به نام خدا

پیچیدگی در نرم افزار

بدلیل تفاوت ذاتی بین نرم افزار و سخت افزار پیچیدگی خاصی در ابعاد مختلف از جمله تعریف نرم افزار، طراحی و پیاده‌سازی، تست و نگهداری آن وجود دارد که:

با پیچیدگی سیستم‌های طبیعی و محصولات فیزیکی ساخت است بشر متفاوت است.

یک خاصیت ذاتی سیستمهای نرم افزاری بزرگ

بنابراین نمی‌توان این پیچیدگی را از بین برد بلکه باید آنرا کنترل نمود.

انواع پیچیدگی:

intelleictually intractivility (تمردپذیری و اجازه پذیرفتن برای آشفتگی):

پیچیدگی بطور ذاتی در ساخت سیستم وجود دارد، پیچیدگی ممکن است از بزرگی سیستم ، یا از واسینگیها، بدعت‌ها و پیاده‌سازی تکنولوژی و . . . بوجود آید.

Management intractivility (تمرد پذیری مدیریتی):

پیچیدگی در سازمان و فرآیند بکار گرفته شده در ساخت سیستم، ممکن است از اندازهٔ پروژه (تعداد افردی که در تمام جهات ساخت سیستم درگیر هستند)، وابستگیهای پروژه، فاصله جغرافیایی سیستمها و . . . بعبارتی عوامل تولید کننده نرم افزار غیر قابل کنترل هستند چون سازمان، افراد و فرآیند هستند و ماشین نیستند که کنترل شوند و

سرمایه‌های اولیه برای تولید نرم افزار الزاماً ماشین، سرمایه و پول نیست بلکه یکسری عوامل انسانی متغیری هستند که تحت مدیریت قرار می‌گیرند.

راهکارهای معماری

حق مشکل I : معماری نرم افزری می‌بایست سیستم را قابل هضم و بطور هوشمند قابل مدیریت بوسیله مهیا کردن تجریدی که بدون نیاز به جزئیات، مهیا کننده مفاهیم ساده و یکسان باشند تجزیه سیستم و . . .

حل مشکل IF : معماری نرم افزاری نمی‌بایست توسعه سیستم را آسانتر برای مدیریت بوسیله ارتقای ارتباطات، مهیا کرن بهتر با جدا کردن کار با کاهش زیاد وابستگیهای قابل مدیریت و غیره.

اما مسائل جدید پیدا شده مرتبط با تجزیه سیستم برای حل پیچیدگی بایست توسط معماری بررسی شوند.

چگونه سیستم را به قطعات بشکنیم، یک تجزیه خوب اصل از بین رفتن کوپلاژ بین مؤلفه‌ها (یا قطعات) را بوسیله واسطهای واضح و توانمند، ساده کردن بوسیله تقسیم به قطعات منتقل قابل استدلال که دوباره می‌توانند جدا شوند، ارضا می‌کند.

آیا تمام قطعات مورد نیاز را داریم ساختار می‌بایست وظیفه مندی و یا سرویس‌های مورد نیاز سیستم را پشتیبانی کند بنابراین رفتار دینامیکی سیستم زمان طراحی معماری می‌بایست بحساب آید. همینطور می‌بایست زیربنای ضروری برای پشتیبانی این سرویس‌ها را داشته باشیم.

آیا این قطعات با هم بدرسیت کار می‌کنند؟ این موضوع واسط و رابطه‌های بین قطعات می‌باشد. اما تطابق خوبی که جامعیت سیستم را مدیریت می کند و همچنین با شرایط سیستم کار کند زمانیکه این قطعات ترکیب می‌شود خصوصیات خوب داشته باشند. مورد لزوم است.

شکل زیر وسعت تصمیم و تأثیرات مستقیم را معین می‌کند. بخشی از تصمیمات در حوزه محدود به توسعه‌های محلی (Local) است و اثری روی معماری ندارد و در سطح تک تک مؤلفه‌ها است و از نوع غیر معماری می‌باشد.

بخش دیگر Local نیست ولی تأثیر زیادی ندارد. از خود تقسیم‌بندی سیستماتیک و Local می‌باشد. خود سیستماتیک شامل Highimpaet می‌باشد که ما بدنبال Highimpnet می‌باشیم (اولویت بالا برای ما مهم است).

تأثیر زیاد (اولویت بالا، مهم برای حرفه‌ها تمرکز تصمیمات معماری	غیرمعماری	تأثیر کم سیستماتیک
بطور کلی غیر معماری(ممکن است مجموعه‌ای از سیایت و خطوط راهبردی معماری نیاز باشد)	غیرمعماری	سیستماتیک

و بدلیل اینکه تصمیمات معماری روی جنبه‌های مختلفی از جمله ۱- Sysstempriority (قراردادهای اولویت: مثلاً آیا Perdormance اولویت بیشتری دارد یا Security):

۲- تجزیه و ترکیب سیستم ۳- مسائل مربوط به راههای میامنبر ٤- جامعیت سیم، . . . اثر می‌گذارد، نباید سیستمهای عاری از لایه‌های مختلف تجرید رخ دهد. که متمرکز اصلی بر روی عناصر ساختاری سیستم را خصوصیات قابل روئیت از بیرون و روابط ما بین آنها می‌باشد.

مدل لایه‌بندی و تصمیمات معماری:

به تا سطح تصمیم معماری نرم افزار وجود دارد.

۱- سطح بالاتر از معماری (Meta- Architecture): dictionary معماری می‌باشد مجموعه‌ای از تصمیمات سطح بالا است که ساختاری، تجزیه و مجموعه‌ای از تصمیمات سطح بالا را شامل می‌شود. دورنمای معماری ، اصول- لیک‌ها- مفاهیم کلیدی و مکانیزمها را شامل می‌شود.

بررسی تصمیمات سطح بالا که بطور محکمی ساختار سیستم را تحت تأثیر قرار می‌دهند، قواعد معین می که انتخاب کند و راهنمای کننده انتخاب تصیمات و مصالحه در بین دیگر قواعد می‌باشد، تمرکز دارد.

۲- سطح معماری: ساختار و رفتار، دیده‌های دینامیکل و استارستکی، فرضیات و منطبق را شامل می‌شود.

بر روی تجزیه و انتسایب وظایف، طراحی واسط ، انتساب فرآیندها و نخ‌ها تمرکز دارد. خود شامل سه سطح ۱- معماری ادراکی ۲- معماری منطقی ۳- معماری اجرا می‌باشد.

۲-۱: معماری ادراکی: شامل دیاگرامهای معماری و CRC-R کارنها می‌باشد.

تمرکز بر روی تعیین مؤلفه ها و انتساب وظایف به مؤلفه‌ها دارد.

۲-۲: معماری منطق: شامل را به روز کردن و دیاگرامهای معماری (نشان دادن واسطها)، تعیین واسط، تعیین مؤلفه‌ها و راهنماییهای کاربردی آنها می‌باشد.

تمرکز بر روی طراحی واسطه‌های مؤلفه‌ها ، پروتین‌ها و مکانیزم اتصال و طراحی واسط و تعیین آن مهیا کردن تعریف ضمن از اطلاعات برای کار برای مؤلفه‌ها، دارد.

۲-۳ خطوط راهنمایی و سیاستهای معماری:

شامل کاربرد مدلها و خطوط راهنمای، الگوها طراحی و مکانیزمها؛ چهارچوبهای کاری، استانداردها و ساختارهای زیرین می‌باشد.

بر روی: راهنمای مهندسین در ساخت طراحییهایی که شامل جامعیت معماری می‌باشد تمزکز دارد.

۲-۳ معماری اجرایی:

ایده‌های فرآیند (نشان داده شده د ر دیاگرامهای همکاری) می‌باشد بر روی، انتخاب و آدرس دهی فضاها؛ چگونه آنها با هم تبادل می‌کنند و هماهنگ می‌شوند، چگونه منابع فیزیکی به آنها انتساب داده می‌شوند، تمرکز دارد.

دیدهای معماری: ۱- هر دو دید ساختاری و رفتاری برای تفکر و ارائه معماری مهم می‌باشند:

دید ساختاری: اگر ما بپذیریم که «معماری بالاترین سطح ساختار سیستم شامل مؤلفه‌ها، روابط مابین آنها ، و خصوصیات قابل روئیت از خارج آنها می‌باشد، دید ساختاری محوری است . دید ساختار شامل: دیاگرام معماری(مقوله‌بندی دیاگرام کودس LUML ، و تعیین مؤلفه و واسط آنها می‌باشد.

