بسمه تعالي

اصول مهندسی نرم افزار

تجزیه و تحلیل سیستمها بشیوه شي گراء

بکمک بزبان UML

(جزوه دوم)

تهیه و گردآوری : حسن – علی اکبرپور

haliakbarpour@gmail.com

منابع دیگر :

سایتsoftwareengineer.blogsky.com-جزوه آقاي غفور عليپور-جزوه آقاي ايمان رحيمي نيا-كارگاه آموزشي متدولوژي شئ گراي

مدل سازي سيستم ها

در مهندسي نرم افزار پروژه اي را موفق مي گويند كه:

- در زمان مشخص و با هزينه مشخص به اتمام برسد. پروژه اي كه در زمان مشخص به اتمام نرسد و يا با هزينه اي بيشتر از برآورد اوليه به اتمام برسد ناموفق ناميده مي شود.
- نياز مشتري و بازار را جواب دهد. اگر پروژه اي تحويل شود كه نياز مشتري را به صورت كامل جواب دهد ولي فاقد زيبايي باشد خيلي موفق تر از پروژه اي است كه داراي زيبايي باشد ولي نياز مشتري را جواب ندهد . بنابراين ابتدا بايد نياز مشتري در اولويت قرار گيرد و برنامه اي با حداقل امكانات كه كليه نيازهاي مشتري را جواب مي دهد توليد گردد و سپس در مراحل بعد به سرعت و راحتي استفاده و زيبايي آن پرداخته شود.

آماري كه در سال ۲۰۰۲ و ۲۰۰۳ در مورد پروژه هاي نرم افزاري آمريكا در دست داريم اين است كه حدود ۹ درصد از پروژه ها موفق بوده اند و بقيه ناموفق بوده اند (در اصطلاح fail شده اند) در اينجا ناموفق بودن به اين معنا است كه يا پروژه بعد از زمان تعيين شده تحويل شده است يا بيش از مبلغ تعيين شده هزينه برده است.

تعريف مدل

تعريف مدل : مدل ، ساده شده دنياي واقعي است.

تعريف ديگرمدل : مدل ، يک سري اجزاء و روابط بين اجزاء در دنياي واقعي را عنوان مي كند.

مدل ايده آل :

مدل ايده ال : مدلي است كه بتواند تمامي اجزاء يک سيستم و تمامي روابط اجزاء را مشخص كند.

در عمل چنين مدلي نمي توان داشت و اگرمدلي داشته باشيم كه بتواند تا ۷۰ درصد روابط بين اجزاء را مشخص كند مدل خوبي خواهد بود ولي ۱۰۰ درصد امكان پذير نيست.(يا حداقل به سختي مي توان به چنين مدلي دست يافت).

حال ببينيم مدل سازي سيستم نرم افزاري چه كمكي به ما مي كند؟

مدل سازي در نرم افزار مثل يک نقشه راه (Roadmap) عمل مي كند. يعني به ما مي گويد كه چگونه از نقطه شروع به سمت نقطه پايان حركت كنيم كه در پايان بتوانيم محصول مورد نظر خود را تحويل دهيم.از نظر مهندسي نرم افزار چيزي كه اهميت دارد توليد محصول است. تا كنون خيلي از روشهاي مدلسازي به بازار ارائه شده اند كه هر كدام داراي ويژگيهاي خاص خود هستند. ولي همانطور كه گفته شد محصول اهميت دارد نه اينكه از كدام يک از اين روشها استفاده مي كنيم.

نتايج (فوايد) مدل سازي

۱- ديد روشن و شفاف نسبت به سيستم مي دهد.

باعث مي شود كه پيچيدگي ها و زواياي تاريک سيستم از بين برود. مثلا سيستمي مانند حقوق و دستمزد را در نظز بگيريم. در ابتدا سيستمي ساده به نظر مي رسد ولي وقتي وارد اين سيستم مي شويم پيچيدگي هاي خاص خود را دارد. بنابراين با مدلسازي اوليه آن مي توان پيچيدگي هاي آنرا تا حد زيادي از بين برد.

۲- معماري و چارچوب سيستم مشخص مي شود.

باعث مي شود كه بتوان زمان و نيروي انساني مورد نياز براي پروژه را تخمين زد. مثلا اگر شما معماري Net. را براي پروژه خود انتخاب كنيد پس به متخصصين Net. احتياج داريد.

۳- ريسک را مي توان مديريت كرد.

با بررسي موارد ۱ و ۲ مي توان ريسک هاي سيستم را شناسايي و مديريت كرد.
تعريف ريسک : هر چيزي كه از نظر زمان و بودجه پروژه را دچار مشكل كند ريسک مي گويند.
معروفترين ريسک Problem Behind Problem است. اين ريسک زماني است كه شما نياز كاربر را(به هر دليلي) اشتباه در نظر گيريد.

۴- مستند سازي و رسيدن به قابليت استفاده مجدد (Reuse) .

مدل سازي شما را وادار مي كند كه مستند سازي كنيد. و تمامي اين نتايج براي رسيدن به Reuse است. يعني از دوباره كاري در پروژه ها جلوگيري مي كند.

اصول مدل سازي سيستم ها

اصل اول : مدل بايد بهينه باشد

اگر مدل بهينه نباشد تيم به بيراهه مي رود و ممكن است در نهايت محصول مورد نظربا امكانات خواسته شده توليد نشود. مطلب ديگري كه در اصل اول نهفته است اينكه مدلسازي كاري گروهي است. يعني افرادي با تخصصهاي مختلف بايد در مدلسازي همكاري كنند. زيرا هر كس از نظر تخصص خودش به مدل نگاه مي كند مثلا طراح با يک ديد و مدير بانک هاي اطلاعاتي با ديد ديگر و ... به مدل نگاه مي كنند و اين باعث مي شود كه مدل به سمت بهينه شدن حركت كند.

اصل دوم : مدل را از زواياي مختلف مشاهده كنيد.

نگاه كردن به مدل از زواياي مختلف پاسخ به دو پرسش پاسخ است . What, How

What: چه كاري بايد انجام گيرد؟

How: چگونه بايد انجام داد؟

يعني مدل بايد به سئوالهاي تحليلگر و طراح و هم برنامه نويس جواب دهد.

اصل سوم : مدل را در دنياي واقعي مشاهده كنيد.

در مدلسازي ايده ال گرا نباشيد و امكانات دنياي واقعي را در نظر بگيريد. ممكن است كه تحليلگر مساله را كاملا ايده ال در نظر بگيرد كه از نظر طراح نتوان با تكنولوژيهاي موجود آنرا پياده سازي كرد.در اينجا ممكن است بين تحليلگر و طراح شكاف ايجاد گردد كه يک ريسک به حساب مي آيد.

تحليلگر : نياز كاربررا مي گيرد به منطق نرم افزاري تبديل مي كند.

طراح : كسي كه منطق را با تكنولوژي تركيب مي كند و خوراک پياده سازي را فراهم مي كند.

اصل چهارم : مدل به تنهايي كافي نمي باشد بلكه بايد به قطعات كوچک قابل پياده سازي و مديريت با كمترين ارتباط شكسته شود.

وقتي مدل به قطعات كوچكتر شكسته مي شود و ارتباط كاهش مي يابد تغيير در يک قسمت باعث تغيير در قسمتهاي ديگر نمي گردد وهزينه تغييرات كاهش مي يابد.

اين اصل ، برنامه نويسي بر اساس Component را ايجاب مي كند . يعني برنامه به تعدادي Component شكسته مي گردد كه هر كدام به تنهايي به درستي كار مي كند و با كنار هم قرار گرفتن آنها سيستم به طور كامل كار مي كند.هم اكنون در دنيا برنامه نويسي بر اساس Service جاي برنامه نويسي بر اساس Component را گرفته است و مفهومي تحت عنوان SOA(Service Oriented Approach) مطرح شده است.

تا اينجا به اين نتيجه رسيده ايم كه تمام پروژه های نرم افزاری بايد مدلسازی شوند.

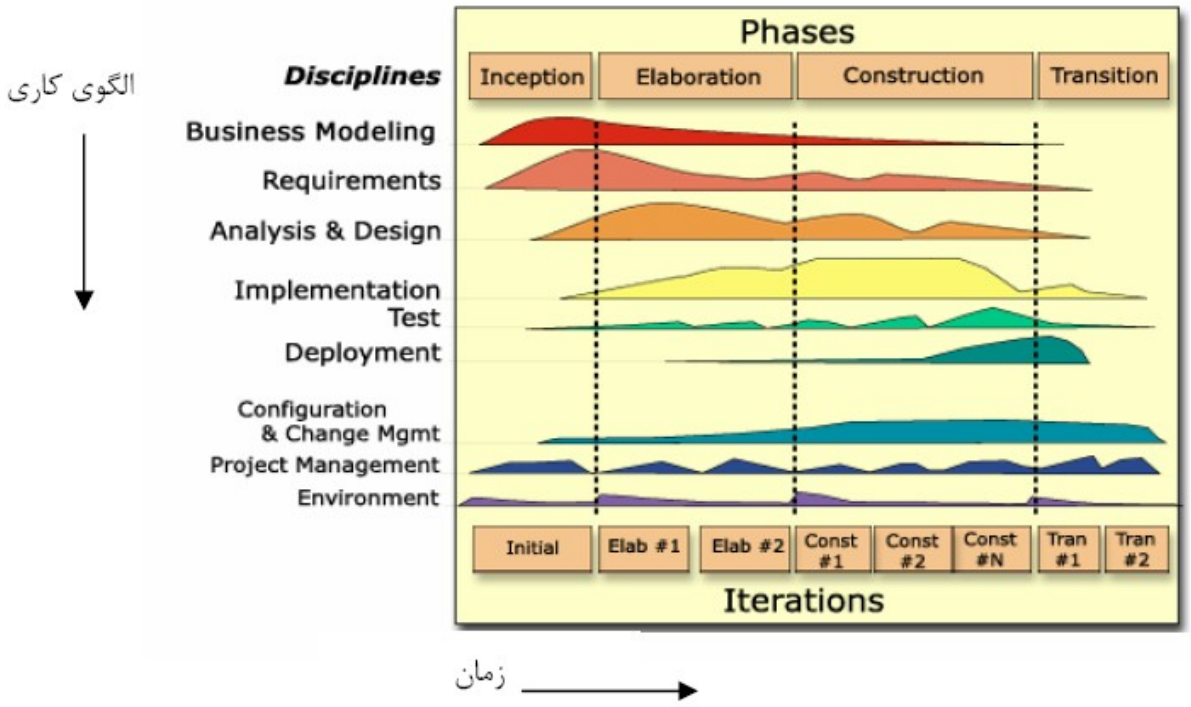
متدلوژیRUP

مقدمه

RUP فرآیند تولید نرم افزار می باشدکه فعالیت ها و مسئولیت ها را به ترتیب جهت توسعه نرم‌افزار مشخص می‌کند و هدف آن حصول اطمینان از تولید نرم افزار با کیفیت بالا براساس نیازهای کاربران و با توجه به زمان و بودجه در دسترس می باشد.

تاریخچه RUP

با پیشرفت تکنولوژی کامپیوتر، نیاز هرچه بیشتر به گسترش علوم نرم افزاری نیز احساس می‌شد،لذا متدولوژی‌های گوناگون تعریف شده و بمرور تکامل یافت .مهم‌ترین آنها متدولوژی‌های ساخت‌یافته بالاخص SSADM با نگرش آبشاری بود. در ابتدا، این روشها مناسب بوده و جوابگوی نیازهای آن زمان بودند، ولی با افزایش داده‌ها و پیدایش مفاهیمی همچون شبکه، Web و ... دیگر کارآیی لازم را جهت پیاده‌سازی و هدایت پروژه‌های نرم افزاری نداشتند. پس مفاهیم برنامه نویسی شیءگرا پا به عرصه وجود گذاشت و در سال ۱۹۹۱ بطور جدی مورد مطالعه و بحث قرار گرفت. استفاده از این روشها و متدهای برنامه نویسی قدرت و انعطاف بسیاری را به برنامه‌ها داد و شرکتهای نرم افزاری توانستند با کاهش هزینه‌ها و بهینه سازی کدهای خود، نرم افزارهای قویتری را به بازار عرضه کنند، ولی این روش جدید نیز نیاز به مدیریت و یکپارچگی داشت. پس روشها و متدولوژیهای جدیدی مطرح شد که شامل Booch، OMT، OSE و ... بودند. در سال ۲۰۰۰ شرکت Rational روشی را تحت عنوان RUP مطرح ساخت که بعد از روش MSF شرکت مایکروسافت، به دنیای نرم افزار عرضه شد و امروزه از طرفداران بسیاری برخوردار است.



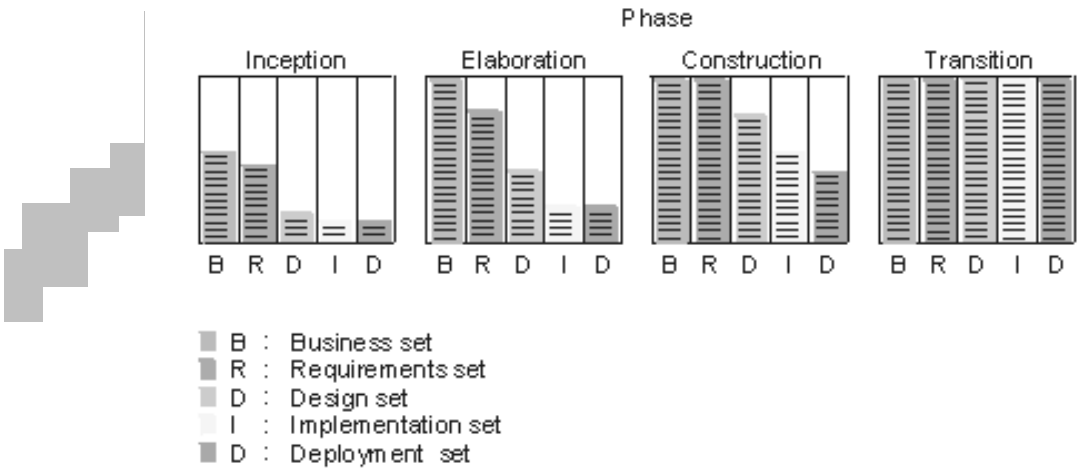
RUP داراي دو بعد است :

۱- محور عمودي : ساختار ايستا

شامل : نقش‌ها – فعاليتها - فرآورده‌ها - نظم‌ها -گردش‌کارها - جزئيات گردش‌کارها - عناصر ثانوي

۲- محور افقي : ساختار پويا

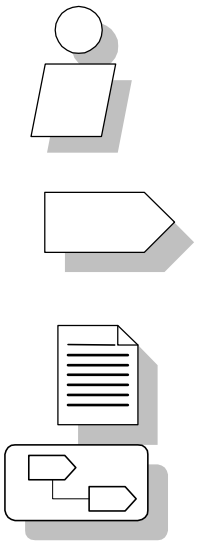
شامل : فازآغازين - فاز تشريح - فاز ساخت - فاز انتقال



رشد مجموعه هاي فراورده ها در طي چهار فاز توليد

ساختار ايستا :

انتظار مي‌رود از يك فرآيند توليد كه معين‌كند چه كسي(Who)، چه چيزي را بايد انجام دهد(What)، به چه صورت(How)، و در چه زماني(When) واقع مي شود"

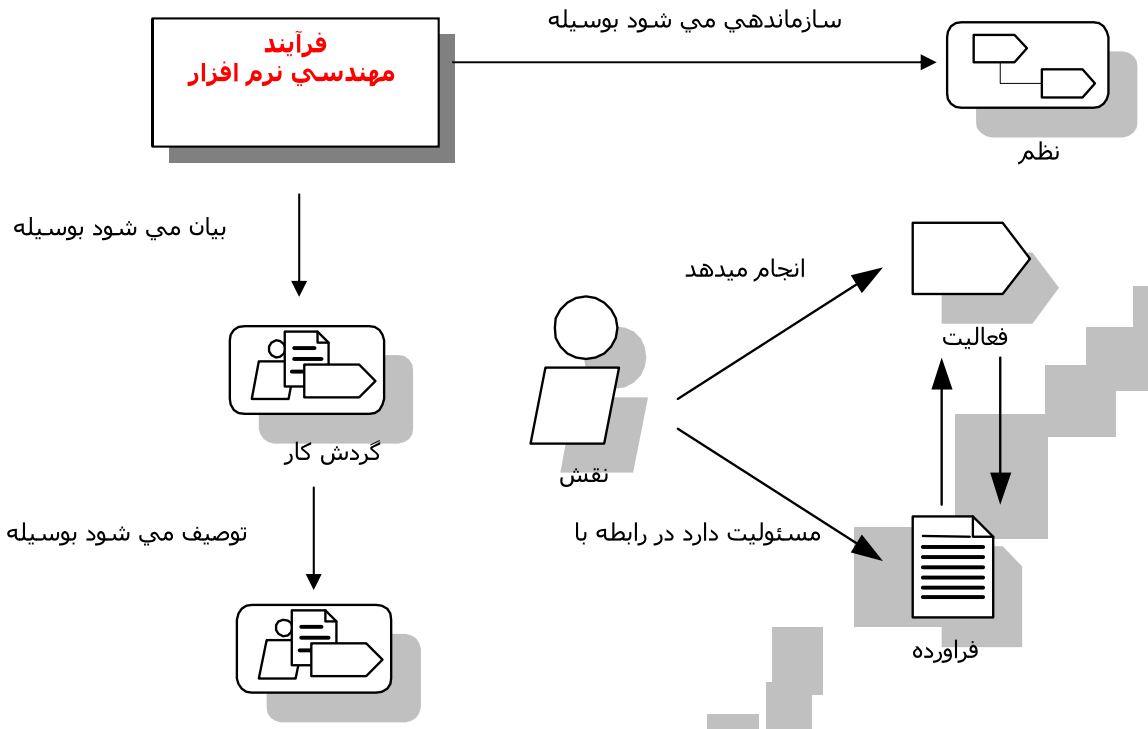


Who :(Roles)نقشها

How :(Activities)فعاليتها

What :(Artifacts)فرآورده‌ها

When :(Disciplines)نظمها



رابطه بين مؤلفه هاي اساسي بعد ايستاي RUP و فرآيند توليد نرمافزار

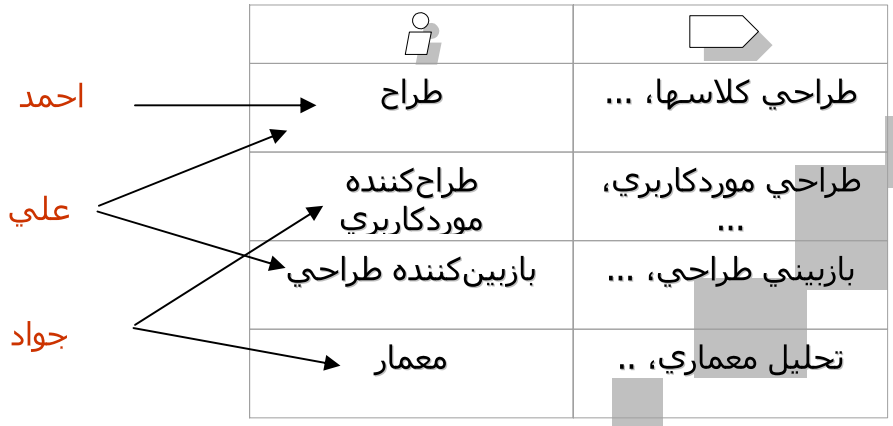
نقش (Role)

" نقش (Role) ، رفتار و مسئوليتهايي كه يك نفر (يا افراد يك تيم) در پروژه بعهده دارد، را مشخص مي‌نمايد "

رفتار ← فعاليتهايي كه يك نفر بايد انجام دهد

مسئوليتها ← وظيفه يك نفر در رابطه با توليد، به روز رساني، يا استفاده از فرآورده‌ها

مصداقهاي نقش:
۱) افراد متخصص : تحليلگر سيستم، طراح، معمار، ...
۲) سهامداران سيستم : مشتري، كاربر نهائي ، ...



نگاشت نيروي انساني به نقشها

فعاليتها (Activities)
كارهايي كه يك نقش بايد انجام دهد به صورت فعاليت بيان مي شوند

يك فعاليت:
۱) داراي هدف روشني است.
۲) تنها به يك نقش انتساب مي‌شود.
۳) بر يك فراورده (يا مجموعه كوچكي از فراورده‌ها) اثر مي‌گذارد.
٤) مدت زماني انجام فعاليت: چند ساعت تا چند روز
٥) در برنامه‌ريزي و زمانبندي واحد انجام كار

مثالهايي از فعاليت:
- يافتن موارد كاربري و عامل ها بوسيله تحليلگر سيستم
- بازبيني طراحي بوسيله بازبين كننده طراحي

گامهاي فعاليت:
۱) انديشيدن
۲) اجرا
۳) بازبيني

مثال: فعاليت: موارد كاربري و عوامل مربوطه را بيابيد
۱) عوامل را پيدا كنيد
۲) موارد كاربري را پيدا كنيد ----------> انديشيدن
۳) نحوه ارتباط عوامل با موارد كاربري را توصيف نماييد

٤) موارد كاربري و عوامل را با هم بسته بندي نماييد
٥) نمودار موارد كاربري را ايجاد كنيد ----------> اجراء
٦) مدل موارد كاربري را مستند سازيد

۷) نتايج را ارزيابي نماييد ----------> بازبيني

فراورده‌ها (Artifacts)
فراورده‌ها عبارتند از محصولاتي (يا قطعه‌هاي اطلاعاتي) كه در طي فرآيند توليد نرمافزار، ايجاد، استفاده، يا به‌روزرساني مي‌شوند.

فراورده‌ها:
- خروجي قابل لمس فرآيند توليد را تشكيل مي‌دهند
- ورودي و خروجي فعاليت‌ها هستند
- تنها يك مسئول(Owner) دارند، ولي استفاده‌كنندگان مختلفي مي‌توانند داشته باشند

نمونه‌هايي از فراورده‌ها:
- مدل ها: مدل موارد كاربري، مدل طراحي، و...
- عناصر مدل ها: كلاس ها، موارد كاربري، زير سيستم ها، و..
- مستندات، كد منبع، فايلهاي كتابخانه‌اي(مانند DLL)، مولفه‌هاي اجرائي(مانند OCX)، فايلهاي اجرايي، و...

در RUP فراورده‌ها الزاما يك مستند كاغذي نيستند.

در RUP فراورده‌ها در ۹ مجموعه ذيل طبقه بندي مي شوند
۱) مجموعه مدلسازي كاري:
- مدل موارد كاربري كاري Business Use Case Model
- مستند معماري كاري Business Architecture Document
- مستند ديدگاه كاري Business Vision Document

۲) مجموعه نيازمنديها:
- مستند ديدگاه Vision Document
- نيازمنديها: نيازهاي سهامداران، مدل موارد كاربري و مشخصات تكميلي(Supplementary Specification)
- طرح مديريت نيازمنديها

۳) مجموعه تحليل و طراحي:
- مدل تحليل Analysis Model
- مستند معماري نرمافزار Software Architecture Document

- مدل طراحي Design Model
- مدل استقرار Deployment Model

٤) مجموعه پیاده سازي:
- کد منبع و فایلهاي اجرائي
- فایلهاي داده اي مورد نیاز

٥) مجموعه آزمایش:
- طرح آزمایش Test Plan
- روال آزمایش Test Procedure
- مدل آزمایش Test Model
- مورد آزمایش Test Case

٦) مجموعه استقرار :
- طرح استقرار
- محصول نهائي
- مستندات کاربر
- مواد آموزشي Training Materials

٧) مجموعه مدیریت پیکربندي:
- نقشه مدیریت پیکربندي Configuration Management Plan
- مستند درخواست تغییر Change Request Document

٨) مجموعه مدیریت پروژه:
- فراورده‌هاي برنامه ریزي: طرح توسعه نرم افزار، نقشه تکرار، فهرست ریسکها، ومورد کاري (Business Case)
- فراورده هاي عملکردي مانند توصیف نشرها (Release Description) ، تشخیص وضعیت پروژه،...

٩) مجموعه محیط (Environment Set) :
- مورد توسعه Development Case
- راهنمائیهاي مدلسازي کاري
- راهنمائیهاي طراحي

نظمها (Disciplines)

نظم عبارتست از مجموعه‌اي از فعاليتهاي مرتبط كه به يكي از نواحي مهم (Area of Concerns) پروژه وابسته باشند.

نواحي مهم اشاره‌اي به مراحل كلاسيك فرآيند توليد آبشاري را دارد

نظمها طبيعت نيمه مرتبي دارند چون انسانها درآن دخالت دارند.

نظمها به دو گروه تقسيم مي‌شوند:

۱) نظمهاي فرآيندي Process Disciplines

۱. مدل سازي كاري Business Modeling Workflow
۲. جمع آوري نيازمنديها
۳. تحليل و طراحي
٤. پياده سازي
٥. آزمايش
٦. استقرار

۲) نظمهاي پشتيباني Support Disciplines

۱) مديريت پروژه
۲) مديريت پيكربندي
۳) محيط Environment Discipline

هر نظم با يك گردش‌كار (Workflow) نمايش داده مي شود.

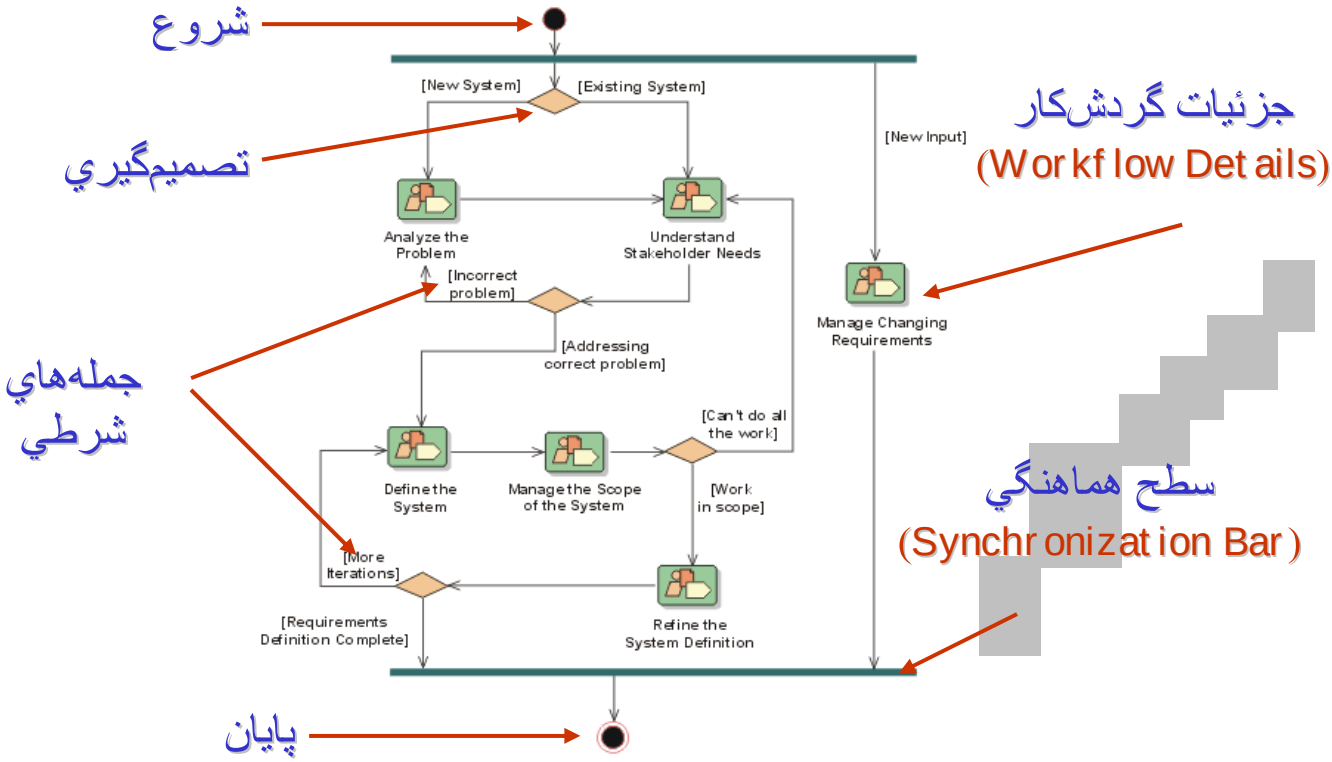
گردش‌كارها (Workflows)

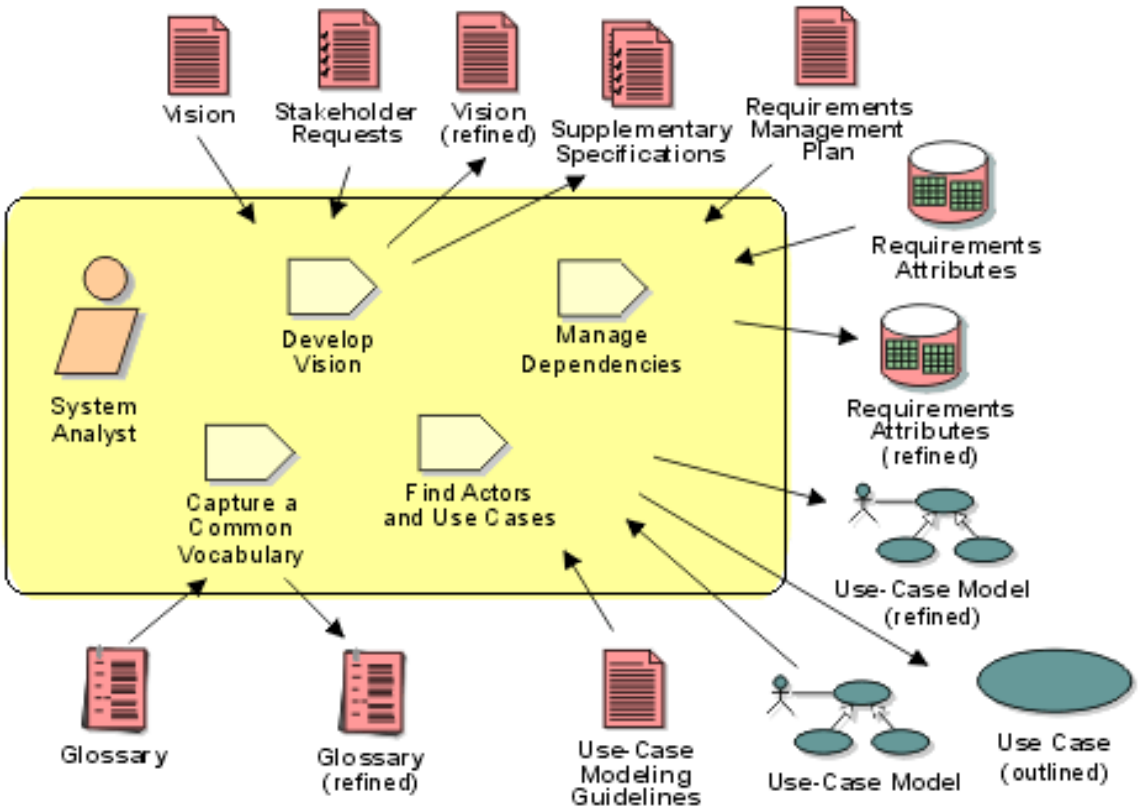
گردش‌كار عبارتست از توالي مجموعه‌اي از فعاليت‌ها كه نتيجه با ارزشي درپي‌دارند

در RUP براي نمايش گردش‌كار يك نظم از نمودارهاي فعاليت (در UML) استفاده مي شود.

