فصل پنجم

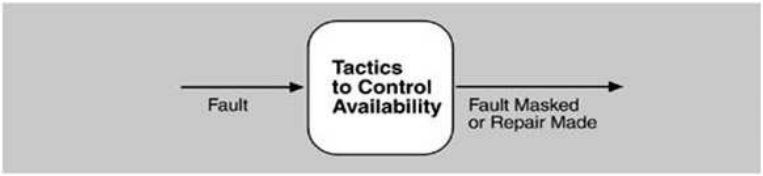
در این فصل همانطور که از نام آن مشخص است بحث رسیدن به صفات کیفی است.
در فصل چهارم که صفات کیفی را معرفی کردیم راه های رسیدن به آنها را نگفتیم و ما فقط معرفی کردیم که مثلا چقدر امنیت می خواهیم یا دسترسی پذیری یعنی چی و یا قابلیت تست یعنی چی البته لازم به ذکر است که در این فصل فقط راجع به صفات کیفی سیستم صحبت می کنیم. حالا می خواهیم در این فصل راهی را معرفی کنیم که به آن صفات کیفی برسیم.
چیزهای خیلی مهمی در این اسلایدها نیست و فقط در همین حد صحبتهایی که برای شما می کنم کفایت می کند.
ما علاقه مند هستیم به این که ببینیم چه جوری معمار می تواند به یک صفت کیفی خاص برسید. نیازمندی های کیفی که مشخص کردیم هرکدام از آنها مسئول این هستند که نرم افزار ما را به یکی از اهدف کسب و کار برساند در این فصل می خواهیم یک سری تاکتیک هایی را معرفی کنیم که با استفاده از آن تاکتیک ها معمار می تواند یک الگوی طراحی یا یک الگوی معماری یا یک استراتژی معماری را ایجاد کند به عنوان مثل فرض کنید یکی از اهداف کسب و کار ایجاد یک خط تولید باشد اگر بخواهیم یک خط تولید ایجاد کنیم باید قابلیت تغییر را در کلاس های خاصی بدیم پس هدف کسب وکار ما را وادار می کند تا به یک صفت کیفی مثل قابلیت تغییر اهمیت بدهیم و برای رسیدن به این صفت کیفی یک تاکتیک باید داشته باشیم. تاکتیک این است که مثلا سیستم را براساس یکسری کلاس ها ایجاد کنیم که این کلاس function هایی دارد که می توان آنها را به راحتی تغییر داد که کل این مفاهیم در اسلاید بعد توضیح داده شده است.



یکسری نیازهای کسب و کار داریم که توسط کارفرما تعریف شده از روی آنها نیازهای کیفی را بیرون می کشیم بعد از اینکه سناریوها را نوشتیم برای پیاده سازی آنها یکسری تاکتیک هایی را اجرا می کنیم تا آن سناریو ها را پیاده سازی کنیم که این تاکتیک ها از قبل وجود دارند که درواقع همان pattern ها هستند پس این تاکتیک ها را داخل patternها طراحی و pattern های معماری پیاده سازی می کنیم.
پس می خواهیم یک سری تاکتیک معرفی کنیم که با استفاده از آن تاکتیک ها به سناریوهایی که داریم برسیم. پس از اول شروع می کنیم به ازای هر صفت کیفی سیستم یک دسته از تاکتیک ها را بیان خواهیم کرد.

Tactics

- **A tactic is a design decision that influences a quality attribute.**



- **e.g. using redundancy to increase availability**
- **Tactics can be refined to other tactics to become more concrete; e.g. redundancy: redundancy of data + process**

یک تعریفی از تاکتیک اینجا مطرح شده است. تاکتیک یک تصمیم طراحی است که روی صفت کیفی تاثیر می گذارد. به عنوان مثال بحث fault در صفت کیفی دسترسی پذیری بود وقتی که fault وارد می شود ما سناریو داشتیم که حالا به ازای این fault باید به آن یک پاسخی بدهیم و برای اینکار باید یک تاکتیکی را برای کنترل دسترس پذیری داشته باشیم.

مثلا می توانیم با استفاده از افزونگی دسترس پذیری را افزایش دهیم که دسترس پذیری می تواند روی داده باشد یا افزونگی فرآیند باشد مثلا افزونگی می تواند یک کپی از دیتابیس باشد که مثلا این دیتابیس خراب شده و الان جواب نمی دهد بزاریم روی یک دیتابیس دیگر که کپی از آن است. که این خودش یک تاکتیک است.

تاکتیک ها خودشان می تواند به تاکتیک های خاص تر دیگر تقسیم شوند که به عنوان مثال افزونگی می تواند داده باشد یا فرآیند باشد مثلا یه ماژول محاسباتی را از آن کپی داشته باشیم یا یک داده ای را از آن کپی داشته باشیم.

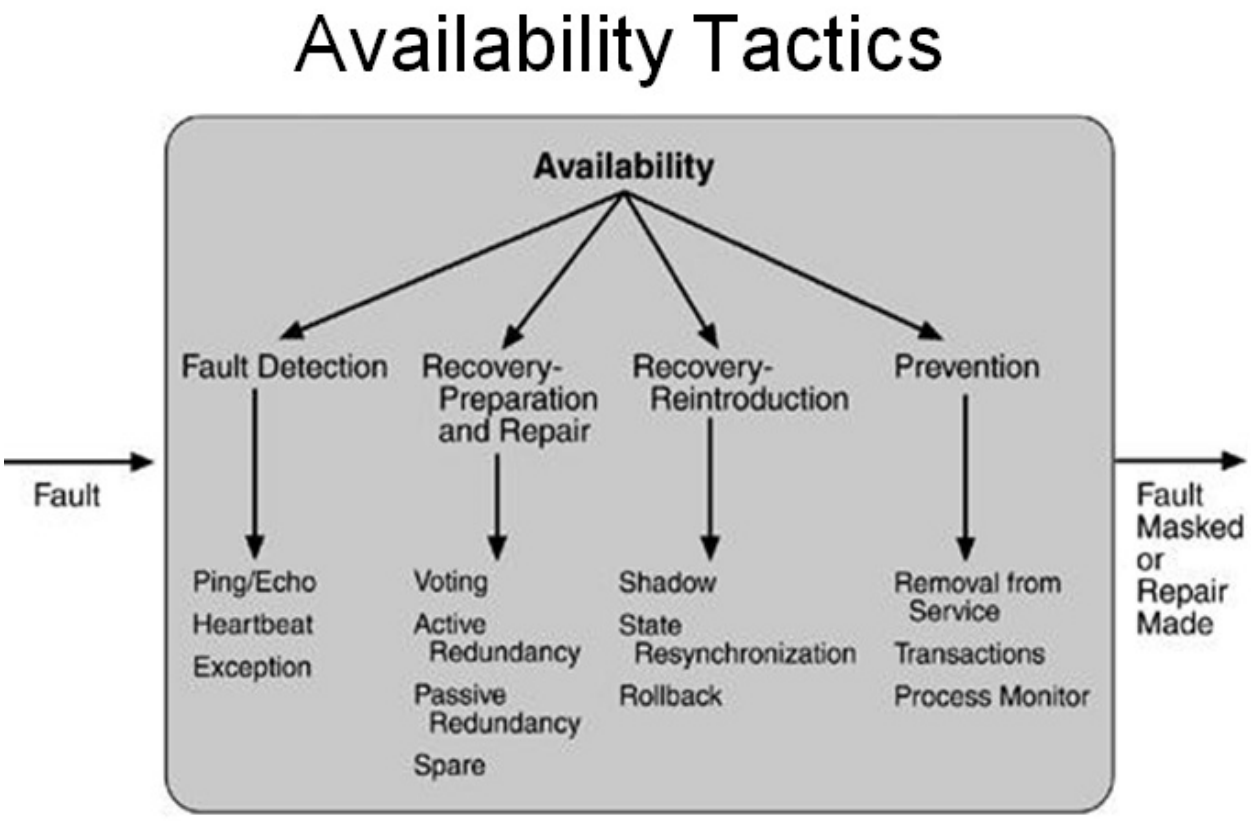
تفاوت تاکتیک با تکنیک در این است که تکنیک می شود ابتکاری که برای پیاده سازی تاکتیک و تاکتیک همان روشی است که شما پیش گرفته اید. پس اینجا تاکتیک همان تصمیم طراحی می شود.

طراحی یک سیستم شامل مجموعه ای از تصمیمات می شود که بعضی از این تصمیمات کمک می کند به کنترل پاسخ های صفت های کیفی وبرخی دیگر ما را به نیازهای functionality می رساند و هرتاکتیکی یک آپشن یا اختیار طراحی است برای معمار. برای مثال یک تاکتیک افزونگی را برای افزایش availability سیستم معرفی می کند. هرکدام از این اختیاراتی که معمار دارد می تواند availability را افزایش داد اما فقط همین یک اختیار نیست بلکه یک مجموعه از این تاکتیک ها وجود دارد برای رسیدن به یک سناریو خاص و معمولا برای رسیدن به دسترس پذیری بالا از طریق افزونگی نیاز به یک بهنگام سازی پیوسته ای خواهیم داشت. یعنی در اثر استفاده از یک تاکتیک مجبوریم از تاکتیک های دیگر نیز استفاده بکنیم. پس با این توصیفاتی که گفتیم مجموعه از تصمیمات طراحی را تاکتیک می گوییم .

دو مورد از این موارد گفته شده وجود دارد یکی اینکه تاکتیک ها از تاکتیک های دیگر بیان شده که همانطور که بیان شد افزونگی برای رسیدن به دسترس پذیری است که افزونگی هم می تواند در دیتابیس باشد و هم می تواند محاسباتی باشد. یعنی اضافه کردن سیستم های کنترلی جاسازی شده. مطلب دوم اینکه ما pattern هایی داریم که یک بسته ای از تاکتیک ها است که مثلا pattern که برای پشتیبانی از availability است مثل استفاده از هردو تاکتیک های افزونگی و هنگام سازی. یعنی اگر ما افزونگی داشته باشیم باید تاکتیک هایی برای هنگام سازی این نسخه هایی تکراری نیز داشته باشیم.

اسلاید فوق موضوع خاصی ندارد و فقط بیان می کند که برای هر شش صفت کیفی می خواهیم یکسری تاکتیک هایی را معرفی کنیم.

از تاکتیک های دسترس پذیری شروع کرده که در جدول اسلاید بعد قابل مشاهده است .



کلا دسترس پذیری راجع به این بود که fault رخ داده که پوشش داده نشود و تصحیح نشود منجر بهfailure شدن سیستم می شود پس زمانی که یک fault رخ می دهد یک تاکتیکی پیش بینی کنید تا بتوان این fault را رفع کرد که چهار دسته تاکتیک داریم که یک دسته فقط fault ها را شناسایی می کنند مثل: ping/Echo و heartbeat و Exception یک دسته دیگر از تاکتیک ها Recovery می کنند ولی از دید این که سیستم را ترمیم می کند که بتوانند مثل این fault سیستم به کار خودش ادامه بدهد مثل: ,voting ,Active Redundancy, passive Redundancy, Spare و دسته سوم دسته ای از تاکتیک هایی است که مشخص می کند که اگر زمانی که برای سیستم مشکل پیش آمد و بعد از ابینکه ما Recovery کردیم حالا برای اینکه به سیستم برگردد چه تاکتیک هایی وجود دارد که شامل Shadow , State Resynchronization , Rollback است . و دسته آخر کلا جلوگیری می کنند که سیستم دچار faultشود و کاری می کنند که هیچ وقت fault یی ما را به failure نرساند.

در این اسلاید تاکتیک های فوق را توضیح خواهد داد. از دسته تاکتیک های availability برای شناسایی fault استفاده می کنید که سه مدل دارد که اولی ping/Echo است. وقتی که یک مولفه ای یک ping را ارسال می کند به یک مولفه دیگر در واقعه یک مجموعه از مولفه ها هستند که بایکدیگر کار می کنند و برای اینکه ای مولفه ها مشخص بشوند که کدامشان فعال هستند هر مولفه ای برای مولفه ی دیگر یک pingمی فرستد و نیاز دارد که این pingٍ نیزEcho شود واگر در مدت زمان تعریف شده برگشت یعنی آن مولفه مورد نظر سالم است و اگر برنگشت یعنی آن مولفه fault شده یا خراب شده است. این برای کلاینت ها استفاده شود تا مطمئن شوند یک سرور و مسیر ارتباطی که برای آن سرور وجود دارد سالم است با نهایت کارایی مورد انتظارش.

تاکتیک heartbeat یعنی همان ضربان قلب که در این مورد یک مولفه یک پیام های ضربان قلبی را به طور متناوب به مولفه های دیگر ارسال می کند و مولفه های دیگر به آن گوش می دهد و اگر این ضربان توسط سایر مولفه ها شنیده نشود نشان دهنده آن است که آن مولفه که این ضربان را ارسال می کند دیگر سالم نیست بنابراین در این جا هم یک مجموعه از مولفه ها وجود دارد که یکسری پیام ها را برای دیگران ارسال می کند این هم سالم بودن آن مولفه را نشان می دهد و هم یک کار مثبتی است که انجام می شود مثلا یک داده ای است که ارسال می کند با این روش مابقی که گوش می دهد نشان می دهد این مولفه سالم است و زمانیکه این پیام شنیده نشود مشخص است که برای آن مولفه یک مشکلی به وجود آمده به عنوان مثال یک ماشین گویا داریم که می تواند به صورت متناوب یک پیام هایی از لاگ های سیستم را بفرستد و این پیام ها هم نشان دهنده سالم بودن سیستم است و شامل داده هایی است که باید برای سیستم ارسال شود.

تاکتیک بعدی Exception یا حالت های استثنا است که می توانیم این حالت های را شناسایی کنیم این یک روشی است که fault را تشخیص می دهد با تشخیص استثنائات فقط فرقی که با حالات های قبل دارد این است که Exception داخل یه مولفه است که خود آن مولفه ای که دچار fault شده این خرابی را شناسایی کند و برخلاف حالت های قبلی که چند مولفه با یکدیگر کار می کردند و مولفه های دیگر متوجه می شود که درون یک مولفه دیگر fault رخ داده است اما در این حالت یک مولفه است که خودش داخل خودش برای شناسایی fault ها از Exception استفاده می کند.

تاکتیک های ping/Echo و heartbeat با پروسس های مختلف عمل می کنند اما تاکتیک Exception معمولا در داخل یک پروسس عمل می کنند.

کنترل کننده Exception ها معمولا یک ترجمه معنایی از یک fault را به یک فرم قابل پردازش تبدیل می کنند.

تا اینجا راجع به تشخیص fault صحبت کردیم اما در این اسلاید راجع به دسته تاکتیک های دسترسی پذیری که از نوع Recovery است برای فراهم کردن مجدد سیستم و تعمیر سیستم صحبت می کنیم.

ترمیم خرابی شامل فراهم کردن ترمیم و تعمیر کردن سیستم می شود که برخی از تاکتیک ها در زیر آمده است:

- voting: فرآیندها روی چند پردازنده افزونه اجرا می شوند هرکدام ورودی یکسان می گیرند و یک خروجی متفاوتی را برای voter می فرستند اگر voter یا رای گیرنده یک رفتا انحرافی را از یکی از پردازنده مشاهده کند متوجه می شود که یک خرابی اتفاقی افتاده است. الگوریتم voting می تواند برساس قوانین اکثریت باشد یا براساس مولفه ارجح تر باشد یا هر الگوریتم دیگری. voting را هم به عنوان شناسایی fault می شود در نظر گرفت هم به عنوان recovery به این شکل که چندتا پردازنده داریم که همه یک کاری را انجام می دهند که انگار ما افزونگی ایجاد کردیم و backup داریم و همه آنها خروجی را برای voter می فرستند voter براساس جوابهایی که برای آنها می آید اگر قوانین اکثریت باشد اینجوری برداشت می شود که آنهایی که بیشترین تعداد جوابهای یکسان را دارد آن جواب درست است و آن مولفه ای جواب متفاوت داده است مولفه ای است که به عنوان مولفه ای دارای failure است شناسایی می شود و بنابراین با این روش هم مولفه ای که دچار failure شده شناسایی می شود و هم جواب درست مشخص می شود یعنی با خراب شدن یکی و دوتا باز سیستم به کار خودش ادامه می دهد و گاهی اوقات ممکن است قوانین براساس مولفه ارجح تر باشد بنابراین مولفه های براساس ارجحیت دسته بندی می شوند و آن مولفه ای که ارجحیت بالاتری دارد پاسخ اش به عنوان پاسخ صحیح شناسایی می شود.

دو روش دیگر مربوط به Recovery که در ادامه توضیح داده خواهد شد مرسوم تر هستند

- Active redundancy: افزونگی فعال یا hot restart منظورش این است که همه مولفه های فعال پاسخ می دهند به یک رویداد به صورت موازی ولی معمولا اولین پاسخ استفاده می شود و مابقی آنها دور ریخته می شود وقتی که یک fault رخ می دهد زمان down time سیستم با استفاده از این تکنیک فقط در حدود میلی ثانیه خواهد بود زیرا backup هایی که از سیستم وجود دارد آماده استفاده هستند و این چند ثانیه صرف این خواهد شد که ما سوییچ بکنیم روی پاسخ یک مولفه پشتیبان دیگر و این معمولا در پیکربندی های کلاینت و سرور و پایگاه داده مخصوصا در سیستم های مدیریت پایگاه داده استفاده می شود که پاسخ سریع در حتی در زمانی که fault رخ داده است برای ما مهم است. در واقع ما افزونگی ایجاد می کنیم به منظور دسترس پذیری بعد اینا همه با هم به صورت موازی کار می کنند و این می شود Active و در این حالت که با هم کار می کنند فقط جواب یکی را در نظر می گیریم و زمانی که مشکلی پیش می آید فقط یک چند ثانیه ای صرف می شود برای سوییچ کردن به یک نسخه دیگر این بدرد وقتی می خورد که برای ما recovery سیستم خیلی مهم است و می خواهیم هر جوری که شده است سیستم پاسخ بدهد.

- passive redundancy: در این حالت هزینه کمتری دارد که در این حالت نسخه های پشتیبان مولفه در حالت کار نیستند بلکه به صورت خاموش هستند و آن مولفه آنلاین وقتی یک کاری را با موفقیت انجام می دهد برای تمام پشتیبان ها ارسال می کند و پشتیبان ها با یک گام عقب تر از مولفه اصلی به روز هستند و هروقت برای آن مولفه اصلی مشکلی ایجاد شد باید سراغ پشتیبانی برویم که جدید ترین به روز رسانی بر روی آن انجام شده و آن را برداریم یعنی در این حالت ممکن است ما یک دسته ای از آخرین عملیات هایی که کامل نشده اند را از دست بدهیم در مقابل مجبور نیستیم که چند مولفه داشته باشیم که همه آنها فعال باشند. در واقع افزونگی غیرفعال یا warm restart وقتی که یک مولفه ای به یک رخداد پاسخ می دهد این را به مولفه های دیگر اطلاع می دهد و وضعیت آنها را به روز رسانی می کنند و زمانی که یک fault یی رخ می دهد سیستم باید ابتدا مطمئن شود که وضعیت آن پشتیبان ها به اندازه کافی تازه است تا دوباره سیستم را از طریق آن پشتیبان ها از سر بگیرید این سیستم معمولا برای سیستم های کنترلی استفاده می شود که هنگامی که ورودی ها در مسیرهای ارتباطی دچار مشکل می شوند و یا از سنسورهایی داریم داده جمع می کنیم بنابراین در زمان خطا از یک سیستم به یک سیستم دیگر سوییچ می کنیم.

- spare: زاپاس یا یدک این حالت بازم backup است اما backup از پلت فرم است یعنی یک پلت فرم محاسباتی زاپاسی داریم که به صورت standby نگهداشتیم و این را پیکربندی کردیم بابت تعداد زیادی از مولفه های که fail کردن بنابراین اگر فقط یک مولفه fail کرد از روش های پیشین اقدام می کنیم ولی اگر بیش از یک مولفه fail کردن کلا از یک پلت فرم باید بریم روی یک پلت فرم دیگر. یعنی انگار از سیستم اولیه یک کپی پشتیبان داریم و این باید به پیکر بندی نرم افزار مناسب برگردانده شود. که یکسری وضعیت های مناسب در آن وجود داشته باشد که این مال قبل از اتفاق افتادن آن خطا باشد. یعنی باید یکسری check Point در سیستم داشته باشیم که در آنها وضعیت سیستم را ثبت کنیم و سپس این checkpoint را برای آن سیستم زاپاس ارسال می کنیم بنابراین این وضعیت سیستم همیشه در آن نسخه پشتیبان وجود دارد و هر وقت در آن سیستم fail رخ می دهد که چندتا مولفه های آن با یکدیگر دچار خرابی می شوند ما مهاجرت می کنیم به آن پلت فرم زاپاس و از آخرین نقطه ای که وجود دارد دوباره کار را شروع می کنیم.

این چهار مورد که گفته شده مال زمانی بود که اگر خطایی رخ داد چه جوری سیستم بازم کار را ادامه دهد. اما بحث بعدی این است که آن مولفه ای که خراب شد و خرابی برای آن رخ داد چه جوری به کار برگردد که این می شود دسته تاکتیک های سوم دسترس پذیری که در اسلاید بعد توضیح داده شده است.

- Shadow: آن مولفه ای که قبلا خراب شده می تواند در حالت سایه برای مدت زمان کوتاهی اجرا شود تا ما مطمئن شویم که رفتار درستی از خودش نشان می دهد و آن را دوباره به سیستم برمی گردانیم.
- وضعیت هنگام سازی مجدد: مثلا در تاکتیک ها passive و Active ما نیاز داریم که مولفه وضعیت آپگرید شده اش را restore بکنیم تا آن را به سیستم برگردانیم یعنی باید اصلاح کنیم بعد به سرویس برگردانیم تا مثل بقیه شود و همزمان با بقیه شود. که این مدت زمانی هم که طول می کشد تا این آپدیت را انجام دهیم بستگی دارد به مدت زمان بسته هایی که باید این آپدیت را انجام دهیم و این مهلت زمان هم down time آن مولفه را تعین می کند.

- CheckPoint/RollBack: که شبیه همان چیزی است که در spare دیدیم که یک checkpoint وضعیت هایی که ایجاد می شود را ثبت می کند که این ثبت کردن ها به صورت ترتیبی می تواند باشد یا با پاسخ به هر رخداد خاصی که روی می دهد و هنگامی که یک سیستم failمی کند با شناسایی یک وضعیت غیر هنگام و غیر سازگار سیستم باید برگردد به آخرین وضعیت سازگاری که وجود دارد و همه تراکنش هایی که رخ داد در این فاصله باید حتما ثبت شوند.

روشهای های دسته چهارم دسترس پذیری عبارتند از:

- Removal from Service: یعنی جلوگیری بکنیم از این که مولفه دچار failure شود یعنی زمانی که داره نزدیک می شود به fault کلا حذف کنیم. این تاکتیک یک مولفه ای از سیستم را که سطح فعالیت های آن پایین آمده را حذف می کند برای این که از failure های پیش بینی شده جلوگیری کند به عنوان مثال یک مولفه ای را ریستارت می کنیم برای اینکه از فقدان حافظه جلوگیری کنیم و این فقدان حافظه بعدا ممکن است سیستم را دچار مشکل کند این حذف از سرویس اگر در استراتژی های معماری دیده شده باشد می تواند به صورت اتوماتیک باشد یا به صورت دستی باشد.

- Transaction: تراکنش ها چهار خصوصیت داشتند که همان ACID است و اگر یک رفتاری را در قالب یک تراکنشی بیان کنید هرچقدر هم fault در آن رخ دهد باز هم سیستم را در حالت پایدار نگه می دارد و تغییرات را اعمال نمی کند تا زمانی که آن با موفقیت به پایان برسد بنابراین اگر از اول سیستم را به صورت تراکنش معرفی کنیم این مشکل حل خواهد شد.

Atomic یعنی یا همه دستورات انجام می شود یا هیچکدام.

consistency یعنی یک تراکنشی که انجام می شود داده های آن همگون باقی می مانند یعنی سازگاری رعایت می شود.

Isolation یعنی دستورات تراکنش بتواند با هم به صورت همروند اجرا شوند و اثری روی هم نمی گذارند اگر چند تراکنش همزمان با هم انجام شوند.

Durability: یعنی بعد از پایان یافتن تراکنش تاثیر آن همچنان پایدار باقی می ماند و به هیچ عنوان لغو نمی شود.

- process monitor: وقتی یک fault یی در یک فرآیندیی شناسایی می شود یک process monitor می تواند آن فرآیندی که کار درست را انجام نمی دهد را حذف کند و یک فرآیند جدید را جایگزین کند و مقدار دهی اولیه برای آن فرآیند جدید به وجود آمده می تواند شبیه همان تاکتیک spare انجام شود و یعنی جلوگیری از یک failure را به عهده یک process monitor می گذاریم و این process monitor یک فرآیند نظارتی است که بررسی می کند سیستمی که درست کار نمی کند را قبل از اینکه بخواهد fault یی رخ بدهد که بعد بخواهد failure شود آن را حذف می کند.

تاکتیک های قابلیت تغییر:

تاکتیک هایی که برای کنترل قابلیت تغییر هستند دارای اهداف زیر هستند:

-کنترل کردن زمان و هزینه پیاده سازی ، تست و اعمال تغییرات است.

یاد آوری:[قبلا راجع به کلا سناریو هایی برای قابلیت تغییر صحبت شد اشاره کردیم که برای اجزای تغییرات کارهایی که انجام می دهیم عبارتند از : ابتدا کد تغییرات را می نویسیم و آنرا پیاده سازی میکنیم و سپس تست می کنیم که آیا درست است یا خیر؟ و بعد انرا اسقرار (deploy) می کنیم (یعنی وصلش میکنیم و اعمالش میکنیم) .پس سه کار انجام می دهیم. مهمترین بحثی که داریم اینکه این سه کار زمان و هزینه دارد- یک زمان طی میکنیم تا انجامش دهیم و هزینه ایی که صرف انجام آن می کنیم . در همین راستا مسئله مهم دیگر اینکه چه زمانی می توانیم این تغییرات را اعمال کنیم و این تغییرات توسط چه کسی باید انجام شود یاد اوری طبق سناریو و متون source ها چی هست بستگی به این دارد که چه کاری بخواهی انجام دهید و در چه زمانی . یادتان باشد طراح ، پیاده ساز و حتی کاربر می تواند این تغییرات را اعمال کند و می خواهیم پیاده سازی کنیم .]

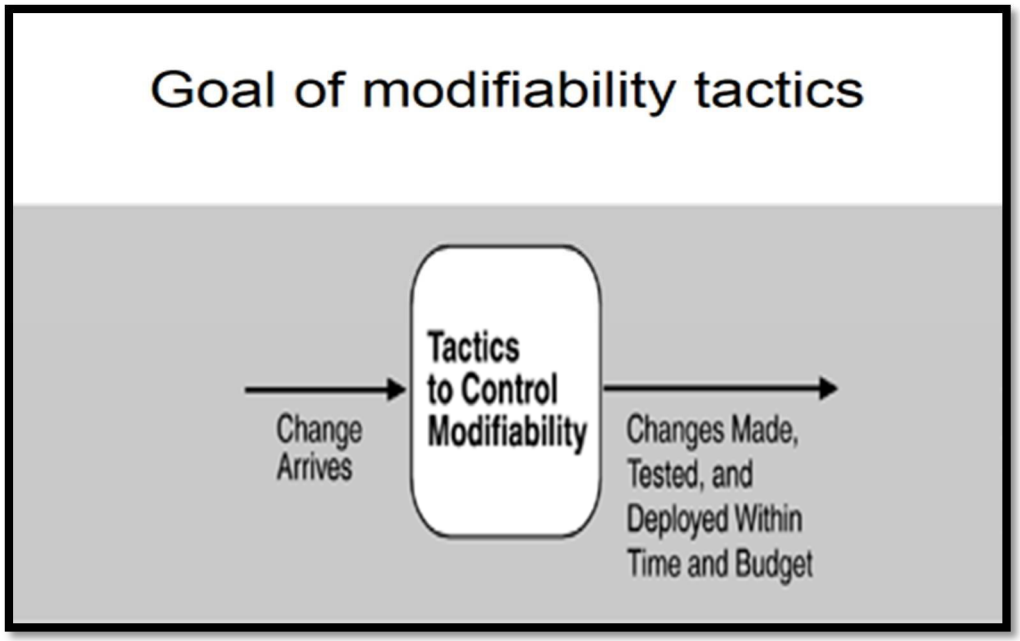
ما می خواهیم تاکتیک هایی را برای قابلیت تغییرات در دسته ها و مجموعه هایی براساس اهدافشان.

دسته اول — یک مجموعه از این تغییرات هدفشان کاهش تعداد ماژول هایی است که بصورت مستقیم تحت تاثیر تغییر قرار می گیرند که این دسته را Localize Modification یا تغییرات محلی شده می نامند پس دسته اول تاکتیک ها ، مربوط به تاکتیک هایی است که می خواهد تعداد ماژولهایی که مستقیما اثر می گیرند از اعمال تغییرات را کم بکند که نامش را Localize Modification گذاشته اند.

دسته دوم تغییرات – دسته دوم هدفشان کاهش تغییرات فقط به همان ماژول های محلی شده است یعنی میخواهیم از اثر تکرار شونده تغییرات جلوگیری کنیم.یعنی اینکه ماژول هایی که غیر مستقیم تحت تاثیر قرار میگیرند را کم کنیم اما بعضی وقتها شما میخواهید سیستم تان یک سرویسی را اضافه بدهید مجبورید یک یا چند ماژول را تغییر دهید .یک ماژول را کم کنید و یا ماژول جدید را اضافه کنید.پس مستقیم سراغ یک یا چند ماژول می آیند تا آن سرویس جدید را اعمال کنید. این دسته از تاکتیک هایی است که ماژول ها را مستقیم تحت تاثیر قرار میگیرند را بررسی میگند اما بعضی وقتها با تغییری که روی ماژول A دادید ماژول Bچون از ماژول A استفاده میکرده یا وابستگی خاصی به آن داشته آنهم تغییر میکند ، این جزء دسته انتشار اثر قرار میگیرند ما میخواهیم تاکتیک هایی داشته باشیم که اول ماژول هایی که غیر مستقیم تحت تاثیر قرار میگیرند یا در مسیر انتشار اثر قرار میگیرند را کم کنیم یعنی کلا هدف اینست که با کمترین میزان تغییر به آن تغییری که میخواهیم برسیم .

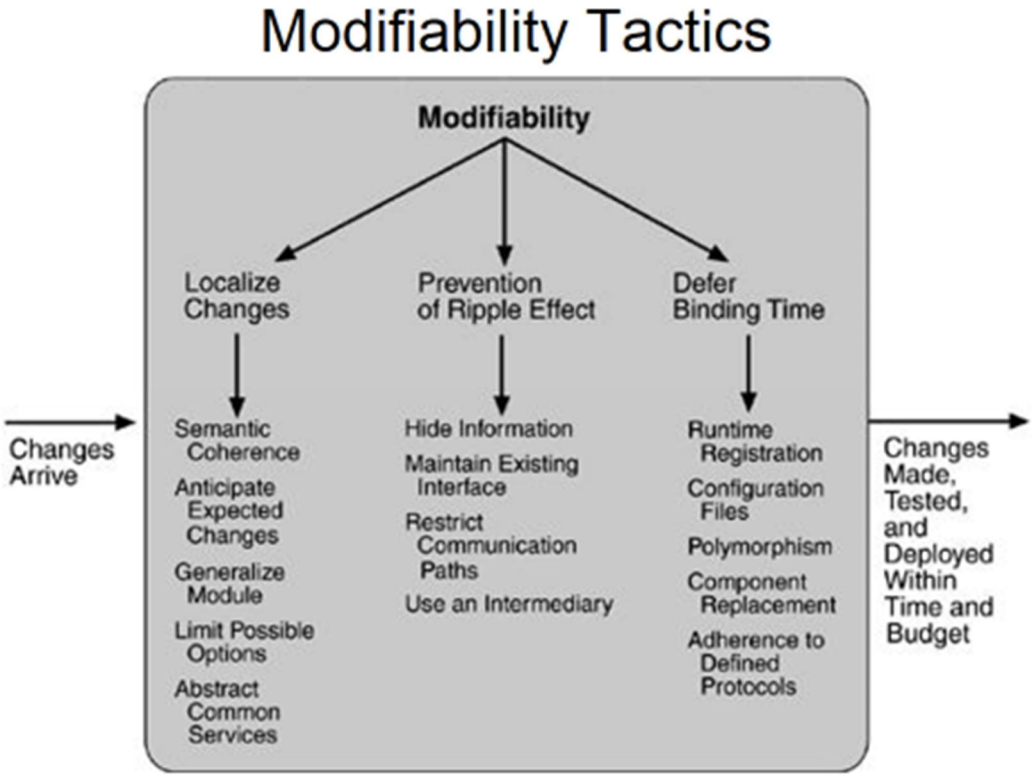
دسته سوم تاکتیک ها – در مورد کنترل کردن زمان deploy و هزینه است یعنی اینکه مربوط به این می شود که یکسری تاکتیک ها را پیش بگیریم اجازه اعمال تغییر توسط کاربر پایانی و در زمان اجرا را فراهم کند یعنی وقتی که شما کاربر نهایی تان بتواند تغییر انجام دهد زمان هزینه کم می شود مخصوصا زمان deploy. یعنی اینکه شما انگار قبلا تغییرات را دیدید ، آماده اش کردید ، deploy اش کردید و آنرا در سیستم قرار دادید و وقتی کاربر پایانی میخواهد اعمالش کند کافیست خیلی سریع در زمان اجرا در واقع deploy را implement(تکمیل کردن ، اجرا کردن) کند. تست کند آن کار یا تغییر از پیش انجام شده را.

نام این دسته را defer binding time یعنی به تاخیر انداختن زمان انقیاد .(که بعدا توضیح داده می شود).



این شکل توضیحی را جع به هدف تاکتیک های قابلیت تغییر ارایه می دهد.

یک change (یا تغییری) به سیستم می رسد ما یکسری تاکتیک هایی را برای کنترل این قابلیت تغییر فراهم می کنیم و خروجی آنکه می توانیم بسازیم تست ش کنیم و نصبش کنیم با کمترین زمان و هزینه.



این اسلاید دسته بندی از تاکتیک ها را نشان میدهد.

که شامل سه دسته شدند:

۱-localize changes

۲- Preventation of Ripple Effect . که هرکدام چند نمونه دارد که به آن خواهیم پرداخت

می خواهیم ببینیم چه کاری باید انجام دهیم تا میزان ماژولهایی که می خواهند مستقیما اثر میگیرند از تغییر ما کم شود.هدف این تاکتیک ها در این مجموعه اینست که انتساب بدهد مسئولیت را به ماژولها در طول فاز طراحی بطوری که تغییراتی که از قبل پیش بینی کرده اید توی یک محدوده خاص محدود شوند پس در زمان طذاحی این تغییرات انجام میشوند، این تغیییرات از قبل پیش بینی شده اند و ممکن است در آینده نیاز باشد که این تغییرات انجام شوند و می خواهیم کاری بکنیم برای انجام این تغییرات کمترین محدوده از ماژولهایمان تحت تاثیر این تغییرات قرار بگیرند.

