سازمان سما

وابسته به دانشگاه آزاد اسلامی

دانشگاه سما واحد حاجی آباد

مهندسی نرم افزار

منبع : مهندسی نرم افزار(پرسمن)

دکتر شیر افکن

WWW.HREZAPOUR.IR

حمیدرضا رضاپور

مهندسی نرم افزار

فصل هشتم : مفاهیم طراحی

Design Concepts

WWW.HREZAPOUR.IR

نگاهی گذرا

طراحی چیست؟

طراحی، نمایش یا مدلی از نرم افزار ایجاد می کند.

مدل طراحی جزئیات مربوط به معماری نرم افزار ، ساختمان داده ها ، واسط ها و مولفه های لازم برای پیاده سازی سیستم را فراهم می کند.

موارد کانون توجه مدل خواسته ها : توصیف داده ها ، قابلیت ها و رفتار مورد نیاز

چه کسی آن را انجام می دهد؟

مهندس نرم افزار

چرا اهمیت دارد؟

طراحی به شما این امکان را می دهد تا سیستم یا محصولی را که قرار است ساخته شود، مدل سازی کنید.

طراحی جایی است که در آن **کیفیت** نرم افزار تثبیت می شود.

مراحل کار کدام است؟

- معماری سیستم یا محصول باید نمایش داده شود.
- واسط هایی که نرم افزار را به کاربران نهایی و همچنین به مولفه های سازنده خودش مرتبط می سازند ، مدل سازی می شوند.
- مولفه های نرم افزار که در ساخت سیستم به کار می روند ، مدل سازی می شوند.

محصول کار چیست ؟

یک مدل طراحی که شامل نمایش هایی از معماری و واسط ، نمایش در سطح مولفه ها و نمایش های استقرار می شود .

چگونه اطمینان حاصل کنم که درست از عهده کار بر آمده ام؟

طراحی نرم افزار

طراحی برزخ میان دو دنیا می باشد.

هدف طراحی ، ایجاد مدل یا نمایشی است که استحکام ، تناسب و لذت از خود نشان دهد.

برای رسیدن به این مقصود ، باید تنوع و سپس همگرایی را عملی سازید.

WWW.HREZAPOUR.IR

طراحی در حیطه مهندسی نرم افزار

Design within the Context of Software Engineering

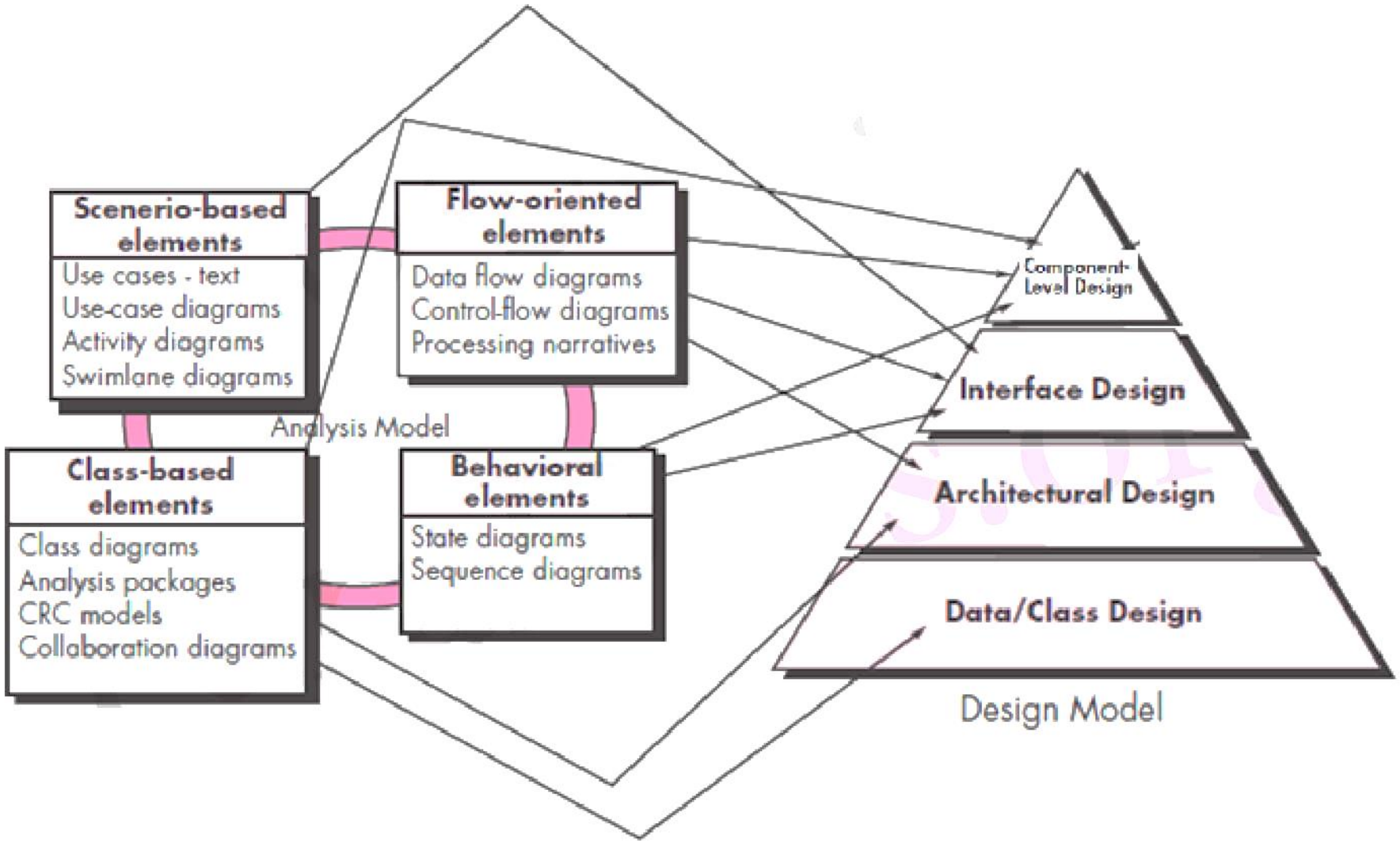
طراحی نرم افزار هسته اصلی نرم افزار را تشکیل می دهد و به کار گیری آن مستقل از نوع مدل فرآیند نرم افزار مورد استفاده است.