دید رفتاری: در تجزیه سیستم به مؤلفه‌ها و طراحی و اسطه‌هایشان؛ و طراحی مکانیزمهای برای آدرس دهی به تهدیدهای میانبر مربوطه مساحتی بایست به سؤال: این چگونه کار می‌کند؟ همچنین، در تفهیم و کاربرد معماری، ما می‌بایست قادر به جواب دادن به همان سؤال پاسصخ دهیم. این نقش دید رفتاری، با دیاگرامهای توالی یا همکاری (مقوله‌بندی دیاگرامهای همکاری و توالی در UML) می‌باشد.

دیدهای ساختاری و رفتاری برای هر یک از دیدها (لایه‌های) ادراکی، منطقی و اجرایی معماری همانگونه که در جدول زیر نشان داده شده است قابل کاربرد می‌باشد.

دید ساختاری	دید رفتاری	
دیاگرام معماری مؤلفه‌های غیررسمی (CRC-R)	دنبال کردن همکاری	معماری ادراکی (تجرید)
تعیین واسط دیاگرام معماری با I/FS	دیاگرامهای همکاری	معماری منطقی (با جزئیات بیان شده)
ای فعال	دیاگرامهای همکاری نشان داده شده در فرآیندها	معماری اجرای (دید فرآیند و دید استقرار)
دیاگرام معماری نشان داده شده از مؤلفه		

Archi tecture Views:

در چارچوب کاری تصمیمات معماری

۱- metanrchiteetune

۲- Archilecture

۲-۱ conceputual

۲-۲ Logicalony

۲-۳ Execution Ar

یک مجموعه ای از دیدهای استاندارد ارائه می‌شود. دیدهایی که ما داریم در راهنمایی معمارانی که تصمیمات معماری را می‌سازند که مفید باشد- آمی ابزارهای فکری مفیدی برای در نظر گرفتن تصمیمات و انتخاب بین آستریا ستوهای می‌باشد.

آنها همینطور از طریق اینکه ما مجموعه کاملی از تصمیمات معماری در سطوح انتخاب از تجرید، تعین و اساسی برای تعین معماری می‌باشند. مثلاً دید منطقی، دید ادراکی، دید اجرا، ..

در معماری نرم افزار بسته به خروجهای سطح بالا توجه داریم و اینکه چگونه قبل از Derelope کرده نرم افزار می‌توان آنرا ارزیابی کرد این ارزیابی یک معماری قابل اجرا است. مثلاً prototype مهندس نرم افزار یک نوع معماری قابل اجرا است معماری قابل اجرای سیستم های توزیع شده و همروند ایجاد می‌شوند نگاشت مؤلفه‌های به

فرآیندهایی سیستم فیزیکی با توجه به تمرین بر روی مفاهیمی از قبیل گذردهی و deplogmentriew scalability کد نوع معماری قابل اجرا می‌باشد.

Nrchirecture Business cycle:

تأثیری پذیری معماری از محیط و بالعکس را چرخه معماری کار گویند. شکل زیر این چرخه را نمایش می‌دهد.

۱- معماری بر ساختار سازمان در حال توسعه اثر می‌گذارد. یک معماری یک ساختاری برای یک سیستم تجویزی می‌کند.

۲- معماری می تواند بر اهداف سازمان در حال توسعه تأثیرگذار باشد. یک سیستم موفق ساخته شده از معماری می‌تواند یک شرکت را به مهیا کردن جای پای در نواحی مشخص قادر سازد.

۳- معماری می تواند بر نیازمندیهای مشتریان برای سیستم بعدی از طریق فرصت دادن به مشتریان برای دریافت یک سیستم (بر اساس همان معماری) در اطمینان بشتر، بموقع‌تر و حالت مقدمه به صرفه‌تر از اینکه اگر سیستم بدوی از چرک نویس (سیستم قدیمی دارای اشکال)

٤- فرآیند ساخت سیستم می تواند تجربه معمار را برای با سیستم بعدی از طریق اضافه کردن اساس همکاری تجربه تحت تأثیر قرار دهد.

۵- تعدادی از سیستمهای تأثیر و تغییر واقعی بر فرهنگ مهندس نرم افزاری می‌گذارند، آن فرهنگی که محیط تکنیکی از آهن که سازندگان سیستم عمل کردن و زیاد می‌گیرند.

Desighing the Architecture

برای یک روش طراحی معماری برای برآورده کردن هردو نیازمندیهای کیفی و نیازمندیهای وظیفه مندی طراحی مبتنی بر معماری (ADD) می باشد. ADD یک مجموعه ای از سناریوهای صفات کیفی را بعنوان ورودی گرفته و دانش مربوط به روابط صفات کیفی قابل دستیابی و معماری را بخاطر طراحی معماری بکار می گیرد. روش ADD می تواند بعنوان یک توسعه ای از دیگر روشهای استقرار از قبیل RUP، دیده شود. RUP چندین مرحله دارد که نتیجه در سطح بالای طراحی یک معماری است اما با طراحی همراه با جزئیات و پیاده سازی پردازش می کند. ولی ADD تغییر دهد مراحل RUP را با طراحی سطح بالای معماری تغییر داده و فرآیند Rational را دنبال می کند.

Architecture Description Langnague.(ADL):

ADL نتیجه یک روش زبانی برای ارائه رسمی یک معماریها می باشد، و همچنینبنی نقایص ارائه های رسمی را برطرف می کنند. ADL های پیچیده آنالیز. سریع و آزمایش توانائیهای تصمیمات طراحی معماری را اجازه می دهند.

مثال: Rapiol C2[2] . Darcvin . Wright و ...

مثلاً: Rapid بر روی رخدادهای سیستم، رفتار دینامیکی سیستم بکار برای الگوهای رخدادی تمرکز دارد.

یا Wright بر روی کانکتورها، رفت زیر سیستمهای دینامیکی تمرکز دارد.

Product Lines:

یک مجموعه ای از سیستمها یک مجموعه مدیریتی خواص ساخته شده از یک مجموعه معمول (مشترکی) موجودیهای هسته نرم افزار را به اشتراک می گذارند. این موجودیها (دارئیها) شامل یک خط شالوده معماری و یک مجموعه ای از مولفه های کشترک و شاید قابل اتصال می باشد. حلقه های بازخور چرخه کاری معماری (ABC) که بازخور می شوند تا تاثیرات را بر یک سازمان شامل Produet line منعکس کنند.

خطوط تولید سرمایه گذاری در جهت کاربرد یک معماری (مولفه ها مربوط به معماری) در چندین سیستم می باشد که منجر به خواصی همچون کاهش هزینه ساخت و کمتر کاهش زمان بازاریابی می شود.

یک مدل مرجع یک طبقه بندی از وظیفه مندی به همراه جریان داده میان مولفه ها می باشد به بیان دیگر یک شیوه استاندارد و شناخته شده از مسائلی است که توانسته قطعاتی که حل کننده آن مسئله می باشند را پیدا کرده و بین قطعات تمایز قائل شود.

تعدادی مسئله شناخته شده و تعدادی راه حل مخصوص در مدل مرجع وجود دارد.

مجموعه ای از مدلها که در یک حوزه خاص و شناخته شده وجود دارد، می باشد. مدل مرجع مسئولیتهای اصلی مولفه ها را تشخیص می دهد.

مثلاً طراحی یک DB در مدل مرجع تا این حد می دانیم که هدف چیست و مولفه هایی که باید حضور داشته باشند و ارتباط و وظایف این مولفه ها را می دانیم.

Reference Architecture:

معماری رمجع مبتنی است بر مدلهای مرجع. اگر مدل مرجع را به مولفه های نرم افزاری MAP کنیم بگونه ای که گردش کار بین مولفه ها را بخوبی بتوان نشان داد به آن یک معماری مرجع گویند.

خود معماری مرجع پیاده کننده وظایف موجود در مدل مرجع می باشد. معماری مرجع مول انطباق مسئولیتهای مشخص شده از مدل مرجع به مولفه های نرم افزاری می باشد.

مثلاً: در طراحی DB قبل، مسئولیت انطباق وظایف مشخص شده در مدل مرجع به مولفه های نرم افزاری بر عهده معماری مرجع می باشد.

Migration Plane :

طرحی است که ما را از معماری وضع موجود به معماری وضع مطلوب می رساند.

تعریف دیگر:

معماری قابل اجرا یک پیاده سازی جزئی است که ساخته می شود تا شرح دهد که طراحی معماری قادر خواهد بود وظیفه مندی کلیدی را تضمین کند و، بهتر برای نمایش خواص درست در سترهایی از کارآیی، گذردهی، ظرفیت، قابلیت اطمینان، مقیاس پذیری و غیره می باشد.