اولین نمونه از تاکتیک های محلی کردن تغییرات: maintain semantic coherence (حفظ وابستگی معنایی):

وابستگی معنایی اشاره دارد به ارتباط بین مسئولیت های در داخل یک ماژول .

-هدف اینست که مطمین شویم همه این مسئولیتها با همدیگر بدون هیچ وابستگی اضافی به ماژول های دیگر درون یک ماژول خاص کار انجام می دهند.

- یک زیر تاکتیک از این دسته تاکتیک های اینست که سرویس های رایج را abstract (انتزاع- چکیده) کنیم .

مثال: استفاده از اپلیکیشن هایی مانند framework ها و نرم افزارهای middleware ها . و وقتی می آییم اینها رو معرفی می کنیم در واقع از وابستگی معنایی استفاده کرده ایم یعنی شما می خواهید ماژول تان تا آنجا که می شود مستقل کارش را انجام دهد. کل یک مسئولین را که بعهده میگیرد برای انجام مسئولیتهایش متکی به ماژولهای دیگر نباشد که اگر خواستید تغییری در آن مسئولیت بدهید تاثیری روی ماژولهای دیگر نداشته باشد و خود همان ماژول را بتوانید عوض کنید .وقتی شما از frameworkاستفاده میکنید مانند Net framework.

ها. چکار میکنید شما یک abstractی از platformسیستم تان (سیستم عامل و سخت افزار)فراهم میکنید،یعنی سرویسهای رایجی که می خواهد استفاده کند مستقل از سیستم عامل تان باشد.این را همین لایه انجام می دهد.

-اگر سرویس های رایج abstract(انتزاع) شوند تغییر دادن آنها نیازی ندارد به اینکه به ماژول ی که از آن سرویس استفاده می کرده هم تغییر پیدا کند و بعلاوه تغییرات برای ماژولها با استفاده از چنین سرویس هایی تاثیری روی استفاده کننده های دیگر نخواهد داشت . این تاکتیک بنابراین پشتیبانی می کند نه تنها از محلی کردن تغییرات بلکه جزء تاکتیک های جلوگیری از تاخیر هم می شود. دقیقا این نوع تاکتیک داره تاکتیک abstract common service را مثال می زند از دسته تاکتیک های حفظ وابستگی معنایی و زیر تاکتیک این است.

هم برای ما محلی کرددن تغییرا را فراهم میکند و هم جلوگیری از انتشار تاخیر را فراهم میکند.

کل مطلب میگه : یعنی کاری بکن که تا آنجایی که ممکنه یک مسئولیت ،نگاشت کنیم به یک ماژول. اینجوری کاملا تغییرات توی مسئولیت محلی می شود و درون همان ماژولی که باید اینکار را انجام دهد.

دسته بعدی از تاکتیک هایی که است به تغییراتی که تغییرات مورد انتظار را پیش بینی میکنیم.

در نظر میگیریم یکسری تغییرات را از قبل و یک روشی هم برای ارزیابی اینکه چطوری این مسئولیت ها انتساب دهیم برای رسیدن به آن تغییراتی که از قبل پیش بینی کردیم حداقل اثرات را روی تغییرات دیگر بزاریم.اینهم یک روشی است.

- سوال اصلی اینجاست که برای هر تغییر آیا تجزیه پیشنهاد شده محدود میکند مجموعه ماژولهایی که باید برای انجام آن تغییر عوض شوند؟ یادآوری[decomposite ساختار تجزیه ماژول به شیوه code base بودند.وقتی میخواهند قابلیت تغییر را نشان دهید باید ساختار این تجزیه ماژول را جلوی رویمان باشد.]
- درواقع این تاکتیک استفاده از آن سخت است بخودی خود زیرا نمی توانیم همه تغییرات را پیش بینی کنیم به همین دلیل معمولا همراه با دسته تاکتیک وابستگی معنایی استفاده می شود.

سومین تاکتیک—جنرال کردن یک ماژول یعنی یک ماژول را بیایید (برعکس اولی ها که براساس وابستگی معنایی بود) کلی بیان کنید یک ماژول کلی و عمومی داسته باشیم. هرچه ماژول هارا عمومی تر کنید به ما اجازه میدهد رنج بیشتری از محاسبات را براساس ورودیها داشته باشیم پس خیلی کلی باشند به احتمال زیاد اکثر تغییرات روی همان یک ماژول انجام می شود اما ورودی به همچین ماژولی که جنرال شده می تواند بعنوان یک زبان تعریف شده برای ماژول در نظر گرفته شود می تواند به سادگی یکسری پارامترهای ثابت ورودی باشد یا به پیچیدگی یک مفسر باشدورودی آن هم بصورت یک زبان برنامه نویسی می شود .

توضیح : میخواهیم ماژول مان را جنرال کنیم (کلی و عمومی کنیم) تا رنج بیشتری از function ها را انجام دهد حالا این ماژول که کلی شده ورودی به آن طوری تعریف کنید که ورودی ساده ای داشته باشد که یک ماژول ساده می شود و یا ماژول آنقدر پیچیده باشد که ماژول خودش مفسر یا کامپایلر باشد که ورودی به آن یک زبان برنامه نویسی می شود پس اگر یک ماژول قراره خیلی کار را انجام دهد پس باید ورودی هایش هم انواع مختلف باشد پس روی یک ورودی ساده که نمی تواند کارهای زیادی انجام دهد. پس ورودی هاش همباید مجموعه کارها باشد که یک کد باشد و یک برنامه.

چهارمین تاکتیک از انواع تاکتیک های محلی کردن تغییرات: امکانات موجود را کاهش دهیم (یعنی میخواهید صورت مسئله را پاک کنید)

یعنی میخواهید بگویید تغییرات در یک خط تولید ممکنه بیش از حد مجاز باشد و ماژولهای زیادی را تحت تاثیر قرار دهد.پس باید تابیید تغییراتی که امکانش هست اعمال کنی را محدود کنید مثل خیلی از سیستم هایی که ما امروزه داریم یعنی اینکه برای اینکه اثر تغییرات را کم کنیم کلا تغییرات ممکنی که وجود دارد را کم کن و بگو سیستم من تا حد خیلی کم و محدود قابلیت تغییر دارد مخصوصا زمانی که داری خط تولید ایجاد میکنی اینکار را انجام بده.

مثال : یک نقطه تغییر در یک خط تولید ممکنه اجازه بدهد که یک پروسسوررا تغییر دهد . مثلاقبلا در حالت طراحی گفتی که درخط تولید این نقطه ،نقطه تغییر است اینجا را می توانی تغییر بدهی مثلا باید CPU را عوض کنی و حالا بیا optionهای موجود راجع به این تغییر را کم بکن .فرضا نوع پروسسورهایی که باید تغییر بدهی را در یک دسته کوچک قرار بده مثلا بگو cPU هایی از این نوع دسته را اینجا قرار بگیرند.پس option های موجود را کم بکن نه لزوما انواع تغییر های که می خواهی ایجاد کنی به این ترتیب خاصیت Modify ability هم خیلی تحت تاثیر قرار نمی گیرد.

دسته دوم تاکتیک ها تاکتیک جلوگیری از انتشار تغییر است.

یک انتشار تغییر از یک تغییر لزوما نیاز هست تغییرات به ماژول اعمال شوداما بصورت غیر مستقیم.بطور مثال اگر ماژول Aتغییری انجام دهد ممکن است ماژول B هم تغییر کند به دلیل تغییر در ماژول A. زیرا ماژول B به دلایلی وابسته به ماژول A بوده است.

** برای اینکه بتوانیم تاکتیکهایی را معرفی کنیم که از انتشار تاخیرها جلوگیری کنیم یا کمترش کنیم اول از همه انواع وابستگی هایی که دوماژول باهم دارند را یاد میگیریم و بعد می رویم سراغ تاکتیکهایی که در ارتباط با Prevent Ripple Effect هستند را یاد میگیریم.

اول از همه ۸ نمونه از وابستگی ها یی که بین ماژول ها ممکن است وجود داشته باشد را یاد می گیریم (که تاکتیک نیستند بلکه وابستگی هستند).

دسته اول :وابستگی لغوی یا syntax ی.خود وابستگی لغوی دونوع است ۱-وابستگی syntaxی از جنس Data ۲- وابستگی syntaxی از جنس سرویس.

-وابستگی لغوی یا نحوی Data :برای اینکه ماژول B کامپایل یا اجرا شود بصورت صحیح ،نوع یا فرمت داده ایی که تولید می شودتوسط A و استفاده می شود توسط B .باید سازگار باشد با نوع و فرمتی داده ای که ماژول B فرض کرده است.

توضیح:Aداده ایی (مثلا داده عددی از نوعinteger) تولید کرده و B آنرا مصرف میکند. اگر A تغییر کند و خروجی آن integer از ۲ بایت به ۴ بایت تغییر میکند و یا آنرا float کند بطور غیر مستقیم رویB اثر می گذارد.

-وابستگی لغوی از نظر سرویس: سرویس ها signature(امضا) دارند.یعنی روشی برای فراخوانی سرویس و استفاده از سرویس دارند مانند زمانی که تابعی را صدا می زنیدباید بلد باشید که پارامترهای آنرا چگونه set می کنید و به چه ترتیبی بنویسید. این شبیه همون signature می شود. اگر امضا سرویسی که توسط Bاز A ، invoke(درخواست کردن) می شودآنرا از A عوض کنند Bهم دچار مشکل می شود یعنی باید این امضا signature از syntaxاش همانی باشد که B تصورمی کند.

وابستگی دوم که بین دوماژول می تواند وجود داشته باشد وابستگی semanticیا معنایی است که از جهت Data یا از جهت سرویس است. بازهم مانند قبل است ، معنای داده یا سرویسی که A تولید میکند و B مصرف میکند باید منطبق باشد با فرضیات Bدر نظر گرفته است .

نکته مهم : وابستگی معنایی را نمیتوان مشکلش را حل کرد و نمی توان با تمام تاکتیک هایی که تاکنون معرفی شده رفع اشکال کرد. اگر A تغییر کند میاواند روی معنا اثر بگذارد و B را تحت تاثیر قر ار دهد ،مثلا فرض کنید خروجی که A و B می دهد برحسب کیلوگرم بوده حالا شده گرم. این نه فرمت عوض شده نه نوع آن عوض شده بلکه اختلاف وجود دارد یعنی در واقع آنچه که در معنا درک کرده اند تغییر کرده است یا مثلا از نظر سرویس در نظر بگیرید . سرویس اینطوری بوده که میخواهیم ماکزیمم را بدست آورد حالا ممکن است از نظر معنا عوض شده باشد و پیدا کردن ماکزیمم در رنج خاصی بوده و اینها ماکزیمم تو رنج متفاوت دیگری تعریف کرده است یا مصلا مرتب سازی صعودی بوده و نزولی شده است.

دسته بعدی وابستگی ترتیبی یا sequenceاست که دونوع است – داده -کنترل. برای اینکه Bبه درستی انجام شود باید داده ایی که Aتولید وB استفاده میکند باید به همان ترتیبی باشد که B در نظر گرفته است.

مثال: headerیک packet را در نظر بگیرید ،header ابتدا باید بیاید بعد bodyآن .حالا ایت ترتیب اگر عوض شود روی Bکه تحت یک پروتکل با A ارتباط برقرار می کرده و این packetرا مبادله می کرده اثر می گذارد و از نظر کنترل اگر B به درستی اجرا شود باید A ابتدا اجرا شود در یک بازه زمانی تا B بتواند به صحت اجرا شود . تا A اجرا نشود در طی ۵دقیقه ،Bهم نمی تواند اجرا شود. حالا انتشار تغییر چگونه روی وابستگی ترتیبی اثر می گذارد اگر Aرا تغییر دهیم و باعث شود زمان اجرایش تغییر کند و نتواند در یک بازه زمانی اجرا شود B هم نمی تواند اجراشود. مانند دستگاه ATM ،اول ماژول تایید کارت و برقراری ارتباط با بانک مربوطه باید اجرا شود و بعد ماژول محاسباتی اجرا شود بعد اگر در یک مدت زمان مناسبی کارت تایید نشود کارت پس داده می شود و ماژول B اجرا نمی شود.

چهارمین وابستگی ،شناسایی یک interfaceاز A: Aممکن ایت چند اینترفیس داشته باشد برای اینکه Bرا کامپایل کنیم و بدرستی اجرا کنیم شناسایی کردن یعنی اسم یا handleواسطی که بین این دو تا بوده باید مطابق باشد با همان فرضیات B یعنی وابستگی آنها از طریق یک اینترفیس است و واسطی وجود دارد که A و B از طریق آن واسط باهم ارتباط برقرار می کنند.مثلا واسط را می توان هرچیزی در نظر گرفت ساده ترین آن پودتی است که بصورت سخت افزاری وجود دارد که انتظار داریم AوB از پورت فلان USB استفاده کنند ولی حالا تغییر کرده است.

وابستگی پنجم ،مکان A ،در زمان اجرا ،برای اینکه Bبدرستی اجرا شود محل قرارگیری A در زمان اجرا باید با فرضیات B هماهنگ باشد. برای مثال B فرض میکن A قرار گرفته روی فرایندمتفاوتی روی پروسسور مشابه.اگر Bبداند روی پروسسور یا فراین متفاوتی هستند بحث ارتباط بین فرایند هایی که وجود دارد مانند انحصار متقابل و کنترل همروندی و ... مطرح می شودولی اگر نوع وابستگی عوض شود روی یک فرایندباشند بسیاری از این کنترلها نیازی نیست و اگر روی پروسسورهای متفاوتی باشند باز منابع آنها عوض می شوند و خیلی اثر گذار نیست پس مکان A وابستگی ایجاد میکند برای B.

وابستگی بعدی ،کیفیت سرویس/داده است که توسطA فراهم می شود و توسط B استفاده می شود. اگر B بدرستی بخواهد اجرا شود چون Bاز A استفاده میکند کیفیت داده ایی که A در اختیار B قرار میدهد باید مطابق با فرضیات B باشد و گرنه B نمی تواندبدرستی اجرا شود.
مثال : فرض کنید داده ایی که توسط یک سنسور خاصی فراهم می شود باید دقت آن به همان دقت الگوریتمی که روی Bاست و قراره باآن کار کند باشد.بعنوان مثال سیستم های کنترل ترافیک را در نظر بگیرید ، دوربین هایی که سر چهارراه نصب شده اند کیفیت تصویری که این دوربین می فرستد به نرم افزارهای پردازش تصویری که در مرکز کنترل ترافیک وجوددارد باید طوری باشد که بفرض بتواند پلاک خودرورا تشخیص دهد.و اگر این کیفیت پایین باشد و این نرم افزار پردازش نتواند روی آن کار کند مشکل ایجاد می شود.
این وابستگی یعنی وجود A است. برای اینکه B بدرستی اجرا شود باید A وجود داشته باشد ،اول A باشد بعد B . بعنوان مثال اگرB درخواست سرویسی از یک objectمربوط به A را داشته باشد ولی هنوز A وجود نداشته باشد ویا نتواند بصورت پویا آنرا ایجاد کند پس B بدرستی اجرا نخواهد شد.
وابستگی بعدی رفتار منبع A است .برای اینکه B بدرستی اجرا شود رفتار منبع A باید با فرضیات B هماهنگ باشد.این می تواند حتی در مورد استفاده از منبع باشد فرض کنید A از یک حافظه مشترک با B استفاده میکندو اینکه خودش را صاحب منبعی می داند ولی B برعکس فرض می کند که این منبع متعلق به A است یعنی چی رفتار منبه؟ یعنی نحوه استفاده از منبع.آیا منبع را منحصرا مال خود میدانی ؟ یا می دانی از منبع بصورت اشتراکی داری استفاده میکنی .اگر B فرض کند که منبعی را با A بصورت اشتراکی استفاده میکند و A فکر کند این منبع منحصرا متعلق به خودش است ،این اختلاف که بین فرض A وB وجود دارد سبب اشکال می شود و اثر آن روی B خواهد بود.
تا اینجا ۸نوع وابستگی را یاد کرفتیم .حالا می خواهیم تاکتیک هایی را تازه معرفی کنیم که بتوانند با این وابستگی ها را کنترل کنند و کاری کنند که با وجود این وابستگی ها بتوانیم قابلیت تغییر را داشته باشیند ولی از انتشار تغییر جلوگیری کنند و گاهی اوقات وابستگی هارا حذف کنند و با حذف وابستگی ها مارا به حالتی برسانند که این انتشار تغییر را نداشته باشیم.
یادآوری می کنیم هیچکدام از این تاکتیک هایی که میخواهیم بگوییم از تاکتیک انتشار تغییر معنایی (semantic) جلوگیری نمی کنند این تاکتیکها شامل information hiding(مخفی کردن اطلاعات) –حفظ واسط های موجود – شکستن زنجیره وابستگی ها .

مخفی سازی اطلاعات: اینه که تجزیه کنیم مسئولیت های یک سیستم یا موجدیت به قسمتهای کوچکتر و مشخص کنیم کدام اطلاعات publicو کدام private شوند. مخفی سازی اطلاعات شما با آن برخورد کردید و در شی گرایی داریم ،چه کمکی به ما میکند؟ وقتی کلاسی را می نویسید تعدادی از متدها یا داده ها را public و یکسری را private تعریف می کنید و وقتی privateتعریف می کنید چه کمکی به شما میکند این همان مخفی سازی اطلاعات است و متد یا داده ای را مختص خودتان میکنید.حالا در نظر بگیرید بین این وابستگی هایی که وجود دارد وقتی که قسمتی از آنراprivate میکنید در واقع دارید وابستگی را از نظر آن داده یا متد حذف میکنید پس اگر تغییری ایجاد شود این تغییر انتشار شونده نخواهد شد این یک تاکتیک است.
تاکتیک دیگر اینست که اینترفیس ها یی که هستند بزاریم باشند مثلا A وB وابستگی هایی باهم دارند مانند وابستگی syntax ی.یک واسطیرا قرار میدهیم یا اگر از قبل وجود دارد میگذاریم بماند که این واسط تغییر را رفع کن که اعمال شود نثلا اگر syntax ی خروجی A داره عوض می شود این واسطه همین خروجی ها را دوباره تبدیل کند به همان چیزهایی که نوع و فرمت آنها متناظر بوده با همان نوع و فرمت فرضیات B و یا اگر از قبل واسطه ایی وجود داشته اجازه دهیم همینطور باقی بماند. واسطه اگر بماند ولی خود A تغییر کند روی Bدیگه تاثیر نخواهد گذاشت. این تاکتیک لزوما کار نمیکند اگر که B یک وابستگی معنایی با A داشته باشد زیرا تغییرات توی معنای داده/سرویس ،ماسک کردن آنها سخت است . یعنی باید جلوگیری کنید از اینکه معنای داده ایی یک سرویس عوض شود.

تاکتیک سوم : محدود کردن مسرهای ارتباطی .محدود کردن ماژولهایی که باهم داده به اشتراک می گذارند باز اینهم جزء دسته تاکتیک های است که صورت مسئله را داره تغییر میدهد بجای اینکه راه حل بدهد.کاهش دادن ماژولهایی که دارن داده مصرف میکنند توسط یک ماژول معینی و همنین کاهش دادن تعداد ماژولهایی که داده تولید میکنند. اگر اینکار را بکنید باعث می شود اگر یکی از ماژولهایی که تولید کننده یا مصرف کننده تغییربکنند روی بقیه اثری نگذارند البته ماژول مصرف کننده اگر تغییر کند می تواند انتشار تغییر عوض شود. چیزی که قبلا مصرف می کرد نیاز داره عوض شود و تولید کنند هم براساس نیاز مصرف کننده عوض شوند.

آخرین تاکتیک از این دسته : استفاده از یک واسط . اگر که B یک نوعی از این وابستگی ها(۸ تا) بجز sementic به A داره ممکنه قرار دادن یک رابط یا واسط بین A و B بتواند مدیریت کند این وابستگی را و این مشکل را حل کند (شبیه اولی ها حفظ واسط موجود).مانند یک convector یا یک مترجم قرار دهی خصوصا در بحث syntax ی براحتی می تواند تبدیل کند فرضیات داده/سرویس را به فرضیاتی که B دارد.

نکته : این تاکتیک modify ability که استفاده از intermediaryاست کاملا در تضاد است با تاکتیک performance .اضافه کردن یک ماژول بعنی اضافه کردن فعالیت یعنی مصرف کردن زمان و هزینه و افزایش زمان پاسخ و افزایش زمان انتظار.یعنی همه اینها متقابل است با performanxe . شما دارید یک ماژول محاسباتی اضافه می کنید .اینجا می توانیم یادآوری کنیم که تاکتیکهایی که اینجا داریم ممکنه مارا به یک صفت کیفی برساند و ممکنه از یک صفت کیفی دیگر دور کند.

دسته سوم از تاکتیک های modify ability، به تاخیر انداختن زمان انقیاد. دو دسته تاکتیک localize modification و prevent ripple effect به این درد میخورد که شما تعداد ماژولهایی که بصورت مستقیم یا غیر مستقیم تحت تاثیر یک تغییر قرار میگیرند را کم بکنید.اما اگر بخواهید اجازه دهید یک کاربر نهایی در زمان اجرا هم بتوانند تغییرات را انجام دهد از دست تاکتیک های به تاخیر انداختن زمان انقیاد می توانید استفاده کنید. یعنی شما تغییر را انحام می دهید تست میکنید ولی deploy نمی کنید می گذارید deploy و detکردن پارامترها برای انجام deploy بصورت یک تابع آماده شده زمان اجرا تا کاربر نهایی آنرا اجرا کند. سناریو های قابلیت تغییر شامل دو تا المنت می شود که با تاکتیک های کاهش تعداد ماژولهایی که تغییر میکنند به اینها نمی رسیم satisfy نمی شود یکی زمان deploy کردن و اینکه اجازه دهیم غیر توسعه دهندگان هم تغییرات را انجام دهند. دسته تاکتیک هایی که در رابطه با به تاخیر انداختن زمان انقیاد هستند از هردو اینها پشتیبانی می کنند.

تاکتیک های زیادی هستند که می توانند در زمان اجرا یا لودچیده شوند.رجیستری در زمان اجرا پشتیبانی می کند از عملیاتهای plug & play در هزینه سربار اضافی برای .runtime register . در واقع تنظیم میکند چه سرویسی در زمان اجرا به ما داده شود.

Runtime registration این تاکتیک می گوید پشتیبانی می کند از عملیاتهای plug &play در هزینه سربار اضافی جهت مدیریت کردن registration یا همان رجیستر کردن publish/subscribe ها برای مثال می توانیم پیاده سازی کنیم در زمان اجرا یا زمان لود.

با معماری publish/subscribe با معماری plug &play که میگویند publish/subscribe یکک نمونه از plug &play است آشنا هستید[این معماری که امروزه مد شده است ودر بحثهای اینترنت از آن استفاده می کنیم در بین سرویس ها یک باس نرم افزاری وجود دارد و یکسری از منتشرکننده ها که دارن به آن گوش میدهند، بعد نودهایی وجود دارند که علاقه مندی های خودشان را میگذارند روی این باس و میگویند مثلا آموزش زبان خاصی را می خواهیم ،من فلان فیلم را می خواهم ببینم و مثل اینها. علاقه مندی های خودشان را روی این باس نرم افزاری می گذارند و این شنونده ها که می توانند خودشان انتشاردهنده هم نباشند گوش می دهند و درخواستی یا علاقه مندی خودشان را می رسانند به انتشار دهنده خاصی که وجود دارد که نام این را می توان یک اپلیکیشن تحت وب گذاشت که می تواند این کار را برای شما انجام دهند و اینه چکار میکنند ؟شما در زمان اجرا تنظیم میکنید که چه سرویسی باید به شما داده شود و چگونه برای شما اجرا شود.در واقع مثل یک نرم افزار واسطیانگار وجود داشته باشد که تمام این انتشاردهنده بعنوان یک component(مولفه) برایش باشد ولی همه اینها را از ابتدا اجرا نمیکند،میبیند که کاربر چی می خواهد و بعد مولفه مورد نظر را اجرا می کند این روشی است که از این معماری اسفاده میکنند.]

برای اینکه deffer bindingفراهم کنیم مثلا می توانیم از configuration file ها استفاده کینم .نرمافزارهایی که فایل پیکربندی دارند را دیدید اینها فایل متنی دارنند که داخل آن تنظیم میکنید که نرم افزار شما با چه مولفه هایی اجرا شود .مانند نرم افزار های شبیه ساز مانندNS2که یک فایل متنی دارند و داخل آن تنظیم میکنند که شبکه ایی که داریدشبیه سازی میکنید با چه الگوریتمی یا پروتکل مسریابی اجرا شود تعداد نودها ،سرعتش چقدر باشد اگر شبکه وایفای است رنج انتقال نود ها چقدر باشد و... پس کلا با یک فایل پیکر بندی تنظیم می شوند.که سیستم چطوری رااندازی شود یعنی bind کردن مولفه ها ی یک سیستم را به تاخیر انداختید تا زمان اجرا توسط کاربر پایانی.

-آنهایی که می توانند مولفه هاراجابجا کنند ،چیدمان مولفه ها را تنظیم کنند باید در زمان لود باشند.

-امکان اضافه کردن یا تعریف کردن پروتکل های اضافی را در زمان اجرا می تواند وجود داشته باشد.مثلا نرم افزارها و اپلیکیشن ها این امکان را فراهم میکنند که بتوانیم حتی پروتکل ارتباطی را خودمان تعریف کنیم اگر اینکار را بکنیم یعنی بر اساس یک پروتکل ،یا تبعیت از یک پروتکلخاصی که خودمان تعریف می کنیم پردازش انجام شود.

تاکتیک های performance: سومین صفت کیفی که باهم مطالع میکنیم . در تاکتیک های کارایی (performance) تولید کردن پاسخ به رویدادهای ورودی است در یک زمان معین .کلا هر وقت از کارایی حرف میزنید از کلمات زمان پاسخ ،زمان انتظار یا میزان تاخیر و trueput یعنی تعداد عملیاتهایی

که در واحد زمان انجام می شود، روبرو می شویم و کارایی رراجع به اینها صحبت میکند. [در فصل قبل یاد گرفتیم که کارایی با Eventها سروکاردارد کهیا بصورت پریودیک و ...وارد سیستم می شود و ما باید به آنها پاسخ دهیم بطوری که بتوانیم حداقل زمان پاسخ و حداقل زمان انتظار و حداقل trueput را داشته باشیم]

یادآوری : اینکه Eventها می توانند چه چیزهایی باشند: تک تک یا بصورت دسته ای وارد سیستم می شوند ،می توانند پیام باشند،انقضای یک فاصله زمانی باشند،یا شناسایی یک تغییر حالتی باشند یا هرتغییری می تواند یک رویداد باشد.

تاکتیک کارایی : کنترل میکند زمانی را که ما باید پاسخ دهیم یا یک پاسخ تولید می شود.

Latency: مدت زمان بین ورود یک رویداد و تولید یک پیاصخ است که شما می خواهید latencyیا درنگ کم باشد.

تاکتیک های کارایی : بعد از اینکه Eventی واردسیستم شود یا سیستم آنرا پردازش می کند یا به دلیلی بلاک میکند که این منجر به دو مشارکت اصلی یا بحث اصلی در مورد زمان پاسخ می شود .یعنی اینکه منبع مصرف شود یا نه و زمان بلاک داشته باشیم یا نه ،اینها روی زمان پاسخ اثر می گذارند.یا به زبان سادهتر گفته شود : زمان پاسخ از چه چیزهایی تشکیل شده؟ از مدت زمانی که شما صرف می کنید تا محاسبه ایی را انجام دهید یا پاسخ رویدادی را بدهید یعنی مدت زمانی که طول می کشد تا از منابع استفاده کنید تا پاسخ رویداد را بدهید بعلاوه مدت زمانهایی که مجبورید به دلایلی زمانهای تعلیق یا بلاک شدن را داشته باشید .پس زمان پاسخ =مدت زمان مصرف منبع + زمان بلاک یا انتظار.

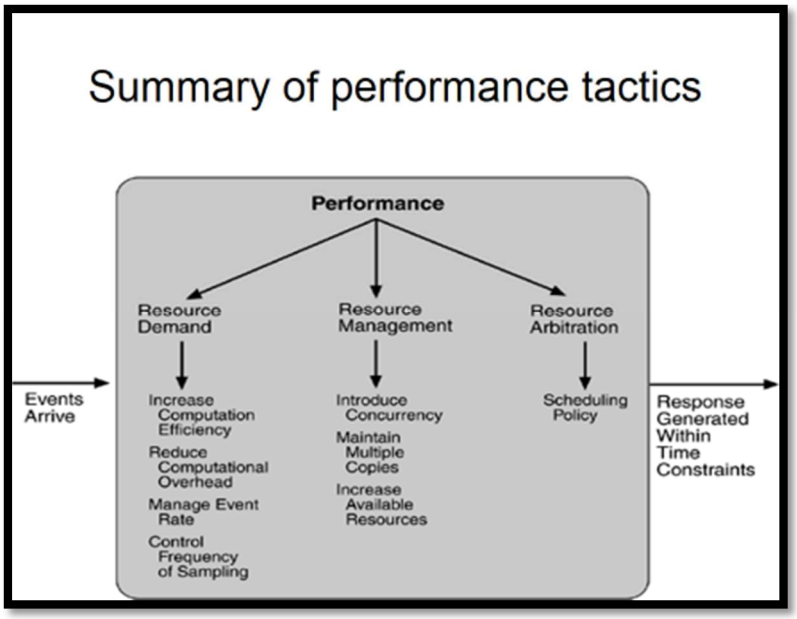
مصرف منبع :منبع شامل CPU ، مخازن نگهداری داده ، مسیرها یا پهنای باند شبکه ، حافظه و هر موجودیت دیگری که توسط سیستم دیگری طراحی شده است بطور مثال یک بافر می تواند منبع باشد که باید مدیریت شود و دسترسی آن در ناحیه بحرانی ،مثلا دسترسی اشتراکی روی آن موجود دارد که یک ناحیه بحرانی اینجاد می شود که باید انحصار متقابل روی آن اعمال کرد تا یک دسترسی ترتیب داشته باشیم.

Block time: بلاک تایم یک محاسبه ممکن است بلاک شود برای استفاده ایز یک منبعی .به دلیل اینکه یا ازدحامی روی آن منبع است (چندین رویداد منتظر استفاده از آن منبع هستند) یا ممکن است به این دلیل بلاک رح دهد که منبع در حال حاضر درسترس نیست یا محاسبات وابسته به بکسری رویدادهای دیگر است که هنوز آماده نیستند و باید منتظر بمانیم . پس ما چندین دلیل داریم برای اینکه بلاک تایم داشته باشیم و هر دلیل باعث می شود یک مدت زمانی که از این بلاک تایم ها داشته باشیم و همه اینها نهایتا روی latency ما اثر خواهد داشت. ازدحام برای منبع ،منابع ممکن است بصورت تکی یا چند تایی وارد شوند وقتی رویدادی بصورت چند تایی وارد می شوند و می خواهند یک منبعی را بگیرند می گوییم ازدحام (contention) یا رقابت روی آن منبع وجود دارد.

دلیل بعدی در دسترسی به منبع: اگر ازدحام نباشد محاسبات ما ممکن است انجام نشود اگر منبع در دسترس نباشد .چرا منبع دردسترس نباشد؟ممکن است منبع آفلاین باشد یا یک failer برای آن رخ داده باشد و دلایل دیگر .که معمار آن را شناسایی باید بکند و دوباره به مشارکت برگرداند تا بتوانیم کمتری Latency را داشته باشیم. که قطعا از همان روشها و تاکتیک هایی که برای abilityها یادگرفتیم روی آن پیاده می کنیم.

دلیل بعدی وابستگی محاسباتی ، مثلا یک محاسبه ممکن است منتظر بماند بخاطر اینکه داره بصورت همروند و همزمان از نتیجه محاسبه دیگر استفاده میکند. بعنوان مثال خواندن اطلاعات از دومنبع متفاوت اگر که دومنبع به ترتیب باید خوانده شوندیک Latency که وجود دارد قطعا بیشتر از وقتی خواهد بود که دومنبع بصورت موازی باید خوانده شوند.

کلا با این توصیفات با یان background ی که داریم سه دسته تاکتیک باید معرفی کنیم برای کارایی مان. کارایی راجع به زمان پاسخ صحبت میکند و زمان پاسخ هم تحت تاثیر Latency است و کلا Latency هم یعنی اولین فاصله درخواستی که وارد می شود تا اولین پاسخ که داده می شود میخواهد این را کم کند . Latency برابر مدت زمان مصرف منابع + مدت زمانهای بلاک شدن.



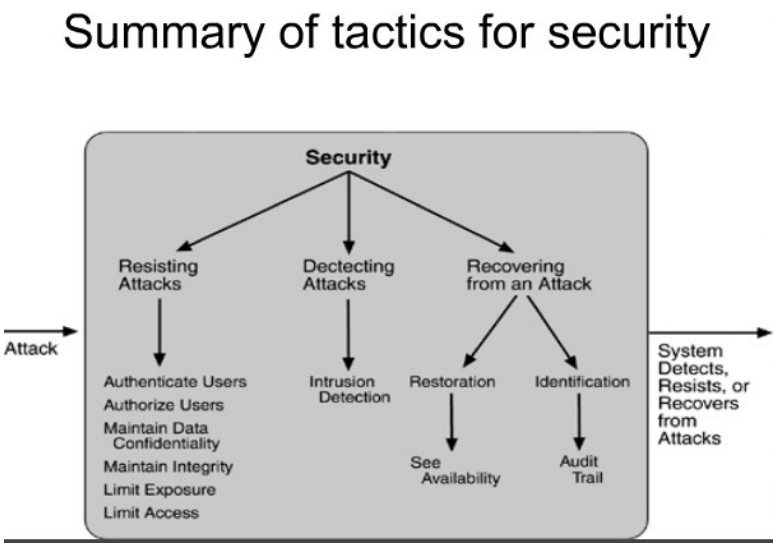
سه دسته تاکتیک کارایی را در این اسلاید می بینید.یک رویداد یا دسته ای از رویدادها که وارد می شوند ،یک دسته از این تاکتیک های کارایی درخواست منبع (resource Demand) است و دسته دیگر مدیریت منبع (resource management) و دسته دیگر داوری منبع (resource arbitration) . دسته اولی روی نحوه درخواستها اثر میگذارد مدل درخواستها چزوری باشد تا Latency را کم کنیم . دسته دومی روی خود منابع اثر می گذارد که چطوری از آنها استفاده کنیم.و دسته سوم مربوط به الگوریتمهای زمانبندی است.