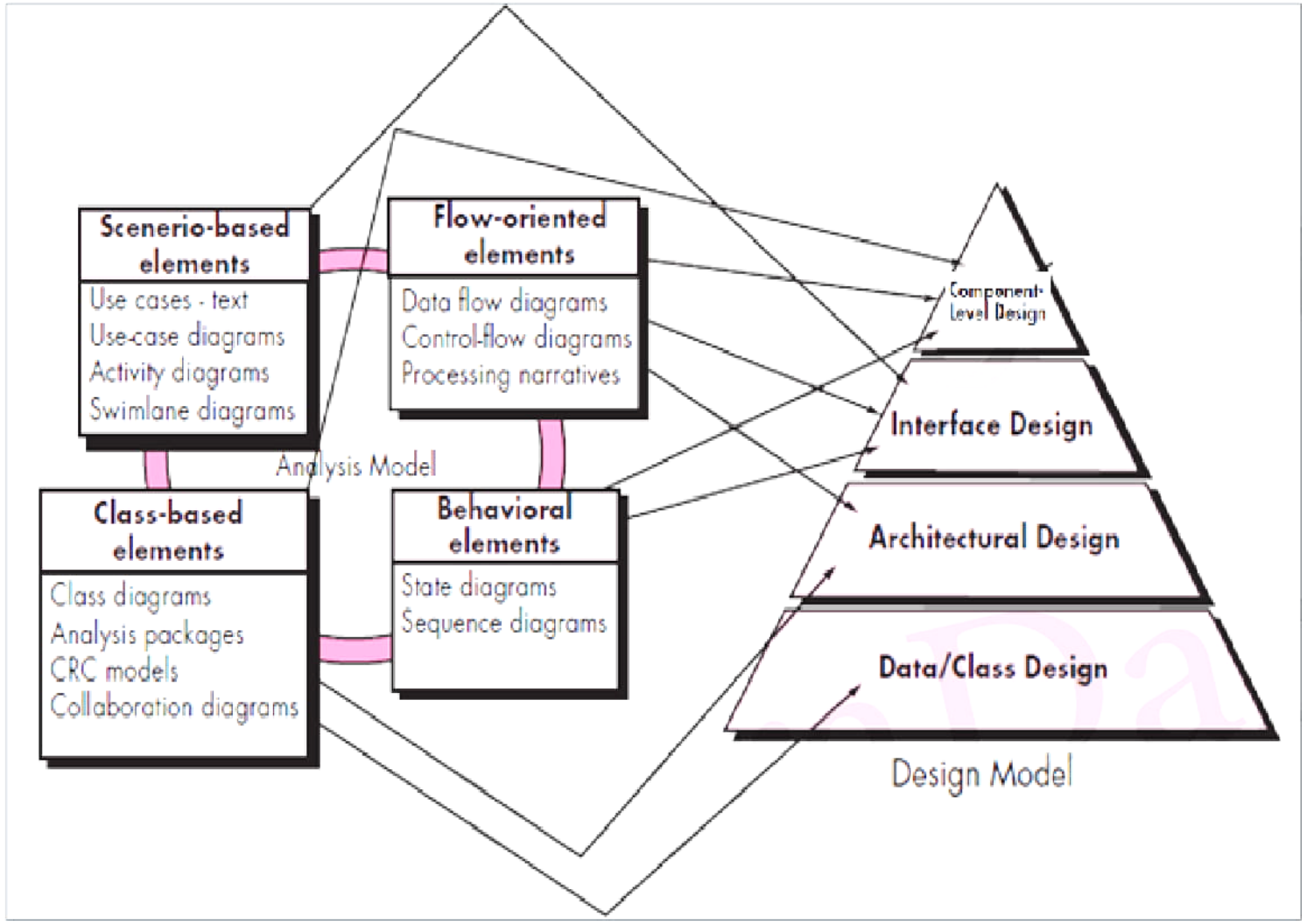
طراحی نرم افزار :

- پس از تحلیل و مدل سازی خواسته ها آغاز می شود.
- آخرین کنش مهندسی نرم افزار در فعالیت مدل سازی است.
- صحنه را برای ساخت (تولید و آزمایش کد) آماده می کند.

