آموزش توسعه نرم افزار های شیء گرا
توسط UML

با تشکر
حامد ذوالقدری

فصل اول- مفاهیم شیء گرایی

مقدمه

شئ گرایی برای توسعه نرم افزار اولین بار در سال 1960 پیشنهاد شد، این روش پس از 20 سال به طور گسترده مورد استفادة جامعه نرم افزاری قرار گرفت. توسعه دهندگان نرم افزار در دهه 1980 توجه جدی خو د را روی شئ گرایی معطوف کردند. تکنولوژی شئ، قابلیت استفاده مجدد را برای مؤلفه های نرم افزاری به ارمغان آورد و این نیز به نوبه خود در تسریع توسعه نرم افزار و تولید محصول با کارایی بالا تاثیر بسزایی دارد؛ بعلاوه سیستمهای شئ گرا، براحتی قابل توسعه و به سهولت با محیط سازگار- از نظر تعامل با سیستمهای موجود در محیط استفاده از نرم افزار- می شوند . دیدگاه شئ گرایی یک سیر تکاملی دارد؛ همچنانکه در بخشهای بعدی خواهیم دید، تعیین همه کلاسهای لازم برای یک سیستم دریک تکرار تا اندازه ای غیرممکن است و به محض تکمیل مدلهای تحلیل و طراحی نیاز به کلاسهای جدید در سیستم نمایان می شود.

درک سیستمهای پیچیده وتولید نرم افزار برای چنین سیستمهایی توسط افرادی که در این زمینه تجربه کافی ندارند، کاری بس مشکل است . همچنین محصولی که این افراد تولید می کنند کارایی لازم را نخواهد داشت، در اینجا مهندسی نرم افزار به کمک افراد آمده و با مطالعه روشها و فنون مختلف مسیر توسعه و تولید نرم افزار را هموار می- سازد. تجربیات بدست آمده در این زمینه، متدها و فرآیندهای متنوعی را برای توسعه نرم افزار در اختیار توسعه دهندگان قرار داده و ابزارهای مناسبی نیز این روشها را پشتیبانی می کنند.

درتوسعه یا ساخت نرم افزار برای یک سیستم، مشتری باید تعریف دقیقی از سیستم را در اختیار توسعه دهنده قرار دهد. در توصیف سیستم، زبان طبیعی تا آن اندازه دقیق نیست که بتوان همه نیازمندیها، ساختار و رفتار سیستم را با آن بیان کرد و کد نویسی نیز چنان وارد جزئیات می شود که به یکباره نمی توان سیستم را در این سطح تشریح کرد. لذا برای درک سیستم دست به مدل سازی می زنیم و مؤلفه های سیستم ، زیر سیستمها و رفتار سیستم را به صورت نمودارهای گرافیکی ترسیم می نماییم تا موارد قابل کاربرد و مهم به صورت برجسته به چشم بخورد و هیچ موردی در حوزة سیستم از قلم نیافتد .

در متد شئ گرا از زبان مدلسازی استاندارد UML که در فصل چهارم به تفصیل خواهدآمد، استفاده می شود. این زبان به وسیله ابزارهای مختلفی نظیر visio ، Rational Rose و ... پشتیبانی می شود، میتوان از UML در فرآیندهای مختلف استفاده کرد.

مفاهیم اساسی

در این بخش مفاهیم اساسی توسعة نرم افزار شئ گرا را معرفی می کنیم. در بالا به متد و فرآیند اشاره شد اما هیچ تعریفی از آنها ارائه نشد، حال این دو مفهوم کلی را بصورت زیر تعریف می کنیم.

متد، متدلوژی و اشیاء

متد مجموعه ای از وظایف را جهت تعیین نیازمندیها، تحلیل، طراحی، برنامه ریزی، تست و پشتیبانی مشخص می کند. از نظر فنی فرآیند توسعه نرم افزار- متدلوژی- یک قالب کاری برای وظایف لازم جهت ساختن یک نرم افزار با کیفیت بالاست. در واقع متدلوژی، فرآیندی ساختارمند جهت توسعه نرم افزار است که به وسیله فنون و ابزارها حمایت می شود.

متد شئ گرا برپایه شئ استوار است، دیدگاه شئ گرا دنیای واقعی مسئله را بصورت مجموعه ای از اشیاء مرتبط به هم می بیند. شئ یک موجودیت است که در دامنة مسئله نقش تعریف شده ای دارد و دارای حالت، رفتار و شناسة خاص خودش است. شئ می تواند یک ساختار ، نقش ، مکان و ... باشد؛ شئ داده و رفتار را در خود کپسوله میکند و از دسترسی اشیاء دیگر به داده های خود جلوگیری و همچنین تا ثیر تغییرات محیطی بر این داده ها را کاهش می دهد و تنها راه دسترسی به این داده ها استفاده از اعمال یا سرویس های خود شئ می باشد. کلاس نو ع اشیاء را نشان می دهد و شامل ویژگی های مشترک مجموعه ای از اشیاء می باشد، شئ نمونه ای از کلاس است . داده های شئ تحت عنوان صفات در کلاس شناخته می شوند و مقادیر این صفات است که شئ را از دیگر اشیای همنوع متمایز می نمایند. اعمال به دستکاری تعداد محدودی از صفات می پردازند و ارتباط بین کلاس ها و دیگر عناصرسیستم نیز از طریق همین سرویسها- اعمال – صورت می گیرد. به عبارت دیگر کلاس یک مشخصه کلی (قالب ، الگو یا طرح اولیه)است که مجموعه ای ازاشیاء مشابه را نشان می- دهد.نماد گرافیکی کلاس در شکل زیر نشان داده شده است، این نماد شامل سه قسمت است که بترتیب نام کلاس ، لیست صفات و لیست اعمال را نشان می دهند.

نام کلاس
لیست صفات
لیست اعمال

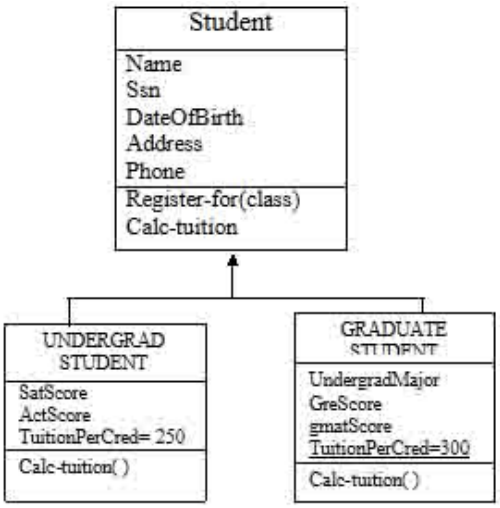
با تعریف کردن اشیاء موجود در سیستم از نو ع یک کلاس خاص، این اشیاء همه صفات، اعمال و روابط کلاس مربوطه را به ارث می برند. یک فوق کلاس شامل ویژگی های مشترک (صفات و اعمال)جمعی از کلاسهاست و زیرکلاس یک حالت خاص

ازفوق کلاس است که به آن تخصیص نیزگفته می شود. این تعاریف از وجود یک سلسله مراتب نشان می دهد که در آن کلاسهای تعمیم(فوق کلاس) توسط کلاسهای تخصیص به ارث برده می شوند، ممکن است که هر کدام ازکلاس های تخصیص دارای یکسری صفات و اعمال اختصاصی اضافی باشند. مجموعه مقادیر موجود برای یک صفت در یک کلاس، دامنه مقادیر آن صفت را نشان می دهد.

پیامها وسیله برقراری ارتباط و تعامل بین اشیاء می باشند ، این پیامها شئ مقصد را تحریک می کنند تا یک کار خاص را انجام دهد. سرویسی که در شیء فرستنده پیام تولید می کند، یک پیام با قالب

message:[destination, operation, parameters]

ارسال میکند که در آن destination شیء گیرنده و operation سرویسی از شیء گیرنده است که پیام را دریافت می کند و parameters شامل اطلاعات لازم جهت انجام موفق سرویس خواسته شده است. شکل 1-2 مثالی از کلاسهای تعمیم و تخصیص را نشان می دهد که در آن برای دانشجو یک فوق کلاس دانشجو داریم که شامل داده ها و اعمال مشترک بین دانشجویان دورهٔ لیسانس و فوق لیسانس است، همچنین دو زیر کلاس تخصیص جداگانه برای دانشجویان لیسانس و فوق لیسانس نشان داده شده است که حالات خاصی از کلاس دانشجو هستند. در عمل ما شیئی از نوع فوق کلاس دانشجو نخواهیم داشت، در این حالت به کلاس student یک کلاس مجرد گفته می- شود . کلاس مجرد کلاسی است که هیچ شیئی از آن نوع نداشته باشیم.



شکل ۱-۲ : کلاسهای تعمیم و تخصیص دانشجو

کپسوله سازی، ارث بری و چند ریختی

با توجه به مطالب ذکر شده در بالا، شیء گرایی به واسطه سه خاصیت مهم کپسوله سازی، ارث بری و چند ریختی یک روش منحصر بفرد است . بطور کلی کپسوله سازی تکنیکی است که جزئیات پیاده سازی داخلی شئ را از دید سایر اشیاء و مؤلفه های سیستم پنهان می کند(مخفی سازی اطلاعات). عمل، تابعی است که در تمام نمونه های یک نوع وجود دارد و سایر اشیاء تنها از طریق اعمال موجود در یک شئ می توانند به اطلاعات این شئ دسترسی داشته باشند، بنابراین اعمال یک واسط برای کلاس بشمار می روند. واسط رویهٔ بیرونی کلاس را بدون اینکه ساختار درونی وچگونگی پیاده سازی اعمال را نشان دهد ، نمایان می سازد. خاصیت کپسوله سازی داده و عمل دریک کلاس بطور عمده مزایای زیررا دارد.

- جزئیات پیاده سازی داخلی داده و روتین ها از دنیای خارج قابل مشاهده نیست (مخفی سازی اطلاعات). این خاصیت تاثیر تغییرات محیطی بر شیء را کاهش می-دهد.
- ساختمان داده ها و اعمالی که برای دستکاری داده ها در نظر گرفته شده با هم ادغام شده و تحت یک نام (اسم کلاس) شناخته می شود و این اساس مؤلفهٔ قابل استفاده مجدد را فراهم می نماید.
- واسطه های ما بین اشیاء کپسوله شده ساده اند، لزومی ندارد که شئ فرستندهٔ پیام از جزئیات ساختمان داده درونی شئ مقصد اطلاع داشته باشد.

خاصیت کلیدی بعدی که روش شیءگرا را از سایر روشها متمایز می دارد، ارث بری است. در شکل بالا زیر کلاس Graduate student همه صفات و اعمال متناظر شده با فوق کلاسش(Student) را به ارث می برد، بدین معنی که تمام ساختمان داده و الگوریتمهایی که برای فوق کلاس student طراحی و پیاده سازی شده برای کلاس تخصیص نیز در دسترس می باشد و هیچ کار اضافی لازم نیست انجام بگیرد . و عملاً از قابلیت استفاده مجدد استفاده می کند.

اگر تغییری در داده ها و اعمال فوق کلاس انجام داده شود بلافاصله توسط تمام زیر کلاس های آن فوق کلاس به ارث برده می

شود. لذا از مکانیزم سلسله مراتبی برای انتشار تغییرات در سیستم استفاده می شود و این مهم است که در هر سطح از سلسله مراتب صفات یا اعمال به آنچه که توسط سطح بعدی به ارث برده می شود، اضافه می‌شود. بنابراین وقتی که لازم است کلاسی جدید ایجاد شود، مهندس نرم افزار می تواند به یکی از راه کارهای زیر عمل کند:

- می توان بدون استفاده از ارث بری کلاس را بطور مستقل طراحی نمود.
- بررسی ساختار سلسله مراتبی کلاس های موجود و یافتن سطحی از ارث بری که بیشترین صفات و اعمال کلاس جدید را داراست، سپس کلاس جدید در سطح بالاتر قرار داده می شود و صفات و اعمال موجود در سطح پایین تر را به ارث می برد.
- سلسله مراتب را ساختاربندی مجدد می نماییم تا کلاس جدید بتواند صفات و اعمال لازم را به ارث ببرد.

چند ریختی مشخصه ای دیگر از شئ گرا است که تا حد زیادی از کارهای لازم جهت توسعه سیستم موجود می کاهد. در واقع چند ریختی بمعنای انجام چند کار مختلف توسط یک چیز است؛ بعنوان مثال برای یک روتین که چهار شکل گرافیکی مختلف، بیضی ، دایره ، مربع و لوزی را رسم کند در حالت عادی برای هر نوع باید متد مخصوص نوشته شده و برای نوع جدیدتر نیز باید متد خاص خود را اضافه نماییم. برای پاسخ گویی به این مشکل در شئ گرایی از چند ریختی استفاده می کنیم، در اینجا همه این انواع به عنوان زیر کلاسهایی از کلاس عمومی Graphقرار می گیرند . هر زیر کلاس یک عمل بنام draw تعریف می کند، شئ می تواند یک پیام draw را به هر یک از اشیاء نمونه از یک زیر کلاس بفرستد، شئ گیرندهٔ پیام از تابع draw ی خود می خواهد که رسم مناسب را تولید کند. وقتی که یک رسم الخط دیگر اضافه شود، یک زیر کلاس با تابع draw ی خاص خودش ایجاد می شود و بدین ترتیب کار طراحی ساده تر و از انجام کارهای تکراری جلوگیری می شود. بطور خلاصه، چند ریختی این توانایی را به ما می دهد که چند عمل مختلف را تحت یک نام واحد داشته باشیم .