درخواست منبع : دسته ای رویداد ها مبدا درخواست منابعی هستند. اینجا دوویژگی وجود دارد :۱-زمان بین ورود رویداد ها در یک جریان است. ۲-چقدر منبع به ازای هر درخواست باید مصرف شود .
برای کاهش Latency که هدف اصلی ما در این مقوله است ۳ تا تاکتیک داریم: ۱-منابع مورد نیاز برای پردازش رویدادها را کم کنیم .با کم کردن میزان ممنابع استفاده شده توسط دسته ای از Eventها، میتوانیم Latency را کم کنیم.حالا چطوری می توانیم اینکار را انجام دهیم . یا میتوانیم کارایی محاسباتی مان را افزایش دهیم . مثلا از سمافور استفاده کنیم وقتی از سمافور استفاده میکردیم که روی ناحیه بحرانی عمل میکردندو دوتا دستور داشت signal و wait .وقتی یک فرایندوارد ناحیه بحرانی می شد و اگر فرایند دیگری در داخل ناحیه بود باید آنرا waitمی کرد و وقتی بیرون و آمد signalمی داد تا بقیه بیدار شوند تا بتوانند وارد ناحیه بشوند.دقیقا در مدیریت منابع داشتیم .حالا اگر بتوانیم از الگوریتم های قویتر سمافور استفاده کنیم میتوان کارایی محاسباتی را افزایش دادو Latency را کاهش.
-کاهش دادن سربارهای محاسباتی. می توانی از واسطه استفاده نکنی . در تاکتیک های قبلیModify ability از واسط استفاده میکنیم تا وابستگی ها حفظ شوند ولی اینجا میگوییم واسط نگذاری. خوب واسط خودش یک منبع است و واسط نگذاریم یعنی میزان استفاده از منبع کم می شود.و وقتی جریانی از رویدادها وارد می شود منبع کمتری مصرف می شود. و در نتیجه Latencyکاهش می یابد که این نقطه تضاد با فبلی هاست.
دسته تکتیک بعدی میگوید تعداد رویدادهایی که باید پردازش شوند را کم کن. دسته قبلی میگفت تعدادمنابع را برای استفاده کم بکن .میزان استفاده از منبع را کم بکن اینجا می گوید خود رویدادهارا کم کن .چطوری میشه ؟ با مدیریت نرخ رویداد.این کنترل کردن فرکانس های نمونه برداری جزء مدیرت نرخ رویدا د می باشد مثلا یک شبکه سنسور را در نظر بگیرید کار این شبکه چیه؟ یک سری سنسور توی محیط وجود دارد که تعداد عوامل را حس (sence) میکند و بعد برای یک کوردیناتور یا همونsyncارسال می کنند حالا تعدادنمونه هایی که ارسال می شود رویدادهای ما میشود که باید sync اینها را تجمیع کنند تا بتواند اختلالات موجود را شناسایی کند یا اینکه تشخیص دهد که دما بالا رفته یا باد می آید یا نمی آید بستگی دارد به اینکه سنسور شما چه باشد و چهحسی را بیرون بکشداگر فواصل زمانی سنسور ها را حس می کنند و داده ها را می فرستند کم بکنیم یعنی نرخ رویدادها را مدیریت کردیم ،بسته به اهیمت اصلاعاتی که میخواهی از محیط بگیری و احتمال وقوع رویدادی که منتظرش هستی می توانی نرخ نمونه برداریها یا فرکانس خود نرخ نمونه برداریها و تعداد دفعاتی که نمونه برداری انجام می دهد را مدیریت کنی مثلا اگر جایی هست که سیستم کنترل احتراق دارد اگر احتمال آتش سوزی در آن زیاد است مثلا دیگ بخاری در آن هست تعداد دفعاتی که دما و فشار را حس میکند بیشتر از موقعی است که در ساختمان مسکونی احتما حریق را چک میکند.
-تاکتیک بعدی کنترل استفاده از منبع است که برای این کار روی میزان استفاده از منبع کار میکنی .یکی اینه که بیایید زمان اجرا را کم کنی و مدت زمانی که هررویداد از منبع می خواهد استفاده کند اینمدت زمان را محدود کنی و سقفی برایش بگذاری یا طول صف انتظار برای دریافت منبع را محدود بکنی اینطوری می توانی Latencyرا کم بکنی.
مدیریت منابع: اگر درخواست منابع کنترل نشوند با تاکتیکهای فبلی (resource demand) شاید بتوانی از این تاکتیک های برای کنترل استفاده کنی .
-معرفی همروندی .کلا همروندی را معرفی کنی می توانی روی Latency کار بکنی یعنی اجرای لا به لای هم فرایندها و رویدادها باهم می تواند به کاهش زمان انتظار و Latencyکمک بکند.
-حفظ چندین کپی از داده یا محاسبات.اینجا اگر از عهده درخواستها بر نمی آیی منابع را زیاد بکن تا Latency کم شود. چندین کپی از آنها داشته باش تا استفاده موازی داشته باشیم و استفاده موازی سبب کاهش زمان انتظار و Latency می شود.
-میزان در دسترس بودن منابع را افزایش بده.باید از آن تاکتیهای abilityمانند hot restart استفاده کن و درجه ability را بالا ببر . اینجوری همیشه منابع در دسترس هستند در نتیجه میزان بلاک تایم کم می شود و در نتیجه Latency هم کاهش می یابد.

دسته تاکتیک داوری منابع: این تاکتیک به درد زمانی میخورد که ازدحام در دسترسی به منابع وجود دارد. اگر ازدحام وجود دارد یعنی چندتا هستند که می خواهند این منابع را بگیرند می توانید استفاده از منابع را زمانبندی کنید .چندین رویداد هم در گرفتن منابع باهم رقابت می کنند منابع مانند پروسسورها ،CPU ها ، بافرها، شبکه .
این تکنیک دو قسمت دارد یک اولویت دهی میکند و دومی بر اساس اولویت ،زمان استفاده از منبع را توزیع می کند این اولویت دهی ها هم خودش بصورت ثابت داده میشود یا بصورت پویا.

رایج ترین سیاستهای زمانبندی را معرفی میکند.۱-سیاست fifo که اولین ورودی ،اولین خروجی است. رویدادهای ما به ترتیب وارد می شود و به همان ترتیب ورودی اولویت میگیرند و به آنها پاسخ داده میشوند. ۲- سیاست اولویت ثابتاست که یک اولویت داده می شود و به غیر از اولویت زمان ورود در fifo .درخواستها را مرتب میکند و براساس اولویت آنها را داوری میکند. ۳- اولویت اهمیت معنایی . براساس مشخصات دامنه اولویت داده می شود.اگر این درخواستها از یک دامنه خاصی باشند از اولویت بالاتری برخوردارند مثلا اولویت درخواست مدیریت از درخواست معاونت بالاتر است و زودتر به پاسخ داده می شود. ۴- بر اسا مدت زمان deadline که برایشان تعریف شده هرکدام که خط مرگ کوتاهتری دارند از اولویت بالاتری برخوردارند مثلا فرایندهایی که تا ۵ ثانیه اگر اجزا نشوند دیگر بدرد نمی خورند.یعنی یک مدت زمان deadline دارند که در سیستمهای بلادرنگ مورد استفاده دارند.یعنی هرکدام که زمان مرگ کوتاهتری دارنداولویت بالاتری دراند. ۵- بر اساس مدت زمان استفاده(نرخ استفاده) اولویت دهی میشود یعنی هرکدام که مدت زمان اجرایش کوتاهتر باشد اولویت بالاتری دارد.

اولویت دهی به دوصورت انجام میشه یک ایستا که تا حالا همگی بصورت ایستا هستند یعنی از همون اول وارد سیستم می شوند اولویتی میگیرند و تا به آخر همون اولویت را دارند و عوض نمی شوند. اما اگر اولویت دهی پویا باشد یعنی در مدت زمانی معینی اولویت دهی تغییر میکند ممکن است پس از مدت زمانی اولویت آنها کمتر یا بیشتر شود/ که معروفترین آنها الگوریتم Round Robin است که برش زمانی داره اولیت یک رویداد در برش زمانی تغیین می شود. مثلا اولویت اینطوری است که هر رویداد مثلا دوبرش زمانی فرضت دارد که از CPU استفاده کند که مدت زمان تمام شد نوبت بعدی می شود اگر کارش تمام شده باشد بیرون میرود وگرنه در انتهای صف قرار میگیرد تا دوباره نوبتش برسد. یا الگوریتمی که deadlineآن زودتر فرا برسد آنرا در برش های زمانی محاسبه می کنید که ممکن است با گذشت زمان deadlineبعضی ها نزدیکتر شود پس به آنهایی که منتظر هستند اولویت میدهیم..



۶ تا صفت کیفی اصلی سیستم داریم که باید برای پیاده سازی آنها روشی داشته باشیم و یک طراحی انجام دهیم برای اینکه مابه امنیت برسیم چه کارهایی باید انجام دهیم ؟ در واقع در security صحبت از attack و حمله است و ما باید یک پاسخ برای آن پیدا کنیم که به سه دسته تقسیم می شوند :

۱)یا در برابر حمله مقاومت میکنیم ۲) یا می گذاریم حمله انجام می شود و بعد شناسایی اش می کنیم ۳) چگونه حمله انجام شده را بتوانیم در سیستم خودمان recovery کنیم.برگرداندن به اخرین حالت امن که بوده است.

همه این سه دسته تاکتیک ها مهم هستند برای مثال وقتی که ما یک قفلی را روی درب منزل می گذاریم که در برابر حمله مقاومت کنیم ولی در نهایت دزد وارد منزل ما می شود وقتی که ما منزل باشیم می توانیم همان موقع دزد را بیابیم و شناسایی کنیم یا اینکه دزد می آید حمله می کند و بعد باید کاری انجام دهیم که این دزدی را جبران کنیم مثلا اگر قبلا بیمه اموال کرده باشیم این دزدی بیمه می شود .

تاکتیک ها به سه دسته تقسیم می شوند .

۱) مقاومت در برابرحمله
۲) شناسایی حمله
۳) بهبود یا ترمیم حمله

۱. مقاومت از حمله

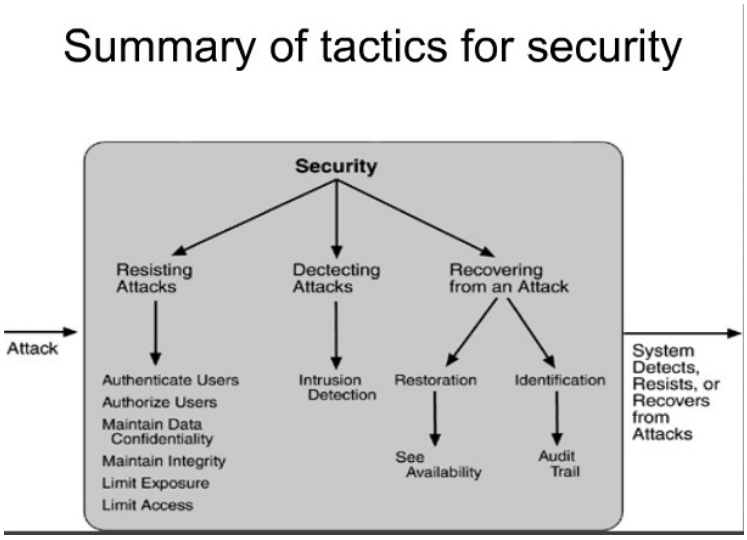
- در مقاومت در برابر حمله انواع تکنیک های امنیتی برای اینکه جلوی ورود حمله را بگیریم مانند تائید هویت کاربران است یعنی اینکه ما مطمئن شویم که کاربر یا کامپیوتری که وارد سیستم ما می شود همان شخص موردنظر ما است یا خیر.

می توانیم پسورد بگذاریم ، امضاء دیجیتال ، شناسایی های بیومتریک مانند اثر انگشت ، عنبیه و ....

- Authorize برای زمانی استفاده می شود که یک کاربر تائید هویت شده داریم اما می خواهیم دسترسی های آن را محدود کنیم توسط الگوهای کنترل دسترسی که می توانیم به کاربران بدهیم یا به کلاسی از کاربران که تمامی این کارها توسط ادمین سیستم انجام می شود درواقع همان تعیین سطح دسترسی

-حفظ قابلیت اطمینان داده ها که اگر حمله رخ داد ازمرز تعیین هویت گذشت و از سطح دسترسی هم رد شد به داده رسید نتواند به داده دسترسی پیدا کند چطوری می توان این کار را کرد با رمزنگاری روی داده و مسیرهای ارتباطی که بهترین حالت آن همان رمزنگاری عمومی است که کلید خصوصی وعمومی است مثلRSA

-حفظ یکپارچگی داده است به همان ترتیب، کیفیت، و چیزی که تولید شده است همان طوری که تحویل داده شده است ما می توانیم از افزونگی اطلاعات مثل checksum یا result hash که می تواند هم وابسته به داده اصلی رمزگذاری شود و هم می تواند مستقل از داده اصلی باشد این نوع رمزگذاری را انجام دهیم بر طبق شناخت خطا مثل بیت های پریتی، یا check sum مثلا براساس یک تابع hash در مبدا که ان را بهم ریخته می کند و یک کد تولید می کند و دوباره در مبدا نیزمعکوسش را انجام داده چیزی مثل رمزنگاری است.



- محدود کردن ارائه است در واقع می گوید حمله وقتی رخ می دهد که یک نقطه ضعفی در سیستم وجود داشته باشد که از این نقطه ضعف حمله کننده وارد شود و داده و سرویس های ما را تحت تاثیر می گذارد در اینجا ما میزان داده ها و سرویس های که available هستند را به حداقل می رسانیم تا حد امکان نقطه ضعف هایی که وجود دارد را کم بکنیم معماری ما بگونه ای باشد که سرویس ها را به هاست ها اختصاص بدهد که سرویس ها محدود باشد و دسترسی به هاست ها انقدر کم باشد که کمترین نقطه ضعف را داشته باشیم.

- firewall همان دیوار و خط اتش است که بتواند دسترسی یا پیام هایی که از هر مبدا می آید را کنترل کند پیام هایی که از منابع ناشناخته می آید شاید یک حمله باشد که جلوی آن را می گیریم تازمانی که مطمئن شویم که یک منبع شناخته شده است مثل ویروس کش ها که شدیدا امکان حمله را کم می کند اما availability را کاهش می دهد.

۲. اگر حمله رخ داد چه کنیم

غالبا شناسایی حمله با الگوهای ترافیکی شبکه از طریق شناسایی نفوذ است یا الگوهای استفاده در پایگاه داده که از آنها کمک گرفته و استفاده می کنیم وقتی که شناسائی می کنیم که یک سوء استفاده ای رخ داده است می آییم الگوی ترافیکی خودمان را مقایسه می کنیم با الگوهای تاریخی یا قدیمی حمله های شناخته شده و اگر ببینیم که مشابه بودند مقایسه می کنیم ما هم احتمال این را می دهیم که شاید سوء استفاده ای در سیستم ما رخ داده است.

در موردی که anomaly را می خواهیم شناسایی کنیم (حالت غیرمعمولی) الگوی ترافیکی را مقایسه می کنیم با یک الگوی ترافیکی سابق خودمان که مطمئن هستیم مشکل و سابقه ای وجود نداشته است از ان و سیستم در حالت سالم کارخودش را انجام داده است اگر با الگوی ترافیکی حملات شناخته شده باشد حملات خاص را شناسایی می کنیم.

بعضی مواقع است که حملات جدید است و نمی توانیم نفوذ را شناسائی کنیم اول باید anomaly را شناسائی کنیم و براساس آن recovery را انجام بدهیم و بعد حمله های شناخته شده را معرفی کنیم.

۳. Recovering from an attack

خود به دو دسته تقسیم می شود :

۱)restoration ۲)identification

حالت اول زمانی است که حمله رخ داده و تمام شده است اولین کاری که ما انجام می دهیم این است که برمی گردانیمش به حالت قبل از حالتی که تغییرات حمله خنثی کنیم.

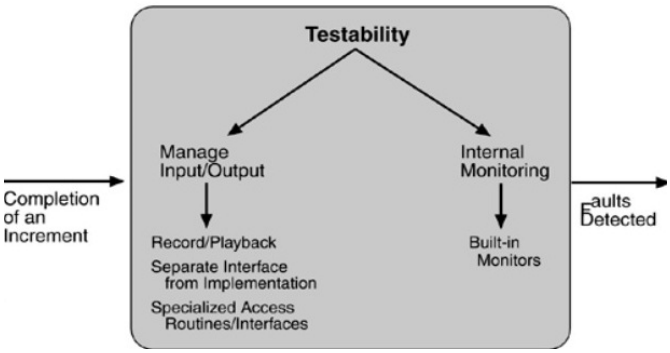
حالت دوم شناسایی اینکه تغییرات رخ داده است و اینکه چه کسی این حمله را انجام داده که از این به بعد پیشگیری کنیم و دنباله ای از کارهایش را ضبط کنیم که اگر بار دیگر وارد سیستم شد بتوانیم تشخیص بدهیم.

بعد از وقوع حمله و زمانی که ما شناسائی اش کردیم در حالت اول دسته تاکتیک هایی استفاده می شود که برای برگرداندن به حالت قبل از شناسائی حمله که الگوهای آن برای این کار به صورت زیر می باشد.

وقتی که حمله رخ می دهد می توانیم از تاکتیک availability استفاده کنیم که مشکل رفع شود ما می توانیم در این تاکتیک از restoreکردن سیستم با داده برای حالت سازگار و ایستا قبلی استفاده کنیم مثل check point یا rollback بصورتی که مولفه ای که خراب می شود بعد به حالت قبلی برمی گردانیم و تنها تفاوتش این است که ما مجبوریم یک هزینه ای داشته باشیم برای حفظ کپی های افزونه مثل پسورد، لیست کنترهای دسترسی کاربران، سرویس های domain و داده های پروفایل های کاربران، باید یک اطلاعات امنیتی را داشته باشیم.

یک تاکتیکی که برای شناسئی یک حمله کننده استفاده می کنیم که به آن اصطلاحا audit trail می گویند در واقع حفظ کردن دنباله ای از کارهای حمله کننده است که در واقع یک کپی از تمامی تراکنش هایی که روی داده اعمال می شود به علاوه اطلاعات شناسائی را ثبت می کنیم و این اطلاعات را می توانیم برای ردیابی فعالیت یک حمله کننده استفاده کرده و پشتیبانی کنیم از حملات غیرقابل انکار، یک کاربر قانونی که یک کار را انجام داده است و الان انکار می کند که این کار را کرده است یا خیر و همچنین برای پشتیبانی از ترمیم سیستم نیز مفید است.

Summary of testability tactics



تاکتیک های قابلیت تست به دو دسته تقسیم می شود:

۱)ورودی و خروجی ها را کنترل می کند ورودی تولید می کند برای تست کردن و خروجی می گیرد.

۲) استفاده از ماژول ها و نرم افزارهای خاص برای مانیتورکردن یا ردیابی کردن سیستم سابق یا سیستم جدید که بتوانند مقایسه انجام دهند شبیه anomaly است که ببینیم فعالیت درست انجام شده است یا خیر.

این تاکتیک ها برای این است که ما بتوانیم هزینه های تست در نرم افزار را کاهش دهیم باید از ابزارهایی استفاده کنیم مثل تست یکپارچگی، یا کلی سیستم، تست ماژول ها که درواقع هدف این است که انجام تست و توسعه افزایشی نرم افزار آسانتر شود، وقتی که ما مولفه ای را تست می کنیم هر مولفه را به هم وصل می کنیم و بعد تست را انجام می دهیم که نیاز داریم به یک سری از ابزارها که ورودی را بگیرند و سیستم را یک به یک داریم بزرگتر می کنیم.

که در واقع ما به یک teat harness نیاز داریم که یک سری از نرم افزارهایی است که برای ما ورودی تولید می کنند و تست را انجام می دهند و خروجی آن را بعد از تست ثبت می کند ما در نظر نمی گیریم طراحی و تولید teat harness را، ما فرض می کنیم که teat harness وجود دارد و کار خودمان را انجام می دهیم. به دو دسته تقسیم می شود.

۱) تاکتیکی که ورودی تولید می کند و خروجی ثبت می کند.

۲) تاکتیکی که از internal monitoring برای انجام تست استفاده می کنیم.

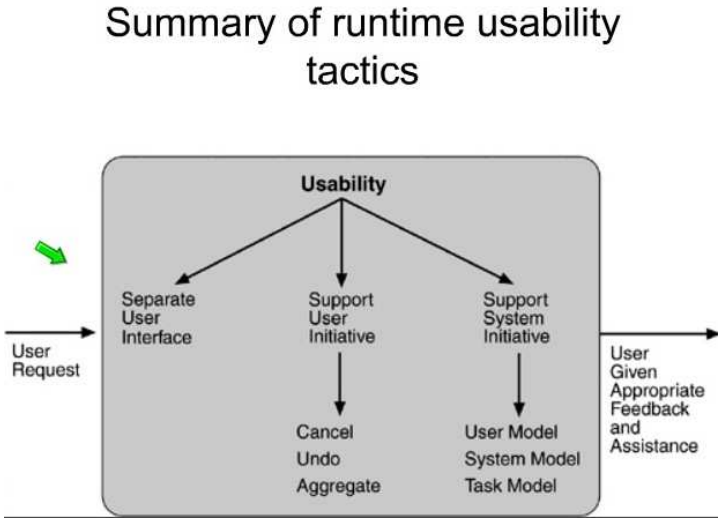
در این دسته تاکتیک های هستند که ورودی تولید می کنند و خروجی ثبت می کنند. که خود به سه نوع تقسیم می شوند.

۱,۱)records/playback اولین مشکل این است که با چه داده ای تست را انجام بدهیم در واقع اشاره دارد به ثبت کردن اطلاعاتی که می گذرد از طریق یک رابط گرافیکی کاربر و از همان برای ورودی اطلاعات تست استفاده می کنیم مثلا یک فرم ورود اطلاعات است در سیستم که سابقه یک مدت کار است اطلاعات را ثبت می کنیم و بعد می دهیم برای تست نرم افزار که teat harness این کار را برای ما انجام می دهد.

۲,۱)جداسازی interface از پیاده سازی است ما می توانیم واسط و رابط را از پیاده سازی جداکنیم کل سیستم درست کار نمی کند و ما مطمئن نیستیم که یک واسطی تولید کند شبیه واسط و به جایی وصل نباشد این را در اختیار کاربران قرار می دهد و شروع میکند به وارد کردن داده از داده هایی که از طریق interface ثبت شده است و از انها استفاده می کند و انها را داخل teat harness می ریزیم، مزیت این روش این است تست های مختلفی را می تواند انجام دهد وقتی این روش برای زمانی استفاده می شود تا وقتی اطلاعات ورودی ما وارد شود ما به پیاده سازی تست های دیگر سیستم می پردازیم.

۳,۱)خاص کردن مسیردسترسی: اگر یک واسط تست خاصی داشته باشیم که بتواند ثبت کند یا مشخصه بندی مقدارهای مختلفی برای یک مولفه از طریق teat harnessو مستقل باشد از اجرای نرمال، یعنی اینکه ما یک داده ای داریم که از قبل آماده شده است و جدا شده از داده هایی که قرار است بعدا وارد سیستم شود ولی ازجنس انها باشد. مثلا یک سیستم role base داریم از طریق همان metadata را برای ورودی تست استفاده می کنیم از ورودی های داده که وجود دارد استفاده می کنیم.

۱,۲) مانیتورینگ داخلی : یک مولفه می تواند تاکتیک های مبتنی بر وضعیت داخلی را پیاده سازی و پشتیبانی کند از طریق پروسه تست مانیتورهایی که درون مولفه جاسازی شده اند واسط هایی هستند که به عنوان مونیتور کار می کنند.
در واقعی این کامپوننتی است به اسم مانیتور که می تواند حالت، ظرفیت،امنیت، کارایی و دیگراطلاعات را از طریق یک واسط نگهداری کند که واسط می تواند دائمی یا بصورت موقت از طریق تکنیک های برنامه نویسی یاز ازطریق یک ماکروپیش پردازنده ایجاد شده باشد.کاراین مولفه این است که تست را انجام می دهد همیشه می آید کنترل می کند وضعیت های سیستم را بار کارایی ظرفیت و امنیت و .... را ثبت کنترل می کند.
این تاکتیک ها به دو نوع تقسیم می شوند نوع اول تاکتیکی است که دسته زمان اجرا هستند که پشتیبانی هایی است که سیستم برای کاربرش تولید می کند در طول زمان اجرا و دسته دوم تاکتیکی است که مبنی بر طراحی واسط کاربر و پشتیبانی از توسعه واسط درزمان طراحی است.
در واقع اینکه سیستم شما چه مواردی را می تواند طراحی بکند برای کاربر و چه کارهایی می تواند انجام دهد و چقدر به کاربر کمک می کند . چقدر خطاهایی که انجام می دهد را تشخیص می دهد که همان فراهم کردن قابلیت اطمینان است برای کاربر در زمان انجام کار.
یک سری تصمیمات است که برای کاربر در زمان طراحی می گیریم و یک سری کارهایی که موقع اجرا فراهم می کنیم که کاربر در زمان اجرا بتواند استفاده کند.



حالت دو وسه درشکل برای زمان اجرا است و حالت یک برای زمان طراحی
تاکتیک های زمان اجرا وقتی است که راه اندازی توسط کاربر است یعنی ما کاری کردیم که کاربر خودش از طریق اجرای یک سری دستورات کار خودش را انجام دهد.کاربر اطمینان داشته باشد اگر اشتباهی انجام داد کارش را مجدد می تواند انجام دهد. cancel، undo
Aggregate: یعنی اینکه ما می توانیم قبلا داده هایی که وارد کردیم را با تغییرات جدید وارد کنیم یعنی جمع آوری تمام داده هایی که وارد شده اند. خیلی ها می گویند که این روش امنیت را پایین می آورد برای اینکه ما می توانیم از اطلاعاتی که قبلا کس دیگری وارد کرده است در سیستم استفاده کنیم.
اگر فرد راه انداز سیستم باشد برای پشتیبانی سه کار را انجام
می دهد.

- درحفظ کردن یک مدل از وظیفه است که از یک مدلی استفاده می شود برای شناسائی متن که سیستم برخی ایده ها را داشته باشد که کاربران چگونه کار می کنند و چطور.

بطورمثال دانستن اینکه جملات معمولا با حروف بزرگ شروع می شوند این اجازه را به ما می دهد که یک application اصلاح کند کلمات کوچک را که شروع می شوند . اگر یک وظیفه طوری تعریف شده باشد که یک متن می خواهیم بنویسیم قانون این است که اول پاراگراف یا اولین حرف هر کلمه در یک جمله با حروف بزرگ باشد در واقع یک مدل می نویسیم و از ان استفاده می شود وقتی که یک کاربر با حروف کوچک می نویسد حروفش را طبق تعریف قبلی خودش درست می کند.

- یک مدل از خود کاربر بسازیم یک مدل حفظ و ایجاد می کنیم که رفتار، دانش و زمان پاسخ های مورد انتظاریک کاربر یا گروهی از کاربران را ایجاد می کند برای مثال ما اجازه بدهیم که اگر متنی بزرگ تر شده است از صفحه کاربرتوسط scrolling بتواند جابه جا شود ولی این کار انقدر سریع انجام نشود که نتواند متنی را بخواند.

- یک مدل از سیستم داشته باشیم در این روش مدلی از سیستم فراهم می کنیم برای کاربر مشخص در سیستم و این مدل پیش بینی شده است برای انجام یک کار مشخص که معمولا نشان بدهیم رفتاری که خود سیستم انجام می دهد را بتوانیم در این مدت زمان برای کاربر نشان دهیم که چند ثانیه طول می کشد که کارش را انجام داده و رفتارهایی که خود سیستم انجام می دهد را به کاربر میگوید.

جداسازی واسط کاربر از بقیه applicationها است که تغییرات مورد استفاده را محلی می کنیم.
واسط کاربر در طول طراحی مدام دچار تغییرنمی شود.واسط کاربر ایجاد و طراحی می شود وبصورت جداگانه تغییرات را محلی می کنیم. کاری می خواهیم بکنیم که واسط ما براساس اصول و منطقی که مبتنی است بر نیازهای کاربران کلی سیستم طراحی شود.واسط کاربر را جدا طراحی می کنیم و سیستم را جدا و بعد مرتبط می کنیم این طوری است که اگر هر تغییرات در سیستم ما بوجود بیاید منطق ما تغییر نمی کند یک سطح انتزاعی تعریف می کنیم که واسط را به عنوان یک لایه مجزا باشد که تغییرات روی آن و نیازهایش اثر نگذارد.الگوهای معماری نرم افزار که برای پیاده سازی هستند از modificationها پشتیبانی می کنند.
چند تا patterns هستند برای طراحی که از این تاکتیک ها استفاده کرده اند.

Patterns یک سری اجزا هستند که ارتباط بین آن ها و یک سری محدودیت های استفاده که بین آن ها هست اولین کاری که معمار باید انجام دهد این است که patterns داشته باشیم اول یک مدل مرجع انتخاب می کنیم و بعد یک pattern معماری انتخاب می کنیم و مدل مرجع را روی pattern پیاده می کنیم و از معماری مرجع به معماری سیستم و از معماری سیستم به سیستم می رسیم.

یک معمار معمولا انتخاب می کند pattern یا مجموعه ای از patterns که طراحی می کند برای اینکه محقق بسازد یک یا چند تاکتیک را. ما می خواهیم به یک سری صفات کیفی برسیم برای رسیدن به انها باید تاکتیک معرفی کنیم که باید برای استفاده از آن ها یک pattern بگوییم که این patterns اماده هستند حالا این pattern راه حل هایی اماده هستند کا از قبل برای حل مسائل مشابه نوشته شده اند و هر کدام می توانند یک یا چند تاکتیک را پیاده سازی کنند چه خوب چه بد هر کدام از این patterns و تاکتیک های داخلشان با صفات کیفی مختلفی درگیر هستند هر پیاده سازی از pattern یک انتخابی است تاکتیک ها برای ما فراهم می سازد.

بنابراین ما در فاز تحلیل که شامل فهمیدن ، درک تاکتیک های patterns می باشد مانند یک لاگی از چیزهای فعال که در فاز تحلیل در یک audit trailحفظ می کنیم را انجام می دهیم و در فاز طراحی یک انتخاب قضاوت مندانه خواهیم داشت که چه ترکیبی از تاکتیک ها ما را به اهداف مطلوب سیستم خودمان می رساند.

خوب یک یادآوری کنیم که موضوع ما چه بود . ما بعد از اینکه آمدیم سناریوها را با هم یاد گرفتیم ، یعنی نحوه تعریف صفت کیفی را یاد گرفتیم ، رفتیم سراغ روش هایی برا ی پیاده سازی این صفات کیفی که انتخاب کردیم ودر قالب سناریو معرفی کردیم که این روش را گفتیم برای این کار از تاکتیک ها استفاده می کنیم و به ازای هر صفت کیفی که در سیستم داشتیم ، ۶ صفت کیفی مهمی که داشتیم ، آمدیم دسته های تاکتیک تعریف کردیم . مثلاً برای دسترسی پذیری ، برای قابلیت تغییر ، کارایی، قابلیت تست در واقع (Usability) و همین ها.

بعد آمدیم حالا بحث بعدی که داریم راجع به (Pattern) بود ، یک نموداری داشتیم . اول این اسلاید هایی که اینجا داریم همین فصل . که ما آمدیم گفتیم اگر تاکتیک ها را بخواهیم پیاده سازی کنیم ، برای پیاده سازی آنها کافی است ما بیاییم (Pattern) را پیاده سازی کنیم . با پیاده سازی کردن (Pattern) های معماری و طراحی ما در کل پیاده سازی می کنیم دسته ای از تاکتیک ها را مربوط به هر صفتی .

بحث (Pattern) معماری :

در این اسلاید گفته که یک معمار یک (Pattern) را انتخاب می کند یا مجموعه ای از (Patterns) هایی که طراحی شده برای محقق کردن یک یا چند تا صفت . پس هر (Pattern) پیاده سازی می کند چند تا تاکتیک ر ا که ممکن مطلوب باشند یا نباشند . یعنی شما ممکن است مثلاً ۲ یا ۳ تاکتیک را نیاز داشته باشید اما مجبور باشید با پیاده سازی این (Pattern) چند تا تاکتیک را پیاده کنید که این ۲ یا ۳ جزء آنهاست . ممکن که نا مطلوب هم جزء آنها باشد ولی می شود (Customize) کرد . بعد هرکدام از این (Pattern) ها درگیر هستند با صفات کیفی مختلف یعنی هدف آنها رساندن ها به صفات کیفی مختلف است.

مثلاً (Pattern) مثل (Call and return) که حالا می بینید یک نمونه آن (Layered) است ،که خود (Layered) به

- چه درد می خورد؟

(Layered) مهمترین خصوصیت صفت کیفی را که برای ما فراهم می کند قابلیت تغییر است . قابلیت حمل یا (Portability) که خود آن جزئی از قابلیت تغییر است . خوب پس هرکدام از آنها در مورد یا بیشتر اهمیت می دهند به یک دسته ازصفات کیفی و پس هر پیاده سازی از این (Pattern) ها گفته در واقع انتخابی است در مورد تاکتیک هاست . بنابراین پروسه تحلیل شامل همه تاکتیک های جاسازی شده در یک پیاده سازی ، یک (Pattern) می شود.

- برای مثال گفته یک پیاده سازی می تواند این طور باشد که (Log) را ازیک کارنامه ای از درخواست ها ثبت کنید تا یک (Active Object) ایجاد کنید . برای فقط یک یا دنبال کردن یک ردیابی (audit trail) یعنی ردیابی کردن باشد یا پشتیبانی کردن (Testability) باشد

یعنی چه ؟

یعنی اگر یادتان بیاید تاکتیک های ما برای رسیدن به (avalibilly) ، (active redundancy) داشتیم اینجا (active object) که گفته منظور همان است . برای یکی از تاکتیک هایی که ما بتوانیم (attack) در امنیت حمله را شناسایی کنیم و بتوانیم (recover) کنیم سیستم را (audit trail) بود . همچنین (audit trail) برای قابلیت تست هم به عنوان یک مانیتور استفاده می شود. بعد، پس با پیاده سازی یک یعنی فقط یک (Log) گرفتن ما بتوانیم به هر ۳ دانه این تاکتیک ها برسیم . حالا پروسه تحلیل پس شناسایی این تاکتیک های جاسازی شده دریک پیاده سازی است .

- پروسه (design) یا طراحی چیست ؟

فرآیند طراحی : طراحی ما شامل یک انتخاب قضاوتمندانه ای که کدام ترکیب از تاکتیک ها ما را به اهداف مطلوب از سیستم می رساند.

در این اسلاید شروع می کند از (pattern) های معماری می گوید . چیزی که در این اسلاید می گوید ، (pattern) معماری در نرم افزار خیلی شبیه همان (Style) های معماری که شما دریک ساختمان می بینید . مثلاً (styles) های معماری که در ساختمان داریم.

مثال : سبک یونانی ، سبک مصری، سبک سلطنتی، مثل مثلاً اینهایی که نام برده مدل خانه های قدیمی می سازند یعنی نمای بیرونی آن را چطوری بسازید حتماً جابجایی های داخلی آن هم دارد.

- مثلاً سنتی ایرانی چطوری می شود ؟

فرض کنید مثل خانه هایی که در کاشان و یزد وجود دارد ، یکی می آید بادگیر می سازد یکی حیاط درست می کند بعد طاق می گذارد نمی دانم یک تزئینی شبیه به کاه گل انجام می دهد. اینها سبک دارد یا اینکه سبک یونانی پیاده می کنند، بعضی ها سبک مصری خیلی پیاده می کنند یا مثلاً مدل چینی می سازند چه سقف هایی می گذارند، رومی درست می کنند. مصری راه راه درست می کند دو رنگ درست می کنند بعد قوس می زنند از این جورکارها انجام می دهند، اینها سبک معماری است. که البته بیشتر مثلاً در ایران ظاهر بیرونی را درست می کنند دو رنگ درست می کنند ولی همه اینها باید در تمام قوانین داخلی هم رعایت شود. مثلاً فرض کنید اگر سبک سنتی ایرانی می سازند مثلاً چیزی به اسم آشپزخانه (open) نباید باشد . فرض کنید اصلاً نباید باشد.