برگردان مدل خواسته ها به مدل طراحی

جریان اطلاعات طی طراحی نرم افزار





طراحی داده ها / کلاس ها:

مدل های کلاس ها به کلاس های طراحی و ساختمان داده های لازم برای پیاده سازی نرم افزار تبدیل می شوند.

طراحی معماری:

رابطه ی میان عناصر ساختار اصلی نرم افزار ، سبک های معماری و الگوهای معماری تعریف می شود.

طراحی واسط ها:

چگونگی برقراری ارتباط نرم افزار با سیستم هایی که با آن همکاری متقابل دارد و با افرادی که از آنها استفاده می کنند ، توصیف می شود.

طراحی در سطح مولفه ها:

عناصر ساختاری معماری نرم افزار را به توصیف روالی از مولفه های نرم افزار تبدیل می کند.

اطلاعات بدست آمده از مدل های مبتنی بر کلاس ها، مدل های جریان و مدل های رفتاری به عنوان مبنایی برای طراحی مولفه ها عمل می کنند.

WWW.HREZAPOUR.IR

طراحی در مقایسه با کد نویسی

SAFEHOME

Design versus Coding

The scene: Jamie's cubicle, as the team prepares to translate requirements into design.

The players: Jamie, Vinod, and Ed—all members of the *SafeHome* software engineering team.

فرآیند طراحی

The Design Process

طراحی نرم افزار، فرآیندی مبتنی بر تکرار است که از طریق آن خواسته ها به نقشه ای برای ساخت نرم افزار ترجمه می شوند.

WWW.HREZAPOUR.IR

صفات و دستور العمل های کیفیت نرم افزار

Software Quality Guidelines and Attributes

سه خصوصیت که به عنوان راهنمایی برای تکامل طراحی خوب به شمار می روند:

۱- طراحی باید همه ی خواسته های صریح موجود در مدل خواسته ها را پیاده سازی کند و باید همه ی خواسته های ضمنی مطلوب طرف های ذینفع را پاسخ گو باشد.

۲- طراحی باید یک راهنمای خوانا و قابل فهم برای کسانی باشد که کدها را تولید می کنند و برای کسانی که نرم افزار را آزمایش و بعداً پشتیبانی می کنند.

۳- طراحی باید تصویر کاملی از نرم افزار باشد که دامنه های داده ای ، عملیاتی و رفتاری را از دیدگاه پیاده سازی به نمایش بگذارد.

دستور العمل های کیفیتی

Quality Guidelines

۱ - طراحی باید معماری را نشان دهد که

(۱) با استفاده از سبک ها یا الگوهای معماری شناخته شده ایجاد شده باشند.

(۲) از مولفه هایی تشکیل شده باشد که خصوصیات طراحی خوبی از خود به نمایش بگذارند.

(۳) به شیوه ای تکاملی قابل پیاده سازی باشند تا به این ترتیب ، پیاده سازی و آزمایش تسهیل گردد.

۲- طراحی باید پیمانه بندی شده باشد ؛ یعنی نرم افزار باید به طرزی منطقی به عناصر یا زیر سیستم هایش افراز شده باشد.

۳- طراحی باید حاوی نمایش های متمایزی از داده ها ، معماری ، واسط ها و مولفه ها باشد.

۴- طراحی باید به ساختمان داده ای منجر گردد که برای کلاس هایی که قرار است پیاده سازی شوندو از الگوهای داده ای قابل تشخیص بیرون کشیده می شوند ، مناسب باشد.

۵- نتیجه طراحی باید مولفه هایی باشد که از خود ، خصوصیات عملیاتی مستقل به نمایش بگذارند.

۶- طراحی باید به واسط هایی بیانجامد که پیچیدگی ارتباطات میان مولفه ها و ارتباط آنها با محیط خارجی را کاهش دهند.

۷- طراحی باید با استفاده از روشی تکرارپذیر به دست آید که خود با اطلاعات حاصل از تحلیل خواسته های نرم افزار به دست می آید.

۸- طراحی باید با به کارگیری نمادهایی ارائه شود که معنا و مفهوم را به خوبی برساند .

Quality Attributes

صفات کیفیتی

صفات کیفیتی FURPS :

- **قابلیت عملیاتی (Functionality)**

با تعیین مجموعه ویژگی ها و قابلیت های برنامه ، عملیات کلی که تحویل می شوند و امنیت کل سیستم ارزیابی می شود.