جزئيات گردش‌كار عبارتست از مجموعه‌اي از فعاليتها كه معمولاً با يكديگر انجام مي‌شوند.

جزئيات‌گردش‌كار، جريان اطلاعات و نحوه ارتباط فعاليتها بوسيله فراورده‌هاي متفاوت را نمايش مي دهد


شروع
تصميم‌گيري
جزئيات گردش‌کار
(Workflow Details)
جمله‌هاي شرطي
سطح هماهنگي
(Synchronization Bar)
پايان
[New System]
[Existing System]
[New Input]
Analyze the Problem
Understand Stakeholder Needs
[Incorrect problem]
[Addressing correct problem]
Manage Changing Requirements
[Can't do all the work]
Define the System
Manage the Scope of the System
[Work in scope]
[More Iterations]
[Requirements Definition Complete]
Refine the System Definition



نمونه‌اي از يك جزئيات گردش‌كار مربوط به نظم جمع آوري نيازمنديها

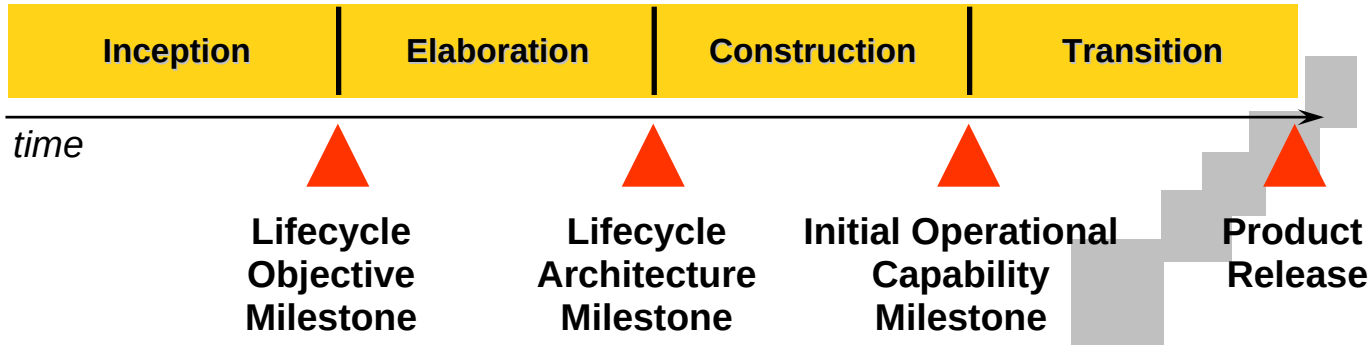
به نيمه‌مرتب بودن جزئيات گردش‌كار توجه نماييد

بعلاوه مولفه‌هاي ذكر شده، مولفه‌هاي ذيل جزو محصول RUP:

- **راهنماها Guidelines**
- **الگوها Templates**
  - **Microsoft Word, Project, FrontPage**
  - **Rational SoDA**
  - **Adobe FrameMaker**
- **راهنمايي‌هاي ابزار Tool Mentors**
- **برخي از مفاهيم پايه: ريسك، تكرار، فرسنگ شمار ،...**
- **چارچوب فرآيند Process Framework**

ساختار پويا

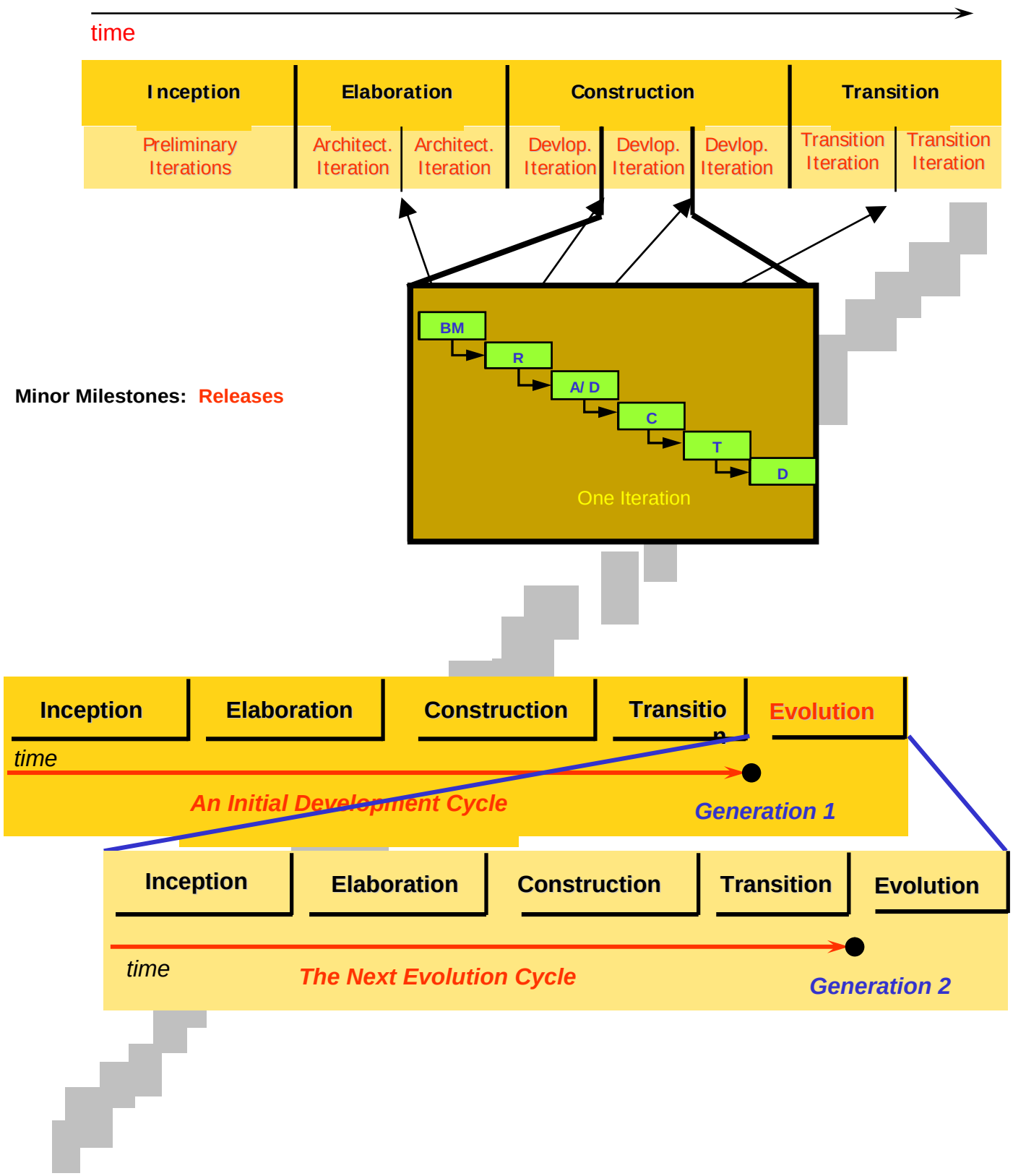
از ديدگاه مديريت سيستم ،چرخه زندگي يك نرم افزار با استفاده از متدلوژی RUP در طول زمان (پويا) به ۴ فاز متوالي اصلي تقسيم ميشود كه هر فاز با يك نقطه كليدي (Milestone) به پايان ميرسد:



- **فاز آغازين (Inception): محدوده پروژه را مشخص نماييد**
- **فاز تشريح (Elaboration):برنامه ريزي پروژه، بدست آوردن معماري پايا**
- **فاز ساخت(Construction): ساخت محصول**
- **فاز انتقال(Transition): انتقال محصول به جامعه كاربران**

فرسنگ‌شمارها (Milestones) عبارتند از نقاطي در زمان كه در آن بر اساس معيارهاي دقيق بايد معين نماييم كه آيا مرحله قبل موفق بوده يا نه .

يك تكرار (Iteration) عبارتست از يك گذر كامل از همه نظمها كه منتهي به يك نشرداخلي يا خارجي مي گردد.


time
Inception
Elaboration
Construction
Transition
Preliminary Iterations
Architect. Iteration
Architect. Iteration
Devlop. Iteration
Devlop. Iteration
Devlop. Iteration
Transition Iteration
Transition Iteration
Minor Milestones: Releases
BM
R
A/ D
C
T
D
One Iteration
Inception
Elaboration
Construction
Transition
Evolution
time
An Initial Development Cycle
Generation 1
Inception
Elaboration
Construction
Transition
Evolution
time
The Next Evolution Cycle
Generation 2

فاز آغازين Inception phase

هدف اصلي اين فاز بررسي امكان انجام پروژه از نقطه نظر اقتصادي بوده سپس اطمينان از توافق همه سهامداران روي صورت مسأله(پروژه) و اهداف آن است.

اهداف فاز آغازين:

- تعيين محدوده سيستم نرم افزاري، شرايط مرزي و شرايط ارزيابي تكرارهاي اين فاز
- شناخت موارد كاربري مهم وحياتي سيستم
- بدست آوردن يك معماري اوليه
- تشخيص زود هنگام خطرات احتمالي
- برآوردن تقريبي هزينه، زمان، و سودآوري پروژه
- برنامه ريزي براي فاز بعدي

فعاليتهاي فاز آغازين:

- تشخيص محدوده پروژه كه با تعيين مهمترين نيازمنديهاي سيستم و شناخت محدوديتهاي موجود انجام مي پذيرد. بدين صورت مي توان به معيارهاي ارزيابي محصول نهائي دست يافت
- تهيه و آماده كردن مستند مورد كاري وارزيابي جايگزين هاي موجود براي مديريت ريسك، استخدام نيروي انساني و برنامه ريزي پروژه است. همچنين موازنه بين فاكتورهاي گوناگون مؤثر در توليد سيستم مانند هزينه و زمان مورد نياز پروژه و سودآوري سيستم از جمله فعاليتهاي اين فاز است
- ارزيابي معماري پيشنهادي و جايگزين هاي موجود براي طراحي است، كه تصميم گيري درمورد خريدن/توليد/استفاده مجدد از منابع موجود را نيز دربردارد. بدين صورت زمان و منابع مورد نياز به صورت واقعيتر قابل پيش بيني هستند

فرآورده‌هاي فاز آغازين:

- مستند ديدگاه: اين مستند يك ديد اوليه و كلي، با يك نگرش فني، درباره نيازمنديهاي اصلي، ويژگي هاي كليدي و محدوديت هاي اساسي سيستم را به توسعه دهندگان مي دهد.
- مدل موارد كاربري: اين مدل دربردارنده همه موارد كاربري و عواملي كه در اين فاز قابل تشخيصند.
- فهرست اوليه: اين فهرست شامل اصطلاحات مهمي كه در پروژه استفاده مي‌شود همراه تعريف دقيق آن است
- مورد كاري ابتدائي:
  - شرح محيط شغلي
  - فاكتورهاي موفقيت(برآورد درآمد، شناخت بازار،...)
  - برآورد هزينه هاي مالي
  - پيش بيني ابتدائي ريسك هاي احتمالي
  - برنامه ريزي براي پروژه و زمانبندي آن(زمان شروع فازهاي چهارگانه و تكرارهاي آنها)
- تخمين منابع مورد نياز در اين فاز يا كل پروژه است

فرآورده‌هاي انتخابي فاز آغازين :

- مدل دامنه (Domain Model) كه از فهرست انعطاف پذيري بيشتري دارد
- مدل كاري (Business Model) كه فرآيندها كاري سيستم را بيان مي نمايد
- نمونه‌هايي (Prototypes) از سيستم مورد نظر

فرسنگ شمار : **Life-Cycle Objective**

- توافق سهامداران روي تعيين محدوده سيستم و برآوردهاي انجام شده روي زمان و هزينه مورد نياز.
- پايداري موارد كاربري تشخيص شده در اين فاز است. اين پايداري معياريست براي كيفيت درك نيازمنديهاي سيستم تلقي مي گردد
- منطقي بودن برآورد هزينه ها، زمان مورد نياز و ريسك ها آيا نقشه فاز ساخت حاوي جزئيات كافي است؟

ناتواني پروژه از گذشتن از اين فرسنگ شمار به معني عدم موفقيت آن خواهد بود

فاز تشريح Elaboration phase

هدف اصلي اين فاز تحليل دامنه مسأله، بدست آوردن معماري مناسب و مستحكم براي سيستم، توسعه نقشه پروژه و جلوگيري از ريسك هاي حياتي سيستم است .

در كل اين فاز اين قانون را در نظر داشته باشيد : كه طول كيلومتري ولي عمق سانتي متري يعني در اين فاز تا آنجا كه امكان دارد تمام موارد را پوشش دهيد ولي با جزئيات و عمق كم

اهداف فاز تشريح:

- بدست آوردن يك معماري بنيادي (Architecture Baseline) مناسب و پايا بوده به طوريكه زمينه اصلي و نقطه شروع توسعه سيستم در فازهاي بعدي باشد
- بدست آوردن يك ديدگاه مناسب كه بعنوان ديدگاه بنيادي عمل مي نمايد
- بدست آوردن يك نقشه پايا(بنيادي) براي توسعه فاز ساخت
- نشان دادن اين است كه معماري بنيادي قدرت پشتيباني از ديدگاه بدست آمده با هزينه و زمان مناسب داراست

فعاليتهاي فاز تشريح:

- توسعه و بدست آوردن جزئيات ديدگاه.
- مشخص نمودن محيط هاي توسعه مورد نياز و جايگاه ابزارهاي CASE در خودكار سازي فرآيند توليد.
- توسعه معماري و انتخاب مؤلفه هاي لازم است. مؤلفه هاي در دسترس ارزيابي مي شوند و درباره ساختن/خريدن/استفاده مجدد از مؤلفه هاي مورد نياز، تصميم گيري هاي لازم اتخاذ مي شوند و بدين صورت مي توان هزينه و زمان مورد نياز فاز بعدي(ساختن) پيش بيني و براي آن برنامه ريزي مناسبي نمود .

فرآورده‌هاي فاز تشريح:

- مدل موارد كاربري(حد اقل بايد ۸۰% آن كامل باشد) كه در آن بيشتر موارد كاربري سيستم و عوامل آن، شناسائي و مستند شده باشند .
- نيازمنديهاي تكميلي (Supplementary Requirement) كه شامل نيازهاي غير وظيفه‌مندي و نيازمنديهايي كه به يك مورد كاربري معيني انتساب داده نشده اند
- توصيف معماري سيستم
- نمونه(آزمايشگاهي) از يك معماري قابل اجرا
- فهرست ريسك هاي و موارد كاري بازبيني شده
- نقشه مفصل توسعه كل پروژه

فرسنگ شمار : Life-Cycle Architecture

- آيا به يك ديدگاه پايا (Stable Vision) رسيده‌ايم؟
- آيا معماري بدست آمده پايدار است؟
- آيا نمونه هاي اجرائي ساخته شده نشان مي دهند كه ريسك هاي اصلي به خوبي شناخته و راه مقابله با آن مشخص شده است؟
- آيا نقشه فاز ساخت حاوي جزئيات كافي است؟

- آيا همه سهامداران بر توانايي دستيابي به ديدگاه مورد نظر بوسيله اجراي دقيق نقشه فعلي و با توجه به معماري فعلي اتفاق نظر دارند؟
- آيا مصرف حقيقي منابع، تا به حال، با مصرف پيش بيني شده سازگار است؟

اگر پروژه نتواند از اين فرسنگ شمار بگذرد يا اجراي آن بايد قطع گردد يا دربارة آن بايد تجديد نظر نمود.

فاز ساخت Construction phase

فاز ساخت، از يك نگاه، عبارتست از فرآيند توليد صنعتي (Manufacturing) كه در آن روي مديريت منابع، كنترل عمليات، بهينه سازي هزينه‌ها، زمانبندي و كيفيت تاكيد مي شود.

اهداف فاز ساخت:

- به حد اقل رساندن هزينه هاي توليد بوسيله بهينه سازي استفاده از منابع و ناديده گرفتن بعضي از كارهاي تكراري و غير مهم
- بدست آوردن يك كيفيت عالي در سريعترين زمان عملي ممكن
- رسيدن به نسخه هاي قابل استفاده عملي كاربران (آلفا، بتا، ...) در سريعترين زمان ممكن
- مديريت منابع و كنترل آن و همچنين بهينه سازي فرآيند توليد.
- تكميل توسعه مؤلفه ها و انجام آزمايش هاي گوناگون با توجه شرايط ارزيابي (Evaluation Criteria).
- ارزيابي نشرها در مقايسه با ديدگاه مطلوب(همان شرايط ارزيابي).

فرآورده‌هاي فاز ساخت:

- محصول نهائي نرم افزار.
- دفترچه راهنماي كاربران.
- توصيف نشرهاي فعلي.

فرسنگ شمار : **Initial Operational Capability**

- **آيا نشر محصول به اندازه كافي محكم و پايدار است كه براي استفاده بوسيله كاربران آماده باشد؟**
- **آيا هزينه واقعي منابع با هزينه پيش بيني شده هنوز سازگار است؟**

فاز انتقال Transition phase

هدف اصلي اين فاز عملياتي كردن نرمافزار يا انتقال آن به جامعه كاربران است .

اهداف فاز انتقال:

- انتقال نرم افزار به محيط كاربران و گرفتن نظرات آنها در مورد نحوه عملكرد سيستم جديد
- بدست آوردن توافق همه سهامداران درباره كامل بودن Deployment Baseline و سازگار بودن آن با شرايط ارزيابي ديدگاه
- بدست آوردن Product Baseline نهائي درسريعترين زمان و با كمترين هزينه ممكن
- انجام جنبه هاي مهندسي مربوط به استقرار كه شامل بسته بندي و نصب محصول .
- انجام فعاليت هاي بهينه سازي مانند اصلاح خطاها و سرعت بخشيدن به اجراي برنامه
- انجام آزمايش بتا براي آزمايش سيستم و ارزيابي نتايج اين آزمايش با توجه به عملكرد مورد انتظار كاربران
- آماده سازي مستندات ، آموزش كاربران و آماده پاسخگويي و پشتيباني از آنها
- اجراي هر دو سيستم، قديمي و جديد با هم به صورت موازي، براي مدتي از زمان، براي مقايسه عملكرد اين دو سيستم

فرآورده‌هاي فاز انتقال:

- ***تكميل دفترچه راهنماي كاربران.***
- ***تكميل دفترچه نصب و نگهداري.***
- ***مستند Release Notes: اطلاعات مربوط به اشكالات برنامه، شماره نسخه فعلي***

فرسنگ شمار: Product Release

- ***آيا كاربر راضي است؟***
- ***هزينه هاي پيش بيني شده با هزينه هاي واقعي چه تفاوتي دارند؟***

آشنايي نظم مدلسازي كاري Business Modeling Discipline

چه موقع از مدلسازي كاري استفاده كنيم؟ معمولا در پروژه هايي :

- ✓ كه كاربران زيادي دارند
- ✓ يا حجم زيادي از داده‌ها بايد پردازش گردد

يكي از مزاياي استفاده ازتكنيكهاي مشترك در مهندسي نرمافزار و مدلسازي كاري، تسهيل درك ارتباط بين اين دو فعاليت است. همچنين نگاشت فراورده‌هاي مدلسازي كاري به فراورده‌هاي مدلسازي نرمافزار مشكل نخواهد بود.

مروری بر UML (Unified Modeling Language)

زبان مدلسازی یکنواخت :

زبان مدلسازی یکنواخت یا(Unified Modeling Language (UML ، یک زبان مدلسازی است که برای تحلیل وطراحی سیستمهای شی‌گرا بکار می‌رود UML .اولین بار توسط شرکت Rationalارائه شد و پس از آن از طرف بسیاری از شرکت‌های کامپیوتری و مجامع صنعتی و نرمافزاری دنیا مورد حمایت قرار گرفت؛ به طوریکه تنها پس از یک سال، توسط گروه Object Management Group، به عنوان زبان مدلسازی استاندارد پذیرفته شد UML .تواناییها و خصوصیات بارز فراوانی دارد که می‌تواند به طور گسترده‌ای در تولید نرمافزار استفاده گردد. در ادامهٔ این مقاله ابتدا به تاریخچهٔ UML و در ادامه به معرفی، ویژگیها و نمودارهای آن پرداخته می‌شود و در پایان، روند حرکت به سمت UML و اهمیت آن برای ایران، بررسی خواهد شد .

تاریخچهٔ UML :

دیدگاه شی‌گرایی (Object Oriented) از اواسط دهه ۱۹۷۰ تا اواخر دهه ۱۹۸۰ در حال مطرح شدن بود. در این دوران تلاشهای زیادی برای ایجاد روشهای تحلیل و طراحی شی‌گرا صورت پذیرفت. در نتیجهٔ این تلاشها بود که در طول ۵ سال یعنی ۱۹۸۹ تا ۱۹۹٤، تعداد متدولوژیهای شی‌گرا از کمتر از ۱۰ متدولوژی به بیش از ۵۰ متدولوژی رسید. تکثر متدولوژیها و زبانهای شی‌گرایی و رقابت بین اینها به حدی بود که این دوران به عنوان" جنگ متدولوژیها" لقب گرفت. از جمله متدولوژیهای پرکاربرد آن زمان می‌توان ازBooch ، OOSE، OMT، Fusion، Coad-Yourdan، Shlayer-Mellorوغیره نام برد. فراوانی و اشباع متدولوژیها و روشهای شی‌گرایی و نیز نبودن یک زبان مدلسازی استاندارد، باعث مشکلات فراوانی شده بود. از یک طرف کاربران از متدولوژیهای موجود خسته شده بودند، زیرا مجبور بودند از میان روشهای مختلف شبیه به هم که تفاوت کمی در قدرت و قابلیت داشتند یکی را انتخاب کنند. بسیاری از این روشها، مفاهیم مشترک شی‌گرایی را در قالبهای مختلف بیان می‌کردند که این واگرایی و نبودن توافق میان این زبانها، کاربران تازه‌کار را از دنیای شی‌گرایی زده می‌کرد و آنها را از این حیطه دور می‌ساخت. عدم وجود یک زبان استاندارد، برای فروشندگان محصولات نرمافزاری نیز مشکلات زیادی ایجاد کرده بود.
اولین تلاشهای استانداردسازی از اکتبر ۱۹۹٤ آغاز شد، زمانی که آقای Rumbaurgh صاحب متدولوژی OMT به آقای Booch در شرکت Rational پیوست و این دو با ترکیب متدولوژیهای خود، اولین محصول ترکیبی خود به نام" روش یکنواخت" را ارائه دادند. در سال ۱۹۹۵ بود که با اضافه شدن آقای Jacobson به این دو، روش یکنواخت ارائه شده با روش OOSE نیز ترکیب شد واین خود سبب ارائهٔ UML نسخهٔ ۰,۹ در سال ۱۹۹٦ گردید. سپس این محصول به

شرکتهای مختلفی در سراسر جهان به صورت رایگان ارائه شد و استقبال شدید شرکت‌ها از این محصول و تبلیغات گسترده شرکتRational ، سبب آن شد که گروهOMG ، نسخة ۱,۰ UMLرا به عنوان زبان مدلسازی استاندارد خود بپذیرد. تلاشهای تکمیلی UML استاندارد ادامه پیدا کرد و نسخة ۱,۱ آن در سال 1997 و نسخه ۱,۳ آن در سال ۱۹۹۹ ارائه گردید .

اهداف UML :

۱ - Visualization : تصویری روشن و شفاف از مدل به ما می دهد. این زبان به علت ترسیمی بودن ، قدرت زیادی داردو به راحتی توسط همه افراد قابل فهم است.

۲ - Construction : به ما کمک می کند چارچوب و معماری مدل را مشخص کنیم.

۳ – Specification : به ما کمک می کند اجزای مدل را مشخص کنیم.

۴ - Documentation : به ما کمک می کند برای مدل خود مستند سازی کنیم.

در حقیقت همان چهار نتیجه مدلسازی را برای ما فراهم می کند.

چرا مدل و مدلسازی؟

ایجاد یک مدل برای سیستمهای نرمافزاری قبل از ساخت یا بازساخت آن، به اندازه داشتن نقشه برای ساختن یک ساختمان ضروری و حیاتی است. بسیاری از شاخه‌های مهندسی، توصیف چگونگی محصولاتی که باید ساخته شوند را ترسیم می‌کنند و همچنین دقت زیادی می‌کنند که محصولاتشان طبق این مدلها و توصیفها ساخته شوند .مدلهای خوب و دقیق در برقراری یک ارتباط کامل بین افراد پروژه، نقش زیادی می‌توانند داشته باشند. شاید علت مدل کردن سیستمهای پیچیده این باشد که تمامی آن را نمی‌توان یکباره مجسم کرد، بنابراین برای فهم کامل سیستم و یافتن و نمایش ارتباط بین قسمتهای مختلف آن، به مدلسازی می‌پردازیم UML .زبانی است برای مدلسازی یا ایجاد نقشه تولید نرمافزار .
به عبارت دیگر، یک زبان، با ارائه یک فرهنگ لغات ویک مجموعه قواعد، امکان می‌دهد که با ترکیب کلمات این فرهنگ لغات و ساختن جملات، با یکدیگر ارتباط برقرار کنیم. یک زبان مدلسازی، زبانی است که فرهنگ لغات و قواعد آن بر نمایش فیزیکی و مفهومی آن سیستم متمرکزند. برای سیستمهای نرمافزاری نیاز به یک زبان مدلسازی داریم که بتواند دیدهای مختلف معماری سیستم را در طول چرخة تولید آن، مدل کند .
فرهنگ واژگان و قواعد زبانی مثل UML به شما می‌گویند که چگونه یک مدل را بسازید و یا چگونه یک مدل را بخوانید. اما به شما نمی‌گویند که در چه زمانی، چه مدلی را ایجاد کنید. یعنی UML فقط یک زبان نمادگذاری (Notation) است نه یک متدولوژی. یک زبان نمادگذاری شامل نحوة ایجاد و نحوة خواندن یک مدل می‌باشد، اما یک متدولوژی بیان می‌کند که چه محصولاتی باید در چه زمانی تولید شوند و چه کارهایی با چه ترتیبی توسط چه کسانی، با چه هزینه‌ای، در چه مدتی و با چه ریسکی انجام شوند.

ویژگیهای : UML

UMLدارای ویژگیهای بارز فراوانی است که در این قسمت به آنها می‌پردازیم.

UML یک زبان مدلسازی است اما چیزی فراتر از چند نماد گرافیکی است. بطوریکه در ورای این نمادها، یک سمانتیک) معناشناسی) قوی وجود دارد، بطوریکه یک تولیدکننده می‌تواند مدلهایی تولید کند که تولیدکننده‌های دیگر و یا حتی یک ماشین آن را بخواند و بفهمد. بنابراین یکی دیگر از نقش‌های مهم" UML تسهیل ارتباط" بین اعضای پروژه و یا بین تولیدکنندگان مختلف می‌باشد. این ارتباط بسیار مهم است. شاید دلیل اصلی اینکه تولید نرمافزار به صورت فریبنده‌ای دشوار است، همین عدم ارتباط مناسب بین اعضای پروژه باشد و اگر در تولید نرمافزار، بین اعضای پروژه گزارشهای هفتگی و مداوم وجود داشته باشد، بسیاری از این دشواریها برطرف خواهد شد .

البته این را هم باید در نظر گرفت که UML کمی پیچیده است و این به خاطر آن است که سعی شده است نمودارهایی فراهم شود که در هر موقعیتی و با هر ترتیبی قابل استفاده باشند. دلیل دیگر پیچیدگی از آنجا ناشی می‌شود که UML ترکیبی است از زبانهای مختلف، که برای حفظ سازگاری و جمع کردن خصوصیات مثبت آنها، ناگزیر از پذیرش این پیچیدگی می‌باشد .

UML موفقیت طرح را تضمین نمی‌کند، اما در عین حال خیلی چیزها را بهبود می‌بخشد. به عنوان مثال استفاده ازUML ، تا حد زیادی، هزینه‌های ثابتی نظیر آموزش و استفاده مجدد از ابزارها را در هنگام ایجاد تغییر در سازمان و طرحها کاهش می‌دهد.

مساله دیگر اینکه، UMLیک زبان برنامه‌نویسی بصری (visual) نیست، اما مدلهای آن را می‌توان مستقیماً به انواع زبانهای مختلف ارتباط داد. یعنی امکان نگاشت از مدلهای UML به کد زبانهای برنامه‌نویسی مثل Java و VC++ وجود دارد که به این عمل "مهندسی روبه‌جلو" می‌گویند .عکس این عمل نیز ممکن است؛ یعنی این امکان وجود دارد که شما بتوانید از کد یک برنامه زبانی شی‌گرا، مدلهای UML معادل آن را بدست آورید. به این عمل "مهندسی

معکوس" مـی‌گوینـد. مهندسـی روبـه‌جلـو و معکـوس از مهمتـرین قابلیتهـای UML بـه شـمار می‌روند، البته نیاز به ابزار Case مناسبی دارید که از این مفاهیم پشتیبانی‌کنند .

