مهندسی نرم‌افزار

Software Engineering

مهندس حمید رضا نیرومند

http://niroomand.ir

منابع:

- Software Engineering - A Practitioner's Approach (5th Ed)(2001) McGraw Hill

  by: Roger S. Pressman, Ph.D.

- Wikipedia.org

- **جزوه درسی استاد شافوری**
- **جزوه پایگاه داده مهندس نیرومند**
- **ویدئوهای شرکت VTC (VTC UML Training Course)**
- **ویدئوهای دانشگاه Harvard**

توجه:

- استفاده از جزوه بدون حضور در کلاس‌ها پیشنهاد نمی‌شود.
- این یک انتشار غیررسمی از جزوه و ویژه دانشجویان مهندس نیرومند است.
- تایپ و ارائه این جزوه توسط دانشجویان انجام شده و هنوز بازبینی و اصلاح نشده است.
- این نسخه ۱,۱,۰ از جزوه است و به مرور غنی‌تر خواهد شد. (لطفاً اشتباهات سهوی که مشاهده می‌کنید را به ایمیل info@niroomand.ir ارسال نمایید.)
- نرم‌افزارهای مورد استفاده در دوره:

Visual Paradigm for UML
Posiedon
PowerDesigner v15.2.0.3042
EA
EDraw
SSADM
Visio
Ratinal Rose 2003

تعریف مهندسی نرم‌افزار (Software Engineering)

Software engineering (SE) is a profession dedicated to designing, implementing, and modifying software so that it is of higher quality, more affordable, maintainable, and faster to build.

مهندسی نرم‌افزار، حرفه‌ای‌ست که مربوط است به طراحی، پیاده‌سازی و ویرایش نرم‌افزار و کمک می‌کند به بالاتر رفتن کیفیت نرم‌افزار، مؤثرتر بودن آن، نگهداری راحت‌تر و ساخت سریع‌تر آن.

واژه مهندسی نرم‌افزار

این واژه اولین بار در سال ۱۹۶۸ در کنفرانس مهندسی نرم‌افزار ناتو به کار برده شد این کنفرانس برای همفکری به دلیل بحران نرم‌افزار Software Crisis که در همین سال‌ها رخ داد برگزار شده بود.

علل پیدایش بحران مهندسی نرم‌افزار

در سال ۱۹۶۰ تا ۱۹۷۰ با افزایش تعداد کامپیوترهای شخصی نیاز کاربران به نرم‌افزارهای مختلف بسیار زیاد شد و این موضوع سبب تولید فراوان نرم‌افزار شد بدون اینکه هیچ قانونی برای نظارت بر تولید نرم‌افزارها در نظر گرفته شود. این بی‌قانونی، سبب به وجود آمدن بحران نرم‌افزار شد که این بحران مقدمه پیدایش مهندسی نرم‌افزار شد.

علل اعلام بحران نرم‌افزار

۱. پروژه‌ها بیش از بودجه‌ی در نظر گرفته شده، هزینه در برداشتند.
۲. پروژه‌ها دیرتر از زمان مورد نظر تحویل داده می‌شدند.
۳. نرم‌افزار تحویل داده شده با مشخصات تعیین شده تطابق نداشت.
۴. کیفیت پایین نرم‌افزار
۵. نگهداری پرهزینه نرم‌افزارها
۶. افزایش کاربردهای کامپیوتر
۷. افزایش پیچیدگی کاربردها
۸. کمبود نیروی انسانی متخصص
۹. افزایش روز افزون قدرت سخت‌افزارها

نکته: یک مهندس نرم‌افزار لزوماً یک برنامه‌نویس نیست، بلکه می‌تواند با تجزیه و تحلیل و به کارگیری ابزارها و تکنیک‌های موجود بهترین راه حل را برای انجام پروژه‌ها انتخاب و پیاده‌سازی کند.

تعریف نرم‌افزار / اجزای نرم‌افزار

نرم‌افزار، مجموعه‌ای است از:

۱. دستورالعمل‌ها (Instructions)
۲. ساختار داده‌ای (Data structure)
۳. مستندات (Documents)

انتظارات از مهندسی نرم‌افزار

1. Low cost of production
2. High performance
3. Portability
4. Low cost of maintenance
5. High reliability
6. Delivery on-time

۱. کاهش هزینه‌ی تولید
۲. کارایی بالا
۳. قابلیت حمل: یعنی قابلیت اجرا بر روی سیستم‌عامل‌ها و سخت‌افزارهای مختلف
۴. کاهش هزینه نگهداری
۵. قابلیت اعتماد بالا
۶. تحویلِ به‌موقع به مشتری

انواع نرم‌افزار

۱. نرم‌افزارهای سیستمی
۲. نرم‌افزارهای بلادرنگ
۳. نرم‌افزارهای تجاری
۴. نرم‌افزارهای علمی و مهندسی
۵. نرم‌افزارهای توکار
۶. نرم‌افزارهای کامپیوترهای شخصی
۷. نرم‌افزارهای تحت وب
۸. نرم‌افزارهای هوش مصنوعی

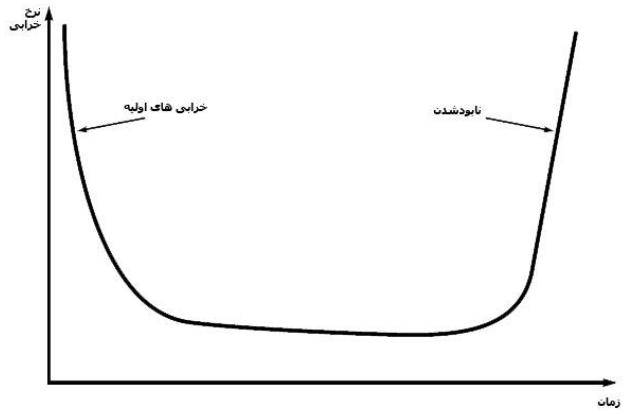
مقایسه سخت‌افزار و نرم‌افزار از دیدگاه مهندسی نرم‌افزار

هر چند به نظر می‌رسد شباهت‌هایی بین یک محصول سخت‌افزاری و نرم‌افزاری باشد اما تفاوت‌های اساسی بین یک پروژه سخت‌افزاری و نرم‌افزاری وجود دارند که عبارتند از:

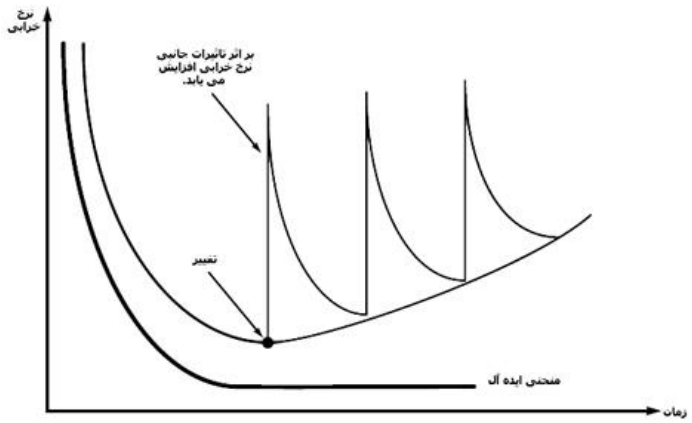
- سخت‌افزار یک عنصر فیزیکی است. انسان آنچه را که قادر به لمس کردن باشد بهتر مدیریت می‌کند، اما نرم‌افزار یک مفهوم منطقی (logical) است و درک آن مشکل است.
- مهم: بررسی نمودار خرابی سخت‌افزار و نرم‌افزار، فرق مهمی را مشخص می‌کند:

در تولید سخت‌افزار پس از مشاهده خرابی‌های اولیه و رفع آن‌ها برای مدتی حالت پایدار (stable) را شاهد هستیم و سپس سخت‌افزار کم‌کم به پایان عمر خود می‌رسد.

اما نرم‌افزارها در هر زمان بسته به شرایط تغییرات (change) دارند و هر تغییر خرابی‌هایی در پی خواهد داشت و می‌توان گفت در نرم‌افزار هیچ‌گاه حالت پایدار نخواهیم داشت، ضمن اینکه به مرور زمان به دلیل تغییرات سیستم چند تکه شده و نرخ خرابی رو به افزایش خواهد بود.



شکل ۱ - نمودار خرابی سخت‌افزار



شکل ۲ - نمودار خرابی نرم‌افزار

فرایند تولید نرم‌افزار

فازهای تولید نرم‌افزار

۱. تعریف
۲. تولید و توسعه
۳. نگهداری

- **فاز تعریف**

در این فاز به تجزیه و تحلیلِ خواسته‌ها و کل سیستم می‌پردازیم:

- چه اطلاعاتی یا داده‌هایی باید پردازش شوند؟
- چه عملیات و یا کارهایی لازم است؟
- چه واسطه‌هایی (نرم‌افزار و سخت‌افزار) لازم است؟
- چه محدودیت‌های وجود دارد؟
- ضوابط ارزیابی سیستم چیست؟

- **سه گام اصلی در فاز تعریف**

۱. **تجزیه و تحلیل سیستم** (System engineering)**:** در این گام نقش هر مؤلفه در سیستم که قرار است تولید شود و نقش کل سیستم تعیین می‌شود به عبارت دیگر حوزه عمل سیستم (Domain یا Scope) و بستر لازم (چه نرم‌افزاری و چه سخت‌افزاری) تعیین می‌شود.
۲. **برنامه‌ریزی** (Planning)**:** در این گام برنامه‌ریزی انسانی، هزینه و زمانِ انجامِ کارها مشخص می‌شود.
۳. **تجزیه و تحلیل نیازهای سیستم** (System Requirements)**:** در این مرحله عملیات مورد انتظار از سیستم تعیین، بانک‌های اطلاعاتی لازم تعریف و سپس ورودی و خروجی نرم‌افزار مشخص می‌شود.
نکته: دقت کنید که در گام سوم به یک تکه از کل نگاه می‌شود، ولی گام اول کل سیستم را می‌بینند که متأسفانه در خیلی از پروژه‌ها به کل توجهی نمی‌شود و در نتیجه سیستم به صورت چند تکه و وصله وصله می‌شود.

- **فاز تولید و توسعه**

در این فاز به چگونگی تبدیلِ گامِ سومِ فازِ تعریف به نرم‌افزار می‌پردازیم.

- **سه گام اصلی در فاز تولید**

۱. **طراحی نرم‌افزار:** در این گام نیازمندی‌های نرم‌افزار به صورت مجموعه‌ای از نمایش‌ها (گرافیکی، جدولی) در می‌آید که بیانگر ساختمان داده‌ها و معماری و الگوریتم‌هاست.
۲. **پیاده‌سازی نرم‌افزار (تولید کد):** در این گام نتایج طراحی با کمک یک زبان برنامه‌نویسی به کد تبدیل می‌شوند.

۳. **نگهداری(تست نرم‌افزار):** در این گام جهت اطمینان از برآورده شدن از نیازهای مشتری قسمت‌هایی مثل Unit test (تست تک تک اجزاء) یا Module test (تست ماژول‌ها) و System test (تست کل نرم‌افزار) صورت می‌گیرد.

- **فاز نگهداری**

در این فاز تغییراتِ لازم پس از تحویل به مشتری انجام می‌شود.

پس از تحویل نرم‌افزار به مشتری معمولاً ممکن است سه دسته تغییر نیاز باشد.

- **انواع تغییرات در فاز نگهداری:**

۱. **تغییرات اصلاحی** (Corrective maintenance)**:** برای مثال سیستم در مورد یک عدد خاص به مشکل برمی‌خورد.

۲. تغییرات تطبیقی (Adaptive maintenance)**:** برای مثال اگر یک شماره به شماره تلفن اضافه شود یا سیستم‌عاملِ محیط عملیاتی عوض شود یا قوانینِ سازمان تغییر کند و یا سخت‌افزار عوض شود و نیاز به تطبیق نرم‌افزار با شرایط جدید باشد.

۳. تغییرات تکمیلی (Perfective maintenance)**:** برای مثال دانشجویانِ شبانه هم پشتیبانی شوند، یا عکس دانشجویان اضافه شود.

ضوابط ارزیابی نرم‌افزار

هدف: تولید برنامه با کیفیت خوب.