- **قابلیت کاربرد (Usability)**

با اندازه گیری عوامل انسانی ، زیبایی شناسی کلی ، سازگاری و مستندسازی سنجیده می شود .

- **قابلیت اطمینان (Reliability)**

با اندازه گیری فراوانی و شدت شکست ها ، صحت نتایج خروجی ، میانگین زمان شکست(MTTF) ، توانایی خلاصی یافتن از شکست و قابلیت پیش بینی برنامه تعیین می شود.

- **کارایی (Performance)**

- **قابلیت پشتیبانی (Supportability)**

ترکیبی است از قابلیت نگهداری (بسط پذیری ، قابلیت انطباق و قابلیت سرویس)، قابلیت آزمایش ، سازگاری ، قابلیت پیکربندی ،

سهولت نصب سیستم و سهولت پیدا کردن مسائل.

همه صفات کیفیتی نرم افزار به یک میزان اهمیت ندارند و در کاربردهای مختلف اهمیت متغیری دارند.

بنابراین ، صفات کیفیتی را باید همان زمان که طراحی شروع می شود ، مورد توجه قرار داد نه پس از کامل شدن طراحی و شروع ساخت.

تکامل طراحی نرم افزار

The Evolution of Software Design

تکامل طراحی نرم افزار، فرآیندی پیوسته است که اکنون نزدیک به شش دهه را پشت سر گذاشته است.

Early design work concentrated on criteria for the development of modular programs and methods for refining software structures in a top- down manner.

Procedural aspects of design definition evolved into a philos- ophy called *structured programming*.

Later work proposed methods for the translation of data flow or data structure into a design definition.

Newer design approaches proposed an object-oriented approach to design derivation.

More recent emphasis in software design has been on software architecture and the design patterns that can be used to implement software architectures and lower levels of design abstractions.

Growing emphasis on aspect-oriented methods ,model-driven development , and test-driven development emphasize techniques for achieving more effective modularity and architectural structure in the designs that are created.

خصوصیاتی مشترک در همه ی روش های طراحی

۱- سازوکاری برای ترجمه مدل خواسته ها به نمایش طراحی

۲- یک نمادگذاری برای نمایش مولفه های عملیاتی و واسط های آنها

۳- ابتکاراتی برای پالایش و افراز

۴- دستوالعمل هایی برای ارزیابی کیفیت

WWW.HREZAPOUR.IR

مفاهیم طراحی

مفاهیم بنیادی طراحی نرم افزار ، چارچوب لازم برای درست انجام دادن را فراهم می آورند.

مفاهیم مهم طراحی نرم افزار که هر دو شیوه سنتی و شی ء گرا را برای توسعه نرم افزار شامل می شوند:

- انتزاع(Abstraction)
- معماری(Architecture)
- الگوها (Patterns)
- جداسازی دغدغه ها(Separation of Concerns)
- پیمانه بندی(Modularity)
- پنهان سازی اطلاعات(Information Hiding)
- استقلال عملیاتی(Functional Independence)
- پالایش(Refinement)
- جنبه ها (Aspects)
- بازآرایی (Refactoring)
- مفاهیم طراحی شیء گرا (Object-Oriented Design Concepts)
- کلاس های طراحی(Design Classes)

انتزاع (Abstraction)

هنگامی که راهکاری پیمانه ای را برای مسأله در نظر می گیرید ، سطوح انتزاع متعددی ممکن است پیش آید.

در بالاترین سطح انتزاع ، راهکار در قالب عبارت هایی کلی و با استفاده از زبان محیط مسأله ، بیان می شود.

در سطوح پایین انتزاع، توصیف مشروحی تری از راهکار ارائه می شود.

در پایین ترین سطح از انتزاع ، راهکار به شیوه ای بیان می شود که به طور مستقیم قابل اجرا و پیاده سازی باشد.

انتزاع فرایندی: دستورالعمل هایی است که وظیفه ای مشخص و محدود دارند.

انتزاع داده ای : مجموعه ای از داده ها با نام مشخص است که شی ء داده ای را توصیف می کند.

معماری(Architecture)

معماری نرم افزار به ساختار کلی نرم افزار و شیوه هایی مربوط می شود که این ساختار باعث یکپارچگی مفهومی در سیستم می گردد.

معماری در ساده ترین شکل خود، ساختار یا سازماندهی مولفه های برنامه، شیوه ی تعامل این مولفه ها و ساختمان داده های قابل استفاده توسط این مولفه هاست.

ولی از دیدگاه گسترده تر، مولفه ها را می توان طوری تعمیم بخشید که عناصر اصلی سیستم و تعامل های آنها را نشان دهد.

یکی از اهداف مهندسی نرم افزار، به دست آوردن یک نمای معماری از سیستم است.

WWW.HREZAPOUR.IR

مجموعه ای از خواص که خوب است به عنوان بخشی از معماری در نظر گرفته شوند:

- **خواص ساختاری**

در این جنبه از نمایش طراحی معماری ، مولفه های سیستم و شیوه ی بسته بندی و تعامل آنها با یکدیگر تعیین می شود.