اگر با زبانهای مدلسازی دیگر کـار کـرده باشـید، بـرای کـار بـا UML مـشکل چنـدانی نخواهیـد داشت. امـا بـرای شـروع کـار بـا UML بـه عنـوان اولـین زبـان مدلـسازی، بهتـر اسـت فقـط بـا نمودارهای خاصی کار کنید. برای این کار بهتر است ابتدا بـا نمودارهـای مـورد کـاربرد و تعامـل کار کنید و پس از مـدتی کـار و آشـنا شـدن بـا ویژگیهـای اولیـة آن، بـه یـادگیری و اسـتفاده از نمودارها واجزای دیگر بپردازید. در مقایسه با زبانهای مدلسازی دیگر مثل ERو زبان فلوچارتی DR، زبان UML نمودارهـای قـویتر و قابـل‌فهمتـری را ارائـه مـی‌دهدکـه شـامل تمـامی مراحـل چرخة حیات تولید نرمافزار (تحلیل، طراحی، پیاده‌سازی و تست) می‌شود .

یکی دیگر از ویژگیهای مهم UML این است که مستقل از متدولوژی یـا فراینـد تولیـد نـرمافـزار می‌باشد و این بدان معنـی اسـت کـه شـما بـرای اسـتفاده ازUML ، نیـاز بـه اسـتفاده از یـک متدولوژی خاص ندارید و می‌توانید طبق متدولوژی‌های قبلی خود عمل کنید با این تفـاوت کـه مدلهایتان را با UML نمایش مـی‌دهیـد. البتـه مـستقل‌بـودن از متـدولوژی و فراینـد تولیـد، یـک مزیت برای UML می‌باشد؛ زیرا بسیاری از انواع پروژه‌ها و سیستمها نیاز به متـدولوژی خـاص خود دارند. اگر UML در پی پیاده کردن همة اینها بر مـی‌آمـد، یـا بـسیار پیچیـده مـی‌شـد و یـا اسـتفاده خـود را محـدود مـی‌کـرد. البتـه متـدولوژیهایی براسـاس UML در حـال شـکل‌گیـری می‌باشند.

از دیگـر ویژگیهـای UML مـی‌تـوان بـه پـشتیبانی از مفـاهیم سـطح بـالای شـی‌گرایـی مثـل Collaboration، Framework، Pattern و Component اشاره کرد. همچنـین UML بـا اسـتفاده از یک سری مکانیزمهای گسترش‌پذیر امکان می‌دهد که بتـوان زبانهـای مدلـسازی جدیـدتری (با گسترش مفاهیم پایه‌ای موجود) ایجادکرد .

روند حرکت به سمت UML در جهان :

قبل از ارائهUML ، زبان مدلسازی استانداردی وجود نداشت و استفاده‌کنندگان مجبور بودند از میان زبانهای مختلف موجود که هیچیک تقریباً کامل نبودند و تفاوتهایی با هم داشتند، یکی را انتخاب کنند. تفاوتهای زبانهای مدلسازی، چندان قدرت مدلسازی را افزایش نداده بود، اما در عوض باعث افول صنعت شی‌گرایی و سردرگمی کاربران شده بود. در چنین شرایطی طبیعی بود که استقبال زیادی از یک زبان مدلسازی استاندارد که ویژگیهای بارز زیادی داشت، بشود. بسیاری از شرکتها در همان اوایل کار به UML روی آوردند و تعداد دیگری نیز پس از تثبیتUML ، آن را به عنوان استراتژی تولید ومستندسازی خود پذیرفتند .

OMGکه کنسرسیومی است متشکل از ۷۰۰ شرکت معتبر آمریکا، از UML حمایت کرد و آن را به عنوان زبان مدلسازی استاندارد خود اعلام کرد. البته علاوه بر استاندارد شدن، حمایت جداگانه شرکت‌های بزرگ دنیا مثلHewlett-Packard ، I-Logix، Microsoft،IBM ، Oracleو بسیاری دیگر، خود سبب افزایش کاربرد آن در محافل صنعتی و نرمافزاری دنیا گردید. امروزه نیز با ارائه نسخه ۱,۳ و رفع مشکلات گذشته، روز به روز بر کاربران آن افزوده می‌شود .

روند حرکت به سمت UML در ایران :

در ایران حرکت برخی شرکتها به سمت UML سریعتر انجام شد؛ بطوریکه در همان زمان استاندارد شدن UML در سال ۱۹۹۷، شرکتهایی در ایران، این ابزار را به عنوان استاندارد خود پذیرفتند و از آن در تولید محصولات خود استفاده کردند .

یکی از مشکلات پذیرش این زبان در ایران، مقاومتهایی است که در رابطه با خود شی‌گرایی مطرح می‌شود. البته نظیر این مقاومتها در دنیا نیز وجود داشت و سرو صداهای بسیاری را سبب شد. اما تا قبل از ظهور UML و با ارائه متدهای فراوان شی‌گرایی، این مشکل تا حدودی حل شده بود . با توجه به روند حرکت شتابان به سمت UML در دنیا و با توجه به اهمیت استانداردسازی برای صنعت نرمافزار کشور، حرکت هرچه‌سریعتر به سوی این فناوری در کشور توصیه می‌شود .

اهمیت ترویج UML در کشور :

در کشور ما شرکتهای نرمافزاری که روشهای علمی طراحی و مهندسی نرمافزار را به کار برند بسیار کمیاب هستند. در واقع رقابت بین شرکتها بیشتر بر سر کاهش قیمت است و نه بهبود کیفیت. ممکن است تصور شود عامل اصلی بروز این مشکل، فرهنگ مصرف غلط در کشور است و لذا حل مشکل نیز به دست مصرف‌کنندگان است. اما بایستی از خود پرسید که مصرف‌کنندگان چگونه خواهند توانست یک محصول نرمافزاری را ارزیابی کرده و انتظارات

خود را به طور دقیق مطرح نمایند؟ در این زمینه دولتها وظایف مهمی دارند و می‌توانند ابزارهای لازم را فراهم نمایند .
هرچند UML یک استاندارد برای تشخیص کیفیت نرمافزارها نیست ولی استانداردی برای مدلسازی نرمافزار است ولذا مراحل مختلف تعریف، طراحی و حتی تست نرمافزار را تسهیل نموده و کار تیمی و ارزیابی ناظران خارجی را آسان و ممکن می‌نماید. اگر استفاده از UML در تولید نرمافزار در کشور به یک فرهنگ تبدیل گردد، گام بزرگی به سوی دقت، کیفیت، مستندسازی و رعایت اصول مهندسی نرمافزار برداشته شده است.

نمودارهای UML :

در این بخش به معرفی نمودارهای UML می‌پردازیم وعلاقمندان به آشنایی بیشتر را، دعوت به مطالعه مراجع معرفی شده، می‌نماییم :

نمودار کلاس :(Class Diagram)

این نمودار،کلاسها، واسطها و همکاری و روابط بین آنها را نمایش می‌دهد. و نمودار اصلی و مرکزی UML می‌باشد. که بیان‌کننده ساختار ایستای سیستم نرمافزاری می‌باشد .

نمودار اشیاء :(Object Diagram)

این نمودار، اشیاء سیستم و روابط بین آنها را نمایش می‌دهد. در واقع یک تصویر لحظه‌ای از نمودار کلاس می‌باشد .

نمودار موردکاربرد :(Usercase Diagram)

این نمودار، تعامل کاربران خارجی و سیستم را مدل می‌کند و از جهاتی شبیه نمودار سطح صفر DFD می‌باشد که جنبه‌های رفتاری سیستم را نمایش می‌دهد. این نمودار نقطه ورودی برای تمامی نمودارهای دیگری است که به تشریح نیازمندیها و معماری و پیاده‌سازی سیستم می‌پردازند .

نمودارهای تعامل :(Interaction Diagram)

این نمودارها، بیان کننده تعامل هستند که شامل اشیاء مختلف و روابط بین آنها و همچنین پیغامهایی که بینشان رد و بدل می‌شود می‌باشند. این نمودارها جنبه‌های پویای یک سیستم را مدل می‌کنند و خود بر دو نوعند: نمودار توالی (Sequence Diagram)که ترتیب زمانی تعامل‌ها را نشان می‌دهد و نمودار همکاری (Collaboration Diagram)که تاکید بر نمایش ساختاری تعامل‌ها دارد .

نمودارحالت :(Statechart Diagram)

این نمودار، بیان‌کننده جنبه‌های رفتاری سیستم می‌باشد و در واقع توصیف رسمی یک کلاس بوده که شامل حالات، انتقال بین حالات، رخدادها و فعالیتها می‌باشد. از این نمودارها برای نمایش دادن چرخه حیات اشیاء یک کلاس خاص نیز می‌توان استفاده کرد .

نمودار فعالیت :(Activity Diagram)

این نمودار، نوع خاصی است از نمودار حالت، که انتقال جریان از یک فعالیت به فعالیت دیگر را نمایش می‌دهد. این نمودار جنبه‌های پویای یک سیستم را نمایش می‌دهد. در واقع حالات این نمودار، گامهای ترتیبی انجام یک عمل را نمایش می‌دهند .

نمودار اجزاء :(Component Diagram)

از جمله نمودارهای پیاده‌سازی می‌باشد و سازماندهی و روابط بین مجموعه‌ای از اجزاء را نمایش می‌دهد. این نمودار، جنبه های ایستای پیاده‌سازی یک سیستم را مدل می‌کند .

نمودار به‌کارگماری :(Deployment Diagram)

پیکربندی گره‌های پردازشی زمان اجرا را نمایش می‌دهد. که برای مدل کردن جنبه‌های ایستای به‌کارگماری یک معماری بکار می‌رود. همچنین نمایش‌دهندة اجزای استفاده‌شده زمان اجرا مثل کتابخانه‌هایDLL ، فایل‌های اجرایی، کدهای مبدا و روابط بین آنها می‌باشد . البته این نمودارها تمام نمودارهای UML نیستند بلکه بسته به نیاز و با کمک ابزارهای Case میتوان نمودارهای دیگری نیز تعریف و استفاده کرد .

(سناریو)

تهیه سناریو اولین قدمی است که برای انجام تجزیه و تحلیل سیستم برداشته می شود .
در سناریو ما شمای کلی سیستم و نحوه کار را توضیح می دهیم
این سناریو از مجموعه دروس آموزش رشنال رز می باشد که در آن به این نکته اشاره شده که یک سناریو دارای چه بخشهایی باید باشد :

Precondition : (شرایطی که باید ایجاد شود تا Use case فعال شود)

هنگامی که پرسنل جهت مراجعه یا ترک محل کار ورود و خروج می نمایند .

Postcondition : (شرایطی که بعد از اتمام کار Use case ایجاد می شود)

باید میزان حضور پرسنل در محل کار محاسبه شود .

Goal : (هدف)

محاسبه میزان عملکرد پرسنل در شرکت مورد نظر .

Describtion : (شرح مختصری از فعالیتهای سیستم)

پرسنل جهت محاسبه میزان عملکرد ، ساعت ورود و خروج خود را ثبت می کنند تا بر اساس آن حقوق و مزایا دریافت کنند .

Main Flow : (جریان اصلی کار Use case)

۱) پرسنل هنگام مراجعه به محل کار کد پرسنلی خود را در اختیار مسئول قسمت اداری قرار می دهند .
۲) مسئول قسمت اداری به منوی برنامه رفته و ورود پرسنل مورد نظر را ثبت می کند .
۳) پرسنل هنگام ترک محل کار کد پرسنلی خود را دوباره در اختیار مسئول قسمت اداری قرار می دهند .
۴) مسئول قسمت اداری به منوی برنامه رفته و با ورود کد پرسنلی مشخصات + ساعت ورود پرسنل مورد نظر را مشاهده می کنند .
۵) مسئول قسمت اداری ساعت خروج را ثبت و سیستم میزان عملکرد پرسنل در آن روز را محاسبه می نماید .

Altrnative Flow : (جریان فرعی کار Use case)

۱) امکان دارد پرسنل در یک روز مرخصی ساعتی یا در ماموریت باشد آنگاه دیرتر در محل کار حاضر شود یا اینکه زودتر محل کار را ترک کند

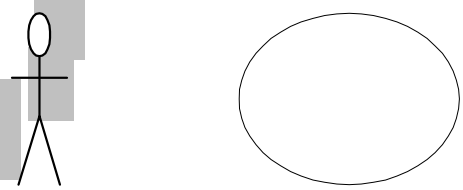
۲) امکان دارد پرسنل در یک روز مرخصی یا در ماموریت روزانه بوده و آن روز به محل کار مراجعه نکند .

(UseCase view)

در این بخش به معرفی UseCase view ، اولین view از view های Rational Rose که در سمت چپ این نرم افزار در زمان اجرا قرار دارد می پردازیم .

برای اینکه ما با UseCase view کار کنیم و به راحتی آن را درک کنیم باید سه مفهوم را بدانیم . مفهوم اول خود UseCase است ، دومین مفهوم ، مفهوم سناریو و سومین مفهوم Actor است که هر یک را توضیح خواهیم داد .

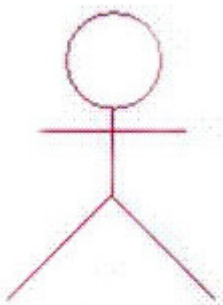
UseCase : UseCase در واقع عبارت است از هر سرویسی که سیستم در اختیار کاربران قرار می دهد . شکل آن در نمودارهای UML مانند شکل زیر است .



Use Case

سناریو : سناریو در واقع متنی است که فعالیتهای UseCase را بطور کامل شرح می دهد . هر سناریو دارای استانداردهایی است که آنها را در بخش قبل(سناریو) با مثال توضیح داده ایم .

Actor : هر کس که با UseCase کار می کند یک Actor است . Actor ها کسانی هستند که اطلاعاتی از UseCase دریافت می کنند و یا اینکه اطلاعاتی به آن تزریق می نمایند . شکل Actor در نمودارهای UML مانند شکل زیر است .



Actor1

در واقع ما در UseCase دیاگرام عمدتاً با سناریوی مربوط به به هر UseCase و Actor های مرتبط با آن سرو کار خواهیم داشت .

هدف نمودار UseCase چیست ؟

بوسيله نمودار UseCase كاربران و خبرگان دامنه‌كاري (Business Domain Experts) سيستم مي‌توانند همراه توسعه دهندگان سيستم به درك مشتركي از سيستم مطلوب برسند.

ما در نمودارهای UseCase به دنبال نیازمندیهای کاربران هستیم پس ابتدا باید کاربران را تشخیص دهیم در قدم بعد باید بدانیم هر کاربر از سیستم چه می خواهد .

در UseCase دیاگرام هدف ما مستند سازی همه آن چیزی است که سیستم به کاربران خود ارائه می دهد .

در واقع يك Usecase بايد هدف را براي Actor را مشخص كند.

در واقع UseCase دیاگرام نقطه ورود یک سیستم است ، از آنجا مسئله تحلیل می شود و سپس در اختیار طراحان و برنامه نویسان قرار می گیرد.

UseCase دیاگرام در عین سادگی یک دید کلی نسبت به آن چه در سیستم انجام می شود را نشان می دهد به همین خاطر قابل درک برای کاربران خواهد بود . لذا تحلیل گر سیستم می تواند برای بیان شناخت خود از سیستم و انتقال مفاهیم به کاربران از این نمودار استفاده کنند . در نمودارهای UseCase ما بایدActor ها و UseCase ها و همچنین روابط مابین آنها را مشخص کنیم ، اینها در واقع سه عنصر تشکیل دهنده UseCase دیاگرام می باشند .

اهداف تهیه UseCase ها :

۱) با نگاه به UseCase ها میتوان فهمید چه عملیاتی توسط سیستم باید انجام شود
۲) تعیین محدوده سیستم در حال توسعه
۳) ابزاری هستند جهت تست سیستم
۴) مبنائی جهت تهیه راهنمای کاربران

هر یک از مستندات Use Case برای نمایش ضمیمه ها بکار می روند . در این بخش شرح کاملی از هر یک از بخشهای الگوی Use case آماده شده است .

تعیین هویت Use Case

- **شماره Use Case :** هر یک از Use Case ها باید توسط یک عدد صحیح بی همتا مشخص شود . برای شماره گذاری متناوب Use Case هایی که در یک گروه قرار دارند از شکل X:Y استفاده می شود .
- **نام Use Case :** توضیح مختصری برای نامی که برای Use Case انتخاب شده است . این نام ها وظیفه های مورد نیاز که برای انجام نیازهای سیستم که شامل یک کار یا اسمی که شبیه مثالهای زیر است را منعکس می کند .

۱) نمایش اطلاعات قسمت عددی

۲) نشانه گذاری دستی منابع و فرامتنها و ایجاد لینک به مقصد

۳) مکان یک دستورالعمل برای CD با بروز شدن نسخه نرم افزار

- **تاریخچه Use Case :**

۱) چه کسی Use Case را ایجاد کرده است .

۲) در چه تاریخی Use Case ایجاد شده است .

۳) آخرین بار توسط چه کسی به روز شده است .

۴) تاریخ آخرین به روز رسانی چه موقع بوده است .

Actor

Actor ها کسانی هستند که با سیستم کار می کنند ، از آن اطلاعات می گیرند و به آن اطلاعات می دهند . یک Actor در واقع کسی است که UseCase در جهت سرویس دادن به آن عمل می کند . معمولاً Actor ها از این بابت مهم می باشند که سیستم ساخته شده باید جوابگوی نیازهای آنها باشد . می توان گفت که شناسایی Actor ها اولین قدمی است که برای رسم UseCase دیاگرام برداشته می شود .

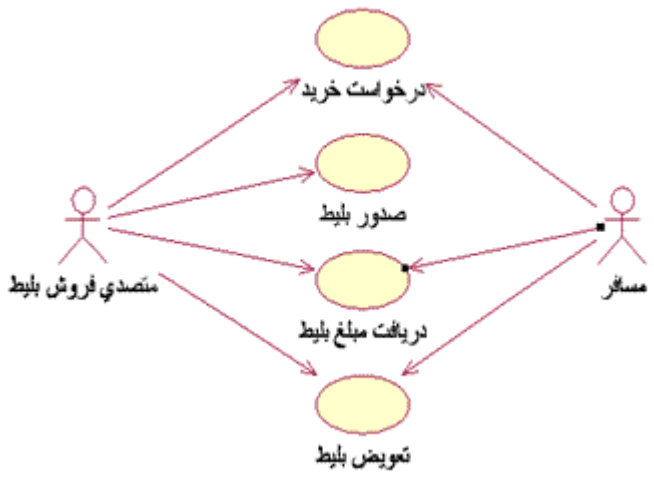
از دیدگاه کلی انواع Actor ها عبارتند از :

۱) Actor های اصلی
۲) Actor های فرعی

از دیدگاه دیگر می توان گفت که Actor ها عبارتند از :

۱) کاربران سیستم
۲) سیستمهای دیگر
۳) زمان

Actor های اصلی: Actor های اصلی معمولاً کسانی هستند که از سیستم اطلاعات می گیرند یا به آن اطلاعات تزریق می کنند . مثلاً در سیستم صدور بلیت ، متصدی فروش همان گونه که در تصویر مشاهده می کنید یک Actor اصلی است زیرا مستقیم پشت سیستم کامپیوتری نشسته و با درخواست بلیت ، صدور بلیت و تمامی UseCase های ارتباط مستقیم دارد و او فردی است که مستقیماً از طریق صفحه کلید اطلاعات مربوط به مشتریها را وارد سیستم می کند . در این گونه Actor ها بر ارتباط و رابطه مستقیم بسیار تاکید می شود .



Actor های فرعی : Actor هایی هستند که مستقیماً با سیستم در ارتباط نیستند ، ولی وجودشان برای فعال نگه داشتن سیستم ضروری می باشد . در مثال قبل مشتری یا مسافر یک Actor فرعی است زیرا تا زمانی که مسافری نباشد متصدی فروش بلیت به عنوان یک Actor اصلی نمی تواند اطلاعاتی وارد سیستم کند . همانگونه که گفته شد وجود مسافر به عنوان یک Actor فرعی جهت فعال نگه داشتن سیستم ضروری است .

مشخص نمودن اینکه کدام Actor اصلی و کدام Actor فرعی است از این بابت مورد توجه است که یک سیستم در نهایت ، جهت استفاده Actor های اصلی تهیه می شود و نیازمندیهای آنها بالاترین اولویت را برای تولید سیستم برای ما ایجاد می کند .

نکته : اولین قدم برای تهیه نیازمندیها سیستم شناسایی Actor ها می باشد زیرا اگر ما بدانیم Actor کیست و چه می خواهد به راحتی می توانیم UseCase های مورد نظر را استخراج کنیم .

با اتکا به مطالب بالا می توان گفت که کاربران سیستم جزء Actor های اصلی می باشند . همچنین سیستمهای دیگر می توانند برای سیستم ما یک Actor باشند مثلاً در سیستم محاسبه کارکرد پرسنل ، سیستم محاسبه حقوق می تواند به عنوان یک Actor باشد که از نتیجه سیستم محاسبه کارکرد پرسنل برای محاسبه حقوق استفاده می کند . و بالاخره از زمان می توان به عنوان یک Actor استفاده کرد ؛ مثل زمانبند سیستم عامل که ما مشخص می کنیم در یک زمان خاص عملی صورت بپذیرد .

چگونه Actor ها را بیابیم ؟

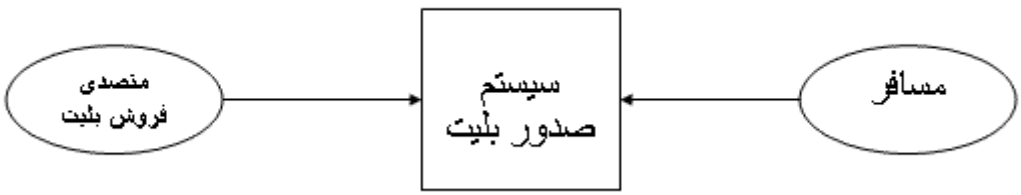
راههای متفاوتی برای تشخیص Actor ها وجود دارد که در زیر به چندتای آنها اشاره می کنیم .

- کسی که مستقیماً از سیستم استفاده می کند
- کسی که مسئول نگهداری سیستم است
- یک سخت افزار خارجی که با سیستم استفاده می شود
- سیستم دیگری که برای کار کردن نیازمند به این سیستم می باشد

علاوه بر موارد گفته شده در بالا ، رسم دیاگرام متن (Context Diagram) که از دیاگرامهای متدولوژی SSADM می باشد در این رابطه می تواند کمک زیادی به ما کند .

به این نکته توجه کنید که دیاگرام متن جزء دیاگرامهای استاندارد UML نیست ولی ما در جهت درک و شناسایی بهتر Actor ها از آن استفاده می کنیم.

به شکل زیر توجه کنید .



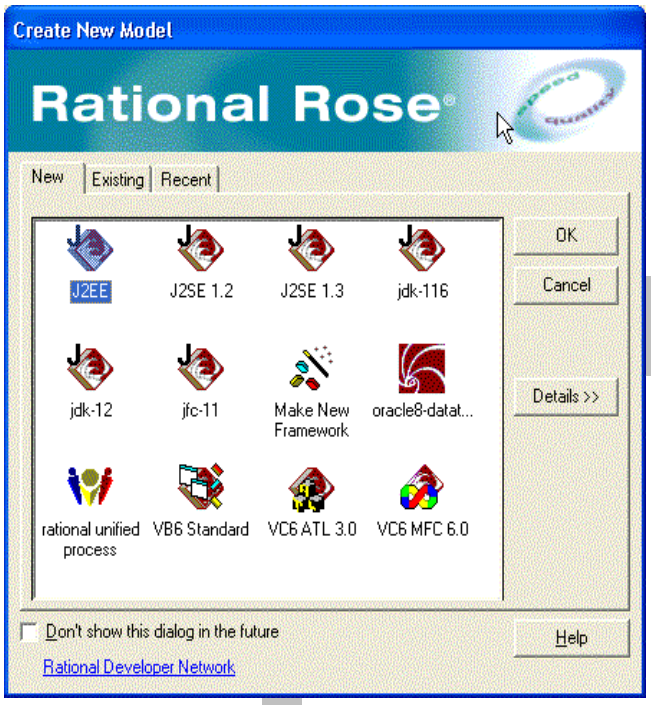
البته به این نکته توجه داشته باشید که شکل بالا ترکیبی از دیاگرام متن و دیاگرام متن منطقی می باشد (مباحث مربوط به متدولوژی SSADM) . حال برای درک دیاگرام بالا به توضیح آن می پردازیم .

ما سیستم صدور بلیت را بدون در نظر گرفتن جزئیات در وسط ، داخل مربع قرار داده ایم و بررسی می کنیم که این سیستم با چه سازمانها یا افرادی در ارتباط است . همانگونه که مشاهده می کنید سیستم فروش بلیت با مسافر و متصدی فروش بلیت در ارتباط است که به عنوان موجودیت خارجی در نظر گرفته و در درون بیضی قرار داده ایم . البته لازم به ذکر است ارتباط سیستم با موجودیتهای دیگر بسته به سطح انتزاع می باشد . مثلاً در یک سطح انتزاع دیگر از دیاگرام فروش بلیط ، سازمان بیمه و راهنمایی و رانندگی علاوه بر مسافر و متصدی فروش بلیط می توانند به عنوان موجودیتهای خارجی با سیستم در ارتباط

باشد . با توجه به دیاگرام متن بالا ما متوجه می شویم که سیستم را برای متصدی فروش بلیط تهیه می کنیم و مسافر برای بقای سیستم لازم می باشد پس متصدی فروش بلیط Actor اصلی و مسافر Actor فرعی می باشد .

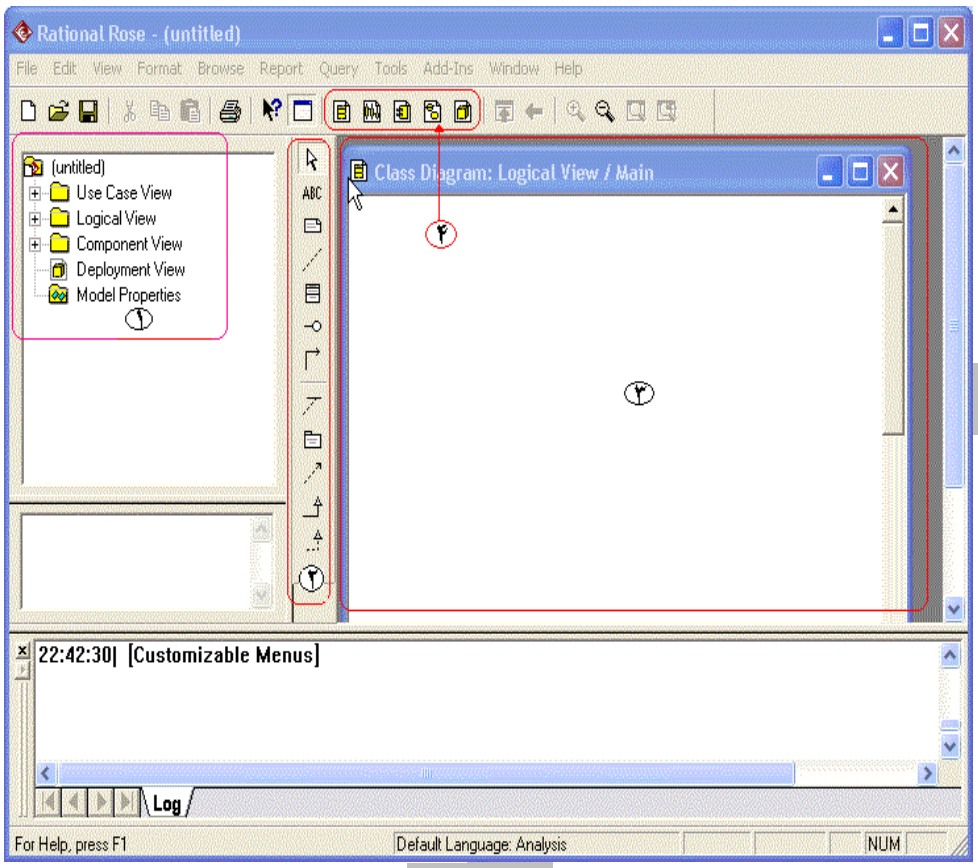
آشنایی با محیط(Rational Rose)

در این درس می خواهیم آشنایی مختصری با محیط نرم افزارRational Rose ۲۰۰۲ پیدا کنیم . وقتی که این نرم افزار را باز می کنیم ابتدا کادری شبیه شکل زیر مشاهده می کنید که حاوی انواع پروژها رشنال می باشد .



به دلیل اینکه در این قسمت قصد نداریم پروژه خاصی را مورد بررسی قرار دهیم بر روی گزینه cancel کلیک می کنیم . البته ما می توانیم بسته به نوع پروژه خود یکی از انواع پروژها داده شده را انتخاب کنیم .

حال شما پنجره ساده Rational Rose را مشاهده می کنید که شبیه شکل زیر می باشد .



همانطور که مشاهده می کنید برای درک بهتر پنجره را به چند قسمت تقسیم کرده ایم که به شرح ذیل می باشد .

۱) در این قسمت انواع view ها را می بینیم

۲) شما در این قسمت جعبه ابزار مربوط به هر view را مشاهده می کنید

۳) در این قسمت پنجره های داخلی را نمایش می دهد .

۴) در این قسمت که در بالا قرار دارد انواع diagram ها را مشاهده می کنید .