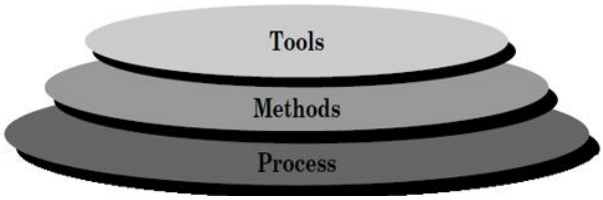
شرایط یک برنامه خوب از نظر کاربر نرم‌افزار (عوامل خارجی) و متخصصین کامپیوتر (عوامل داخلی):

۱. صحت برنامه (Correctness): برنامه باید محاسبات و روال‌ها را به درستی انجام دهد و نتیجه صحیحی نمایش دهد.
۲. استحکام (Robustness): در برابر شرایطِ جدیدِ احتمالی، جواب مناسبی داشته باشد.
۳. قابلیت توسعه (Extendibility): افزودن امکانات جدید به نرم‌افزار به راحتی ممکن باشد.
۴. قابلیت استفاده مجدد (Reusability): سیستم به صورت ماژولار باشد و هر ماژول در سیستم‌های جدیدتر قابل استفاده باشد.
۵. قابلیت حمل (Portability): نرم‌افزار بر روی بسترهای مختلف قابل اجرا باشد.
۶. سازگاری (Compatibility): نرم‌افزار نسبت به تغییراتی که ممکن است در بستر ایجاد شود، سازگار باشد؛ به طور مثال با تغییر یا آپدیتِ مرورگرِ سیستم‌های سازمان، نمای سایتی که طراحی شده به هم نریزد.
۷. کارآیی (Efficiency): سرعت اجرای نرم‌افزار، بالا باشد و مصرف حافظه‌ی آن، پایین.

سه لایه مهندسی نرم‌افزار

Software Engineering is a layered technology.

مهندسی نرم‌افزار یک تکنولوژی لایه لایه است، که لایه‌های آن عبارتند از:



Tools: ابزارهای مهندسی نرم‌افزار؛ مانند Sudo، Rational rose، Visio

Methods: چگونگی انجام کارها از نظر فنی بررسی می‌شود.

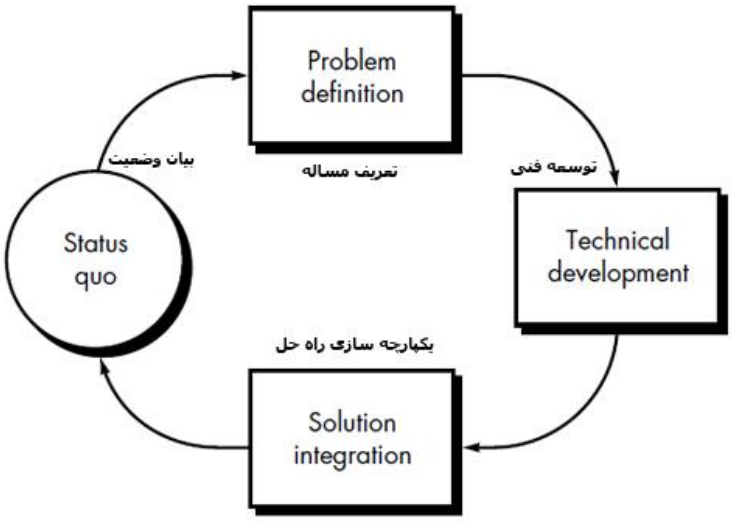
Process: روال یا فرایند تهیه مستندات، گزارش‌ها، مدل‌ها، داده‌ها، فرم‌ها و...

مدل‌های فرایند تولید نرم‌افزار

جهت برطرف کردن بحران نرم‌افزار، الگوها یا مدل‌های مناسبی برای تولید نرم‌افزار ارائه شده است.

چند نکته:

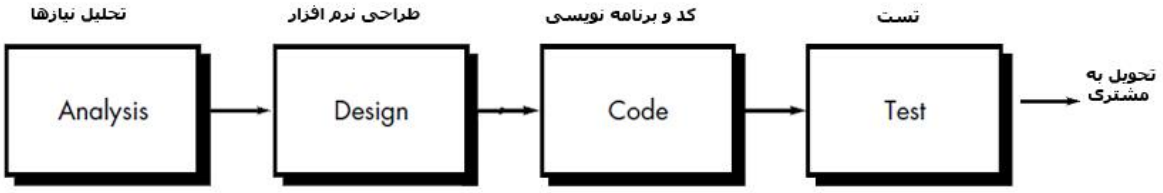
- مدلِ فرایند نرم‌افزار، یک مدل انتزاعی (Abstract) از فرایند است و شرح فرایند را از تعدادی دیدگاه خاص نمایش می‌دهد.
- مدل فرایند برای یک پروژه، بر اساس ماهیت و طبیعت پروژه، کاربران آن، روش‌ها و ابزار مورد استفاده، کنترل‌ها و مستنداتی که لازم است ارائه شود انتخاب می‌شود.
- تمام کارهای توسعه و ارائه یک نرم‌افزار را می‌توان یک حلقه طبق شکل زیر در نظر گرفت:



بررسی چند مدل مهم تولید و توسعه نرم‌افزار

۱. مدل ترتیبی-خطی (Linear-Sequential Model)

ساده‌ترین و کارآمدترین روش برای تولید نرم‌افزارهای نسبتاً محدود می‌باشد که در آن نرم‌افزارها، تحلیل نیازمندی‌ها و تعریف صورت مسأله به سادگی انجام می‌شود.



- **مزایای مدل ترتیبی- خطی**

۱. در این مدل فقط نیاز به تخصص برنامه‌نویس وجود دارد، یعنی از نظر نیروی متخصص مشکلات کمی وجود دارد.

۲. چون برنامه‌نویس، همه کاره است مشکلی در یافتن و به کار بردن روش‌های مختلف و تبدیل این مراحل به هم وجود ندارد.

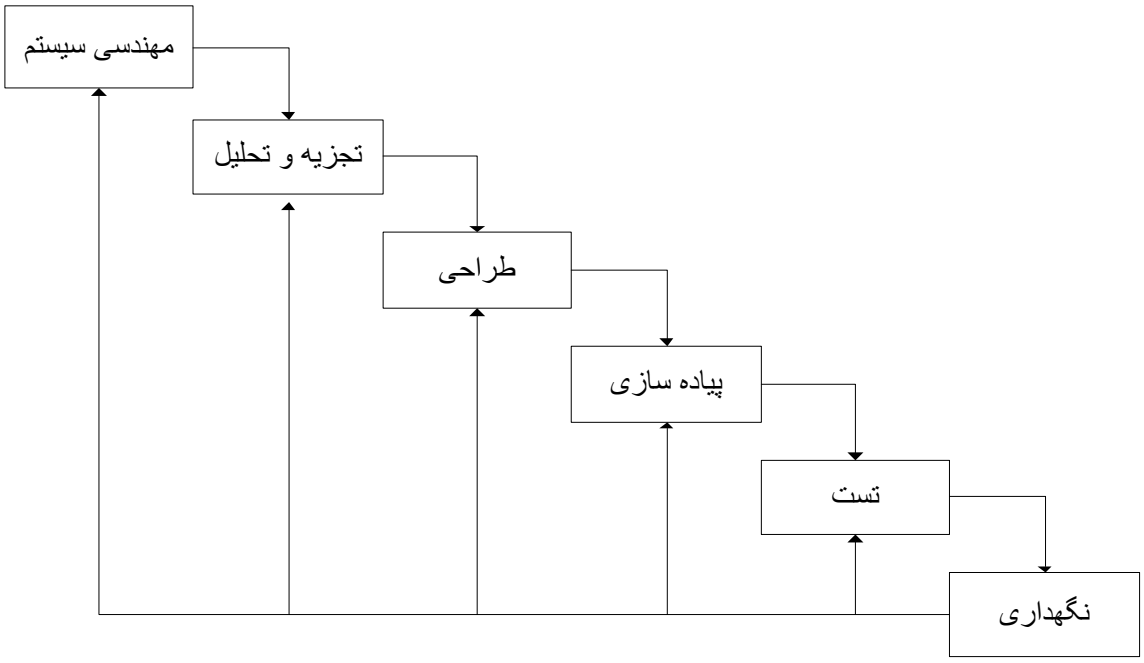
- **معایب مدل ترتیبی- خطی**

۱. سیستمِ تولید شده متکی به فرد است (یعنی برنامه‌نویس).
۲. کد تولید شده در این مدل چون با تغییرات زیادی همراه بوده دارای ساختار مناسبی نیست و تغییرات بعدی روی آن به راحتی صورت نمی‌پذیرد.
۳. معمولاً توجه لازم و کافی به نیازهای کاربر نمی‌شود و نرم‌افزار در اکثر مواقع دارای مشخصات لازم نیست.
۴. قابلیت اطمینان در نرم‌افزار تولیدی به این روش وجود ندارد چون نرم‌افزار دائماً در حال ترمیم می‌باشد و بر اساس طراحی صحیحی صورت نگرفته است.
۵. در این مدل همه چیز در برنامه‌نویس خلاصه می‌شود.
۶. پیگیری نرم‌افزار تولیدی و نگهداری آن با مشکلات اساسی روبه‌رو است.

۲. مدل آبشاری (Waterfall Model)

- یک مدل سنتی و متداول است.
- به آن مدل Life cycle (چرخه حیات) نیز گفته می‌شود.
- در این مدل، تولید نرم‌افزار مراحل مختلفی دارد که هر مرحله دارای ورودی و خروجی خاصی است و خروجی هر مرحله در این مدل ورودی مرحله بعد است.
- ساده‌ترین و قدیمی‌ترین مدل است.
- چون در این مدل، مدت‌زمانِ اجرای پروژه، طولانی می‌شود، در پروژه‌ها و نرم‌افزارهای وسیع کمتر استفاده می‌شود.

- **شکل کلی مدل آبشاری**



- **مزایای مدل آبشاری**

۱. چون خروجی هر مرحله، ورودیِ مرحله‌ی بعد است، اگر اولین مرحله به خوبی انجام شده باشد -یعنی مهندسی نیازها از ابتدا به خوبی انجام شده باشد- مراحل بعدی نیز به خوبی انجام خواهد شد.
۲. مراحل این مدل بسیار شبیه مراحلِ کلیِ مهندسی نرم‌افزار است که در همه‌ی مدل‌های مهندسی نرم‌افزار کاربرد دارند.

- **معایب مدل آبشاری**

۱. با این که در این مدل، مرحله‌ی اول بسیار مهم است ولی عملاً توجه لازم به این مرحله نمی‌شود و در نتیجه کل پروژه دچار اختلال می‌شود.
۲. در اکثر مواقع، در شروعِ کار، بیان صریح همه نیازمندی‌ها برای مشتری مشکل است؛ در نتیجه ممکن است هر لحظه یک نیاز جدید مطرح شود و مدام برگشت به عقب داشته باشیم.

نکته مهم: اگر در طی مراحل تولید، نیازها در حال تغییر باشد بهتر است از این مدل استفاده نشود.

۳. مشتری باید صبور باشد؛ چون نسخه عملیاتیِ نرم‌افزار، غالباً تا اواخر پروژه در دست وی قرار نمی‌گیرد.
۴. یک اشتباه بزرگ که تا اجرای برنامه تشخیص داده نشده باشد می‌تواند فاجعه تولید کند.
۵. برگشت به عقب به هر دلیلی ممکن است اتفاق بیفتد و این برگشت هزینه را بالا می‌برد.
۶. از آنجا که خروجی هر مرحله، ورودی مرحله بعد است برخی اعضای تیم باید منتظر بمانند تا کار مراحلِ قبل تمام شود.

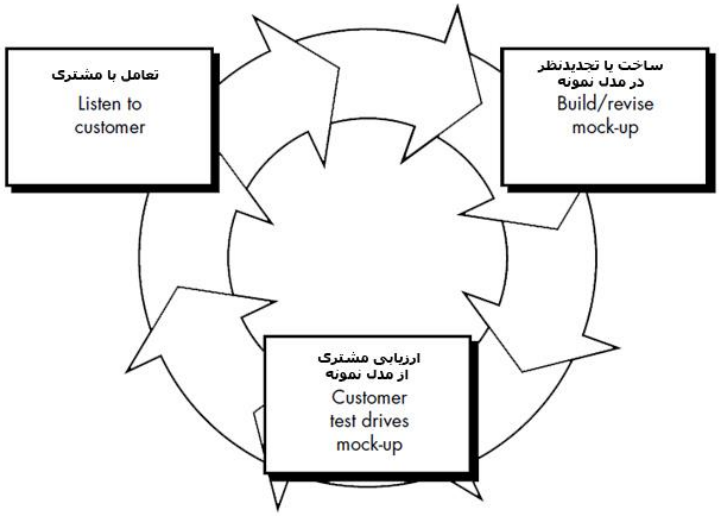
۳. مدل نمونه‌سازی (Prototype Model)

با توجه به معایبِ روش آبشاری و اینکه معمولاً مشتری یک سری اهداف کلی را برای نرم‌افزار تعریف می‌کند و جزئیاتِ پردازشِ ورودی و خروجی را مشخص نمی‌کند و به تعبیری نیازهای مشتری مشخص نیست، برای رفع این مشکل، «مدل نمونه‌سازیِ سریع» در مهندسی مطرح شد و به عنوان روشی مناسب مورد توجه عده زیادی از متخصصان کامپیوتر قرار گرفت.