- **خواص عملیاتی اضافی**

می بایست چگونگی برآورده شدن خواسته های کارایی ، ظرفیتی ، قابلیت اطمینان ، امنیت ، انطباق پذیری و سایر خصوصیات سیستم در معماری طراحی در نظر گرفته شود.

- **خانواده های سیستم های مرتبط**

در طراحی معماری باید الگوهایی تکرارپذیر بدست آورده شود که به طور متداول در طراحی خانواده هایی از سیستم های مرتبط با آن مواجه می شویم.

با توجه به خواص مذکور ، طراحی معماری را می توان با به کارگیری یک یا چند مدل متفاوت به نمایش درآورد:

در **مدل های ساختاری** ، معماری به صورت مجموعه ای سازمان یافته از مولفه های برنامه نمایش داده می شود.

مدل های چارچوبی با تلاش برای شناسایی چارچوب های طراحی معماری تکرارپذیری که در انواع مشابه کاربردها مشاهده می شوند ، سطح انتزاع را در طراحی بالا می برند.

مدل های پویا به جنبه های رفتاری معماری برنامه می پردازند و چگونگی تغییر پیکربندی سیستم یا ساختار را به عنوان تابعی از رویدادهای خارجی مشخص می کنند.

در **مدل های فرایندی** طراحی فرایند تجاری یا فنی ای که سیستم باید در نظر بگیرد ، کانون توجه قرار می گیرد.

مدل های عملیاتی را می توان برای به نمایش درآوردن سلسله مراتب عملیات ها در یک سیستم به کار برد.

الگوها (Patterns)

الگوی طراحی ، توصیفی است از :

یک ساختار طراحی که یک مسأله طراحی را در حیطه ای خاص حل می کند و

نیروهایی بینابینی که ممکن است بر شیوه به کار گیری و استفاده از این الگو تأثیرگذار باشند.

هدف هر الگوی طراحی، فراهم ساختن توصیفی است که طراح به کمک آن بتواند تعیین کند که:

۱- آیا این الگو برای کار فعلی قابل استفاده است.

۲- آیا این الگو قابل استفاده مجدد است.

۳- آیا این الگو می تواند به عنوان راهنمایی برای توسعه ی یک الگوی مشابه ولی با ساختار و عملکرد متفاوت عمل کند.

جداسازی دغدغه ها

(Separation of Concerns)

دغدغه: ویژگی یا رفتاری که به عنوان بخشی از مدل خواسته ها برای نرم افزار مشخص می شود.

جداسازی دغدغه ها ، یک مفهوم طراحی است که پیشنهاد می کند هر مسأله پیچیده ای را می توان بهتر حل کرد اگر به قطعاتی تقسیم گردد که هر یک را بتوان به طور مستقل ، حل و یا بهینه سازی کرد.

با جداسازی دغدغه ها به قطعات کوچکتر ، زمان و تلاش کمتری صرف حل مسأله می شود.

پیمانه بندی

(Modularity)

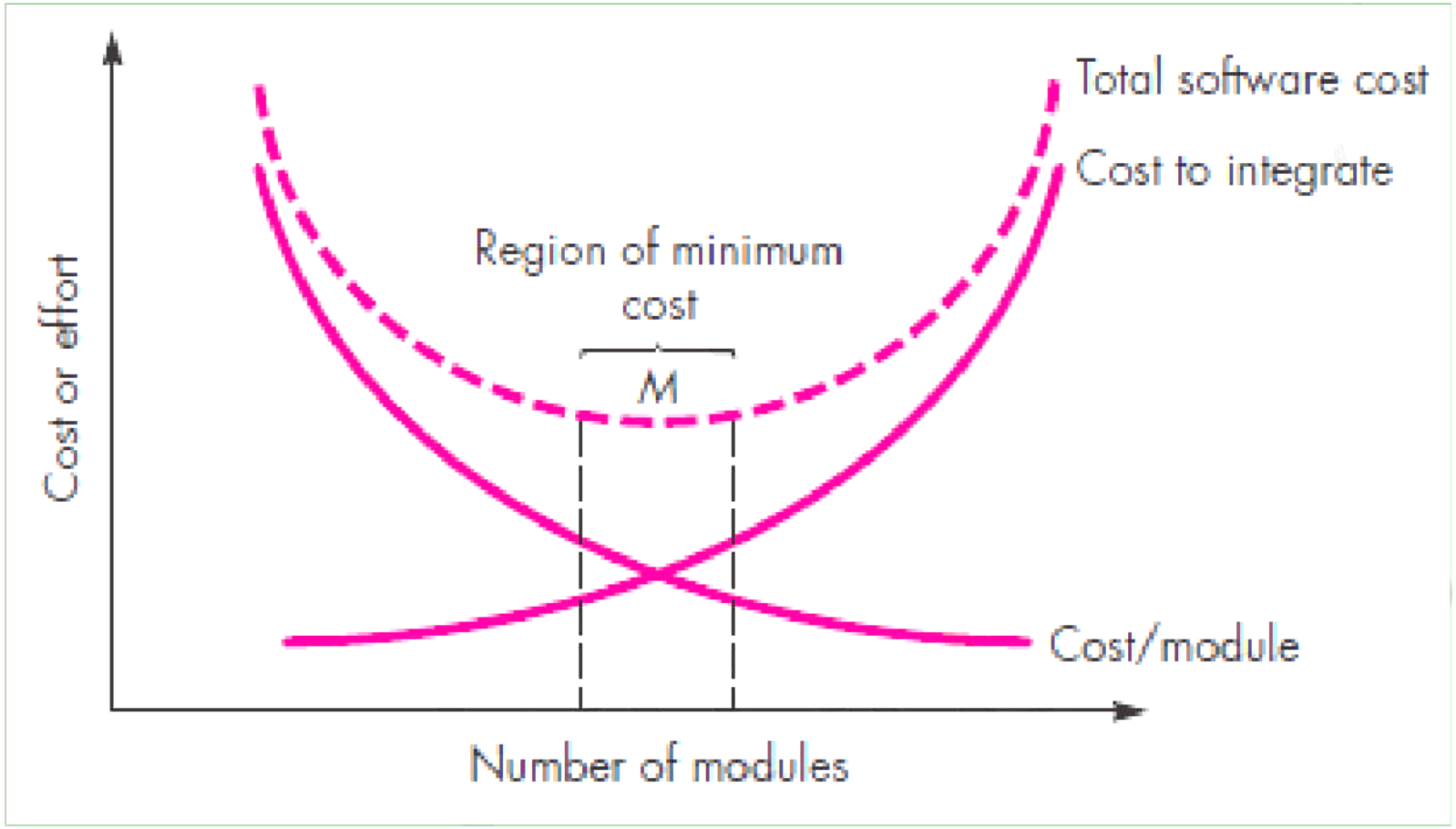
پیمانه بندی، متداول ترین نمود جداسازی دغدغه هاست.