View هایی که شما ملاحظه می کنید به شرح زیر می باشند :

اولین view که مشاهده می کنید UseCase view می باشد که در درس دوم به آن اشاره شده . UseCase view در واقع نشان دهنده کل کارهایی است که سیستم باید انجام دهد . در فارسی UseCase را به معنی « مورد کاربرد » ترجمه کرده اند . در اصل هر UseCase نشان دهنده سرویسی است که سیستم در اختیار کاربران قرار می دهد خواه این کاربر یک فرد باشد یا یک سازمان یا یک سیستم دیگر . در واقع UseCase دیاگرام ، وظیفه دارد که از یک دید بالا کلیه فعالیتهای درون سیستم را مدل کند . در بخش بعدی درباره UseCase و

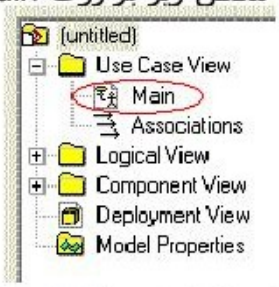
موارد کاربرد آن توضیحات بیشتری را به شما ارائه می دهیم .

دومین view که به آن می پردازیم Logical view است . Logical view نقشه اصلی سیستم یا طرحی است که باید پیاده سازی شود . یعنی آنچه در UseCase view نمایش داده شده است اکنون کمی تخصصی تر شده . به این معنی که ما در Logical view با مفاهیم برنامه نویسی بیشتر سرکار خواهیم داشت . عناصر اصلی Logical view همان کلاسها و Attributeها و Methodها هستند . عمده کار Logical view نمایش ارتباط واقعی میان کلاسها در سیستم است .

view سومی که مورد بحث قرار می دهیم Component view است . در Component view باید کلاسهای ایجاد شده در مرحله قبل به Component تبدیل شوند . در این view همچنین باید ارتباط میان این Component ها مشخص شود . این ارتباط می تواند از نوع ارتباط زمان اجرا یا زمان ترجمه یا حتی ارتباطات زمان کامپایل باشد . این Component ها همچنین می توانند زیر سیستمها باشند . این زیر سیستمها معمولاً مجموعه ای از کلاسهای هستند و از لحاظ منطقی با هم سازگاری داشتند و در کنار هم قرار گرفته اند . در این view عمده فعالیتهای طراحی باید به صورت کدهای برنامه نویسی درآید .

آخرین view که مورد بررسی قرار می دهیم Deployment view است . در Deployment view باید محل قرار گرفتن Component های مرحله قبل را دقیقاً مشخص کنیم . نوع پلت فرم را هم مشخص نماییم . در واقع این view به بستر نرم افزاری و سخت افزاری سیستم می پردازد . این view اغلب برای کسانی می تواند مفید باشد که در صدد توسعه یک نرم افزار قدیمی هستند .

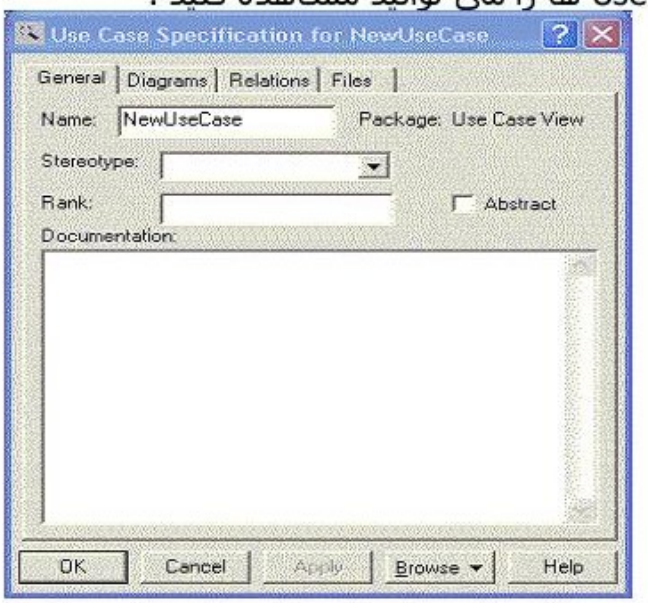
در این درس نحوه ترسیم UseCase دیاگرام را شرح می دهیم . در درس قبل آشنایی مختصری با محیط رشنال رز پیدا کرده ایم . حال با همدیگر اقدام به ترسیم یکی از مهمترین دیاگرامهای UML می پردازیم . وقتی رشنال رز را اجرا می کنید به طور پیش فرض پنجره Class Diagram باز می شود . چون ما فعلاً با این پنجره کاری نداریم پس این پنجره را ببندید و برای باز کردن پنجره UseCase دیاگرام ، مطابق شکل زیر بر روی Main دابل کلیک نمایید .



پنجره UseCase دیاگرام به شما نمایش داده می شود . همانطور که در شکل زیر مشاهده می کنید جعبه ابزار آن نیز برای ما کاملاً آشنا است .

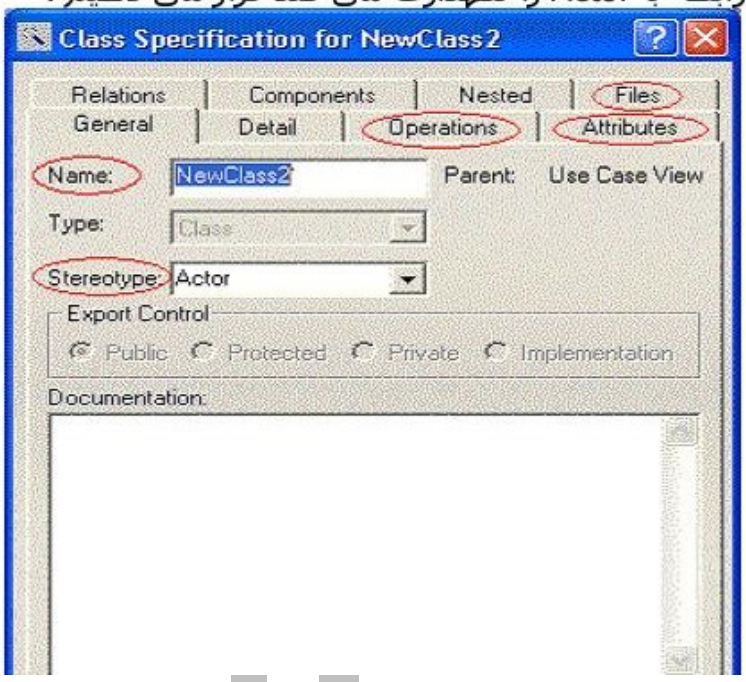
برای اضافه کردن یک UseCase به دیاگرام خود کافی است بروی ایکون آن در جعبه ابزار یک بار کلیک کنیم تا به حالت انتخاب درآید . سپس به داخل پنجره UseCase دیاگرام رفته ، مشاهده می کنید که ایکون موس به شکل «+» در می آید . برای درج UseCase درون پنجره کافی است

که در پنجره یک بار کلیک کنید . مشاهده می کنید یک UseCase جدید به دیاگرام شما اضافه شده . برای تغییرات جزیی در UseCase مثل نام و ... بر روی UseCase دابل کلیک می کنیم منویی مانند شکل زیر ظاهر می شود که می توان نام UseCase راتغییر و در تب Files آن می توان فایلهایی که به UseCase الحاق کرده ایم مشاهده کنید . در تب Diagram شما می توانید دیاگرامهای ترسیم شده برای UseCase را مشاهده کنید . در تب Ralations ، ارتباطات UseCase با سایر Actor ها و UseCase ها را می توانید مشاهده کنید .

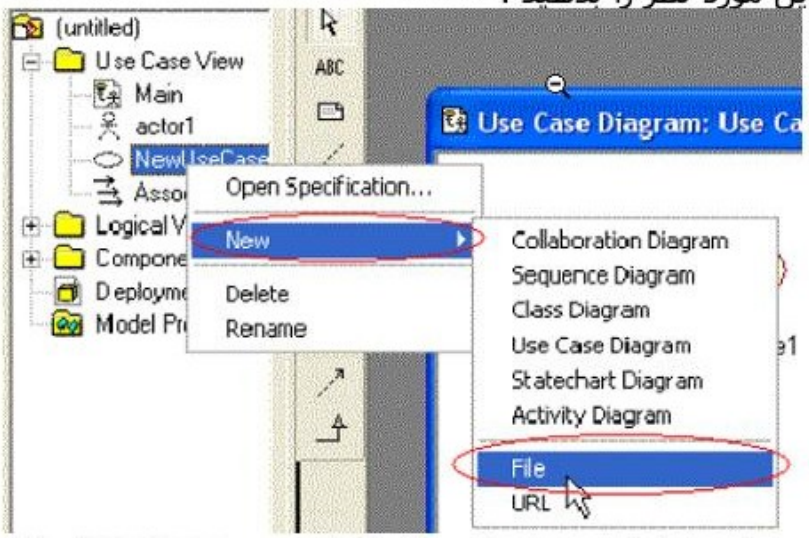


1

اضافه نمودن یک Actor به UseCase دیاگرام شبیه اضافه کردن یک UseCase به UseCase دیاگرام می باشد . پس برای اضافه نمودن یک Actor به UseCase دیاگرام ابتدا روی آیکون Actor در جعبه ابزار کلیک کرده تا به حالت انتخاب درآید سپس در پنجره UseCase دیاگرام کلیک می کنیم تا Actor به دیاگرام ما افزوده شود . می توانیم بر روی آن کلیک کنیم تا منویی مانند شکل زیر ظاهر شود . در اینجا شما می توانید نام Actor یا Streoetype آن را تغییر دهید . Streoetype در واقع شکلهایی هستند که از لحاظ مفهومی با معنایی یکسانی دارند . مثلاً Actor یک آدمک است ، در واقع نشان دهنده کسی است که می تواند کاری را انجام دهد . همچنین برای یک Actor ما می توانیم Operations ها ، Attributes ها را مشخص کنیم و در قسمت Files فایلیهایی که اطلاعات جانبی را در رابطه با Actor را نگهداری می کند قرار می دهیم .



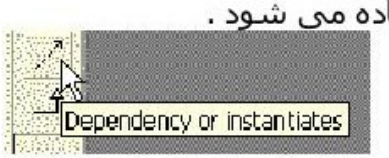
برای اضافه کردن یک سناریو به یک UseCase می توانیم در قسمت سمت چپ یک بار روی آن کلیک نماییم ، بر روی New رفته و گزینه File را انتخاب کنید . پنجره ای باز می شود که از آنجا می توانید آدرس فایل مورد نظر را بدهید .



بعد از انتخاب فایل سناریو مشاهده می کنید که فایل به زیر UseCase اضافه شده است . برای دیدن آن می توانیم بر روی آن دابل کلیک نمائیم .

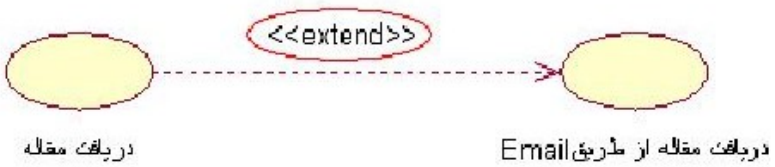
حال می خوایم در مورد روابط بین UseCase ها با Actor ها و UseCase ها با UseCase ها صحبت کنیم . اولین دسته از روابط ، روابط Association ها (شکل زیر) هستند که معمولاً ما بین Actor ها و UseCase ها قرار می گیرند . پیشنهاد می شود از این رابطه این UseCase ها استفاده نکنید .

نوع دیگر رابطه ، رابطه Dependency (شکل زیر)که دارای دو نوع Include و Extended است . این رابطه بین دو UseCase استفاده می شود .



برای بررسی رابطه Dependency نوع Include به شکل زیر توجه کنید . این رابطه به این معنی است که UseCase شماره 1 از عملیلتی که در UseCase شماره 2 انجام می شود استفاده می کند تا فعالیت خود را به پیش ببرد .

و برای بررسی رابطه Dependency نوع Extended به شکل زیر توجه کنید . نوع Extended شبیه نوع Include است با این تفاوت که سرویس گیرنده در شرایط خاص می تواند از عملیات UseCase سرویس دهنده استفاده کند . مثلاً در شکل زیر سایتی را در نظر بگیرید که افراد می توانند جهت مطالعه و دریافت مقاله ها از آن استفاده کنند . حال اگر افراد معمولی بخواهند به سایت مراجعه کنند می توانند مقالعه را مطالعه و ذخیره کنند . اما اگر افرادی که قبلاً عضو سایت هستند به این قسمت مراجعه کنند می توانند دریافت مقاله را از طریق Email خود انجام دهند . پس UseCase دیافت مقاله در حالت خاص و با برقراری شرایط ویژه قادر به استفاده از خدمات UseCase " دریافت مقاله از طریق Email " می باشد.در حقیقت این نوع رابطه در حالتIF Then برقرار می باشد .



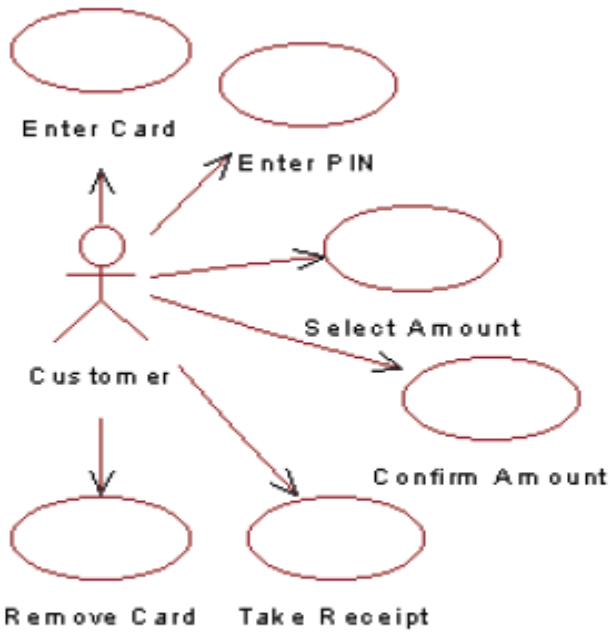
حال نوبت به بررسی رابطه Generalization یا رابطه ارث بری (شکل زیر) فرا رسیده است . این نوع روابط معمولاً یا بین Actor ها هستند یا بین UseCase ها .

تکه تکه بودن مورد کاربرد (Use Case Granularity)

تصمیم گرفتن درباره تکه های یک مورد کاربرد سخت است – مثلاً، آیا هر کاربر سیستم باید با یک مورد کاربرد تعامل داشته باشد؟ یا، آیا موارد کاربرد باید تمام تعاملات را به صورت کپسول شده درآورند؟ برای مثال یک سیستم خود پرداز را در نظر بگیرید. ما باید سیستمی بسازیم که کاربر قادر باشد از آن پول دریافت کند. در این کار با یک سری از تعاملات معمولی به شرح زیر رو به رو می شویم :

- کارت را وارد کن
- شماره رمز را وارد کن
- مقدار پول مورد نظر را انتخاب کن
- مقدار پول مورد نظر را دریافت کن
- کارت را بردار
- رسید را بگیر

آیا هر کدام از این مراحل (مثلاً کارت را وارد کن) یک مورد کاربرد است؟



به نظر شما، آیا این یک مورد کاربرد خوب و کارامد است؟

این یک اشتباه عمومی در ساخت موارد کاربرد است. در اینجا تعداد زیادی از موارد کاربرد کوچک و بی اهمیت را به وجود آوردیم. اگر این اشتباه را در سیستم های بزرگ مرتکب شویم، در نهایت تعداد زیادی مورد کاربرد خواهیم داشت که همین امر موجب افزایش شدید پیچیدگی می شود.

برای کنترل این پیچیدگی ها باید موارد کاربرد را تا جایی که امکان دارد در سطوح بالا نگاه داریم.

برای تولید یک مورد کاربرد خوب باید یک قانون کلی را در ذهن داشته باشیم، و آن این است که : یک مورد کاربرد باین هدف را برای بازیگر مشخص کند.

با به کار بستن این قانون ساده در مثال بالا ما باید از خودمان سوال کنیم که آیا هدف بازیگر در نمودار مورد کاربرد بالا گرفتن رسید است؟ مسلم است که نه.

با بکار بردن این قانون برای تمام موارد کاربردی که در نمودار بالا است متوجه می شویم که هیچ کدام از آنها هدف کاربر نیستند. هدف کاربر دریافت پول است. و این باید مورد کاربرد باشد.

یک مورد کاربرد کارآمد

کشف موارد کاربرد

یک راه برای کشف موارد کاربرد، مذاکره و مصاحبه با کاربران سیستم است. این کار سختی است، چون ممکن است دو نفر دید کاملاً متفاوتی از اینکه سیستم قرار است چه کاری را انجام دهد، داشته باشند. در این کار بعضی از توسعه گران مصاحبه تک به تک و بعضی دیگر مذاکره گروهی را ترجیح می دهند. راه دیگری که برای کشف موارد کاربرد وجود دارد و همچنین محبوبتر هم است، کارگاه (workshop) است.

کارگاه طرح ریزی نیازمندی ها (Joint Requirements Planning Workshops (JRP))

کارگاه تعدادی از افرادی که می خواهند سیستم را توسعه دهند را به دور یکدیگر می نشاند. در این گروه همه افراد دیدگاه خودشان را درباره کارهایی که سیستم باید انجام دهد را ارائه می کنند.

کلید موفقیت این کارگاه ساده انگاری است. همه افرادی که حاضر هستند را تشویق می شوند تا دیدگاه خود را راجع به سیستم ارائه دهند، و مطمئن هستند که تمام دیدگاههای آنها گرفته خواهد شد.

در جلسه یک منشی نیز حاضر است و همه چیز را مستند می کند. منشی ممکن است روی کاغذ کار کند ولی بهتر است که نمودارها بر روی پروژکتور باشد.

در اینجا سادگی نمودارهای مورد کاربرد اجتناب ناپذیر است، چون همه افراد حاضر، حتی کسانی که در عمل کامپیوتر نیز تجربه ای ندارند باید مفهوم این نمودارها را به آسانی درک کنند.

روند ساده اجرای یک کارگاه می تواند به ترتیب زیر باشد :

1. در ابتدا همفکری در باره تمام بازیگران
2. همفکری در باره تمام موارد کاربرد
3. توجیه تک تک موارد کاربرد به وسیله شرح یک خطی یا یک پاراگرافی برای هرمورد کاربرد.
4. نمادسازی موارد کاربرد.

چند نکته که در کارگاه باید رعایت شود :

- در مورد کشف تمام موارد کاربرد و تمام بازیگران سخت گیری نکنید. این طبیعی است که بعضی از موارد کاربرد بعداً در طول فرایند پدیدار شوند.
- اگر در مرحله سوم نتوانستید یک مورد کاربرد را توجیه کنید، ممکن است اصلاً آن یک مورد کاربرد نباشد، بنابراین آن را پاک کنید (در این کار تردید نداشته باشید و هر مورد کاربردی را که احساس کردید لازم نیست به راحتی حذف کنید چون اگر لازم باشند بعداً می توان آنها را برگرداند).

باید مراقب باشید که نصایح بالا موجب ناکارامدی و بی نظمی در نمودار ها نشود، اما به خاطر داشته باشید که منفعت فرایند تکراری این است که در ابتدا هیچ چیز 100٪ کامل نیست و رفته رفته در تکرارهای بعدی کامل می شود.

چند نکته که در مشورت باید رعایت شود :

- همه ایده ها را مستند کنید، حتی ایده هایی که به نظرتان مزخرف می آید را هم مستند کنید چون ممکن است این ایده های نا کارامد ایده های بسیار خوبی را در ذهن دیگر شرکت کنندگان به وجود آورد.
- هیچ ایده ای را ارزیابی یا نقد نکنید.

مثال رسم Usecase Diagram: هدف از سيستم تعميرگاه فراهم نمودن مديريتي كارا براي همه جنبه هاي چرخه سرويس دهي و تعمير از تعريف كارهاي مورد نياز مشتريان گرفته تا خاتمه يافتن اين كارها است. سيستم بايد تسهيلات زير را ارائه نمايد:

- رزرو كارها(شامل سرويس و تعمير)
- شناسائي قطعات يدكي مورد نياز و درخواست آنها
- زمانبندي كارها
- ثبت جزئيات كارهاي انجام شده
- مسائل مربوط به اتمام يك كار: مانند تحويل ماشين و محاسبه هزينه كار
- اينجا كارها بر دو نوعند: معمولي و اولويت دار

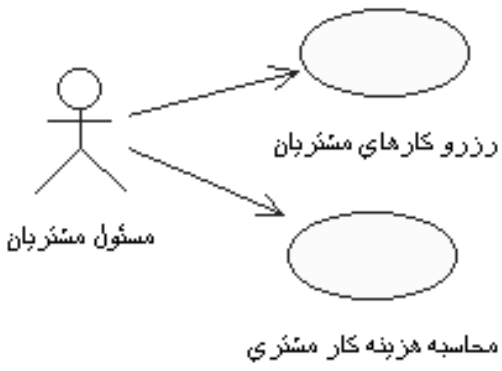
۱) شناسائي عوامل :

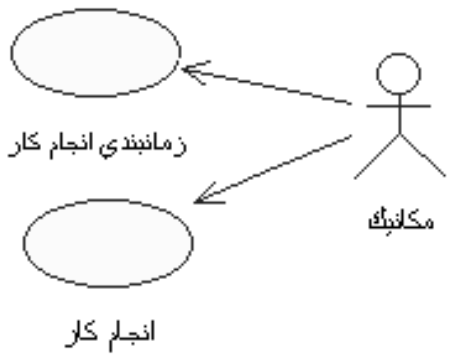
عامل	شرح
مسئول پذيرش مشتريان	مسئول ارتباط با مشتريان و شناسائي نيازهاي آنها
كنترل كننده قطعات	مسئول نگهداري وتهيه قطعات يدكي مورد نياز و پيش بيني نيازهاي مشتريان
مكانيك	مسئول زمانبندي كارها، اطمينان از درستي انجام آنها، و ثبت كارهاي انجام شده

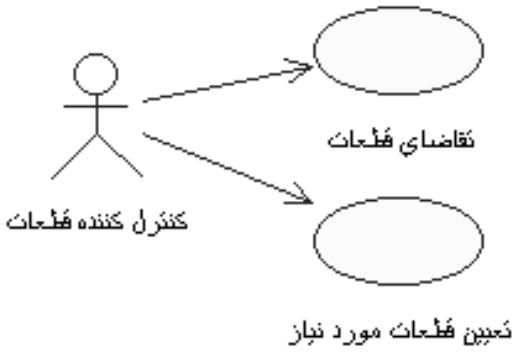
۲) شناسائي موارد كاربري

عامل	مورد كاربري
مسئول پذيرش مشتريان	ثبت كار مورد نياز مشتري
كنترل كننده قطعات	تعيين قطعات مورد نياز
كنترل كننده قطعات	درخواست قطعات
مكانيك	زمانبندي كارها
مكانيك	مديريت كار از ابتدا تا خاتمه، اطمينان از درستي انجام آن و ثبت جزئيات كار انجام شده
مسئول پذيرش مشتريان	اطمينان از رضايت مشتري، دريافت مزد كار و تحويل ماشين به مشتري

۳) ايجاد نمودار موارد كاربري

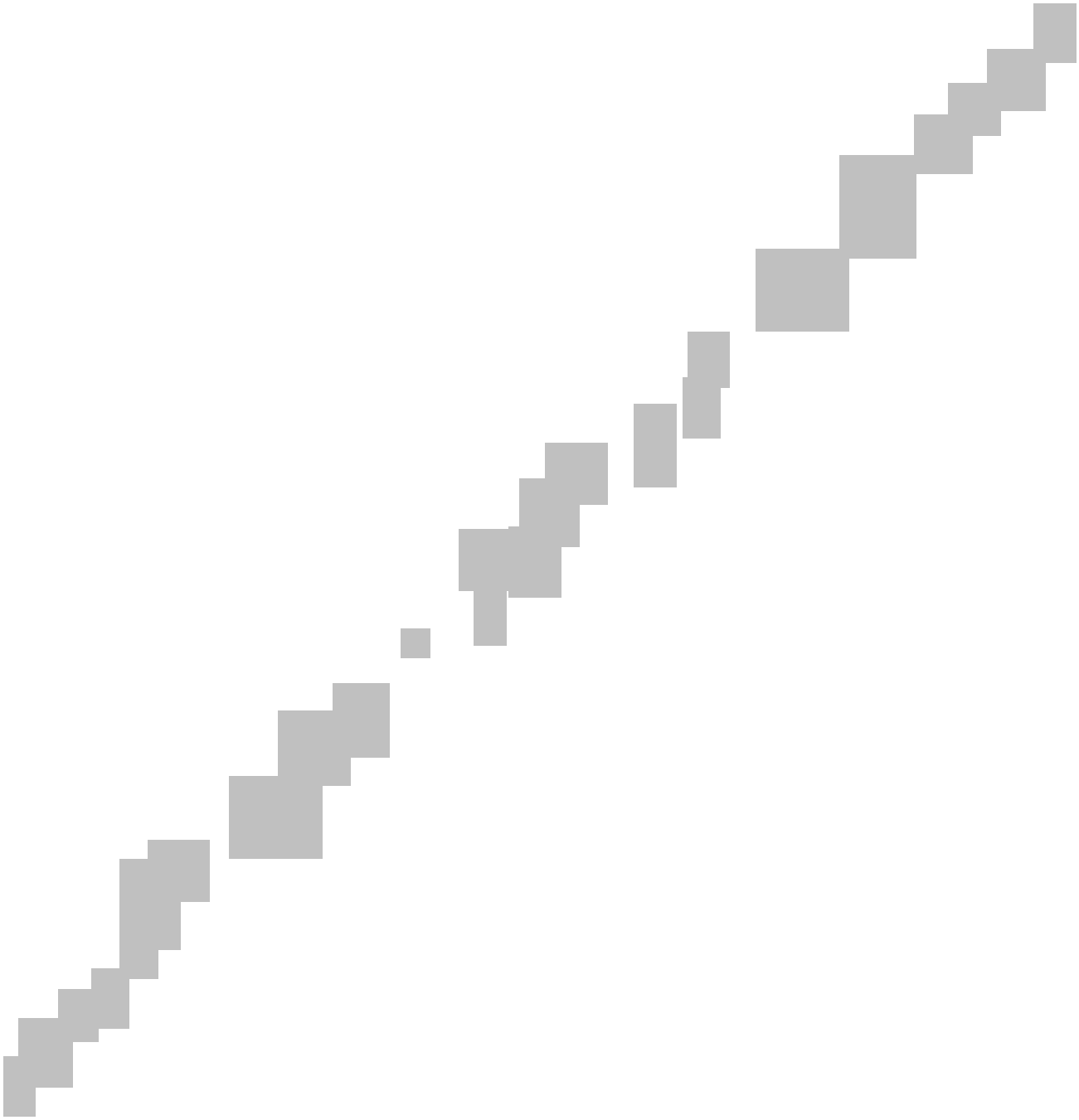






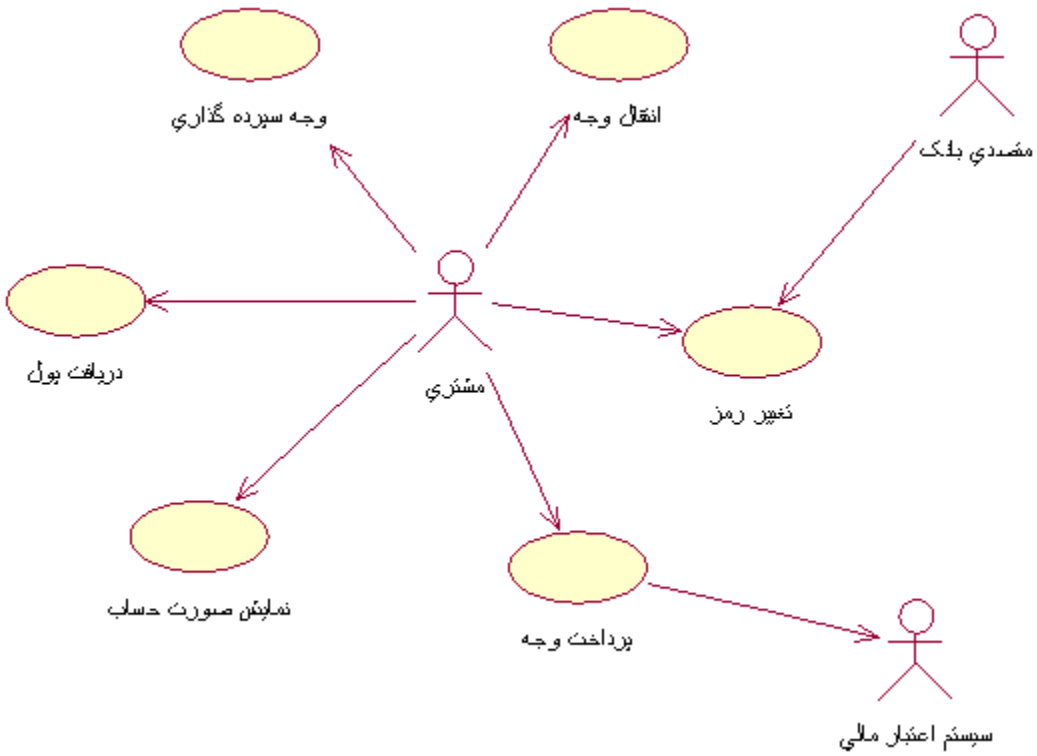
۳) تشريح موارد كاربري

نام مورد كاربری	رزرو كارهای مشتريان
توصيف مختصر	تعيين كار(سرويس) مناسب كه نيازهای مشتريان را به نحو احسن برآورد سازد
جريان رويدادها	**جريان اصلی** ۱- بدست آوردن جزئيات خودرو و مشتری: - برای مشتريان فعلی جزئيات مربوط به آنها استخراج كنيد. - برای مشتريان جديد مشخصات مورد نياز (مانند: نام، آدرس، شماره خودرو، مدل آن، و سال ساخت) را ثبت نماييد. ۲- اگر كار مورد نياز سرويس باشد، سرويسهای مناسب مدل خودرو را پيدا كنيد. ۳- اگر كار مورد نياز تعمير باشد، هزينه آنرا پيش بينی نماييد. ٤- زمان و ساعت كار با توافق مشتری مشخص نماييد. ٥- مشخصات ذكر شده را، پس از تاييد مشتری، ثبت نماييد.