Enterprise Architectec tuve Planning (EAP):

برنامه ریزی معماری سازمانی

EAP فرآیند تعریف معماریها برای استفاده از اطلاعات در حمایت از جرفه و طرح پیاده سازی آن معماریهاست. این متدولژی یک رویکرد حاوی چگونگی ایجاد دو سر بالای چارچوب زکمن، برنامه ریز و صاحب است طراحی سیستمها که در سطر سوم آغاز می شود، خارج از حوزهEAP می باشد.

تعریف دیگر

عبارت است از فرآیندی که به منظور تعریف معماریهای لازم و برنامه ریزی و جهت پیاده سازی معماریها انجام شود و هدف از آن فراهم ساختن زمینه های استفاده موثر از اطلاعات جهت پشتیبانی از ماموریتهای سازمان است.

Eind user:

رفتار، کارآیی، امنیت، قابلیت اطمینان، قابلیت کاربرد

رفتار از قبیل سازگار، با بستر، قابلیت کار با دیگر سیستم ها)

Customer (مشتری):

هزینه پایین، خیلی وقتها تغییر نکردن، زمان سریع عرضه به بازار

Marketer(بازاریاب):

ویژگیهای خالص، زمان کوتاه بازاریابی، هزینه پایین، رقابت بیشتر با محصولات هم رده

Maintainerنگهداری کننده نرم افزار:

قابلیت تغییر

:(تولید کننده نرم افزار)Developer

هزینه پایین، نگهداری افراد استخدام شده

:(مدیر تولید نرم افزار) Derelopment Manager

مدیر (به همان اندازه که در مورد هزینه و زمانبندی نگران است) که معماری به تیمها اجازه دهد که بطور مستقل وسیع کار کنند. در روشها (راه های) کنترل شده و منظم فعالیت کنند.

:(مدیر سیستم)System Administratio

Architect:

معمار در مورد استراتژیهای فکر می کند که به تمام این اهداف (یا مصاعدمبین آنها) دست یابد.

در چرخه تاثیرات محیط و افراد بر کامپیوتر و بالعکس یکسری Teedbace هایی از خود معماری و سیستمی که برای آن ساخته شد بر سهامداران تاثیر می گذارد.

معماری بر ساختار سازمان تحت توسعه تاثیر می گذارد. یک معماری یک ساختاری برای یک سیستم تجویز می کند، معماری واحدهای نرم افزاری که می بایست پیاده سازی شوند (یا می بایست بدست آیند) و تجمع می شوند تا پروژه تحت توسعه می باشد. تیمهایی برای واحدهای نرم افزاری مجزا شکل دهی می شوند (برا مثال توسعه، تست، و فعالیت های مجتمع سازی هم اطراف واحدها تغییر می کنند). همچنین زمانبدی و بودجه (اما به منابع در قطعاتی مرتبط با واحدها انتساب می یابند. اگر یک

کمپانی در ساخت خانواده هایی از سیستمهای مشابه ماهر شد، آن کمپانی تمایل دارد تا سرمایه گذاری کند در هر تیمی توسط متخصصان آن حوزه. این یک فیدبکی است از معماری به سازمان تحت توسعه.

– معماری می تلند بر روی اهداف سازمان تحت توسعه اثر بگذارد. یک سیستم موفق که از آن ساخته شد می تواند یک کمپانی برای مهیا کردن یک جای پایی در یک محدوده خاص بازار یار می کند. یک معماری می تواند فرصت هایی برای تولید بارآور استقرار سیستمی مشابه فراهم کند، و سازمان ممکن است اهدافش را تنظیم کند تا برتری کارشناس جدید را برای تراز بازار بدست آورد.

این فیدبکی است از سیستم به سازمان تحت توسعه و سیستمهایی که می بایست ساخته شوند.

– معماری می تواند بر روی نیازمندیهای مشتری برای سیستم جدید اثر بگذارد از طریق دادن فرصت به مشتری تا یک سیستمی دریافت کند (بر اساس همان معماری) در یک اطمینان بیشتر، به موقع، و حالت مقرون به صرفه تر اگر سیستم بعدی از سیستم قبلی (پیش نویس) ساخته شود. مشتری ممکن است خواهان تخفیف یکسری نیازمندیها باشد بخاطر بدست آوردن این صرف جویی های تولید انبوه، باشد.

– فرآیند ساخت سیستم بر روی تجربیات معماران برای سیستم بعدی از طریق افزودن به پایه تجربه مربوطه، اثر می گذارد. یک سیستمی که بطور موفقیت آمیز اطراف یک گذرگاه ابزار یا ساخته شد یا ماشینهای حالت متناهی را محصور کرد (سیستمهای مشابه

ساتخته شده بر اساس همان روش بعداً تولید می شوند. بعبارت دیگر معماری که شکست خورد امکان خیلی خیلی کمی دارد که برای پروژه های بعدی انتخاب شود.

- یکسری از سیستمها تاثیر می گذارند و واقعاً فرهنگ مهندسی نرم افزار را تغییر می دهند، آن محیطی تکنیکی است که در آن سازندگان سیستم فعالیت کرده و یاد می گیرند. اولین DB های رابطه ای، تولید کنندگان کمپانی، و OS های مبتنی بر جدول این تاثیر را در OS ۱۹٦ و سریعتر در OS ۱۹۷ داشته اند، صفحه گستره جهانی (www) یک مثالی است برای OS ۱۹۹. ۲EE (ممکن است یک مثالی باشد برای اوسالیانه دهه قرن ۲۱.

نکته دیگر این است که نیازمندیهای سهامداران در واقع ویژگیهای کیفی مورد انتظار از یک سیستم می باشند که معماری آنها را از طریق انتخاب سیک های معماری بر آورده می کند. برای مثال:

قابلثیت تغییر را که ویژگی مورد انتظار Maintainer نگهداری کننده نرم افزاری می باشد می توان از سبک Call Sneturn استفاده نمود.

تحوه نمایش توسط UML:

UML یک نوعی از ساختارها را برای نهمایش انواع مختلفی از Modnie ها فراهم می کند. UML یک ساختار CLASS وارد که توصیف شئی گراری از یک Module می باشد. Pack age ها می توانند در مواردی که گروهبندی عمکردها (وظیفه مندیها) مهم می باشد، مثل زمانی که می خواهند لایه ها (Layers) و کلاسها را نمایش دهند. ساختار

Sub System می تواند بکار گرفته شود اگر یک توصیفی از واسط و رفتار مورد نیاز باشد.

طبق شکل زیر بیان می دارد که ذات رابطه ها به دیدهای ساخت Module توسط UML معنی می شوند.

Module Deromposition بوسیله رابطه Is - Pare - of (Aggregation) برآورده می شود. Module USes توسط رابطه dependency بدست می آید. و ایده Module Class توسط Generalization ، یا رابطه Is-a (که Inhertance نامیده می شود) تامین می شود.

Aggregation : در UML، ساختار Sabayatem برای نمایش Module هایی که شامل دیگر Module ها می باشند می تواند بکار رود، Class box معمولاً برای برگهای Decomposition بکار می رود.

Subsytem هم بطور Package ها و هم بطور Classiher ها (کلاس بندی) بکار برده می شوند. بعنوان Package آنها می توانند تجزیه (Decom pere) شوند و از این رو برای Module Aggregation مناسب می باشند. بعنوان کلاس بندی، آنها محتویاتشان را محصور می کنند و می توانند یک واسط صریح مهیا کنند.

Aggegation به یکی از سه روش زیر خودش را نشان می دهد.

- Module ها ممکن است تو در تو باشند

– نتیجه ای (رابطه قبل و بعدی) از دو تا دیاگرام (در حد امکان متصل) می تواند نشان داده شود، جائیکه دومی یک Depe از محتویات یک Module نشان داده شده در اول می باشد.

– یک کمانی که معین می کند Composition بین پدر و فرزند کشیده شده است.

در UML٦ Composition یک شکلی از Aggregation با رابطه غیر صریح مالکیت می باشد که رابطه بین قسمتهای زنده و همراه با کل (Whole) می باشند. اگر Module ، A ترکیبی از ماژولهای B و C می باشد B بنابراین B یا C نمی توانند بدون A موجود باشند، اگر A در زمان اجرا خراب شود، B و C خراب می شوند. بنابراین رابطه Composition مربوط به UML یک غیر صراحتی تحت ساختار پیاده سازی Unit (واحدها) دارد. رابطه همچنین Endows عناصر را باریک ویژگی زمان اجرا. بعنوان یک معمار باید مطمئن شد که با این خاصیت قبل از بکارگیری رابطه Composition مربوطه UML را راحت هستید.