شناسایی عناصر مدل شئ

در زیر مواردی را مطرح می نماییم که با استفاده از آنها شناسایی عناصر مدل شئ که که شامل کلاس، شئ ، صفات ، اعمال و پیامها است، راحتتر صورت می گیرد.

شناسایی کلاسها و اشیاء

هنگامی که محیط اطراف خود را نگاه می کنیم براحتی اشیاء فیزیکی زیادی را می- توان شناسایی و تعریف (صفات واعمال) نمود، اما وقتی که فضای مسئله یک نرم افزار را بنگریم، ملاحظه می کنیم که کار مشکلتراست. برای شناسایی اشیاء روی متنی که با زبان طبیعی سیستم را توصیف کرده یک پارس انجام می دهیم و تمام اسمها یا عبارات اسمی رادر یک جدول قرار می دهیم سپس مترادفها را از جدول حذف می نماییم، حال از این اسمها یا عبارات اسمی، آنهایی شئ هستندکه در یکی از موارد زیر صدق کنند:

- موجودیتهای خارجی (سیستمهای دیگر ، ابزارها و افراد) که تولیدکننده یا مصرف کننده اطلاعات در سیستم کامپیوتری هستند.
- اشیائی (گزارش ، علائم ، سیگنال)که قسمتی از دامنه های اطلاعاتی مسئله هستند.
- رویدادهایی (مانند ارسال اطلاعات برای تصمیم گیری یک ربات) که در بطن عملیات سیستم نهفته است.
- نقشهایی (مدیر ، مهندس ، فروشنده) که افراد در این نقشها با سیستم تعامل دارند.
- واحد سازمانی (تقسیمات، گروه ، تیم) که مربوط به برنامه کاربردی است.
- اماکن (کف اتاق، جای خالی در سیستم رزرواسیون) که عملکرد سیستم به آن وابسته است.
- ساختارها (حسگرها، کامپیوترها) که نوع نقاط انتهایی یا کلاسهای مرتبط با سایر اشیاء را تعیین می نمایند.

شناسایی صفات شئ

شیئی که در مرحلهٔ قبل مشخص شده بوسیله مجموعه صفاتش تعریف می شود ، صفات تصویر واضحی از شئ در خلال مسئله به ما می دهند. برای توسعه یک مجموعه با معنا از صفات یک شئ، آنالیست باید متن پروسه را دوباره مطالعه کند و آنچه که درباره یک شئ معین است، مشخص بنماید.

شناسایی اعمال شئ

اعمال، رفتار شئ را تعریف می کنند و صفات شئ را با برخی روشها تغییر می دهند. بطور مشخص یک عمل مقدار یک یا چند صفت را که در شئ قرار دارد ، تغییر می- دهد . بنابراین اعمال باید از طبیعت صفات شئ و ساختمان دادهٔ اشتقاقی از آنها اطلاع داشته و با روشهایی پیاده سازی شوند که قادر به دستکاری این ساختمان داده باشند.
اگر چه انواع مختلفی از اعمال وجود دارد اما عموما به سه دسته تقسیم می شوند:

1. اعمالی که داده ها را با برخی روشها (اضافه کردن، قالب بندی مجدد، انتخاب) دستکاری می کنند.
2. اعمالی که محاسبات را انجام می دهند .
3. اعمالی که شئ را برای وقوع رویدادهای کنترلی نمایش می دهند.

بطور مشابه شناسایی اعمال با پارس دوباره متن زبان طبیعی و مشخص کردن افعال یا عبارات فعلی انجام می گیرد، همچنین اعمال اضافی می تواند با درنظر گرفتن چرخه زندگی شئ در پیامهایی که با سایر اشیاء سیستم رد و بدل می کند ، معین شود. تعیین اعمال آخرین مرحله مشخص سازی شئ است . چرخهٔ زندگی یک شئ با مشخص کردن آنچه که موجب ایجاد شئ ، تغییر دادن، دستکاری کردن ، خواندن شئ و احتمالاً حذف شئ، معین می شود. در قسمت بعدی مدیریت پروژه نرم افزاری را برای پروژه- های شئ گرا را توضیح می دهیم.

چارچوب یک فرآیند معمولی برای پروژه های شئ گرا

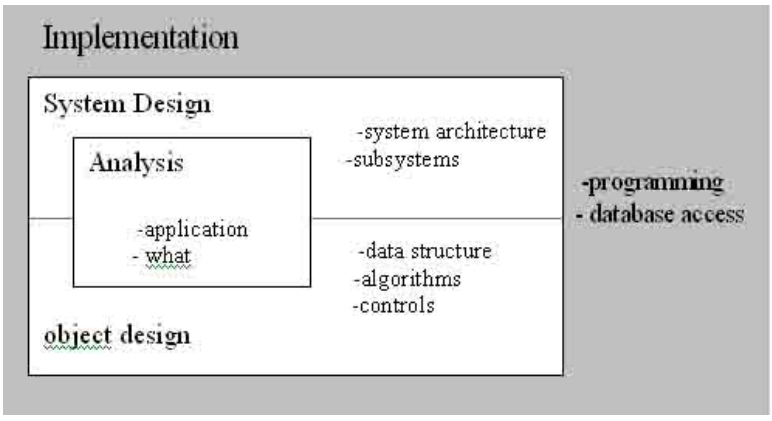
چارچوب یک فرآیند معمولی روشهایی سازمان یافته را برای مهندسی نرم افزار تعریف می کند. این روش ها به ساخت و نگهداری نرم افزار، وظایف، تفکیک مراحل مختلف توسعه (نقطه بررسی مرحله کاری پروژه) و محصولاتی که در هر مرحله لازم است، می پردازد. مهندس شئ گرا یک فرآیند تکراری را در توسعه نرم افزار طی می کند، به عبارت دیگر نرم افزار شئ گرا با طی کردن سیکلهایی کامل می شود و یک فرآیند توسعه شئ گرا باید ذات تکاملی داشته باشد تا آنجا که برخی از صاحبنظران در این زمینه یک مدل بازگشتی/ موازی را برای این کارپیشنهاد می کنندکه در اساس بصورت زیر عمل می کند:

1. تحلیل مسئله را تا جایی که کلاسهای اصلی مسئله و ارتباطات بین کلاسها مشخص شود، ادامه می دهیم .
2. با طراحی یک طرح اولیه تعیین می کنیم که آیا کلاسها و ارتباطات تعیین شده در مرحله قبل در عمل قابل پیاده سازی است یا نه؟
3. اشیائی که مجدداً می توانند مورد استفاده قرار بگیرند از کتابخانه اشیاء استخراج می- کنیم تا به وسیله آن یک الگوی سطح بالا از پروژه بسازیم.
4. طرح آزمایشهایی جهت کشف خطاهای الگو
5. گرفتن فیدبک از مشتری در رابطه با الگوی ساخته شده
6. تغییر و تحلیل دوباره مدل براساس آنچه که از الگو، طراحی و فیدبک مشتری عاید شده است.
7. اصلاح مدل طراحی جهت وفق دادن تغییرات.
8. تولید کد برای برخی از اشیاء (آنهایی که از قبل قابل دسترسی نیستند)
9. اسمبل کردن یک الگوی جدید با استفاده از اشیاء کتابخانه ای و اشیائی که اخیراً ساخته ایم.
10. طرح آزمایشات جهت کشف خطاها در الگوی جدید.
11. گرفتن فید بک از مشتری در رابطه با الگوی جدید.

با تجزیه کردن سیستم به مؤلفه های کاملاً مستقل، این کار را تا جایی که الگوی هر مؤلفه کامل شود، تکرار می کنیم. تکرار کردن فرآیند بازگشتی / موازی نیاز به برنامه ریزی، مهندسی(تحلیل ، طراحی ، استخراج کلاس، الگوسازی و تست کردن) و فعالیتهای تکاملی دارد.

سیکل توسعهٔ شئ گرا

چرخهٔ عمر توسعهٔ شئ گرا که در شکل 1-3 نشان داده شده است، شامل پیشرفت موازی نمایان ساختن اشیاء در سه فاز تحلیل، طراحی و پیاده سازی است که در مراحل اولیه توسعهٔ یک مدل انتزاعی بر پارامتر های خارجی- کیفیت سیستم کاربردی - تاکید دارد، با تغییر مدل بیشتر وارد جزئیات می شویم بطوریکه بیشتر به چگونگی ساخت سیستم و چگونگی عملکرد آن می اندیشیم - معماری ، ساختمان داده و الگوریتمها . سرانجام مانند هر سیستم اطلاعاتی، توسعه دهندهٔ سیستم باید به تولید کد و روتینهای دسترسی به پایگاه داده بپردازد.



فصل دوم - تحلیل و طراحی شئ گرا

فرآیند تحلیل و طراحی شئ گرا
در این فصل به روندکاری تحلیل و طراحی شئ گرای سیستم می پردازیم. در فاز تحلیل، مدلی از دنیای واقعی نرم افزار در حال توسعه که خواص مهم آن را نشان می- دهد، ساخته می شود. این مدل مفاهیم موجود در دامنه سیستم را بصورت انتزاعی نشان می دهد و بیانگر این است که سیستم چه کاری را باید انجام دهد و به چگونگی انجام (از دید فنی)آنها نمی پردازد. مدل تحلیلی رفتار عملی سیستم را مستقل از محیطی که نهایتا با آن در ارتباط خواهد بود، مشخص می کند. آنالیست باید زمانی را برای کشف نیازمندیهای سیستم صرف کند و مدل باید تمام این نیازها را پاسخگو باشد. توجه شود که ایجاد تغیرات در طول فاز تحلیل بسی آسانتر و با هزینة کمتری نسبت به فازهای بعدی قابل انجام است.

در فاز طراحی چگونگی برآورده کردن نیازهای کشف شده در مدل تحلیلی از مسئله، در محیط پیاده سازی بررسی می شود. Rumbaugh عملیات فاز طراحی را به دو مرحله تقسیم کرده است:

1. مرحله طراحی سیستم
2. مرحله طراحی شئ

طراح سیستم یک دید کلی از معماری سیستم را در نظر می گیرد و سیستم را در اجزایی کوچکتر که زیرسیستم نامیده می شود، سازماندهی می کند. این دید مبنای تصمیم گیری در شناسایی همروندیها، تخصیص فرآیندها به پردازه ها، دسترسی به منابع و... قرار می گیرد.
در طراحی شئ یک مدل طراحی با جزئیات پیاده سازی بیشتری ساخته می شود؛ مانند ساختار بندی دوبارة کلاسها برای بهبود کارایی، ساختمان داده های درونی ، پیاده سازی کنترل سیستم، الگوریتم پیاده سازی هر کلاس، پیاده سازی روابط و بسته بندی کردن در ماژولهای فیزیکی . برای مدل آنالیز شده، فاز طراحی با فاز پیاده سازی دنبال می شود که مدل طراحی شده به کد در زبان برنامه نویسی ترجمه شده و از DBMS برای پایگاه داده های موجود استفاده می کند.

تحلیل شیءگرا OOA
هنگامی که می خواهیم یک محصول یا سیستم جدید بسازیم چگونه آن را توصیف کنیم تا با تکنیک های مهندسی نرم افزار شئ گرا بتوانیم یک سیستم مطمئن تولید نماییم؟ آیا سوالهای خاصی در این زمینه وجود دارد که باید از مشتری بپرسیم ؟ اشیاءمربوط به سیستم کدامند؟ اشیاء موجود در سیستم چه ارتباطی باهم دارند ؟ چگونه مسائل را مشخص یا مدل کنیم تا یک طراحی موثر داشته باشیم؟

هرکدام از این سوالها در بطن تحلیل شئ گرا جواب داده می شود. تحلیل شی گرا اولین مرحله فنی است که به عنوان قسمتی از مهندسی نرم افزار شئ گرا انجام داده می شود. جهت ایجاد یک مدل تحلیلی ازسیستم اصول پایه ای زیر بکار برده می شود.

1. دامنه اطلاعات باید مدلسازی شود.
2. کارکرد سیستم توصیف شود.
3. رفتار سیستم ارائه گردد.
4. مدلهای داده ای، عملی و رفتاری تقسیم بندی شود تا جزئیات بیشتری از مسئله ارائه شود.
5. مدلهای اولیه ماهیت مسئله را نشان می دهند در حالیکه مدلهای پایانی جزئیات پیاده سازی را ارائه می کنند.
منظور ازتحلیل شیءگرا تعریف تمام کلاس های مربوط به مسئله ای است که باید جواب داده شود - اعمال و صفات متناظر با آنها ، رابطه بین این کلاسها و رفتاری که از خود نشان می دهند. جهت انجام این کار باید وظایف زیر انجام داده شود.

1. نیاز های اساسی کاربر که باید از طریق مصاحبه با کاربر به اطلاع مهندس نرم افزار برسد .
2. کلاسها باید شناسایی شوند.
3. سلسله مراتب کلاس باید مشخص شود.
4. رابطه شیء به شیء باید نشان داده شود.
5. رفتار شیء باید مدلسازی شود.
6. وظایف 1تا 5 باید آنقدر تکرار شود تا مدل تحلیلی کامل شود.

هدف تحلیل شیء گرا توسعه مدلی است که یک نرم افزار کامپیوتری را - جهت بیان نیاز های تعریف شده توسط کاربر - توصیف می کند.دردهه اخیر آقایانBooch,Raubaugh,Jacobson با ترکیب بهترین ویژگی های روش های شخصی و برخی از روشهای موجود تحلیل و طراحی شیءگرا روش Unified را معرفی کردند که بصورت گسترده ای در صنعت

استفاده می شود. در روش Unified از زبان مدلسازی یکپارچه که در فصل بعد به تفصیل خواهد آمد، استفاده می شود . این زبان به مهندس نرم افزار اجازه می دهد تا با استفاده از علائم مدلسازیی که به وسیله مجموعه ای از قواعد نحوی و معنایی کنترل می شود، یک مدل تحلیلی ازسیستم نشان دهد.