حالا بگذریم ولی دقیقاً ببینید (pattern) هم همینطوری است. یعنی یکسری قوانین خاص را پیاده می کنید برای طراحی آن ساختمانتان . اینها همان (pattern) یا (Style) هستند دیگر.

بعد در این کتاب کلمه (pattern و style) را یکی گرفته ولی جلوتر تعریف کرده یکسری تفاوت هایی بین این ۲ کلمه هست که از کتاب دیگر آورده است.

پس اینجا نوشته این سبک ها پس شامل می شود از یکسری ویژگیهای کلیدی و قوانینی برای ترکیب آنها به طوری که یکپارچگی معماری حفظ شود. این یکی از مشابهاتی است که (pattern) دارد.

یک (pattern) یا (Style) معماری یک توصیفی از نوع مولفه ها و یک الگویی از کنترل های زمان اجرا یا انتقال داده است. که یک (Style) یا یک (pattern) :

- در عمل به دفعات دیده می شود. یعنی می توان در تعدادی از مسائل مشابه به هم از آن استفاده کرد.
- یک (package) از تصمیمات طراحی است که تصمیم طراحی همان تاکتیک می باشد، پس یک بسته ای از تاکتیک هاست.
- ویژگی های شناخته شده ای دارد که قابلیت استفاده را فراهم می کند.
- و توصیف می کند یک کلاس از معماری هارد.

تعریف (pattern) : مجموعه ای از مؤلفه ها و ارتباطات بین آنها و محدودیتهای که روی استفاده از آنها وجود دارد و در این اسلاید هم گفته که یکسری نوع مؤلفه و الگوی زمان اجرا انتقال داده آنها است. یعنی همان تعریف محدودیتهای آنها که داشتیم.

بعد در این اسلاید یک تعریف دیگر دارد . در این جا یک (pattern) یا (Style) مشخص می شود به وسیله مجموعه ای از مولفه ها.همان توضیحی که داریم مثل (Data Repository) مخزن نگهداری داده ها . حالا جلوتر که برویم می بینیم که مخزن داده به صورت (active) داشته باشیم یا (passive) و دیتابیس معمولی باشد یا (Black bored) باشد.

فرآیند یا (object) : اینها مؤلفه های آن هستند یک جانمایی توپولوژیکال، یعنی نحوه چینش خاص ، یعنی گراف یا دیاگرام خاصی که ارتباط بین اینها را نشان بدهد و چینش خاص به همبندی آنها را بیان بکند.

یک مجموعه از محدودیتهای معنایی مثلاً یک مخزن داده نمی تواند اجازه بدهد که مقدارهای ذخیره شده را تغییر بدهید.

اگر یادتان باشد همیشه می گفتم در دیتابیس داده ها را همه می توان پاک کرد واقعاً به صورت فیزیکی ولی اگر (Repository) وقتی اعلام کنیم یعنی بایگانی اطلاعات . در بایگانی هیچ وقت هیچ چیزی پاک نمی شود. ورژن می خواهد که مثلاً این نسخه حذف شده است یا قدیمی و بعد می رود در تاریخچه ثبت می شود. ولی می ماند. عین همه کارهایی که شما در یک بانک انجام می دهید. در یک بانک همین الان که شما می توانید تمام تراکنش هایتان را ببینید، دلیل آن همین است که چون همه رکورد شما تغییر می کند ، نسخه جدید فقط نمی ماند همه قبلی ها هم می ماند. یعنی الان بانک می داند که ۵ سال پیش در فلان روز یا روز ۵ ماه اردیبهشت موجودی شما چقدر بوده است. پس این از بین نرفته است. یعنی چیزی که بانک دارد دیتابیس ساده نیست. و یک مجموعه ای از مکانیزم های تعامل است. مثلاً فراخوانی زیر برنامه است یا رویداد یا (pipe) .

اینها انواع مکانیزهای تعامل است یعنی مؤلفه ها چطوری با هم ارتباط برقرار می کنند. پس می شود :

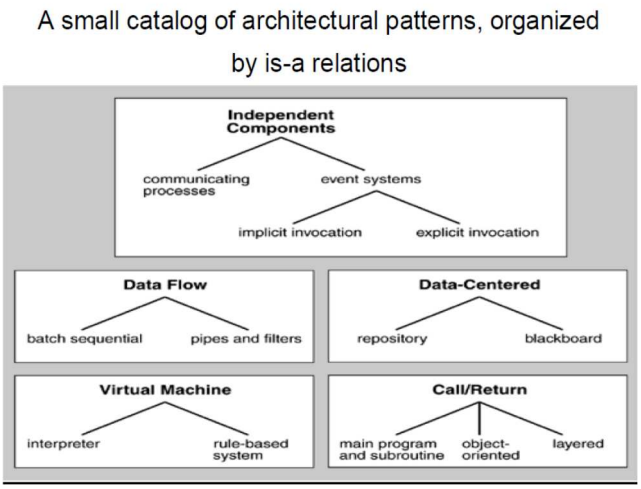
۱- مجموعه ای از مؤلفه ها

۲ و ۴ - نحوه برقراری ارتباط آنها با یکدیگر

۳ - و محدودیتهایی که در برقراری ارتباط وجود دارد.

در اینجا آمده که گارلن و شاوو ۲ تا محقق هستند که در سال ۱۹۹۵ اولین نسخه از یک کاتالوگ را ارائه کردند که (pattern) های معماری را در آن لیست کردند.

که در اسلاید بعدی یک لیست از انواع (pattern) های معماری آورده است. یعنی (pattern) های معماری ممکن و موجودی که بودند جمع آوری کردند .تا ما بتوانیم بر اساس این کاتالوگ مثلاً انتخاب کنیم که کدام (pattern) به کدام صفت کیفی ما بیشتر می خورد و چه دسته تاکتیک های مناسبتری را برای ما فراهم می کنند در فاز تحلیل و فاز طراحی ما بیابیم آن (pattern) را آماده کنیم برای پیاده سازی . بعد که گفته این لیست ها کامل نیست یعنی بازهم به آن اضافه می شود و (Unique) هم نیست یعنی منحصر به فرد است با لیست های دیگر (overlap) دارد و گفته همپوشانی دارد. بعد (Style) ها خودشان با هم همپوشانی دارند یعنی ممکن است که یک (Style) در واقع یک (Style) دیگر هم باشد از یک دیدهایی بشود یک زیر (Style) جزء ۲ دسته (Style) گذاشت. و گفته معمولاً سیستم های بزرگ به این صورت است که بیش از یک (pattern) در پیاده سازی آنها شرکت دارد فقط یک (pattern) نیست.



کاتالوگ را کشیده است. در این کاتالوگ این شکل هایی که در آن می بینید، البته این هم به صورت یک کاتالوگ کوچک از (pattern) های معماری که فقط بر اساس روابط ارتباطاتشان (is - A) بوده یعنی این هست آن. یعنی رابطه سلسله مراتبی ایجاد کردند این دارد این را نشان می دهد.

حالا اینهایی که می بینید اینجا اینها را ما می خواهیم در فصل بعدی که یکسری اسلاید به عنوان (pattern)معماری گذاشته بودم. اینها را می خواهیم دانه دانه باز کنیم و یاد بگیریم. پس حواستان باشد که ببینید من در جزء درس (pattern) معماری من به شما می گویم ولی در موضوع های تحقیق یعنی (pattern) هایی که ما استفاده از آنها ما می توانیم معماریمان ایجاد کنیم.

باز برای یادآوری که بدانیم چه می خوانیم در فصل ۲ یک نمودار بود که چطوری ما می توانیم برسیم به معماری که گفتیم که معمار اول یک (pattern) معماری را انتخاب می کند یا می سازد یک مدل ارجاع هم درست می کند بعد این را (MAP) می کند روی (pattern) معماری به معماری مرجع می رسد بعد معماری مرجع را کمی دستکاری می کند تا به معماری اصلی برسد.

آن معماری که گفتیم می خواهیم انتخاب کنیم اینها هستند. این دسته بندی که از آنها برده مؤلفه مستقل یک دسته است که خود آن همکاری فرآیندهاست ، فرآیندهایی که باهم همکاری دارند یا (event Styles) یا سیستم های رویداد . بعد یکسری هم (pattern) جریان داده هستند که خود آن می شود سریال یا ترتیب دسته ای . یکی دیگر هم می شود ، (pipes an filters) و بعد (pattern) (Data Centred) که ۲ نوع دارد :

۱- Repository

۲- black board

بعد (pattern) معماری ماشین مجازی داریم که مفسرهای سیستم های (rule-based) را جزء شان نام برده است.

(call and Return) داریم که برنامه اصلی وزیر برنامه یک مدل آن است و (object oriented) یک مدل آن و (layered) هم یک مدل دیگرآن است. در اسلایدهای فصل بعد این نمودار را یادتان باشد دانه دانه اینها را می خوانیم.

اینجا ارتباط (pattern) ها و تاکتیک ها را آورده است که دائماً تکرار می شود. گفته که ارتباط بین معماری (pattern) ها تاکتیک ها چه هست.

همانطور که قبلاً گفتیم ، ما تاکتیک ها را به عنوان یک فنداسیون یا یک بلوک های سازنده طراحی می دانیم. یعنی در واقع اینها مثل آجرهایی می مانند که تاکتیک ها که روی هم چیده می شوند. طراحی ما را کامل می کنند. از این نظر است که با استفاده از آنها ما می توانیم چه چیزی را بسازیم؟ (pattern) های معماری ساخته می شوند استراتژی هایی ایجاد می شوند.

پس یک دانه (pattern) معماری از چندین بلوک ساخته می شود که هر بلوک ساخت را به آن می گویند یک دانه تاکتیک.

اینجا دوباره گفته که یک (pattern) معمولاً راه حل های انتزاعی برای مسائل انتزاعی رایج که می توانند دنبال بشوند در یک شرایط خاص و یکسره ویژگی های از پیش تعریف شده دارند.

یعنی اینکه مثلاً شما یکسری مسائل رایج دارید تولید فرم ورود اطلاعات به چه صورت باشد. مثلاً من اگر می خواهم (portability) داشته باشم باید چه کار کنم. مثلاً اگر من می خواهم یک گزارشگیری پویا داشته باشم که در زمان اجرا هم می شود تولید کرد آن را یا تنظیم کرد چه کاری انجام دهم. اینها مسائل رایج هستند یا مثلاً من می خواهم سیستم انبارداری داشته باشم.

یک *مسئله کلی تر*: یا مثلاً انبارداری ها خوب ببینید که مسئله انتزاعی رایج حالا انبار می باشد چه چیزی می خواهید در آن نگهداری کنید مهم نیست. مهم این است که همه انبارها برای همه جای دنیا یک قسمت ثبت قطعات دارد یک قسمت خرید دارد یک قسمت برداشت دارد، خالی کردن یا خروج از انبار دارد.

پس حالا این را برای هر مسئله ای که می خواهیم استفاده کنید (pattern) معماری و طراحی و پیاده سازی آماده برای آن هست. از اینها می توانید استفاد کنید.
الان ما (pattern) معماری را می خوانیم ولی می گوید که این سطح های انتزاعی برای مسئله های شما که اینجا گفته سطوح مسائل انتزاعی فقط که مسائل معماری نیست می توانند مسائل :

۱- Business
۲- تحلیل
۳- معماری
۴- طراحی
۵-پیاده سازی یا همان (Idioms)
۶- (pattern) های تست باشند.

یعنی این انواع (pattern) هایی که من به شما دادم به عنوان موضوع تحقیق چقدر انواع مدل های مختلف (pattern) بود؟
همه اینها هر کدام برای دسته راه حل های از پیش تعریف شده و انتزاعی هستند. برای دسته مسائلی که مربوط به یک موضوع خاص می شود. مثل تحلیل مدل معماری ، مثل پیاده سازی ، طراحی در هر کدام که هستید انتخاب می کنید. یعنی با این توصیفاتی که گفتیم ما می آییم فقط (pattern) معماری ها را کار می کنیم و بقیه آنها را دوستان شما ارائه خواند داد که چه هستند.

دوباره گفته که (Style) های معماری می توانند توصیف شوند به وسیله :
۱- طبیعت مؤلفه هایشان و کانکتورهایشان .
۲- توپوگرافی آنها یعنی نحوه به همبندی آنها .
۳- و نوع کیفیت ها و دلایلی که آنها پشتیبانی می کنند یعنی چه عواملی را پشتیبانی می کنند به چه دلایلی دیگری مدل هستند.
(style) های معماری مهم هستند از این جهت که اثر می گذارند روی صفات کیفی خاص :
۱- (Style) استفاده می شوند برای یک سیستم بستگی دارند به این گه چه صفت کیفی مطلوب اند. برای آن سیستم همان فصلی که گفتیم یعنی براساس آن صفت کیفی که می خواهید به آن برسید (Style) یا (pattern) انتخاب میکید.
۲- (Style) ک روش خلاصه شده ایی برای طراحی و توصیف یک سیستم فراهم می کنند یک روش دقیقی را .

(اسلایدهای ۹۰ تا ۹۲: (کاتالوگ گارلن و شاوو) است .)
در این اسلایدها باز یک تعریفی از (pattern) آمده است. از کتاب گارلن و شاوو در سال ۱۹۹۶ که گفته یک (Style) معماری تعریف می کند یک فرهنگ لغاتی از مؤلفه های و نوع ارتباطاتشان و مجموعه محدودیتهای روی چگونگی ترکیب آنها با هم.
بعد دوباره گفته یک (pattern) معماری یک (Style) معماری بیان می کند یک الگوی سازماندهی ساختیافته پایه ای را بر روی سیستم نرم افزاری فراهم می کنند مجموعه ای از انواع عناصر از پیش تعریف شده و ویژگی های آن ها ، مسئولیتهای آنها که شامل یکسری از قوانین و خط مشی هایی برای سازماندهی و ارتباط بین آنها می شود این تعریف (Buschman) است.
بعد دوباره گفته ببینند (pattern) ما بهترین تمرینات را در دنیای واقعی اعمالی شدند از حل مسائل شناخته شده هستند و ببینید اینجا یک (practice) بزرگترین مهارست ها در تمرین ها برای مسائل شناخته شدن هستند .

در (Anti – Pattern):
در واقع شما می دانید که (pattern) های شما ره حل هایی برای رسیدن به مسئله های مشابه. مثلاً شما یک مسئله می خواهید انتخاب کنید در تحلیل آن ماندید. مثلاً می خواهید یک تخته سیاه ایجاد کنید، یک معماری (fablicsay) بسازید. می گویید برای تحلیل از چه (pattern) ا استفاده کنم. فرض کنید می آیید از (pattern)، (data centerd) استفاده می کنید. حالا جلوتر می گویم که به چه معناست. حالا بعد که می روید این را انتخاب و پیاده کنید از طرف دیگر هم می بینید که یکسری مسائلی هم هستند که می گویند (Anti – pattern) ها. اینها می گویند که از چه راه حل هایی نرو یعنی شکست های دیگران را لیست می کنند. ضد (pattern) است. یعنی در آن می گوید که ببین چه کسانی از چه راه حل ها ، تجربیات به دیگران ، تجربیاتی که منجر به شکست شده ببین چه کسانی چه کاری کردند و شکست خوردند. این کل مفهوم (Anti – pattern) است. این را می زنند تجربیات منفی ایشان که شما دیگرآنها را تکرار نکنید.
خوب این خیلی خوبه ، همیشه که نمی شود کارهای مثبت را تکرار کرد. مهم این است که کارهای منفی را هم بدانید که تکرار نکنید آنها را.

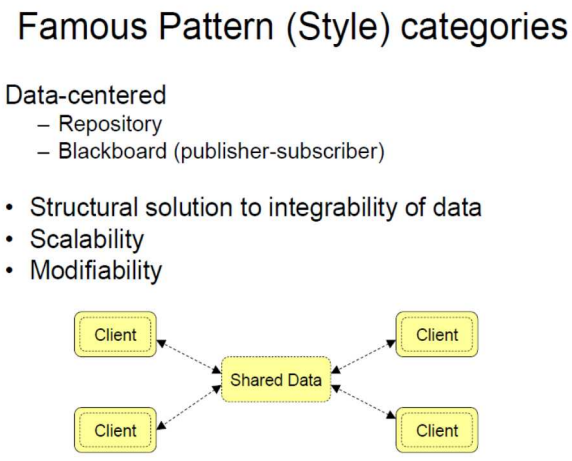
بعد گفته از اثرات ثبت شده روی صفات کیفی را نشان می دهند. مثلاً همان (Style layered) گفته افزایش می دهد قابلیت تغییر را اما کاهش می دهد کارایی را.

این اسلاید همان کاتالوگی بود که آنجا کشیده بود در این قالب در ایجا آورده است که انواع (pattern) هاست.
اینها برای (rozanski , woods) است ۲۰۰۵ ارائه دادند. در واقع همه آن قدیمی ها هستند و اینها هم به آنها اضافه شدند که خیلی جدید هستند مخصوصاً همچنین الان روی (pattern) معماری (peer to peer) و (publisher subscriber) شدیداً روی آنها کارمی شود. نمی دانم با آنها آشنا هستید یا نه اما همه شما دارید با آنها کار می کنید.

مثلاً : سیستم های (peer to peer) تمام شبکه های اجتماعی که شما عضو هستید اینها همه - (Application peer to peer) دارند ، همه معماری (peer to peer) دارند. یعنی اینها وابسته به یک مرکز خاصی نیستند و هر کس مدیریت خودش را دارد. هم سرور و هم (Client) و هم نود که هستند. یعنی همین که خودتان به طور مجزا می توانید گروه تشکیل بدهید بدون وابستگی به کسی به یک سرورخاص هستید. این کلاً دلیلش این است.

دوباره یک تعریف از (pattern) هاست از (Buschman) آورده که تکراری است. که همه را گفتیم.

اسلایدهای (pattern) مشهور معماری



در این اسلاید از همان دسته بندی مثلاً از اولی به نامئ (Data center) شروع کرده که گفته داده متمرکز ۲ نوع است :
۱- Repository
۲- Black board : که خود تخته سیاه می شود (منتشر کننده–مصرف کننده)که همان Subscriber –publisher .
اینجا اسم آورده از انواع آنها و یک شکل هم آورده بعد مهمترین صفات کیفی که این (pattern) هدفش است که به آنها برسد را معرفی کرده یعنی نام برده است. یعنی اصلاً به خاطر رسیدن به این صفات کیفی این (pattern) که داخل خود مجموعه ای از ساکویک ها را دارد که ما را به همین صفات می رساند را ارائه دادیم.
حالا اینها را اسم برده بعد در ادامه دانه دانه این (pattern) ها را نام برده به طور خلاصه و لیست شود بعد ادامه باز کرده و توضیح داده است. یعنی الان من فقط اینجا نام می برم یعنی (Data center) چه هست ، (Repository) چه هست. (black board) یعنی چه که روی اسلاید خودش می گوییم.

۱- Data center
۲- pattern data falw
۳- virtual machin
۴- Call and return
۵- مولفه مستقل (independent Components)

(pattern) معماری داده متمرکز: (Data centered)
همانطور که از روی شکل آن دارید می بینید یک مخزن داده مرکزی دارد که به اشتراک گذاشته می شود برای یکسری از (Client) ها بعد در دو نوع پیاده سازی می شود :
۱- *Repository :* که به آن می گویند اصطلاحاً مخزن داده (Passive) یا غیر فعال. اولین بحث این است که این می تواند خودش هم یک (repository) باشد یعنی بایگانی اطلاعات باشد هم می تواند یک دیتابیس باشد. اگر تراکنش بخواهد اجرا بشود تراکنش ها همه با دیتابیس کار می کنند اگر درس پایگاه داده را خوانده باشید می دانید تراکنش ها با دیتابیس کار دارند حالا یا قوانین (repository) که دارید حاکم است مخزن ذخیره

سازی یا اینکه خود دیتابیس از هر رابطه ای یک شی رابطه ای یک شی گرا حالا هر مدلی ست. خوب پس این مخزن نگهداری داده این مخزن مرکزی که به اشتراک گذاشته شده مثلاً می تواند یک دیتابیس معمولی باشد یا ساده.

۲- *Black board :* مخزن داده فعال است. اما اگر که یک بانک دانش داشته باشد یعنی یکسری متادیتا و نالج داشته باشد که بتوانند استنتاج انجام بشود باز اگر درسهایی مثل سیستم خبره از این طور چیزها را خوانده باشید. اگر همچنین چیزی داشته باشید می شود یک (Black board) که فعال یعنی (active) . (active) بودن آن به این معناست که هر کس هر تغییری روی آن بدهد همه ی (client) ها با خبر می شوند یعنی اصلاً مفهوم تخته سیاه همین است.

- تخته سیاه مهم ترین خصوصیاتش چیست؟

هر کس هر چیزی روی آن بنویسید هم می بیند اصلاً برای این ایجاد شده که اعلان کنند هم مسائل را یعنی هم می آیند هر کس نیازمندیهایش را اینکه می گویند (publisher – subscribe) اینها هم یک مفهوم دارد.

شما یک تخته اعلامیه ها یا اطلاعیه ها را ببینید در خیابان ، سازمان ، شرکت ، دانشگاه هر جا یکسری نیازمندیهایشان را می آیند به آن الصاق می کنند. پس فرض کنید اینجا مفهوم این اینکه روی وال همان که روی وال می آورند که شبکه های اجتماعی می شنوید یعنی اعلام می کنند به همه (client) ها یا دسته خاصی از (client) که ما این نیازمندی را داریم. یا برعکس یکسری ها آن چیزهایی را که خودشان منتشر می کنند را اعلام می کنند. یعنی ما می توانیم این محصولات، این امکانات، این خدمات را ارائه بدهیم. خوب اگر نیامندی اعلام بکنید که بقیه نگاه بکنند می توانند بیاید ببینند با هر کسی که این سرویس را ارائه می کند ارتباط برقرار بکند با آن (client) خاص اگر محصول ارائه بکنند یعنی سرویس را (publish) کنید (subscrib) ها یا مصرف کننده ها می توانند بیایند استفاده بکنند از آنها.

پس مهم ترین خصوصیت بانک اطلاعاتی که به صورت مخزن داره یا (Black board) است این است که با استفاده از بانک دانش که دارد می تواند (publisher) را به (subscrib) ها متصل بکند و اینکه هر تغییری که در داده ها ایجادمی شود همه خبردار می شوند از این جهت می گویند یک مخزن داده (active) .

(این توضیحات با زیرنویس یا بخش توضیحات اسلایدهای پایین تر آن داده شده حالا شماره اسلاید را خواهم گفت داده شده یعنی نوشته شده اگر فرصت نکردید بنویسید بدانید دارد جزوه را)

- حالا به چه درد می خورد که حالا یعنی یک مخزن داده متمرکز داشته باشیم و و (client) ها به آن وصل باشند چه صفات کیفی و کلیدی را برای ما فراهم می کند؟

مهم ترین صفت کیفی یا گفته کلاً این روش (pattern Data center) برای این معرفی شده که یک راه حل ساختیافته ای را برای یکپارچگی داده فراهم کند. (Data center) یا داده متمرکز کلاً برای وقتی ایجاد می شود که یک حجم عظیمی از اطلاعات را دارید و می خواهید نگهداری کنید و به اشتراک بگذارید برای یک تعداد زیادی از (Client) که بازخود این (client) ها کم و زیاد می شوند، متحرک هستند یعنی پویا هستند . بعد حالا شما وقتی حجم زیادی از اطلاعات را دارید چه خوب که یک جا متمرکزش کنید حالا در قالب (Repository) یا (Blackboard) معرفی کنید. فقط مسئله این است که چون (client) های مختلف با آن سروکار دارند بحث یکسان سازی یکپارچگی داده در آن مهم می شود اگر یکجار متمرکز باشد یکپارچگی آن راحت تر می شود. البته باز وقتی می گوییم متمرکز یعنی باز نظر انتزاعی متمرکز یعنی می شود یک داده به اشتراک گذاری شده را به هم صورت توزیع شده پیاده سازی کرد، این امکان هست. یعنی سرور دیتا خودتان را روی چندین نقطه جغرافیایی پخش می کنید. بعد (scalability) هم دارد. یعنی خاصیت مقیاس پذیری که باز جزئی از خاصیت قابلیت تغییر است و برای ما فراهم می کند.

- به چه صورت است ؟

ببینید با انتزاعی که از مدل ذخیره سازی داده ها ایجاد می کند، از مدل استفاده (Client) یک انتزاع ایجاد می کند یک فلش می گذارد این فلش ها (interface) هستند. این (interface) ها می گذارد به این صورت اگر (Client) عوض شود ، هیچ اثری روی بقیه (Client) و مخزن داده یا داده متمرکز ندارد. اگر که مخزن داده شما تغییر کند بازچون (interface) یا فلش هایی که سر جای آن است اثری روی (Client) ها نمی گذارد ، باز اگر (Client) عوض شود روی (Client) دیگر اثری نمی گذارد. چون این (Client) ها مستقل از هم عمل می کنند.

این همان خاصیت (modifiability) می شود ، (Scalability) هم به این مفهوم است که تعداد (Client) ها به صورت بی شمار می تواند اضافه بشود یا می تواند حذف شود. این مقیاس پذیری آن پس از نظر تعداد مصرف کنندگان یا میزبانهای او است. پس اگر این (pattern) اگر که جایی دارید که مخزن ذخیره سازی بزرگ دارید و می خواهید به این صفات برسید برای شما مهم است که بیاید از (pattern Data center) استفاده کنید.

Famous Pattern (Style) categories

Dataflow
- Bach sequential
- Pipes and filters

- Reusability
- Modifiability
- Not interactive
- Poor performance

(pattern) جریان داده : (Data falw)

باز وقتی که مثلاً ما یک داده ای داریم که پشت سرهم یکسری تغییرات روی آن انجام می شود. یک ورودی و یکسری تغییرات باید روی آن انجام بشود تا به یک خروجی برسد. یعنی جریان داده باید حرکت کند تا به یک جوابی برسیم. این از (pattern Data falw) استفاده می کنیم که خود آن دو نوع است :

۱- یا ترتیب های بسته ای

۲- یا اینکه (pipe and filters)

شکلی که اینجا کشیده است همان شکل (pipe and filters) آن است. () و اینها را فکر کنید ماژول هستند. (➡)و اینها هم رویدادها که دارند جریان پیدا می کنند .

در واقع هر کدام از این عناصر ما حکم یک فیلتری را دارند که داده از آن عبور می کند. یعنی فیلتر همیشه چه چیزی را فراهم می کند. یعنی فیلتر همیشه داده را تغییر می دهد یا از نوع به یک نوع دیگر منتقل می کند یا تبدیل می کند .

- حالا این به چه دردی می خورد؟

این (pattern) این صفات کیفی را برای ما فراهم می کند :

۱- قابلیت استفاده مجدد دارد.

۲- قابلیت تغییر دارد.

۳- وقتی که می خواهیم اصلاً تعامل نداشته باشید مثل محاسبات پیچیده که خیلی محاسبه آن زیاد است مثلاً یک فرمول بزرگ را می خواهید یا یک معادله خیلی بزرگ را می خواهید محاسبه کنید از این (pattern) استفاده می کنید .

۴- کارای آن خیلی ضعیف است .

- اگر گفتید چرا(pipe and filter) کارایی آن ضعیف است؟
- به همین شکل درون اسلاید نگاه نکنید چرا کارایی آن ضعیف است.(poor) پردازش دسته ای یا (pipe and filter) چرا کارایش ضعیف است؟
- ببینید کارایی با زمان اجرا و زمان انتظار سروکار داشت زمان اجرا احتمالاً در این طولانی می شود چرا؟

ببینید اجرای موازی را در این نمی توانید فراهم کنید. شما پردازش موازی نمی توانید داشته باشید. پردازش ترتیبی است. چون وابستگی ورودی یکی خروجی یکی دیگر خواهد بود. پس به اندازه جمع زمان اجرا می شود. جمع زمان اجرا های هر کدام از اینها که دارد طول می کشد. هیچ کدام را نمی توان موازی اجرا کرد. پس کارایی این ضعیف خواهد بود. بعد حتی اگر چند تا (cpu) هم داشته باشد یعنی سخت افزار شما هم اجازه بدهد یعنی از لحاظ واقعی هم توزیع شده باشید ، نمی توانید چه کار کنید این موازی را داشته باشید.

Interactive :

تعامل می شود یعنی اینکه شما یک برنامه نوشته باشید که یک معادله بزرگ را محاسبه می کند یا یک فرمول بزرگ را محاسبه می کند. شما تنها تعاملی که دارید متغییرها را که دفعه اول دادید و جواب آخر را از آن می گیرید. یعنی در حین عملیات نه از شما سؤالی می کند و نه اطلاعاتی به شما می دهد. این یعنی تعامل نداشتن. یعنی این اطلاعاتی که از اینجا می گیرد از کاربرآن دیگر با کاربر تعامل ندارد تا این آخر که داده به کاربر می دهد .

(pattern Virtual Machine) :

همه بلد هستید و همه با آنها کار کردید می توانید چند تا ماشین مجازی نام ببرید؟

۱- VPS
۲- VMware
۳- .net framework
۴- شبیه سازها
۵- SDK-android

- (این ها هم ماشین مجازی هستند)

(Vmwave) : جاوا ، ویندوز نصب می کنید ، لینوکس را می آورید روی ویندوز ویا برعکس .

همه این ها مفسر دارند. یعنی شما مثلاً مفسر های کلاً خود این کامپایلرها به طریقی جزء ماشین های مجازی حساب می شوند چون دارند زبان سطح بالا را تبدیل می کنند به یک زبان قابل فهم برای (plat form) ها درسته؟ خوب همین است که این را فراهم می کند . یعنی در کل همه این اختلاف بین زیر ساخت که همان (plat form) شما باشد یعنی (OS) سخت افزار شما را با (Application) رفع می کند . اینطوری (Application) بدون توجه به اینکه زیر ساخت شما چه چیزی هست می تواند اجرا شود . .همه آنها همین کار را انجام می دهند .

- اصلاً مهم ترین خاصیت (Virtul machin) چیست ؟

همه این کارها می کنند که (Porttability) را فراهم کنند یعنی حمل کنیم از یک (plat form) به یک (plat form) دیگر به درد شبیه سازی می خورد ، به درد قابلیت تطبیق می خورد یعنی با هر وضعیتی بتواند خود را هماهنگ بکند . ولی کارایی آن پایین است ، اما (Poor) نیست . یعنی فقر کارایی ندارد اما کارایی پایین دارد.

- به نظر شما چرا کارایی این (Virtul machin) پایین است ؟
- (Application)هایی که تحت این ماشین ها اجرا می شوند سرعتشان پایین تر است یا آنهایی که مستقیماً روی (OS) اجرا می شود؟ (آنهایی که مستقیماً روی (OS) اجرا می شوند .)

به این خاطر است که شما یک لایه اضافه کردید یعنی یک ماژول اضافه کردید یعنی یک کار بیشتر دارید انجام می دهید . کار بیشتر ماژول بیشتر محاسبات بیشتر نتیجه کارایی کمتر است. شما آمدید اختلاف را با ایجاد یک لایه حل کردید اما خود این هم باعث پایین آوردن کارایی می شود.

پس اینطور نوشته لورپانارپ این آنقدر مزیت برای ها فراهم می کند . یعنی آنقدر خاصیت (Porttability) برای ما مهم است که کارایی پایین برای ما قابل چشم پوشی است. یک دلیل دیگر هم این است که سرعت پیشرفت سخت افزار همیشه بیشتر از نرم افزار است که هم قبول دارید. پس آنقدر سخت افزارها جلوتر و پیشرفته تر هستند که این -

(Low Performance) قابل چشم پوشی می شود در مقابل (Porttability).

(pattern Call and return) :

که در واقع می شود گفت که اولین (pattern) است که شما در زندگی از آن استفاده می کردید . به عنوان یک تکنسین کامپیوتر،مهندس کامپیوتر همین (Call and return) است .

یعنی همان موقع که می آید یادتان هستند برنامه هایی که می نوشتید به زبان (C و ++ C) حتماً زمان کاردانی نوشتید .

- (pattern) معماری برنامه های (C++) چگونه است ؟
- (Main program and sub-routine) یعنی یک برنامه اصلی دارید و بعد فراخوانی تابع دارید . این به چه درد می خورد همیشه ؟
- می گفتیم به این درد می خورد که می توانیم چه کار کنیم ؟

تابع ها را هر جا که خواستیم از آن استفاده کنیم . هروقت خواستیم تابع را تغییر بدهیم ، یک تابع دیگر به جای آن بگذاریم . وردیابی خطا آسان می شد . همه اینها در(Medi fiability) برای ما فراهم می شود یا نماد می شود . پس یک (pattern Call and return) داریم که خودش انواع دارد :

۱- برنامه اصلی – زیر برنامه

۲- فراخوانی از راه دور

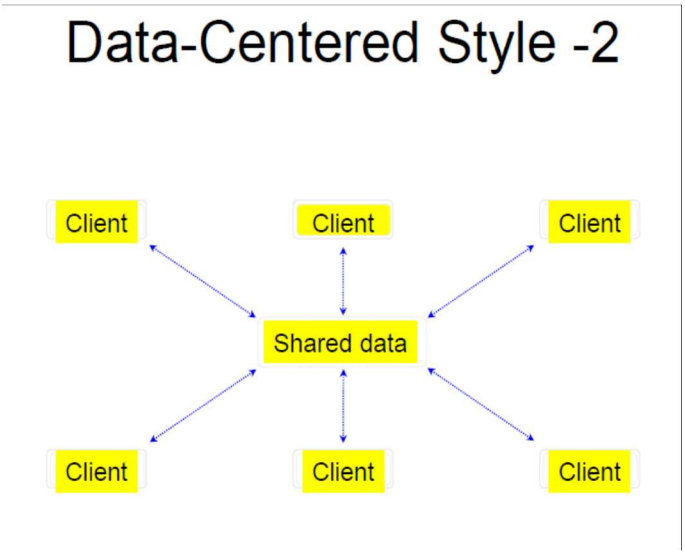
فراخوانی از راه دور : که وقتی شما می خواهید (Remote Procedure) کار انجام بدهید . می خواهید شما ، اصلاً برنامه های client site و server site ، مثلاً شی یک داده دارید شما از طرف (client) هستید داده را می فرستید سرور برای شما اجرا بکند . یک سرور تا قسمتی از پردازش را می فرستد که(client) اجرا بکند . این موقع دارد یک فراخوانی از راه دور انجام می دهد . یعنی از دید برنامه کاربردی (Application) به نظرش می آید که دارد فراخوانی یک تابع محلی .روی همین سیستم عامل را انجام می دهد در حالی که این تابع توانایی اجرایی روی سیستم راه دور را دارد و باید به صورت یک پیام باید بسته بندی و ارسال شود بعد تابع یک جای دیگر اجرا شود . خوب این برای ترکیب کردن کارایی است .

یک مدل دیگر (Object – Oriented) یا (adt) ، (abstract data type) است . همان شی گرایی که حالا توضیح بیشتر دارد و به درد (Modifiability) و قابلیت استفاده مجدد می خورد و (Call and return) یک مدلش (Layered) است . الان اگر که یک مقدار نگاه بکنید (pattern Call and return) یک نمونه آن (Layered) ولی بالا در -

(Virtul machin) اولین چیزی که به ذهن شما می آمد در قالب لایه پیاده سازی می شود ماشین مجازی یک لایه می شود.