Generalizatian : این رابطه قلب UML هست در حالیه Module ها توسط Class ها (یا بوسیله Sub System ها) نشان داده می شوند. شکل زیر نمادگذاری او سالیه (پایه) موجود در UML را نشان می دهد.

هر دوتا دیاگرام از لحاظ مفهومی یکسان می باشند. UML اجازه می دهد کی بیضی (...) در عوض یک Sub Module بکار برده شود. تا نشان دهد که یک Module می تواند فرزندی بیشتری از نشان داده شده دارد و بیشتر از یکی مشابه هستند. Module

شکل (Shap) پدر ماژول های Polygon، Circle، و Spline می باشد که هر کدام یک زیر کلاسی (Sub clacc)، بچه (Child) ، یا (Sdescen daw) از Shap می باشند. Shap خیلی کمی است، در حالیکه بچه هایش نسخه های خاص تری می باشند.

Dependency : نماد (پایه) برای Dependenc بصورت می باشد. بیترین مظهر با ارزش معماری Dependency در لایه ها (Layers) پیدا می شود. متاسفانه UML هیچ نوع عنصر رپیالیه (Primitive) مربوط به یک Layer (لایه) ندارد. بهر حال، UML می تواند لایه های ساده را با یکبار بردن Package ها بصورت نشان دهد. اینها مکانیزم کلی برای سازماندهی عناصر در گروه ها می باشند. UML، Package های از پیش تعریف شده برای System ها و Sub System ها دارد ما می توانیم یک Peckage اضافه تری برای لایه ها (Layers) با تعریف کردن آن بعنوان یک Package Streotype معرفی کنیم. یک لایه (Layer) می تواند بعنوان یک Package با محدودیت هایی Module ها را با هم گروه بندی می کند نشان داده شود. و Dependecy بین وPackage ها Allowed to use می باشد.

می توانیم یک لایه (Layer) را بکارگیری نشان (نماد) Package با نام Sterot ype >> Layer >> که معین می کند نام لایه را طراحی کنیم یا یک فرم و یژوال جدید از قبیل مستطیل سایه زده شده معرفی کنیم.

هیچ استراتژی مرجع تنهایی جهت مبتکرین دیدهای C δ C C در UML وجود ندارد، اما یکسری آسترنایتو وجود دارد. هر آسترنایتوی مزایا و معایب خودش را دارد. یک کاندید طبیعی جهت نمایش انواع C δ & C با مفهوم کلاس در UML شروع می شود.

شکل زیر ایده کلی کاربرد یک سیستم ساده Filter δ Pipe را نشان می دهد.

در این شکل نوع معماری Filter در UML با کلاس Filter نمایش داده می شود. نمونه هایی از Filter از قبیل Spliter بصورت Obsect های مربوطه نمایش داده شدند.

Component ها:

رابطه Type/instance در توصیفات معماری یک تطابقی نزدیکی به رابطه Class/ Objeot در مدل UML دارد. کلاسهای UML (Class) شبیه Compenrnt type در توصیفات معماری موجودیتهای Tist - Class هستند و ساختارهای از توصیف UML جهت توصیف ساختار، ویژگیها، و رفتار یک کلاس موجود می باشند، این را یک انتخاب خوبی جهت تجسم جزئیات و مولفه های معماری می توانند بصورت صفات کلاس یا با Association هایش Generalization جهت مرتبط ساختن مجموعه ای از Companen ها بکار برده می شود. معانی یک Insatanc یا Type می تواند با اتصال (حشمیه کردن) کی از Stereotyp های استاندارد تکوین یابد (برای مثال، مقوله بندی << Process >> می تواند به یک Compenent ضمیمه شود تا نشان دهد آن مطالعه بصورت یک Proiss جداگانه اجرا می شود.

توجه کنید به رابطه بین Merge & Sort و زیر ساختار ش که بر کار برد یک رابطه Dependecy ده است دارد.

Interface : واسطه های مولفه ها گاهی اوقات Port نامیده می شوند با پنج روشی که در شکل زیر نشان داده شده نمایش داده شوند.

توصیف اشکال برجسب درجه گویایی:

Option 1 : نمایش غیر صریح: این منجر به ساده ترین دیاگرامها می شود اما اجازه به یک مشکل آشکاری که هیچ روشی جهت ویژگی بندی نامها یا ویژگیهای و اسطهادر نمایش اولیه نمی شود. این روش قابل قبول است اگر مولفه فقط یک واسط داشته باشد، اگر واسطها بتوانند از سیستم با توپوثری استتاج شوند یا اگر دیاگرام جای دیگری پالایش شود.

Ptio 2 : واسطها بعنوان حاشیه (یادداشت): یک مهر برای اطلاعات در مورد واسطها مهیا می کند، گر چه حاشیه ها هیچ مقیاسی در UML ندارند از این رو نمی توانند یک واسط مرتبط نباشند، این روش معقول می باشد.

Option 3 : واسطها بعنوان صفات Class/ Obiect ها واسطها را بعنوان قسمتی از مدل ساختاری فرمال می کند، اما آنها می توانند فقط یک نمایش ساده در یک Class Diagram داشته باشند، اساساً یک Name و یک Type.

Option 4 : Interface ها بصورت Ineface های UmL نشان O – در UML یک توصیف سرجودی ازیک واسط دریکClassdiagram بستگی به نوع یک Compenent،

مهیا می کند. در یک Instance diagram یک نفش مربوط UML، مرتباً با یک نمونه واسط می شود و با نام نوع واسط توصیف می شود و یک روش فشرده با نشادن اینکه می نمونه مولفه طریق یک نمونه واسط ویژه در حال تعامل می باشد را نشان می دهد.

این روش بصورت ویژوال سیستم و جداگانه ای از Component ها و Interface مهیا می کند، در واسطهایی که بطور واضح و مفید دیده شوند.

بهر حال، این استراتژی هیچ وسیله ای جهت ترسیم سرویسهای مواد درخواست شده از طرف محیطComponet یک قسمت کلیدی از یک Interface می باشد، فراهم نمی کند. برخلاف Class ها، واسطهای UML صفات یا زیر ساختار ندارند.

Option 5: Interface بعنوان کلاسها: که واسطها را بصورت کلاسهایی شامل یک Compoent غالب بر فقدان گویایی و آسترنایتو ذکر شده قبلی می باشد. (حالا می توان زیر ساختار واسط را نمایش داد و می توان بیان کرد که یک Component teppe چندین واسط از همان Type دارد. یک نمونه ای Component بصورت یک Obiectی شامل یک مجموعه ای از Interfaceobject ها مدل می شود. بهر حال با نمایش Interface ها بصورت کلاس ها، مانه فقط دیاگرام را پازایت دارد می کنیم بلکه روشنی تشخیص ویژوال بین Interface ها و Componet ها را از دست می دهیم. ما می توانیم یک نمادگذاری مختلف دیگری بکار ببریم که در آن واسطها Interface ها شامل Class باشند، همانطوری که قسمت پاینی Option 5 نشان داده شده است.

تعیین نقاط تعامل دور از عقل می باشد، بهرحال بعنوان محدودیت که معمولاً مشخص می کند که یک کلاس شامل دیگر کلاسهایی است که نمونه هایشان ممکن است قابل دستیابی از طریق نمونه های کلاس پدر باشند یا نباشند.

Connectors : سه مورد مستدل در مورد نمایش Connector ها وجود دارد. این انتخاب بین گویایی و تطابق معنایی در یک است و پیچیدگی در دست دیگر مقابل هم قرار می گیرند.

Optionl : Type اتصالات بعنوان Assoca tion ها و Instance اتصالات بعنوان Linkها:

در یک دیاگرام Box & line معماری یک سیستم خطوط بین مولفه ها Connector ها می باشند. یک حالت نمایش Connector ها در UML، Association بین کلاسها یا Link های بین Object ها می باشد. این روش معمولاً بصورت ویژوال ساده می باشد، یک فاصله واصخی بین Component ها و Connecdor ها ایجاد می کند، و معروفترین رابطه در دیاگرامهای کلاس UML یعنی Association را بکار می برد. بعلاوه Association ها می توانند برچسب گذار می شوند، و یک Association جهت دار با Conncector می تواند با یک فلش معین شود. متاسفانه، Connector ها و Affociation ها معانی مختلفی دارند. یک سیستم در یک توصیف معماری بوسیله انتخاب مولفه ها با رفتار نمایش داده شده از طریق واسطهایش ساخته می شود و آنها را با Commector هایی که رفتارشان را هماهنگ می کند بهم متصل می کند. (فشار یک

سیستم بصورت رفتار جامع یک مجموعه ای از مولفه هایی که تعامل تعریف شده و بوسیله اتصالات بین آنها محدود می شود، تعریف می شود.