UseCase Diagram مربوط به سیستم ATM و سیستم فروش بلیط

در این بخش قصد داریم آنچه تا به حال آموخته ایم در قالب usecase دیاگرام به تصویر بکشیم . همانطور که قبلاً گفته ایم usecase **دیاگرام** مجموعه ای از usecase ها ، Actorها و روابط موجود بین آنها در یک نمودار به نام رسم می شود که در بر گیرنده تمام آن چیزی است که از سیستم می خواهند ، در واقع usecaseدیاگرام نمونه ای تصویری از مدل نیازمندیها برای سیستم است . حال برای درک بهتر این دیاگرام به شکل زیر که مربوط به usecaseدیاگرام سیستم ATMمی باشد توجه کنید .



همانطور که مشاهده می کنید با یک نگاه می توان فهمید این سیستم دارای چه قسمتهایی است و چه کاری انجام می دهد .

مشتری در واقع کسی است که بانک به او یک رمز (و کارت) جهت استفاده می دهد .

این رمز در بار اول توسط متصدی بانک ساخته می شود و در اختیار مشتری قرار می گیرد .

مشتری با استفاده از دستگاه ATM می تواند رمز خود را تغییر دهد .

مشتری برای افتتاح حساب باید وجهی جهت سپرده گذاری در نزد بانک قرار دهد .

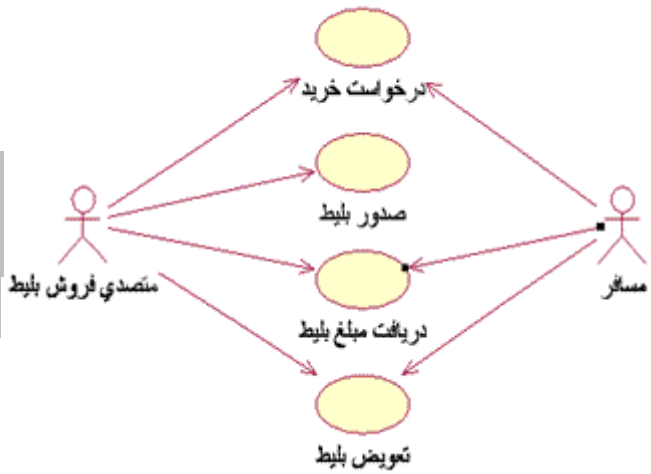
مشتری می تواند از این وجه برداشت کند .

همچنین مشتری می تواند صورت حساب خود را از دستگاه دریافت کند .

مشتری توسط دستگاه ATM می تواند وجهی به سایر حسابها پرداخت کند .

شما به این نکته توجه کنید که خروجی"UseCase پرداخت وجه" به عنوان ورودی عامل (Actor) سیستم اعتبار مالی مورد استفاده قرار می گیرد . در اینجا سیستم اعتبار مالی با وجود اینکه یک سیستم است ، ولی بدلیل اینکه برای ادامه کار خود نیازمند اطلاعات این سیستم است به عنوان یک Actor در نظر گرفته شده است .

حال به مثال دیگری توجه کنید



در این مثال که همان سیستم فروش بلیط می باشد :

ابتدا مسافر درخواست خرید بلیط می کند .

متصدی فروش درخواست را دریافت کرده و اقدام به صدور بلیط می کند .

مسافر بهای بلیط را پرداخت می کند و بلیط را تحویل می گیرد .

همچنین امکان این وجود دارد که مسافر بلیط خود را تعویض کند .

امیدوارم که مسئله کاملاً برای شما قابل فهم باشد. در پایان به این نکته اشاره می کنیم که تجزیه و تحلیل سیستم ها امری سلیقه ای است و می توان گفت مانند نوشتن انشاء می باشد . پس نباید انتظار داشت که تجزیه و تحلیل دو نفراز یک سیستم شبیه هم باشد . چون امکان دارد این دو نفر از دو دیدگاه کاملاً متفاوت به قضیه نگاه کنند.

نمودار توالي (Sequence Diagram)

ما برای توضیح بهتر Use case های خود ، می توانیم برای آنها نمودارهایی ترسیم کنیم . نمودار توالی یکی از آن نمودارها می باشد .

نمودار توالی یکی از نمودارهای Interaction می باشد که روند یک Use case را مرحله به مرحله نشان می دهد . دیگر نمودار Interaction نمودار همکاری یا Collaboration می باشد که در بخشهاي بعدی راجع به آن بحث می کنیم .

نمودار توالی برای نشان دادن جریان عملیات در یک Use case بر حسب زمان استفاده می شود . این نمودار موقعی مفید است که کس بخواهد روند منطقی یک سناریو را بازدید کند .

یک آبجکت چیست ؟ آبجکتها در اطراف ما قرار دارند . آبجکت آن چیزی است که اطلاعات و روشها را در خود کپسوله(نگهداری) می کند .

یک کلاس چیست ؟ طرح کلی برای یک آبجکت را کلاس آن فراهم می کند .به عبارت دیگر ، یک کلاس تعیین کننده اطلاعاتی است یک آبجکت می تواند نگهداری کند و نشان دهنده رفتارهایی است که می تواند داشته باشد .

نمودار توالی موارد زیر را در بر می گیرد :

Objects (آبجکت ها) : یک نمودار Interaction می تواند از نام آبجکت ها ، نام کلاسها و یا از هر دوی آنها استفاده کنند .

Messages (پیغام ها) : با استفاده از یک پیغام ، یک آبجکت یا کلاس می تواند از یک آبجکت یا کلاس دیگر ،برخی عملیات خاص را در خواست نماید .

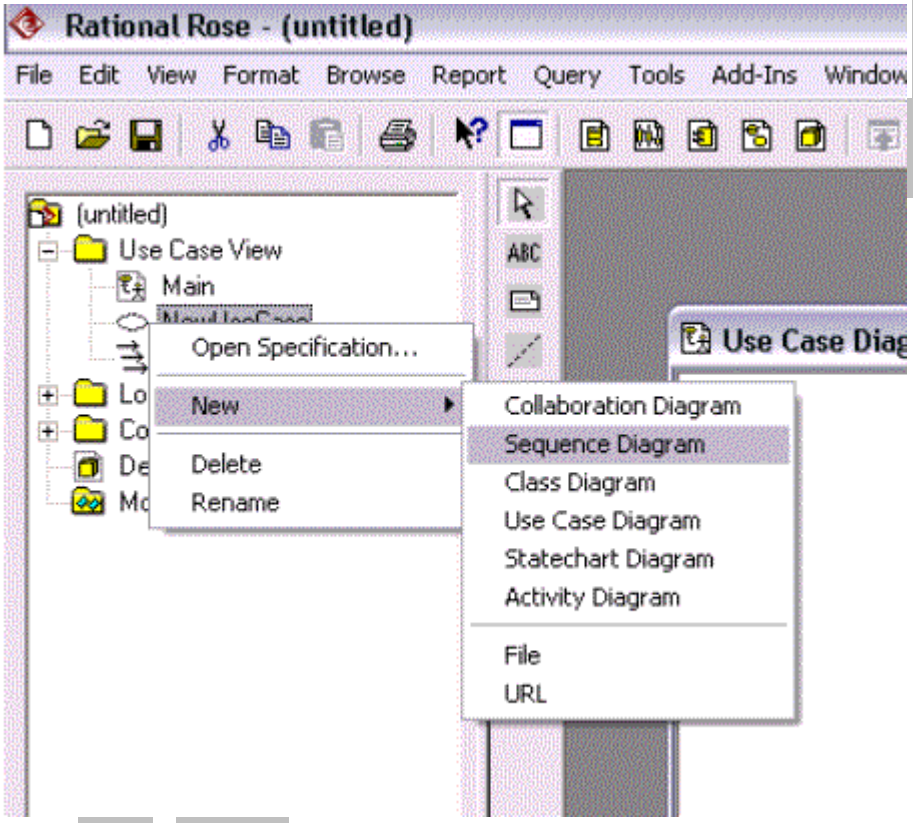
عاملهای وابسته در بالای نمودار نشان داده می شوند . همچنین آبجکت هایی که سیستم نیاز دارد تا Usecase را به نتیجه برساند در بالاترین نقطه نمودار نشان داده شده است . هر فلش یک پیغام ارسالی بین عامل و آبجکت ، یا آبجکت و آبجکت را نمایش می دهد تا عملیات مورد نیاز را به انجام برساند .

کاربران می توانند به این نمودار نگاه کنند و مشخصات پردازش خود را ببینند .تحلیلگران جریان پردازش را در نمودار توالی می بینند . برنامه نویسان آبجکت هایی که به کد نیاز دارند را به همراه عملگرهای آن آبجکت می بینند .مهندسین تضمین کیفیت می توانند جزئیات پردازش و تولید وTest case مبتنی بر پردازش را ببینند . خلاصه اینکه نمودار توالی (Sequence Diagram) برای همه افرادی که با پروژه در ارتباط هستند مفید می باشد .

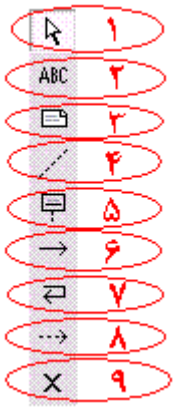
ساخت یک نمودار توالی :

این نمودار می تواند در نمای Use Case یا در نمای منطقی (Logical) ساخته شوند . نمودارهای توالی باید مستقیماً درون یک Use Case قرار گیرد و یا درون یک بسته قرار بگیرند .

برای ایجاد یک نمودار توالی به شکل زیر توجه کنید .



برای ایجاد یک Sequence Diagram بر روی Use Case مربوطه کلیک راست کنید و همانطور که در شکل بالا مشاهده می کنید مسیر New - Sequence Diagram را بروید . مشاهده می کنید که یک نمودار توالی به Use Case شما اضافه می شود . این نمودار دارای نوار ابزار مخصوص به خود می باشد که در شکل زیر آن را مشاهده می کنید .



ایـــــن دکمه ها از بالا به پایین به شرح ذیل می باشند :

دکمه اول : برای انتخاب یک آیتم، مکان نما را به یک فلش تبدیل می نماید .

دکمه دوم : یک کادر متن را به نمودار می افزاید .

دکمه سوم : یک یاداشت را به نمودار می افزاید .

دکمه چهارم : یادداشتی را به یک آیتم درون نمودار می افزاید .

دکمه پنجم : یک آبجکت جدید را به نمودار می افزاید .

دکمه ششم : پیغامی را بین دو آبجکت مبادله می نماید .

دکمه هفتم : یک پیغام بازتابی را طراحی می نماید .

دکمه هشتم : جهت پاسخ به پیغام ارسال شده توسط آبجکت استفاده می شود .

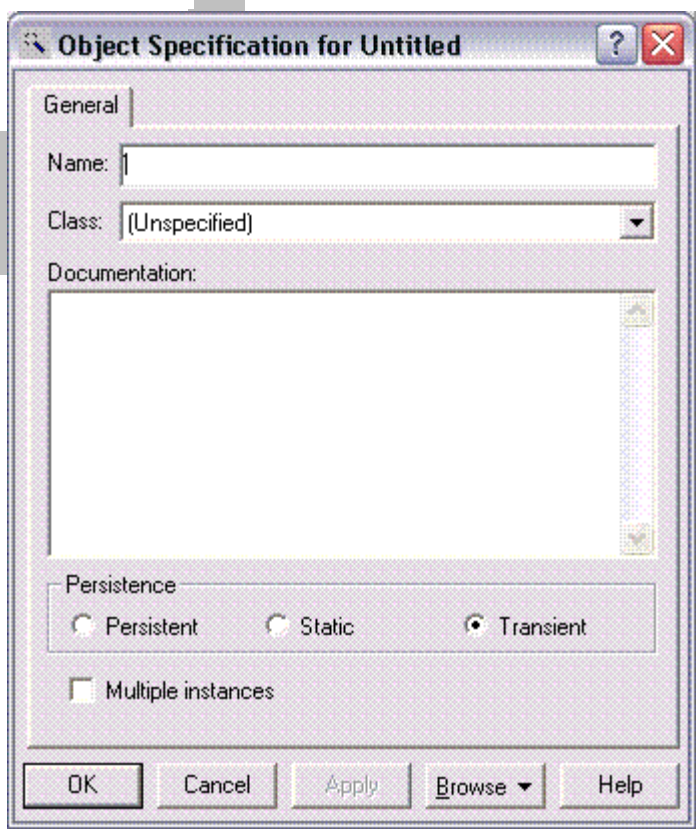
دکمه نهم : انتهای کار را در نمودار مشخص می نماید .

چه کسي نمودار توالي (Sequence Diagram) را ترسیم میکند؟

نمودار توالی (Sequence Diagram) ، نمودار همکاری (Collaboration Diagram) ، نمودار فعالیت (Activity Diagram) و نمودار حالت (StateChart Diagram) همگی مربوط به UseCase Diagramمی باشد . یعنی وقتی شما UseCase Diagram خود را ترسیم کردید و مشخص شد در سیستم شما چه UseCase هایی وجود دارد آنگاه می توانید برای هریک از UseCaseهای خود نمودارهای ذکر شده بالا را ترسیم کنید . در واقع چهار نمودار یاد شده بالا هر کدام از زاویه ای UseCase شما را تشریح می کنند تا مشخص شود UseCase ی که شما در UseCase Diagram خود قرار داده اید باید چه روند کاری را پشت سر بگذارد و دارای چه وظیفه ای است . همچنین اگر شما به آموزش نمودارهای ذکر شده توجه کنید ، متوجه خواهید شد که نمودارهای بالا را ما در قسمت UseCase Diagram به پروژه خود اضافه کرده ایم.

در بخش قبل تا حدودی با نحوه ایجاد یک نمودار توالی و همچنین با نوار ابزار آن و آبجکت ها و پیغامها آشنا شدیم . حال می خواهیم از دانش خود جهت ترسی یک نمودار توالی استفاده کنیم .

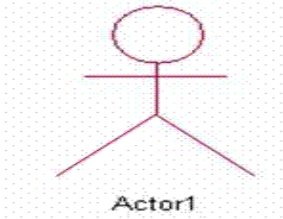
برای اضافه نمودن یک آبجکت به دیاگرام خود کافی است در نوار ابزار دکمه آبجکت را به حالت انتخاب در آورده سپس در دیاگرام خود کلیک کنیم . ملاحظه می کنید که یک آبجکت به دیاگرام شما اضافه شده است . برای حذف یک آبجکت کافی است آن را انتخاب کنید سپس کلیدهای Ctrl + D را بفشارید . برای نام گذاری آبجکت کافی است بر روی آن دابل کلیک کرده یا کلیک راست نمائید و از گزینه open specification راانتخاب کنید تا پنجره ای شبیه پنجره زیر برای شما به تصویر کشیده شود .



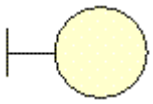
شما می توانید در این پنجره نام ، کلاس ، مستندسازی ، Persistence (پایداری) و اینکه آبجکت چندین خصوصیت دارد یا خیر را تنظیم کنید .

آبجکت ها دارای Stereotype های مختلفی هستند که در زیر آنها را معرفی می کنیم .

Actor : یا عامل که قبلاً درباره آن بحث نموده ایم .



Boundary : به معنای User Interface هستند . یعنی هرکجا خواستیم بگوئیم واسط کاربر از این شکل استفاده می کنیم . در زیر شکل آن را مشاهده می کنید .



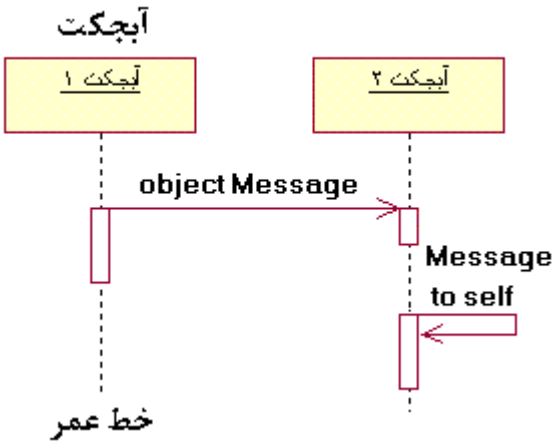
Control : این آبجکت ها همان اشیاء کنترلی هستند یعنی هرکجا در تحلیل قصد نمایش اشیاء کنترلی را داشتیم از این شکل استفاده می کنیم . در زیر شکل آن را مشاهده می کنید .

Entity : اشیای هستند که در سیستم وجود دارند . مثلاً شی بلیط را در سیستم صدور بلیط با این شکل نمایش می دهند . در زیر شکل آن را مشاهده می کنید .

Table : اگر از میان اشیاء از جدولی از پایگاه داده استفاده می کنید می توانید برای نمایش آن از این شکل استفاده کنید . در زیر شکل آن را مشاهده می کنید .

برای اضافه نمودن یک پیغام کافی است که دکمه Object Message را از نوار ابزار به حالت انتخاب در بیاوریم . سپس برای اضافه شدن پیغام بین دو آبجکت یا عامل و آبجکت ، موس را

از خط عمر (life line) آبجکت یا عامل در حال ارسال پیغام به آبجکت یا عامل در حال دریافت پیغام بکشید . به شکل زیر توجه کنید .

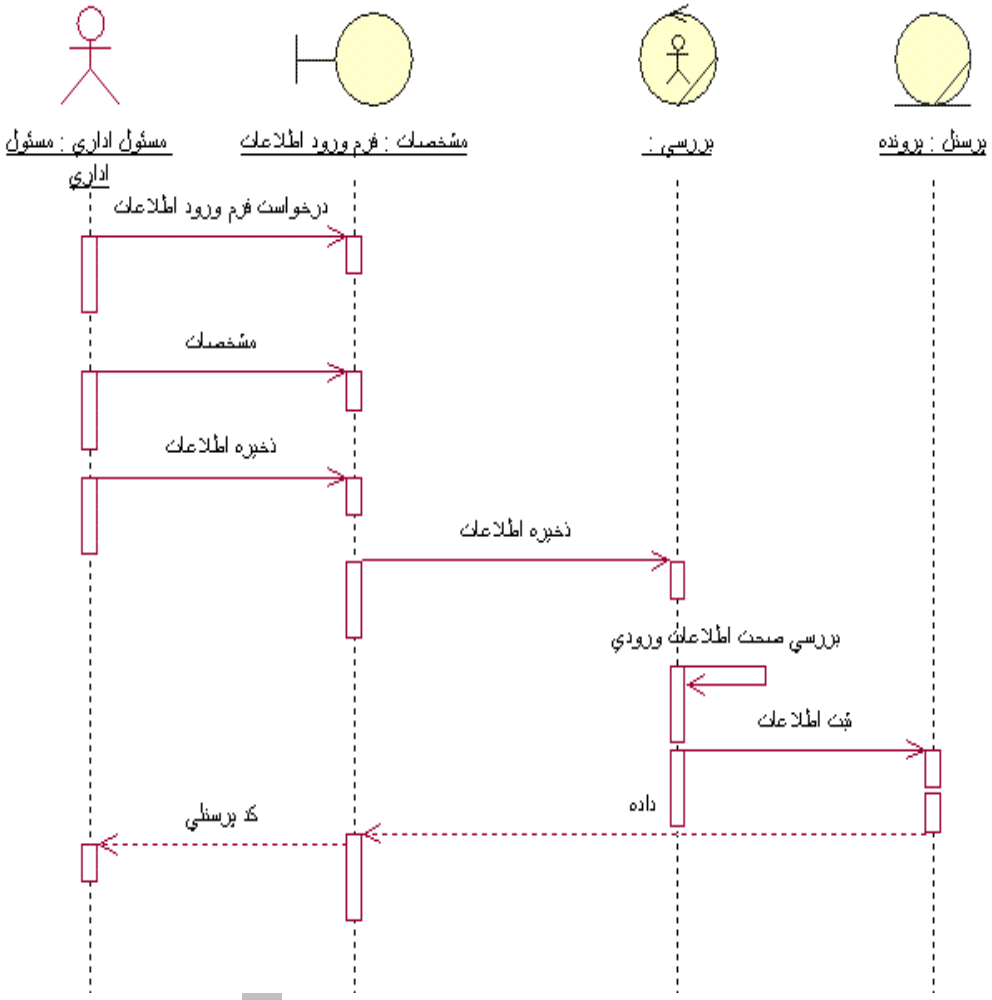


اضافه نمودن یک Message to self شبیه Object Message عمل می کنیم با این تفاوت که فقط موس را بر روی خط عمر آبجکت یا عامل مورد نظر یک بار کلیک می کنیم .

برای گذاشتن اسم بر روی پیغامها کافی است روی آن دابل کلیک یا بر روی آن کلیک راست کرده و گزینه open specification را انتخاب کنیم . در پنجره ظاهر شده در قسمت Name می توانیم نامی برای پیغام خود انتخاب کنیم .

حال که طریقه اضافه نمودن آبجکت و پیغام را به دیاگرام خود آموختیم وقت آن رسیده که نمودار توالی خود را ترسیم کنیم . برای ترسیم نمودار توالی باید از روی سناریو اقدام نمائیم یعنی از ابتدای سناریو شروع می کنیم و هر کجا به شی برخورد کردیم که با اشیاء دیگر در رابطه است ، آن شی را رسم و نحوه ارتباط آن را با دیگر اشیاء نیز مشخص می کنیم . در پایان این دیاگرام ما باید مدلی تصویری از آنچه در Use case اتفاق می افتد را تولید کرده باشیم .

در زیر شما نمونه ای از یک نمودار توالی (Sequence Diagram) که برای یک Use Case طراحی شده را مشاهده می کنید .


مسئول اداري : مسئول اداري
مشخصات : فرم ورود اطلاعات
بررسي :
پرسنل : پرونده
درخواست فرم ورود اطلاعات
مشخصات
ذخيره اطلاعات
ذخيره اطلاعات
بررسي صحت اطلاعات ورودي
ثبت اطلاعات
داده
كد پرسنلي

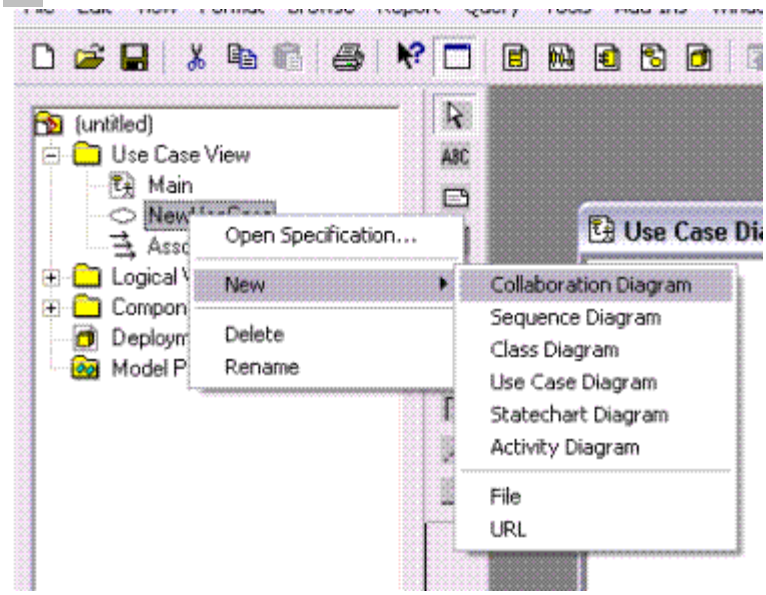
نمودار همکاری (Collaboration Diagram)

یکی دیگر از نمودارهای Interaction ، نمودار همکاری می باشد . نمودار همکاری شباهت بسیاری به نمودار توالی دارد ولي اصلی ترین تفاوت آنها در شمای ظاهری آنها می باشد . دیاگرام همکاری بیشتر بر روی رابطه بین آبجکت ها متمرکز می شود . در حالی که یک دیاگرام توالی اعمال آبجکت ها را در یک توالی زمانی نشان می دهد و بر حسب زمان منظم می شود .

در نمودار همکاری دید متفاوتی از روند عملیات Use Case ارائه می شود . در این نمودار مشاهده ارتباط بین آبجکت ها آسان تر است .

در Rose شما می توانید از روی یک نمودار توالی (Sequence Diagram) به آسانی یک نمودار همکاری (Collaboration Diagram) بسازید . برای این کار یا کلید F5 را فشار دهید یا Browser و سپس Diagram (Collaboration Sequence) Create را انتخاب کنید . نموداری که از این طریق ساخته می شود کمی آشفته است . برای نظم بخشیدن به نمودار خود کافی است آبجکت ها را بوسیله موس در محل های مناسب قرار دهید .

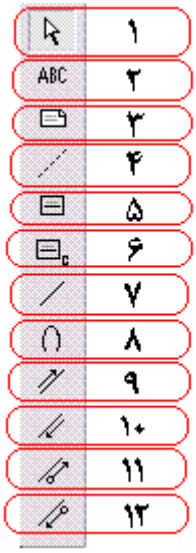
شما همچنین می توانید بر روی Use Case مورد نظر در مرورگر کلیک راست کرده و مسیر New – Collaboration Diagram را انتخاب کنید (به شکل زیر توجه کنید) .



هر کدام از روشهای بالا را که بروید در نهایت نمودار همکاری برای شما ایجاد می شود . نمودار همکاری مانند نمودار توالی دارای نوار ابزار مخصوص به خود است که در ادامه آن را برای شما توضیح می دهیم .

نوار ابزار نمودار همکاری :

نوار ابزار نمودار همکاری مانند شکل مقابل است.



دکمه اول : برای انتخاب یک آیتم مکان نما را به یک فلش تبدیل می کند.

دکمه دوم : یک کادر متن را به نمودار می افزاید.
دکمه سوم : یادداشتی را به نمودار می افزاید.
دکمه چهارم : یادداشتی را به آیتمی درون نمودار متصل می کند .
دکمه پنجم : یک آبجکت جدید را به نمودار می افزاید .
دکمه ششم : یک نمونه کلاس جدید را به نمودار می افزاید .
دکمه هفتم : مسیری را برای ایجاد ارتباط بین دو آبجکت می سازد .
دکمه هشتم : نشان می دهد که یک آبجکت می تواند عملیات شخصی خود را فراخوانی نماید .
دکمه نهم : پیغامی را بین دو آبجکت یا از یک آبجکت به آبجکت دیگر می افزاید .
دکمه دهم : پیغامی را در جهت مخالف بین دو آبجکت یا از یک آبجکت به آبجکت دیگر می افزاید .
دکمه یازدهم : جریان اطلاعات بین دو آبجکت را نشان می دهد .
دکمه دوازدهم :جریان اطلاعات را در جهت مخالف بین دو آبجکت را نشان می دهد.
هر نمودار توالی یا همکاری باید دارای آبجکت عامل باشد . آبجکت عامل یک محرک خارجی است که به سیستم اعلام می کند تا یک عملیات را راه اندازی کند . آبجکت های عامل برای نمودار Interaction ، عاملهایی که در نمودار Use Case با Use Case ارتباط دارند را نشان می دهد .

برای ایجاد یک آبجکت عامل بر روی نمودار Interaction :
برای ایجاد یک آبجکت عامل بر روی نمودار Interaction :

۱. نمودار Interaction (توالی یا همکاری) را باز کنید .
۲. عامل را در مرورگر انتخاب کنید .
۳. عامل را از مرورگر به نمودار باز بکشید .

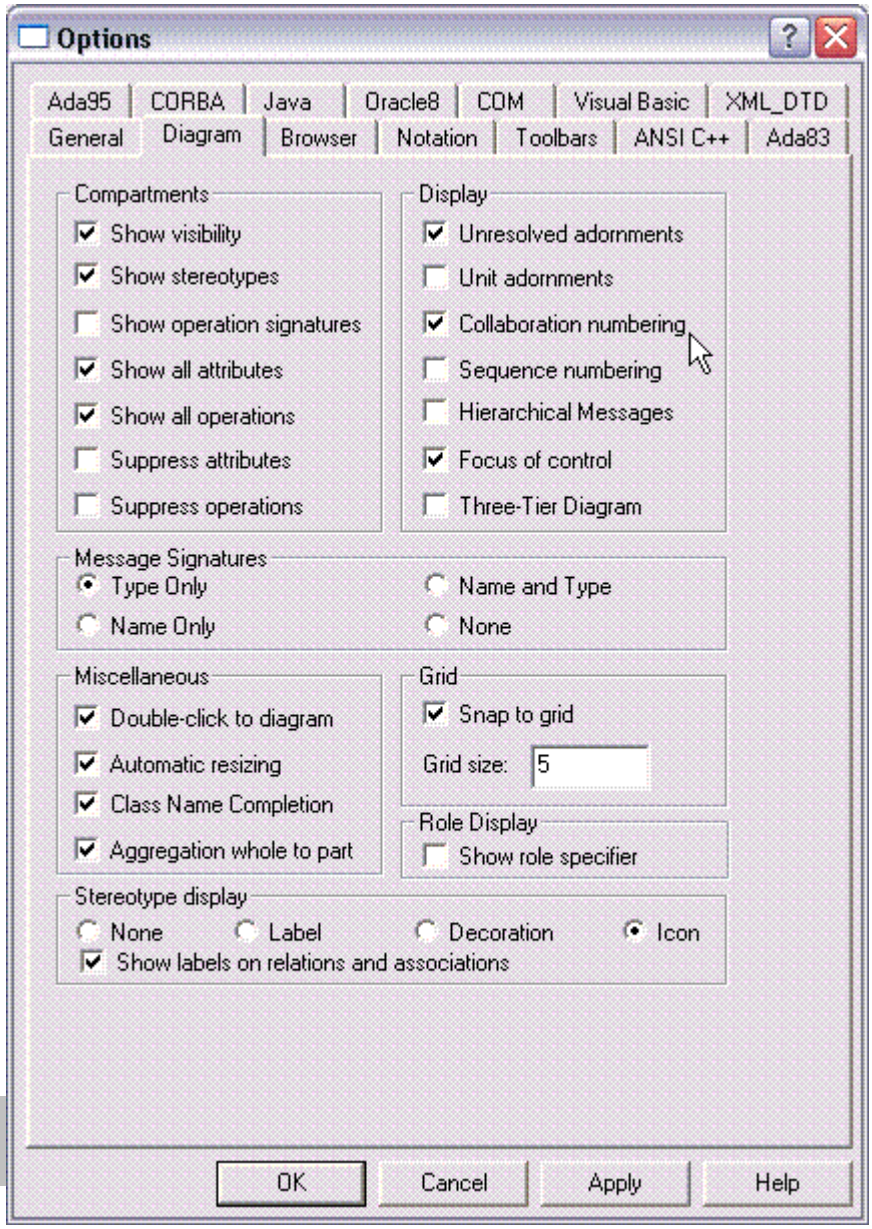
افزودن هر یک از گزینه های نوار ابزار نیز به راحتی امکان پذیر است . کافی است دکمه مورد نظر را انتخاب سپس در دیاگرام خود در مکان مورد نظر کلیک کنیم (مثل افزودن آبجکت یا یادداشت و یا پیغامی که بر روی یک مسیر ارتباطی بین دو آبجکت وجود دارد و ...) یا دکمه مورد نظر را انتخاب کرده و از یک آبجکت به آبجکت دیگر در درون دیاگرام بکشیم (مثل افزودن یک مسیر ارتباطی) .