در این مدل، تولید کننده یا همان طراح نرم‌افزار، مدلی از نرم‌افزار -هر چند مختصر و مفید و شاید حتی روی کاغذ- تولید می‌کند به طوری که کاربر ارتباط بین خود و کامپیوتر را احساس کند و متوجه عملکرد نرم‌افزار شود. سپس کم‌کم با شناسایی نیازها و اجزای لازم، پروژه تکمیل می‌شود. به طور مثال یک سری فرم‌ها، پنجره‌ها و صفحات تولید می‌شود که شکل و شمایل نرم‌افزار را تشکیل می‌دهد.

نکته: می‌توان برای سرعت بخشیدن به تولیدِ نمونه، از محیط‌های ویژوال استفاده کرد؛ مانند Access یا Visual Studio و...

- **شکل کلی مدل نمونه‌سازی**



- **مزایای مدل نمونه‌سازی**

۱. در مواردی که نیازهای مشتری واضح نیست و در درخواست‌های کاربر، ابهام وجود دارد این مدل بسیار مناسب است.
۲. در طولِ طراحیِ نرم‌افزار، ارتباطِ مشتری با طراح و تولیدکننده‌ی نرم‌افزار، همواره برقرار است.

- **معایب مدل نمونه‌سازی**

۱. مشتری یک نسخه ابتدایی از نرم‌افزار می‌بیند و نمی‌داند که این نمونه بدون در نظر گرفتن کیفیتِ کلیِ نرم‌افزار یا قابلیت نگهداری دراز مدت، ساخته شده اگر به مشتری گفته شود که قرار است یک نسخه‌ی دیگر ساخته شود ممکن است نتواند با این موضوع کنار بیایید.
۲. توقف این سیکل مشکل است. مشتری دوست دارد دائماً یک سری ویژگی‌ها اضافه شود.
۳. مشتری فکر می‌کند تولید نرم‌افزار به سادگی همین ماکت است؛ بنابراین در زمان تحویل نرم‌افزار و دریافت هزینه‌ی انجامِ پروژه از مشتری، ممکن است با مشکل مواجه شوید.
۴. ممکن است مشتری وقتِ لازم برای تعامل و ارزیابیِ ماکت را اختصاص ندهد یا مشتری به سادگی در دسترس نباشد.
۵. تولید کننده معمولاً برای اینکه نمونه را سریع به پایان برساند به خیلی از مسائل توجه نمی‌کند؛ به طور مثال ممکن است یک سیستم‌عامل نامناسب یا زبان برنامه‌نویسیِ نامناسب فقط به این دلیل که مشهور هستند انتخاب شوند و بعد از مدتی تولید کننده به این انتخاب‌ها عادت کند و دلایل نامناسب بودنِ آن‌ها را فراموش کند.

۴. مدل RAD[1] (مدل توسعه سریع برنامه‌ی کاربردی)

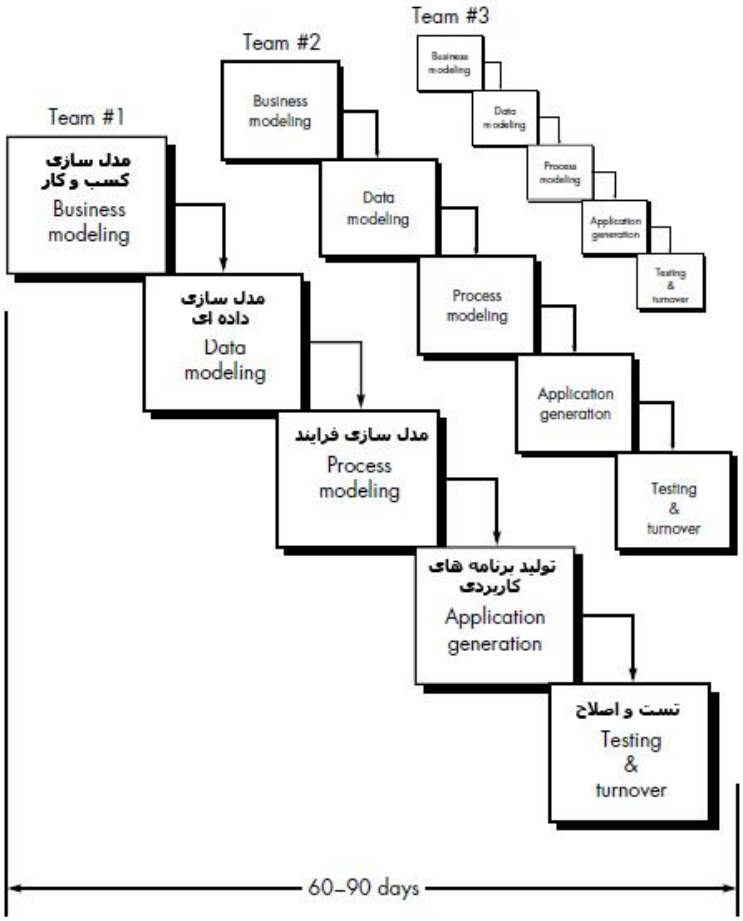
- این مدل بر کار تیمی و استفاده از componentهای از پیش ساخته‌شده تأکید دارد.
- شکلِ پر سرعتِ مدل ترتیبی-خطی است که در آن با استفاده از ایجادِ مبتنی بر مؤلفه‌ها، توسعه‌ی سریع صورت می‌گیرد.
- اگر خواسته‌ها به خوبی درک شده باشند و دامنه پروژه محدود باشد، مدل RAD این امکان را به تیم تولید کننده می‌دهد که سیستم کاملاً عملیاتی را در مدت زمان بسیار کوتاه (مثلاً بین ۶۰ تا۹۰ روز) ایجاد کنند.

پس دو شرط استفاده از این مدل عبارتند از:

۱- نیازهای سیستم (نرم‌افزار مطلوب) کاملاً مشخص باشد.

۲- باید بتوان سیستم را به زیر سیستم‌های مستقل از هم تقسیم کرد.

- **شکل کلی مدل** RAD



[1] Rapid Application Development

- **مزیت مدل RAD:**

مزیت بزرگ مدل RAD این است که می‌توان بین ۶۰ تا ۹۰ روز سیستم نسبتاً بزرگی را تولید کرد. مزیت دیگر این است که احتمالاً اشتباه یک تیم بر روی کار تیم دیگر خلل ایجاد نمی‌کند.

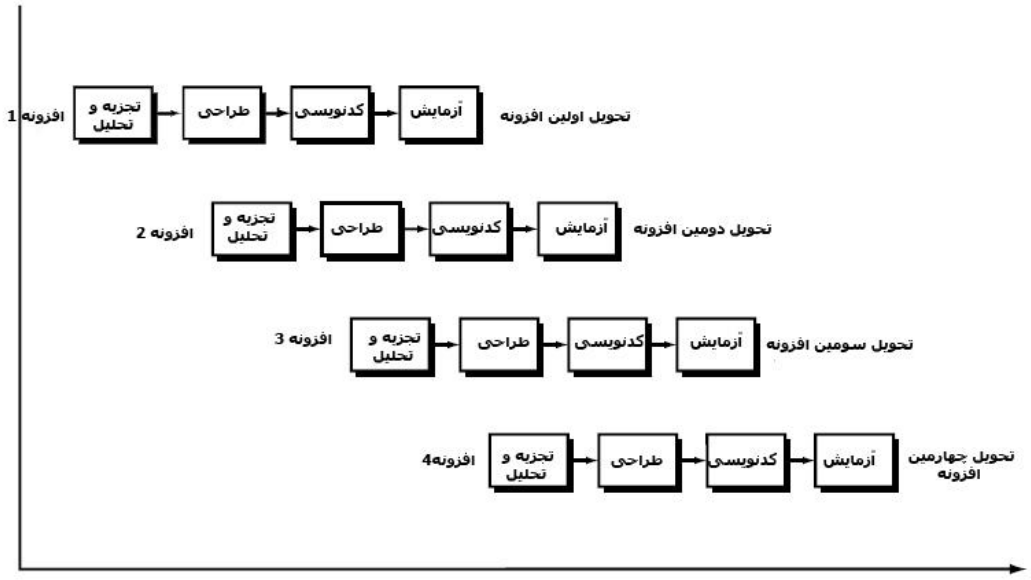
- **معایب مدل RAD:**

۱. نیاز به نیروی انسانی زیاد.
۲. اگر سیستم را نتوان به زیر سیستم‌های مستقل تقسیم کرد امکان استفاده از مدل RAD وجود ندارد.
۳. مدل RAD نیازمند تولیدکننده و مشتری‌هایی است که بتواند برای رسیدن به یک سیستم کامل فعالیت‌های لازم را در یک چارچوب زمانی کوتاه با سرعت زیاد انجام دهند. اگر هر یک از طرفین همکاری نکنند و متحد نباشند پروژه شکست خواهد خورد.
۴. مدل RAD برای سیستم‌هایی که کارایی بالا از آن‌ها انتظار می‌رود مناسب نیست.

۵. مدل‌سازی فرایند نرم‌افزارِ تکاملی[۲] یا مدل افزایشی[۳]

این مدل، مدل ترتیبی-خطی را با روال کلی مدل نمونه‌سازی ترکیب می‌کند.

- **شکل کلی مدل تکاملی:**



- معمولاً خروجیِ اولین افزونه، هسته سیستم (System Core) می‌باشد که نیازهای اصلی مشتری را در برمی‌گیرد و طی تکرارهای بعدی کامل می‌شود؛ بنابراین یک مدل تکاملی است.
- به جای تحویل کل سیستم، روال توسعه و تحلیل به تعدادی افزونه (Increment) شکسته می‌شود و طی هر افزونه بخشی از سیستم با عملکرد لازم تحویل مشتری داده می‌شود.

---

[2] Evolutionary Software Process Modeling

[3] Incremental Model

- نیازهای کاربر اولویت‌بندی می‌شود و نیازهای با اولویت بالاتر زودتر در اولین افزونه‌ها در نظر گرفته می‌شود.

فرق مدل افزایش با مدل نمونه‌سازی

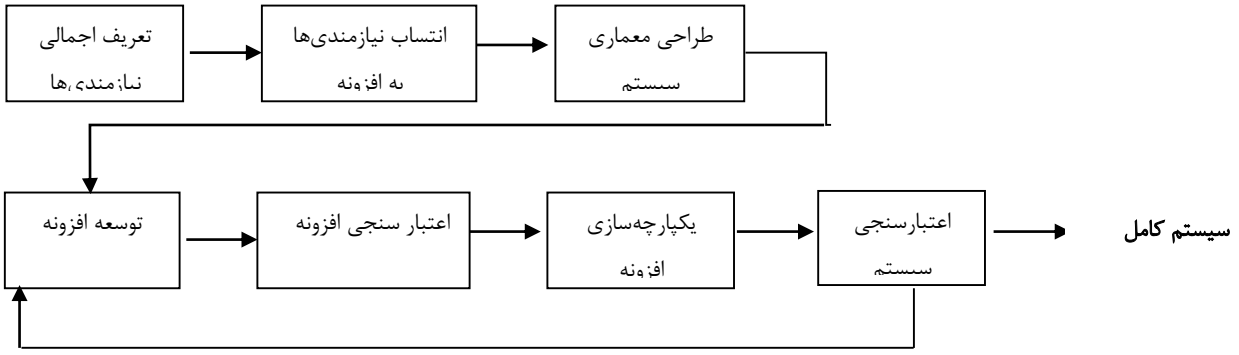
هر چند این مدل مانند مدل نمونه‌سازی ماهیت تکرارشونده دارد اما بر خلاف مدل نمونه‌سازی در طی هر تکرار روی تحلیل یک نسخه عملیاتی به مشتری تأکید می‌شود.