در مقابل، گر چه یک Association یا Link در UML یک توانایی را جهت تعامل بین عناصر مرتبط را نشان می دهد، مکانیزم Association اصولاً یک روشی است جهت توصیف رابطه ادراکی بین دو عنصر، بعلاوه یک Association یک رابطه ای است بین عناصر UML، از اینرو خودش بتنهایی نمی تواند در یک مدل UML باشد. نتیجتاً، یک Connector type بطور مجزا نمی تواند رد Uml نشان داده شود. در عرض شما می بایست دوباره قرار دادهای نامگذاری یا مقوسه بندی کردن معانی بدست آمده توسط توصیف را در زبان محدودیت شئی UML، طبقه بندی کنید، ثانیاً این روش اجازه تعیین واسطهای اتصالات Connetor's interfacesرا نمی دهد.

زبان محدودیت شیء UML، طبقه بندی کنید ، ثانیاً این روش اجازه تعین واسطهای اتصالات را نمی دهد.

: option2

Connecter type as assoction class

یک راه حلی با فقدان گویایی واجد شرایط assocation با یک کلاسی است که connector type را نمایش می دهد. در این شور connector type یا connector attributes می تواند از طریق صفات یک کلاس یا شیء بدست آید. متأسفانه،این تکنیک هنوز هیچ روشی از نمایش صریح واسطهای connector فراهم نکرده است.

: otion3

connector type as class & connector instance as bojects

یک روش برای دادن حالت به connecter های first-class در UML نمایش connector type از طریق کلاسها و connector instance ها از طریق اشیاء می باشد. کاربردن کلاسها و اشیاء همان چهار option برای نمایش که برای interface ها داشتیم می باشد. نه در همه آنها، بعنوان تفاسیر و واسطها بوسیله یک کلاس عینیت می یابد یا بعنوان کلاسهای فرزند، با یک کلاس cennecter شامل می شود. یک طرحی برای نمایش interface ها داده شده ، یک پیوستگی بین یک interface مؤلفه و یک interface اتصال ممکن است بصورت یک association یا یک dependency نمایش داده شود.

System ها:

جهت نمایش مؤلفه های ویژه و اتصالات و type هایشان نیاز است تا گرافهایی از مؤلفه ها و اتصالات یعنی system ها محصور شوند.

: otpion1

system as UML Subsystems

مکانیزم اولیه UML برای گروهبندی عناصر مرتبط package ها می باشد. در حقیقت، UML یک package stereotype استاندارد که subsystem نامیده می شود، تعریف می شود که مدلهایی UML که بیانگر قسمت منطقی یک سیستم می باشند را گروهبندی می کند. انتخاب subsystem ها جهت هر نگاشتی از مؤلفه ها و اتصالات مناسب می باشد،

و بطور خاص جهت گروهبندی کلاسها خوب کار می کند. یکی از مشکلاتی در ارتباط با بکاربردن subsystem ها که در UML 10 تعریف شد عبارت است از گر چه آنها یک classifier و یک package معنا کاملاً واضح نیست.

یکسری که می بایست قادر بود با یک subsystem بعنوان یک کلاس واحد شبیه موجودیت در مراحل معین فرآیند تولید رفتار کرد و بعداً قادر بود آنرا در ترمهایی از یک substructere خیلی جزئی شده پالایش کرد. داشتن این قابلیت انجام دادن اینکار subsystem را برای مدسازی مؤلفه های معماری بطور مناسب می سازد.

option2 : system as conaned objects : محتوی شیء می تواند جهت نمایش سیستمها بکار گرفته شود. مؤلفه ها بعنوان نمونه هایی از کلاسهای حاوی ، و connector ها با بکاربردن یکی از option های بررسی شده ، مسئول می شوند. اشیاء یک مرز محصور قوی ایجاد کرده و از طریق آنها نشان گذاری که هر نمونه ای از کلاس associated substructure خودش را دارد حمل می شود. بهر حال، این روش مشکلاتی دارد، مهمترین آن association می باشد که جهت مدل کردن connector های بین کلاسهای حامل از طریق کلاس محدود نمی شوند، مشکل آنست که ممکن نیست که بگوییم یک زوجی از کلاسها از طریق یک connector خاص تعامل می یابند، و بصورت association مدل می شوند. بنابراین، برای مثال، تعیین دو نوع کلاس حامل که از طریق یک association تعامل دارند برای نمونه هایی از کلاسهای بکارگرفته از هر جا صحیح نیست در غیر اینصورت در مدل

system as collaborations: option3

یک مجموعه ای از اشیاء در حال تبادل متصل از طریق likne های در UML با بکاربردن یک collaboration توصیف می شوند. اگر component ها بصورت object ها نمایش داده شوند، می توان collaboration ها را برای نمایش system ها بکار برده یک collaboration یک مجموعه ای از شرکاء و روابطی که برای یک هدف داده شده با معنی می باشند را تعریف می کند، در این مورد جهت توصیف ساختار زمان اجرای سیستم می باشد. شریک نقشهای دستبدی کننده ای که اشیاء بازی می کنند یا زمان تعامل با یکدیگر می کنند را تعریف می کند. بطور مشابه، روابط نقشهای association ی که اتصالات می بایست تطابق دهند را تعریف می کنند.

نمودارهای coliaboration می توانند جهت نمایش collaboration ها در مشخصات و سطح instance بکار روند. سطح مشخصات یک نمودار collaboration نقشهایی را نمایش می دهد که در collaboration تعریف می شود در یک الگو جهت توصیف کردن زیرساختارهای سیستم مرتب می شود.

یک سطح instance نمودار collaboration اشیاء و اتصالات شکل دهنده نقشها با نقشها در سطح مشخصات و تعامل این هدف را نشان می دهد.

بنابراین یک collaboration نمایش داده شده در سطح instance بهتر است جهت نمایش ساختار زمان اجرای یک سیستم بکار گرفته شود.

شکل زیر این روش را نمایش می دهد.

type معماری filter قبلا نمایش داده شد، Instance های فیلترها و pipe ها بصورت نقشهای دستبدی کننده مربوطه نمایش داده می شوند - برای مثال، splitter / نقش splitter را و نقشهای مرتبط را معین می کند.

اشیاء و اتصالات مطابق با آن نقشهای نمایش داده شده در دیاگرامهای collaboration در سطح instance بوسیله نامهای زیر خطی معین می شوند.

گر چه این روش طبیعی جهت توصیف ساختارهای زمان اجرا می باشد، اشتباه است که بگوییم هیچ روشی جهت نمایش صریح ویژگیهای سطح سیستم وجود ندارد.

همچنین یک عدم تطابق معنایی وجود دارد: یک c ollaboration یک تعامل قابل ارائه ای بین اشیاء توصیف کرده و یک توصیف جزئی در جائیکه یک پیکربندی معماری جهت بدست آوردن یک توصیف کامل لازم می باشد مهیا می کند.

دیدهای Allocation

در UML یک نمودار deplomentیک گرافی از گرههای متصل شده از طریق association های متبادل می باشد. شکل زیر یک مثالی برای این موضوع می باشد.

گره ها ممکن است شامل نمونه های component می باشد که معین می کنند که مؤلفه زنده است یا در گره اجرا می شود. مؤلفه ها ممکن است حاوی اشیائی باشند که معین می کنندن که شیء یک قسمتی از مؤلفه می باشد. مؤلفه ها با یکدیگر از طریق خطوط نقطه چین dependency متصل می شوند (در حد امکان از طریق واسطها) این بیان می کند که یک مؤلفه سرویس های مؤلفه دیگری را بکار می برد: یک stereotype ممکن است جهت تعیین کردن dependency دقیق اگر نیاز باشد بکار گرفته شود.

دیاگرام نوع depleyment ممکن است جهت نمایش مؤلفه هایی که ممکن است بر روی گره ها (nedes) اجرا شوند، بکار رود که این کار با بکاربردن خطوط نقطه چین با تضمین های مقوله بندی بکار رود.

یک گروه یک شیء فیزیکی زمان اجرا است که یک منبع پردازش را نمایش می دهد. گرهها شامل وسایل محاسباتی هستند همچنین شامل منابع پردازش مکانیکی یا اشیائی می باشند. گره ها ممکن است type هایی از نمونه های instance را نمایش دهند. نمونه هایی محاسباتی زمان اجرا، هر دوی اشیاء و مؤلفه ها ممکن است در نمونه های گره باقی بمانند (قرار گیرند). گرهها ممکن است توسط association ها به دیگر گرهها متصل شوند. یک association بیان کننده یک مسیر تبادلی بین گرهها می باشد. یک association ممکن است یک stereotype جهت تعیین طبیعیت مسیر تبادلی (برای مثال، نوع کانال یا نوع شبکه) داشته باشد.