در UML با استفاده از پنج دید مستقل که سیستم را از چشم اندازهای مختلف توصیف می کنند، سیستم به نمایش در می آید. هر دید به وسیله مجموعه ای از نمودارها مشخص می گردد. این دیدگاه ها عبارتند از:

- دید مدل کاربر: دید مدل کاربر سیستم را از چشم انداز کاربر نمایش می دهد. مورد قابل کاربرد روشی برای مدلسازی این دیدگاه است . این نمایش تحلیلی، سناریو های مورد استفاده از چشم انداز کاربر نهایی را توصیف می کند.
- دید مدلسازی ساختاری: در این دید داده و عملکرد درونی سیستم نمایش داده می- شود که ساختار ایستای سیستم(کلاسها،اشیاء و روابط بین آنها) را مدلسازی می کند.
- دید مدل رفتاری: این قسمت از مدل تحلیلی، سیستم را از دید رفتاری بصورت پویا مدل می کند. این دید همچنین تعامل و همکاری ما بین عناصر مختلف ساختاری که در دو دید قبلی توصیف شد، را به تصویر می کشد.
- دید مدل پیاده سازی: این دید، چشم اندازهای رفتاری و ساختاری را آنگونه که باید ساخته شوند، نمایش می دهد.
- دید مدل محیط پیاده سازی: چشم اندازهای ساختاری و رفتاری محیطی که سیستم باید در آن پیاده سازی شود، ارائه می شود.بطور کلی مدلسازی تحلیلی UML روی دید های کاربر و ساختاری سیستم متمرکز می شود و مدلسازی فاز طراحی در UML شامل دید های رفتاری، پیاده سازی و محیط پیاده سازی می شود.

تحلیل مدل شئ گرا می تواند در سطوح مختلف انتزاعی انجام شود . مدل کردن کار تلاشی است جهت تعریف و شناسایی کلاسها ،اشیاء روابط و رفتارها که کل کار را مدل می کند. مدل شیء در سطح کار روند کاری سیستم تحت مطالعه را نشان می‌دهد، اما مدل شیء در سطح نرم افزار کاربردی روی نیازمندیهای مشتری ـ نیازمندیهایی که نحوهٔ ساخت سیستم را تحت تاثیر قرار می دهد- متمرکز می شود.

تحلیل دامنه نرم افزار که شامل شناسایی، تحلیل و مشخص سازی نیازهای عمومی در یک دامنه کاری خاص عموماً به منظور استفاده مجدد محصولات پروژه در دست ساخت، در پروژه های بعدی با دامنه کاری مشابه انجام می گیرد. بعبارت دیگر در تحلیل دامنه نرم افزار به دنبال الگو سازی یا استفاده از الگوهای موجود هستیم. در واقع مؤلفه هایی می سازیم که به طور گسترده در پروژه های دیگری نیز بتوانند مورد استفاده قرارگیرند.

اجزای کلی یک مدل آنالیز شده شئ گرا:
آنالیز شامل تقسیم دقیق، ساده، قابل فهم و درست مدلی است که از دنیای واقعی گرفته شده است. برای توسعه چنین مدلی از دنیای واقعی، مهندس نرم افزار باید علائمی را جهت نمایش اجزای کلی مدل تحلیلی شئ گرا انتخاب نماید. اجزای کلی مدل تحلیلی(مدلی که در فاز تحلیل ایجاد می شود) عبارتند از:

- یک دید ایستا از کلاسهای معنایی
- دید ایستا از صفات
- دید ایستا از روابط بین کلاسها
- دید ایستا از رفتار (این رفتار با تعریف یک ترتیب از اعمال ، تعیین می شود.)
- دید پویا از تعامل بین کلاسها و زیر سیستم ها
- دید پویا از کنترل زمانی سیستم

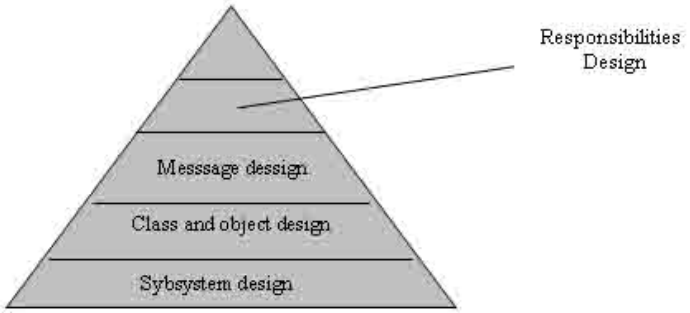
طراحی شئ گرا OOD
طراحی شئ گرا مدل ساخته شده در فاز تحلیل را به یک مدل در فاز طراحی تبدیل می کندکه این مدل به عنوان یک طرح اولیه برای ساخت نرم افزار استفاده می شود. برخلاف روشهای سنتی طراحی نرم افزار، طراحی شئ گرا دارای قابلیت پیمانه ای در سطوح مختلف می باشد. مؤلفه های مهم سیستم در زیر سیستم ها سازماندهی می شوند، پیمانه سطح سیستم ، داده ها و اعمال در اشیا کپسول می شوند - فرم پیمانه ای که بلوکی از سیستم شئ گرا را می سازد. علاوه بر این طراحی شئ گرا باید سازماندهی مناسبی را برای داده ها ی مربوط به صفات و همچنین جزئیات رویه های هر عمل را مشخص کند و بدین صورت قطعات مختلف داده و الگوریتم بعنوان قسمتی از پیمانه کلی معین می شود.

طبیعت منحصر به فرد طراحی شئ گرا در توانایی آن در طراحی نرم افزار بر اساس چهار مفهوم زیر می باشد:

- مجرد سازی(Abstraction)
- مخفی سازی اطلاعات (Information hiding)
- استقلال تابعی(Functional independency)

- پیمانه ای (Modularity)

بطور کلی فعالیتهای ساخت یک سیستم شئ گرا عبارتند از: طراحی شئ گرا، برنامه نویسی شئ گرا و تست شئ گرا. در شکل 2-1 یک هرم چهار لایه ای برای طراحی شئ گرا نشان داده شده است.



شکل ۲-۱ : لایه های طراحی شئ گرا

لایه های طراحی شئ گرا

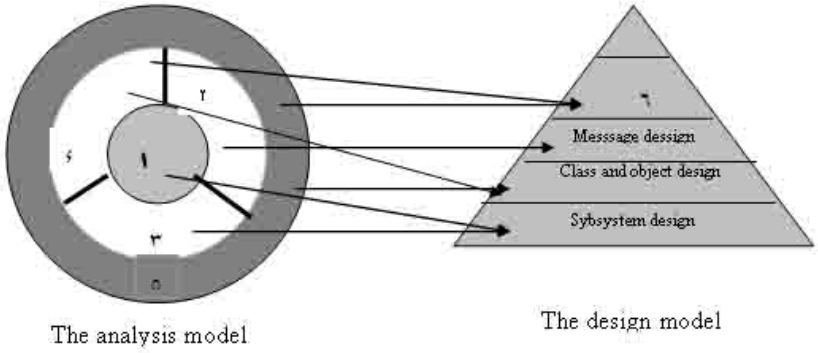
لایة زیر سیستم(subsystem layer): این لایه هر یک از زیر سیستم ها را نشان می دهد که نرم افزار را قادر به برآوردن نیازهای تعریف شده - توسط کاربر - می نماید و همچنین با استفاده از این لایه نرم افزار، فراساختارهای تکنیکی که نیازهای کاربر را پشتیبانی می کند،پیاده سازی می نماید.

لایة شئ و کلاس(class and object layer): این شامل کلاسهای سلسله مراتبی است که سیستم را قادر به استفاده از تعمیم و تخصیص می سازد، این لایه همچنین شامل نمایشی از هر شیء است.

لایه پیام(message layer): لایة پیام شامل جزئیات طراحی است و شئ را قادر می- سازد تا با دیگر اشیاء تعامل داشته باشد. این لایه بر واسط های نرم افزاری داخلی و خارجی سیستم استوار است.

لایة مسؤلیتها (responsibility layer): این لایه شامل طراحی داده ساختارها و الگوریتم ها برای تمام صفات و اعمال موجود در هر شئ می باشد.

شکل 2-2 رابطه بین مدل تحلیلی شی گرا و مدل طراحی که از آن مشتق شده را نشان می دهد. طراحی زیرسیستم از توصیف نیازهای کلی مشتری در قالب موارد قابل کاربرد، رویدادها و حالاتی که از دید یک بیننده غیر تکنیکی که توسط مدلهای رفتاری - برای اطلاعات بیشتر در مورد نمودار های رفتار و موارد قابل کاربرد به فصل چهارم مراجعه شود - تصویر شده ، اشتقاق می شود. طراحی کلاس و اشیاء از توصیف صفات، اعمال و همکاری های موجود در مدل CRC نگاشت می شود، طراحی پیامها از مدل رابطه بین اشیاء و سرانجام طراحی مسؤلیتهای شئ ازصفات، اعمال و همکاری های توصیف شده در مدل CRC مشتق می شود.



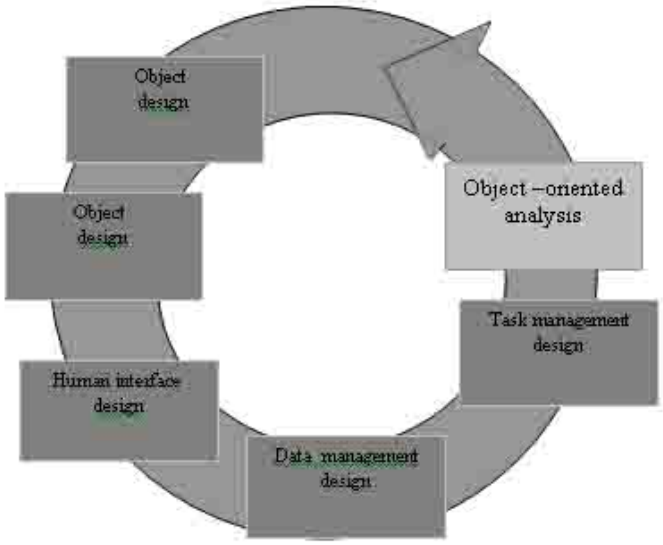
شکل ۲-۲: ترجمهٔ مدل تحلیلی به یک مدل طراحی که نواحی ٦ گانه در این شکل بصورت زیر است

1- use case 2 - Object_relationship model 3- Object_behavior model
4- CRC index cards 5-Attributes, operations, collaborators 6-responsibilities design

روش Unifiedدر طراحی شیءگرا

در طول فاز مدلسازی تحلیلی، دیدهای مدل کاربر و مدل ساختاری ارائه می شوند. این مدلها بینشی از سیستم را در قالب سناریو های کاربردی جهت هدایت مدلسازی رفتاری سیستم ارائه می دهند و همچنین اصولی را برای دیدهای پیاده سازی و محیط پیاده سازی با شناسایی و توصیف عناصر ایستای ساختاری از سیستم، فراهم می کنند. همچنانکه در بالا اشاره شد UML نیز فعالیتهای لازم در فاز طراحی را به دو دسته کلی تقسیم می کند که شامل طراحی سیستم و طراحی شئ بود.

طراحی سیستم معماری نرم افزار را نشان می دهد درحالیکه طراحی شی روی تعریف اشیاء و تعامل آن با اشیاء دیگر متمرکز می شود. جزئیات مشخص سازی داده ساختارهای صفات و طراحی رویه های مربوط به اعمال در طول فاز طراحی شیء انجام می شود. در این فاز دید برای هر یک از صفات کلاس ها و واسط بین اشیاء جهت تعیین جزئیات یک مدل کامل پیام بطور مفصل بررسی می شود.
طراحی شئ و سیستم در UML به منظور توصیف طراحی واسط های کاربری ، مدیریت داده ها برای سیستمی که باید ساخته شود و مدیریت وظایف زیر سیستم های مشخص شده ، توسعه داده شده است. فرآیند طراحی واسط های کاربری با توجه به دید مدل کاربرتهیه می شود. طراحی مدیریت داده برمجموعه ای از کلاسها و همکاریهایی استوار است که به سیستم اجازه مدیریت داده های ماندگار می دهند. طراحی مدیریت وظایف با ایجاد فراساختارهایی انجام می شود که وظایف زیر سیستم ها را سازماندهی و همروندی وظایف را کنترل می نماید. شکل 2-3 جریان کار را در فاز طراحی نشان میدهد.



شکل ۲-۳: جریان پروسه در OOD

در فرآیند طر احی با UML، دید های مدل کاربر و مدل ساختاری حاصل از تحلیل سیستم به تفصیل بررسی شده و نمایشی پیرامون این دیدها بطور دقیق در فاز طراحی ارائه می شود.

فرآیند طراحی سیستم
طراحی سیستم جزئیات معماری مورد نیاز برای ساختن سیستم را توضیح می دهد، این فرآیند شامل فعالیت های زیر است:

- تقسیم مدل تحلیلی به زیر سیستمها
- شناسایی همروندهایی که درتعریف مسئله قید شده است
- تخصیص زیر سیستم ها به پردازه ها و وظایف
- توسعه طرح برای واسط کاربری
- انتخاب یک استراتژی اساسی برای پیاده سازی مدیریت داده ها
- شناسایی منابع عمومی و درنظر گرفتن یک مکانیزم کنترلی برای دسترسی به آنها
- طراحی یک مکانیزم کنترلی مناسب برای سیستم برای مدیریت وظایف
- تعیین چگونگی برخورد با شرایط مرزی

درزیر روال طراحی هریک از مراحل فوق الذکر را با جزئیات بیشتری می آوریم.