اینجاست که درآن تعریف ها نوشته بود (pattern)ها باهم(overlap) دارند اینطور نیست که باهم (Noun over laping) باشند. منظور این است، یعنی اینکه یک(pattern)می تواند از یک زیر(pattern) یک نمونه از (pattern) می تواند نمونه ای از(pattern) دیگر هم باشد.

یک مدل دیگر از (pattern) معماری مؤلفه مستقل است (Independdent Components) که همانطور که می بینید این است که چند تا فرآیند باهم همکاری داشته باشند یا اینکه سیستم های (eventy) هست که باز جلوتر می بینید آنها را . کلاً برای (intrupt)ها و سیگنال و این چیزها است . بعد این (Modifiability) را برای ما فراهم می کند .
(اینجا این توضیح ها را داده یعنی معرفی کرده است حالا می آید دانه دانه توضیحات این (pattern) ها را می دهد .)



توضیح (Data – center) :

گفتیم هدف آن این است که یکپارچگی داشته باشد و مقیاس پذیری را فراهم بکند که مقیاس پذیری از جهت اضافه کردن (client)های جدید یا داده جدید به آسانی در مخزن نگهداری داده مرکزی است .

بعد ۲ مثال هست :

۱- گفته اگه ذخیره سازی داده (passive) یا غیر فعال باشد باید همان الگو (repository) را باید پیش بگیرید.

۲- اگر (active) باشد که باید (Black board) باشید.

(Virtual machin) :

گفته هدف آن این است که می خواهد (Functionality)های غیر محلی یا غیر بومی را شبیه سازی بکند . برای اینکه قابلیت جابه جایی یا ساختمان نمونه اولیه را فراهم کند.

پس مثال های آن هم مفسرها به این صورت هستند ، سیستم های مبتنی بر قانون وپروسسورهای زبان مشترک .

پس همانطور که می دانید (Virtual machin) مفهومش چی هست یعنی فقط شما می خواهید چه کار کند تابع ها یا همان عملیات ها، کاربردهایی که بومی هم نیستند را ، یعنیر بومی اجرا نمی شوند در واقع قابلیت اجرا شدن (OS) را ندارد کاری کنید آن را که اجرا بشود .

در این اسلاید یک شکلی کشیده است که یک حالت کلی و مجازی ازیک(pattern) ماشین مجازی را برای شما کشیده است . یک توضیحی بدهم برای شما از این :

ببنید کلاً در این گفته ۳ نوع داده داریم در این :

۱- یکی برنامه ای که تفسیرشده _____

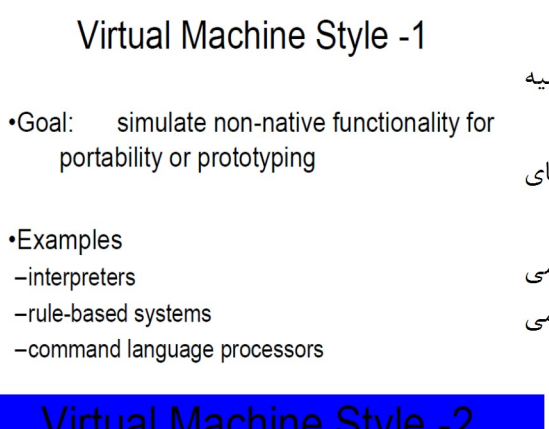
این است . Program being interpreted)(

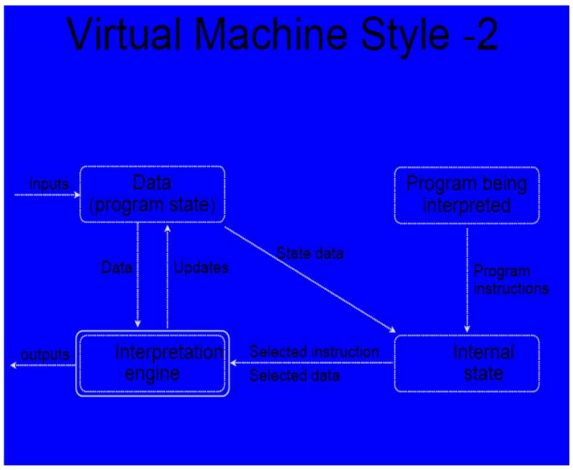
(یعنی زبان ورودی ما هست) .

۲- یکی داده های آن برنامه است (Data program state) (یعنی اصلاً آن داده هایی که به متغییرهای آن انتساب می دهیم).

۳- یک نوع داده دیگر که داریم وضعیت داخلی مفسرما است (internal state) .

البته این شکلی که (Virtual machin) کشیده است برای یک مفسر کشیده است . این مفسربراساس وضعیت داخلی خود یعنی اینکه الان یعنی از نظر مفسر این کلاً (interpretation engine) که اینجا می بینید که یک خروجی تولید می کند . این می آید یک سیستم عامل مجازی را شبیه سازی می کند یعنی سیستم عامل و (plat form) ها یعنی برای خود انگار که روش





هایی فراخوانی سرویس های سیستم عامل را ایجاد می کند . اینکه (cpu) چه مدل باید باشد را فراهم می کند ایجاد می کند ، نوع داده ها را فراهم می کند . همه این ها رابرای خودش ایجاد می کند. حالا این اطلاعات رابه ازای آن برنامه هایی که دارد تفسیر می شود در وضعیت داخلی خود نگهداری می کند که الان ثبات های (cup) مجازی من چه مقدارهایی را دارند. یعنی دقیقاً مثل بحث فرآیندها است در سیستم عامل که می گفتیم (centext) فرآیند داریم ____ (treid context) داریم . اینجا هم همین را دارد می گوید. پس یک وضعیت داخلی که حالت جاری که باید اجرا بشود را نگهداری می کنند .

- این وضعیت داخلی همیشه براساس چه چیزهایی تغییر می کند ؟

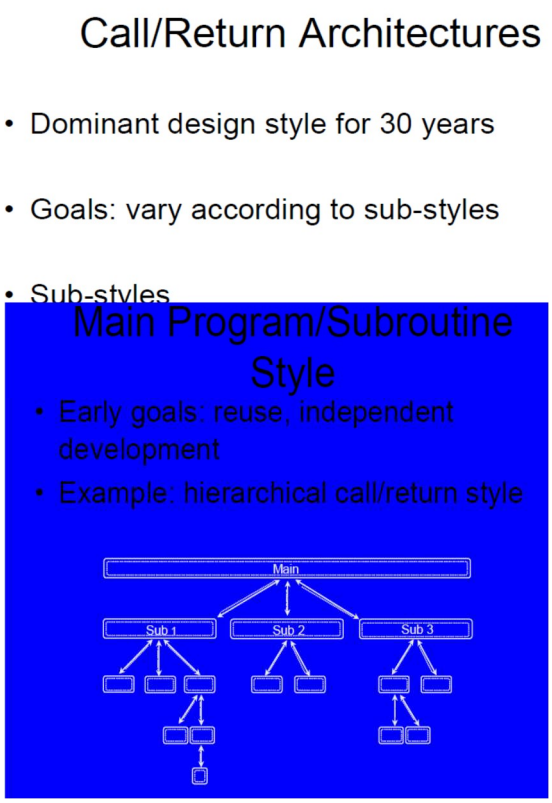
براساس نوع داده هایی که وارد می شود و بر اساس دستورالعمل جاری که لز برنامه می خوانیم که میخواهیم اجرا کنیم که تغییر خواهد کرد. بعد بر همان اساس می آید انتخاب می کند و دستوالعمل انتخاب شده یا داده انتخاب شده را می فرستد برای مفسر یعنی در اختیارمفسر قرار می دهدو مفسر دستوالمعل انتخاب شده را اجرا می کند . این کار آن باعث تغییر روی داده می شود . سپس داده (update) می شود . همانطور که در اینجا نوشته است و نهایتاً یک خروجی را می دهد. پس می گوید که یک (interpretation engine) را دارد که این کار را برای ما انجام می دهد. پس اینکه اینجا می بینید (interpreter engine) این همان لایه اضافه است که ما وارد کردیم انعطاف پذیری را برای ما فراهم می کند دیگر مهم نیست که نوع داده ها و نوع برنامه ها چطوری باشد. ولی خوب این همان خودش ماژول اضافه است.که (per formance) را می آورد پایین.

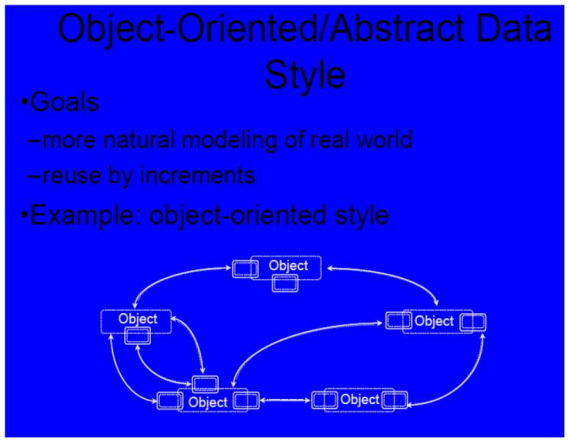
(call / return) :

که گفته این(pattern) بیش از ۳۰ سال که این معماری دارد. استفاده می شود در طراحی ها و بعد هدف آن هم گفته خیلی وابسته هست به زیرنمونه های خود _____ (sub-styles)های آن هدف هایی که دارد و بعد مثلاً اگر (main program / subroutine) باشدیکسری هدف ها را پیش می گیرد و اگر (object oriented) باشد به یک روش دیگر و اگر (layered hierarchies) لایه های سلسله مراتبی باشد یک طور دیگر است.

(حالا جلوتر دارد این ها را دانه دانه باز کرده زیر لایه هایش یا زیر استایل هایش را.)

مثلاً ببینید (main program / subroutine style) گفته اهداف اولیه آن این است که قابلیت استفاده را فراهم بکندو توسعه مستقل داشته باشد. شما مثلاً اگر یک برنامه (main) داشته باشید در پروژه دانشجویی زمان لیسانس شما حتما این کار کردید دیگر.یک برنامه بزرگ که مثلاً با همان (c++) می خواهید بنویسید چند نفر از هم گروهی ها مسئول این می شود که تابع ها را بنویسید و یک نفر مسئول این می شود که همه تابع ها را در برنامه اصلی به هم بچسباند و فراخوانی ها را انجام بدهد. این توسعه مستقل را ایجاد می کند و هر کدام از این تابع ها را می شود جای دیگر هم می توان استفاده کرد. این هم قابلیت استفاده مجدد را فراهم می کند و کلاً یک ساختار سلسله مراتبی دارد یعنی (main) فراخوانی می کند اینها را ، اینها هر کدام دوباره زیر تابع می توانند فراهم بکنند. البته هر کدام از اینها که می بینید می توانند توسط چند تا از این (sub) ها فراخوانی شوند این امکان وجود داشت.





بعد یک زیر (style) دیگر از (style call and return) یعنی فراخوانی – بازگشت این است که ۲ تا آنها البته با هم آورده (object oriented) همان بحث های شی گرایی یکی هم (abstract data stule) است یعنی داده انتزاعی که هدف همه آنها این است که بیاید دنیای واقعی را شبیه سازی کند یا مدلسازی کند. یعنی دقیقاً همه بحث هایی که در شیء گرایی بلد هستید .

- شیء گرایی چکار می کند؟

می آید می گوید که یک شیء مثل دنیای واقعی از یک کلاسی می آید که یک بهینه ای است از مشخصات و رفتارها با هم دیگر.مثلاً صندلی یک کلاس است که این از مشخصاتش جنس ، رنگ ، وزن و میز دارد یا ندارد و پشتی دارد یا ندارد ، چند تا پایه دارد . از این ها تشکیل شده است.

- صندلی به چه درد می خورد؟

عمل نشستن را انجام می دهیم روی آن . برای اینکه روش بشینیم استفاده می شود . یا مثلاً ما یک شیء را تعریف می کنیم حالا خیلی بی ربط شد ولی مثل دانشجو.

- دانشجو چیست؟

دانشجو مجموعه ای از مشخصات است و مجموعه ای از رفتارها مثل گرفتن درس ، پاس کردن درس ، مثل ثبت نمره ، حذف و اضافه کردن و اینطور چیزها و تحقیق کردن ، نمره پایان نامه و اینطور چیزها.

پس این همان بحث های (object oriented) است یعنی بستگی به مسئله تان انتخاب می کنید که (object) داشته باشید. و قابلیت استفاده مجدد دارد در بحث های (increments) مخصوصاً . یعنی شما اگر یک کلاس تعریف می کنید هر چقدر بخواهید می توانید برای آن (object) تعریف کنید و با آن کار کنید.

بعد در اینجا ۲ مدل را گفته است :

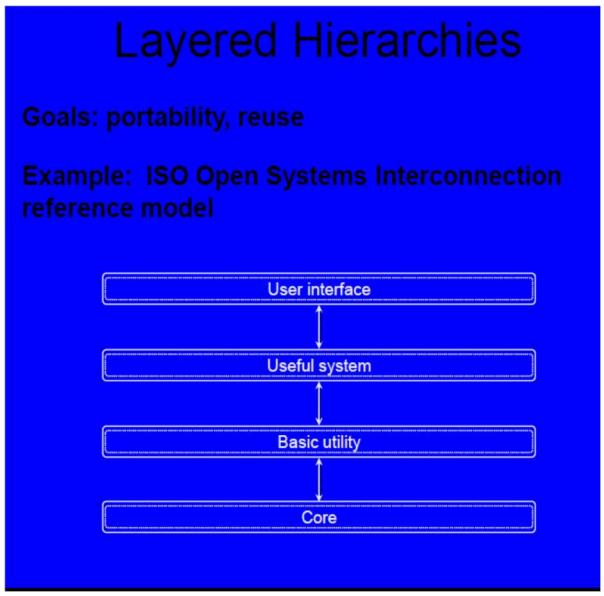
۱- (adt)

۲- Object-oriented

ببینید در (adt) هم به آن می گویند فرق اینکه مثلاً شما اگر انواع داده هایتان زیاد باشد نمایش های مختلفی اگر بخواهید از آنها داشته باشید ، می آید از مدل (adt) می روید یعنی دقیقاً هم (object-oriented) هم (adt) هر دو آنها برای ما مخفی سازی اطلاعات را فراهم می کنند . یعنی انتزاع را ایجاد می کنند یعنی مثلاً وقتی شما می آیید یک شیء را صدا می زنید یا یک نوع داده مثل (string) صدا می زنید به نحوه پیاده سازی سرویس هایی که اینها می دهند یا کارهایی که می کنند ، کاری ندارید یعنی این (infermatien caiving) را برای شما فراهم می کنند. هر دو آنها ولی تنها تفاوتی که با هم دارند و کپسول سازی هم ایجاد می کنند یعنی متد ها و (property) ها را با هم یک جا جمع می کنند فقط شما فراخوانی می کنید آنها را . اما تنها تفاوتی که دارند در (adt) اصلاً ما بحثی از ارث بری نداریم اما در شیء گرا ارث بری داریم.

- در شیء گرا ارث بری ها چگونه بود؟

می گفتیم ویژگیهای مشترک را می گرفتیم و می کشیم روی کلاس بالایی می گذاریم بعد زیر کلاس های آن که از آن ارث می برند . همان مشترک ها و ویژگی های خاص خود را هم دارند . مثلاً الان همه ما از کلاس انسان همه ارث بری داریم . کارهایی که همه انسانها انجام می دهند مثل خوردن و خوابیدن و گریه کردن و خندیدن و غیره همه این چیزها ۲ تفاوتی که بین این ۲ وجود دارد.



یک نوع دیگر (call and return) ، (layered) است این همان لایه بندی که در این لایه بندی باز خود شما بهتر از من می دانید که معروفترین آن همان بحث استاندارد (iso) در شبکه است.

استاندارد ۷ لایه ای کلاً اگر شما بخواهید یک مجموعه ای از (functionallty) ها داشته باشید یعنی عملیات و سرویس هایی که سیستم می خواهد بدهد بعد ببینید اینها به هم مرتبط هستند ، یک سازگاری و انسجامی با هم دارند . اگر آنهایی که با هم انسجام دارند دسته دسته بگذارید در لایه های مختلف اینطوری انتزاع از لایه ها ایجاد بکنید. بعد بگویید که هر لایه فقط به بالایی سرویس بدهد و از پایینی سرویس بگیرد.یعنی اینکه (pattern) لایه ایجاد کردید. عین همان چیزی که داشتیم . هدفش هم این است که ((protability) فراهم کنید به خاطر انتزاعی که لایه ها نسبت به هم دارند و قابلیت استفاده. یعنی اینکه همانطوری می تواند جای دیگر هم استفاده شود، می شود لایه را اضافه و حذف هم کرد.که معمولاً لایه آخری هست که همینطور که اینجا می بینید می آید روی لایه به عنوان هسته ماشین یا سیستم عامل تدبیر می شود ، قرار می گیرد .

خیلی از معماری هایی که در کتاب های مختلف دیدید معماری لایه ای است. مثلاً سیستم عامل لایه ای معرفی می کنید آنها را. درس های ذخیره و بازیابی اطلاعات ، رسانه ها لایه ای معرفی می کنند. مثلاً سیستم فایل لایه ای معرفی می کنند آنها را. سیستم فایل خودش یک لایه ای دارد که با درایورهای هارد کارد می کنند باز دوباره یک لایه دارد که سیستم فایل منطقی دارد. یک لایه سیستم فایل فیزیکی دارد ، بعد دسترسی مشترک دارد. از این چیزها که فراهم می شود تا همان لایه بالایی که (application) می شود که نحوه فراخوانی یک فایل و باز کردن یک فایل دست خودش است و خودش فراهم می کند.

خوب در این اسلاید این چند تا دانه را با کمی جزئیات توضیح داده است. باز دارد در ادامه اینها ، فقط یک چیزی که اینجا آمده اضافه کرده این است که یک توضیحی اینجا داده که این(pattern) هایی که ما معرفی کردیم . اینطوری نیستند که یک سیستم داشته باشید همه آن از یک (pattern) باشد یک سیستم ممکن است از چندین (pattern) تشکیل شده باشد.

- حالا این (pattern)ها چه ارتباطی با هم دارند یعنی چطوری یک سیستم با اینکه مجموعه ای از (pattern) ها را داخل خود دارد در چه قالبی پیاده سازی می شود؟

می گوید که این مجموعه از (pattern) ها می تواند یا به صورت (locationally) وجود داشته باشند ، با هم ارتباط داشته باشند یا به صورت (hierarchically) سلسله مراتبی یا به صورت در واقع همزمان یا متقارن وجود داشته باشند.

پس گفته تعداد کمی از سیستم ها هستند که ساخته شده باشد فقط از یک تک (style) یا یک تک (pattern)و اغلب سیستم ها ناهمگون هستند.(hetero generous) یعنی ناهمگون هستند و ترکیبی از چندین (style).

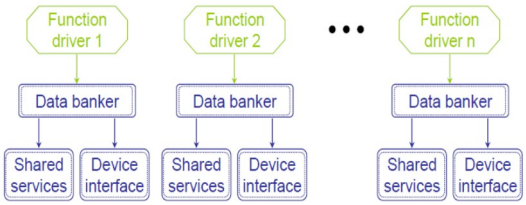
۳ نوع از سیستم (hetero generous) یا ناهمگون وجود دارد.

۱- Locational
۲- Simultaneeul
۳- Hierarchical

که هر کدام را توضیح داده است.

Locational Heterogeneity

- Different styles in different parts of the system
- Example: A-7E used a cooperating process style in function drivers and call/return style elsewhere.



ببینید اگر که (style) خیلی ساده باشد می گوییم فقط در حد این تعریف ها بدانید کافی است.

اگر (style)های مختلف در قسمت های مختلف سیستم شما بدون هیچ قانونی ریخته شده باشند قاعده خاص این می شود (locational).

مثلاً اینجا یک سیستم کنترل پرواز را معماری آن را معرفی کرده که اگر اشتباه نکنم این سیستم (A-7E) که برای فصل ۳ یا فصل ۶ است که معماری آن را اینجا معرفی کرده (A-6E) از یک جهت هایی یک سری خلبان خودکار است.

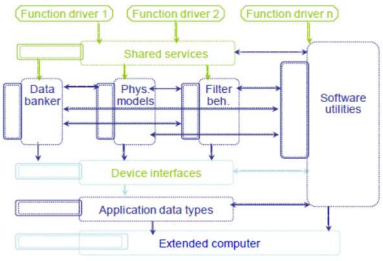
خلبان خودکار :

این خوب کلی تابع و پیچیدگی دارد . معماری یک مدلش این است که (function) هایی که برای راننده یا درایورهای خود است این n تا(function)که دارد.(function)ها مثلاً ازروش فرض بکنید از(pattern call and return) پشتیبانی کرده باشد بعد یا فرض کنیم از الگوی پروسس در (function) های درایورهایش استفاده کنیم از الگوی (call-return) در قسمت های دیگر آن استفاده بکنیم.

توضیحات بیشتر : یعنی مثلاً اینجا (Data banker) آن فراخوانی (Share date) دارد با این ۲ تا (Device Interface و Shared service) ، (Call- return) فراخوانی نمی کند بعد این سیستم ها پروسس مشترک هستند یعنی هر کدام یک فرآیند هستند در (Style) مؤلفه مستقل است . که در ادامه توضیح آن را داریم .

Simultaneous Heterogeneity

- Different styles due to style overlap
- Example: A-7E simultaneously used layered,
- cooperating process, and object-based styles.



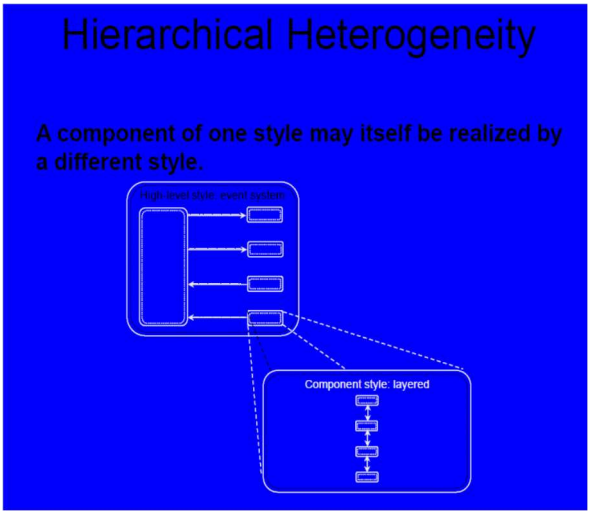
بعد مثلاً همین معماری (A- 7E) که اینجا اگر بخواهید به صورت سلسله مراتبی داشته باشید شما .

- یعنی چه (Style) سلسله مراتبی در آن باشد ؟

گفته اگر (Style)های مختلف همپوشانی داشته باشند یعنی یک ناهمگونی سلسله مراتبی ایجاد کردند که باز همان را که آنجا دیدید آمده گفته (Style) های پروسس های مشترک و (Style Object based) دارد و (Layerd) که اینها هر ۳ به صورت (OverLap) ایجاد شده اند.

الان ببینید باز همین معماری (A- 7E) که داریم خلبان خودکار یک (Layerd) دارد یعنی اگر از جنبه هایی نگاه کنید این سیستم سخت افزاری (Application data type) و این (Device inter faces) ها همه لایه ای هستند و بعد در عین حال خود این (Data banker) و این مربع هایی که کنار آن گذاشته اینها خودشان را (Object – Oriented) صدا می زنند . بعد دوباره اینها خود ازیک جهت هم پروسی هستند یعنی در زمان اجرا در واقع این چیزهایی که فعال می

شوند ، یعنی همه این ۳ تا (Style) هایی که گفتیم باهم قاطی هم مواجه شدند نمی شود از هم مجزا کرد آنها را و بگوید این تکه آن این است این تکه این آن است . این استایل آن استایل.



- حالا بعد اگر سلسله مراتبی باشد یعنی چه ؟

سلسله مراتبی وقتی می گویند یعنی یک سیستم ترکیبی دارد از چندین (Style) ترکیب شده (Style) ها در قالب سلسله مراتبی هستند . مفهوم آنها این است که هرکدام از این عناصر یک (Style) هستند خود آنها به صورت یک (Style) دیگر پیاده سازی شود.

مثلاً : فرض کنید اینجا یک سیستم (event) دارید بعد (Style event) به هرکدام از این (event) خود از (Style layered) رفتند .

پس اگر هر عنصر از یک (Style) خودش براساس یک (Style) دیگر پیاده سازی شود (High logical) . اگر ترکیب (Style) هایی که درون یک سیستم وجود داردو طوری باشد که نشود جدا کرد آنها را و همپوشانی داشته باشد اسم آن می شود همزمان ، سیستم ناهمگون همزمان (نامتقارن). اگر یک ترکیب (Style) های شما به طوری باشد که هیچ قاعده خاصی نداشته باشد و هرگوشه ببنیم که یک (Style) داشته باشد می شود (logal) .

ازاینجا به بعد آمده گفته ، کمی با (Style) های معماری آشنا شدید، می گوید حالا ما چطوری می توانیم یک استایل را انتخاب کنیم با این توصیفاتی که گفتید خوب آمدیم گفتیم، یک کاتالوگ داریم از (Style) ها.

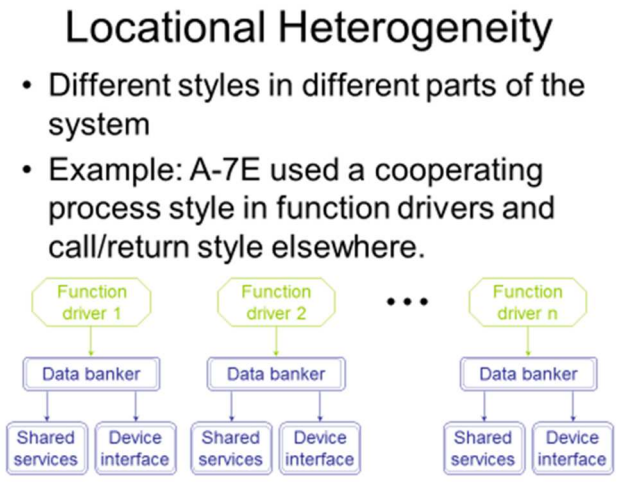
- چطوری می توانیم از بین این (Style) هایی که وجود دارد یک (Style) را انتخاب کنیم ؟

مایک کاتالوگ به تنهایی به دردمان نمی خورد . یعنی شما مثلاً فرض کنید یک کاتالوگ دارید برای انتخاب یک لب تاپ ، مثلاً انواع لب تاپ ها با مشخصات آنها بیان شده است . حالا این کاتالوگ فقط در اختیار شما یک اطلاعاتی می گذارد که چه نوع لب تاپ هایی یا چه نوع کامپیوترهایی وجود دارد و مشخصات هرکدام چطوری است. اما کاتالوگی خوب است که یک راهنمایی هم بکند که بگوید مستقیم به مسئله شما و به شرایطی که شما دارید، کدام یک از این لپ تاپها را انتخاب کنید خوب است.

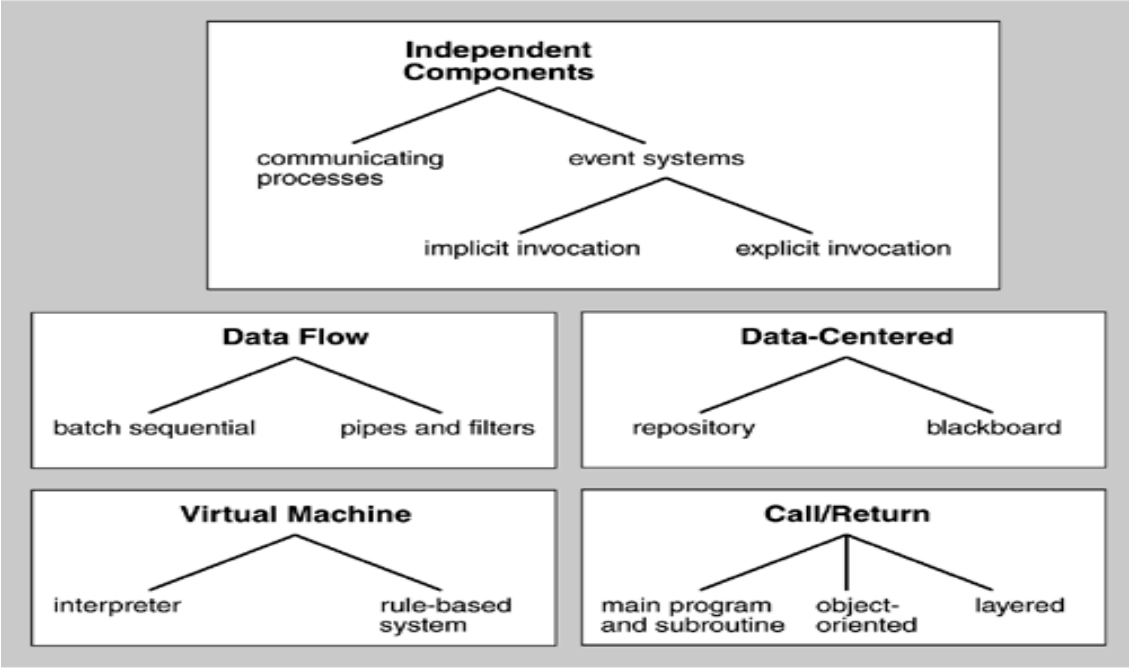
یعنی اگر یک کاتالوگ داشته باشید که یک هدف یک کاتالوگ یا (Style) این است که توسعه بدهد یک (hand beok) و (Design) . یعنی کتاب راهنمای طراحی را که به شما بگوید اگر مسئله شما شبیه مسئله (x) است بیا از ((Style(Y) استفاده بکن . می گوید در (Style) همچنین چیزی هنوز توسعه داده نشده یعنی همچنین راهنمایی نیامده فقط بهترین کاری که می شود کرد این است که ما بیاییم یکسری قواعد سرانگشتی (Rules of thumb) معرفی کنیم که به شما کمک کند . یعنی این قواعد یا توضیحات به شما کمک کند در انتخاب (Style)مناسب شما.

یعنی دقیقاً همان دفترچه کاتالوگی که برای لب تاپ هاست در نظر بگیرید . یک توضیحی داشته باشد که اگر فرض کنید مثلاً شما می خواهید برای کار تدریس از لب تاپ استفاده کنید و انقدر پول برای شما مهم است خیلی می خواهید یک لب تاپ خوب داشته باشید، این مدل ها را بردارید . اگر می خواهید این مارک را داشته باشید، پول کمتری بدهید از این مدل انتخاب کنید . یا یکسری توضیحات بدهد که اگر شما مثلاً می خواهید برای کارهای گرافیکی استفاده کنید ، این مدل را بردارید. باز اگر می خواهید یک لب تاپ مادام العمر داشته باشید یعنی به همین زودی تغییر ندهید آن را ، از این مدل استفاده کنید. یعنی یکسری راهنمایی هایی می خواهد به شما بکند. اینجا هم همینطوری است.

(از اینجا به بعد این اسلایدهایی که داریم می آید یکسری قواعد سرانگشتی می دهدکه به ما کمک می کند در انتخاب یک (style) مناسب.)



این فصل در رابطه با مشهورترین پترن های معماری است که در ابتداکاتالوگ‌ها را معرفی کردیم و سپس جزئیات آن‌ها را بررسی کردیم. معماری‌های ما از چندین پترن تشکیل شده اند که به صورت ترکیبی استفاده می‌شوند.



این کاتالوگ چند دسته از پترن های معماری را بررسی می‌کند. مثل datacenter که به دو دسته هم تقسیم می‌شود. برخی از این پترن هاو نوعشان و صفات کیفی که این پترن ها برای رسیدن به آن‌ها به وجود آمده را بررسی کردیم.
در این اسلاید ناهمگونی‌هایی که در انواع پترن ها ممکن است وجود داشته باشد را بررسی کردیم که به سه دسته تقسیم بندی شوند.
گفتیم که هر کامپوننت از یک پترن در قالب یک پترن دیگر می‌باشد. جلسه قبل مثال‌هایی را بر اساس casestuoly های کتاب توضیح دادیم.
یک پترن اصلی برای سیستم تعریف می‌کنیم. و هر component از این پترن می‌تواند بر اساس پترن دیگری پیاده سازی شود. پترن های مختلف با هم، هم پوشانی دارند و ممکن است هر component جزء دو پترن باشد.
تنها معرفی یک کاتالوگ برای معرفی پترن ها کافی نیست. چطوری بفهمیم بر اساس صفات کافی مد نظرمان کدام پترن مناسب است؟ اگر در کنار کاتالوگ‌ها راهنمایی هم باشد که با استفاده از آن‌ها بتوانیم پترن مناسب را انتخاب کنیم خیلی بهینه تر است.
که کتاب راهنمایی لازم است که به ما بگوید اگر مشکلی مثل x را داشتیم راه حل آن y است. پس ما در این جلسه چند تا قانون را معرفی می‌کنیم.
در راستای این هدف که بتوانیم به خوبی پترن مان را انتخاب کنیم.
پترن جریان داده: از این پترن وقتی که احساس می‌کنیم سیستم ما یک تولید کننده خروجی خوش تعریف است و نتیجه باید به صورت ترتیبی استفاده شود. که خروجی‌ها مستقل از زمان هستند. اگر جریانی از تولید و مصرف داده در سیستممان داشتیم و واضح است که چه چیزی باید تولید شود و ارتباطات اجزاء کامل مشخص است، در این حالت از این پترن استفاده می‌کنیم. در این پترن خروجی یک بخش به عنوان ورودی بخش دیگر است. از این پترن در حالتی که جامعیت برای ما مهم است استفاده می‌کنیم و در این حالت interface ساده ای داریم.

در این اسلاید چند زیر اسلاید را معرفی کرده.

- دیتا flow یک نمونه از زیر استایل است. اگر ورودی و خروجی‌های ما به صورت یک دنباله بازگشتی و قابل تکرار باشد و ارتباط مستقیمی بین اجزای هر سری وجود داشته باشد یا به عبارتی یک شبکه ای از داده‌ها داریم. و اعضای این شبکه در مسیر برگشت داده هم هستند (مثل ring، star و .....) در این حالت از این مدل استایل استفاده می‌کنیم.
- Pipeline filter: یک مثال خط لوله که جریانی از داده در یک لوله حرکت می‌کند که پشت سر هم روی داده محاسبات مختلفی انجام می‌شود.
- Closed loop con: در حالتی که می‌خواهیم نظارت بر جریان داده داشته باشیم از این حالت استفاده می‌کنیم. هدف سیستم در این حالت این است که، انحراف‌ها و آشفتگی‌های غیر قابل پیش بینی خارجی را شناسایی کند و کنترل کند که الگوریتمی که استفاده می‌شود از مسیر خودش خارج نشود.

در این اسلاید بیان می‌کند که، وقتی مؤلفه‌های ما نمی‌توانند تا زمانی که پاسخ مؤلفه‌های دیگر را دریافت نکنند پیشرفتی داشته باشند از این استایل استفاده می‌شود. تا زمانی که مؤلفه ما پاسخی از مؤلفه‌های زیرین خود را دریافت نکند نمی‌تواند کارش را انجام دهد.