نرم افزار به مولفه های جداگانه و دارای نام تقسیم می شود که گاه از آنها با عنوان پیمانه یاد می شود، از انسجام این پیمانه ها ، خواسته های مسأله برآورده می شود.

پیمانه بندی ، تنها صفت نرم افزار است که اداره ی هوشمندانه ی برنامه را میسر می سازد.

تقریباً در همه ی نمونه ها ، باید طراحی را به چندین پیمانه تقسیم کنید تا درک و فهم مسأله آسانتر شود و در نتیجه هزینه لازم برای ساخت نرم افزار کاهش یابد.

پیمانه بندی و هزینه نرم افزار



شکل نشان می دهد که تلاش (هزینه) لازم برای توسعه یک پیمانه نرم افزار، با افزایش تعداد پیمانه ها کاهش می یابد. اگر مجموعه یکسانی از خواسته ها را در نظر بگیریم، بیشتر بودن تعداد پیمانه ها به معنای کوچک تر شدن اندازه ی هر پیمانه خواهد بود.

ولی با رشد تعداد پیمانه ها، تلاش (هزینه) مرتبط با انسجام بخشیدن به این پیمانه ها نیز رشد می کند.

پنهان سازی اطلاعات (Information Hiding)

پیمانه ها باید طوری مشخص و طراحی شوند که اطلاعات موجود در یک پیمانه نتواند در دسترس پیمانه های دیگری قرار گیرد که به این اطلاعات نیاز ندارند.

این مجموعه پیمانه تنها با پیمانه دیگری ارتباط برقرار می کند که حاوی اطلاعات لازم برای دستیابی به عملکرد نرم افزار است.

استفاده از پنهان کردن اطلاعات به عنوان ملاک طراحی برای سیستم های پیمانه ای ، هنگامی بیشترین مزایا را به همراه خواهد داشت که اصلاحاتی طی آزمایش و سپس طی نگهداری نرم افزار لازم باشد.

چرا که احتمال انتشار خطا های سهوی طی انجام اصلاحات در سایر نقاط نرم افزار ، کمتر می شود.

استقلال عملیاتی

(Functional Independence)

مفهوم استقلال عملیاتی ، نتیجه مستقیم جداسازی دغدغه ها ، پیمانه بندی و مفاهیم پنهان سازی اطلاعات و انتزاع است.

استقلال عملیاتی با توسعه پیمانه هایی با عملکرد یگانه و دوری جستن از تعامل بیش از حد با سایر پیمانه ها بدست می آید.

چرا باید در ایجاد پیمانه های مستقل بکوشید؟

توسعه نرم افزارهایی با پیمانه های مستقل، راحت تر است، چون قابلیت های عملیاتی را می توان به واحدهای کوچک تر تقسیم کرد و واسط ها ساده می شوند.

به طور خلاصه ، استقلال عملیاتی کلید طراحی خوب و طراحی کلید کیفیت نرم افزار است.

استقلال با استفاده از دو ملاک کیفیتی قابل ارزیابی است:

یکپارچگی ، نشان از قدرت عملیاتی نسبی یک پیمانه دارد. اتصال ، نشان از استقلال در میان پیمانه ها دارد.