افراز مدل تحلیلی
در طراحی سیستم شئ گرا، افرازی از مدل تحلیلی انجام می دهیم بطوریکه مدل را به مجموعه هایی از کلاس های مرتبط- از نظر تعامل با هم و رفتار- افراز می کنیم. هر یک از این مجموعه ها یک زیر سیستم بشمار می روند. بطور کلی تمام عناصر زیر سیستم برخی از ویژگی ها را به بصورت مشترک در خود دارند. همه این عناصر ممکن است در انجام یک عمل دخیل باشند، یا اینکه همه در یک محصول سخت افزاری مقیم باشند ویا ممکن است همه عناصر، یک کلاس ازمنابع را مدیریت کنند. زیرسیستم ها به وسیله مسؤلیتهایشان مشخص می شوند، یعنی یک زیر سیستم را می- توان بوسیله سرویسهایی که فراهم می کند، شناسایی کرد. در طراحی سیستم شئ گرا، سرویس به معنای جمعی از اعمال است که یک عمل مشخص را انجام می دهند(مانند مدیریت یک پردازشگر لغت، تبدیل یک سیگنال آنالوگ به دیجیتال و...) .
در این مرحله زیر سیستم هایی که تعریف و طراحی شده اند باید با معیارهای طراحی زیر مطابقت داشته باشند.

1. زیر سیستم باید واسط کاربری تعریف شده ای داشته باشد تا ارتباط با بقیه سیستم بتواند از طریق این واسط انجام شود.
2. بجز تعداد کمی از کلاس های ارتباطی بقیة کلاس های زیرسیستم فقط می- توانند باکلاس های آن زیر سیستم همکاری داشته باشند.
3. باید تلاش شود که تعداد زیر سیستمها کمترین مقدار ممکن داشته باشند.
4. یک زیر سیستم می تواند بمنظور کاهش پیچیدگی به زیر سیستم های داخلی افراز شود.

وقتی دو زیر سیستم با هم در ارتباط باشند آنها می توانند رابطه سرویس دهنده /مشتری یا اتصال نقطه به نقطه باهم داشته باشند. درحالت سرویس دهنده / مشتری هر سیستم در یکی از این دو نقش - سرویس دهنده یا مشتری - عمل می کند و جریان ارائه سرویس یکطرفه و از سرویس دهنده به مشتری می باشد. درحالت نقطه به نقطه جریان ارائه سیستم ممکن است دو طرفه باشد.

هنگامی که سیستم به زیر سیستمها افراز می شود، یک عمل طراحی دیگر به نام لایه بندی صورت می گیرد. هر لایه از سیستم شئ گرا شامل یک یا چند زیر سیتسم است، این لایه ها سطوح مختلف مجرد سازی یا عملیاتی مورد نیاز برای انجام دادن عمل زیر سیستم را نشان می دهند. غالباً سطوح مجرد سازی بوسیله میزان قابل رویت بودن عملکرد زیر سیستم از نظر کاربر نهایی تعیین می شود.

بعنوان مثال یک معماری چهار لایه ای ممکن است بصورت زیر باشد:

1. لایه ارائه presentation layer (زیر سیستم های متناظر با واسط کاربری)
2. لایه کاربردی application layer (زیر سیستم هایی که پردازش های مربوط به برنامه کاربردی را انجام می دهند)
3. لایه قالب بندی داده ها data formatting layer (زیر سیستم هایی که داده را برای پردازش آماده می کنند.)
4. لایه پایگاه داده ای data base layer (زیرسیستم های مربوط به مدیریت دادها)

بطور کلی روش زیر برای لایه بندی توصیه می شود:

1. مشخص نمودن معیار های لایه بندی : این معیار ها تعین می کنند که چگونه زیر سیستم ها در یک لایه در معماری لایه ای گروهبندی شوند
2. تعین تعداد لایه ها :تعداد خیلی زیاد لایه ها (بیشتر از حدلازم) لزوماً از پیچیدگی سیستم نمی کاهد از طرف دیگر تعداد خیلی کم لایه ها ممکن است به استقلال عملیاتی زیر سیستم ها صدمه وارد کند.
3. نامگذاری لایه ها و تخصیص زیر سیستم ها (بهمراه کلاس های کپسوله شده)به لایه: باید توجه شود که تعامل بین زیر سیستم ها (کلاس ها) از یک لایه با زیر سیستم ها (کلاس ها) از لایه دیگر منطبق بر معماری مورد نظر باشد بطوریکه در یک معماری بسته پیام ها از یک لایه ممکن است فقط به لایه زیرین- لایه ای که بلافاصله در زیر لایة مورد نظر باشد- ارسال شود اما در یک معماری باز پیام ها به تمام لایه ها در سطوح پایینتر قابل ارسال است.
4. طراحی واسط برای هر لایه
5. بررسی زیر سیستمها به منظور بهینه کردن ساختار کلاس های هر لایه
6. تعریف یک مدل پیام رسانی برای تعامل لایه ها
7. بازنگری طراحی لایه جهت تامین کردن کمترین میزان وابستگی بین لایه ها
8. تکرار کردن این مراحل جهت بهینه کردن لایه بندی .(تصفیه کردن معماری لایه ای)

همروندی و تخصیص زیرسیستم

بررسی مدل رفتار شئ از نظر پویایی از وجود نوعی همروندی میان کلاس ها (یا زیر سیستم ها) نشان می دهد . اگر چند کلاس (یا زیر سیستم) در یک زمان فعال نباشند، نیازی به پردازش همروند درسیستم نیست. و این بدین معناست که کلاس ها (یا زیر سیستم ها) را می توان بر روی یک پردازنده سخت افزاری پیاده سازی کرد. از طرف دیگر اگر کلاسها (ویا زیر سیستمها) باید به صورت آسنکرون بر روی رویدادها دریک زمان عمل کنند، همروندی وجود دارد. هنگامی که زیر سیستم ها بصورت همروند فعال می شوند دو انتخاب تخصیصی وجود دارد: (1) تخصیص هر زیر سیستم به یک پردازنده مستقل (2) تخصیص زیر سیستم ها به یک پردازنده مشترک و فراهم نمودن ملزومات همروندی با استفاده از خواص سیستم عامل مورد استفاده.

همروندی وظایف با بررسی نمودار حالت هریک از اشیاء تعیین می شود. اگر جریان رویدادها و تراکنش ها نشان دهد که درهر زمان فقط یک شئ فعال است، یک بند کنترلی برقرار می شود یعنی هرگاه سیستم این محدودیت را داشته باشیم که وقتی یک شئ برای شیئی دیگر پیامی ارسال کرد، شئ اولی الزاماً منتظر جواب بماند، بند کنترلی بصورت پیوسته ادامه می یابد و در این صورت همروندی نداریم. اما اگر شئ اولی بعد از ارسال پیام به فعالیت خود ادامه دهد، بند کنترلی تقسیم می شود و همروندی بوجود می آید. برای تعیین اینکه کدام یک از دو انتخاب تخصیصی پردازنده برای سیستم موجود مناسب است، طراح باید نیازها ی کارایی، هزینه ها و سربار پردازش تحمیلی برای تعامل بین پردازنده ها را بررسی کند.

مؤلفه مدیریت وظیفه Task management component

غالب ویژگیهای یک وظیفه با معین نمودن چگونگی شروع آن وظیفه تعیین می شود، غالباً وظایف بصورت رویدادگرا یا زمانگرا (وظیفه در زمان های خاص شروع می شود)هستند. هردو نوع وظیفه بوسیله وقفه فعال می شوند اما نوع اول با دریافت یک پیام وقفه از منابع خارجی (مانند پردازندة دیگر ، حسگر) فعال می شود. ولی نوع دوم بوسیله ساعت سیستم سازماندهی می شود.

علاوه برچگونگی راه اندازی وظایف باید اولویت وظایف نیز تعیین شود. وظایف بااولویت بالا باید بلافاصله به منابع دسترسی پیدا کنند.برای طراحی اشیائی که وظایف همروند را مدیریت می کنند استراتژی زیر پیشنهاد می شود.

- تعیین ویژگیهای وظیفه
- تعریف وظیفه ناظر و اشیاء متناظر با آن
- مجتمع سازی وظیفه ناظر و وظایف دیگر.

مؤلفه واسط کاربری user interface component

اگر چه مؤلفه واسط کاربری در بطن دامنة مسئله پیاده سازی می شود، اما خود واسط زیر سیستمی خیلی مهم برای نرم افزارهای کاربردی مدرن است. مدل تحلیلی شامل سناریوهایی کاربردی (موارد قابل کاربرد) و مشخصات نقش هایی است که کاربران (کنشگران) درآن نقش ها با سیستم تعامل دارند و این همان اطلاعات ورودی فرآیند طراحی واسط کاربری است .

بمحض این که کنشگر و سناریوهای مربوطه تعریف شد، سلسله مراتب دستورات مشخص می شود. سلسله مراتب دستورات قسمت های مهم منوی سیستم و زیر تابع های موجود در رابطه با هر قسمت منو را بیان می کند.

بدلیل وجود محیط های متنوع توسعه واسط کاربری، نیازی به طراحی عناصر گرافیکی نیست .کلاس های با قابلیت استفاده مجدد (با صفات و اعمال مناسب) برای پنجره ها ،آیکن ها ، عملکرد ماوس و تابع های تعاملی متنوع دیگر ، دراین محیطها در دسترس هستند. شخص پیاده ساز فقط باید ازکلاسهایی که ویژگی های مناسب را برای دامنة مسئله دارند، نمونه سازی کند.

مؤلفه مدیریت داده Data management component

مؤلفة مدیریت داده شامل دو قسمت جدا ازهم می باشد: (1)مدیریت کردن داده ها که قسمت بحرانی نرم افزار کاربردی هستند و(

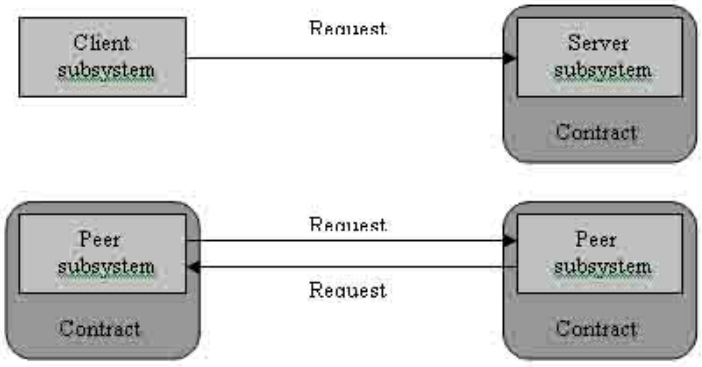
2) ساختن فراساختاری برای ذخیره و بازیابی اشیاء. بطور کلی مدیریت داده به صورت لایه ای طراحی می شود و ایدة آن این است که نیازهای سطح پایین برای دستکاری ساختارهای داده را از نیازهای سطح بالا برای دستگیری از صفات سیستم جدا نماییم. این کار در عمل با استفاده از یک DBMS برای تمام زیر سیستمها انجام می شود، اشیائی که برای دستکاری پایگاه داده لازم است عضو همان کلاسهایی است که در هنگام تحلیل دامنه مشخص شده اند.

مولفه مدیریت منابع Recourse management component

در سیستم های شئ گرا منابع متنوعی وجود دارد که در بسیاری از موارد زیر سیستم ها بطور همزمان در دسترسی به این منابع رقابت می کنند. منابع عمومی سیستم می تواند موجودیتها ی خارجی مانند دیسک گردان ، پردازنده ، خطوط ارتباطی ، یا مجرداتی مانند پایگاه داده یا شئ باشد. بدون درنظر گرفتن نوع منبع، مهندس نرم افزار باید یک مکانیزم کنترلی برای این منابع طراحی نماید، پیشنهاد Rambaughبرای این مکانیزم این است که هر منبع باید با یک شئ محافظ (نگهبان) تصاحب شود، این شئ محافظ به عنوان یک دربان دسترسی به منبع را کنترل و از تصادم درخواست ها جلوگیری می- کند.

ارتباط بین زیر سیستم ها Intersubsystem communication

پس ازاینکه هر زیر سیستم مشخص شد، لازم است که همکاریهای موجود میان زیرسیستمها بصورت سیستماتیک تعریف شود. بصورت کلی مدلی که برای همکاری شئ- به- شئ استفاده می شود را می توان برای زیرسیستمها نیز توسعه داد. هنچنانکه قبلاً ذکر شد، ارتباط می تواند بصورت اتصال سرویس دهنده/ مشتری یا نقطه به نقطه باشد. شکل 2-4 این ارتباط را نشان می دهد، باتوجه به شکل باید قراردادهایی بین زیرسیستمها مشخص شود.قرارداد نشان می دهد که چگونه یک زیرسیستم می تواند با یک زیر سیستم دیگر تعامل داشته باشد.



شکل ۲-٤: مدل همکاری بین دو زیر سیستم

فرآیند طراحی شئ

در مرحله طراحی سیستم، معماری نرم افزار مشخص می شود و طرح هایی که برای سیستم در نظر گرفته می شود، طرح هایی کلی می باشند. حال وقت آن است که به جزئیات بپردازیم و برای این روی طراحی شئ متمرکز می شویم دراین مرحله با اصول و مفاهیم در سطح طراحی مؤلفه سرو کار داریم، داده ساختارهای محلی (برای صفات) تعریف می شوند و الگوریتم ها (برای اعمال) طراحی می شوند.