- Object- oriented

هر گاه نیازمندی‌های اصلی ما inerrability modify ability بود از این مدل استفاده می‌شود.

- ADT :Abs tract data type

اینجا هم بحث کپسول سازی را داریم ولی شی گرایی نداریم. و این تفاوتش با object oriented است. وقتی ما در سیستممان نوع داده‌های مختلفی را داریم (data type) و روش تعریف آن‌ها در سیستم‌های مختلف متفاوت است. و اگر تغییری ایجاد شد ما می‌خواهیم در نحوه استفاده آن‌ها تغییری ایجاد نشود. پس وقتی نوع داده‌های زیادی داریم و هر کدام در سیستمی متفاوت هستند از نوع داده انتزاعی استفاده می‌کنیم.

Objects: یک زیر مدل در این مدل می‌باشند.

وقتی ما نیاز به مخفی‌سازی داریم از این مدل استفاده می‌کنیم. از خصوصیات ارث‌بری استفاده می‌کنیم و ماژول پدری تعریف کنیم. به عبارتی این مدل ترکیب object ها، abstract (adt) هست، در این حالت ما یکسری ماژول‌هایی داریم که مشترکاتی دارند و ما اشتراک‌های آنها را بیرون می‌کشیم و به عنوان ماژول پدر از آن ارث‌بری می‌کنیم.

در بحث ارث‌بری از استایل object و در بحث کپسوله‌سازی از

استایل adt استفاده می‌کنیم.

برای داشتن ترکیب هردو از call and Return استفاده می‌شود.

- مدل client server: مدل call and Return مبتنی بر client server بیان می‌کند که هرگاه قابلیت تغییر باتوجه به تولید داده و چگونگی مصرف آن برای ما مهم باشد از الگوی client server استفاده می‌کنیم.

پنجمین زیر الگو از الگوی call and Return مدل Layered است.

هروقت یک application داریم که از آن می‌توانیم چندین application مختلف بیرون بکشیم و آنها را لایه لایه روی هم بزاریم و می‌خواهیم همه آنها از plat form محاسباتی که در زیر آنها قرار گرفته مستقل شوند از این مدل استفاده می‌شود.

اگر portability برای ما مهم است و اگر می‌خواهیم از یک لایه محاسباتی که توسعه پیدا کرده استفاده کنیم و روی آن می‌خواهیم سیستممان را سوار کنیم این مدل برای ما مناسب است.

پترن مؤلفه مستقل:

وقتی سیستم‌ها روی یک Plat form چند پروسسوره قرار است اجراء شود از این پترن استفاده می‌کنیم یا زمانی که سیستم ما قابلیت ساختاربندی دارد یعنی می‌خواهیم چندین کامپیوتر را به هم متصل کنیم (توزیع شده) که از Cpu همه آنها استفاده کنیم و اگر حافظه‌های مشترک هم داشته باشیم به این مجموع توزیع شده.

یک مؤلفه کار خودش را ادامه دهد با کمک مؤلفه‌های دیگر که هرکدام هم مستقل هستند.

Performance tuning: ما می‌خواهیم پروسس را در بین پروسس‌های مختلف reallocate کنیم یا کارمان را از یک پروسس به پروسس دیگر منتقل کنیم (جهت افزایش کارایی).

Re-allocate کردن کاربین پروسس‌های مختلف جهت توازن بار است. برای داشتن حداکثر Performance. که می‌توان وقتی بار کاری یک پردازنده که حجم کارش زیاد است را روی بقیه پردازنده‌ها تقسیم کرد. و یا اگر پروسس ما از چندین کار تشکیل شده و بخواهیم آنها را تقسیم کنیم از این مدل در این زمان‌ها استفاده می‌کنیم.

در کل این مدل مربوط به زمان اجرا است. این پترن نحوه ارتباط و اجرای فرآیندها را در زمان اجرا بیان می‌کند.

زیرالگوهای مؤلفه مستقل:

الگوی اول:

کافی است مکانیسمی برای تعادل بین مؤلفه‌ها تعریف کنیم مثل message Passing (تبادل پیام).

- Light weight presser :وقتی چند فرآیند داریم که از داده‌های مختلف استفاده می‌کنند
- Distributed object :وقتی مدل object oriented را داریم از این مدل استفاده می‌کنیم

الگوی دوم:

- Broadcast: وقتی در زمان‌هایی معینی مؤلفه‌ها باید با هم همگام باشند و یا وقتی که availability برایمان مهم است از این استایل استفاده می‌شود.

- Event system

وقتی ما می‌خواهیم از سیگنالهای ارسالی‌مان به چندین مشتری سرویس دهیم از این الگو استفاده می‌کنیم ویا وقتی که تعداد پروسس‌هایی که کار واحدی را انجام می‌دهند زیاد شوند.

Data centered style:

وقتی در شرایطی هستیم که از یک مخزن داده مشترک که باید طول عمر زیاد هم دارند استفاده می‌کنیم از این الگو استفاده می‌کنیم که این مخزن حجم داده های زیادی را باید مدیریت کند.

زیر الگوهای این استایل:

- زیر الگوی data base, repository رابطه‌ای که این دو دریک دسته هستند و در این دسته ترتیب اجرای مؤلفه‌ها به وسیله یک جریان از درخواست‌های ورودی مشخص می‌شود. برای دسترسی و یا به روز رسانی داده‌ها، و این داده‌ها کاملاً ساخت یافته هستند و بسته به نوع داده‌ها که رابطه‌ای وش رابطه‌ای یاشی‌گرا هستند پس داده‌ها ساختار مشخصی دارند پس از پترن داده متمرکز و زیر الگوی repository آن استفاده می‌کنیم ولی اگر داده‌های ما بخواهیم مصرف‌کننده را زیاد کنیم بدون اثری برتولیدکننده‌ها یعنی scalability و هم اینکه بخواهیم modifiability را از نظر اینکه چه کسانی تولید می‌کنند و چه کسانی مصرف می‌کنند داشته باشیم از black board استفاده می‌کنیم.

تفاوت black board و

transactional data base/ repository: b ← Active Publish است.

db ← passive است.

b پایگاه دانش داره ولی db معمولی یا repository می‌تواند پایگاه دانش نداشته باشد. از نمونه‌های black board مثل Publish subscribe.

استایل یا پترن Virtual machine: وقتی که ما می‌خواهیم محاسباتی را طراحی کنیم ولی هیچ ماشین ثابتی برای اجرا نداریم از این استایل استفاده می‌کنیم. یعنی وقتی ما یک application را می‌نویسیم و نمی‌خواهیم اهمیتی بدهیم که روی چه سیستمی اجرا خواهد شد از این پترن استفاده می‌کنیم.

ازاینجا تا آخر اسلایدهای این فصل در مورد مثالی که به آن اشاره خواهیم کرد صحبت شده.

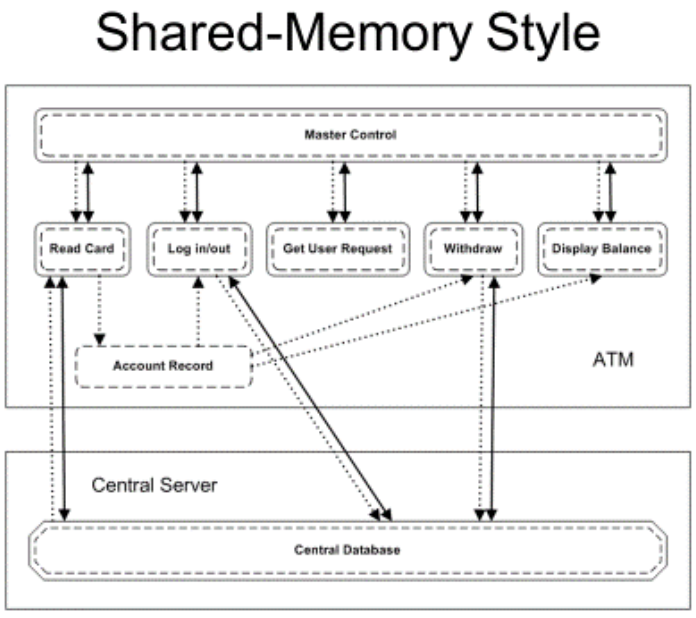
در این مثال پترن یک ATm معرفی شده. سه معماری را برای یک دستگاه ATm بررسی کرده و مشخص نموده که کدانم پترن چه صفات کیفی را پوشش می‌دهد.

User operation یا همان عملیات user یک ATm را در این بخش معرفی کرده که این عملیات شامل:

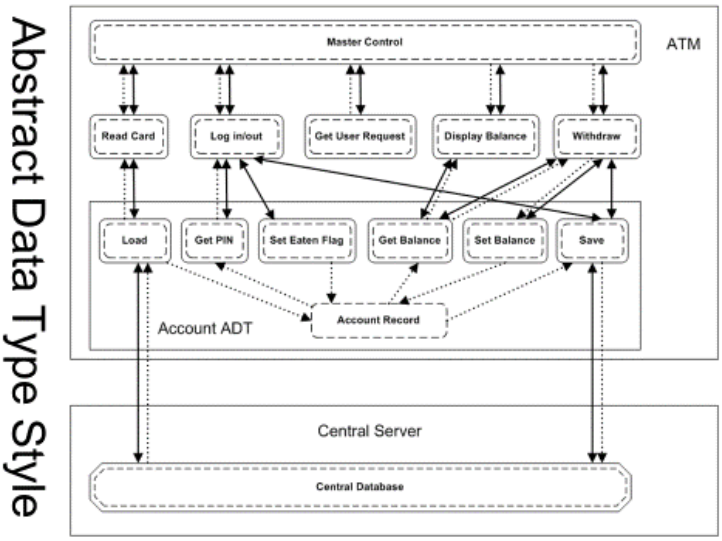
- گذاشتن کارت و وارد کردن Pin کد آن.
- برداشتن پول و – چک کردن حساب است.

در این اسلاید یک استایل shard memory را معرفی کرده از مسئله atm :

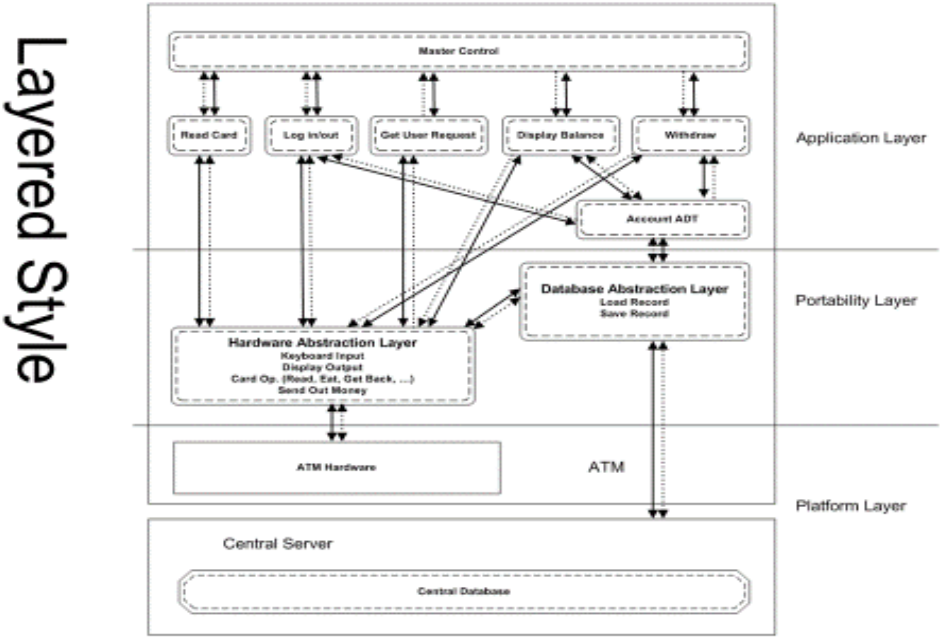
Shared- memory style:



یعنی استایل Data center: که در اینجا فرض می‌کنیم سرور مرکزی یک db متمرکز هست و ATmها Clientها هستند. در این شکل خود ATm ها را بررسی نکرده ولی خود Atmها یک کنترل مرکزی دارد که این کنترل روی ماژولهای خواندن کارت، ثبت ورودی و وخروجی‌ها و گرفتن درخواست از کاربر، برداشت پول و نمایش خروجیی کار می‌کند و ماژول Account Record هم در ارتباط با ماژول‌های خواندن کارت و ثبت ورودی و خروجی‌ها و برداشتن پول ومشاهده موجودی است. که این مدل گفته شده مدل data متمرکز بود.
در شکل بعدی مدل Abstract data type است. که در اینجا یک حساب مجازی نوع داده‌ها (انتزاعی) ایجاد می‌کند. در قبلی سه ماژول Log in/out, withdraw, Read card هرکدام خودشان به query, db که این نوع داده مجازی با استفاده از قسمت Save, Load اش. پس از Adt Account استفاده می‌کند.



درشکل بعدی از الگوی Layered استفاده کرده که در اینجا یک application Layer داریم که تمام کارهایی که ATm باید انجام دهد را با کمک همان Adt Account انجام می‌دهد. لایه Port ability یک لایه انتزاعی هم از db و هم از hardware ایجاد کرده یعنی application Layer ما با Port ability کار می‌کند و از طرف دیگرPort ability روی Plat form اصل سواد می‌شود که قسمت اصلی Plat form که data base abstraction Layer می‌باشد به یک Central server وصل می‌شود وبخش hardware آن هم سوار یک hardware ATm می‌شود. حال مزیتی که داریم این است که در این معماری، مدل دستگاه سخت‌افزاری، نوع بانک و اینکه Lodata center مربوط به چه بانکی و یا شعبه‌ای است هیچ اهمیتی ندارد. در جدول بعدی مقایسه و امتیاز دهی به سه نمونه معماری ذکر شده را داریم که امتیازدهی براساس صفات کیفی مانند Performance می‌باشد.



Analysis and Comparison

	Shared-Mem	ADT	Layered
Performance	3	2	1
Change account record format	1	3	3
New service: close account and withdraw the remained balance	1	2	3
Portability	1	2	3
Availability and Reliability	2	2	2
Buildability and Integrability	1	2	3
Sum	**9**	**13**	**15**

در shared memory بالاترین Performance را داریم زیرا ماژولها مستقیم با db کار می‌کردند و می‌دانیم که اضافه کردن لایه یعنی کار اضافه و پایین بردن Performance. در سطح account هرتغییری که ایجاد شود روی adt و Layered تأثیری ندارد ولی روی shared memory تأثیر می‌گذارد.

New service: دراین بخش اضافه کردن هر سرویس جدید مثل برداشتن باقیمانده حساب یا بستن حساب در معماری Layered بسیار راحت و آسان است نسبت به دوتای دیگر زیرا کافی است این سرویس را در لایه application اضافه کنیم و به بقیه لایه‌ها هیچ ارتباطی پیدا نمی‌کند و در ADT کمی بهتر است ولی در shared خیلی بد است.

Port ability: یعنی حمل کردن applicationاز روی یک Plat form به Plat form دیگر که مسلماً در Layered بهتر از بقیه است و راحت‌تر می‌باشد و Layered با هدف ایجاد Port ability ایجاد شده.

از نظر قابلیت اطمینان ودسترس (availibity) هرسه معماری در یک سطح می‌باشند.

از نظر قابلیت ساخت و یکپارچگی در Layered بالاتر است و می‌توان از لایه‌های آماده سیستم‌های دیگر راحتتر است (integrabiliy).

پس مزیت shared memory فقط درPerformance است.

در امتحان ممکن است همین مثال‌های کتاب داده شور مثل ATm که ازروی آن صفات کیفی و اصلی و سناریو و تاکتیک‌ها را بنویسیم.

باتوجه به مطالب گفته شده در جدول فوق دیدیم که Layered بیشترین مزیت را دارد.

استایل‌های ادامه قبلاً درس داده شده و مثال دیگری است که می‌توانید از کتاب آن را مطالعه کنید.

فصل ۷: طراحی معماری Designing the Architecture

از الان می‌خواهیم در مورد طراحی یک معماری صحبت کنیم در این بحث ۴ تست داریم:

۱- معماری نرم‌افزار در چرخه تولید نرم‌افزار که همان مباحث یادآوری جلسه اول را شامل می‌شود که جایگاه معماری را درچرخه حیات بیان می‌کند که در توسعه تدریجی معماری را جای دادیم و نقش معمار را مشخص کردیم و اسناد تولیدی را در فازهای مختلف بیان کردیم.

۲- در قسمت بعدی در مورد طراحی معماری و انجام آن صحبت می‌کند.

۳- ساختار تیم و شکل‌دهی اعضای تیم را برای طراحی معماری را بیان می‌کند.

۴- ایجاد یک اسکلت از سیستم که در واقع طراحی‌ای را بعد از ایجاد معماری توصیف می‌کند که در واقع اگر فازهای چرخه حیات سیستم را تعیین نیازمندیها- طراحی معماری و طراحی جزئیات براساس معماری و پیاده‌سازی براساس معماری و استقرار کردن سیستم و تست جامعیت و پشتیبانی در نظر بگیریم، ایجاد اسکلت سیستم یعنی شروع فاز پیاده‌سازی و یا طراحی ریز جزئیات سیستم براساس معماری.

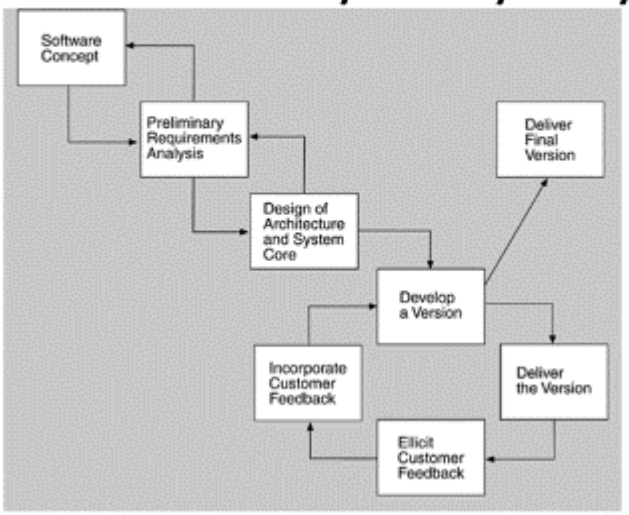
معماری در چرخه حیات: در شکل چند اسلاید پایین‌تر کامل و واضح توضیحات لازم نشان داده شده.

در چرخه حیات توسعه تدریجی معماری را جای می‌دهیم- یعنی وقتی مفاهیم معماری را به دست آوردیم و براساس آن مفاهیم، نیازمندیهای اصلی (nonfunctional- functional) (هم use case وهم سناریوها) را تحلیل کردیم وبیرون کشیدیم، با تکرار (فلش رفت و برگشت

در شکل) می‌توانیم طراحی معماری را شروع کنیم. وقتی ما راه‌اندازهای معماری را کشف کردیم (Driverها)، طراحی معماری آغاز می‌شود. معماری ممکن است در یک رفت‌وبرگشت (از سمت چپ شکل از بالا، بین state دوم و سوم که طراحی معماری و هسته سیستم می‌باشد) بر روی نیازمندیهای ما و یا اولویت آنها اثربگذارد و تغییر ایجاد کند. پس براساس نیازمندیهای تأییدشده معماری طراحی می‌شود و بعد از آن وارد فاز طراحی و پیاده‌سازی می‌شویم و چون در توسعه مشتری می‌دهیم تا با آن کار کند و باز خوردهایی که از مشتری دریافت می‌کنیم را بررسی می‌کنیم و مجدد روی دو ژنمان کار می‌کنیم تا تکمیل شود و سپس گذشتن دوره تکامل این حلقه ورژن ما تکمیل می‌شود.

Architecture in the Life Cycle

Figure 7.1. Evolutionary Delivery Life Cycle

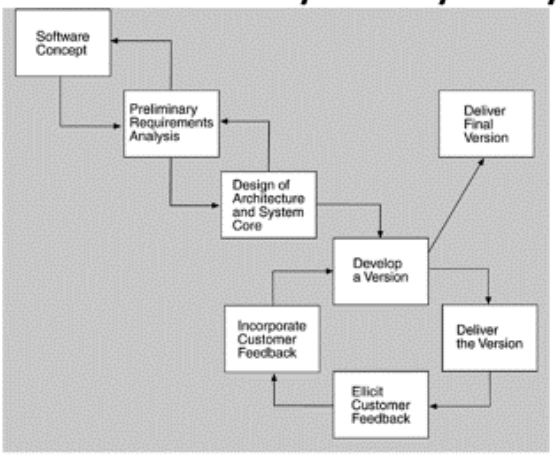


چه موقع می‌توانیم طراحی معماری را شروع کنیم؟

این سؤال مهم می‌باشد همانطور که در مدل چرخه حیات می‌بینیم طراحی معماری به صورت یک کار بازگشتی می‌باشد با تحلیل نیازمندیهای اصلی واضح است که ما نمی‌توانیم طراحی معماری‌مان را شروع کنیم مگر اینکه دیدی از نیازمندیهای سیستم داشته باشیم. در واقع وقتی که ما Driverها یا راه‌اندازهای معماری راشناسایی کردیم می‌توانیم طراحی معماری‌مان را شروع کنیم که سؤالاتی که در طول طراحی معماری شروع می‌شود روی پروسه تحلیل نیازمندیها اثر می‌گذارد. پس به همین خاطر در شکل فلش دو طرفه داریم.

Architecture in the Life Cycle

- **Figure 7.1. Evolutionary Delivery Life Cycle**



معماری به وسیله مجموعه‌ای از نیازمندیهای کسب و کار و صفات کیفی و عملیاتها شکل داده می‌شوند پس راه‌اندازهای معماری نیازمندیهای اصل کسب وکار، صفات کیفی و functionalityهای ما هستند یعنی یک تعدادی از سناریوهای اصلی و یک تعدادی از use case های اصلی و نیازمندیهای کسب و کار که به عنوان مثال درفصل ۳ و ۶ و ۸ چند case study و مورد هرکدام راه‌اندازها را بیان کرده مثلاً در فصل ۳ در طراحی خلبان خود کار، معماری A7E، نیازمندیهای Performance, modify ability راه‌اندازهای آن هستند. در فصل ششم که سیستم کنترل ترافیک هوایی است راه‌انداز معماری آن availability هست.

در شبیه‌ساز پرواز در فصل ۸ راه‌اندازهای معماری کارایی و قابلیت تغییر است.
در نمونه‌سؤالات یک معماری درب اتوماتیک گاراژ را داده که ساختار تجزیه ماژول آن است و گفته صفات کیفی اصلی را نام ببرید که این خودش بخشی از راه‌اندازهای معماری است و در ادامه گفته براساس صفات کیفی برای آن معماری، سناریو بنویسید و سپس تاکتیک‌ها را براساس سناریو معرفی کنید.
مثال‌های کتاب را برای نمره کامل مطالعه کنید.
برای مشخص کردن راه‌اندازهای معماری ما بالاترین اهداف کسب و کارمان را مشخص می‌کنیم که باید تعدادش کم باشد که اهداف کسب و کار براساس سناریوها و use caseها تعریف کرده‌ایم و این اهداف را لیست می‌کنیم و از این لیست آنهایی را که بیشترین اثر را روی معماری دارند انتخاب می‌شوند که معمولاً تعداد انتخابی‌ها کمتر از ۱۰ تا هست و سپس به روش ATAM: Architecture Tradeoff Analysis method:
که متد تحلیل سود وزیان معماری می‌باشد استفاده می‌کنیم.
که در این متد یک روش به نام درخت سودمندی دارد که به ما کمک می‌کند تا از روی راه‌اندازهای کسب وکار، صفات کیفی‌مان را بیرون بکشیم. و از این روش در تعیین راه‌اندازهای معماری استفاده می‌کنیم.
پس از روش ATAm در تعیین راه‌اندازهای معماری استفاده می‌کنیم سپس راه‌اندازهای معماری کمتر از ۱۰ تا دونه از صفات کیفی و use caseها هستند که از روش ATAm برای تعیین‌شان استفاده کردیم می‌باشند.
متدی برای طراحی معماری داریم که هم‌نیازهای کیفی و هم عملیاتی را برای ما فراهم می‌کند که به این روش

Attribute- Driver- Design (ADD) گوییم یعنی طراحی صفت‌گرا: ADD یک الگوریتم بازگشتی است با سه گام اصلی.

۱- ورودی Add مجموعه‌ای از صفات کیفی و سناریوهای صفات کیفی است و دانش مربوط به دستیابی به این صفات کیفی و معماری را به کار می‌برد به منظور دستیابی به طراحی معماری. متد ADD به صورت یک توسعه‌ای از متدهای توسعه دیگری مانند Rational unified process (RUP) مشاهده می‌شود. در واقع ADD یک ورژن توسعه‌یافته مثل RUP یک متدولوژی است برای تولید سیستم‌های مبتنی بر معماری.
ADD یک شیوه برای طراحی معماری نرم‌افزاری است که مبتنی است بر پروسه تجزیه صفات کیفی سیستم تا کامل شدن سیستم. ADD طراحی صفت‌گراست و یک متدولوژی مبتنی بر معماری است. ADD براساس پروسه تجزیه ماژول کار می‌کند و یک پروسه تجزیه بازگشتی است و در هرمرحله‌اش تاکتیکها و پترنهای معماری‌اش انتخاب می‌شوند برای اینکه ما را به مجموعه‌ای از سناریوهای کیفی برساند و سپس کارهایی را که سیستم باید انجام دهد را انتصاب می‌دهد به ماژول‌های مربوطه که در پترن معرفی شدند که مشخص شود چه ماژولی چه کاری باید انجام دهد. ADD در چرخه حیات نرم‌افزار بعد از تحلیل نیازمندیها قرار گرفته و وقتی راه‌اندازهای معماری‌مان را تا حد قابل اطمینانی شناسایی کردیم می‌توانیم ADD را شروع کنیم.

۲- مرجعی ADD دید تجزیه ماژول است. می‌دانیم که ۴+۱ دید (view) داریم. می‌دانیم که معماری در آخر کار یکسری نمودار روی کاغذ را به stack holderها تحویل می‌دهد که این نمودارها همان دیدهاست. ما معماری‌مان را با استفاده از یکسری view و دید مستند می‌کنیم درنهایت تمام صحبت‌هایی که از اول ترم بیان شده جهت پیاده‌سازی ADD می‌باشد. ما وارد جزئیات طراحی ADD نمی‌شویم و تفاوت بین معماری تهیه تولید شده توسط ADD با معماری که آماده پیاده‌سازی است در ریز جزئیاتی است که در فاز Design تعریف می‌شود. ما معماری را طراحی می‌کنیم و بعد یک فاز طراحی و یک فاز پیاده‌سازی داریم. فاصله بین فاز طراحی و پیاده‌سازی را یکسری جزئیات پرمی‌کند که در فاز طراحی بیان می‌شوند. پس معماری طراحی شده به وسیله ADD می‌تواند به صورت ذاتی تصمیمات را به تعویق بیندازد تا مرحله طراحی که بتواند انعطاف‌پذیری بالاتری را داشته باشد.
پس داشتیم که ورودی‌ها به ADD مجموعه‌ای از نیازمندیهاست، نیازمندیهای functional که در قالب یک use case بیان می‌شود، محدودیت‌هایی که روی آن ورودی که وجود دارد و متدهای طراحی مختلفی که برای آنها است و یکسری از نیازمندیهای کیفی که براساس صفات کیفی خاص سیستم تعریف می‌شوند.

ADD steps: سه تا step دارد:

گام ۱) یک ماژول را برای تجزیه انتخاب می‌کنیم درلحظه اول که ADD شروع می‌شود این ماژول کل سیستم ما است. در لحظه اول تمام نیازمندیهای ما ورودی‌های به این ماژول هستند یعنی همان موارد بالا که شامل functional Requirement ها، محدودیت‌ها و quality Requirement ها.

نکته: این سه گام به صفت بازگشتی با انتخاب یک ماژول در هر مرحله انجام می‌شود و هر بار هر سه گام را رفته و مجدد برمی‌گردیم به گام اول و یک ماژول دیگر انتخاب می‌کنیم فقط در اولین مرتبه، اولین ماژول انتخابی، کل سیستم ما منظور می‌شود.
۲-گام دوم: نحوه تجزیه ماژول‌ها به ماژول انتخابی: اول راه‌اندازهای معماری را از مجموعه سناریوهای صفات کیفی و نیازمندیهای functionalمان انتخاب می‌کنیم. این مرحله مشخص می‌کند که برای تجزیه چه چیزی برای ما مهم است.
b- در این قسمت یک پترن معماری انتخاب می‌کنیم که ما را به driverهای معماری برساند. پترن را یا می‌سازیم یا انتخاب می‌کنیم البته براساس تاکتیک‌هایی که ما را به درایورهای مدنظرمان می‌رساند و وقتی پترن انتخاب می‌شود فرزندان ماژول‌ها که برای پیاده‌سازی تاکتیک‌ها موردنیازند شناسایی می‌شوند. زیرا که پترن به ما می‌گوید که چه عضوهایی باید داشته باشیم مثلاً وقتی استایل Client/ Server را انتخاب می‌کنیم یکسری عضو Client داریم و یکسری عضو Server. یعنی چند تا زیر ماژول تولید کردیم و از ماژول اصلی تبدیل شد به چند تا Client و یکی دوتا سرور.
C- کارها و functionalityهایی که سیستم باید انجام دهد را به ماژول‌هایمان انتصاب می‌دهیم یعنی بیان می‌کنیم که چه Clientهایی چه کارهایی بکند و Serverها چه کارهایی بکند که می‌توانیم از Viewها جهت اینکار استفاده کنیم.
d. واسط و روابط بین بچه‌ها را مشخص می‌کنیم دراین بخش به طورمثال روابط بین Clientها و سرورها را تعیین می‌کنیم. ساختار تجزیه ماژول‌ها و محدودیتی که روی تعادل بین آنها وجود دارد در این بخش به ما کمک می‌کند و در نهایت این ارتباط را مستند می‌کنیم که چگونه است یعنی ارتباط بین ماژول‌ها را سند می‌کنیم.
e. اعتبارسنجی و صحت‌سنجی انجام می‌دهیم روی تجزیه‌ای که انجام دادیم وپترنهایی که انتخاب کردیم که چیزی از قلم نیوفتد.
گام ۳. در این مرحله همه stepها را مجدد تکرار می‌کنیم براساس ماژول‌هایی که باید تکرار شوند یعنی ماژول‌های فرزند را برای تجزیه بیشتر انتخاب می‌کنیم و دوباره از فاز یک شروع می‌کنیم به انجام ۳ گام روی این ماژول فرزند ماژول فرزند قبلی واین کار را تا زمانی که ماژول دیگری برای تجزیه بیشتر وجود نداشته باشد ادامه می‌دهیم.
سپس ADD یک فرآیند طراحی down به Top مبتنی بر نیازمندیهای کیفی یا صفات کیفی برای تعریف Patternهای معماری مناسب و روی استفاده از نیازمندیهای کیفی برای معرفی نوع ماژول‌ها که پترنها ارائه داده‌اند.

شکل‌دهی ساختار تیم **Formatting the team structure**

بخش سوم این فصل نحوه انتخاب اعضای تیم پروژه و شکل‌دهی به ساختار تیم پس از طراحی معماری است. وقتی چند سطح اول از ساختار تجزیه ماژول ساخته شد و پایدار شد این ماژول‌ها را می‌توان به تیم‌های توسعه انتصاب را در جهت پیاده‌سازی تولید آنها، قبلاً در مورد دید work assignment در 4+1 دیدی که داشتیم جهت کردیم (فصل ۲) که خودش از نگاه stockholderها به ساختار انتصاب کار که از ساختار Allocation structure می‌آمد ایجاد می‌شد. برای انجام فرم‌دهی ساختار تیم می‌توان از این ساختار یا structure استفاده کنید. برای انتصاب ماژول‌ها می‌توان از واحدها و اعضای موجود استفاده کرد و یا اعضای جدید اضافه کرده و تعریف کرد.
سؤال: چرا ساختار تیم ما آینه‌ای از ساختار تجزیه ماژول است؟ یعنی چرا اول ساختار تجزیه ماژول را می‌سازیم و بعد اعضای تیم را برای آنها می‌گذاریم؟ بخاطر information hiding چون قرار است هر ماژول به طور مستقل پیاده‌سلز می‌شود اطلاعات Private خودش بدون وابستگی به ماژول‌های دیگر پیاده‌سازی شود پس می‌توان به صورت موازی این ماژول‌ها را پیاده‌سازی کرد (در تیم‌های مجزا) – دلیل دیگر این است که ما کاری کردیم که هرکدام ماژول‌های روی دامنه کوچکی مشارکت داشته باشد پس خبرگی و دانش خاص خودش را فقط لازم دارد و ما می‌توانیم بسته به نوع و کار ماژول متخصص خاص خودش را انتخاب کنیم و علت سوم این است که یک نگاشت ذاتی بین تیم‌ها و ماژول‌های ساختار تجزیه ماژول وجود دارد که مثال‌هایی در کتاب در این رابطه ذکر شده.
شکل‌دهی اسکلت سیستم:
بیان می‌کند که وقتی یک معماری به خوبی طراحی شد و اعضای تیم به درستی سرجای خودشان قرار گرفتند حال ما می‌توانیم یک اسکلت از سیستم‌مان بسازیم. این اسکلت یک چهارچوبی است که یک توسعه تکراری را برای ما فراهم می‌کند. (alterative develop).
ایده اسکلت سیستم یک توانایی را برای پیاده‌سازی functionalityهای سیستم را به صورت مؤثر فراهم می‌کند.
اگر بدن انسان را بخواهیم مهندس مجدد کنیم اول به ساختارش دقت می‌کنیم و اول اسکلت را می‌سازیم بعد ماهیچه‌ها و رگ‌ها و اعصاب و جاسازی اندام‌ها و ... که این روند تکرار دارد و یک functionality است. در واقع ما یک functionality داریم که به مرور functionalityهای دیگر به آن اضافه می‌شوند. پس ما در این بخش یاد می‌گیریم که براساس معماریمان چگونه یک اسکلت از سیستم‌مان بسازیم. که چند فاز را معرفی کرده (۳ تا فاز) برای ساختن اسکلت سیستم و در آخر مزیت ساخت این اسکلت را بیان کرده. با استفاده ازمعماری به عنوان راهنما ترتیب موردنیاز برای ما واضح می‌شود.

اول ما باید زیر ساختاررا آماده کنیم یعنی نرم‌افزار یا سخت‌افزارهایی را ایجاد کنیم و تهیه کنیم که قرار است معماری روی آن سوار شود. مثلاً در یک سیستم بلادرنگ اول یک زمانبند تهیه کنیم. اگر یک سیستم rule based داریم اول یک موتور Rule یا به عبارتی موتور fire پیاده‌سازی کنیم. اگر یک سیستم Client/Serverای داریم اول باید یک سیستم هماهنگ‌کننده Clientها و Serverها را پیاده‌سازی کنیم.

اغلب مکانیزم‌های تعاملی اصلی به وسیله میان افزارهای نسل سوم فراهم می‌شوند که در این مورد کارها تک تک آماده می‌شوند و نصب می‌شوند به عنوان نمونه‌ای از پیاده‌سازی یعنی برای بحث interaction از میان‌افزارها استفاده می‌شود که این می‌تواند یک موضوع تحقیق باشد.