- **یکپارچگی (cohesion)**

یکپارچگی بسط طبیعی مفهوم پنهان سازی اطلاعات است.

یکپارچگی، شاخصی کیفی از میزان تمرکز یک پیمانه بر تنها یک چیز است.

پیمانه یکپارچه ، به تعامل اندک با سایر مولفه های موجود در بخش های دیگر سیستم نیاز دارد.

- **اتصال (coupling)**

اتصال شاخصی کیفی از میزان ارتباط یک پیمانه با سایر پیمانه ها و جهان خارج است.

اتصال به پیچیدگی واسط های میان پیمانه ها، نقطه ای که در آن ورود یا ارجاع به یک پیمانه

انجام می شود وداده هایی که از واسط عبور می کنند ، بستگی دارد.

پالایش (Refinement)

پالایش مرحله ای، یک راهبرد طراحی از بالا به پایین است.

پالایش در واقع همان فرایند تعیین جزئیات است.

با توصیف عملکرد که در سطح بالایی از انتزاع تعریف می شود، کار خود را شروع می کنید.

سپس جزئیات مربوط به بیان اولیه را تعیین می کنید و با هر بار پالایش پیاپی ، جزئیات بیشتر و بیشتری به آن اضافه می کنید.

پالایش و انتزاع، مفاهیمی مکمل یکدیگرند.

به کمک انتزاع می توانید روال ها و داده ها را از نظر داخلی مشخص کنید، ولی نیاز افراد متفرقه به داشتن آگاهی از جزئیات سطح پایین را برطرف می سازد.

پالایش به شما کمک می کند تا با پیشرفت طراحی، جزئیات طراحی را آشکار کنید.

(Aspects)

جنبه ها

در همان حال که تحلیل خواسته ها رخ می دهد، مجموعه ای از دغدغه ها آشکار می شود.

دغدغه پیش نیاز، خصوصیتی از سیستم است که در میان خواسته های متفاوت کاربرد دارد.

جنبه ، نمایشی از یک دغدغه پیش نیاز است.

شناسایی جنبه ها به طوری که طراحی بتواند به صورت مناسب آنها را ضمن انجام پالایش و پیمانه بندی، در برگیرد، اهمیت دارد.

در حالت ایده آل ، هر جنبه به صورت پیمانه ای جداگانه (مولفه) پیاده سازی می شود نه به صورت تکه هایی از نرم افزار که در سرتاسر چندین مولفه پراکنده و در هم و بر هم شده باشند.

برای دستیابی به این مقصود، معماری طراحی باید از ساز و کاری برای تعریف یک جنبه پشتیبانی کند.

consider two requirements for the SafeHomeAssured.com WebApp.

Requirement *A* is described via the **ACS-DCV** use case.

A design refinement would focus on those modules that would enable a registered user to access video from cameras placed throughout a space.

Requirement *B* is a generic security requirement that states that *a registered user must be validated prior to using* SafeHomeAssured.com.

This requirement is applicable for all functions that are available to registered *SafeHome* users.

As design refinement occurs, *A** is a design representation for requirement *A* and *B** is a design representation for requirement *B*.

Therefore, *A** and *B** are representations of concerns, and *B* crosscuts A**.

the design representation, *B**, of the requirement ***a registered user must be validated prior to using*** SafeHomeAssured.com, is an **aspect** of the *SafeHome* WebApp.

بازآرایی (Refactoring)

یک فعالیت مهم طراحی که برای بسیاری از روش های چابک پیشنهاد شده است، بازآرایی است که تکنیکی برای سازمان دهی مجدد به شمار می رود و طراحی (یا کد) یک مولفه را ساده می کند، بدون اینکه قابلیت عملیاتی رفتار آن را تغییر دهد.

بازآرایی : فرایند تغییر دادن سیستم نرم افزاری به گونه ای که رفتار خارجی کد(طراحی) تغییر نکند و در عین حال، ساختار درونی آن بهبود یابد.

هنگامی که نرم افزار بازآرایی می شود، طراحی موجود برای زوائد، عناصر استفاده نشده ی طراحی ، الگوریتم های ناکارآمد یا غیر ضروری ، ساختمان داده ای ضعیف یا نامناسب، یا هر گونه شکست طراحی دیگر که قابل اصلاح باشد، بررسی می شود تا طراحی بهتری به دست آید.

نتیجه، نرم افزاری خواهد بود که راحت تر می توان به آن انسجام بخشید و آزمودن و نگهداری آن آسان تر است.

کلاس های طراحی

(Design Classes)

به موازاتی که مدل طراحی تکامل پیدا می کند، مجموعه ای از کلاس های طراحی را تعریف می کند که کلاس های تحلیل را با فراهم آوردن جزئیات طراحی پالایش می کنند و یک زیر ساخت نرم افزاری را پیاده سازی می کنند که راهکار تجاری را پشتیبانی می کند.