مشخصات شئ

طراحی مشخصات شئ به یکی از دو شکل زیر صورت می گیرد:
توصیف پروتکل که واسطی را برای شئ با تعریف هر یک از پیام هایی که شئ قادر به دریافت آنهاست و عملی که شئ در نتیجة دریافت یک پیام خاص انجام می دهد، برقرار می کند .
توصیف پیاده سازی که جزئیات پیاده سازی را برای هر یک از اعمال مربوط به پیام دریافتی را نشان می دهد. جزئیات پیاده سازی شامل بخش اطلاعات خصوصی شئ است، اما جزئیات داخلی مربوط است به داده ساختاری که صفات را بیان می کند و جزئیات رویه ای که اعمال را توصیف می کند. توصیف پروتکل چیزی جز مجمو عه ای از پیام ها و توضیحات مر بوط به هر پیام نمی باشد. برای سیستم های بزرگ با پیام- های زیاد می توان پیامها را طبقه بندی کرد. توصیف پیاده سازی شئ جزئیات داخلی (مخفی) مورد نیاز برای پیاده سازی را فراهم می کند اما این جزئیات برای درخواست لازم نیست. طراح شئ باید توصیف پیاده سازی را فراهم کند لذا باید جزئیات داخلی شئ را بررسی کند، اما طراح دیگر یا پیاده ساز که از شئ یا نمونه های دیگر آن استفاده می کنند فقط به توصیف پروتکل نیاز دارند و به توصیف پیاده سازی هیچ نیازی نیست.

توصیف پیاده سازی ترکیبی از اطلاعات زیر است؛

1. مشخص سازی نام شئ و ارجاع آن به یک کلاس.
2. مشخص سازی ساختمان دادة خصوصی که به داده ها و انواع اشاره دارد.
3. توصیف رویه هر عمل.

توصیف پیاده سازی باید اطلاعات کافی را برای دستگیری از تمام پیام های موجود در توصیف پروتکل، داشته باشد.

طراحی الگوریتمها و ساختمان داده ها
نمایش های متنوع در مدل تحلیلی و طراحی سیستم مشخصات تمامی اعمال و صفات را فراهم می کند، الگوریتم برای پیاده سازی مشخصات عمل ساخته می شود. در بسیاری از موارد الگوریتم شامل یک سری محاسبات ساده یا یک توالی رویه ای است که قابل پیاده سازی در یک ماژول نرم افزاری می باشد. اما اگر عمل دارای توصیف پیچیده ای باشد لازم است که عمل را بصورت پیمانه ای پیاده سازی کنیم.
ساختمان دادها بصورت همروند با الگوریتم ها طراحی می شوند. از آنجا که اعمال بطور یکنواخت و ثابت، صفات کلاس را دستکاری می کنند طراحی ساختمان داده ای که بتواند صفات را منعکس کند باید قویاً با طراحی الگوریتم مرتبط باشد.

مؤلفه های برنامه و واسط ها
یک نمود مهم در کیفیت طراحی نرم افزار پیمانه ای بودن آن است، بطوری که شامل مشخصات مؤلفه ای برنامه (پیمانه) است که برای تشکیل یک برنامهٔ کامل با هم ترکیب می شوند، روش شئ گرا، شئ را به عنوان یک مؤلفه برنامه تعریف می کنند که می- تواند به مؤلفه های دیگر متصل شود. اما تعریف شئ و اعمال کافی نیست، در طول فاز طراحی باید واسط های بین اشیاء و ساختاری کلی را برای اشیاء فراهم کنیم. اگرچه مؤلفهٔ برنامه طراحی مجردات است ولی این مؤلفه ها باید در یک زبان برنامه نویسی که برای پیاده سازی استفاده می شود ارائه شود.

طراحی الگو
طراحان خوب در هر زمینه دارای توانایی بالا در درک مسئله و تشخیص الگوهای مربوطه -که با ترکیب آنها می توان مشکل را حل نمود- می باشند. در طول فرآیند طراحی شئ گرا مهندس نرم افزار باید موقعیت هایی را جستجو کند که الگو های طراحی شده ای برای استفاده مجدد موجود باشد.

توصیف طراحی الگو
تمام طراحی الگوها را می توان بامشخص کردن اطلاعات زیر توصیف کرد:

1. نام الگو
2. هدف الگو
3. طراحی مواردی که الگو را تحریک می کنند
4. پاسخ های مربوط به هر محرک
5. کلاس های لازم برای پیاده سازی جواب
6. مسؤلیتها و همکاریهای بین کلاس های جواب
7. راهنمایی هایی جهت پیاده سازی مؤثر
8. کد منبع نمونه یا قالبهای کد منبع
9. مراجعه متقابل الگوهای طراحی مرتبط

نام الگو یک انتزاع است که معنی ویژه ای را برای درک هدف و توانایی الگو القا می- کند. طراحی محرک ها نیازمندیهای داده ای، عملی یا رفتاری متناظر با قسمت خاصی از نرم افزار را بیان می کنند که الگو در آن قسمت بکار برده می شود. در اساس این موارد محیط و شرایطی که در آن بتوان از الگو استفاده کرد، بیان می کنند.

استفاده از الگو در طراحی
در سیستم شئ گرا با بکار بردن دو مکانیزم متفاوت می توان از الگو استفاده کرد. این دو مکانیزم عبارتند از: ارث بری و ترکیب
با استفاده از ارث بری الگوی طراحی موجود یک قالب برای زیر کلاس جدید محسوب می شود. صفات و اعمال موجود در الگو قسمتی از زیر کلاس خواهد شد. ترکیب مفهومی است که منجر به تجمع اشیاء می شود، آن وقتی است که ممکن است مسئله به اشیایی با عملیات پیچیده احتیاج داشته باشد. شئ پیچیده را می توان با انتخاب مجموعه ای از الگوهای طراحی و ترکیب اشیاء مناسب (ویا زیر سیستم) اسمبل کرد. با هر الگوی طراحی می توان بصورت یک جعبه سیاه رفتار نمود و ارتباط میان الگوها فقط از طریق واسط های تعریف شده صورت می گیرد.

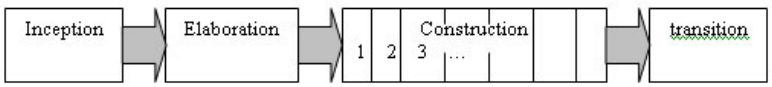
فصل سوم

فرآیند توسعه
همانطور که در فصل اول اشاره کردیم برای تولید نرم افزار و توسعه آن متدلوژیهای مختلفی وجود دارد و متدلوژی را به عنوان یک فرآیند ساختارمند برای حل یک مسئله که بوسیله ابزارها و تکنیک هایی پشتیبانی می شود، تعریف کردیم. دراین فصل درباره فرآیند توسعه نرم افزار با جزئیات بیشتری بحث می کنیم.

فرآیند توسعه نرم افزار چیست؟
فرآیند مشخص می کند که چه کسی ، چه کاری را درچه موقعی و چگونه انجام دهد، تا به یک هدف خاص برسیم(who?, what?, when? And how) . در مهندسی نرم افزار منظور از این هدف خاص، ساخت یک نرم افزار جدید یا بهبود یک نرم افزار موجود می باشد. یک فرآیند موثر مسیری را برای توسعه کارآمد نرم افزار فراهم می نماید، این فرآیند با توجه به شرایط موجود در سیستم تحت مطالعه نمونه های عملی متناسب را ارائه می دهد که بهترین راهکارهای موجود را برای شرایط جاری پیشنهاد می کند. در واقع فرآیند با ارائه راهنماییها و نمونه های عملی توسعه نرم افزار میزان ریسک را کاهش می دهد و به توانایی توسعه دهنده برای پیشگویی آینده پروژه می افزاید و بطور کلی یک دید و فرهنگ عمومی را برای توسعه نرم افزارهای مختلف ترویج می دهد.
ما به فرآیندی نیاز داریم که همه افراد دخیل در توسعه نرم افزار را هدایت کند - مشتریان ،کاربران،توسعه دهندگان و مدیران اجرایی. بنابراین فرآیند مذکور باید بصورت گسترده دردسترس باشند تا همه افراد ذینفع نقش آن را در توسعه نرم افزار درک کنند. نهایتاً فرآیند باید با توجه به تکنولوژی، ابزارها و ساختارهای سازمانی قابل تغییرباشد.

ابزارها(Tools): فرآیندها و ابزارها باید به صورت موازی گسترش یابند.
افراد: افراد برای کار با فرآیند نیاز به مهارت کمی داشته باشند و یا مهارت مورد نیاز را به سرعت و سهولت کسب کنند.
ساختارهای سازمانی : ساختارهای سازمانی در نحوه طراحی موثراست و باید فرآیند آن را درنظر بگیرد. مثلا توزیع تیم کاری در سطوح مختلف سازمانی ، کارمند پاره وقت بودن یا عامل قرارداد بودن .

با توجه به نوع پروژه می توان فرآیند متناسب را انتخاب نمود - برای یک پروژه می توان از چند فرآیند مختلف نیز استفاده کرد - و برای نشان دادن نتایج تحلیل و طراحی در هر فرآیند می توان از UML استفاده کرد. عوامل مختلفی در توسعه نرم افزار بر فرآیندهای گوناگون اثر می گذارند، که شامل نوع نرم افزاری است که باید توسعه داده شود (بلادرنگ، سیستم اطلاعاتی یا یک محصول رومیزی) ، اندازه (یک توسعه دهنده، گروه کوچک ،گروه بیش از 100 نفر) و غیره .
درواقع تیم توسعه دهنده باید با توجه به پروژه برای توسعه یک فرآیند خاص را با استفاده از فرآیند های موجود- به عنوان توصیه های مهم نه به عنوان یک فرآیند استاندارد که حتما باید آنرا دنبال کرد- گسترش دهند. شکل 3-1 یک دید سطح بالا را از فرآیند توسعه نشان می دهد. این فرآیند، یک فرآیند توسعه تکراری - افزایشی است. که درآن محصول نهایی پس از طی نمودن چندین مرحله - هر مرحله یک حالت میانی از محصول را تحویل مرحله بعد می دهد بطوری که خروجی هر مرحله از مرحله قبل کاملتراست - تولید می شود.مرحله ساخت دارای تکرارهای بیشتری است که در هر



شکل ۳-۱: یک دید سطح بالا از فرآیند توسعه

تکرار یک ویژگی از نرم افزار تکمیل شده سپس تست و مجتمع سازی انجام می گیرد و زیر مجموعه ای از نیازهای پروژه را برآورده می شود . هرتکرار شامل تمام فازهای چرخهٔ عمر یعنی تحلیل، طراحی، پیاده سازی و تست است.
مرحله اول مرحله شروع است، دراین مرحله یک امکان سنجی و بررسی مقدماتی سیستم و تعیین نیازمندیهای کلی کاربر برای تعیین محدوده سیستم انجام می شود ومجوز ادامه کار را از کاربر می گیریم .
مرحله بعدی مرحله تفصیل است، دراین مرحله تمام نیازمندیهای کاربر را جمع آوری می کنیم و تحلیل و طراحی سطح بالایی را برای ایجاد خطوط پایه معماری سیستم انجام می دهیم. سپس برای مرحله ساخت برنامه ریزی می کنیم . با این نوع فرآیند ها- فرآیندها ی تکراری - بعضی از کارها باید در آخرین مرحله یعنی مرحله انتقال انجام گیرد. مانند: بتا تست، آموزش کاربرو...
در شکل بالا فقط مرحله ساخت بصورت تکراری نشان داده شده است . درواقع در همه مراحل می توانیم تکرار داشته باشیم و این کار برای مراحل بزرگ خیلی مفید است. تا اینجا دید سطح بالایی از فرآیند توسعه نشان دادیم در زیر جزئیات کار را بررسی خواهیم کرد.

شروع Inception
مرحله شروع با توجه به اهمیت و اندازه پروژه به شکل های مختلف انجام می گیرد. برای پروژه های کوچک و غیر رسمی ممکن است مرحله شروع ظرف چند ساعت گفتگو با مشتری و طرح یک نقشه کلی که پروژه را تصویر می کند، انجام شود اما در پروژه های با اهمیت و بزرگ با برگزاری جلسات متعدد رسمی و قراردادهای امضاء شده رسمی این مرحله طی می شود. در این حالت ممکن است مرحلهٔ شروع ماهها طول بکشد. در طول فاز شروع به امکان سنجی می پردازیم و موارد کاری پروژه - ارزش تقریبی پروژه، زمان لازم برای پروژه و ... - را مشخص می کنیم و اساس کار سیستم و محدوده سیستم تعیین می شود و ممکن است برای انجام این کارها به یک تحلیل مقدماتی نیاز داشته باشیم در پایان پس از توافق با مشتری قرارداد منعقد می شود و رسماً باید کار را شروع کنیم.

تفصیل Elabboration
بعد از اینکه مجوز ادامه کار را از مسؤل پروژه گرفتیم، در این مرحله ما تنها لیستی ناقص از نیازهای کلی سیستم که مبهم و غیر گویا است، داریم در اینجا می خواهیم درک بهتری از مسئله داشته باشیم و بررسی می کنیم که: (1) واقعاً می خواهیم چه چیزی بسازیم؟ (2) چگونه سیستم موردنظر را بسازیم؟ درواقع دراین مرحله نیازهای کاربر بصورت دقیق، با جزئیات کامل در قالب موارد قابل کاربرد شناسایی می شود. این مرحله غالباً با ریسک همراه است، انواع ریسک های ممکن به چهار دسته تقسیم می- شوند:
ریسک نیازمندیها : نیازمندیهای سیستم کدامند؟ در تولید نرم افزار بدترین حالت هنگامی است که سیستم نادرستی تولید شود یعنی تفسیر نادرستی از نیازهای کاربر را در ساخت سیستم ملاک قرارداده ایم و سیستم تولید شده نیازهای کاربر را برآورده نمی‌کند(درک نکردن سیستم مورد نیاز).