شماره گذاری پیغامها در نمودار همکاری :

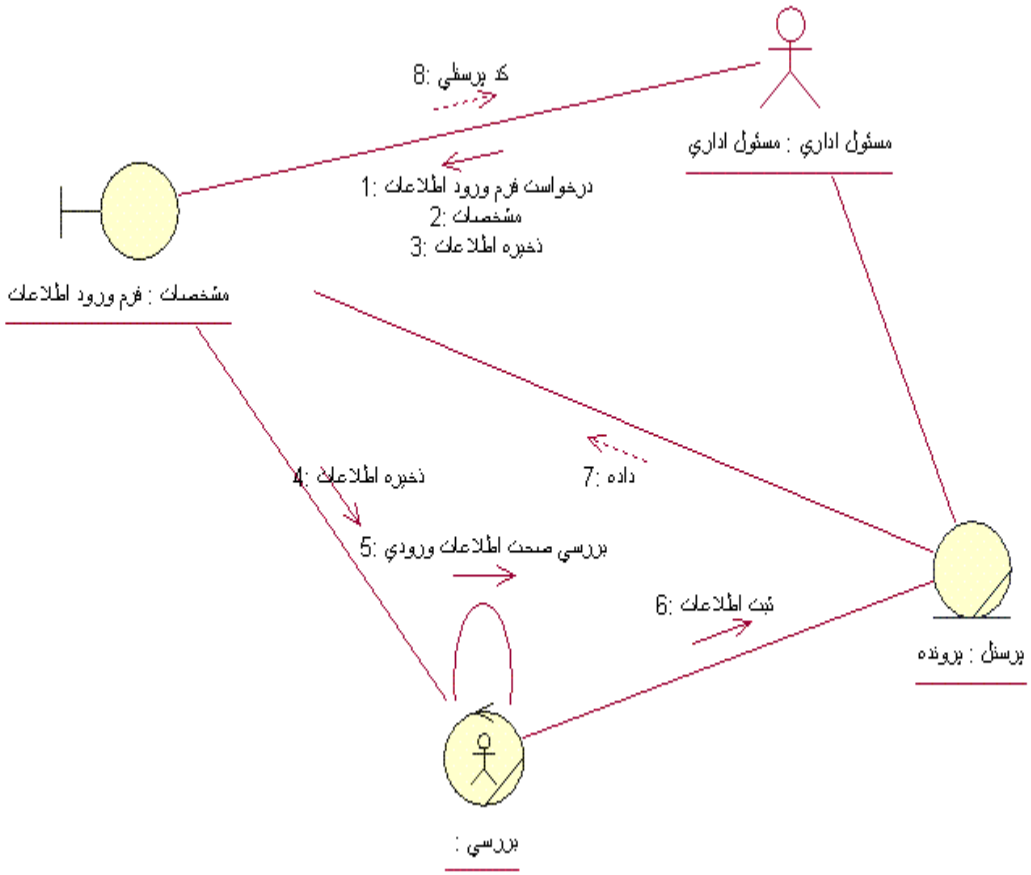
در نمودار توالی چون نمودار را از بالا به پائین می خوانید ، شماره گذاری پیغامها آن چنان ضروری به نظر نمی رسد . ولی در مودار همکاری چنانچه شماره گذاری پیغامها را حذف کنید ، اطلاعات مربوط ، تناوب خویش را از دست می دهند . با اینکه رعایت نکات ذکر شده مهم است ولی در Rose برای نمودارهای همکاری می توانید بطور دلخواه ، شماره گذاری پیغامها را غیر فعال کنید .

برای غیر فعال کردن یا فعال کردن شماره گذاری پیغامها :

۱. از منوی Tools گزینه Options را انتخاب کنید .
۲. برگه Diagram را انتخاب کنید .
۳. کادر انتخاب Collaboration and Sequence Numbering را فعال یا غیر فعال کنید .



برای نمونه به نمودار همکاری « ثبت مشخصات پرسنل » که نمودار توالی آن در درس هشتم برای شما نشان داده شده بود توجه کنید .



در پایان آموزش نمودارهای Interaction توجه شما را به چند نکته جلب می کنم .

نمودارهـای تـوالی اطلاعـات را بـه ترتیـب زمـانی نـشان مـی دهنـد . نمـودار تـوالی بـرای مسیرهای متناوب به یک Use Case ساخته شده اند. آنها بـرای مـشاهده پیـشرفت عملیـات یک Use Case مفید می باشند . نمودارهای همکاری ، روند اطلاعات را نشان می دهند ولی در اینجا ترتیب زمانی در نظر گرفته نشده است . نمودارهای همکاری رابطه بین آبجکت ها و پیغام های بین آبجکت ها را شرح می دهند

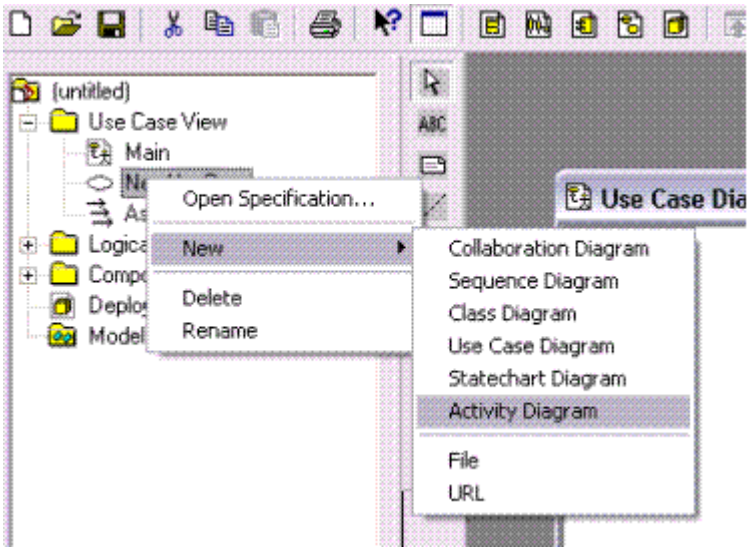
نمودار فعالیت (Activity Diagram)

در این نمودار چگونگی جریان انجام یک کار صرف نظر از فاعل آن مشخص می شود . بر خلاف نمودارهای همکاری که فاعلان کار (Actors) در جریان انجام کار وجود دارند . این نمودار را می توان برای شرح Use Caseو یا هر یک از افعال (Operation) کلاسها ترسیم نمود .

نمودارهای فعالیت بیشتر برای مدل کردن یک عملیات مورد استفاده قرار می گیرد ، یعنی گاهی اوقات که یک عملیات پیچیده می شود ، می توان از این مدل برای توضیح بیشتر استفاده کرد . این نمودار شباهت فراوانی به فلوچارت دارد و از لحاظ معنایی نیز همان مفهوم را دنبال می کند . درمدلهای شی گرایی از این نمودار کمتر استفاده می شود زیرا همانطور که گفتیم بیشتر برای مدل سازی عملیاتها از این نمودار استفاده می شود ، حال آنکه تمرکز برنامه های شی گرا عمدتاً روی اشیاء است . با این وجود شما به عنوان یک طراح ، هرگاه که لازم دانستید از این نمودار برای شرح یک Use Case یا متد از آن استفاده کنید . این نمودار برای افرادی که به روش Oriented Process برنامه می نویسند بیشترین کاربرد را در مدل سازی سیستم پیدا می کند .

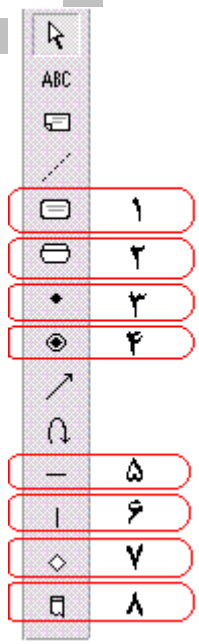
ایجاد یک نمودار فعالیت

ایجاد یک نمودار فعالیت شبیه ایجاد یک نمودار توالی یا همکاری است . ابتدا بر روی Use Case مورد نظر که می خواهیم برای آن نمودار ترسیم کنیم کلیک راست کرده و مسیر New > Activity Diagram را می رویم . مشاهده می کنید که یک نمودار فعالیت برای شما ایجاد شده است . به شکل زیر توجه کنید .



در ادامه به معرفی اشکال استاندارد نمودار فعالیت می پردازیم .

قبل از توضیح نوار ابزار نمودار فعالیت این نکته را یادآور می شوم که ، به دلیل اینکه بعضی از دکمه های این نوار ابزار شبیه دیگر دیاگرام ها نمودارهای توالی یا همکاری یا Use Case است ، از توضیح آن خودداری و شما می توانید برای کسب توضیحات آن دکمه ها به دروس گذشته مراجعه کنید .



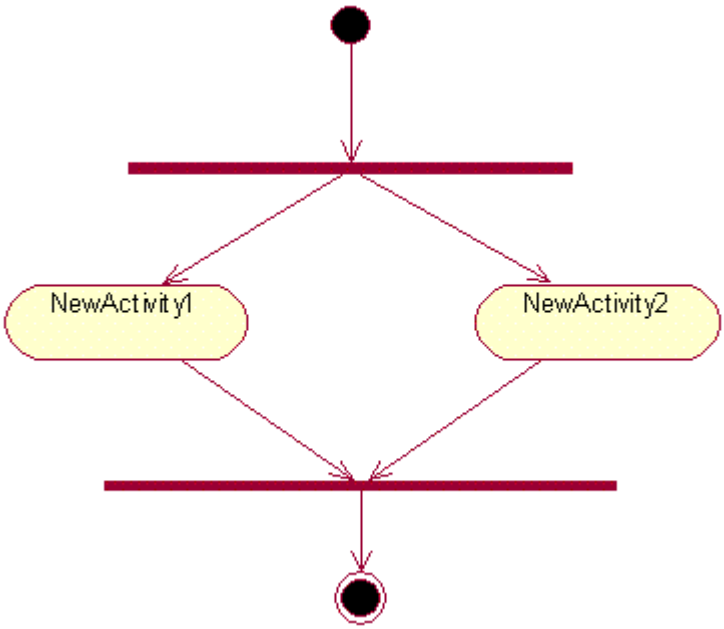
دکمه اول (State) : بیانگر حالت سیستم در یک جریان کار می باشد . (در درس State Diagram بیشتر در مورد آن صحبت می کنیم)

دکمه دوم (Activity) : بخشی از یک جریان کار (Workflow) است که باید انجام شود ، به عبارت دیگر قسمتی از کلمه ای است که در یک جریان کار اجرا می شود .

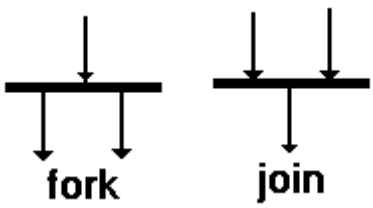
دکمه سوم (Start state) : بیانگر شروع یک جریان کار است .

دکمه چهارم (End state) : بیانگر اتمام یک حالت است .

دکمه پنجم (Horizontal Synchronization) : نحوه کاربرد آن برای همزمانی فعالیتها است . به شکل زیر توجه کنید . در این شکل مشخص شده NewActivity1 با NewActivity2 همزمان شروع به فعالیت می کنند .



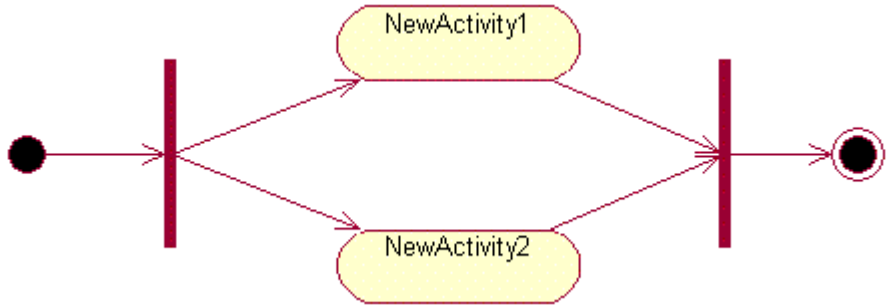
نکته : با میله های همزمان سازی می تواند بطور همزمان دو یا چند انتقال در سیستم رخ دهد در این صورت می توان با دو یا چند میله همزمان سازی چند فعالیت موازی را join یا fork کرد .



Fork : با اتمام یک فعالیت یا حالت چند فعالیت یا حالت شروع می شود .

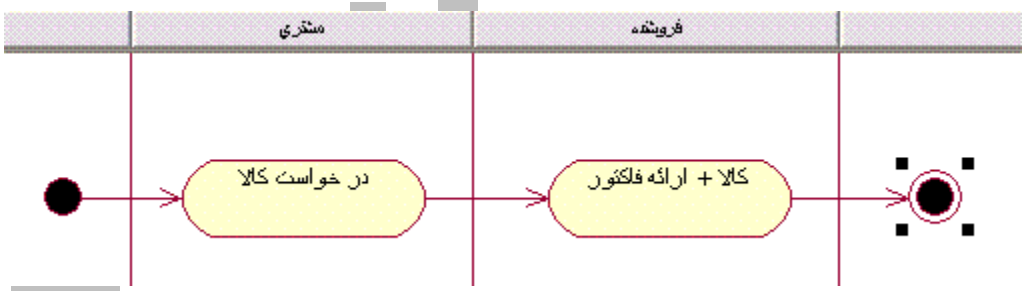
Join : با اتمام چند فعالیت یا حالت یک فعالیت یا حالت شروع می شود .

دکمه ششم (Synchronization Vertical) : کاربرد این دکمه نیز همانند دکمه شماره ۵ است . به شکل زیر توجه کنید . در این شکل نیز مشخص شده NewActivity1 با NewActivity2 همزمان شروع به فعالیت می کنند .



دکمه هفتم (Decision) : برای نشان دادن شرط ها و اما و اگرها از این دکمه استفاده می شود. دقیقاً مشابه شکلی است که در فلوچارتها برای نمایش شرط ها استفاده می شد .

دکمه هشتم (Swim lane) : با این می توان Actor ی که فعالیت را انجام می دهد را مشخص نمود . به شکل زیر توجه کنید . مشتری درخواست کالا می دهد و فروشنده کالا را به همراه فاکتور به مشتری ارائه می دهد .



نمودار حالت State Chart Diagram

نمودار حالت نسبت به نمودارهای توالی و همکاری کمتر مورد استفاده قرار می گیرد . این نمودار همانطور که از نامش پیداست حالت های مختلفی که یک شی در آن قرار می گیرد را مدل می کند . در واقع این نمودار تصویری از چرخه حیات شی (life cycle Object) را به نمایش می گذارد .

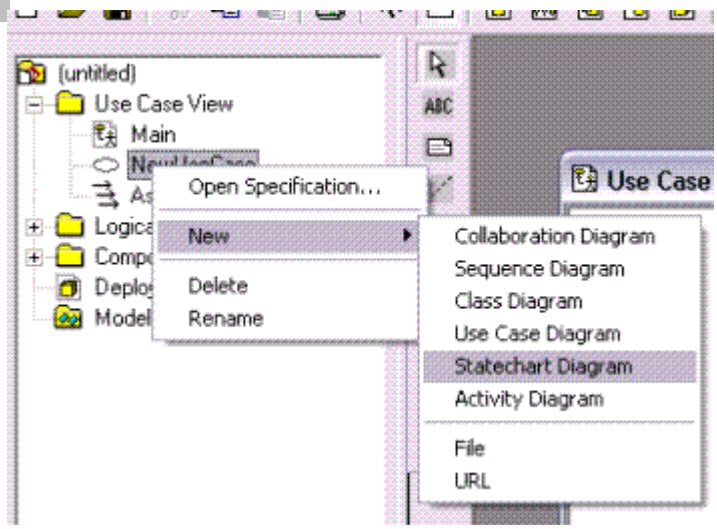
موارد استفاده از نمودار حالت (State Chart Diagram)

۱. اشیائی که دارای تعداد زیادی حالت هستند .
۲. اشیائی که برای Update کردن صفات خاصه خود شروط متنوعی دارند .
۳. اشیائی که معمولاً به صورت سخت افزاری هستند .
۴. اشیائی که عملکرد بعدی آنها به عملکرد قبلی شان بستگی دارد .

اشیا گفته شده بالا اشیائی هستند ، که رسم نمودار حالت معمولاً برای آنها مفید خواهد بود . بررسی نمودارهای حالت به این دلیل است که نمودارها قابلیت این را دارند که یک سری از رفتارهای کلاسها را برای ما مشخص نمایند .

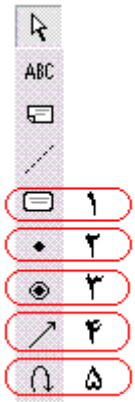
ایجاد نمودار حالت

ایجاد یک نمودار حالت مشابه ایجاد نمودارهای توالی ، همکاری یا فعالیت می باشد . برای این ایجاد یک نمودار حالت کافی است بر روی Use Case مورد نظر کلیک راست کرده و میسر New > StateChart Diagram را دنبال کنید . ملاحظه خواهید کرد که یک نمودار حالت به Use Case شما اضافه شده است . به شکل زیر توجه کنید .



نوار ابزار نمودار حالت

این دیاگرام نیز شبیه سایر دیاگرامهای Rose برای خود دارای نوار ابزار مخصوص به خود است . در زیر تصویر آن به همراه تشریح دکمه هایی که مخصوص این است را ملاحظه می کنید .



دکمه اول (State) : بیان کننده حالت یک شی می باشد .

دکمه دوم (Start State) : نقطه شروع چرخه حیات شی است .

دکمه سوم (End State) : بیان کننده نقطه پایانی چرخه حیات یک شی می باشد یعنی طول عمر شی در این نقطه به پایان می رسد .

دکمه چهارم (Transition State) : مسیری برای عبور شی از یک حالت به حالت دیگر را نشان می دهد .

دکمه پنجم (to self Transition) : مسیری را نشان می دهد که شی در آن تغییر حالت نمی دهد یعنی حالت تغییری پیدا نمی کند .

انواع فعالیتهای یک حالت (State)

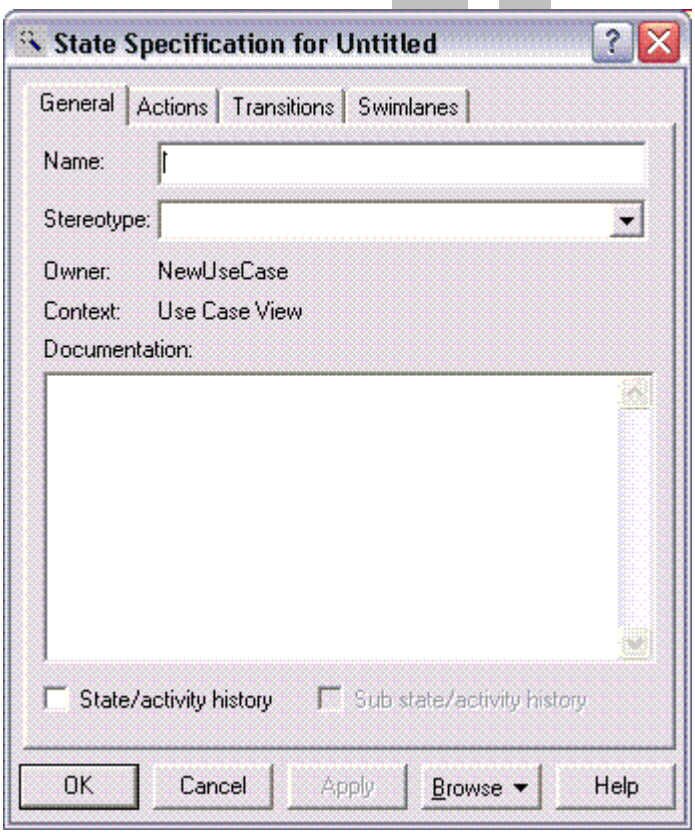
۱. Entry : مجموعه فعالیتهایی که در زمان ورود شی به یک حالت (State) باید انجام گیرند را در Entry قرار می دهند .
۲. Exit : مجموعه فعالیتهایی که در زمان خروج شی از یک حالت (State) باید انجام شوند را در Exit قرار می دهند .

Do : حالت (State) برای انجام یک سری فعالیت ایجاد شده است یعنی شی به یک حالت (State) می رود تا یک سری عملیات را انجام دهد . این فعالیت ها با نام Do در حالت (State) قرار می گیرند .

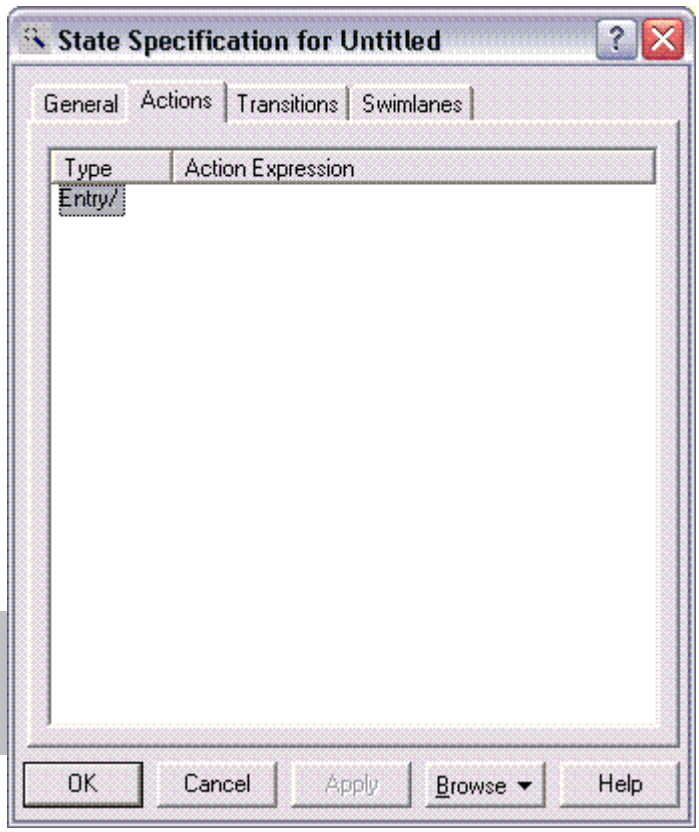
نحوه انتقال State از نوار ابزار به داخل State Chart Diagram

برای این کار کافی است بر روی شی State کلیک کرده تا به حالت انتخاب در آید . آنگاه به داخل صفحه دیاگرام رفته و یک بار کلیک می کنیم . می بینید که یک State به نمودار شما اضافه شده است . به شکل زیر توجه کنید .

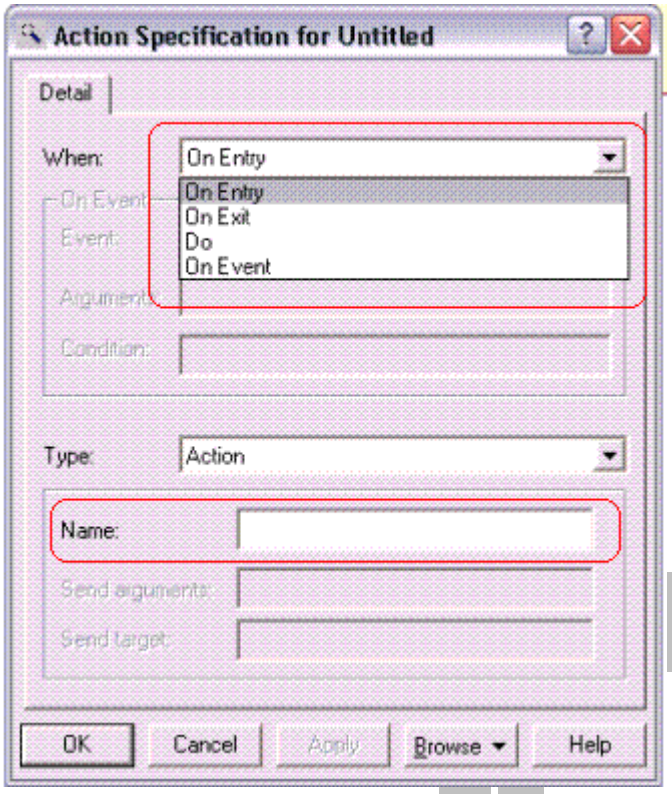
در مورد State ها آنچه باید گفت این است که می توانید روی آن دوبار کلیک نمایید . منوی زیر برای شما باز می شود . در قسمت Name می توانید نام State را وارد کنید .



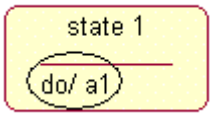
قسمت مهم Action ها هستند که ما می توانیم آنچه در رابطه با Entry و Do و Exit گفتیم مشاهده کنیم . در تب Actions می توانید کلیک راست کنید و گزینه Insert را انتخاب کنید ، ملاحظه می کنید که یک Entry اضافه شده است .



بر روی Entry می توانید دابل کلیک کنید ، در قسمت مشخص شده در شکل زیر می توانید نوع event را مشخص کنید . عملی که باید انجام شود را در قسمت Name وارد کنید .



خواهید دید که event انتخابی شما به همراه عملی که باید انجام شود به نمودار شما اضافه شده است . در شکل زیر event انتخابی Do می باشد .



برای Transition ها نیز ابتدا آن را انتخاب کرده و سپس از مبدا به مقصد آن را می کشیم . در مورد Transition ها باید گفت آنچه مهم است event است که منجر به چنین Transition شده است .

همانطور که در درس گذشته بیان کردیم ، عمده کاربرد نمودار حالت در مدل سازی اشیاء سخت افزاری می باشد . به همین دلیل ما در این درس قصد داریم رفتار یک عابربانک را با نمودار حالت (StateChart Diagram) مدل کنیم .

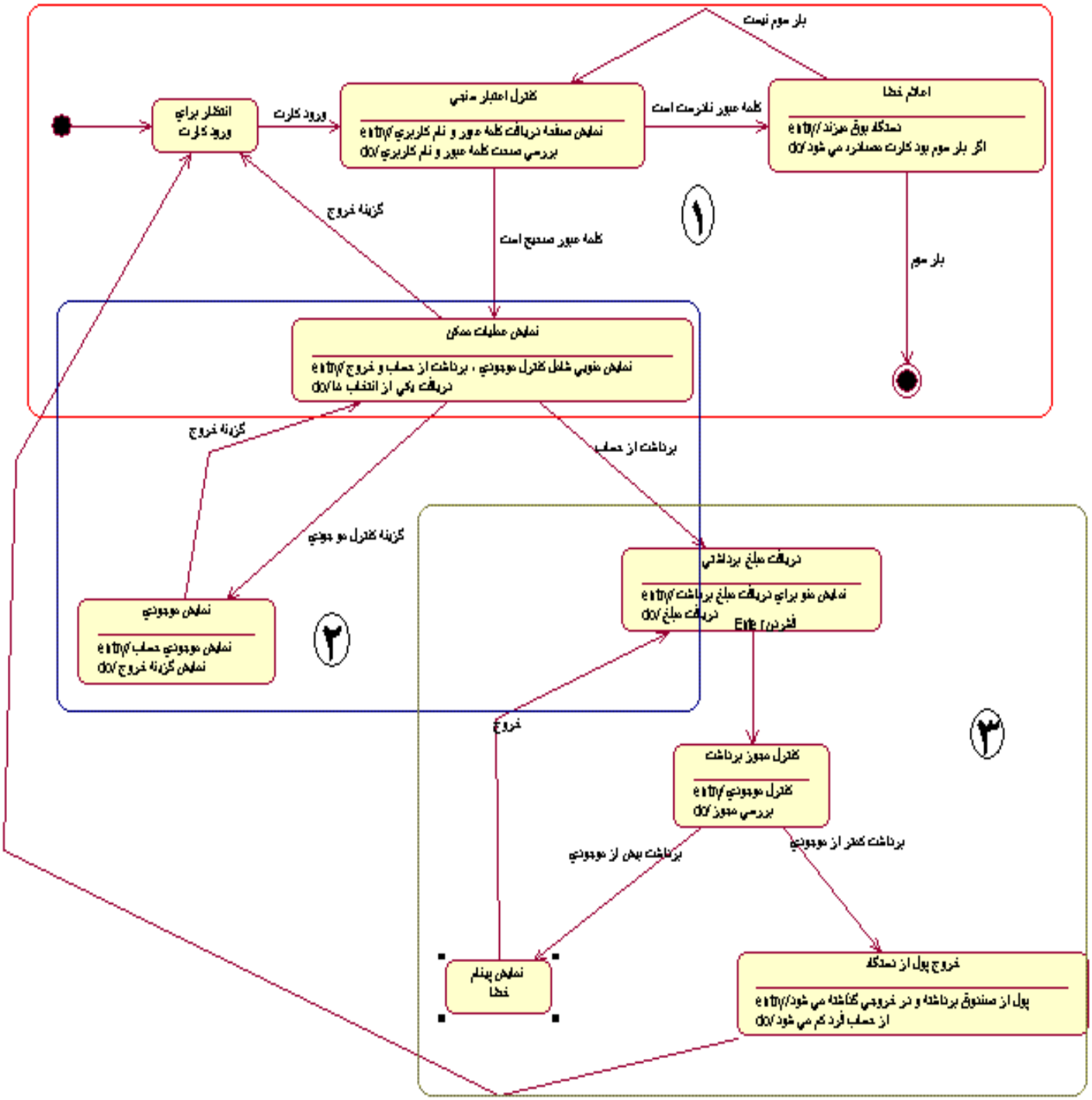
فرض کنید یک Use Case به نام "پرداخت اتوماتیک" داریم که می خواهیم برای آن یک نمودار حالت را بر اساس فایل الصاقی به آن رسم کنیم . با هم دیگر نگاهی به این فایل الصاقی می اندازیم .