- **مزایای مدل افزایشی**

۱. نتایج و باز خوردِ[۴] افزونه‌های قبلی می‌تواند در افزونه‌های بعدی تأثیر خوبی داشته باشد.
۲. تحویلِ زود هنگامِ برخی از اجزای مفیدِ سیستم، گردشِ پول را بهبود می‌بخشد و بخشی از سرمایه زودتر برمی‌گردد.
۳. کنترل و مدیریتِ زیر-پروژه‌های کوچک‌تر، ساده‌تر و بهتر است.
۴. احتمال تغییر در قوانین و نیازمندی‌ها به دلیل فاصله‌ی کوتاه بین طراحی و تحلیلِ هر یک از افزونه‌ها، به اندازه پروژه‌های بزرگ و یکپارچه نیست.
۵. کاهش ظاهرسازی: به این معنی که درخواست ویژگی‌های غیر ضروری، که در واقع استفاده نیز نمی‌شوند از طرف مشتری کاهش می‌یابد.

- **چند نکته در مورد مدل افزایشی**

- به طور ایده‌آل، مدتِ هر افزونه، کمتر از یک ماه باشد و در نهایت، بیش از سه ماه طول نکشد.
- هر افزونه منافعی برای مشتری داشته باشد و بخشی از عملکرد سیستم (System Functionality) را بپوشاند.
- برای تعیین اولویت افزونه‌ها از نسبت ارزش به هزینه (Value-to-Cost) می‌توان استفاده کرد.
- بهتر است برای جلوگیری از تکه‌تکه شدن کل سیستم و برای یکپارچه نمودنِ هرچه بهترِ افزونه‌ها، ابتدا معماری کل سیستم را طبق شکل زیر به دست آورید:



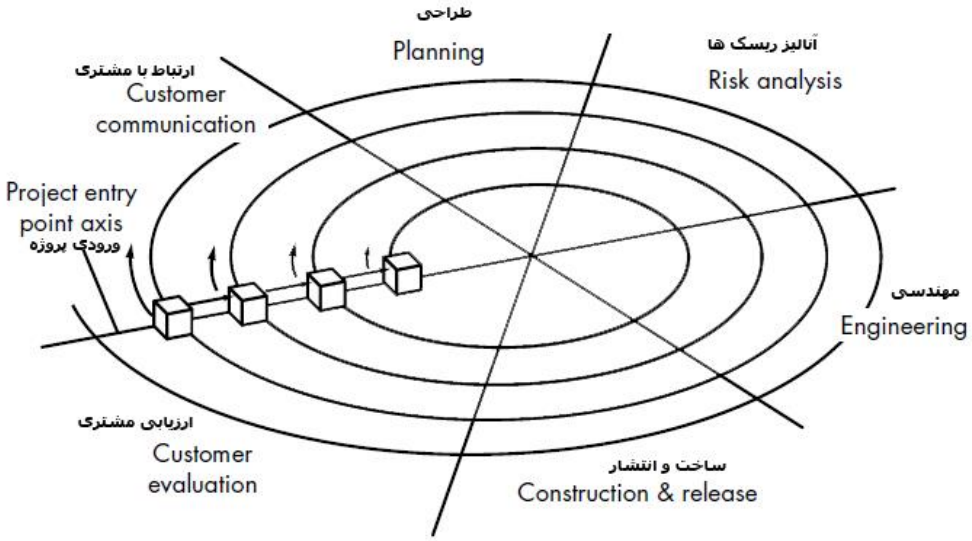
[4] feed back

۶. مدل حلزونی

مدل حلزونی که در اصل توسط آقای Boehm پیشنهاد شد، یک مدل فرایند تکاملی است که ماهیت تکراری مثل نمونه‌سازی را با مفاهیم سیستماتیک و کنترل شده مدل ترتیبی-خطی تلفیق می‌کند. این مدل بر اساس نمونه‌سازی[۵] بنا شده و فرض می‌کند که سیستم را بتوان به زیر سیستم‌های مستقل تقسیم نمود.

روش کار: برای تولید نرم‌افزار با این مدل، ابتدا هسته سیستم را با استفاده از نمونه‌سازی پیاده‌سازی می‌کنند و سپس موارد ریسک یا خطر را بررسی کرده و با استفاده از تجربه‌ی قسمت قبل، نمونه‌ی جدید را برای سیستم تولید می‌کنند که ویژگی‌های جدیدی به آن افزوده شده است و این عمل تا تکمیل سیستم ادامه پیدا می‌کند.

- **شکل کلی مدل حلزونی**



- **خطر یا Risk در فرایند تولید نرم‌افزار:**
  - سیستم نیازهای کاربر را فراهم نسازد.
  - هزینه‌ی به کار گرفته شده بیش از پیش‌بینی باشد.
  - زمانِ به‌کار گرفته شده بیش از پیش‌بینی باشد.
  - تناقض در نیازهای کاربر وجود داشته باشد.
  - استفاده از فناوری‌های جدید
  - مقاومت کاربران

[5] Prototyping

۷. مدل تکنیک‌های نسل چهارم (4GT)

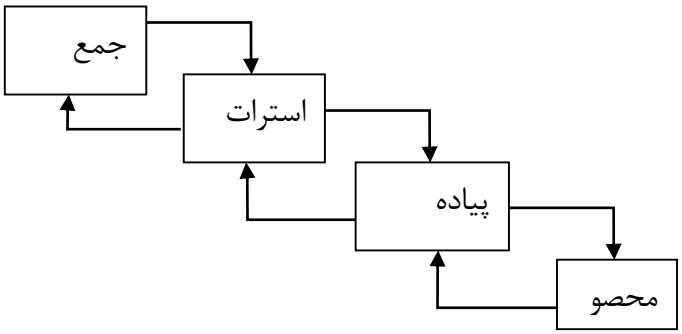
مفهوم تکنیک‌های نسل چهارم، ابزارهای نرم‌افزاریِ بسیاری را شامل می‌شود که همه آن‌ها یک چیز مشترک دارند: همه به مهندس نرم‌افزار این امکان را می‌دهند که یک سری ویژگی‌های نرم‌افزار را در سطح بالا
(High level) مشخص کند و در نهایت توسط این ابزارها به صورت اتوماتیک کدهای مربوطه را ایجاد کند.

در این ابزارها ویژگی‌های زیر جهت تسریع و تسهیلِ کار فراهم است:

- تولید گزارش‌ها
- تولید فرم‌های ورودی
- تولید کد
- قابلیت‌های گرافیکی
- زبان‌های غیر رویه‌ای برای پرس و جو از پایگاه داده‌ها

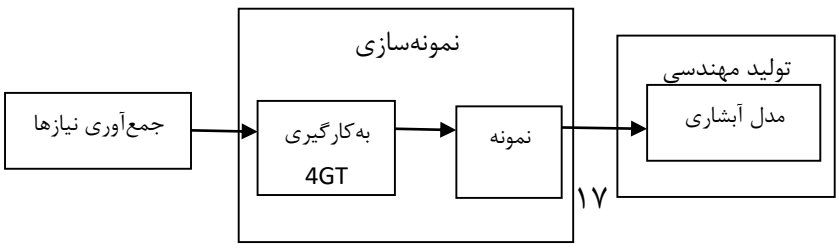
به طور کلی تحقیقات نشان می‌دهد که استفاده از 4GT برای سیستم‌های کوچک و متوسط زمان تولید نرم‌افزار را کاهش می‌دهد ولی در سیستم‌های بزرگ، میزانِ این کاهش به بزرگی بار اول نیست، ولی چون تولید کد به صورت خودکار است، در کل، زمان تولید کاهش می‌یابد.

ترتیب عملیات در مدل 4GT به صورت زیر است:



۸. ترکیب مدل‌های فرایند

ممکن است بسته به شرایطِ پروژه، مدل‌های مختلف با هم ترکیب شود؛ به طور مثال:



مراحل مدل‌سازی پروژه

۱. مدل‌سازی کسب و کار

در این مرحله به سؤالات زیر پاسخ داده می‌شود:

- کدام اطلاعات فرایند کسب و کار را به پیش می‌برد؟
- چه اطلاعاتی تولید می‌شود؟
- چه کسی آن‌ها را تولید می‌کند؟
- اطلاعات کجا می‌روند؟
- چه کسی آن‌ها را پردازش می‌کند؟
- ....

۲. مدل‌سازی داده‌ای

در این مرحله، اطلاعات به صورت مجموعه‌ای از اشیا (objects) در نظر گرفته می‌شود، ویژگی‌ها یا صفات هر شیئ (attributes) مشخص شده و رابطه میان اشیا معلوم می‌گردد.

۳. مدل‌سازی فرایند

در این مرحله، نحوه تغییر objectها و جریان اطلاعات مدل‌سازی می‌شود. فرایندهای لازم (process)، ورودی و خروجی هر فرایند (منظور، توابع است) و شرح هر فرایند مشخص می‌شود.

۴. تولید برنامه‌ی کاربردی

در این مرحله با استفاده از تکنیک‌های نسل ۴ یا 4GT[۶] به جای تولید نرم‌افزار با زبان‌های برنامه‌نویسی، از componentهای موجود یا مؤلفه‌های قابل استفاده مجدد استفاده می‌شود.

در هر حالت از ابزارهای ایجاد خودکار برای تسهیل در کار تولید نرم‌افزار استفاده می‌شود.

نکته: از آنجا که مدل RAD به استفاده مجدد بر مؤلفه‌های موجود تأکید می‌کند و بسیاری از مؤلفه‌ها قبلاً آزمایش شده است این موضوع باعث کاهش آزمایش کل سیستم می‌شود، اما مؤلفه‌های جدید باید آزمایش شود و ارتباط بین اجزا یا مؤلفه‌ها باید آزمایش شود.

[6] fourth generation techniqes

تحلیل‌گر سیستم اطلاعاتی (System Analyst)

در کار با کامپیوتر دو گروه کاربر وجود دارد:

۱. Business User: نیاز به، به‌کارگیری کامپیوتر دارد.
۲. Technicians/Programmers: متخصص‌اند و نحوه کار با کامپیوتر را می‌دانند.

تحلیل‌گر وظیفه ارتباط بین این دو را دارد.

تعریف تحلیل‌گر

تحلیل‌گرِ سیستم، شخصی‌ست که مسائل، مشکلات و نیازهای یک سازمان را مطالعه می‌کند تا تعیین کند چگونه انسان‌ها، روش‌ها و تکنولوژیِ کامپیوتر می‌تواند امور کاریِ آن سازمان را بهبود بخشد و کارایی را بالا برد. بنابراین تعیین مشخصاتِ نیازمندی‌های سیستمِ جاری و ارزیابیِ سایر راه‌حل‌ها برای بر طرف کردن مشکلاتِ موجود، از وظایف تحلیل‌گر سیستم است.

وظیفه برنامه‌نویس (Programmer)

برنامه‌نویس تنها وظیفه نوشتن برنامه و پیاده‌سازی الگوریتمی را دارد که به وی داده شده است. در کل با زبان‌های برنامه‌نویسی و سیستم‌های عامل و سایر برنامه‌های کمکی[۷] سر و کار دارد و ارتباط‌های انسانیِ وی اندک است. (حداکثر، با سایر برنامه‌نویسان و تحلیل‌گران)

وظایف تحلیل‌گر سیستم

کار تحلیل‌گرِ سیستم، فراتر از برنامه‌نویس است.

۱. او مسؤول انتخاب سخت‌افزار و نرم‌افزار است. (Platform)
۲. انتخاب (و شناسایی) افرادی که از سیستم استفاده خواهند کرد. (End-user)
۳. تعیین ساختار فایل‌ها و بانک اطلاعاتی
۴. کار او همواره دقیق نیست و برخی انتخاب‌ها واقعاً مشکل است.
۵. کار او روابط انسانی قوی می‌طلبد باید بتواند با کاربران از طیف‌های مختلف ارتباط برقرار کند؛ مانند مدیر شرکت، برنامه‌نویسان، کاربران عملیاتی و ... پس، از نظر ارتباط کلامی و نوشتاری باید قوی باشد. **نکته**: تحلیل‌گر در عین حال که یک متخصصِ کامپیوتر است باید یک مدیر خوب نیز باشد.