ریسک فنی: آیا تکنولوژیی که برای ساخت نرم افزار انتخاب شده واقعاً کار مورد نظر را انجام می دهد؟
ریسک مهارتها: آیا تخصص های لازم را برای انجام کار داریم؟
ریسک سیاسی: آیا انجام پروژه از نظر سیاسی مشکل ساز نیست؟
در زیر ریسک نیازمندیها به تفصیل بررسی می شود .

ریسک نیازمندیها
بررسی نیازمندیها در UML با موارد قابل کاربرد شروع می شود، کل فرآیند توسعه ازموارد قابل کاربرد مشتق می شود. مورد قابل کاربرد یک تعامل نوعی بین کاربر و سیستم برای دسترسی به یک هدف خاص می باشد. هرمورد قابل کاربرد یک عمل با ارزش را برای کاربر نشان می دهد. موارد قابل کاربرد عناصر پایه ای برای ارتباط مشتری با توسعه دهنده ، در طول برنامه ریزی پروژه است . یکی از مهم ترین کارهایی که در طول فاز تفصیل باید انجام بگیرد کشف تمام موارد قابل کاربرد بالقوه در سیستم در دست ساخت است. البته در عمل کشف همه موارد به یکباره غیر ممکن است اما در هر صورت باید موارد قابل کاربرد مهم که مواردی کلیدی هستند و ریسک را کاهش می دهند ، مشخص شوند و بهمین خاطر است که در طول فاز تفصیل باید نتایج میانی را به منظور جمع آوری بیشتر موارد قابل کاربرد با مشتری در میان بگذاریم . لازم نیست که موارد قابل کاربرد را بصورت خیلی ریز تشریح نماییم ، توصیف موارد قابل کاربرد در یک پاراگراف بطوریکه بروشنی ایده های اساسی را به کاربر نشان دهد و برای توسعه دهنده نیز کارهای پشت صحنه نمایان سازد،کافی است.

موارد قابل کاربرد کل سیستم را منعکس نمی کنند یعنی اسکلتِ یک مدل مفهومی از دامنه مسئله را نشان می دهند. بطور کلی برای بررسی و کشف دقیق نیازمندیها باید به مدل سازی دست بزنیم و با استفاده از مدل سازی دامنهٔ مسئله را تصویر کنیم .
مدلهای مختلفی وجود دارند که ماهیت سیستم مورد مطالعه را به روشنی نشان می دهند که تعدادی ازآنها عبارتند از(1) مدلها ی دامنهٔ مسئله و موارد قابل کاربرد، (2) مدلهای تحلیلی و (3) مدلهای طراحی که در فصل قبل معرفی شد. در UML برای ایجاد مدلهای دامنهٔ مسئله استفاده از نمودار های زیر پیشنهاد می شود:

- نمودارهای کلاس از چشم انداز مفهومی
- نمودارهای فعالیت ؛ این نمودارها در فصل بعد به تفصیل خواهد آمد.

ازمهمترین عناصر در کشف نیازمندیها میزان ارتباط با خبره های دامنهٔ مسئله است. در پایان مرحله تفصیل معماری خط پایه سیستم مشخص می شود این معماری شامل (1) لیست تمامی موارد قابل کاربرد که نیازمندیهای سیستم را بیان می کند. (2) مدل دامنهٔ مسئله که یک نقطه شروع برای کارهای بعدی است و (3) ساختار تکنولوژی انتخاب شده . به این معماری ، طرح اولیه گفته می شود.

برنامه ریزی مرحله ساخت
برای برنامه ریزی مرحله ساخت، روشهای مختلفی وجود دارد که در زیر یک روش برنامه ریزی را توضیح می دهیم.
اساس برنامه ریزی تعیین تکرارها و همچنین موارد قابل کاربردی که باید در هر تکرار تحویل داده شود. بعضی از توسعه دهندگان موارد قابل کاربرد را در اندازه های کوچک دوست دارند بطوریکه بتوانند هر کدام از این موارد را در یک تکرار به

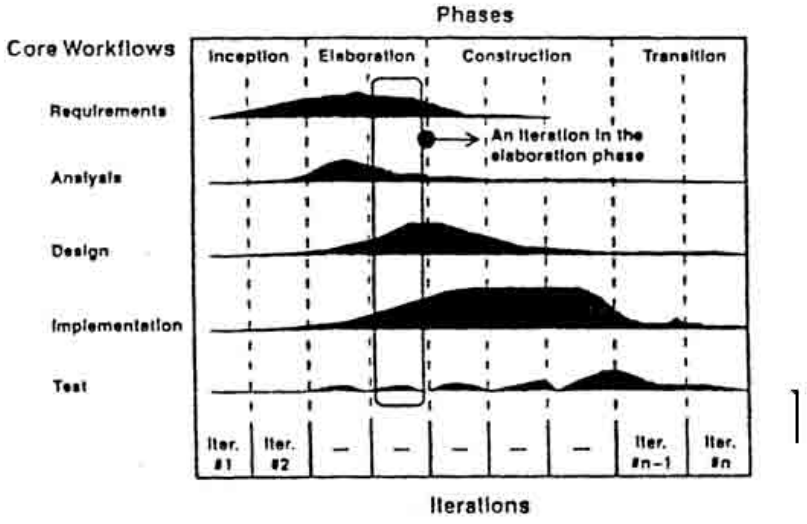
اتمام برسانند اما برخی دیگر موارد قابل کاربرد بزرگتری را دوست دارند و در هر تکرار تعدادی از سناریوهای موجود در مورد قابل کاربرد را انجام می دهند و بقیه را به تکرار های بعدی محول می کنند. اساس کار یکی است و در این روش موارد با اندازه کوچک را درنظر می گیریم، در طول برنامه ریزی با دو گروه از افراد سرو کار داریم که عبارتند از: مشتری و توسعه دهنده

برای برنامه ریزی مرحله ساخت گامهای زیر توصیه می شود:

در قدم اول باید موارد قابل کاربرد طبقه بندی شود، مشتری موارد قابل کاربرد را با توجه به ارزش کار در سه سطح بالا، متوسط و پایین طبقه بندی می کند و به هر مورد قابل کاربرد یک اولویت نسبت داده می شود سپس مشتری مضمون هرطبقه را یادداشت میکند.

در قدم دوم توسعه دهندگان پس از تقسیم بندی مشتری موارد قابل کاربرد را با توجه به ریسک توسعه تقسیم بندی می کنند. ریسک بالا برای چیزهایی در نظر گرفته میشودکه انجام آنها مشکل و یا بدرستی برای توسعه دهنده درک نشده است.

در قدم بعدی توسعه دهنده باید زمان مورد نیاز برای هر مورد قابل کاربرد را تخمین بزند. در این تخمین فرض براین است که در هر تکرار به فازهای تحلیل، طراحی، کدنویسی، تست کردن، مجتمع سازی و مستند سازی نیاز داریم. تعدادی از این فازها در شکل 3-2 نشان داده شده است. توجه شود تخمین زدن را کارشناس توسعه انجام می- دهد نه مدیر.با تمام شدن تخمین زدن باید بررسی کنیم که آیا آمادگی لازم برای برنامه ریزی داریم . اگر موارد قابل کاربرد با میزان ریسک بالا زمان زیادی از زمان پروژه را لازم داشتند باید برای این موارد قابل کاربرد به مرحله تفصیل برگردیم .



شکل ۳-۲: فازهای تکرار در فرآیند توسعه

در قدم چهارم طول هر تکرار را تعیین می نماییم، ما به یک طول ثابت برای تکرار در کل پروژه نیاز داریم با این کار یک ریتم منظم در اتمام تکرار خواهیم داشت. طول تکرار باید طوری انتخاب شود که بتوان مورد قابل کاربرد را دراین زمان انجام داد.حال باید به عنوان کار لازم برای انجام هر تکرار بیاندیشیم، سپس مشخص کنیم که با چه سرعتی می توانیم در پروژه پیشرفت کنیم.

قدم بعدی برنامه ریزی مرحله ساخت نسبت دادن موارد قابل کاربرد به تکرار هاست. این تخصیص با توجه به اولویتها نسبت داده شده در مراحل قبلی صورت می گیرد بطوریکه موارد با اولویت بالا اول انجام داده می شود، در این مرحله ممکن است موارد قابل کاربرد بزرگ به موارد کوچکتری تقسیم شوند ودر تخمین طولهای تکرار و زمان لازم تجدید نظر کنیم . برای پیش بینی مقدار زمان مورد نیاز در مرحله انتقال معمولا 10 تا 35 درصد از مرحله ساخت را بعنوان تخمین مرحله انتقال درنظر می گیرند.

مرحله ساخت

در مرحله ساخت سیستم با طی کردن یک سری تکرارها ساخته می شود. هر تکرار یک پروژه کوچک است باید تحلیل، طراحی، کدنویسی، تست و مجتمع سازی را برای تمام موارد قابل کاربرد منسوب به یک تکرار انجام دهیم و هر تکرار را با نشان دادن نمایی از کار به کاربر و تست کردن سیستم برای تایید اینکه موارد قابل کاربرد بدرستی ساخته شده اند، به پایان می رسانیم و هدف از این کار کاهش ریسک است . تکرارها در مرحله ساخت بصورت تکراری - افزایشی است : افزایشی بودن به این معنا است که درهر تکرار جزئی از کل ساخته می شود و به اجزاء ساخته شده درتکرار های قبلی اضافه می گردد.

تکراری بودن به فاز کدنویسی مر بوط می شود و چنین است که در هر تکرار دوباره نویسی در قسمتهایی از کدهای قبلی به منظور قابل انعطاف پذیرتر نمودن سیستم صورت می گیرد.

مرحله انتقال

در طول مرحله انتقال نرم افزار در اختیار جامعه کاربران قرار می گیرد و هیچ توسعه ای در افزایش عملکرد سیستم صورت نمی گیرد، مگر اینکه یک عمل کوچک یا یک عمل اساسی را به سیستم بیافزاییم، لذا تنها توسعه ممکن برطرف نمودن خطاهاست. یک مثال خوب برای مرحله انتقال فاصله زمانی بین محصول بتا و محصول نهایی است و تست بتا و بهینه سازی در این مرحله انجام می شود.

در پایان مرحله انتقال عملکرد فرآیند توسعهٔ مورد استفاده را بررسی می نماییم و کم و کاستیهای فرآیند را بمنظور استفاده در پروژه های دیگر برطرف می کنیم.

زبان مدل سازی یکپارچه UML

مقدمه
UML یک زبان مدل سازی است، یک متد نیست، بسیاری از متدها حداقل شامل یک زبان مدلسازی و یک فرآیند هستند، زبان مدلسازی اصولاً شامل علائمی گرافیکی است که متدها با استفاده از این علائم مدل های طراحی شده را نمایش می دهند. اما فرآیند نشان می دهد که چه قدمهایی را باید در مرحله طراحی برداریم تا به هدف مورد نظر برسیم.
UML ترکیبی از سه روش مدلسازی OOSE, BOOCH, (Object Modeling Technique) OMT (Object-Oriented Software Engineering) است. این زبان یک استاندارد مدلسازی است، وجود یک زبان استاندارد خیلی مهم است اما لزومی ندارد که حتماً یک فرآیند استاندارد داشته باشیم . در دهه 1980 زبانهای شئ گرا مثل small talk , c++ ابداع شدند، این زبانها باعث شدند که توسعه دهنده بتواند دنیای واقعی را بصورت مجموعه ای از اشیای مربوط به هم در نظر بگیرد و همان تصور ذهنی خود را با استفاده از یک زبان برنامه نویسی شئ گرا پیاده سازی کند، همین امر باعث شد که کتابهای زیادی در زمینه تحلیل و طراحی شئ گرا در سالهای 88 تا 92 منتشر شود. روش Booch در فازهای طراحی و ساخت پروژه به خوبی عمل می کرد و OOSE موارد قابل کاربرد را برای گرفتن نیازمندیهای کاربر و تحلیل و طراحی سطح بالا را بطور عالی پشتیبانی می کرد، روش OMT نیز برای تحلیل و تمرکز روی داده های سیستم اطلاعاتی (data intensive) بسیار مفید بود.
در اکتبر 1994 وقتی که Rambugh از شرکت جنرال الکتریک به Booch ازشرکت Rational پیوست، عملاً کار خود را برای ترکیب روشهای شخصی جهت ایجاد یک زبان مدلسازی واحد، شروع کردند. در اکتبر 1995 نسخهٔ 0.8 روش Unified را ارائه دادند. در همین زمان Jacobson نیز برای شرکت دادن روش شخصی OOSE در پروژهٔ UML با آنها به توافق رسید و مستندات نسخه UML 0.9 را در ژانویه 1996 آماده کردند. سر انجام در سال 1997 نسخه UML 1.0 ارائه شد و در همین سال به عنوان یک استاندارد توسط شرکت OMG شناخته شد.

علائم و فرامدل ها
علائم همان عناصر گرافیکی هستند که در مدلها دیده می شوند و نحو زبان مدلسازی را تشکیل می دهند. بعنوان مثال : علائم نمودار کلاس نمایشی را برای موارد و مفاهیمی نظیرکلاس، تناظر و چند تایی تعریف می کند.
در سازمانهای مختلف با حوزه های مسئله متفاوت، فرآیندهای متفاوتی مورد نیاز است، بنابراین تلاش بر این است که ابتدا بر یک فرامدل مشترک تمرکز شود و در درجه دوم بریک علامتگذاری مشترک تمرکز گردد. لذا فرامدل را چنین تعریف می-کنیم: نموداری- معمولا نمودار کلاس- است که علائم را تعریف می کند.