۲-حال که زیرساخت را ساخته‌ایم درمرحله دوم ساخت اسکلت سیستم elementهایی که functionalityهای سیستم ما را فراهم می‌کنند و باید اضافه شوند به سیستم را انتخاب می‌کنیم. این انتخاب مبتنی بر functionalityهایی است که کمترین ریسک را برای ما دارند و یا مبتنی بر این است که functionalityهایی را که عجله بیشتری برای تحویل آنها به بازار داریم و یا اینکه آنهایی را انتخاب کنیم که نیروی متخصص لازم برای تولیدشان را داریم.

۳- دراین بخش بیان می‌کند درادامه ایجاد اسکلت سیستم پس از مراحل ۱ و۲ حال با استفاده از ساختار uses که بیان‌گر این بود که چه ماژولی از چه ماژول‌های دیگری استفاده می‌کند، functionality دومی که باید درمرحله دوم استفاده شود را انتخاب کنیم یعنی ما مدام از مرحله ۳ به ۲ و ۲ به ۳ حرکت داریم تا تمام functionalityها انتخاب شوند یعنی در مرحله ۳ ما functionality پیدا می‌کنیم و در مرحله ۲ آن را به اسکلتمان اضافه می‌کنیم پس در فاز ۱ زیرساخت را می‌سازیم در فاز ۲، functionality اصلی را می‌سازیم و در فاز سوم با استفاده از uses functionality بعدی را پیدا می‌کنیم و دوباره در فاز ۲ آن را به سیستم اضافه می‌کنیم.

پس پروسه ساخت سیستم براساس اسکلت سیستم یک پروسه پیوسته و افزایشی است وسیستم مرتب رشد می‌کند تا کامل شود و مزیتهایش شامل: مدیریت تحویل به بازار آسان‌تر می‌شود، integration و تست آن راحتتراست، پیدا کردن faultها آسانتر می‌شود، بودجه و زمان‌بندی آن قابل پیش‌بینی‌تر می‌شود زیرا incrementهای کوچک کوچک داریم تا به بزرگ و قابل پیش‌بینی‌تر است.

فصل ۱۱

در این بخش به ارزیابی می پردازیم(یعنی معماری مستند شده حال می خواهیم ارزیابی اش کنیم).
یکی از روش های ساده برای ارزیابی ATAM می باشد.
ATAM مخففArchitecture Tradeoff Analysis Method می باشد.
یعنی متد تحلیل (سبک و سنگین کردن) معماری،پس میتوان گفت ATAM یک متد کیفی است(کمی نمی باشد)که معماری را سبک و سنگین می کند و می گوید سناریوها و فانکشن ها کدام خوب و مناسب و کدام نامناسب می باشد و کدامها باهم تلاقی دارند.
یک روش سراسری و جامع برای ارزیابی کردن نرم افزار می باشد که بیان می کند یک معماری چه مقدار توانسته به خوبی اهداف کیفی خاصی را پوشش دهد.
و صفات کیفی که وجود دارد یک +++ را برای ما فراهم می آورد.
مفهوم کلی Tradeoff : یعنی سود و زیان یک کار را بررسی کنیم که آیا انجام دهیم یا خیر.
ATAM سناریوهایی را که در فصل ۴ بیان شد را اولویت بندی میکند و سپس یک سبک سنگین می کند تا ببیند کدام سناریو برای پیاده سازی بهتر است. برای سبک سنگین کردن تنها معمار نیست که قدرت تصمیم گیری دارد، مشتری و پیاده سازان سیستم و کاربر پایانی باید در این بحث به ما کمک کنند.
در نتیجه این روش ارزیابی یک روش مشارکتی می باشد.
ارزیابی معماری چه چالش هایی دارد؟
معماری برای یک سیستم نرم افزاری بزرگ معمولاً بزرگ می باشد و فهم و درک آن در یک زمان کوتاه سخت می باشد و همچنین این پروسه ارزیابی نیاز دارد که ما اهداف کسب و کار را به تصمیمات تکنیکی که معماری را می سازد متصل کنیم جمع آوری یک perspictives از چندید Stakeholder کار ساده ای نیست
اینها چالش هایی هستند که ما با آنها روبرو هستیم
پس ما در ارزیابی هم باید تصمیمات اکتیکی خود را در نظر بگیریم و اینکه نظرات هه Stakeholder ها را جمع آوری کنیم که این کار سختی می باشد.
پس همانطور که ملاحظه می فرمایید مهمترین چالش ما زمان است. یعنی ما این ارزیابی را نمیخواهیم چند ماه به طول بیانجامد، میخواهیم برای مثال ظرف یک هفته جمع کنیم، ازین لحاظ گفته می شود زمان یکی از مهمترین چالش ها می باشد.
مشارکت کنندگان در عملیات ارزیابی ATAM

- اعضای تیم ارزیاب (ممکن است یا از داخل گروه باشد و یا یک تیم جداگانه)
- تیم تصمیم گیرندگان پروژه (کسانی که حرف آخر را میزنند)
- Stakeholder های معماری (همه Stakeholder هایی که ذکر کردیم)

اعضای تیم ارزیاب

- خارج از پروژه هستند
- ۳ تا ۵ نفر می باشند
- بی طرف می باشند : یعنی هیچ ارتباطی با پروژه ندارند و هیچگونه رابطه ای وجود ندارد که در تصمیم گیری آنها دخالت داشته باشد
- آشنایی کامل با معماری داشته باشند

تصمیم گیرندگان پروژه

- کسانی که در مورد توسعه پروژه می توانند صحبت کنند
- مدیر پروژه
- مشتریی که می تواند تصمیم گیری کند
- معمار
- کسی که وظیفه او ارزیابی سیستم می باشد

در مراحل ارزیابی معماری به این افراد مشاوره داده می شود که چه اضافه و یا کم شود
Stackeholder های معماری چه کسانی هستند؟

- کسانی که معماری را به عنوان یک راهنمایی کننده میبینند
- شامل : توسعه دهنده ها، تست کننده ها، پشتیبانی، متصل کننده های قسمت مختلف معماری،مهندس ها، کاربران پایانی و تعامل ایجاد کنندگان بین سیستم خود و سیستم های دیگر
- شامل ۱۲ تا ۱۵ Stakeholder می باشند

پس ما سه دسته مشارکت کننده در پروسه ارزیابی داریم

خروجی های ATAM چیست؟

- اصلی : خروجی های ارزیابی معماری
- ثانویه : خروجی هایی هستند که بر اثر انجام عمل ارزیابی تولید می شوند
- غیرقابل لمس : به اندازه دو خروجی بالا اهمیت دارد و دارای اثر طولانی تری می باشند، یعنی خروجی طولانی مدت هستند

نکته: خروجی های غیرقابل ملموس چه می تواند باشد؟ رفتار سازمان – تعاملات – (مثلا در بوتیک خود علاوه بر تبلیغات و ... مقداری قیمت خود را پایین تر از بوتیک های همسایه قرار می دهید که نتیجه آن در بلند مدت دشمنی بوتیک های همجوار و کتک کاری و ....)

خروجی های اصلی ATAM

- نمایش مختصر معماری
- طبقه بندی از هدف های کسب و کار
- نیازمندی های کیفی در قالب سناریو (سناریوهای صفات کیفی)
- نگاشت بین تاکتیک ها و صفات کیفی (تاکتیک ها پیاده سازی صفات کیفی)
- مجموعه ای از نقاط حساس و نقاط Tradeoff شناسی شده: نقاط حساس و نقاط Tradeoff در مورد تاکتیک هایی می باشند که شما انتخاب می کنید تا به یک صفت کیفی برسید و فرق بین آنها مثلا بخواهیم به صفت قابلیت دسترسی برسیم از تاکتیک فزونگی (اکتیو یا پسیو) این ایجاد افزونگی نقطه حساسش این است که اعمال سازگاری برایمان حساس میشود.پس ایجاد consistency بین نسخه های متفاوت از یک نمونه داده برایمان حساس می باشد ولی خود consistency اعمالش برای ما نقطه حساس است که اگر بخواهیم به آن برسیم برایمان یک هزرینه ایجاد میکند، که بتوانیم به همگام سازی را داشته باشیم که tradeoff دارد باperformanc.یعنی اگر بخواهیم Backup داشته باشیم، کنترل و بروز رسانی هزینه دارد و performance را پایین می آورد،پس باید سبک سنگین کنیم که آیا بنفع ماست که performance پایین بیاید ، هزینه ایجاد گردد، سرعت اجرا کم گردد ولی قابلیت دسترسی بالا داشته باشیم؟! این موضوع بستگی به نوع سیستم درحال پیاده سازی دارد.
- شناسایی مجموعه ای Risk ها و non-risk ها (ریسک: یک تصمیم معماری که منجر می شود به عواقب غیرمطلوب در رسیدن به نیازمندیهای صفات کیفی) شاید بپرسید چرا نقاط non-risk را نیز مشخص کنیم علتش این است که نقاط بدون ریسک را مشخص می کنیم که شاید این نقطه از نظر ما دارای ریسک نباشد ولی در عمل داراری ریسک باشد، پس آن نقاط را هم کاملا مشخص می کنیم.
- زمینه های Risk : زمینه ریسک دلایلی هستند که ریسک را ایجاد می کند. برای حل ریشه ای ریسک باید ریشه آن را شناسایی کرد)

همه این خروجی ها در گزارش نهایی نوشته می شود.

خروجی های ثانویه

- نمایشی مجدد از معماری، که ایجاد شده که حتی می تواند بهتر از گفته های قبلی باشد
- سناریوهایی که به سناریوهای قبلی اضافه می شود (براساس صحبت با مشارکت کنندگان در قسمت کسب و کار که سناریو ها اولویت بندی می کنیم و سناریوهایی که ما ندیده بودیم و Stakeholder ها دیدند را اضافه می کنیم)
- تحلیل گزار نهایی که منطق هایی را ارائه می دهد تا دلایل تصمیمات نمایش داده شود

خروجی های غیرقابل لمس

- حس مشارکت بین stakeholder ها
- ایجاد کانالهای باز ارتباطی بین معمار و stakeholder ها (مانند تماس تلفنی)
- فهم و درک بهتر معماری و نقاط ضعف و قدرت از دید مشارکت کنندگان

فاز ها

- فاز ۰ : شراکت وآماده سازی اولیه (تعیین مشارکت کنندگان)
- فاز ۱: ارزیابی
- فاز ۲: ادامه ارزیابی (مشارکت کنندگان در این فاز با فاز قبلی فرق می کنند)
- فاز ۳: پیگیری

فاز ۰ : آماده سازی

- رییس تیم ارزیابی و تصمیم گیرندگان پروژه به صورت غیر رسمی باهم ملاقات می کنند،خارج از کار و قبل از شروع کار و یکسری بحثهای جزئی می کنند

- در مورد منطقها و استدلال ها : مشخص می کنند stakeholder ها چه کسانی هستند،گزارش نهایی زمانبندی را تعیین کنند و مشخص می کنند فازها به چه صورت باشد و چه چیزهایی برای فاز ۱ آماده کنیم

فاز ۱ : ارزیابی

- تیم ارزیابی با تصمیم گیرندگان اصلی ملاقات می کنند
- اطلاعات را جمع آوری و تحلیل می کنند
- جزئیات را برای مراحل بعدی مشخص می کنند

فاز ۲ : ادامه ارزیابی

- Stackholder های معرفی شده برای دسته سوم مشارکت کنندگان، در اینجا اضافه می گردند
- تحلیل معماری
- جزئیات بیشتر در مرحله بعد بحث می شود

فاز ۳ : پیگیری

- گزارشات نهایی تهیه می شود
- یک جلسه ای برگزار می شود تا در مورد اتفاقات پس از واقعه صحبت کنند و جوانب مختلف مورد ارزیابی قرار گیرد (سبک و سنگین کردن:چه دیده شد،چه کارهایی خوب پیش رفته است و چه کارهایی بد و...)

ATAM: Phases

Phase	Activity	Participants	Typical uration
0	Partnership and preparation	Evaluation team leadership and key project decision makers	Proceeds informally as required, perhaps over a few weeks
1	Evaluation	Evaluation team and project decision makers	1 day followed by a hiatus of 2 to 3 weeks
2	Evaluation (continued)	Evaluation team, project decision makers, and stakeholders	2 days
3	Follow-up	Evaluation team and evaluation client	1 week

ATAM: Architecture Tradeoff Analysis Method Sharif University of Technology 19

فاز ها	فعالیت	مشارکت کننده گان	مدت زمان اجرا
۰	مشارکت و آماده سازی	رهبر تیم ارزیاب و تصمیم گیرندگان پروژه	بصورت غیر رسمی – چند هفته
۱	ارزیابی	کل تیم ارزیابی و تصمی گیرندگان	از ۱ تا ۳ روز
۲	ادامه ارزیابی	کل تیم ارزیابی و تصمی گیرندگان به همراه Stakeholder ها	۲ روز
۳	پیگیری	تیم ارزیابی (تهیه گزارشات)	۱ هفته

مرور: atam یک روش ارزیابی جامع و کاملی است که برای ارزیابی نرم افزار انجام می دهیم واین یک روش کیفی است و کمی نیست یعنی به ما عدد و مقدار نمی دهد و پایه و اساس این روش بر اساس این است که می آید یک سری سناریوها و عملکردهای اصلی بوده و اهداف کسب و کار اصلی ما را در نظر گرفته و ارزیابی می کند تا ببنید معماری آنها را چقدر پوشش داده است و آنها را اولویت بندی می کند.

شرکت کنندگان ATAM: سه دسته شرکت کننده داریم

۱) تیم ارزیاب(۳ تا ۵ نفره): تیم ارزیاب می تواند از بیرون پروژه هم باشد.

۲) تصمیم گیرندگان اصلی که معمار هم جزو آن هست.

۳) Architecture stakeholder (شامل بازار- مشتری – واحد تست- پشتیبانی و برنامه نویس وکاربر پایانی...)

خروجیATAM: که چند دسته خروجی دارد

۱) خروجی اصلی: ۱- شرح مختصری از معماری را دارد- ۲-طبقه بندی از اهداف کسب و کار – ۳-نیازمندی صفات کیفی که در قالب سناریو معرفی می شود- یک سری تصمیمات معماری یا استراتژیهای معمار یا تاکتیک ها در واقع روشهایی که برای پیاده سازی سناریوها اعمال می کنیم– مجموعه ای از نکات حساس و tread off ها که نقطه ی حساس: مقدار صفات کیفی از یک نقطه نظر برای ما اهمیت خاصی پیدا می کند و نیاز به کنترل داشته و نقطه Tradeoff : در مورد تعامل بین صفات کیفی مختلفی است که ممکن است مقدار صفت کیفی روی یک یاچند صفت کیفی دیگر اثر منفی یا مثبت داشته باشد.

مجموعه ای از risk ها و non - risk ها رو داریم: واقعه یا حالتی که مقدار صفت کیفی مارا بحرانی می کند.

زمینه ریسکی: وضعیتی که ممکن است ما را به ریسک برساند یعنی احتمال وجود ریست در آن وجود دارد.

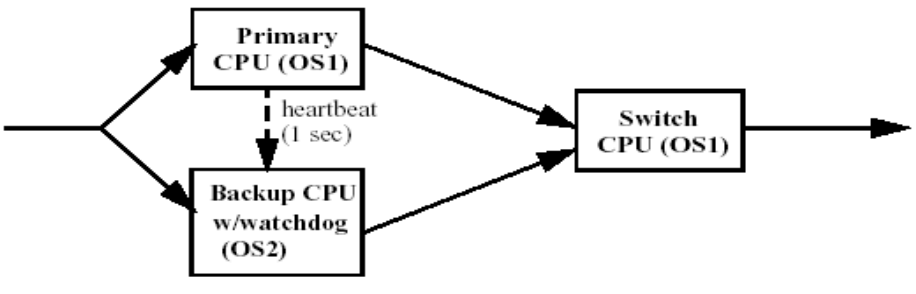
Scenario: S12 (Detect and recover from HW failure of main switch.)
Attribute: Availability
Environment: normal operations
Stimulus: CPU failure
Response: 0.999999 availability of switch

Architectural decisions	**Risk**	**Sensitivity**	**Tradeoff**
Backup CPU(s)	R8	S2	
No backup Data Channel	R9	S3	T3
Watchdog		S4	
Heartbeat		S5	
Failover routing		S6	

Reasoning:

- ensures no common mode failure by using different hardware and operating system (see Risk 8)
- worst-case rollover is accomplished in 4 seconds as computing state takes …
- guaranteed to detect failure with 2 seconds based on rates of heartbeat and watchdog …
- watchdog is simple and proven reliable
- availability requirement might be at risk due to lack of backup data channel … (see Risk 9)

Architecture diagram:



اسلاید بالا: در واقع خروجی اصلی است که ATAM به ما میدهد.توی پروژه های اصلیش. این تمام اطلاعاتی است که در مورد یک سنایو مطرح می شود. این یک کاتالوگی است اطلاعاتی را که تک تک سناریوهایی را که در atam تحویل می شوند رو دارن.بالا یه شماره ای داده مثلا بالای اسلاید یه شماره زده که این شماره سناریو است گفته سناریوی s12 این سناریوش به این ترتیب بوده شناسایی کردن و ترمیم کردن شکست سخت افزاریه سوییچ اصلی در این سناریو اومدن براش اینجور اطلاعات نوشتن. یه صفت availibility هست هر وقت بحث failureهست بحث availibility مطرح است.

سناریوهایی که ما در فصل چهارم مطرح کردیم شش قسمتی بودند ولی سه قسمت اصلی داره از این به بعد در روش atam از سه قسمت اصلی بحث می شود یکیش محیطه که گفته محیط عملیاتی نرمال باشه که اینجا هست – تحریک برای این مثالی که زده یک failure که در cpu ممکن است پیش بیاید.و respons ما این است که ما میخواهیم در کمتر از یک ثانیه ما می خواهیم available بشه سویچمون و در دسترس باشد. حالا برای این سناریو اومده تصمیمات معماری را در ذکر کرده تصمیمات معماری را در قالب ۵ تا تاکتیک گفته چون مجموعه ای از تاکتیک ها در اینجا نیاز است. چون هم شناسایی را می خواهد و هم ترمیم رو. مثلا برای شناسایی اومده از watchdag و heartbeat استفاده کرده است و برای اینکه بخواهد recovery کند اومده از backup استفاده کرده است. و از back up کانال استفاده نکرده است و فقط cpu را back up می کند.از failover استفاده کرده که یه نوع سویچ اتوماتیکه که تعیین می کند که بین چند سویچ کدوم را انتخاب کند. به ازای هر کدوم از تصمیمات معماری یا سناریوها اومده معرفی کرده که هر کدوم از اینها چه ریسکی دارند مثلا r8,r9 را گفته در کتاب برای watchdag , heartbeat, failover routing اومده non -risk ها رو مشخص کرده. نقاط حساسش sensitivity رو مشخص کرده البته برای هر کدوم شناسه براش گذاشته که احتمالا بعدا براش توضیح بیاره که s2,s3,s4,... اینها چی هستند این مثال خاصی است که ما فقط داریم راجع به این صحبت atam چجوری این رو کاتالوک می کنه و مثلا tradoff را مشخص کرده. no backup data channel یک tradoff ی داره که یکt3 tradeodd داره. با یه دونه صفت کیفی دیگه غیر از availability. بعد یک سری reasoning یا دلایل آورده که چرا این تصمیات معماری رو داشته و بعد یک دیاگرام آورده یک دیاگرام معماری است که این مجموعه تاکتیکها را در قالب این دیاگرام نشان داده. مثلا cpu اصلی روی os1 سواره cpu backup یه watchdag داره روی os2 سواره . cpu اصلی یک heartbeat ای را هر یک ثانیه می فرسته به cpu backup .اینجوری هم به بک آپ اعلام می کنه من سالمم و هم آخرین وضعیتاشو می فرسته. به محض اینکه cpu back up فهمیدکه heartbeatنمیاد بعد یک ثانیه می فهمه که cpu اصلی failure کرده توسط سویچه روش سویچ می شه. در اینجا سوال پیش میاد که اگر سویچه براش مشکلی پیش بیاد چه اتفاقی میافته به هر حال این بک آپ وجود داره.

فصل ۱۱ قسمت ٤ شکل ۲-۱۱ مثال برای اسلاید می باشد.

۲) خروجی ثانویه: سناریوهایی که به معماری اضافه تر شده اند و شرح کاملتر ونمایش مجددی از معماری است.

۳) خروجی غیر ملموس: طولانی تر از خروجی اصلی بوده و نتیجه مستقیم کارهای ما نیست نتیجه‌ی غیرمستقیم ارزیابی است، حس مشارکتی بین stackholder ها ایجاد می شه کانالهای معماری بین معمار و stackholder ونقاط قوت و ضعف مشارکت و... را بیان می کند.

فازهای ATAM: ٤ فاز است که در قالب ۹ گام اجرا می شود.

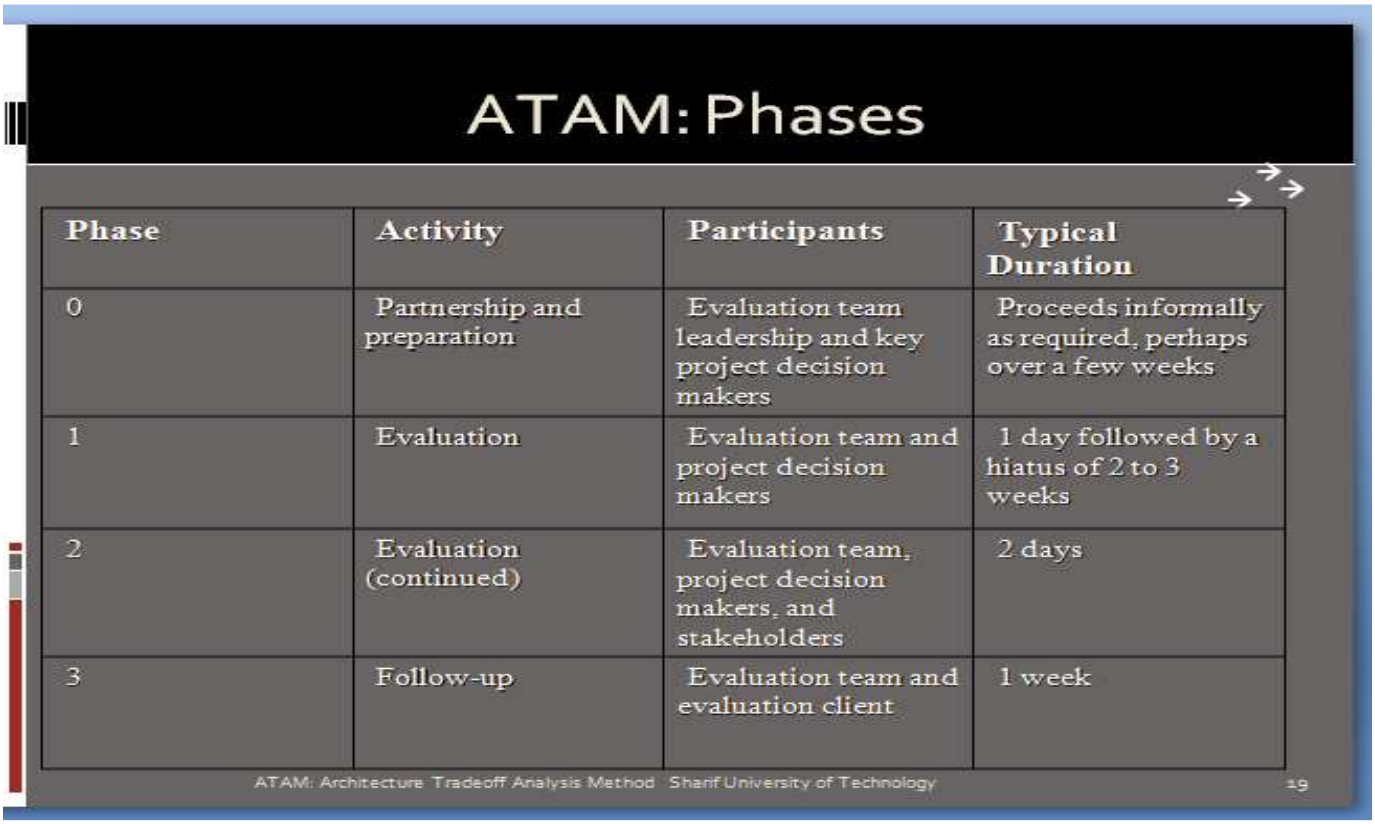
Phase 0 : فاز مشارکت و اماده سازی؛ رئیس تیم ارزیابی و تصمیم گیرندگان اصلی پروژه به صورت غیر رسمی با هم ملاقات می کنند و روی اصولی که بر اساس ان صحبت و ارزیابی خواهند کرد توافق می کنندو تصمیم برای لیست اسامی stakeholder اصلی – زمانبندی روی زمان تهیه گزارش نهایی و آماده شدن برای فاز ۱ میباشد. پس این یک آماده سازی اولیه است.

Phase 1 : ارزیابی معماری و تحلیل آن است. کسایی که تحلیل را انجام می دهند تیم ارزیابی و تصمیم گیرندگان تحلیل خواهند کرد. جزئیات رو بعدا میگیم.

Phase 2: ادامه ارزیابی است و تفاوت آن با فاز۱ در این است که stakeholderها وارد کار می شوند.

Phase 3: فاز پیگیری ؛ گزارش نهایی آماده و یک جلسه پس از واقعه تشکیل می شود که در آن درباره اینکه چه چیزهایی خوب پیشرفته و چه چیزهایی نه بحث می کنند. گزارش ما شامل خروجی های غیرملموس و اصلی و ثانوی است که در گزارش خروجی اصلی و ثانویه نوشته میشود.

(**جدول Atam phasesخوانده شود به جای توضیحات فازها**)



توبحث فازهای atam بودیم که در اینجا این چهار تا فازش رو گفته مشارکت کننده های توشو گفته و مدت زمان

- ❖ زمانبندی فاز ۰ آماده سازی به صورت غیر رسمی است پس یعنی زمان خاصی ندارد ولی معمولا زمانش بیش از چند هفته طول میکشد
- ❖ فاز ۱ : که فاز ارزیابی شروع می شود میشه اونجایی که تیم ارزیابی و تصمیم گیرندگان پروژه هستند از یک روز تا سه هفته طول می کشد
- ❖ فاز ۲:در این فاز stackholder ها که وارد می شود فقط دو روز طول می کشد که یکسری بحثها و نتایج به دست آوردند که اونها را با در stackholder ها در میان میزارند ونظر می دهند که تعداد اونها هم زیاده و بیش از ۱۲ نفرند.
- ❖ فاز ۳ : یک هفته طول می کشد گزارش نویسی است.

* ارزیابی در ۹ مرحله انجام می شود و مربوط به فازهای ۱ و ۲ می باشد.

مرحله ۱ : توضیح درباره ATAM ، چه چیزی دنبال می کندهدف ها و گام ها و خروجی ها.

مرحله ۲ : درایور کسب و کار مشخص می شود (ما نیاز داریم محتوای سیستم را درک کنیم باید درایورهای کسب و کار اصلی شناسایی شود. از دید و جنبه کسب و کار و بازار نگاه کنیم به سیستم؛ یعنی کاربردها و فعالیت های اصلی که سیستم انجام می دهد کدام است، محدودیت های سیاسی و اقتصادی و اجتماعی و مدیریت و تکنیکی... مربوط به پروژه چیست،اهداف کسب و کار و ارتباطشون با پروژه چی هست؟ stakeholder و راه اندارهای معماری چه کسانی هستند و...).

درایورهای معماری یا راه اندازهای معماری : دسته ای از صفات کیفی هستند که درقالب سناریو مطرح می شوند اهداف کسب وکارو functionality های معماری

* **سوال امتحانی**: منظور از business driver چیست؟ جواب مرحله ۲ است.

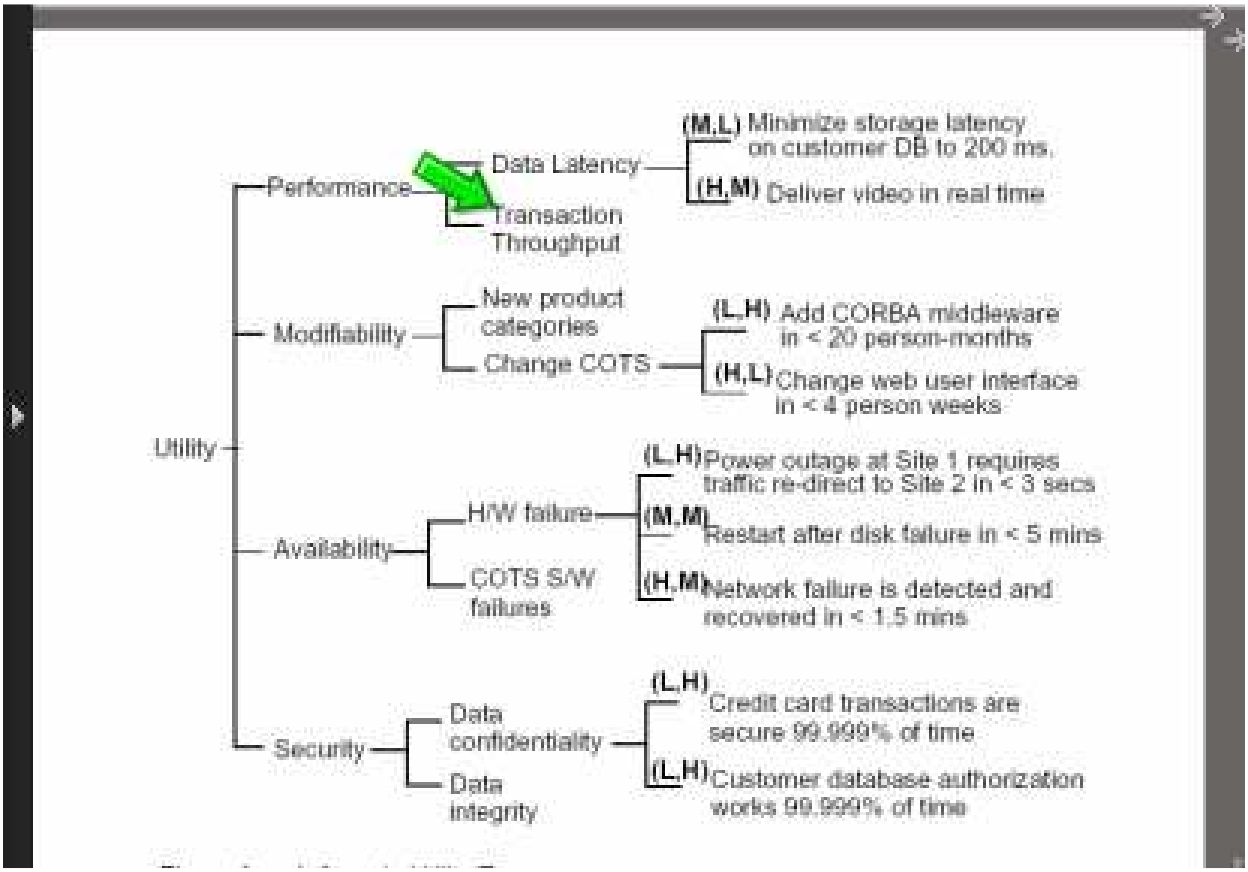
مرحله ۳ : خود معماری معرفی می شود ، ارزیابی در طول ۲ تا ۳ هفته است. معماری در سطح مناسبی از جزئیات که به درد stakeholderها می خورد معرفی می شود ، محدودیت های تکنیکی (محدودیت درباره سیستم عامل- سخت افزار و میان افزار و ...) بررسی شده و در باره شیوه های معماری(تصمیم معماری هم بهش می گن) (مجموعه ای از تاکتیک ها که به ما کمک می کند نیازهایمان را رفع کنیم) بحث می گردد.

* بزرگترین چالش در بحث ارزیابی زمان است.

مرحله ٤ : با ریز جزئیات بیشتر استراتژی معماری مشخص می شود، تمرکز روی تحلیل معماری با درک شیوه های معماری است، pattern ها وروش هایی که مشهود هستند ثبت می شود و این مرحله به عنوان اساس تحلیل مرحله بعد است.

مرحله ٥ :درخت سودمندی صفات کیفی تولید میشود.

درخت سودمندی : ما می اییم مهمترین صفات کیفی سیستم را **تجزیه** و **الویت بندی** و **شناسایی** می کنیم. این درخت می تواند برای ما پوشش دهد که ما تا چه حد صفات کیفی را دیده ایم یعنی سلسله مراتبی از صفات کیفی که ملاقات کرده ایم و تصمیم معماری که برآن گرفته ایم را بکشیم و لیست کنیم(شکل و توضیحات در اسلاید ۲۵ و ۲۶ و ۲۷)



اول می آییم یه ریشه در نظر می گیریم و اسم ریشه سودمندی یا utility می گذاریم و بعد مهمترین صفات کیفی را می گذاریم فرزندان سطح اول در این مثال کارآیی، قابلیت تغییر، قابلیت دسترسی و امنیت به عنوان فرزندان در نظر گرفته شده هر کدام از این فرزندان سطح اول را تجزیه می کنیم به سطح دوم. مثلا performance رو باز کردیم به درنگ داده و بازدهی throughput تراکنشها.

modifiability تجزیه می شود به طبقه بندی محصول جدید و دومی تغییرات cots.

سطح ۳ ← سطح ۲

سطح ۱ (مهمترین صفات کیفی) ← سطح ۲

سودمندی(ریشه)

COTS: از گزینه ای روی میز که موجود است استفاده کنیم برای تولید محصول نرم افزاری (چیزی شبیه مونتاژ).

هدف COTS : برای اماده کردن و تحویل سریع تر محصول به بازار استفاده می شود. اگر فشاری هست برای تحویل سریعتر محصول از cots استفاده می شود.

availability: تجزیه میکنیم از دو جنبه به شکست سخت افزار و شکست درخریدن ونوشتن cots ها.

security تجزیه میشود به قابلیت اطمینان برای داده و یکپارچگی داده بحث کرده .

در سطح ۳ : سناریو ها مربوط به کدام قسمت از صفات کیفی ما میشود.

مثلا در مقوله کارآیی زمینه ی درنگ داده ، درنگ ذخیره سازی برای مشتری را به ۲۰۰ میلی ثانیه برسانم. یه مورد دیگه این است که می خوام دلیور ویدئوها رو بلادرنگ بکنم. پس در آخر سطح میاد سناریوهای خاص را می نویسد.

پس میاد میگه این سناریوهای خاص که مدنظر ما هست و ما میخواهیم راجع بهش تحلیل و ارزیابی انجام میدهیم مربوط به کدوم یک از تاکتیکهاست و کدوم دسته خاص از صفات کیفی به صفات کیفی مهم ما است. بعد اینجوری لیستشون میکنه و در آخر سر یه کار دیگه مونده تا درخت تمام بشه.

در آخر رسم درخت: برچسب اولویت می دهیم به سطح ۳(اولین حرف نشان میدهد که این سناریو چه قدر برای پیاده سازی مهم است و دومین حرف؛ چه قدر پیاده سازی سخت است).