شرایط تحلیل‌گر

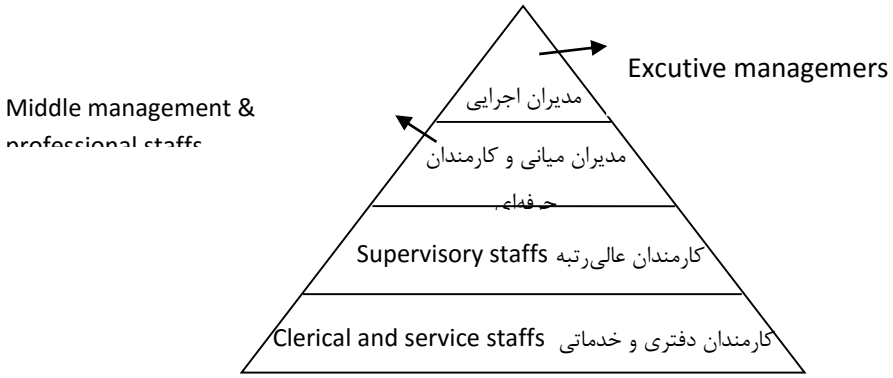
۱- تجربه و مهارت در برنامه‌نویسی

[7] Utility

۲- توانایی حل مسأله (یعنی: تعریف مسأله، ارائه چندین راه حل برای مسأله و انتخاب بهترین راه حل)
۳- توانایی به کارگیری componentهای سخت‌افزاری و نرم‌افزاری
۴- توانایی فهم محیط مشتری
۵- توانایی برقراری ارتباط خوب کتبی و شفاهی با کاربر
۶- تجربه کار تیمی
۷- توانایی تجزیه و تحلیل سیستم‌ها به صورت رسمی و آکادمیک
۸- تجربه کاری
۹- دانش کسب و کار عمومی

رده‌های کاربران

علت شناسایی رده کاربران اولاً درک و برآورده کردن نیازهای آن‌ها و در نظر گرفتن نیازهای آن‌ها در سیستم و ثانیاً تهیه مستندات و راهنمایی‌های لازم برای هر رده است.



تشریح رده‌ها:

- کارمندان دفتری و خدماتی: کسانی هستند که امور روزمره‌ی سیستم را انجام می‌دهند مانند: پر کردن فرم، تایپ نامه‌ها، پاسخگویی به مراجعان و یا انجام بعضی از کارهای دستی و دفتری مانند: منشی‌ها، کتابدارها، کارکنان اداری
- کارمندان عالی رتبه: پایین‌ترین رده مدیریتی در سازمان است. در حقیقت مدیرِ چند نفر هستند؛ مانند: مدیر خط تولید یک کالا، مدیر واحد ورود اطلاعات
- مدیران میانی: این افراد در برنامه‌ریزی‌های کوتاه مدت سازمان تصمیم‌گیری می‌کنند و کنترل‌های لازم را انجام می‌دهند و خود را با کارهای روزمره درگیر نمی‌کنند؛ مانند: مهندسان، مدیران فروش، حسابداران و ...

- مدیران اجرایی: این افراد مسؤول برنامه‌ریزی‌ها و تصمیم‌گیری[8]های دراز مدت سازمان هستند به علاوه، مسؤول تخصیص منابع لازم برای انجام امور سازمان.

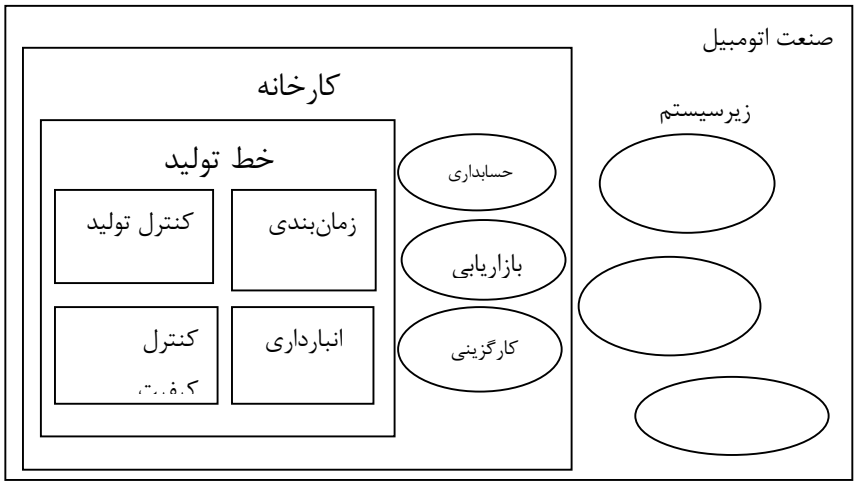
چند نکته کلیدی ویژه تحلیل‌گران

- تحلیل‌گر همیشه باید این چند نکته را در نظر داشته باشد:

۱. "سیستم، مالِ کاربران است". عدم توجه به این موضوعِ مهم ممکن است گران تمام شود.

۲. حرف، حرفِ مشتری و ارباب رجوع است نه حرف من.

- تحلیل‌گر باید برای غلبه بر پیچیدگیِ امور، سیستم را به چند زیر سیستم تقسیم کند:



- برای تشخیص نیازهای کاربران، تحلیل‌گر باید خود، Business را بشناسد. یک Business بر حسب Purposeها، Goal ها، Objectiveها و Policyها شناسایی می‌شود.

Purpose: Purpose یک شرکت، علتِ وجودِ آن شرکت است. تحلیل‌گر باید از Planها(برنامه‌ها)ی شرکت آگاه باشد و معین کند آن برنامه‌ها تا چه حد روی سیستم اطلاعاتی تأثیر خواهند داشت.

Goal: اهدافِ آنی و کلیِ شرکت است؛ مثلاً آیا شرکت می‌خواهد در آینده، میزان سهامش را افزایش یا کاهش دهد یا خیر؟

Objective: اهدافِ خاص برای رسیدن به Goalها هستند؛ مثلاً برای افزایش سهام، نیاز به چه مواردی است؟

Policy: خط مشی‌ها و سیاست‌ها یا قوانینی که برای رسیدن به Objectiveها وضع و پیروی می‌شوند. تحلیل‌گر باید از این قوانین آگاه بوده و آن‌ها را در IS اعمال کند.

[8] Planning

اصولی برای توسعه یک سیستم موفق

1. The system is for the end-users

سعی شود که به نیازهای کاربران توجه کافی شود و آن‌ها را جهت همکاری با خود ترغیب کنیم. آن‌ها را مطمئن سازیم که هیچ نگرانی از سیستم جدید وجود ندارد؛ یعنی آن‌ها نباید سیستم را تهدیدی برای خود بدانند.

2. Establish phases & tasks

چون اغلب پروژه‌ها بزرگ هستند مدیریت و کنترلِ پیشرفت، کار مشکل می‌شود، لذا بهتر است هر پروژه‌ی نرم‌افزاری را به فازها یا مراحل، و هر فاز را به وظایفی تقسیم کنیم تا بر مشکلات غلبه کنیم.

3. System development is not a sequential process

فرایندِ توسعه، یک فرایندِ متوالی نیست، یعنی ممکن است بعضی از فازها هم‌زمان انجام شوند و بعضی کارها در این حین تغییر کنند؛ یعنی فرایند توسعه یک فرایند چرخشی است. گاهی وقت‌ها اگر تغییرات در مرحله‌ی جاری، حیاتی نیستند می‌توان آن‌ها را به عقب انداخت، در غیر این صورت قبل از ادامه‌ی کار، باید تغییرات انجام شود.

4. Systems are capital investment

برای رسیدن به سرمایه، باید راه‌حل‌های مختلف را برای برآوردنِ نیازهای سیستم پیدا کنیم و سپس بهترینِ آن‌ها که مقرون به صرفه‌تر (cost-effective) است را انتخاب کنیم.

5. Don’t be afraid to cancel

باید در هر فازِ توسعه، مقرون به صرفه بودن پروژه را بررسی کنیم و اگر نبودنِ پروژه، مقرون‌به‌صرفه‌تر است، بهتر است از ادامه پروژه منصرف شویم.

6. Documentation is a product of all phases

مستندات را باید در کلیه فازهای سیستم مکتوب کنیم.

انگیزه‌های آغاز پروژه‌های نرم‌افزاری چیست؟

محرک و عامل اصلی همه پروژه‌ها سه چیز است:

۱. مسائل و مشکلات (Problems):

مشکلاتی که سازمان را از فعالیت‌های کامل و کارا باز می‌دارد؛ مثل امنیت یا یک فرم سفارش دستی که وقت زیادی را می‌گیرد و یا ثبت نام دستی.

۲. فرصت‌ها (Opportunities):

شانس‌هایی هستند که با اینکه یک Problem یا مشکل وجود ندارد، اما موجب آغاز یک پروژه می‌شوند؛ مثل ایجاد سیستم کامپیوتری برای کاهش هزینه در یک سازمان علی رغم اینکه مشکل مالی وجود ندارد..

۳. جهت‌ها یا دستورات (Directives):

عواملی هستند که توسط مدیر یا دولت یا سایر سازمان‌ها جهت آغاز پروژه تحمیل می‌شوند؛ برای مثال درخواست لیست کامپیوتری کارکنان از طرف امور استخدامی کشور، کارت سوخت، تغییر شناسنامه‌ها، کارت ملی و ...

برخی افراد مثل آقای James Wetherbe معتقدند انگیزه‌های آغاز پروژه عبارتند از:

۱. نیاز به بهبود کارآیی (Performance)
۲. نیاز به بهبود یا کنترلِ داده یا اطلاعات
۳. نیاز به بهبود اقتصادی یا کنترل هزینه‌ها (Economics)
۴. نیاز به بهبود کنترل و امنیت (Control & Security)
۵. نیاز به بهبود راندمان انسانی و ماشینی
۶. نیاز به بهبود خدمات به مشتریان، شرکا و... (Services)

System Development Life Cycle (SDLC)
چرخه حیات توسعه سیستم

چرخه حیات توسعه سیستم شامل چهار مرحله است:

۱. تحلیل
۲. طراحی
۳. پیاده‌سازی (ساخت)
۴. پشتیبانی

اما برای شرح بهتر این فازها آن‌ها را به چند قسمت زیر تقسیم می‌کنیم:

1. Survey project scope and feasibility

یک پروژه از نظر امکان‌سنجی و تعیین محدوده و دامنه، باید بررسی شود. در این فاز مسائل، فرصت‌ها و محدودیت‌های تکنیکی (constraints) و محدودیت‌های کاری به همراه رده‌بندیِ سطوحِ مختلفِ کاربران بررسی می‌شود و نتیجه کار تحت عنوان «گزارش امکان‌سنجی» (Feasibility Assessment) به مدیریت پروژه یا سازمان ابلاغ می‌شود.

2. Study and analyze the current system

در این فاز، سیستمِ جاری را به تفصیل مورد مطالعه قرار می‌دهیم. سیستم جاری ممکن است دستی یا کامپیوتری باشد. تحلیل‌گر در این فاز، مسائل و طرز کار سیستم را بررسی کرده و آن را یاد می‌گیرد و نتیجه بررسی را به صورت گزارشی با نام «گزارش مسائل و مشکلات» (Problem Assessment) به کمیته تحویل می‌دهد. این فاز در حقیقت تکمیلِ فاز امکان سنجی است.

3. Define the end-users requirements

این فاز را گاهی Definition phase (فاز تعریف) نیز می‌گویند. در این فاز، ورودی‌ها، فایل‌ها، پردازش‌ها و خروجی‌های مورد نیازِ سیستمِ جدید تعریف می‌شود. نتیجه‌ی این فاز، گزارشی به نام «گزارش نیازمندی‌ها» (requirements assessments) است.

4. Select a feasible solution from candidate solutions

انتخاب یک راه حلِ شدنی از بین راه حل‌های کاندید.

چندین راه حل به کمیته‌ی پروژه پیشنهاد می‌شود این راه حل‌ها باید بر اساس موارد زیر باشد:

- Technical feasibility (امکان‌سنجی تکنیکی)
- Operational feasibility (امکان‌سنجی عملیاتی)
- Economic feasibility (امکان‌سنجی اقتصادی)

نتیجه این فاز، تحت گزارشی به نام «طرح سیستم» (System Proposal) به مدیریت پروژه ارجاع داده می‌شود.

5. Acquire computer hardware & software

در این مرحله، امکان خرید نرم‌افزارها و سخت‌افزارهای لازم برای ساخت سیستم جدید را بررسی می‌کنیم. شاید به این نتیجه برسیم که سیستم جدید به طور از پیش‌ساخته‌شده و آماده خریداری شود.

6. Design the new system

وقتی به این مرحله رسیدیم نیازمندی‌های سیستم را در فازهای قبلی مشخص کرده‌ایم، پس حالا نوبت طراحی سیستم جدید است.