چرا مدلسازی می کنیم؟
مدلسازی قسمت عمدهٔ فعالیتهایی است که مقدمه یک نرم افزار مناسب- نیازهای کاربر را برآورده کند- رافراهم می کند، بدین دلیل مدلسازی می کنیم که ساختار و رفتار مناسبی برای سیستم مرتبط با سیستم واقعی در نظر گرفته شود. می دانیم که زبان طبیعی برای این کار بعلت ابهام و پیچیدگی زیاد هنگام کار در سطوح پایین یک سیستم پیچیده مناسب نیست و کد نیز اگر چه دقیق است اما چون به جزئیات می پردازد و به یکباره نمی توانیم وارد این جزییات شویم، مناسب نیست. بنابراین برای اینکه سیستم را به صورت بصری ببینیم و معماری آن را کنترل نماییم و همچنین درک بهتری از سیستم داشته باشیم، مدلسازی می کنیم. یک مدل، نمایی ساده از دنیای واقعی است. بطور کلی مدلسازی از چهارجنبه زیر به ما کمک می کند:

1. مدلها به بصری سازی سیستم -آنچنانچه در دنیای واقعی وجود دارد یا آن طوریکه ما می خواهیم - کمک می کند .
2. مدل ها به ما اجازه می دهند که ساختار ورفتار سیستم را مشخص کنیم .
3. مدلها یک قالب را تعریف می کنند که مارا در مرحله ساخت هدایت می کند.
4. مدلها تصمیماتی که در مورد پروژه اخذ شده، بصورت مستند بیان می کند.

موارد قابل کاربرد Use cases
موارد قابل کاربرد با تکیه بر اینکه سیستم چه کاری را انجام می دهد یا اینکه سیستم جدید چه اعمالی را مضاف برکارهای قبلی باید انجام دهد، یک مدل از سیستم می دهد که مدل مورد قابل کاربرد نامیده می شود. این مدل برای تحلیل نیازهای عملیاتی سیستم بکار می رودکه این مدل درفاز تحلیل توسعه سیستم شئ گرا ایجاد می شود .
مدل کردن موارد قابل کاربرد در اولین مرحله توسعه سیستم برای کمک به توسعه دهنده جهت درک بهتر نیازهای عملیاتی سیستم بدون اینکه به چگونگی انجام این عملیات بپردازد، مفید واقع می شود. توسعه دهندهٔ نرم افزار باید با کاربر در طول پروسه توسعه درارتباط باشد و در مشخصات نیازمندیها به توافق برسند . اگر نیازمندیهای کاربر در طول پریود توسعه نرم افزار تغییر کند، این تغییرات ابتدا در مدل مورد قابل کاربرد اعمال می شود . تغییرات در مورد قابل کاربرد منجر به تغییر در مدلهای دیگر می شود . این مدلها می توانند اثرات اجرایی را نشان دهند، مثلاً شما می توانید اثرات یک تغییر در طراحی یک مورد قابل کاربرد را مشاهده نمایید.مدل مورد قابل کاربرد شامل دو عنصر کنشگر و مورد قابل کاربرد است . کنشگر یک عنصر خارجی است که با سیستم تعامل دارد، در واقع کسی یا چیزی (یک سیستم خارجی) که اطلاعات را در سیستم تغییر می دهد، کنشگر می باشد .
مورد قابل کاربرد نمایشی از مجموعه ای از اعمال مرتبط به هم است که با کنشگر مقدار دهی می شوند. برای تعریف دقیق

مورد قابل کاربرد ابتدا مفهوم سناریو را تعریف می کنیم .
سناریو یک توالی از مواردی است که تعامل بین سیستم و کاربر را بیان می کند.
مورد قابل کاربرد مجموعه ای از سناریو های مرتبط به هم است که یک هدف کاربر را دنبال می کنند. تفاوتی که بین کاربر و کنشگر بتوان به آن اشاره کرد این است که کاربر کسی است که از سیستم استفاده می کند، اما کنشگر نشان دهنده نقشی است که کاربر در آن نقش عمل می کند، ممکن است که یک کاربر در چند نقش متفاوت از سیستم استفاده کند. نام کنشگر باید نشان دهنده نقش کاربر باشد، یک کنشگر کلاسی از کاربران را تعریف می کند و یک کاربر نمونه ای از کلاس کنشگر خاصی است. چون کنشگر ها در خارج از سیستم قرار دارند نیازی نیست که جزئیات آن را بررسی کنیم.
یک شکل ساده مورد قابل کاربرد شامل شرح سناریوی آن بصورت ترتیبی از مراحل شماره گذاری شده بهمراه راه حل های ترتیبی به ازای حالات مختلف مورد قابل کاربرد است.
مورد قابل کاربرد فقط آنچه یک سیستم انجام می دهد را توصیف می کند و از چگونگی انجام آن صحبتی نمی کند. برای این منظور، توصیف جریان وقایع یک مورد کاربرد کار مناسبی است. جریان وقایع عبارت است از متنی که در آن موارد زیر لحاظ می شود:

- مورد کاربرد کی و چگونه شروع و خاتمه می یابد؟
- چه زمانی تعامل با کنشگر صورت می گیرد؟
- چه اشیائی در چه زمانی مورد مبادله قرار می گیرند؟
- چه روالی عملیات اصلی و چه روالی عملیات استثنایی وجود دارد؟

متن زیر جریان کار یک مورد قابل کاربرد ساده را نشان می دهد.

Buy a product:
1. customer browses through catalog and selects items to buy
2. customer goes to check out
3. customer fills in shipping information (address; next-day or 3-day delivery)
4. system presents full pricing information , including shipping
5. customer fills in credit card information
6. system authorizes purchase
7. system confirms sale immediately
8. system sends confirming email to customer
Alternative: Authorization failure
At step 6 , system fails to authorize credit purches
Allow customer to re-enter credit card information and re-try
Alternative: Regular customer
3a. system displays current shipping information , pricing information , and last fouur digits of credit card information
3b. Customer may accept or override these defaults
Return to primary scenario at step 6

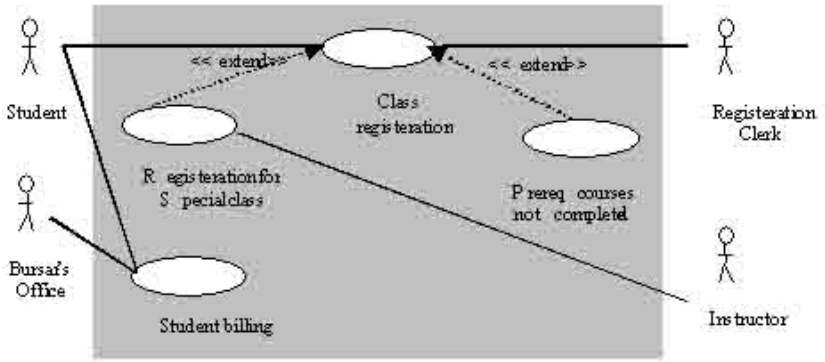
تا به حال به بررسی خصوصیات موارد قابل کاربرد پرداختیم . حال سوال این است که چگونه موارد قابل کاربرد را شناسایی کنیم ؟ برای مشخص کردن موارد قابل کاربرد Jacobson بررسی سوالات زیر را پیشنهاد می کند.

- آیا کنشگر اعمال خواندن و بروز رسانی اطلاعات را انجام خواهد داد؟
- وظیفه اصلی هر کنشگر چیست ؟
- آیا کنشگر الزاما تغییرات خارج از سیستم را باید به اطلاع سیستم برساند؟
- آیا کنشگر باید از تغییرات غیر منتظره آگاهی داشته باشد؟

نمودار مورد قابل کاربرد
موارد قابل کاربرد را بعنوان عناصر اولیه توسعه نرم افزار معرفی نمودیم، این عنصر اولیه باید حاوی اطلاعات سطح بالا در مورد پروژه باشد. Jacobson در سال 1994 نمودار مورد قابل کاربرد را برای بصری سازی این عناصر ارائه نمود. در UML ، کنشگر و مورد قابل کاربرد را بصورت شکل زیر نشان می دهند:

شکل ٤-٣ : نماد گرافیکی مورد قابل کاربرد و کنشگر

شکل زیر بعضی از موارد قابل کاربرد را برای یک سیستم ثبت نام دانشگاه نشان می-دهد.



شکل ۴-۴ :نمودار مورد قابل کاربرد برای ثبت نام

روابط بین موارد قابل کاربرد

سه نوع رابطه بین موارد قابل کاربرد می تواند وجود داشته باشد که عبارتند از: « extend » و « include » و تعمیم. اگر قصد مدل کردن حالتی خاص از یک مورد قابل کاربرد را جهت توسعه سیستم داشته باشیم و موارد قابل کاربرد اصلی به تنهایی و بدرستی وجود داشته باشد، حالت خاص را در یک مورد قابل کاربرد جداگانه نشان می دهیم و از رابطه « extend » استفاده می کنیم بطوریکه رابطه بین دو مورد قابل کاربرد بصورت یک پیکان با برچسب « extend » که جهت آن به سمت مورد قابل کاربرد اصلی است، در نمودار مورد قابل کاربرد نمایش داده می شود. عمل مربوط به مورد جدید لزوماً در چرخهٔ عمر مورد توسعه یافته صورت نمی گیرد. اما اگر بخواهیم یک رفتار مشابه بین دو یا چند مورد قابل کاربرد را بصورت یک مورد قابل کاربرد مجزا نشان دهیم و یا قسمتی از کارهای یک مورد قابل کاربرد را بمنظور کاهش پیچیدگی تحت عنوان یک مورد جدید در نمودار نشان دهیم، از رابطه « include » استفاده می کنیم بطوریکه از هر مورد قابل کاربرد شامل عملکرد مورد قابل کاربرد جدید، یک پیکان با برچسب « include » که به سمت مورد جدید نشانه می رود، نمایش داده می شود. در واقع رفتار مورد قابل کاربرد جدید قسمتی از رفتار مواردی را که با این مورد رابطهٔ « include » دارند، تشکیل می دهد و لزوماً عمل مربوط به این مورد در چرخهٔ عمر موارد تقسیم شده صورت می پذیرد. رفتار مورد قابل کاربرد جدید که با رابطه«include» به مدل اضافه می شود به منظور کامل کردن عملکرد موارد قابل کاربرد خاصی که از قبل وجود داشت و نیز عدم تکرار این رفتار درهر یک از آنها انجام می شود . هنگامی که تمام موارد قابل کاربرد در نمودار نشان داده شود مدل مورد قابل کاربرد کامل است. اما چگونگی تعامل کنشگر با مورد قابل کاربرد و همچنین نحوه عملکرد موارد قابل کاربرد را نشان نمی دهد.
همانطور که اشاره شد، محتوای موارد قابل کاربرد بوسیله متن ساده بیان می شود و هنگام تشریح موارد قابل کاربرد باید به رفتار خارجی مورد بیاندیشیم نه به نحوه عملکرد درونی، و باید چگونگی تعامل کنشگر با مورد قابل کاربرد را از متن استخراج نماییم. دراینجا مثالی دیگر از بیان مورد قابل کاربرد را بررسی می- کنیم.
مثال: می توانیم مورد قابل کاربرد ثبت نام کلاس را بصورت زیر با متن ساده بیان نماییم.

1. دانشجو با وارد کردن اطلاعات زیر فرم ثبت نام را تکمیل می کند

cource number: section number:
year : term :

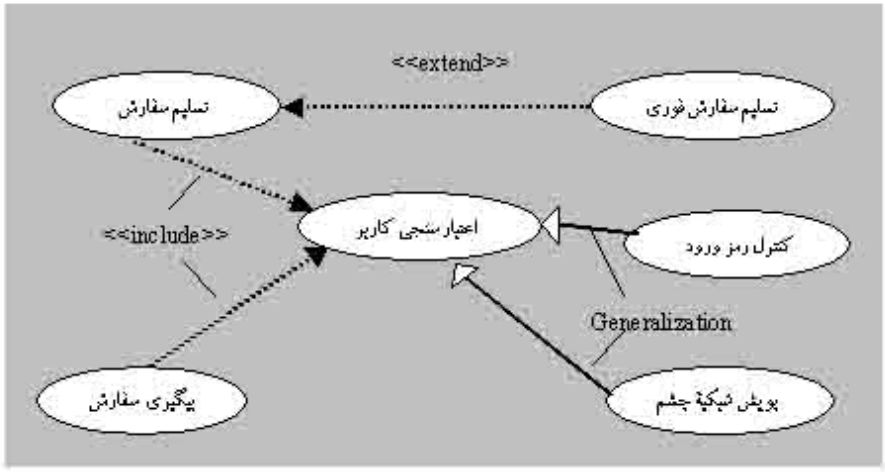
2. دانشجو فرم تکمیل شده ثبت نام را برای امضاء زدن به استاد راهنما می دهد، استاد راهنما نیز پس از بررسی اطلاعات وارد شده آن را امضاء می کند.
3. دانشجو فرم ثبت نام را به کارمندی که در اداره ثبت نام اطلاعات را وارد کامپیوتر می نماید، تحویل می دهد.
4. کارمند ثبت نام نیز پس از وارد کردن اطلاعات یک پرینت از کلاسهایی که دانشجو ثبت نام نموده است، در

اختیار دانشجو قرار می دهد.

توصیف بالا شامل سه مؤلفه زیر می باشد:

- یک مورد قابل کاربرد که دانشجو را در کلاس ثبت نام می کند، قابل مشاهده است.
- کنشگر (دانشجو)که مورد قابل کاربرد را شروع و آن رامقدار دهی می نماید.
- تبادل اطلاعات بین کنشگرها (دانشجو و کارمند ثبت نام)ومورد قابل کاربرد.