تودرتو بودن سیمبول های در سیمبول دلالت بر یک association ترکیبی بین یک کلاس گره و کلاسهای متشکل یا یک اتصال ترکیبی بین یک شی ء گره و اشیاء تشکیل دهنده آن، می کند.

usabolity یا Clearnability

با توجه به تجزیه آن به نواحی زیر قابل اندازه گیری می باشد: - learnability (چقدر ساده و آسان است برای یک کار تا واسط سیستم را یاد بگیرد) - Efficiency (کارآیی):آیا سیستم به درخواستهای کاربران در یک سرعت مناسب پاسخ می دهد؟ - Memorability : کاربر می تواند بیاد آورد چگونه می بایست عملیات سیستم را زمان کاربرد سیستم انجام دهد؟ - Error avadance : آیا سیستم از خطاهای معمول	چون به تعامل بین استفاده کننده و سیستم بر می گردد بنابراین در معماری اثرگذاری است.	تعامل با Modifiability زیر شاخه Efficiency آن با Performance ارتباط دارد.

کاربران و خطاهای سیستم جلوگیری می کند؟ Error handtype آیا سیستم در خطاها به کاربر کمک می کند؟ - Satisfaction : آیا سیستم کار کاربر را به آسانی انجام می دهد؟ - ساخت واسط کاربر برای سیستم که واضح و روشن و آسان برای کاربرد باشد. - تطابق واسط کاربر با مدل ذهنی کاربران سیستم - بکاربردن تشابهات، استانداردها، و بکاربردن قراردادهای واسط و ... مهیاکردن کارآیی کافی و مهیاکردن کنترل برای جزئیاتی از قبیل زمانی که		

یک Radio button یا Check box بکار برده می شوند برای مشخص کردن انتخاب - هر چه واسط کاربر ساده تر باشد usability آن بیشتر است		

Availability (در دسترس بودن) قابل مشاهده در زمان اجرا

میزان فراهم بودن (Availabity) ارزیابی کردن در زمان است که چقدر سیستم بالا بوده (زنده بد) نسبت به کل زمان (مثلاً سیستم را ۲٤ ساعت Run کرده اید و ۲۳ ساعت کار شما را انجام داده و بقیه زمان را مثلاfail کرده است) زمان بین دو خروجی * متوسط زمان شکست = α متوسط زمان مورد نیاز برای اصلاح متوسط زمان شکست - شکست: رخدادی در سیستم پیش می آید که قابل پیش بینی نباشد.	از روشهای معماری (مثلاً سخت افزارها و برنامه backup و ...) استفاده می شود ت این ویژگی برآورده شود. مثلا نمی توان یک نرم افزار نوشت و بعد آنرا روی یک شبکه خراب استقرار داد پس باید ابتدا معمار همه چیز را دانست. و این تصمیمات را براساس (بستر تکنو، شبکه سخت افزار، نرم افزار...) بگیرد.	α با Reliability رابطه مستقیم دارد.

functionality (قابلیت عملکرد)

توانایی سیستم به انجام کار برای کاری که در نظر گرفته شده است. – اگر به مؤلفه صحیح واگذار نشود یا با دیگر مؤلفه ها هماهنگ نشوند (مثلا ندانند در چه زمانی است هر کدام از آنها task را شروع کنند) سیستم قادر نیست وظیفه را انجام دهد. – سیستم را به تعدادی مؤلفه می شکنیم که افراد مختلف در ساختار آن مشارکت داشته باشد پس عملکرد روی شکستن سیستم به مؤلفه ها دخالت دارد.	– این صنعت اینطور به نظر رسد که ذاتاً به معماری مربوط نیست ولی چون ساختار اولیه یک سیستم را نیازهای وظیفه مندی معنی می کند به معماری وابسته است. – معماری نرم افزار شرایط رسیدن به این ویژگی را در مواقعی که سایر صفات کیفی مهمتر باشد محدود می کند.	بر روی دیگر صفات محدودیت اعمال کرده یا خود با آنها تطبیق می دهد.

performance (قابل مشاهده در زمان اجرا)

نحوه اندازه گیری	ارتباط با معماری	ارتباط با دیگر ویژگیها
دو روش: ۱- تعداد رخدادها در یک پریود زمانی ۲- میزان پاسخگویی به یک رخدادخاص برای اینکار میزان زمان انتظار برای هر مشتری (Job) بر مبنای نرخ ورودی و اندازه صفها (اگر قابل پیاده سازی باشد) و تعداد تراکنشهای هر کدام و زمان هر تراکنش تخمین هایی می زنیم و براساس رویه های شبیه سازی میزان کارآیی یک سیستم را می فهمیم.	کارآیی در سطح معماری قابل درک است و می توان آنرا مدل و تحلیل کرد. کارآیی تابعی از معماری می باشد (کارآیی نشان می دهد که تعاملات بین اجزاء سیستم چقدر صحیح دیده شده است). میزان تعاملات توسط فراخوانی زیر رویه ها، همروند سازی فرآیند، با دیگر مکانیزمهای ارتباط (انتقال) منجر به تحریک کارآیی می شوند که سبب می شوند کارآیی تابعی از معماری باشد. با توجه به سابقه مهندسی نرم افزار کارآیی یک عامل مؤثر در معماری می باشد.	کارآیی یک عامل مؤثر در معماری می باشد و خیلی اوقات دیگر صفات کیفی را به خطر می اندازد. برای مثال: در تعامل است.

Security (قابل مشاهده در زمان اجرا)

نحوه اندازه گیری	ارتباط با معماری	
ارسال تقاضاهای مجاز و غیر مجاز بیش از حد معمول به سیستم سبب افزایش overhead می شو.د رسیدن به آدرس مقصد از مسیر کامپیوترهای دیگر	برای تضمین امنیت: سرویس دهنده مسئول احراض – اعتبار را خارج از سیستم قرار دهید که این تصمیم در سطح معماری گرفته می شود. – برنامه هایی که وظیفه شان مبصری شبکه است برای بازرسی شبکه است برای بازرسی و ثبت وقایع رخ داده شده در شبکه نصب می شوند که می تواند جزء خود Application یا جزء سیستم عامل (عمده) باشند برای این کارها نرم افزار می بایست نوشته شود و در نتیجه باید در معماری بگنجد – یک proxy بشکل firwau	

	تقاضاهای غیر مجاز داخلی و بیرونی را جلوگیری کند. این proxy از تصمیمات اثرگذار روی معماری است و در معماری باید در نظر گفته شود. - ایجاد کرنل امن: روشهای فوق درگیر این مطلب می شوند که یک تعداد مؤلفه خاص شناسایی شوند و این مسائل را از بقیه وظیفه مندیهای سیستم جدا کرده و آنها را به امنیت نسبت می دهند و همه اینها راه حلهای معماری می باشند. - خاص کردن مؤلفه ها	

Modifiability (غیر قابل مشاهده در زمان اجرا)

نحوه اندازه گیری	ارتباط	
- مستندات با توجه به اینکه تغییرات در سه رشد: ۱- اصلاحاتی که باعث تغییر در یک مولفه می شوند. ۲- اصلاحاتی که موجب قابل تغییر در بیش از یک مؤلفه می شوند. ۳- اصلاحاتی فراتر از قابلیت تغییر مثلاً در تغییر سبک، سبک client/server خوب نیست و باید تغییر کند. حذف توانمندیهای ناخواسته - تطبیق با محیط عملیاتی جدید - سازماندهی مجدد. به قابلیت نگهداشت براساس مستندات هم می توان اندازه گیری شود.	بیشترین ارتباط را با معماری دارد. توانایی اینکه بتوان تغییراتی با سرعت و توانایی بالا ایجاد کنیم به کیفیت معماری بستگی دارد. - ماژولاریتی ،محصورسازی مؤلفه اه - معماری مؤلفه ها را تعریف می کند و مسئولیت های (Responsibiblity) آنها را.	