رفتار عابربانک

۱. دستگاه منتظر ورود کارت می ماند .
۲. با ورود کارت کلمه عبور و نام کاربری پرسیده می شود .
۳. درصورت صحت کلمه عبور و نام کاربری منویی برای فرد نمایش داده می شود که شامل سه گزینه برداشت پول ، کنترل موجودی و خروج است .
۴. اگر کلمه عبور و نام کاربری اشتباه بود دستگاه بوق می زند و تا دو بار این کار را انجام می دهد و برای بار سوم اگر کلمه عبور و نام کاربری اشتباه بود کارت را مصادره می کند .
۵. اگر منوی کنترل موجودی انتخاب شد میزان موجودی برای فرد نشان داده می شود .
۶. اگر برداشت انتخاب شد دستگاه صفحه ای را نشان می دهد که منتظر دریافت مبلغ برداشتی است .
۷. مشتری مبلغ را نوشته و Enter می کند .
۸. اگر میزان برداشتی بیشتر از موجودی بود یک پیغام به کاربر نمایش داده شود .

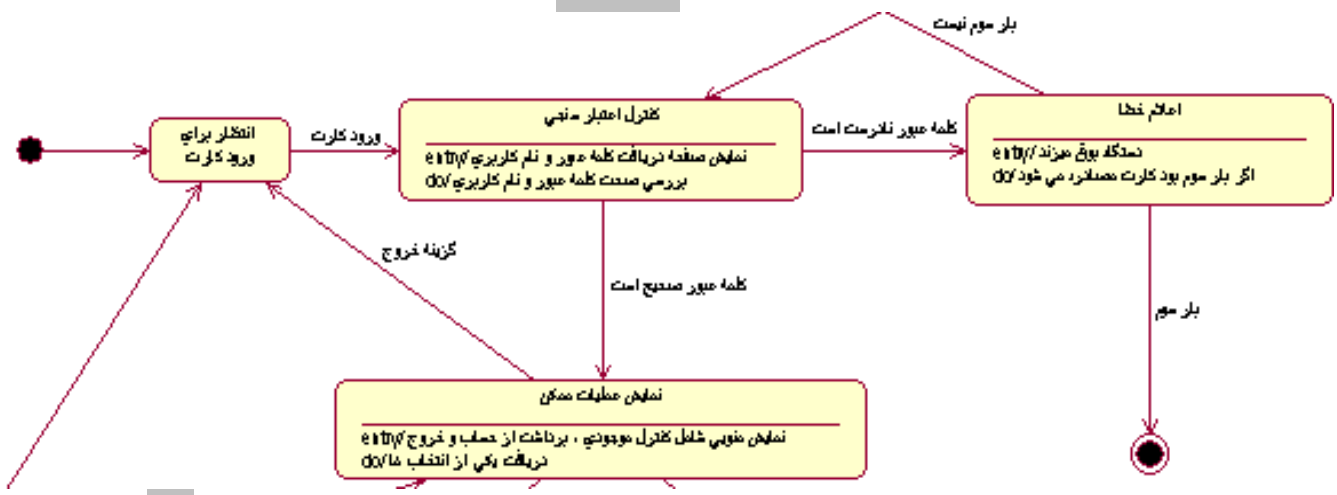
۹. اگر میزان برداشتی کمتر از موجودی بود پول را از خروجی دستگاه به مشتری می دهد .

اکنون قصد داریم در این مرحله از روی این متن یک نمودار حالت را برای عابر بانک تهیه کنیم . دیاگرام زیر شمای کلی نمودار حالت ما می باشد که برای سادگی کار آن را به سه قسمت تقسیم کرده ایم .



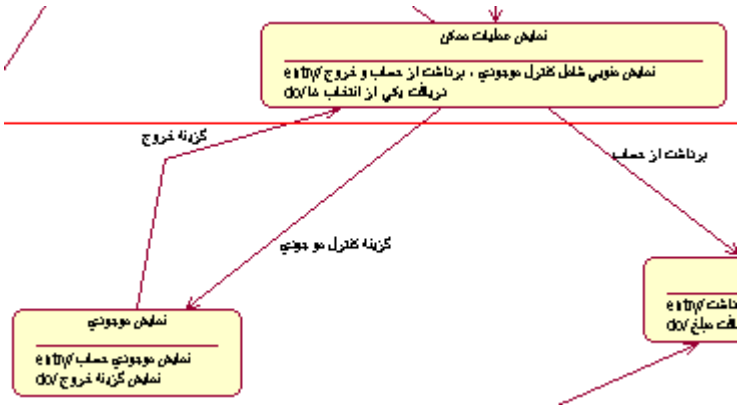
قسمت اول :

ابتدا دستگاه در حالت شروع قرار می گیرد و با روشن شدن وارد State " انتظار برای ورود کارت " می شود . در این مرحله دستگاه با ورود کارت به یک مرحله جدید با عنوان " کنترل اعتبار سنجی " می رسد که کلمه کاربری و رمز عبور را دخواست می کند . اگر کلمه کاربری و رمز عبور اشتباه وارد شود سیستم وارد State " اعلام خطا " می شود ،دستگاه بوق میزند و تا سه مرتبه این حالت می تواند تکرار شود و بعد از بار سوم کارت مصادره می گردد . اگر کلمه کاربری و رمز عبور صحیح بود دستگاه وارد State " نمایش عملیات ممکن " می شود که دارای سه گزینه کنترل موجودی ، برداشت از حساب و خروج می باشد . اگر گزینه خروج انتخاب شود دوباره به State " انتظار برای ورود کارت " می رویم .



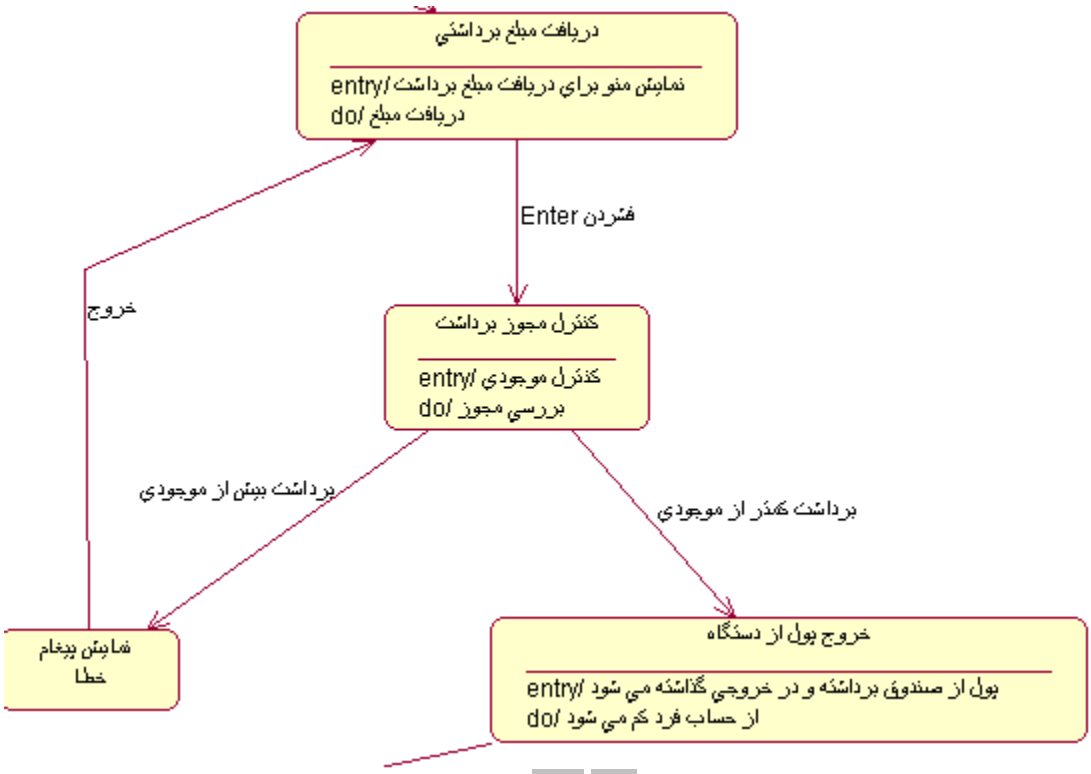
قسمت دوم :

با انتخاب گزینه نمایش موجودی در State " نمایش عملیات ممکن " به State " نمایش موجودی " می رویم . در این State به محض ورود مشتری از میزان موجودی در حساب خود آگاه می شود و گزینه برای خروج در ادامه کار برای او نمایان می شود . با انتخاب گزینه خروج دوباره به State " نمایش عملیات ممکن بر می گردیم .



قسمت سوم :

با انتخاب گزینه برداشت از حساب در State "نمایش عملیات ممکن" به State "دریافت مبلغ برداشتی" می رویم . منویی ظاهر می شود و منتظر وارد شدن مبلغ از طرف مشتری می ماند . بعد از وارد کردن مبلغ توسط مشتری و فشردن کلید اینتر به State " کنترل مجوز برداشت می رویم . با ورود به این State موجودی حساب مشتری کنترل و سپس بررسی می شود که آیا مشتری مجاز به برداشت است یا خیر . در صورت مجاز بودن به State "خروج پول از دستگاه" می رویم که پول درخواستی در خروجی دستگاه قرار داده شده و مبلغ از حساب مشتری کم می شود آنگاه دستگاه دوباره به State "انتظار برای ورود کارت" می رود . در صورت غیر مجاز بودن مبلغ درخواستی مشتری دستگاه با نمایش پیغام خطا دوباره به State "دریافت مبلغ برداشتی" می رود .


دریافت مبلغ برداشتي
entry/ نمایش منو براي دریافت مبلغ برداشت
do/ دریافت مبلغ
فشردن Enter
کنترل مجوز برداشت
entry/ کنترل موجودي
do/ بررسي مجوز
برداشت بیش از موجودي
برداشت کمتر از موجودي
خروج
نمایش پیغام خطا
خروج پول از دستگاه
entry/ پول از صندوق برداشته و در خروجي گذاشته مي شود
do/ از حساب فرد کم مي شود

تفاوت هاي State chart و Activity Diagram

۱. در Statechart ابتدا يک Object را فرض مي كنيم و كليه حالات مربوط به آن را كه در اثر وقوع Event به وجود مي آيد را مدل مي كنيم .
در حاليكه در دياگرام فعاليت به دنبال تغيير حالت نيستيم . بلكه مي خواهيم يک فعاليت خاص را از ابتدا تا انتها مدل كنيم و ممكن است در انجام اين فعاليت چندين Object دخيل باشد .
۲. در دياگرام Activity براي حركت نياز به وقوع Event نيست و به محض اتمام يک فعاليت كنترل به صورت اتوماتيک ديگر منتقل مي شود . در حاليكه در دياگرام حالت نياز به وقوع Event است .

فعاليت ارسال و دريافت با ساير فعاليت ها متفاوت نشان داده شده اند يک دليلش اين است كه شايد معمولاً اين دسته از فعاليتها به شكل غيركامپيوتري بايستي انجام شود و حداقل اينكه از درجه اهميت پايين تري برخوردار است

کلاس دیاگرام ۱ (Class Diagram)

یک کلاس دیاگرام یک دیاگرام برای توصیف یک سیستم است . کلاس دیاگرام شامل آیکون هایی است که کلاسها و نمونه ها و ارتباط بین آنها را نشان می دهد . شما می توانید یک یا بیش از یک کلاس دیاگرام را برای شرح کلاس ها در مدلی سطح بالا بسازید . کلاس دیاگرامها در مدل سطح بالا شامل خودشان نیز می باشند .

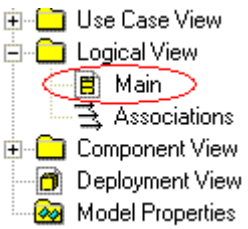
با وجود اینکه نمودار کلاس از نمودار اصلی طراحی شیءگرا می باشد از آن در مرحله تحلیل نیز استفاده می شود . در اینجا ما قصد تولید یک نمودار کلاس تجزیه و تحلیل را داریم . یعنی هدف از ایجاد این نمودار دراین مرحله پیدا کردن مفاهیم مهم سیستم و در نهایت درک مشکلات و نیازمندیهای مشتری می باشد یا به عبارتی در اینجا فقط مفاهیم و ارتباطات بین آنها به تصویر کشیده می شود .

نمودارهای کلاس (Class)

یک نمودار کلاس برای نمایش تعدادی از کلاس ها و بسته های کلاس در سیستم شما استفاده شده است . این نمودار یک تصویر ایستا از قطعات سیستم و ارتباط بین آنها را به شما می دهد .

شما معمولاً برای یک سیستم واحد چندین نمودار کلاس را ایجاد خواهید کرد . برخی از اینها زیر مجموعه ای از کلاس ها و روابط آنها را نمایش خواهند داد . شما می توانید بسیاری از نمودارهای کلاس را که نیاز دارید ، بسازید تا یک تصویر کاملی را از سیستم خود بدست آورید .

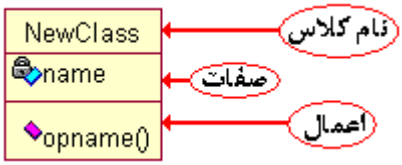
بطور پیش فرض یک نمودار کلاس وجود دارد که Main (اصلی) نامیده شده و مستقیماً زیر نظر نمای Logical است . این نمودار کلاس بسته ها و کلاس های موجود در مدلتان را نمایش می دهد .داخل هر بسته ای نمودار دیگری است که Main (اصلی) نامیده می شود ، که شامل همه کلاس های داخل آن بسته است .



نمودار کلاس یک ابزار طراحی خوب برای تیم می باشند . آنها به برنامه نویسان کمک می کنند تا ساختار سیستم را قبل از اینکه کدی نوشته شود ، ببینند و طراحی کنند و کمک می کنند تا مطمئن شوند که سیستم از ابتدا خوب طراحی شده است .

در نمودار کلاس ، تمام کلاسهای سیستم نشان داده می شود و تمام ارتباط بین آنها بطور کامل باید مشخص شود . به نمونه ای ازاین نمودار که در شکل زیر آمده است توجه کنید . همانطور که دیده می شود هر کلاس از سه بخش تشکیل شده است .

۱. نام کلاس که می توان در ادامه نام کلاس نام بسته را نیز بکار برد .
۲. صفات که می تولند به سه صورت Private , Public , Protected باشند .
۳. اعمال که می توانند به سه صورت Private , Public , Protected باشند .



منابعی جهت استخراج کلاسها

آبجکت های مشابه در نمودارهای تعاملی
بطور کلی اشیاء فیزیکی
ابزار ها
مکانها
نمودار جریان رخداد
وسائل
الگوریتمهای اجرائی
نقش اشخاص (مشتری ، کارمند و ...)
سیستمهای دیگر
....

راههای تجربی جهت شناسایی کلاسها

۱. کلاسها معمولاً حاوی اطلاعاتی هستندکه سیستم قصد دارد بصورت دراز مدت آنها را ثبت و پردازش کند .
۲. یک کلاس حتماً باید دارای صفت باشد .
۳. یک کلاس حتماً یک کلید داشته باشد که کلاس و اجزاء آن را تفکیک نماید .
۴. یک کلاس باید بیش از یک نمونه را تولید کند .

Stereotype کلاسها

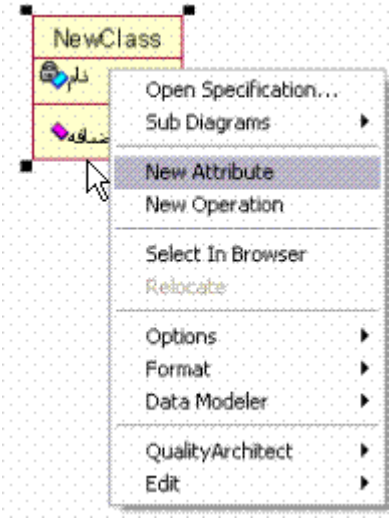
انواع Stereotype های کلاسها در درس هشتم توضیح داده شده است . ما فقط بطور مختصری آنها را یادآوری می کنیم .

Actor یا عامل ، Boundary : که به معنای User Interface هستند ، Control : این آبجکت ها همان اشیاء کنترلی هستند ، Entity : اشیای هستند که در سیستم وجود دارند ، Table : جدول از پایگاه داده هستند .

صفات کلاس و افزودن صفات به کلاس

برای یافتن صفات می توان به Use Case ها رجوع کرده و در جریان رخدادها اسامی را پیدا نمود .

برای افزودن صفات به کلاس کافی است بر روی کلاس مورد نظر کلیک راست نموده و سپس گزینه New Attribute را انتخاب کنید . به این ترتیب می توانید به کلاس خود صفات جدید اضافه کنید .



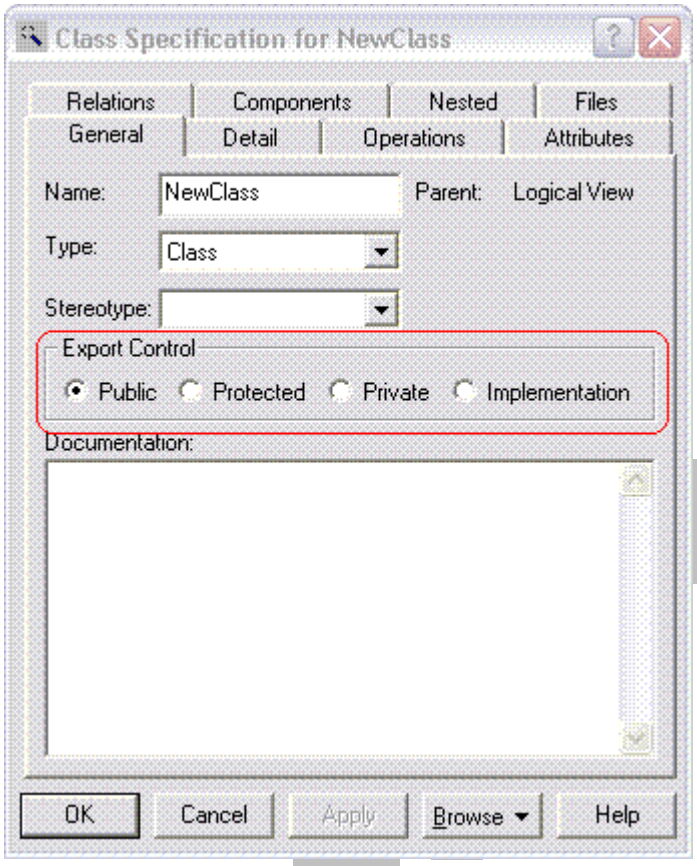
یک صفت از لحاظ Visibility چهار وضعیت زیر را می تواند داشته باشد :

عمومی (Public) : صفت برای تمامی کلاسهای دیگر نیز قابل روئیت است . در UML از علامت + برای نمایش دادن این صفات استفاده می شود .

خصوصی (Private) : این نوع صفات برای کلاسهای دیگر قابل روئیت نیست . در UML از علامت - برای نمایش دادن این صفات استفاده می شود .

محافظت شده (Protected) : این نوع کلاس فقط بتوسط همان کلاس و فرزندانش قابل روئیت است . در UML از علامت # برای نمایش دادن این صفات استفاده می شود .

Implementation : فقط کلاسهای یک بسته می توانند به صفات یکدیگر دسترسی داشته باشند .

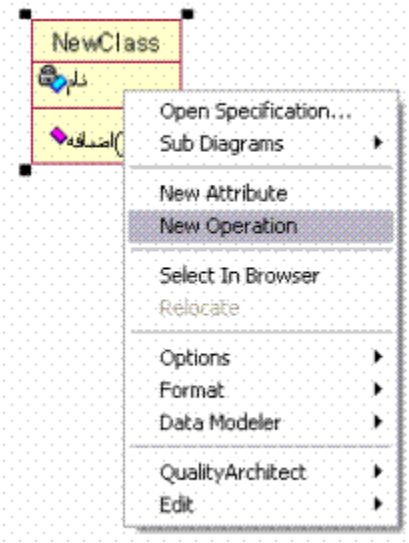


عملیات و یافتن عملیاتها

رفتارها و عملکردهای مربوط به یک کلاس هستند . با رجوع به نمودارهای توالی و همکاری می توان عملیاتها را استخراج کرد . بعد از مشخص شدن عملیلتها موارد زیر را در مورد کلاسها بررسی می نمائیم .

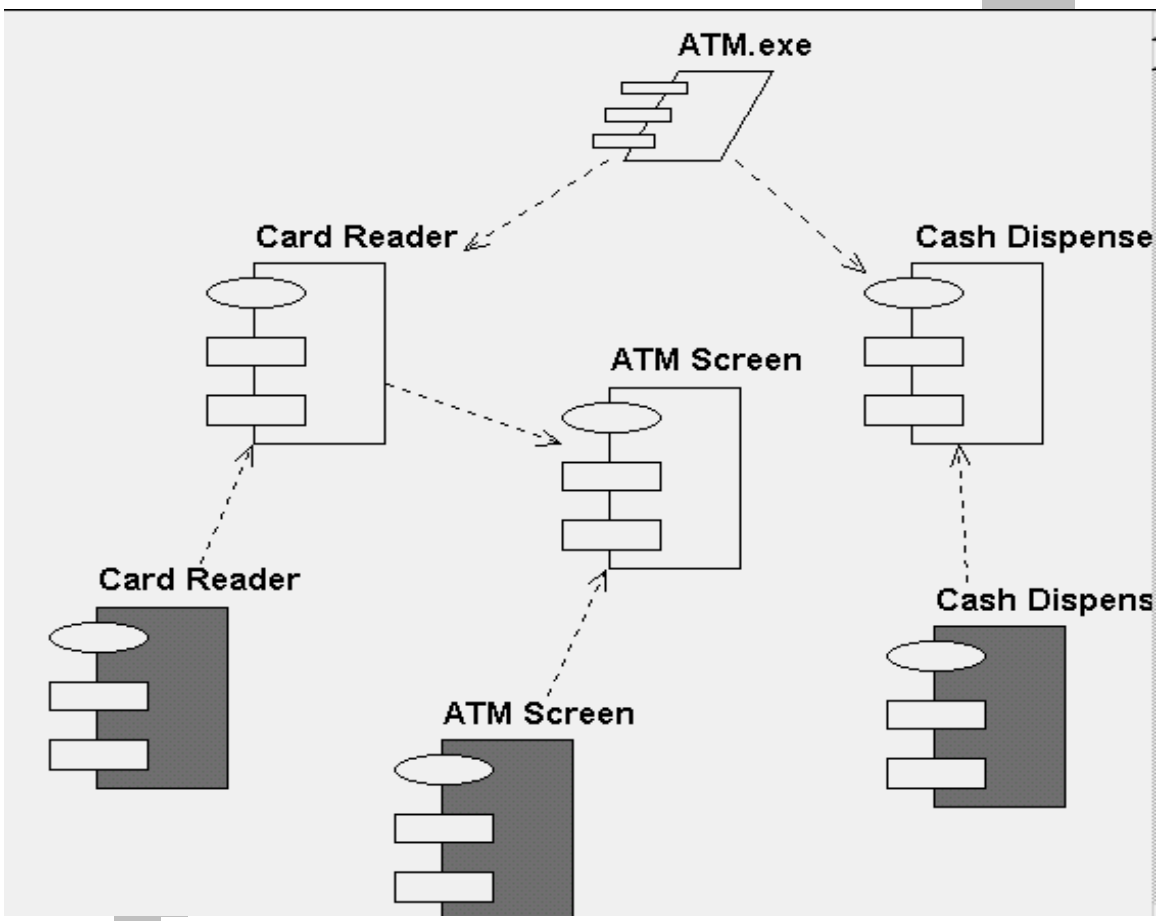
۱. کلاسهایی که دارای یک یا دو عملیات هستند بهتر است با هم ترکیب شوند .
۲. کلاسهایی که فاقد عملیات هستند بهتراست به عنوان یک یا چند صفت مدلسازی شوند .
۳. کلاسهایی که عملیاتهای زیادی دارند را بهتر است به دو یا چند کلاس کوچکتر تقسیم نمود .

برای افزودن عملیات به کلاس کافی است بر روی کلاس مورد نظر کلیک راست نموده و سپس گزینه New Operation را انتخاب کنید . به این ترتیب می توانید به کلاس خود عملیات جدید اضافه کنید .

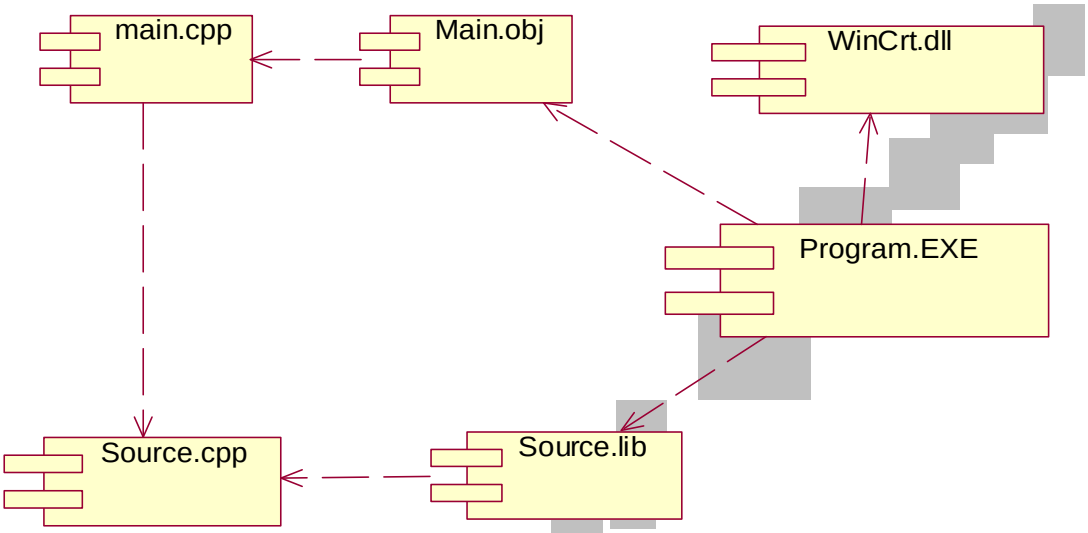

NewClass
Open Specification...
Sub Diagrams
New Attribute
New Operation
Select In Browser
Relocate
Options
Format
Data Modeler
QualityArchitect
Edit

Component Diagram:

نمودارهایComponent يك ديد فيزيكي از مدل ارائه مي نمايد. اين نمودار اجـزاي نـرم افـزاري سيستم و روابط بين آنها را نشان مي دهد.
در این نمودار نماد کامپوننت استفاده می شود. ایجاد این دیاگرام در فاز Elaboration است و در فاز Construction به روزآوری می شود. در این نمودارطراح مشخص می کند که چه کلاسهایی در چه Component هایی باید قرار بگیرد و Component ها از چه interface هایی استفاده می کنند.(توجه : تحلیلگر با این نمودارکاری ندارد).
توجه : نام DLL نام Class و نام Component ها و غیره باید توسط طراح مشخص شود.