یک نوع رابطه دیگر که درمدل مورد قابل کاربرد قابل استفاده است، رابطه تعمیم می- باشد. از تعمیم زمانی استفاده می شود که یک مورد قابل کاربرد شبیه به موردی دیگر داشته باشیم که موردجدید همان کارهای مورد قبلی بعلاوه یکسری کار جزئی دیگر انجام دهد. تعمیم در موارد قابل کاربرد همانند تعمیم در کلاسهاست در اینجا موارد کاربردتخصیص رفتار را از موارد کاربرد تعمیم به ارث می برند.
در شکل 4-5 روابط «include» و «extend» و تعمیم نشان داده شده است. در این شکل کنترل رمز ورود و پویش شبکیه چشم تخصیص هایی از اعتبار سنجی کاربر هستندکه رفتار اعتبار سنجی کاربر را به ارث می برند و علاوه بر آن دارای رفتاری اضافه هم هستند. همچنین تسلیم سفارش فوری یک جریان استثنائی از تسلیم سفارش است که به صورت یک مورد قابل کاربرد با استفاده از رابطه «extend» با مورد تسلیم سفارش در ارتباط است. همین شکل به وضوح نشان می دهد که تسلیم سفارش شامل اعتبار سنجی کاربر است و از«include» استفاده شده است.



شکل ٤-٥ : نمودار مورد قابل کاربرد سیستم سفارش

یک مورد قابل کاربرد ممکن است نقاط قابل توسعه زیادی داشته باشد و مورد قابل کاربرد نیز یکی یا چند تا از این وجوه توسعه را با تعریف یک مورد جدید توسعه دهد . روابط «include»، «extend» و تعمیم به ما اجازه می دهند که مورد قابل کاربرد را به موارد کوچکتری تقسیم کنیم، در طول فاز تفصیل معمولا موارد قابل کاربردی که بسیار پیچیده اند به موارد ریزتری تقسیم خواهند شد. همچنین در فاز ساخت نیز اگر تشخیص بدهیم که در یک تکرار نمی توانیم کل مورد قابل کاربرد را بسازیم از تکنیک تقسیم استفاده خواهیم کرد . بعد از عمل تقسیم ابتدا مورد اصلی و سپس موارد اشتقاق شده را می سازیم.
بطور کلی قوانین زیر را برای روابط موارد کاربرد داریم :

- زمانی که رفتاری را بطور مشابه در دو یا چند مورد جدا از هم داریم برای پیشگیری از تکرار از رابطه « include» استفاده می کنیم .
- هنگامی که می خواهیم راههای متفاوت یک رفتار عادی را توصیف کنیم برای بیان همه این راهکارها از تعمیم استفاده می کنیم.
- برای بیان حالات استثنائی یک رفتار عادی (یک فرم کنترل شده از رفتار عادی) از رابطه « extend » استفاده می کنیم .

موارد قابل کاربرد کار و سیستم
موارد قابل کاربرد سیستم به بررسی تعامل با نرم افزار می پردازد، در حالیکه موارد قابل کاربرد کار روی چگونگی پاسخگویی کار به مشتریان و رویدادها بحث می کند. در مراحل اولیه فاز تفصیل بیشتر روی موارد قابل کاربرد کار عمل می کنیم، و موارد قابل کاربرد سیستم نیز برای برنامه ریزی و بررسی بیشتر موارد کار، مخصوصاً برای توصیف راههای متفاوت

رسیدن به هدف کنشگر، مفید است. بنابراین ابتدا موارد کار را مشخص می نماییم سپس برای تحقق بخشیدن به آنها به موارد سیستم می پردازیم. لذا بعد از مرحله تفصیل باید به ازای هر مورد کار شناسایی شده، یک مجموعه از موارد سیستم را داشته باشیم.

نمودار کلاس class diagram

نمودار کلاس بخش عمدة روشهای شئ گرا را تشکیل می دهد. این نمودار اشیاء موجود در سیستم و انواع مختلف روابط استاتیکی بین آنها را نشان می دهد. در این بخش نشان می دهیم که چگونه نمودار کلاس را با استفاده از علائم UML توسعه دهیم تا یک دید منطقی از سیستم تحت مطالعه ایجاد کنیم .

در اینجا یادآوری می کنیم که در روش شئ گرا ما دنیای واقعی را به صورت مجموعه ای از اشیاء مرتبط بهم می بینیم. و شئ موجودیتی است که نقش تعریف شده- ای در حوزة مسئله داشته باشد و شامل حالات، رفتار و مشخصه خاص خود باشد، شئ یک مفهوم، انتزاع یا چیزی محسوس از دنیای نرم را نشان می دهد که می تواند قابل رؤیت (مانند شخص، مکان) باشد، یک مفهوم یا رویداد(مانند کارایی، ثبت نام و...) یا قسمتی از فرآیند طراحی (مانند کنترلر) باشد.

شئ دارای حالت است و رفتار خاصی را از طریق اعمالی که حالت آن را ارزیابی و یا تحت تاثیر قرار می دهند، از خود بروز می دهد. رفتار، چگونگی عمل و واکنش شئ را بیان می کند و حالت شئ را می توانیم از طریق صفات و روابطی که با اشیاء دیگر دارد، تشخیص بدهیم. هر شئ یک مشخصه دارد بطوریکه هیچ دو شیئی دارای مشخصه یکسان نیستند.

رفتار شئ به حالت شئ و عملی که بر روی شئ اعمال می شود بستگی دارد، شما می توانید یک عمل را سرویسی در نظر بگیرید که از طرف تولید کننده به مشتری ارائه می شود. مشتری یک پیام را به سرویس دهنده ای که می تواند سرویس مطلوب را از طریق عمل مربوطه ارائه دهد، ارسال می کند. در نمودار کلاس گروهی از اشیائی که دارای ساختار و رفتار مشترکی باشند تحت عنوان یک کلاس شناخته می شوند.

چشم اندازها

قبل از اینکه نمودار کلاس را بررسی بکنیم، نحوه نگرش و تفسیر این نمودارها را بررسی می کنیم، بطور کلی از سه چشم انداز مختلف می توان این نمودارها را نگاه کرد.

چشم انداز مفهومی : این چشم انداز مفاهیم حوزة مسئلة تحت مطالعه را نشان می-دهد. این مفاهیم ذاتاً مرتبط با کلاس هایی است که آنها را پیاده سازی می کنند، اما نگاشت مستقیمی وجود ندارد که مفاهیم را به کلاسها تبدیل کند. مدل مفهومی هیچ توجهی به نرم افزار پیاده سازی ندارد و مستقل از آن عمل می کند (به این چشم انداز دید مستقل از زبان نیز گفته می شود).

چشم انداز تشخیصی : در این چشم انداز با توجه به نرم افزار عمل می کنیم اما صرفاً در سطح واسط نرم افزاری به مسئله نگاه می کنیم نه در سطح پیاده سازی .توسعه شئ- گرا بر تفاوت بین واسط نرم افزاری و پیاده سازی تاکید زیادی می کند اما در عمل به دلیل ترکیب شدن این دو مفهوم در سطح زبان برنامه نویسی شئ گرا از آن چشم پوشی می شود و این زیاد جالب نیست چون مزیت عمده برنامه نویسی شئ گرا برنامه نویسی در سطح واسط بین کلاسهاست نه نحوه پیاده سازی کلاسها، در این چشم اندازها بیشتر به انواع توجه می شود تا به کلاسها . در این قسمت با توجه به اینکه روی واسط ها تمرکز داریم، می توان یک نوع را با کلاسهای متعددی پیاده سازی کرد.

چشم انداز پیاده سازی : در این دیدگاه واقعاً به کلاس ها و چگونگی پیاده سازی انواع شناسایی شده در چشم انداز تشخیصی می پردازیم و این چشم انداز عموماً مورد توجه مهندسین کامپیوتر است.

در خلال معرفی نمودار کلاس وابستگی عناصر آن با چشم اندازها را بررسی خواهیم کرد. چشم اندازها جزء UML نیستند اما هنگام مدلسازی و بازنگری مدلها بسیار با ارزش هستند و می توان از UML در هر سه چشم انداز استفاده کرد. در فصل اول نحوة نمایش کلاس را در UML (شکل 1-1) نشان دادیم. این نمایش از کلاس شامل سه قسمت می باشد در قسمت بالایی اسم کلاس نوشته می شود درقسمت وسطی صفات مربوط به کلاسها قرار می گیرد، هر نمونه از کلاس توسط تعدادی از این صفات قابل شناسایی است(مشخصه شئ) و قسمت تحتانی شامل اعمال قابل انجام توسط کلاس است. با این توصیف کلاس یک قالب مشخص را برای نمونه هایی که ماهیت یکسان دارند، فراهم می کند.

صفات و اعمال

صفات خیلی شبیه تناظرهاست. در سطح مفهومی صفت نام مشتری نشان دهنده این است که مشتریان دارای نام هستند، در سطح تشخیصی این صفت نشان دهنده این است که شئ مشتری می تواند نام خودش را به شما بگوید و روشی برای تنظیم نام ها باید وجود داشته باشد. اما در سطح پیاده سازی بدین معناست که customer دارای فیلدی برای نام است. بیان صفات در UML بصورت زیر است:

visibility name : type = default_value

visibility محدوده دید را تعیین می کند، که شامل سه نوع دید عمومی، خصوصی و محافظت شده است. دید عمومی با علامت (+)، خصوصی با (-) و دید محافظت شده با(#) مشخص می شود. در دید عمومی دسترسی به این صفت برای همه کلاسهای

زیر سیستم به شرطی که به نوعی با کلاس شامل این صفت در ارتباط باشند، مجاز است. اما در دید خصوصی دسترسی به این صفت فقط برای اعمال همان کلاس مقدور است. دید محافظت شده بدین معناست که دسترسی به این صفت برای زیرکلاس های کلاس موجود مجاز است name نیز یک رشته از کاراکترهای الفبا عددی است که با یک حرف شروع می شود و نام صفت را مشخص می کند. Type نوع صفت را نشان می- دهد. Default_value مقدار پیش فرض که در لحظه ایجاد شئ در این صفت قرار می- گیرد، را نشان می د هد. برای نشان دادن چندتایی در صفت از شکل کلی

name[multiplicity]

استفاده می شود.
اعمال شامل فرآیند هایی است که کلاس باید آنها را انجام دهد. از نظر دیدگاه تشخیصی اعمال همان متدهای عمومی یک نوع (کلاس) می باشد. در UML اعمال را بصورت زیر تعریف می کنیم:

visibility name(parameter- list):return-type-expression {property- string}

visibility و name همانطوریکه در صفات استفاده می شود در اعمال نیز بکار می رود.
parameter- list لیست پارامترها است که با کاما از هم جدا شده اند و مانند صفات بصورت زیر تعریف می شوند:

direction name : type = Defulte value

با این تفاوت که در اینجا یک عنصر اضافی direction اضافه شده است که بمنظور نشان دادن اینکه پارامتر از نوع ورودی (input(in ، خروجی output(out) یا هر دو ورودی / خروجی (inout) است، استفاده می شود. هر جاکه هیچ جهتی استفاده نشده باشد بدین معناست که پارامتر ورودی است.
return-type-expression : انواع داده های برگشتی که با کاما از هم جدا شده اند را نشان می دهد. اغلب از یک نوع برگشتی استفاده می شود اما چند نوع برگشتی نیز مجاز است.
property- string : شامل مقادیرمشخصاتی است که برای انجام یک عمل داده شده بکار برده می شود. از دیدگاه مفهومی نباید اعمال را بعنوان یک واسط تعبیر کنیم بلکه آن را جزء مسؤلیتهای شئ در نظر بگیریم. اعمال را می توان به سه نوع زیر طبقه بندی کرد:
constructor operation : عملی که نمونه ای جدید ازکلاس را می سازد(یک شئ جدید).
query operations : شامل اعمالی است که به حالت شئ دسترسی دارد ولی حالت آن را تغییر نمی دهد.

Update operation : این اعمال حالت شئ را تغییر می دهند.(به این اعمال setting method نیز گفته می شود که یک مقدار را در یک فیلد قرار می دهند و هیچ کار دیگری انجام نمی گیرد.)
query ها را با هر ترتیبی می توان اجرا کرد اما ترتیب اجرای اعمال Updateبسیار مهم است . تفاوت بین عمل و متد این است که عمل چیزی است که در خواستی از شئ انجام می دهد- فراخوانی رویه - اما متد شامل بدنه رویه است.

تعمیم generalization

از نظر تشخیصی تعمیم به معنای این است که واسط نرم افزاری زیر نوع ها باید شامل همه عناصر واسط فوق نوع باشد، تعمیم از یک جنبه دیگر نیز جای تفکر است وآن قابلیت جایگزینی است. بدین صورت که ما باید قادر باشیم در هر جای برنامه (کد) که به فوق کلاس نیاز داشتیم، هر کدام از این زیر نوع ها را جایگزین کنیم بدون اینکه به عملکرد مسئله لطمه ای وارد شود و همه چیز به درستی کار بکند. مثلا اگر در جایی از کد لازم باشد که یک customer داشته باشیم باید بتوانیم آزادانه هر یک از زیر نوع های personal و corporate را جایگزین کنیم . corporate ممکن است که یک دستور خاص را متفاوت با زیر نوع دیگری انجام دهد اما نباید این تفاوت برای فراخوان دستور مهم باشد.
تعمیم از چشم انداز پیاده سازی با ارث بری در زبان برنامه نویسی متناظر است. زیر کلاس تمام متد ها و ویژگیهای فوق کلاس را به ارث می برد و ممکن است که متدهای به ارث گرفته را به نحو بهتری پیاده سازی نماید.