Portability (غیر قابل مشاهده رد زمان اجرا)

- مؤلفه ها باید محیطهای محاسباتی جدید را بشناسند. (توانایی اجرای سیستم تحت محیطهای محاسباتی مختلف) این محیطها می تواند نرم افزاری، سخت افزرای، یا ترکیبی از هر دو باشند. - قابلیت حمل از Information heding نتجیه می شود. - نرم افزار کاربردی می بایست شود با بکاربردن لایه قابل جمعی و نباید مستقیماً به محیطی که با مجددسازی شارژ می شود دستیابی شود.	محصورکردن ملاحضات مختص بستر (platform) در یک معماری معمولاً لایه قابل حملی را شکل دهی می کنند یک مجموعه ای از سرویسهای نرم افزاری که نرم افزاری کاربردی یک واسط مجرد به محیطش می دهد. این واسط تغییر می کند حتی اگر پیاده سازی لایه از محیطی به محیط دیگر تغییر کند. - قابلیت حمل فقط در ساختار معماری مبتنی بر مؤلفه ها قابل رؤیت است.	با Modifiability تعامل دارد. چون برای برآورد شدن هر دو می بایست محصورسازی مؤلفه ها رعایت شود.

Reusability (غیر قابل مشاهده در زمان اجرا)

به تعداد مؤلفه های مستقل (مجزا) بستگی دارد. - چقدر سیستم از مؤلفه های سیستم قبلی (یا محصولات سیستم قبلی) استفاده کرد یا از ساختار سیستم قبل چقدر استفاده کرده.	Reusability در مؤلفه ها معماری که واحدهای استفاده مجدد می باشند و اینکه چگونه قابل به استفاده مجدد کردن یک مؤلفه بستگی دارد به اینکه چگونه با دیگر مؤلفه ها محکم متصل (کوپلاژ) باشد. به معماری بستگی دارد یا به عبارتی رابطه مؤلفه ها با دیگر مؤلفه ها چگونه است و هر چه مؤلفه ها مستقل تر باشد، شانس بیشتری برای استفاده مجدد دارند و از این نظر به معماری مربوط می شوند.	Reasability می تواند بعنوان نوع دیگر از Modifiability تصور شود. در بعضی موارد با Integrability مترادف می باشد.

Integrability (غیر قابل مشاهده در زمان اجرا)

قابلیت ساخت مؤلفه های سیستم جداگانه وی با همدیگر بدرستی کار کنند. آیا واسط مؤلفه ها تعریف شده یا نه؟ به میزان اینکه چه اندازه پیچیدگی های خارجی مؤلفه ها پنهان شده و به چه میزان مسئولیتها بین مؤلفه ها تقسیم شده است؟ به عملیات مؤلفه ها برای تجمیع و چگونگی ارتباط مؤلفه ها با همدیگر بستگی دارد. - این ویژگی توانایی قسمتهای یک سیستم که با همدیگر کار می کنند را اندازه گیری (بیان) می کند.	- اینکه به چه میزان پیچیدگی خارجی مؤلفه ها پنهان شده ، مکانیزمها و ... شناسایی و تعریف شده و به چه میزان مسئولیت ها بین مؤلفه ها تقسیم شده است هم موضوعات سطح معماری تلقی می شوند - با مسائل ساختاری نظیر سطح مستندات معماری و مخفی سازی اطلاعات ارتباط مستقیم دارد.	به وظیفه مندی (functionality) انتساب داده شده به مؤلفه هایی که می بایست مجتمع شوند و اینکه چگونه آن functionality به functionality محیط مؤلفه جدید، نسبت دارد (رابطه دارد) بستگی دارد.

Testability (غیر قابل مشاهده در زمان اجرا)

اگر نرم افزار یک اشتباه دارد در اجرای بعدی آیا خودش را نشان می دهد یا نه ؟ در حالت خاص به احتمالات بر می گردد که فرض کنید یک خطا داشته است،نرم افزار می بایست آنرا در حالت بعد نداشته باشد). - این به قابلیت مشاهده و کنترل بر می گردد. یک سیستم برای اینکه مهیا تست باشد، آن می بایست امکان داشته باشد تا حالت درونی هر مؤلفه و ورودیهایش را کنترل کند و سپس خروجیهایش را مشاهده کند.	با مسائل ساختاری چندگانه یا موضوعات معماری از قبیل سطح مستندات معماری، تفکیک موضوعات و درجه مخفی سازی اطلاعات ارتباط مستقیم دارد.	برای مثال در توسعه افزایش خاصت testability به روشی که integrability را ارتقا می دهد مفید می باشد.

زمان عرضه محصول به بازار

- هر interation زمانی یک Release و یک product به بازار عرضه می کنیم. - در فاز construction اینچنین محصول base function می باشد و بقیه Extend function می باشند.	معماری به جهت محدودیت های بازار از مسائلی متأثر می شود تا سرعت بیشتر شود.	با Reuseabiltiy رابطه دارد اگر از مؤلفه های پیش ساخته استفاده شود معمولاً زمان حرفه زودتر خواهد بود.

Cost

چقدر مؤلفه شناسایی شده و به ازای هر کدام هزینه لازم برای ساخت واسطهایی برقراری تعامل چقدر است؟	در تولید نرم افاز برنامه ریزی متأثر از معماری است. برنامه ریزی با هزینه (cost) ارتباط دارد و بالعکس cost روی معماری تأثیر می گذارد. - معماری های مختلف از لحاظ هزینه توسعه های مختلفی نتیجه می دهند مثال یک معماری مطمئن بر تکنولوژی که در سازمان قرار نمی گیرد برای عینیت یافتن کمتری در مورد خانه را دارد گرانتر خواهد بود.	این ویژگی متأثر از اکثر صفات کیفی است. برای مثال برای اینکه یک سیستم portuble و با کارآیی بالا بالا باشد می بایست هزینه زیادی مصرف کرد.

طول عمر پیش بینی شده برای سیستم

با توجه به فاکتورهای قابلیت اصلاح و قابلیت جابجایی سنجیده می شود.	مسائل حاشیه ای مثل افزایش طول عمر می تواند تعیین کند که چه ویژگیهایی از معماری اهمیت بیشتری دارند	با Portability و modifiability رابطه دارد.

Trageted Market (بازاریابی)

– آیا خواسته های مشتریان را برآورده می کند یا نه؟ – برای نرم افزاری با هدف کلی، بستری که سیستم در آن اجرا می شود بخوبی مجموعه ویژگیهای اندازه پتانسیل بازار (ظرفیت پذیرش بازار) را معین می کند.	برای یک بازار بزرگ ولی خاص روش خط تولید (product line) می بایست در نظر گرفته شود که در آن هسته اصلی سیستم مشترک اغلب از قبلی ها می باشد (portability) که در هسته لایه هایی از نرم افزار که خاص تر می باشند ساخته می شود که این کار یعنی معماری.	با portability و funotionality بیشتر رابطه دارد. دیگر ویژگی ها از قبیل performance و usability بعنوان یک نقش در بازاریابی تأثیر دارند. متأثر از Extensive legacy ststems می باشد.

Extensive Legace Systems

- آیا سیستمهای قدیمی با سیستم جدید کار می کنند. - برای مثال آیا سیستم های قدیمی با سرویس HTTP برای www کار می کنند.	- محدودیت های معماری همراه با این قابلیت جامعیت ها می بایست تحلیل شود. - می بایست مکانیزمی جهت تعیین جامعیت مناسب موجود باشد.	بر روی targeted market تأثیر می گذارد.

عملیات واحد

۱- Seperation : این امکان را بوجود می آورد که یک وظیفه مندی مشخص و روشن را به یک مولفه واحد و مجزا بگونه ای که این کار را از طریق واسط خوش تعریف به تعامل درآورد معرفی کنیم، بطوریکه:

۱- وظیفه مندی بقدری روشن باشد که بتوانیم در قالب کاملاً روشن، خلاصه کنیم.

۲- آنقدر قابل جداسازی از دیگران باشد که با واسط این کار را انجام دهد.

جداسازی با این هدف که تغییرات محیط بیرون به درون و بالعکس منتقل نشوند.

۲-Abastraction : پنهان سازی اطلاعات و حذف جزئیات غیر ضروری و پرداختن به اساس یک موجودیت فارغ از جزئیات پیاده سازی می باشد. عملیات ایجاد ماشین مجازی می باشد (که یک ماشین مجازی یک مولفه را در خودش پنهان می کند بگونه ای که وظایف را بعهده گرفته و پیاده سازی کرده و از بیرون قابل روئیت نباشد).

۳-Compression : نقطه مقابل Separation می باشد. فشرده سازی زمانی مورد نیاز است که جهت افزایش کارآیی یکسری از قواعد قبلی را بصورت موضعی نقض کنیم. مثلاً برای از بین بردن محدودیت حافظه، ماژولهایی را که قبلاً طراحی کردیم، فشرده کنیم.