7. Construct the new system

فاز ساختِ سیستم، سخت‌ترین، وقت‌گیرترین و خسته‌کننده‌ترین فاز توسعه سیستم است. در این فاز تقریباً تمام کارها بر عهده برنامه‌نویس است و تحلیل‌گر فقط جهت روشن کردن برخی از مشخصه‌ها با برنامه‌نویس همکاری می‌کند. ضمناً در این فاز سیستم ساخته شده باید تست کامل نیز شود.

8. Deliver the new system

در این فاز، سیستمِ ساخته‌شده در محیط عملیاتی نصب و راه‌اندازی می‌شود. آموزش‌های لازم و کتابچه‌های راهنما به کاربران داده می‌شود، در خاتمه، تحلیل‌گر کاربران را جهت گذر از سیستمِ قبلی به سیستمِ جدید یاری خواهد داد.

9. Maintain and improve the system

پس از تحویل و به کارگیری سیستم، ممکن است نیاز به تغییرات یا اصلاحاتی باشد که باید آن‌ها را رفع اشکال نمود یا با افزایش ویژگی‌های جدید، نیازمندی‌ها را برآورده کرد.

کشف و استخراج نیازمندی‌ها یا شناخت وضعیت موجود

طبق تعریف آقای Boehm اگر خطایی در قسمت نیازمندی‌ها رخ دهد، هزینه ترمیم آن در فازهای مختلفِ توسعه‌ی نرم‌افزار به شرح جدول زیر است:

فازی که خطا یافت شده	نسبت هزینه
تحلیل نیازمندی‌ها	۱
طراحی	۳-۶
پیاده‌سازی	۱۰
تست توسعه	۱۵-۴۰
تست پذیرش	۳۰-۷۰
نگهداری	۴۰-۱۰۰۰

استخراج نیازمندی‌ها شامل مراحل زیر است:

۱- پی بردن یا استخراج مسأله و تحلیل آن
۲- استخراج نیازمندی‌های مسأله
۳- مستندسازی و تحلیل نیازمندی‌ها
۴- مدیریت نیازمندی‌ها

تکنیک‌های کشف حقیقت (Fact-Finding Techniques):

۱. نمونه‌برداری و مطالعه مستندات موجود، فرم‌ها و فایل‌ها
۲. استفاده از تجربیات قلمروهای مشابه؛ مثال: بررسی سیستم انتخاب واحد در دانشگاه‌های دیگر
۳. مشاهده محیط کاری
۴. پرسشنامه
۵. مصاحبه
۶. نمونه‌سازی کشفیات

۱. نمونه‌برداری

- بدون برخورد مستقیم با کاربران و کسب و کار، می‌توان با مطالعه مستندات به حقایق بسیاری دست یافت.
- چارت سازمانی و بررسی هر یک از بخش‌های آن حقایق بسیاری را آشکار می‌کند.
- اگر مطالعه‌ی همه‌ی مستندات ممکن نیست با استفاده از تکنیک‌های نمونه‌برداری، زیرمجموعه‌ای از اسناد را انتخاب و مطالعه می‌کنیم.

۲. استفاده از تجربیات قلمروهای مشابه

- تیم‌های مختلفی بر روی سیستم مورد نظر کار کردند با بررسی آن‌ها حقایق بسیاری به دست خواهد آمد.

۳. مشاهده محیط کاری

- مشاهده یکی از مؤثرترین تکنیک‌ها برای گردآوری اطلاعات و درک سیستم می‌باشد.
- تحلیل‌گر از نزدیک شاهد گردش کار در محیط خواهد بود.

- **مزایای مشاهده**
  - اطلاعات جمع‌آوری شده از طریق مشاهده قابل اطمینان‌اند.
  - هزینه‌ی به کارگیریِ این تکنیک نسبت به تکنیک‌های دیگر نسبتاً پایین است.
  - تحلیل‌گر می‌تواند خودش روال کاری را جهت درکِ بهتر، طی کند.
  - شرح کارهای پیچیده، معمولاً مشکل است اما تحلیل‌گر می‌تواند از طریق مشاهده، درکِ بهتری نسبت به آن‌ها داشته باشد.

- **معایب مشاهده**
  - ممکن است فرد در زمانی که مورد توجه قرار می‌گیرد تغییر رفتار دهد.
  - ممکن است تحلیل‌گر با برخی از روال‌های کاری در حین مشاهده سیستم برخورد نکند.
  - کار مشاهده، حجم کار یا سطح پیچیدگی و جزئیاتِ کار را چندان نشان نمی‌دهد.

۴. پرسشنامه

- در این تکنیک سؤالاتی توسط تحلیل‌گر طرح شده و بین کاربران محیط عملیاتی توزیع می‌شود.
- معمولاً وقتی از پرسشنامه استفاده می‌شود که تعداد افراد زیاد باشد و امکان مصاحبه حضوری وجود نداشته باشد.

○ انواع پرسشنامه

- Free-format (از فرمت خاصی پیروی نمی‌کند)
- Fixed-format
  - **انواع پرسشنامه با فرمت ثابت:**
    - چهار گزینه‌ای
    - امتیازدهی (Rating)
    - تعیین رتبه (Ranking)

○ مزایای پرسشنامه

- اکثر پرسشنامه‌ها خیلی سریع جواب داده می‌شوند.
- روش نسبتاً کم‌هزینه‌ای جهت جمع‌آوری اطلاعات از افرادِ زیاد است.
- اکثر افراد نسبت به مصاحبه با پرسش‌نامه راحت‌تر هستند.
- در اکثر پرسشنامه‌ها، هویت فرد مشخص نمی‌شود و خیلی راحت‌تر جواب خواهد داد. (حقایق را بیان خواهد کرد)
- پاسخ‌ها می‌توانند جدول‌بندی شوند و سریعاً تحلیل شوند.
- نسبت به مصاحبه، وقتِ بیشتری برای پاسخ دادن دارند.

○ معایب پرسشنامه

- هیچ ضمانتی وجود ندارد که همه افراد، به سؤالات، پاسخ دهند یا حتی درست پاسخ دهند.
- امکان مشاهده و تحلیل حرکات و رفتار پاسخ‌دهنده وجود ندارد.
- تهیه پرسشنامه‌های خوب معمولاً مشکل است.
- پرسشنامه‌ها قابلیت انعطاف ندارند یعنی مثلاً تحلیل‌گر نمی‌تواند برای تفسیرِ یک پاسخ، سؤال مجدد بپرسد.

○ روال تهیه یک پرسشنامه خوب

اگر طراحیِ خوبی در مورد پرسشنامه صورت نگیرد، شانس موفقیت‌تان کم می‌شود.

برای تهیه یک پرسشنامه خوب روال زیر را طی کنید:

۱- دقیقاً تعیین کنید چه حقایق و نظراتی از چه کسانی باید جمع‌آوری شود. اگر تعداد افرادِ جوابگو، زیاد است، به طور تصادفی از زیرمجموعه‌ی کوچک‌تری از آن‌ها استفاده کنید.

۲- نسبت به شرایط، یکی از دو نوع پرسشنامه free format یا fixed format را انتخاب کنید. البته پیشنهاد می‌شود پرسشنامه، ترکیبی از این دو حالت باشد.

۳- مواظب باشید سؤالات، شامل خطا نباشد یا اشتباه تفسیر نشود ضمن اینکه سؤالات نباید تعصب یا نظر شخصی شما را القا یا پیشنهاد کند؛ به طور مثال: آیا شما موافقید...؟ یا به نظر شما بهتر نیست...؟
۴- سؤالات را روی نمونه کوچکی از جوابگوها تست کنید، اگر مشکلی در جواب دادن وجود دارد یا جواب‌ها مفید نیستند سؤالات را ویرایش کنید.
۵- نهایتاً پرسشنامه‌ها را به تعداد کافی تهیه و توزیع کنید.

۵. مصاحبه

- مهم‌ترین و پرکاربردترین تکنیک جمع‌آوری اطلاعات به ویژه در سیستم‌ها و محیط‌های عملیاتی کوچک‌تر است.
- در این روش، جمع‌آوریِ اطلاعات به صورت face to face (رو در رو یا چهره به چهره) خواهد بود.

○ اهداف مصاحبه

مصاحبه می‌تواند برای دست‌یابی به زیرمجموعه‌ای از اهداف زیر باشد:

- یافتن حقایق
- وارسی یا تأیید حقایق
- شفاف‌سازی حقایق
- تولید انگیزه و جدیت
- تعیین کاربران نهایی
- شناسایی نیازمندی‌ها و جمع‌آوری ایده‌ها و نظرات

○ نقش‌های درگیر در مصاحبه

۱- تحلیل‌گرِ سیستم، که مصاحبه‌کننده است و وظیفه سازماندهی و هدایت مصاحبه را برعهده دارد.
۲- کاربر یا مالکِ سیستم، که مصاحبه‌شونده است.

توجه: در طی مصاحبه، داشتنِ روابط عمومیِ بسیار بالا، مورد نیاز است، چون فردِ مصاحبه‌کننده با افرادی دارای ارزش، اولویت، نظرات، انگیزه و شخصیت متفاوت، برخورد خواهد داشت.

○ مزایای مصاحبه

- امکان گرفتنِ بازخورد[9] از مصاحبه‌شونده
- امکان مشاهده حرکاتِ مصاحبه‌شونده در طیِ مصاحبه (یک تحلیل‌گر خوب، ممکن است اطلاعاتی از طریق حرکات شخصِ مصاحبه‌شونده بدست آورد)
- تحلیل‌گر می‌تواند شرایطی را فراهم کند که مصاحبه‌شونده آزادانه و به راحتی به سؤالات پاسخ دهد.
- امکان تغییر و تطبیق سؤالات با توجه به شخصِ مصاحبه‌شونده وجود دارد.

[9] Feed back

- **معایب مصاحبه**

- مصاحبه، زمان‌بر و پرهزینه است.
- مصاحبه ممکن است به علت موقعیت مصاحبه‌شونده امکان‌پذیر نباشد. (مثال: مصاحبه با رئیس‌جمهور یا رئیس یک سازمان)
- موقعیت مصاحبه، به توانایی تحلیل‌گر در برقراری ارتباط با افراد بستگی دارد.

- **انواع مصاحبه**

- **مصاحبه‌ی فاقدِ ساختار:** این نوع مصاحبه فقط بر اساس هدف کلی یا موضوعی که در ذهن داریم و با یک سری سؤالات خاص پایه‌ریزی می‌شود.

توجه: ممکن است مصاحبه از چارچوبش خارج شود که در این صورت تحلیل‌گر باید مصاحبه‌شونده را دوباره به سمت هدف اصلی یا موضوعِ مورد نظر، هدایت کند.

- **مصاحبه‌ی ساخت‌یافته:** مصاحبه‌کننده، مجموعه‌ی مشخصی از سؤالات را برای پرسیدن از مصاحبه‌شونده در اختیار دارد.

- **عوامل موفقیت مصاحبه**

موفقیت یک مصاحبه شامل به موارد موارد زیر بستگی دارد:

۱. انتخاب افراد مناسب برای مصاحبه
۲. آماده شدن جهت مصاحبه
۳. هدایت مصاحبه
۴. پیگیری مصاحبه

- **چند توصیه در مورد طرح سؤالات مصاحبه**

در طرح سؤالات مصاحبه به توصیه‌های زیر دقت کنید:

- استفاده از کدام سؤال واضح و مختصر
- عدم به کارگیری نظر طراح سؤال (تحلیل‌گر) به عنوان بخشی از سؤال
- اجتناب از سؤالات پیچیده و طولانی
- اجتناب از سؤالات تحلیل آمیز

- **در طی مصاحبه باید قواعد زیر رعایت شود:**

این موارد را باید انجام دهید	این موارد را نباید انجام دهید
مؤدب باشید	ادامه دادن به سؤالات غیر ضروری
به دقت گوش کنید	ابراز نظرِ شخصیِ خودتان در طی مصاحبه
کنترل خود را حفظ کنید	صحبت کردن به جای گوش کردن
کنجکاو باشید	فرضِ اینکه جواب، کامل باشد
کنترل مصاحبه را حفظ کنید	فرضِ اینکه از جواب هیچ چیزی حاصل نشود

صبور باشید	داشتن ذهنیتی درباره موضوع یا مصاحبه‌شونده
جو را طوری تنظیم کنید که مصاحبه‌کننده احساس راحتی کند.	ضبط نوار، مساوی است با گوش‌کردنِ ضعیف

- **راهکارهای زیر می‌تواند برای ارتباط برقرار کردن مناسب باشد:**
  - به جلسه‌ی مصاحبه به دید مثبت نگاه کنید.
  - چهره‌تان شاداب و بشاش باشد.
  - مصاحبه‌شونده احساس کند که شما در حال گوش‌کردن هستید.
  - پرسیدنِ سؤال، نشان می‌دهد که شما در حال گوش‌کردن هستید. ضمن اینکه جهت رفع ابهام سؤال می‌پرسید، این ممکن است منجر به تکمیل شدن جواب شود.
  - هیچ فرضیاتی در ذهن خود لحاظ نکنید. با در نظر گرفتن یک فرضیه ممکن است سریع‌تر بخواهید خودتان جواب سؤال را بدهید و صحبت‌های مصاحبه‌شونده را قطع کنید و در نتیجه اطلاعاتی از دست برود.
  - یادداشت‌برداری کنید. این کار دو هدف دارد:
    ۱. به گوینده می‌گویید که صحبت‌هایش آنقدر مهم است که آن‌ها را یادداشت می‌کنید.
    ۲. نکات اصلی را برای بازنگری و یا برای ملاقات‌های بعدی ثبت کرده‌اید.