L:Low کم H:Hi بالا M:Medium متوسط

جفتی بیان می کنه اینجوری اضافه کردن یک میان افزار corba که برای اینکه ما بتونیم قابلیت تغییر در cots را اگر اضافه بکنه این کار ۲۰۰ نفر ماه طول می کشه. اینجا مشخص کرده این سناریو سطح اهمیتش کمه برای پیاده سازی ولی پیاده سازیش سخته. یا مثلا واسه تغییر واسط گرافیک تحت وب کاربرد که میشه کمتر از ٤ هفته نفر طول می کشه سطح اهمیتش برای پیاده سازی کمه ولی پیاده سازیش راحته .پس اینطوری ما می تونیم اولویت بندی کنیم سناریو را براساس میزان اهمیت و سختی پیاده سازی. بعد از ساختن درخت سودمندی وارد گام ٦ می شویم.

مرحله ٦ : درخت سودمندی در واقع میاد approach های معماری را به ما میدهد. حالا می آییم از بالاترین رنکش تا پایین دونه دونه بحث می کنیم و نظر می دهیم که آیا این برچسبهایی که زدیم خوبه یا نه؟شیوه های معماری تجزیه شده و نقاط حساس و ریسک و غیر ریسک و نقطه Tradeoff شناسایی می کنیم. پس به شکل اسلاید ۲۹ می رسیم یعنی تو گام آخرما این شکل را آماده کردیم و درخت سودمندی داریم.

مرحله ۷ : اولین گام از فاز ۲ است که در فاز ۲، stakeholderها وارد میشوند. همان سناریویی که در مرحله ٦ آماده شده در برابر یک توفان مغزی (جرقه فکری خوب(منظور از توفان مغزی: چیزای خوب یهویی مثلا یک شخصی تویک جمع در مورد یک موضوع یک ایده ی جدید بده و مسیر ذهنی را عوض کند)) برای اولویت بندی در برابر stakeholder قرار می دهیم.

ممکن است بر اساس توفان مغزی سناریوها اولویت بندی عوض بشه یا شیوه معماری تغییرکند .

مرحله ۸ : عین گام ٦ سناریوها اولویت بندی و تحلیل شده اند.همراه با نقاط tradeoff و حساس و ریسک و غیر ریسک نوشته می شود.

مرحله ۹ : ارائه نتایج، تمام چیزایی که روش بحث کردیم را لیست می کنیم .روش معماری مستند شده – سناریوها اولویت بندی شده – ریسک ها و غیر ریسک ها و نقاط حساس و Tradeoff را ارائه و لیست می کنیم.

* **به ازای هر زمینه ریسک راه اندازها را به مرحله ۲ وصل کنیم.**
* **جدول در اسلاید ٣٤ گام ها را نشان داده (۲ یا ۳ ستاره: خروجی خیلی مهم است. ۱ ستاره : خروجی اهمیت کمتری دارد.)**
* **اسلاید ۳۵ خیلی مهم است که خلاصه ATAM می باشد.**

ATAM Outputs

Steps	Prioritized Statement of Quality Attribute Requirements	Catalog of Architectural Approaches Used	Approach- and Quality Attribute-Specific Analysis Questions	Mapping of Architectural Approaches to Quality Attributes	Risks and Non-risks	Sensitivity and Tradeoff Points
1. Present ATAM						
2. Present business drivers	*				*	
3. Present architecture		**			*	*
4. Identify architectural approaches		**	**		*	*
5. Generate quality attribute utility tree	**					
6. Analyze architectural approaches		*	**	**	**	**
7. Brainstorm and prioritize scenarios	**					
8. Analyze architectural approaches		*	**	**	**	**
9. Present results						

ATAM: Architecture Tradeoff Analysis Method Sharif University of Technology 34

■ تعریفATAM در کل یک روش محکم و استواری برای ارزیابی معماری نرم افزار مبتنی بر هایلایت کردن تصمیم های معماری است که منجر به تحقق صفات کیفی میشود.**(سوال امتحانی)**

ATAM ارزیابی کد نیست و نیاز به پیاده سازی و کد نویسی ندارد. تست خود سیستم واقعی هم نیست. یک ابزار دقیق هم نیست(چون یک روش کیفی است). شیوه ارزیابی برای نیازمندی ها نیست(چون نیازمندی ها تثبیت شده و ارزیابی شده به عنوان ورودی به آن داده می شود).

CBAM : (ابزار دقیق)ارزیابی کمی که خروجی ATAM می شود ورودی آن .تک تک سناریوها و هزینه ها و سودهای آن را حساب می کند. یک روش دقیق است.

ATAM: A Case Study است و در امتحان سوال نمیاد. شما باید بدونیم atam چیه اهدافش چیه و چند تا اصطلاح از آن را بدانید.کلمات کلیدیشو بدیم خروجیهای atam منظور از درخت سودمندی چی هست؟ منظور از business driver چیست؟منظور از خروجی غیرملموس؟ تولید کننده نرم افزار سیستم سلامت که در بیمارستانها استفاده می شود که یک چیز بزرگ با چندین میلیون کد است.

فازها:

فاز ۰ : تیم ارزیابی انتخاب – تشکیل جلسه معارفه – سیستم Nightringingمعرفی می شود.

فاز۱ : گام ها شروع شده اند.

در بیمارستانها Business Drivers : کارایی (در حداقل زمان ممکن پاسخگو باشد)– قابلیت استفاده (به راحتی با سیستم کار کرد و حداقل خطای کاربر در آن باشد) و پشتیبانی(از ورود بیمار به بیمارستان تا خروجش او را پشتیبانی میکند) اهمیت دارد .

امنیت : در دسترس بودن – اهمیت کمتری دارند- فعالیت پشتیبانی

فاز۳: معرفی معماری- قابلیت پیکربندی- لایه بندی و مشخص کردن زیر سیستم های اصلی و ...

گام ٤ : شیوه های معماری {۱- لایه بندی کردن ۲- استفاده از تاکتیک Object – orientation ۳- پردازش client-server

٤- استفاده از تاکتیک فایل های پیکربندی برای اعمال قابلیت تغییر ٥- الگوهای معماری Data centric استفاده شده

٦- differ wilding time به تعویق انداختن زمان (؟) که امکان اعمال تغییر برای کاربر خانگی و مدیر سیستم در زمان اجرا می دهد}.

مرحله ٥: درخت سودمندی

مرحله ٦ : تحلیل درخت سودمندی (در درخت سودمندی هر سناریو که برچسب H,H خورده در نظر نمی گیریم و از H,M شروع شده) واثر تغییرات روی معماری بحث میشود.

مرحله ۷ :

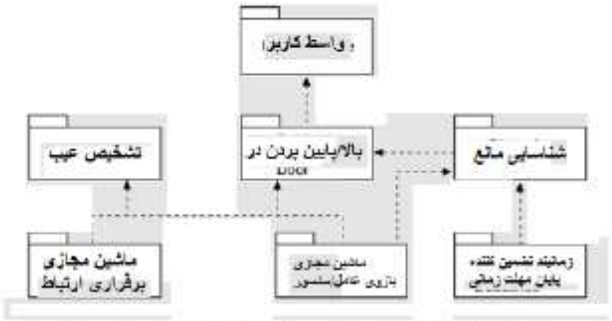
مرحله ۸ : همان مرحله ٦ اجرا می شود

مرحله ۹ : یادداشت برداری نهایی

سوالات پایان ترم مهندسی نرم افزار پیشرفته　　مدت: ۱۲۰ دقیقه

۱. مفاهیم پترن معماری ، مدل مرجع ، معماری مرجع و معماری نرم افزار را تعریف کرده ارتباط بین آنها را بیان کنید.(۲.۵نمره)

۲. شکل زیر دید تجزیه ماژول(Decomposition) از معماری یک سیستم باز کن اتوماتیک درب گاراژ است.به نظر شما مهمترین صفات کیفی (Quality attributes) سیستمی برای این سیستم کدام هستند؟سه تا از آنها را نام برده برای هر کدام از این صفات یک سناریوی خاص بنویسید و سپس برای هر پیاده سازی هر سناریو یک تاکتیک پیشنهاد دهید. (۴نمره)



۳. روشهای ارزیابی معماری نرم افزار ATAM و CBAM را در قالب جدول زیر با یکدیگر مقایسه کنید.(۲.۵نمره)

	ATAM	CBAM
ورودی		
خروجی		
کاربرد		
شرکت کنندگان		
مزایا		
معایب		

٤. مفاهیم زیر را به صورت کاملاخلاصه توضیح دهید:(۶نمره)

الف)ADD(Attributed Driven Design) طراحی صفت گرا　　ب)Conceptual Integrity (یکپارچگی مفهومی)

ج)Architectural Driver(راه انداز معماری)　　د)COTS (Commercial off the shelf)

ه) ABC(Architecture Business Cycle)چرخه کسب و کار معماری

و)Component and connector Structure(ساختار اتصال مولفه)

برای دیدن نمرات و اطلاع از تاریخ ارائه تحقیقات به سایت iranlearner.ir مراجعه کنید.

موفق باشید

۱۱ نمره سوالات حفظی هست چهار نمره تحلیلی.

یادرآوریها تو امتحان نیست. که متدولوژی ها هست در امتحان نمیاد. اسلاید مقدمه عمارت منچستر و نقش معماری در امتحان نمیاد. امتحان از فصلهای خود کتاب است. تعریف ها همش رو باید بلد باشید.

سوال ۱:جواب در فصل دوم هست

سوال ٤: این سوالها حتما میاد شاید همین یه سوال بیاد با یه تحلیلی. عین تعرفا را می خوام چون مهندسی نرم افزار بازی با کلمات است. فصل پترنها فصل مهمی است.فصل ۱,۲,۷,۵,۱۱,۱۲ حفظی

atam , cbam-حتما سوال میاد.

سوال۲: یک معماری بهتون می دهم.از معماریهای داخل کتاب. مهمترین صفات کیفی را شناسایی کنید یا بگم راه اندازهای کسب و کار چیه.با راه اندازهای معماری چیست؟

ما ٦ صفت کیفی اصلی داریم در اینجا کارایی.قابلیت در دسترس بودن availability,performance کدوم اولین اولویت است.در بازبشه مهمتره اولین اولویت availabilityاست بعدی قابلیت تغییرmodifiability

شما میتونید از فصل چهار سناریو بنویسید برای هر صفت دو تا سناریو حفظ کنید.

مثلا یه سناریو برای availabity: با اعمال درخواست بازشدن در از طریق کنترل از راه دور در حالی که سیستم در حال نرمال است درب گاراژ در کمتر از ۱ ثانیه باز شود

تاکتیک: مثلا یه heartbeat بفرسته برای مدیر ساختمان

مثلا یه سناریو برای performance: در اثر رویداد تشخیص مانع ، سنسورها درکمتر از فاصله ۵ سانتی متری مانع را شناسایی کرده و درب را متوقف کند.

برای تاکتیک هم از فصل ۵ بیارید. امکان داره یه سوال بگم که از پترنهایی که میشناسید یک پترن برای این معماری پیشنهاد بدید.یا بگم از روی معماری می توانید تشخیص بدید این معماری چه نوع پترنی دارد؟

فصل ٤و٥ تحلیلی است. فصل ۱و۲و۷ خیلی با ریز جزئیات بخونید.

یادآوری ATAM

در آخرین اسلاید جلسه قبل اشاره شد که ATAM یک صنعت کیفی است و در نهایت گفته شد که یک ابزار دقیقی نبود . حالا به همین خاطر ارزیابی بعدی معرفی شد که روی ATAM اجرا می شود یعنی اینکه ابتدا ATAM اجرا شده که همان سناریوهای مرتب شده هستند به عنوان ورودی به CBAM وارد می شوند . CBAM (متد تحلیلی سود و هزینه) که همانطور که از روی اسمش مشخص است می آید یک عددی به ما می دهد دقیقاً به ازای هر سناریو که در ATAM مشخص کردیم CBAM می آید ROI حساب می کند (بازگشت سرمایه) مطالب مطرح شده در اسلایدها دوباره تکرار می شود CBAM برای چه باید استفاده بشود .

از ایده های موجود در CBAM صحبت می کند . CBAM را مانند ATAM در ۹ گام معرفی می کند .

CBAM

- بزرگترین trade offs که در سیستم های بزرگ و پیچیده وجود دارد معمولا درگیری در مورد اقتصاد (economic) است . (که ATAM در این مورد صحبتی نمی کند .)
- حال منظورش از economic چیه ؟ این است که چه جوری یک سازمان باید روی منابعش سرمایه گذاری بکند تا ماکزیمم عایدی رو داشته باشد و مینیمم ریسک را .
- این اقتصاد شامل هزینه ساخت می شود که البته سودی هم که یک معماری تحویل می دهد را شامل می شود .

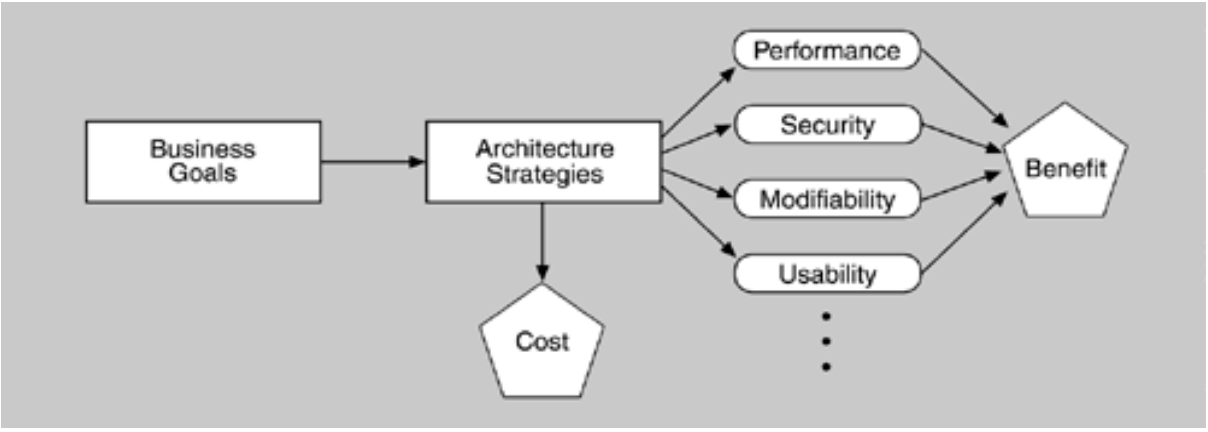
CBAM یک متنی است برای تصمیم گیری

نحوه تصمیم گیری – بدین ترتیب کارش را شروع می کند .

- با نتیجه ATAM کارش را شروع می کند .
- هزینه و سود مرتبط با تصمیمات معماری (یا همان تاکتیک یا استراتژی معماری خود استراتژی می شود چند تاکتیک) را مشخص می کند .
- حال گروه ذینفعان با این کارش می تواند تصمیم بگیرند .
  - به عنوان مثال سخت افزار اضافی استفاده کنند یا از روش Fail over استفاده کنند Load balance (اینها تاکتیک های مختلفی است که در بحث availability داشتیم) .
  - یا اینکه اصلاً بیاییم این هزینه رو صرفه جویی کنیم و سرمایه گذاری کنند روی یک پروژه دیگر

پس در کل CBAM

- یک چارچوب برای تصمیم گیری فراهم می کند .
- به ما کمک می کند که Rol یا نرخ سود هزینه را محاسبه کنیم .



اسلاید بعدی . نمایش Cantent تصمیم گیری
براساس اهداف کسب و کار استراتژی های معمولی معماری را تعیین می کنیم .
هر یک از استراتژی مجموعه ای از تاکتیک ها می باشد .
پس براساس استراتژی تعریف شده هزینه اش محاسبه می شود و این استراتژی که می بینید چند تا صفت کیفی رو به ازای هر کدام از تاکتیک هاش تحت تاثیر قرار میگذارد.

_ سناریوهاش است که ما می خواهیم برای این استراتژی به آن برسیم سودش را محاسبه می کند . پس سود را تقسیم بر هزینه می کند و Rol به دست می آید پس همه این استراتژی ها را براساس سود و هزینه مرتب می کند حالا ذینفع ها می توانند تصمیم بگیرند که کدام استراتژی را پیش بگیرند و کدامیک نسبت به سود و هزینه ی بالاتری برایشان دارد و آن ر انتخاب بکنند .
یادآوری ATAM هنگامی که ATAM به سیستم نرم افزاری ما اعمال می شود ما به عنوان نتیجه اش یا خروجی آن این چیزها را به دست آوریم :
توصیفی است از اهداف کسب و کار که در واقع بحرانی هستند برای موفقیت سیستم مجموعه ای از دید های سیستم را داریم که مستند هستند برای معماری موجود
یک درخت سودمندی داریم که یک تجزیه ای از اهداف ذینفع ها را برای معماری نشان می دهد که در سطح پایین با صفت کیفی شروع می شود و به سناریو ها ختم می شود.
مجموعه ای از ریسک ها را داریم که شناسایی شده .
مجموعه ای از نقاط حساس را داریم (تعریف : تصمیمات معماری که اثر می گذارند روی برخی از نگرانی های صنعت کیفی
مجموعه ای از نقاط trade off را داریم (تعریف : تصمیمات معماری که اثر می گذارند روی بیش از یک مقدار صفت کیفی به صورت مثبت یا منفی)
پس ATAM شناسایی می کند مجموعه ای از تصمیمات اصلی معماری را که مرتبط هستند با سناریوهای کیفی که از ذینفع ها استخراج شده اند .
نتیجه این پروسه در واقع برخی از صفات کیفی خاص هستند .
ما به ازای هر سناریو که راجع به آن صحبت کردیم گفتیم response وجود دارد . جال می خواهیم Response را چند سطحش بکنیم یعنی همیشه یک مدل پاسخ برای یک سناریو مناسب نیست پاسخ هایی که ما برای یک سناریو در نظر می گیریم ارزش های مختلفی برای ما دارد کلاً پاسخ به یک سناریو به این چند روش است . بهترین سناریو، بدترین ، متوسط و پاسخ مطلوب ما همیشه به بهترین پاسخ نمی توانیم برسیم اما اهداف مان این است به پاسخ مطلوب دست پیدا کنیم از طرفی هم گاهی اوقات سطح پایین پاسختان را هر چه بالا ببرید ارزش آن زیاد نمی شود . (در این دو اسلاید راجع به این موضوع صحبت می شود) .
مثال : فرض بکنید availability را اینطور پیاده سازی می کنید که یک ثانیه شود که اگر حال بشود یک صدم ثانیه ارزش دارد ؟ اگر Failer روی یک کامپوننت رخ داد ظرف مثلاً یک ثانیه برگردد سر کارش به حالت اولیه که AV بشود حال اگر بشود یک صدم ثانیه ارزش دارد ؟
مثلاً در یک سیستم بانکی هستند ولی می خواهید پاسخی را که از صفت AV پشتیبانی می‌کند دریافت بکنید که پاسخ ها اگر مشکلی برایش ایجاد شود رفع یک میلی ثانیه مشکلش را حل می کند حال اگر بشود یک صدم میلی ثانیه ارزش دارد .)
پس گفته ATAM تصمیم گیری برای تاکتیک های معماری را پوشش نمی دهد فقط آنها صفات کیفی به اهداف کسب و کار لینک می کند و با این روش در واقع سود و هزینه را حساب نمی کند .
از این رو CBAM می آید سود و هزینه مرتبط با این اطلاعات را استخراج می کند .
با داشتن این اطلاعات ذینفعان می توانند از سخت افزار اضافی استفاده کنند .
checkpoint یا هر تاکتیک مطلوب دیگری را به کار ببرند یا اینکه گفته سرمایه گذاریشان را ذخیره بکنند برای یکسری صفات کیفی دیگر ممکن است عقیده داشته باشیم که کار این بالاتر نرخ سود و هزینه بیشتری برای ما خواهد داشت
CBAM نمی آید برای ذینفعان تصمیم گیری بکند .
فقط مانند مشاور مالی انواع راه ها را پیشنهاد می کند تا شما تصمیم بگیرید پس به طور خلاصه ایده CBAM این است که استراتژی های معماری که (مجموعه ای ازتاکتیک ها معماری هستند) اثر می گذارد روی صفات کیفی سیستم و اینها فراهم می کند برای ذینفعان برخی سودها را که ما به این سود هایی که استراتژی ها فراهم می کنیم که ما به ان سودها utility خواهیم گفت .
میزان منفعت که یک استراتژی برای ذینفعان دارد (utility) هر استراتژی معماری یک سطح سودمندی را برای ذینفعان فراهم می کند که همچنین هر استراتژی زمان و هزینه ای برای پیاده سازی نیاز دارد با این اطلاعات CBAM می تواند کمک بکند به ذینفعان که استراتژی های معماری را براساس (ROI) یا بازگشت سرمایه (نرخ سود و هزینه) حساب بکند .
اساس CBAM :
ما کارمان را با مجموعه ای از سناریوهای که تولید شده به عنوان یک بخشی از ATAM یا به صورت خاص برای CBAM شروع می کنیم.

امتحان و بررسی می کنیم که ارزش های مختلف پاسخ های نمایش داده شده این سناریو ها چقدر است و بعد به آنها به این ارزش ها utility انتصاب می دهیم و بعد utility برای آن ارزش همین ..... منظورش چیست ؟

سودمندی مبتنی بر اهمیت هر سناریویی که در نظر گرفته می شود با توجه به ارزش پاسخ مورد انتظارش

بعد از ارزش گذاری برای پاسخ های سناریوها و سودمندی به آن انتصاب داریم .

در نظر می گیریم استراتژی های معماری را که منجر می شود به تولید پاسخ های مختلف و هر استراتژی هم هزینه ای دارد که می تواند روی چندین صفت کیفی اثر بگذارد .

Utility چگونه حساب می شود ؟

می گوید ما گونه های مختلف از سناریوها را داریم

در CBAM کاری که می کند این است که در واقع سناریوهای concreat را در نظر می گیرد . فقط یک پاسخ بر آن در نظر نمی گیرد بلکه مجموعه ای از پاسخ ها را در نظر می گیرد همانند ATAM

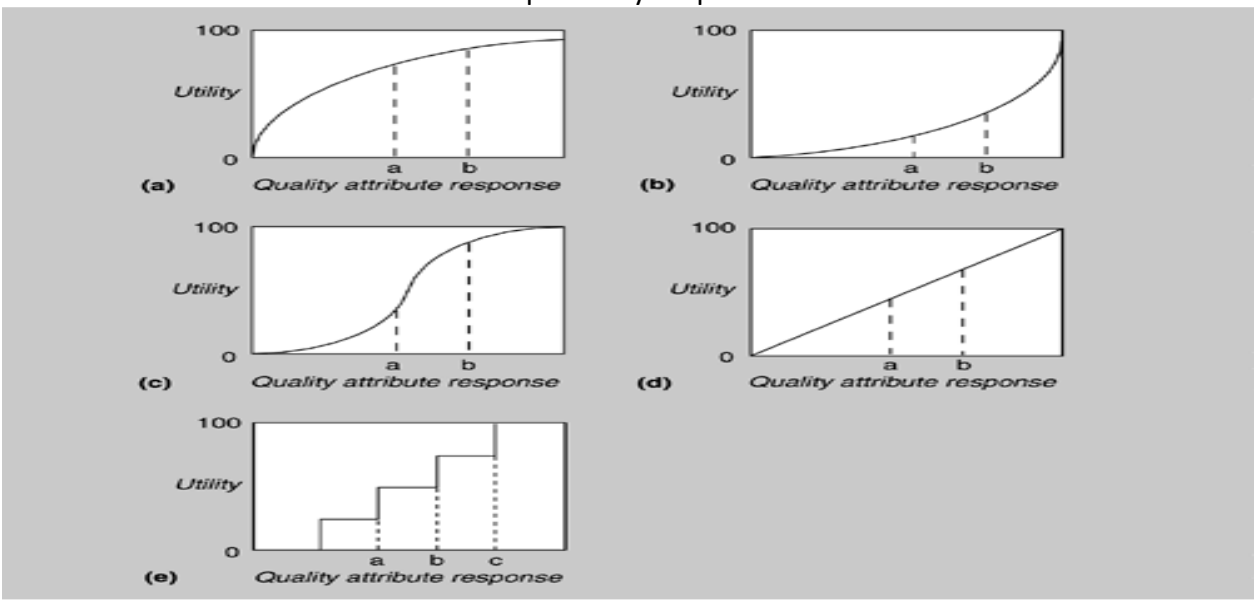
سناریوهایی که در CBAM می آید ۳ قسمت اصلی دارد ۱ – تحریک ۲ – محیط ۳ – پاسخ (سیستم در آن وجود دارد وقتی تحریک وارد می شود)

حالا گفته CBAM مجموعه ای از سناریوها را در نظر می گیرد به جای اینکه مجزا در نظر بگیرد که این امر منجر می شود به تولید مفهومی به نام منحنی پاسخ سودمندی که این منحنی به این ترتیب است .

هر جفت مقدار پاسخ سودمندی پاسخ های متفاوت می توانند با هم مقایسه بشوند . می تواند برای ذینفعان خیلی ارزش داشته باشد و ارزش آن خیلی بیشتر از AV متوسط باشد درنگ کم ارزش آن با درنگ متوسط یکی باشد .

حال درنمودار نشان داده می شود منحنی های سودمندی پاسخ که انواع آن در سیستم های مختلف تحلیل شده و به دست آوردند . این نمودار در مثال CBAM بهتر قابل درک هستند .

Some sample utility-response curves



شکل C : محور عمودی : (سودمندی) محور افقی : (پاسخ) یعنی در این نمودار اگر پاسخ از A به b برود سودمندی با یک شیب کندی افزایش پیدا می کند . پس سناریوها را چند پاسخ در نظر می گیریم و براساس آن منحنی را می کشیم .

ما برای کشیدن این منحنی ها از ۵ نقطع داده ای استفاده می کنیم که ۴ تای آن مستقل از استراتژی معماری است و یکی از آنها وابسته است . این ۴ مقدار مستقل به این ترتیب هستند . ۴ تا :

بهترین مقدار به عنوان مثال ۰,۱ ثانیه RT (زمان پاسخ) برای یک نفر بنابراین اگر زمان پاسخ بشود ۰,۳ ثانیه هیچ ارزشی ندارد و utility این حالت ۱۰۰ =

بدترین مقدار که سودمندی آن صفر است

مقدار جاری یعنی پاسخی که فعلاً وجود دارد یعنی مقدار همان پانچی که سیستم دستی current بین بهترین و بدترین است x ٪ سودمندی برای ما فراهم می کند و مقدار مطلوب بین بدترین و بهترین و y ٪ سودمندی آن است .

پس با این چهار نقطه تولید می کنیم و منحنی ها را برای سناریوها می کشیم و پس تا اینجا منحنی ها تولید می شود برای هر سناریو . سناریوهای مختلف در یک سیستم مشخص سطح های متفاوتی از اهمیت را برای ذینفع ها دارند و از اینرو سودمندی های تفاوتی هم دارند .

حال برای اینکه اهمیت سناریوها را از نظر ذینفعان مشخص بکنیم می آییم به هر کدامشان یک وزنی را اختصاص می دهیم این وزن اهمیت سناریو را مشخص می کند این الگوریتم ۲ گام دارد

در مرحله اول ذینفعان می آیند رأی می دهند به ســـناریوها (اختیاری) این رأی دهی مبتنی بر ارزش پاســـخ مورد انتظار از ســـناریو از دید ذینفعان مربوطه است پس ذینفعان می آیند وزن یک را به سناریوی بالا می دهند (بالاترین رأی را می دهند) و به بقیه کسری از یک را می دهند .
تاثیرات جانبی را نیز در نظر بگیرید .
در هر صورت یک معماری نه تنها روی یک صفات خاص کیفی اثر می گذارد بلکه ممکن است روی سایر صفات کیفی اثر گذار باشد . این مهم است که سودمندی هر پاسخ صفات کیفی که تحت تاثیر صفات کیفی دیگر قرار گرفته اند را مشخص بکنیم و آنها را در استراتژی های معماری در نظر بگیریم و در بدترین حالت هم مجبوریم که ما یک نسـخه جدیدی از سناریو را برای آن صفت تحت تاثیر قرار گرفت در نظر بگیریم و یک منحنی جدید برای آن بکشیم .

خروجی ATAM ← سودمندی ← وزن ← Side effect ها را هم در نظر می گیریم .
تعریف استراتژی : مجموعه ای از تاکتیک ها
چگونه ما از سطح پاسخ کیفی جاری به سطح مطلوب منتقل بشویم .
اسـتراتژی چه کاری باید بکند . پنجمین نقطه داده های که صحبتش را می کردیم مقدار مورد نظر استخراج شده از پاسخ جدیدمان و سودمندی که به آن انتصاب داده می شود از درون یابی چهار نقطه قبل به دست می آید البته با توجه به Side effect ها حال پس ۵ امین نقطه را تعریف می کنیم .
می گوییم این سناریو را داریم اگر این استراتژی را اعمال کنیم این مقدار expected هم اضافه خواهد شد .
۴ نقطه قبلی را خودمان عدد دادیم . ولی روش ۵ امین را از روش درون یابی بین ۴ نقطه به دسـت می آوریم . پس به ازای هر سناریو ۵ نقطه داده ای داریم .
ادامه بحث کار گروهی معماری این است که استراتژی ها را مشخص بکند برای حرکت دادن سطح پاسخ صفت کیفی جاری به مطلوب یا بهترین که ما می توانیم به ازای هر استراتژی مشخص بکنیم . بعد مقدار مورد انتظار هر پاسخ در هر استراتژی یک سودمندی دارد که این را از درون یابی چهار نقطه ای که در حال حاضـر از ذینفعان اسـتخراج کردیم به دسـت می آوریم و در واقع اثر اسـتراتژی های مع روی صفت های کیفی دیگر در نظر گرفته می شود هزینه پیاده سازی هم در نظر گرفته می شود .

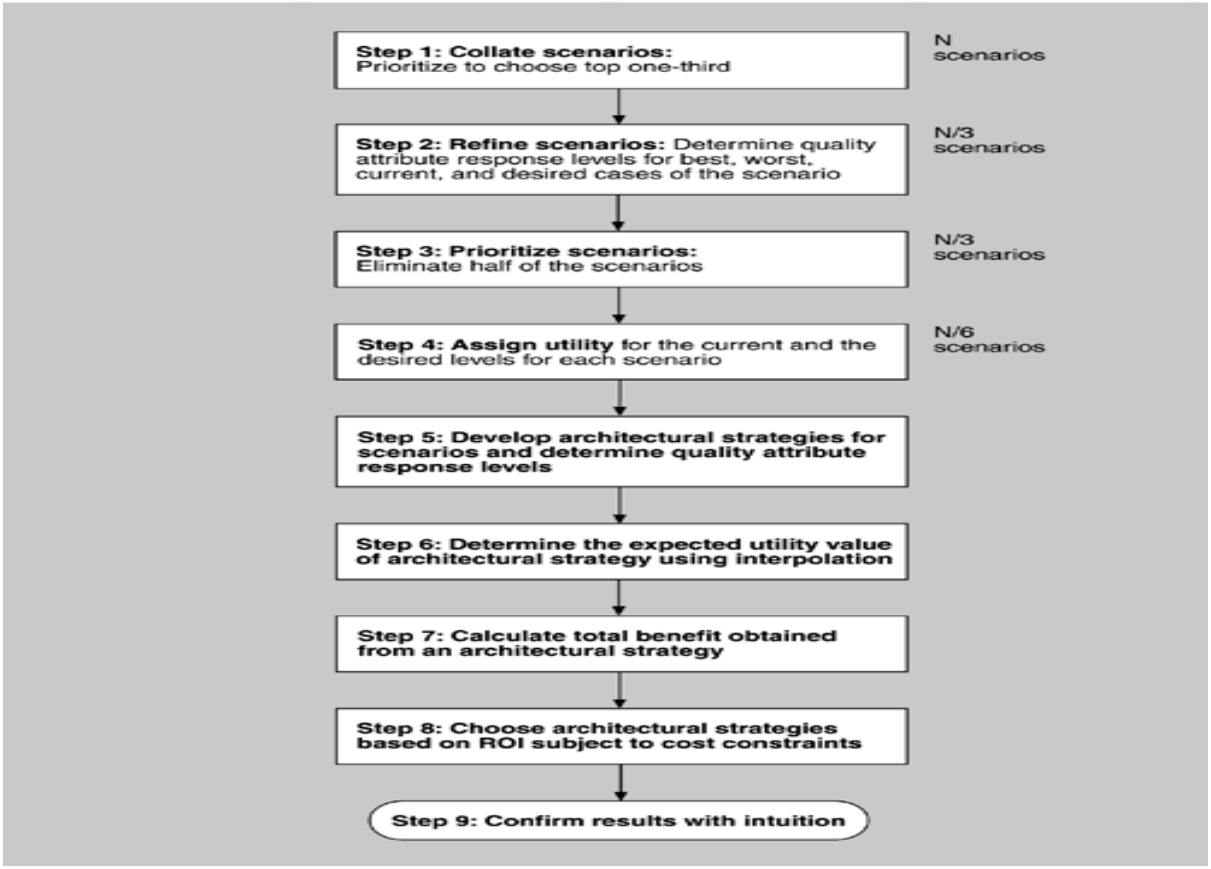
مرحله بعد می آییم سود کلی استراتژی ها را حساب می کنیم با فرمول $B_i = \sum_j (b_{i,j} \times W_j)$

سود استراتژی i ام : Bi

برابر اسـت با $B_i = \sum_j (b_{i,j} \times W_j)$ برابر اسـت با سود حاصله برای سناریوی C ام از استراتژی i ام یا به عبارت دیگر میزان بودن یا اثری که استراتژی روی سناریوی C ام می گذارد × وزن سناریوی C ام .

Bi از میزان سودمندی – میزان مورد انتظار = C و b به دست آمد .

بعد ci که هزینه اش هست را باید حساب کنیم .
Ri که نسبت سود به هزینه است را محاسبه می کنیم .



از اینجا به بعد پیاده سازی ***CBAM*** را دارد که ۹ ***step*** است و روی آن توضیح می دهیم .

گام اول : همه سناریوها را جمع آوری می کنیم و الویت بندی می کنیم و یک سوم از آن سناریوها را بر می داریم (برای مرحله دوم)

گام دوم : سناریوها را پالایش می کنیم به این معنا که سطوح مختلف پاسخ ها را حساب کنیم (بهترین ، بدترین ، جاری ، مطلوب را حساب کنیم .)

گام سوم : نصف سناریوها را حذف می کنیم و الویت بندی می کنیم .

گام چهارم : به هر سناریوی ***utility*** انتصاب می دهیم .

گام پنجم : استراتژی معماری توسعه می دهیم برای سناریوهایمان در واقع سطح پاسخ مورد انتظار برای هر سناریو در اثر اعمال استراتژی ها را تعیین می کنیم .

گام ششم : به پاسخ مورد انتظار به ازای هر سناریو تحت تاثیر استراتژی ها سودمندی اختصاص می دهیم .

گام هفتم : سود کسی را برای هر استراتژی معماری حساب می کنیم با استفاده از فرمول مطرح شده .

گام هشتم : با حساب کردن هزینه به ازای هر استراتژی هزینه / سود و ***Erdi*** را به دست می آوریم .

گام نهم : درا ین گام کافی است سناریو ها را براست ***ROI*** مرتب کنیم و تصمیم بگیریم کدام انتخاب شود حال از این به بعد مثال برای سیستم ناسا : (در کتاب در این فصل یک مثال دارد که جدول دارد و این مثال را ببینید و متوجه ***CBAM*** خواهید شد .)