فشرده سازی عملیات، حذف لایه ها و واسط هایی می باشد که وظایف یک سیستم را از هم جدا می کنند. که این لایه ها ممکن است سخت افزاری (مثل حافظه) یا نرم افزاری (مثل Process CAll) باشند.

٤- اشتراک منابع

این مفهوم مربوط به عملیاتی است که داده و سرویس هایی که مربوط به چند مشتری متصل استفاده می کنند با فرض اینکه مولفه های در حال استفاده مشترک به همدیگر صدمه وارد نکنند، انجام می گیرد.

گاهی برای مولفه کـا هش مولفه بخشهای مشترک را ایجاد می کنیم که تعامل را به حداقل برساند، به شرطی که آن بخش کاملاً محافظن شود(مثلاً DB داشته باشیم که داده ها در آن ذخیره شوند فقط دسترسی به آن کنترل شود). مثلاً در X- WFNDOUS برای کارهای گرافیکی چیزهایی را در Librarg قرار می دهیم و بقیه فقط انرا صدا می زنند و نیازی نیست که در همه برنامه ها آنرا بنویسیم. یا Data Reposistory.

حال هر کدام را در قالب یک مثال نشان می دهیم که چگونه بر کاهش پیچیدگی می تواند نقش مثبتی ارایه کنند. ابتدا صورت مثال توضیح داده شد و سپس عملیات واحد به مثال اعمال می شوند.

صورت مساله (مثال):

اولین مرحله در فهم هر کلاسی از سیستمها در قلمروی داده شده، فهم وظیفه مندیها (عملکردهایی) می باشد که از قبیل یک کلاس از سیستمها محاسبه شده اند. بنابراین اومعاینه مرحله در معماری نرم افزار واسط انسان - کامپیوتر (User Fnerface) می

بایست معین کردن مجموعه حداقلی از وظیفه مندیهای از قبیل معماریهایی که باید پشتیبان شوند، می باشد.

حداقل دو وظیفه منی که هر سیستم با یک واسط کاربر می بایست فراهم کند (انجام دهد) عبارتند از :

Predentation (تعامل با کاربر) و Application (لایه در زیرین سیستم) همچنینی دو تا جنبه و جودی و سلسله مراتبی در ارتباط با HCL (Human Computer Fnterdare) وجود دارد که در وظیفه اصلی می بایست توسط کاربر کامل شود (تصینف یک شند، ایجاد یک صفحه گسترده، فرزستان Mail) به زیر وظایف (ایجاد یک فایل جدید، اصلاح یک تون، تایپ یک پاراگراف) تقسیم می شود تا جایی که به سطحی از عملیات فیزیکی (تایپ یک کارکتر، کلیک کردن بر روی یک آیکون) بریم شکسته می شود که معمولاً یک کاربر در Presentation انجام می دهد.

این تجزیه و قوای می تواند استدلالی از یک Dialopue مابین کاربر و Application باشد. بنابراین سومین نوع از وظیفه مندی که در هر سیستم محاوره ای می بایست پشتیبانی شود، Clialogne می باشد.

مدل یکپارچه تمکام وظیفه مندیهای یک سیستم با هم در یک مدل واحد تابیده می شوند. مدل یکپارچه بسادگی یک مدل قابل پیاده سازی می باشد.

یک ارائه وظیفه مندی از یک ساختار سستم در شکل ۷-۱ نشان داده شده است.

وظیفه مندیها با هم در یک مولفه ساختاری واحد جمع شوند.

یک برنامه ای که با یک زبان سطح بالا از قبیل C نوشته می شود، با فراخوانی مستقیم بسته های نرم افزاری واسط کاربر از قبیل Windows Toolkit و Motif این ساختار را دارند. یک Applieation زمانی که نیاز به ورودی از کاربر دارد یا می خواهد خروجی را به کاربر نمایش دهد وظیفه مندی Toolkit را فراخوانی می کند.

قسمت Dialogue ضمنی است (بعنوان روابطی مابین فراخوانی های Toolkit ها می باشد).

MODEL-VFEW -CONTROLLER

الگوی MVC :

برای دیدن اینکه چگونه NIVC، Modifability و Protablilty را نشان می دهد، نخست با تصویر سیستم یکپارچه شکل ۷-۱ شروع کرده و عملیات واحد را اعمال می کنیم. ابتدا Modifiability را در نظر می گیریم:

این شمابه آن است که یک مجموعه ای از وظیفه مندی و Dialogue و Presentation (تمام کدهای مرتبط با یک شئی Application واحد) قصد دارند منتقل از بقیه برنامه تغییر (اصلاح) یا بنز این کار نیاز به کاربرد عملیات واحد Part - Whole Decompsition دارد، که بهترین روش پشتیبانی از محدود کردن تاثیرات یا محدوده تغییرات می باشد.

اجرای نتایج Part - Whole Decompsition در یک معماری در شکل ۷-۲ نشان داده شده است.

رسیدگی بعدی این است سیستم پا وسایل ورودی و خروجی متفاوت دریچه بندی شود.

برای اینکار ما نخست عمل واحد Seperation را اعمال می کنیم، حالا ما می خواهیم که وسایل ورودی و خروجی را Separate کنیم نه فقط از Dialogue و Applieation بلکه از هر کدام از آنها را بخوبی. این کار شکل ۷-۳ را نتیجه می دهد، جائیکه Presentation از Dialogne و Application تفکیک شده است، و قسمتهای ورودی و خروجی Presentation از دیگری تفکیک شده اند.

همانطور که در شکل ۷-۳ نشان داده شده است نتیجه اعمال این عمالیات واحد نسخه اصلی Model (که ما Dialogue و Application را برای هر خاص فراخوانی کنیم. View (خروجی). و Controler (ورودی) می باشند.

توسعه های بعدی تولیدی، MVC به PAC بسط داده می شود.

MVC به PAC (Presentation, Application, Control) تکوین می یابد.

مدل PAC،واسط شخص - کامپیوتر (HCF) بطور ابتدائی توسط اهداف کیفی نسبتاً مختلف تحریک می‌شود: Modifiabilty و Scability اهمیت Modifiabilty منجر به یک تجزیه وظیفه مندی مشابه آنچه در شکل ۷-۱ نشان داده شده است می شود، جائیکه Presentation (P)، Application (A) و Dialogue (که Control نامیده می شود با C معنی می شود، در شکل نمایش داده می شود.

بهر حال، کاربرد Seperation با یک Presentation مجهز Application و Dialoague فقط اهمیت Modifiab: lity را نشان می دهد.

یک سیستم تعاملی پیچیده یک قسمت بزدگی از نرم افزار است، ما مایلیم که صفت کیفی Dcablity تضمین کنیم.

صفت Scablilty (مقیاس پذیری) که به یک گروهی از مولفه ها اعمال می شود توسط Part - Whole Decompsition تضمین می شود.

هنگام این عمل: هر سه تایی P-A-C می تواند به قسمتهایی تجزیه شود. حالا نام Control بیش از Dialogue با مهمی تر است، از این رو این زیر مولفه نیاز دارد تا اجزایش را بخوبی میانجگری تعاملات ما بین Application و Prodention کنترل کند.

در این روش ساختار موجود در شکل ۷-٤ بطور کامل از عملیات واحد مشتق می شود.

نظر به اینکه PAC برای سیستمهای محاوره ای بکار گرفته شد، مشهود شد که معماریهای تجزیه شده (Decomposed) در این روش Portability را تضمین نمی کنند. به این دلیل است که یک سیستمی که بطور کامل به قسمتهایی یکنواخت تجزیه می شود قادر نیست تاثیرات Porting را محلی کند. برای مثال، هر مولفه ای در سیستم ممکن است یکسری که واسط های کاربر معینی را در خود داشته باشد.

در ترمهای PAC، هر سه تایی ممکن است یک زیر مولفه Presentation داشته باشد و هر کدام از زیر مولفه ها می بایست بطور خاصی Port شوند. هنگامیکه عملیات واحد با همدیگر برخورد دارند یکی از دو استراتژی اهداف کیفی مطلوب ممکن است به خطر افتد با دامنه اجرای عملیات واحد می بایست محدود شود. استراتژی آخر در مورد PAC اعمال می شود. Part - Whole Decompsition کمک برآورده کردن Scablilty یک

کارکرد می کند، آن در مد (حالت) محدود اعمال می شود و فقط به Dialogue اعمال می شود. این به Dialogue اجازه می دهد که به قطعات قابل مدیریت، قابل کد شدن تجزیه شود، هر کدام از آنها دوباره می توانند از طریق مکانیزم منظم Composition به کل مجتمع شود.