- **اهمیت علائم جسمانی در مصاحبه:**

منظور از علائم جسمانی، اطلاعاتِ غیرکلامی است که در طی مصاحبه رد و بدل می‌شود و معمولاً ناآگاهانه است.

پژوهش‌ها نشان می‌دهد که میزانِ اطلاعاتی که از فرد کسب می‌شود بر اساس جدول زیر به تحلیل‌گر منتقل می‌شود.

راه انتقال	میزان اطلاعات کسب شده
از طریق کلام (بر حسب لغات)	%۷
تن و لحن صدا	%۳۸
علائم صورت و بدن	%۵۵

۶. نمونه‌سازی کشفیات (Discovery Prototyping)

- نمونه‌سازی، جهت یافتن حقایق و نیازهای سیستم استفاده می‌شود.
- به عملِ ساختِ یک مدلِ کاری با مقیاس کوچک از نیازمندی‌های کاربران برای کشف یا وارسی و تأیید نیازمندی‌ها، در اصطلاح «نمونه‌سازی کشفیات» گفته می‌شود.

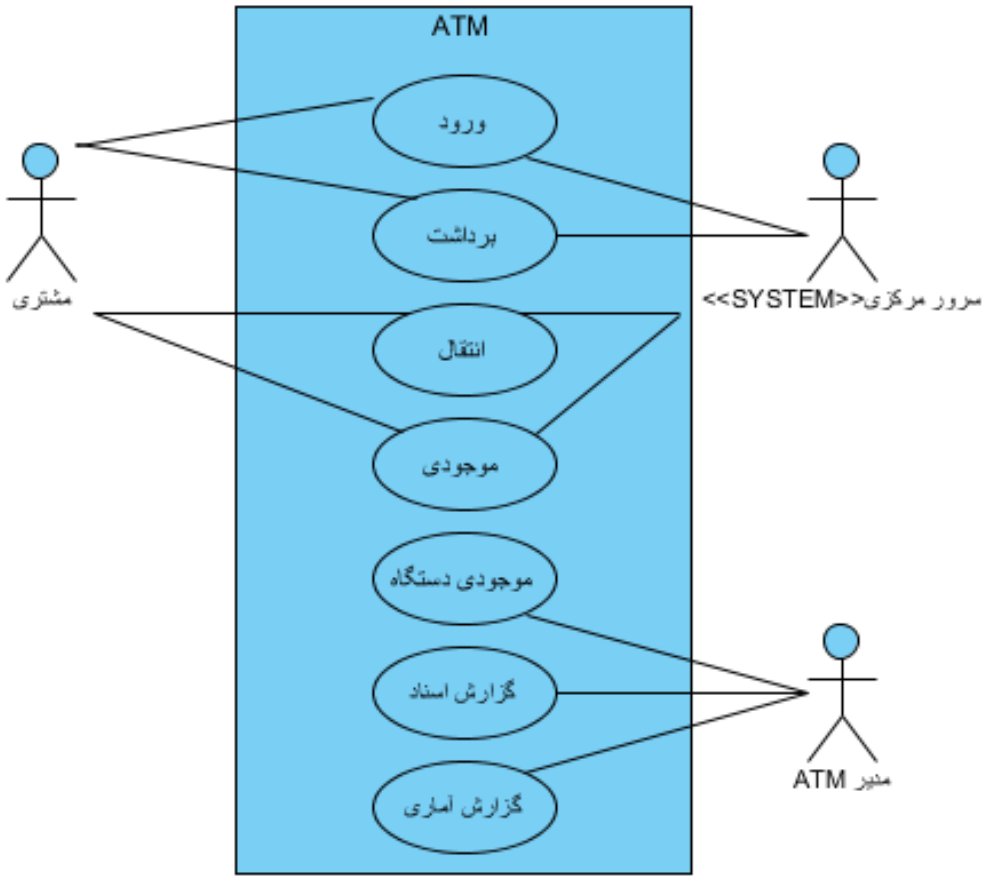
- **تکنیک استخراج موارد کاربری (Use Caseها):**

تعریف Use Case: مجموعه‌ای از اعمال یا سناریوهایی است که سیستم در تعامل با کاربر انجام می‌دهد تا یک نتیجه‌ی ارزشمند برای کاربر فراهم کند.

به کاربر در این مدل «بازیگر» یا «Actor» می‌گویند که می‌تواند یک نقش باشد یا یک فرد. در حقیقت Use Caseها همان سرویس‌ها و خدماتی هستند که سیستم برای کاربرانش فراهم می‌کند. یک تکنیک برای درک بهتر و مستندسازی نیازمندی‌های سیستم، ایجادِ «نمودارِ موارد کاربری» (Use Case Diagram) می‌باشد.

توجه: کاربر خارجی که با Use Case در تعامل است. «بازیگر» یا «عامل» یا «کنش‌گر» یا «Actor» نامیده می‌شود. Actor یک نقش است که می‌تواند یک سیستم خارجی، یک کاربر انسانی و یا یک دستگاه باشد.

مثال: نمودار مورد کاربری برداشت پول از ATM:



- **نحوه شناسایی یا مستندسازی موارد کاربری:**

۱. **شناسایی موارد کاربری یا بازیگران سیستم:** برای این کار بهتر است به دنبال ورودی‌ها و خروجی‌های سیستم و منابعی که آن‌ها را تهیه یا استفاده می‌کنند باشید.

۲. **مستندسازی موارد کاربری سطح بالا:** موارد کاربری سطح بالا را مکتوب می‌کنیم؛ برای این کار به هر مورد کاربری یک نام نسبت دهید (مثل برداشت پول، انتخاب درس، اخراج کارمند، ثبت نام در سایت و ...) و لیست Actorها و توصیف مختصری از آن مورد کاربری را بیان می‌کنیم:

مثال:

- نام کاربری: برداشت وجه و انتقال و ...
- لیست بازیگران: مشتری و سرور مرکزی
- شرح مختصر: کاربر پس از وارد کردن کارت و رمز عبور، صفحه‌ای برای انتخاب وجهِ مورد نظر، مشاهده می‌کند.

۳. **مستندسازی سناریوها با جریان وقایع هر مورد کاربری:** برای هر مورد کاربری، وقایع قدم به قدمِ آن را مشخص و مکتوب می‌کنیم.

برای مستند کردن آن‌ها بهتر است از قالب زیر استفاده کنیم:

۱. نام کاربری
۲. لیست کنشگرها (Actors)
۳. توصیف مختصر [در حد یک پاراگراف]
۴. پیش‌شرط‌ها (حالتی است که قبل از اجرای U.C، سیستم باید در آن حالت یا وضعیت باشد)
۵. توصیفِ جریانِ وقایعِ اصلی (سناریوی اصلی) به صورت قدم به قدم
۶. توصیف جریان فرعی یا سناریوی فرعی (به آن جریان وقایع رقیب یا نظیر نیز گفته می‌شود)
۷. پس‌شرط‌ها (حالتی که پس از اجرای U.C، سیستم باید در آن وضعیت یا حالت باشد)
۸. نیازمندی‌های تکمیلی (سایر فرضیاتی مثل امنیت و کارایی و...)
۹. نقاط توسعه[10]

- شرح تفضیلی[11] یا دقیق مورد کاربری:

۱- نام مورد کاربری: برداشت وجه
۲- لیست Actorها: مشتری، سرور مرکزی
۳- شرح مختصر: پس از تأیید هویت کاربر، مبلغ مورد نظر وی از حساب، کسر شده به وی تحویل داده می‌شود.
۴- پیش شرط‌ها: ورود به سیستم، چک کردن موجودی
۵- سناریوی اصلی:
۱. مشتری روی دکمه برداشت وجه کلیک می‌کند.
۲. سیستم، پنجره تعیین مبلغ را نمایش می‌دهد.
۳. مشتری مبلغ مورد نظر را وارد می‌کند.
۴. سیستم درخواست برداشت را به همراه مشخصات حساب مشتری و مبلغ تعیین شده به سرور مرکزی می‌فرستد و منتظر جواب می‌شود.

[10] Extension Points
[11] Detailed

۵. اگر انجام درخواست مورد قبول بود آنگاه:

۵-۱- سیستم وجه درخواستی را تحویل می‌دهد.

۵-۲- یک رسید برای مشتری چاپ می‌شود.

۶. سیستم به صفحه اختیارات مشتری بر می‌گردد.

۶- سناریوهای فرعی:

a۳: اگر مشتری درخواست کنسل داشت، سیستم به صفحه اختیارات کاربر برگردد.

a۵: اگر موجودی مشتری به حد کافی نباشد:

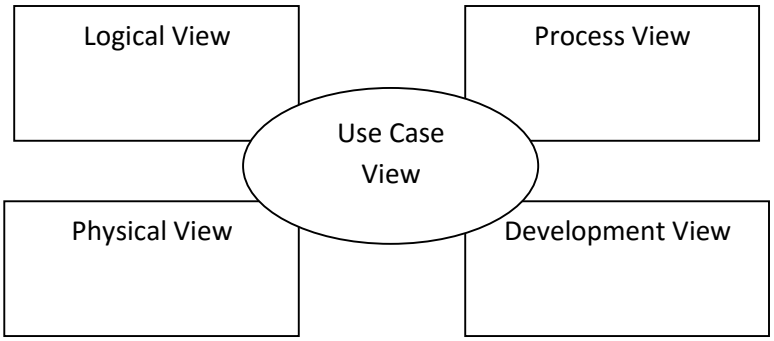
a۵-۱: سیستم پیغام مناسب (موجودی کافی نمی‌باشد) را نمایش دهد.

a۵-۲: برو به گام دوم سناریوی اصلی (نمایش پنجره تعیین مبلغ)

b۵: اگر شبکه قطع باشد سیستم پیغام مناسب (در حال حاضر امکان هیچ پردازشی نیست) را نمایش دهد و کارت مشتری را تحویل دهد.

UML

شخصی به نام Philippe Kruchten در سال ۱۹۹۵ مدلی به نام ۴+۱ ارائه کرد.



طبق این مدل، سیستم از ۵ دیدگاه مختلف مورد بررسی قرار می‌گیرد:

۱- دیدگاه منطقی Logical view
۲- دیدگاه فرایند Process view
۳- دیدگاه فیزیکیPhysical view
۴- دیدگاه توسعه Development view
۵- دیدگاه مورد کاربریUse Case view

۱. دیدگاه منطقی:

این دیدگاه، کل سیستم و روابط بین اجزا را به تصویر می‌کشد. می‌توان گفت این دیدگاه شامل کلاس‌ها و اشیا هستند. دیاگرام‌های UML که برای نمایش دیدگاه منطقی استفاده می‌شود عبارتند از:

1. Class diagram
2. State Diagram
3. Object Diagram
4. Sequence Diagram
5. Communication Diagram

۲. دیدگاه فرایند:

واضح است که این دیدگاه فرایندهای سیستم و هم چنین تمام ارتباطات بین فرایندها را توصیف می‌کند. در واقع مشخص می‌کند که چه چیزی لازم است در درون سیستم اتفاق بیفتد.

تنها دیاگرام UML برای نمایش دیدگاه فرایند عبارت است: Activity Diagram

۳. دیدگاه فیزیکی:

این دیدگاه محیط اجرایی سیستم را واحدسازی می‌کند و منابع نرم‌افزاری را به سخت‌افزاری که آن‌ها را میزبانی می‌کنند map می‌کند. دیاگرام UML برای نمایش دیدگاه فیزیکی Deployment Diagram است.