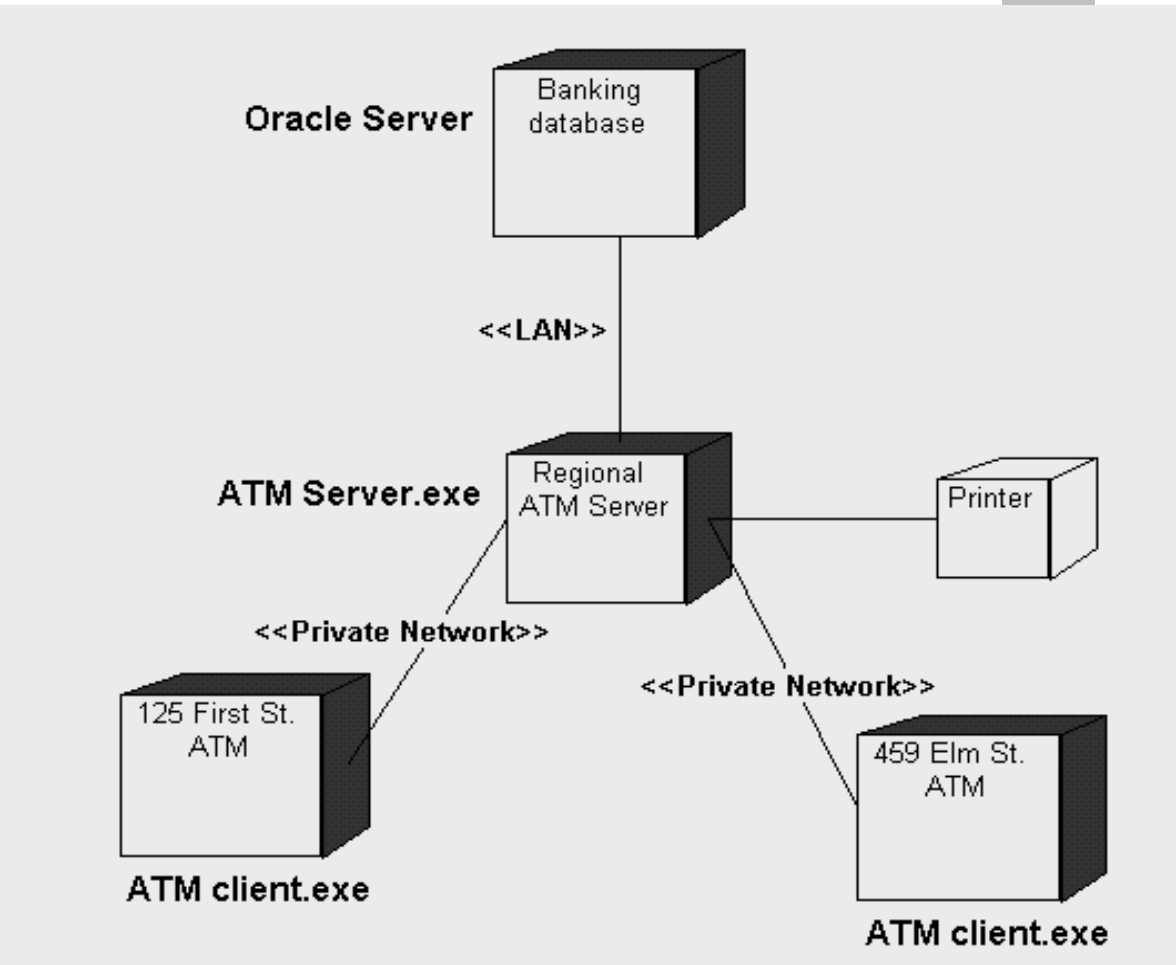


مثال : فرض کنید که مي خواهیم دیاگرام Component مربوط به یک نرم افزار در محیط Windows ترسیم نماییم فرض کنید کل آن به زبان C^{++} نوشته شده باشد در این صورت شکل زیر حاصل خواهد شد :

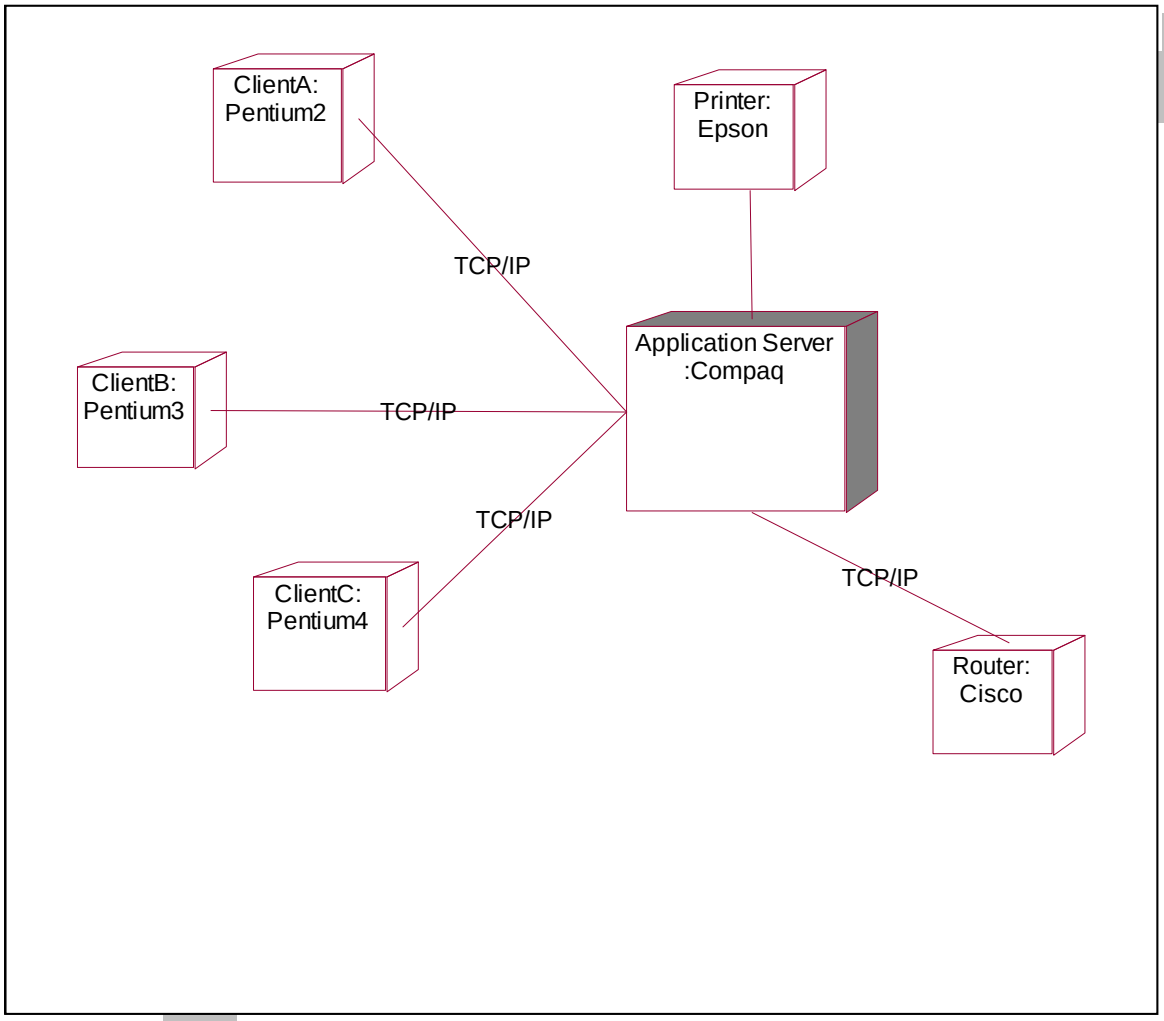


:Deployment Diagram

نمودار Deployment لایه فیزیکي شبکه و جایي که زیرسیستم هاي مختلف با وسایل فیزیکي مجزا مقیم خواهند شد
چه مولفه های سخت افزاری لازم است تا این نرم افزار بتواند با آن کار بکند. این دیاگرام در فاز Construction کشیده می شود. بعد از این باید مشخص کنیم که هر یک از مولفه های نرم افزاری به چه شکل باید در این بستر پخش شوند، این شکل در فاز Construction باید ارائه شود.



مثال : فرض كنيد اجزاي سخت افزاري مورد نياز در پروژه x به شرح زير مي باشد يک Compaq Server تعدادي كامپيوتر Client از نوع PC كه به سه گروه كامپيوترهاي IV III II Pentuin طبقه بندي شده و با پروتكل هاي TCPIP به Server متصل مي باشد يک چاپگر Epson و يک خط ارتباطي به يک روتر اتصال دهنده به Internet با استفاده از دياگرام Deployment اين اجزا را مدل نماييد :



مفاهيم شي ء گرايي

چرا شي ء گرايي خوب است ؟ در مهندسي نرم افزار همیشه یک مشکل مساله (Problem Domain) مطرح مي شود و براي آن یک راه حل(Solution Domain) ارائه مي شود. همواره سعي شده که راه حل به مساله نزدیک شود و بتواند تا حدود زیادي آنرا پوشش مي دهد. به تجربه ثابت شده است که راه حلهای مبتنی بر شیء گرایی اینکار را به بهترین وجه انجام داده اند.

۱ - مفهوم شیء(Object)

تعریف شیء : موجودیتي در دنیاي واقعي که براي ما ملموس باشد ، بتوانیم آنرا توصیف کنیم و داراي ۳ مشخصه باشد:

۱- شناسه (ID) منحصر به فرد داشته باشد.
۲- وضعیت (State) داشته باشد.
۳- رفتار (behavior) داشته باشد.

اشیاءاز طریق فرستادن پیام (Message) با هم ارتباط برقرار مي کنند. مفهومي با نام Message Passing وجود دارد که به معناي رد و بدل کردن پیام بین اشیاء است. در حقیقت مفهوم Event زیر شاخه اي از Message Passing مي باشد.
Event پاسخي است که یک شیء در برابردرخواست یک شیء دیگر مي دهد.
در دنیاي امروز همه چیز را شیء در نظر مي گیریم. از فایلهاي اجرایی (exe) گرفته تا اشیاء درون آن وتا پایین ترین سطح که به انواع داده ای Integer, String و ... مي رسیم همه و همه شیء در نظر گرفته مي شوند. مزیت شیء در نظر گرفتن آنها در این است که مي توانیم از آنها نسخه هاي زیادي تولید کنیم و اگر یکي از آنها خراب شود اشیاء دیگربه کار خود ادامه مي دهند.

۲- کلاس

کلاس طبقه بندي از اشیاء است و وظایف و توانمندیهای اشیاء را بیان مي کند. امروزه کلاس را از ۲ دیدگاه مورد بررسي قرار مي دهند.

- دیدگاه ساختاري (Structural) : توانایي و مقدار کاري است که کلاس مي تواند انجام دهد.
- دیدگاه رفتاري (Behavioral) : این کلاس با کلاسهاي دیگر چگونه ارتباط برقرار مي کند.

برای مثال می توان خودرو را به عنوان یک کلاس در نظر گرفت.

۳- وراثت(Inheritance)

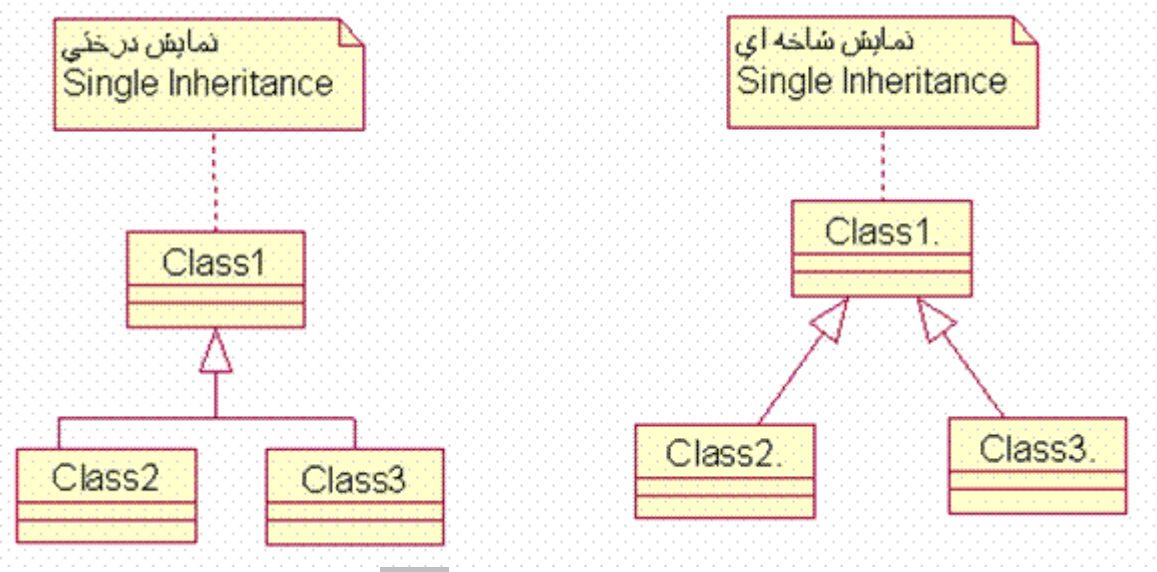
موجودي داريم به نام A که موجود ديگري به نام B از آن ارث مي برد هر چه که A دارد B هم دارد. رابطه وراثت بين دو کلاس را با يک پيکان که در يک سر آن يک مثلث وجود دارد نمايش می دهند.

وراثت را به دو صورت درختي و شاخه اي مي توان نمايش داد.

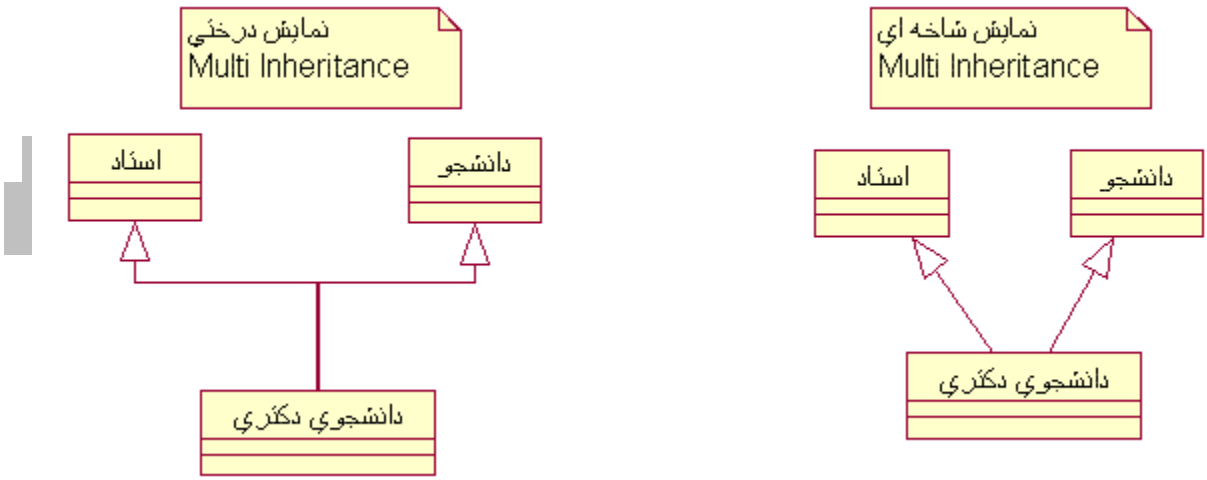
وراثت بر دو نوع است:

Single Inheritance: هر فرزند فقط يک والد دارد.

Multi Inheritance : هر فرزند مي تواند بيش از يک والد داشته باشد.



مثال مشهور اين نوع وراثت همان دانشجوي دکتري است که هم از کلاس دانشجو و هم از کلاس والد ارث مي برد.



نكته مهم : تحليل گر مي تواند Multi Inheritance استفاده كند ولي طراح بايد حتماً از Single Inheritance استفاده كند (در صورتي كه تحليل گر از Multi Inheritance استفاده كند طراح بايد آنرا به Single Inheritance تبديل كند) زيرا كامپايلرهای جديد اين امكان را براي برنامه نويس فراهم نمي كنند كه برنامه نويس بتواند آنرا پياده سازي كند.
وراثت تا هر سطح كه لازم باشد مي تواند ادامه يابد.
نكته ۱: به آخرين كلاسها در درخت وراثت برگ (leaf) مي گويند.
نكته ۲: يكي از خصوصيات وراثت اين است كه فرزند را مي توان به جاي پدر معرفي كرد.
نكته ۳: ارث بري مي تواند در n لايه باشد و هيچ تاثيري هم در كندي سيستم ندارد ولي در مجموع در بعضي موارد نبايد از وراثت استفاده كرد كه در طراحي بحث مي كنيم.

Final & Sealed: بعضي اوقات نمي خواهيم ارث بري از يک سطح خاص به بعد ادامه پيدا كند به اين معني كه كسي نتواند كلاسي را به ارث برد. در اين حالت آن كلاس را به صورت Final يا Sealed تعريف مي كنيم.
اين كار ممكن است به دو علت انجام گيرد : دليل اول امنيت و ديگري سرعت.

- امنيت : در شي ء گرايي مي توان پسر را به جاي پدر معرفي كرد. بنابراين مي توان يک كلاس را از كلاس ديگر به ارث برد و توابع و خصوصيات آنرا به دلخواه تغيير داد و در نهايت آنرا به جاي پدر معرفي كرد در اين صورت كنترل برنامه را مي توان به دلخواه در دست گرفت.
- سرعت : زمانی كه ارث بری از يک سطح به بعد قطع می شود كامپايلر بجاي اينكه فضاي ديناميک براي كلاس در نظر بگيرد استاتيک در نظر مي گيرد چون مثلاً از كلاس C به بعد چيزي به ارث نمي رود پس حافظه ثابت است. (نكته : امروزه اين دليل اهميت چنداني ندارد)

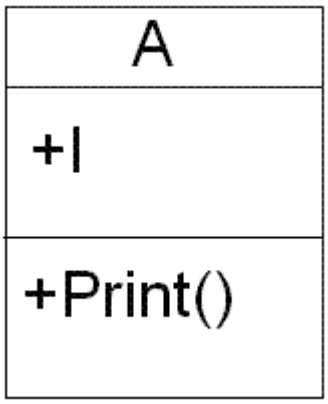
۴- كپسوله سازي (Encapsulation)

دور موجوديت يک پوسته (Capsulate) قرار مي دهيم. كاربردهاي كپسوله سازي به شرح زير هستند:

- پيچيدگي هاي درون سيستم را براي كاربر نهايي از بين مي برد . مثل ADO كه به صورت يک پوسته دور موتور بانک اطلاعاتي را پوشانده است و پيچيدگي هاي كار با بانک اطلاعاتي را از بين برده است.
- ازگسترش تغييرات در سطح سيستم جلوگيري مي كند. مثال درياچه
- باعث پنهان شدن نحوه پياده سازي درون كلاس از ديد كاربر بيرون مي شود.
- باعث پنهان شدن ساختار داده هاي درون كلاس مي شود. اين مفهوم با نام hiding Information شناخته شده است.

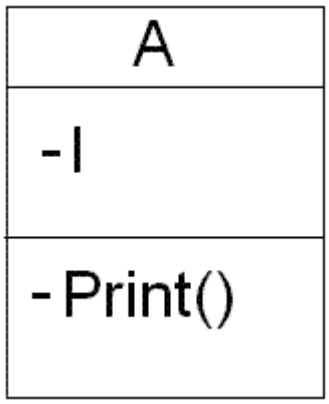
كپسوله سازی باعث بوجود آمدن سه كلمه كليدي Private و Public و Protected در سطح كلاس شد.

از نظر تحليل گر و طراح ، Public توسط علامت + نمايش داده مي شود.
كلاس A در شكل روبرو داراي يک خاصيت و يک متد با سطح دسترسي Public است :



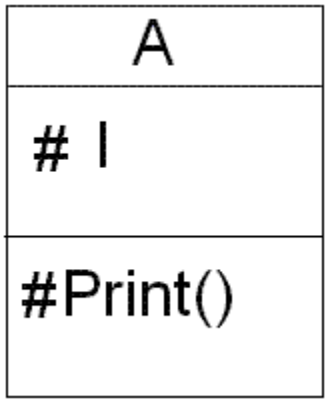
در اين حالت I و Print() هم از بيرون كلاس A قابل دسترس هستند و هم از درون تمام كلاسهايي كه A را به ارث مي برند قابل دسترس هستند.

كلاس A در شكل زير داراي يک خاصيت و يک متد با سطح دسترسي Private است :



در اين حالت I و Print() نه از بيرون كلاس A قابل دسترس هستند و نه از درون كلاسهايي كه A را به ارث مي برند.

كلاس A در شكل زير داراي يک خاصيت و يک متد با سطح دسترسي Protected است :



در اين حالت I و Print() از بيرون كلاس A قابل دسترس نيستند ولي از درون كلاسهايي كه A را به ارث مي برند قابل دسترس هستند.

كلاس مجرد (Abstract Class)

كلاسي كه حداقل يكي از متدهايش داراي پياده سازي نباشد Abstract ناميده مي شود. فرض كنيم كه كلاسهايي داريم كه داراي بخش هاي مشترک هستند. در اين حالت مي توانيم بخش مشترک بين كلاس ها را يک لايه بالاتر بياوريم تا بقيه كلاسها از آن ارث ببرند.

- كلاس تجريد فقط حالت تعريفي دارد و واقعي نيست، يعني نمي توان از آن Object ساخت.

نكته : در تجريد به كلاس پدر Supper Class مي گويند. و به كلاس ارث برنده Sub Class مي گويند.Sub Class اگر در سطح آخر ارث بري باشد و كلاس ديگري از آن ارث نبرد Leaf Class نيز ناميده مي شود.

مثال : فرض كنيم دو كلاس داريم به نامهاي دايره (Circle) و مربع (Rectangle) كه داراي متد Draw هستند. در اين حالت با استفاده از تجريد يک كلاس به نام Shape مي سازيم كه داراي تعريف متد Draw است ولي بدنه ندارد. سپس دو كلاس Circle و Rectangle از آن ارث ببرند و هر كدام بدنه Draw را مطابق نيازهاي خود پياده سازي كنند.

نكته : اگر نام كلاسي بصورت italic بود Abstract است.

۵ - چند ریختی (Polymorphism)

این مفهوم از دو دیدگاه Override و Overload بررسی می شود.

Override: کلاسهای فرزند رفتار کلاس پدر را بازنویسی می کنند.
مثال : یک Button به شکل دایره را در نظر بگیرید. این کلید تمام امکانات یک کلید استاندارد را دارد و فقط شکل آن به صورت دایره است. برای ساخت آن یک کنترل از Button به ارث می بریم و سپس متد On Paint آنرا بازنویسی می کنیم. در حقیقت این متد را Override می کنیم.
زمانی که یک متد را Override می کنیم کلمه کلیدی Override را باید قبل از نام متد ذکر کنیم تا Compiler بفهمد که این متد بازنویسی شده است.
Override را به دو صورت می توان به کار برد یا به صورت کامل رفتار پدر را بازنویسی می کند. و یا اینکه رفتار پدر را استفاده می کند و بعضی رفتارها را به آن اضافه می کند در این حالت در متد Override شده ابتدا متد کلاس پدر فراخوانی می شود و بعد از آن کد مربوط به خودمان را می نویسیم.
نکته : Override بیشتر مورد نظر طراح است تا تکنولوژی خود را افزایش دهد.
نکته : در Override می توان رفتارهای کلاس پدر را طبقه بندی کرد.
Overload: برای یک دست کردن یا تمیز کاری Source استفاده می شود.

مثال: در مواردی تابعی داریم که از نظر مفهومی یک انتظار از آن داریم (مثلا چیزی را برای ما چاپ می کند) ولی پارامترهای مختلفی از نظر تعداد و نوع به آن ارسال می کنیم. در این حالت چند تابع می نویسیم که دارای نام مشترک ولی بدنه و پارامترهای متفاوت باشند.
نکته : abstraction رفتار ندارد ولی فرزندهایش رفتارهای آن را پیاده سازی می کنند بنابراین می تواند جزء Polymorphism است.

۶ - سازنده ها و ویرانگرها (Constructor وDestructor)

سازنده (Constructor) تابعی است در درون کلاس که برای مقدار دهی اولیه متغیرهای کلاس (Initialize) استفاده می شود .

چرا برای Initialize کردن کلاس از سازنده استفاده می کنیم؟
- باعث خوانا شدن Source می شود.
- ممکن است قبل از ساخته شدن کلاس ، در زمان Initialize کردن بخواهیم یک سری موارد را کنترل کنیم.

نکته: کلاس می تواند چند سازنده داشته باشد، (نمونه ای از Overload ها)

ویرانگر (Destructor) تابعی است که برای آزاد کردن منابع استفاده شده توسط Object مورد استفاده قرار می گیرد. در هر کلاس فقط یک ویرانگر می توان داشت.
در Net. آزاد سازی متابع توسط Garbage Collector انجام می شود.
نکته : آزاد سازی منابع برای طراح اهمیت زیادی دارد.

نکته : Member Variables
هر Object به اندازه داده هایی که در آن تعریف شده فضا می گیرد. برای مثال وقتی کلاسی به نام employee تعریف می کنید و متغیری به نام FirstName از نوع String در آن تعریف می کنیم، زمان ایجاد شدن یک موجودیت از این کلاس ، برای متغیر مزبور ۲۵۵ کاراکتر فضا در نظر گرفته می شود حتی اگر مقدار متغیر (Ali) باشد که فقط سه کاراکتر فضا می گیرد. این مطلب زمانی مهمتر می شود که برنامه ما بخواهد تحت وب کار کند یا به صورت remoting کار کند. در این حالت انتخاب اشتباه متغیرها و نوع آنها باعث کند شدن برنامه می شود. مثلا در مثال بالا می توان متغیر را از نوع Varchar(50) انتخاب نمود.
نکته : انتخاب صحیح متغیرها و انواع آنها وظیفه طراح است نه تحلیل گر.

متغیرهای استاتیک (Static Variable) :
- بعضی وقتها ما متغیر ها را از نوع Static تعریف می کنیم. در این حالت برای متغیر استاتیک در تمام Object های ساخته شده از آن کلاس فضای مشترک در نظر گرفته می شود.

Reference ها در heap و Static ها در Stack قرار می گیرند.

مهندسی معکوس با استفاده از: Rational Rose

مهندسی معکوس عبارت است از توانایی گرفتن اطلاعات از کد منبع و ایجاد یا ارتقاء مدل Rose .

یکی از موانع موجود بر سر راه پروژه های فناوری اطلاعات سازگار نگاه داشتن مدل آبجکت با کد است . با تغییر نیازها ، تغییر مسقیم کد می تواند وسوسه انگیز باشد ، تا اینکه مدل را تغییر دهید و سپس کد تغییر یافته را از مدل تولید کنید . مهندسی معکوس به ما این امکان را می دهد تا همیشه مدل را با کد همسان نگاه داریم .

در فرایند مهندسی معکوس ، Rose نسبت به خواندن بسته ، Component ها ، کلاسها رابطه ها ، صفات و عملیات از کد اقدام خواهد کرد . هنگامی که این مدل در یک مدل Rose قرار می گیرد ، می توانید هر تغییر لازمی را ایجاد کرده سپس کد را از طریق امکانات مهندسی مستقیم Rose مجدداً تولید کنید .

گزینه هایی که در اختیار شما قرار خواهند گرفت به نسخه مورد استفاده شما بستگی خواهد داشت .

Rose Modeler : شامل هیچ گونه عملیات مهندسی معکوس نخواهد بود .

Rose Professional : شامل قابلیت های مهندسی معکوس به یک زبان است .

Rose Enterprise : شامل مهندسی معکوس C++ ، Visual C++ ، Visual Basic و جاوا خواهد بود .همانطور مهندسی معکوس شمای Oracle 8 را نیز شامل خواهد بود .

Add_ins : متعلق به Rose قابلیتهای مهندسی معکوس در زبانهای دیگر نظیر PowerBuilder یا Forte را به شما خواهند داد .

عناصر مدل ایجاد شده در طول مهندسی معکوس :

در طول مهندسی معکوس ، Rose به جمع آوری اطلاعاتی درباره موارد زیر خواهد پرداخت .

کلاسها

صفات

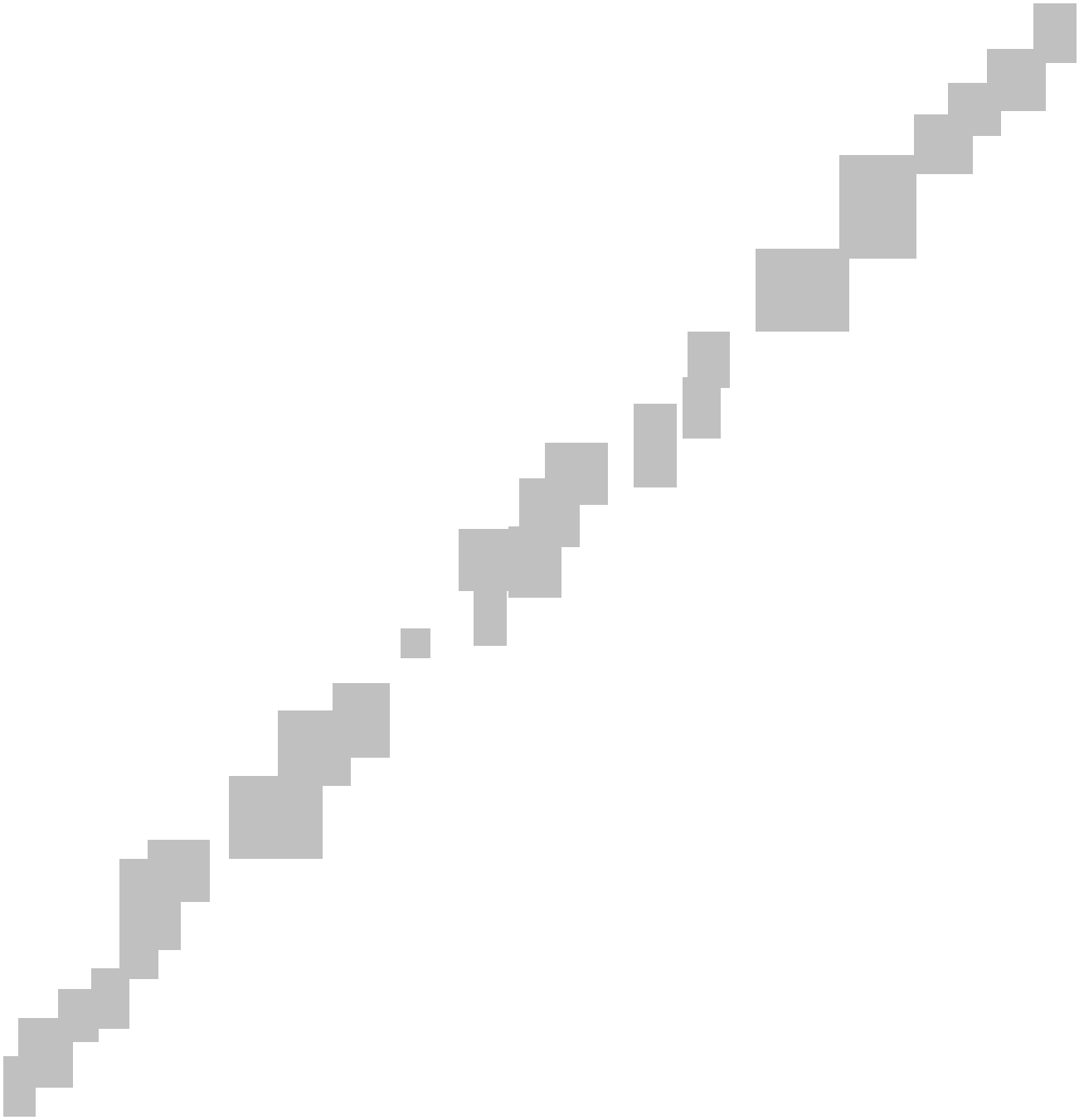
روابط

عملیات

بسته ها

component ها

با استفاده از این اطلاعات ، Rose اقدام به ایجاد یا ارتقاء یک مدل Object خواهد کرد .



This document was created with Win2PDF available at http://www.daneprairie.com.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.