پنج نوع متفاوت از کلاس های طراحی را می توان توسعه داد که هر کدام لایه متفاوتی از معماری طراحی را نشان می دهند:

- **کلاس های واسط کاربری:** تمام انتزاع های لازم برای تعامل میان انسان و کامپیوتر(HCI) را تعریف می کنند.
- **کلاس های دامنه تجاری :** صفات و سرویس ها را مشخص می کنند که برای پیاده سازی عنصری از دامنه تجاری مورد نیازند.
- **کلاس های پردازش:** انتزاع های تجاری سطح پایین لازم برای مدیریت کامل کلاس های دامنه ی تجاری را پیاده سازی می کنند.
- **کلاس های ماندگار:** انباره های داده ها را نشان می دهند که پس از اجرای نرم افزار ، ماندگار می شوند.
- **کلاس های سیستمی:** عملیات های مدیریتی و کنترلی را پیاده سازی می کنند که کارکرد سیستم را میسر می سازند و ارتباط میان درون و بیرون محیط کامپیوتری را برقرار می سازند.

چهار مشخصه برای کلاس طراحی خوش فرم

۱- کامل و کافی (Complete and Sufficient)

طراحی باید پنهان سازی کاملی از همه ی صفات و سندهایی باشد که به طور منطقی برای آن کلاس انتظار می رود. کافی بودن به آن معناست که کلاس تنها حاوی متدهایی باشد که برای دستیابی به هدف کلاس کفایت می کنند ، نه کمتر و نه بیشتر.

۲- سادگی (Primitiveness)

هنگامی که یک سرویس با یک متد پیاده سازی شد ، کلاس نباید راه دیگری برای دستیابی به همان هدف فراهم سازد.

۳- یکپارچگی بالا (High Cohesion)

یک کلاس طراحی یکپارچه ، دارای مجموعه ای کوچک و متمرکز از مسئولیت هاست که صفات و متدها را برای پیاده سازی همان مسئولیت ها به کار می برد.

۴- اتصال پایین (Low Coupling)

اگر در یک مدل طراحی میزان اتصال بالا باشد (همه ی کلاس های طراحی با همه ی کلاس های طراحی دیگر همکاری کنند) ، پیاده سازی سیستم، آزمایش آن و نگهداری آن در گذر زمان دشوار می شود. به طور کلی ، کلاس های طراحی در داخل یک زیر سیستم فقط باید آگاهی محدودی از سایر کلاس ها داشته باشد. این محدودیت ، که قانون دمتر نامیده می شود ، پیشنهاد می کند که یک متد فقط باید به متدهای موجود در کلاس های همسایه پیام ارسال کند.

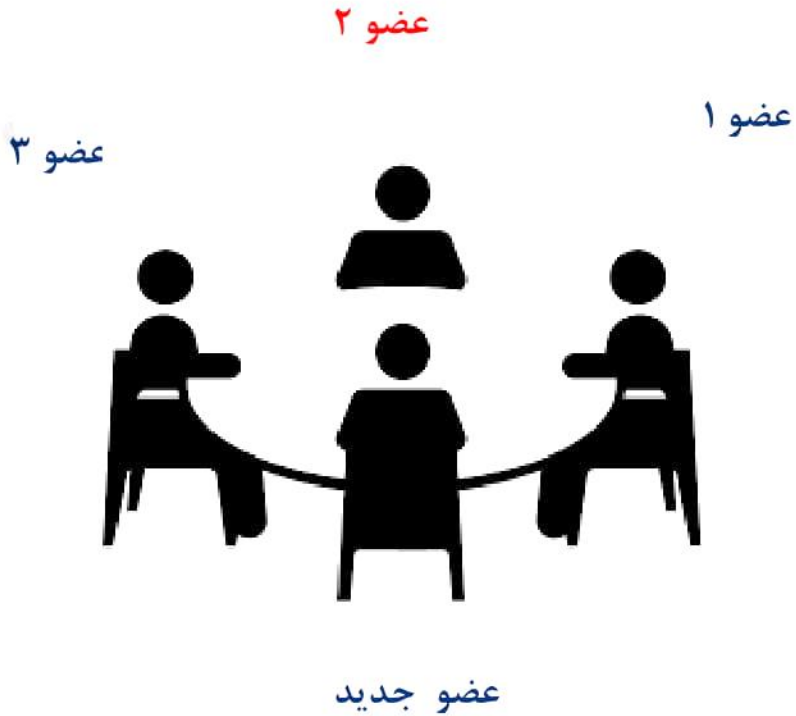
مفاهیم طراحی

SafeHome

Design Concepts

The scene: Vinod's cubicle, as design modeling begins.

The players: Vinod, Jamie, and Ed—members of the *SafeHome* software engineering team. Also, Shakira, a new member of the team.



مدل طراحی

(Design Model)

عناصر طراحی داده ها

عناصر طراحی معماری

عناصر طراحی واسط ها

عناصر طراحی در سطح مولفه ها

عناصر طراحی در سطح استقرار

پایان