۴. دیدگاه توسعه:

این دیدگاه ماژول‌ها و مؤلفه‌های سیستم شامل: Packageها، Sub-systemها (زیرسیستم‌ها) و کتابخانه‌های کلاس‌ها را توصیف می‌کند. دیاگرام‌های UML برای نمایش دیدگاه توسعه عبارتند از: Component Diagram و Package Diagram

۵. دیدگاه مورد کاربری:

این دیدگاه عملکرد سیستم را نمایش می‌دهد. در حقیقت این دیدگاه مشخص می‌کند که این سیستم قرار است در مجموع، چه کاری انجام دهد. از این دیدگاه برای مشخص کردن شرح ساختار و عملکرد سیستم در ۴ دیدگاه دیگر استفاده می‌شود دیاگرام UML برای نمایش این دیدگاه Use Case Diagram است.

مدل‌های ایستا و پویا[۱۲]

مدل‌های ایستا خصوصیات ساختاری[۱۳] و مدل‌های پویا خصوصیات رفتاری[۱۴] سیستم را نمایش می‌دهند.

مثال از خصوصیات ساختاری: نوع اشیاء موجود در سیستم و روابط بین آن‌ها

مثال از خصوصیات پویا: رفتار سیستم در قبال کلیک روی یک دکمه

[12] Static Models vs. Dynamic Models

[13] Structural Characteristics

[14] Behavioral Characteristics

دیاگرام‌های UML برای نمایش مدل‌های ایستا عبارتند از:

- Use Case Diagram
- Class Diagram
- Object Diagram

دیاگرام‌های UML برای نمایش مدل پویا عبارتند از:

- Sequence Diagram
- Communication Diagram
- State Diagram
- Activity Diagram

نوع دیگری از مدل‌ها، مدل‌های پیاده‌سازی یا Implementation Models نام دارند که عناصر نرم‌افزاری و سخت‌افزاری لازم برای چیدمان و آرایش نهایی پروژه را مشخص می‌کنند. دیاگرام‌های UML برای نمایش پیاده‌سازی عبارتند از:

- Component Diagram
- Deployment Diagram

دیاگرام مورد کاربری (Use Case Diagram)

تعریف Actor:

Anyone or anything that has a goal in using the system.

هر شخص یا هر چیزی که هدفی از استفاده از سیستم دارد.

تعریف goal:

What the actor wants to achieve by interacting with the system.

آنچه کنشگر یا عامل از طریق تعامل با سیستم می‌خواهد به‌دست آورد.

تعریف Primary Actor (عامل اولیه): چیزی که یک U.C را آغاز می‌کند؛ مثل انسان یا سازمان. مثلاً در سیستم ATM، «مشتری» عامل اولیه است.

ارتباط بین موردهای کاربری:

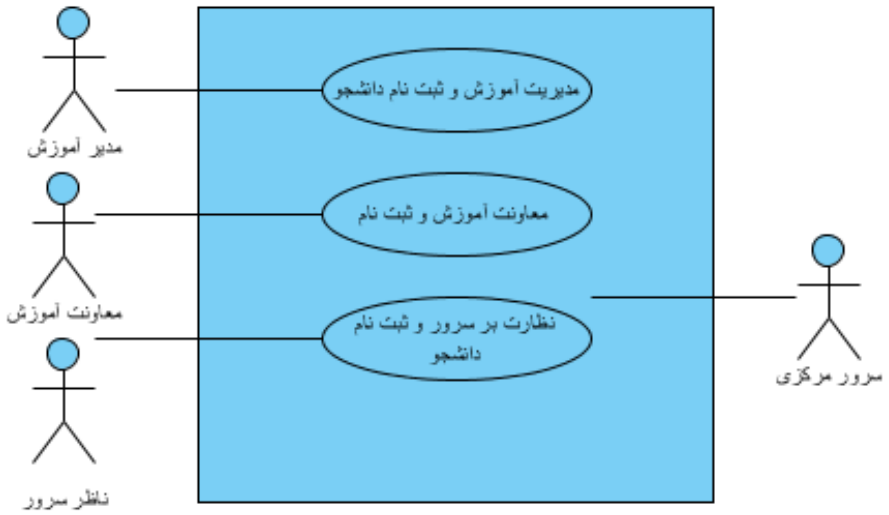
- Include
- Extend
- Generalization

- **Include Dependencies (وابستگیِ «شمول یا شامل شدن»):**

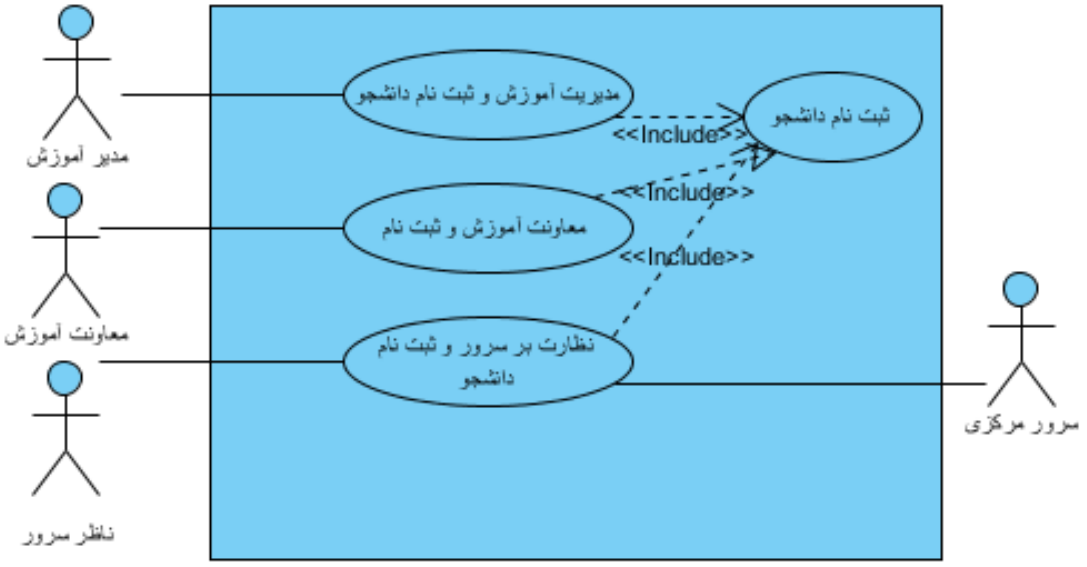
این وابستگی این امکان را می‌دهد که اولاً U.Cهای کلی را به صورت جزئی‌تر بیان کنید، ثانیاً از یکU.C در U.Cهای دیگر استفاده‌ی مجدد کنید.

نحوه نمایش: به صورت خط‌چین و یک پیکان به سمت U.Cی مشترک + عبارت <<include>> روی خط‌چین.

در حالت عادی:



با در نظر گرفتن امکان include:



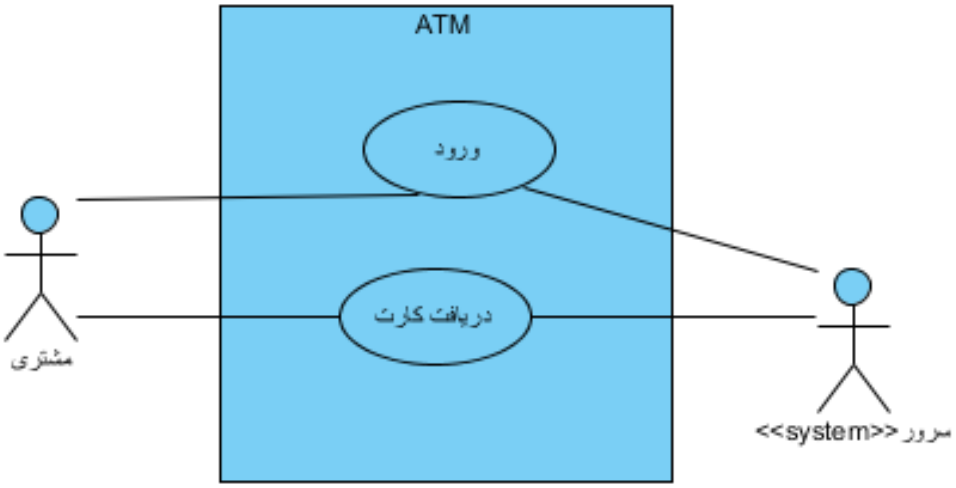
- **Extend Dependencies (وابستگیِ «توسعه»):**

این وابستگی شما را قادر می‌سازد یک مورد کاربریِ جدید به وسیله‌ی اضافه کردن به مراحل مورد کاربریِ جاری، ایجاد نمایید.

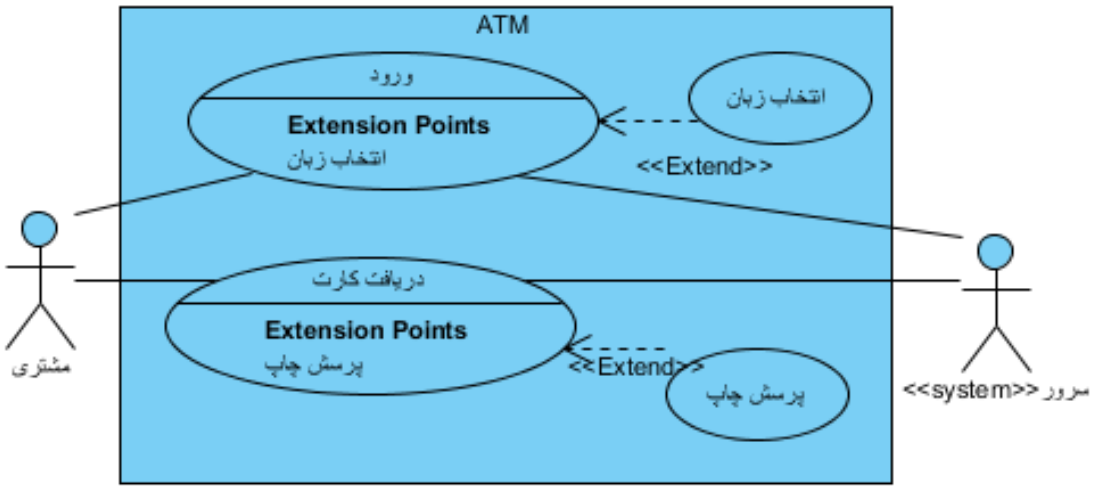
نحوه نمایش: خطچین و یک پیکان به سمت مورد کاربری اولیه + عبارت <<extend>> بر روی خطچین.

نکته: یک U.C را یک actor اجرا می‌کند.

مثال: در حالت عادی:



با استفاده از امکان Extend:



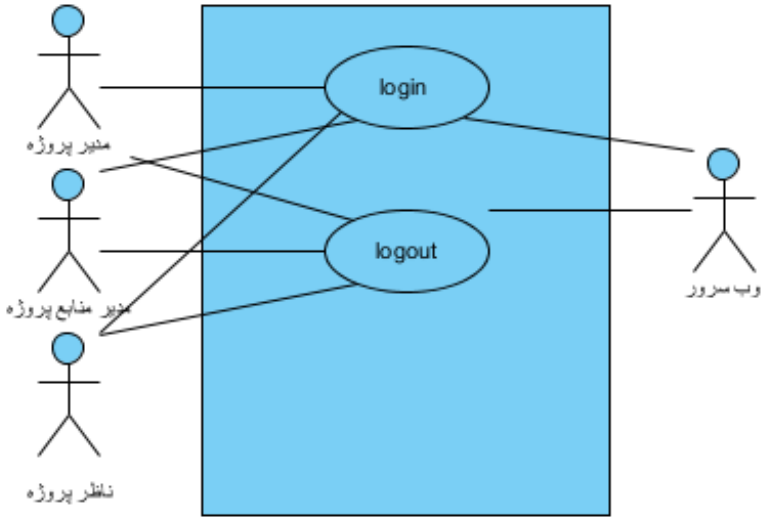
- **Generalization (تعمیم یا عمومیت):**

موارد کاربرد نیز می‌توانند مانند کلاس‌ها در مبحث شیئ‌گرایی، از مواردِ کاربریِ دیگر، به ارث ببرند (همان مفهوم inheritance یا وراثت). در وارثتِ مورد کاربر و مورد کاربر فرزند یا U. C فرزند رفتار و مفهوم را از والد به ارث می‌برند و به رفتار خود اضافه می‌کند شما می‌توانید فرزند را در هر کجا که والد را به کار می‌برید استفاده نمایید.

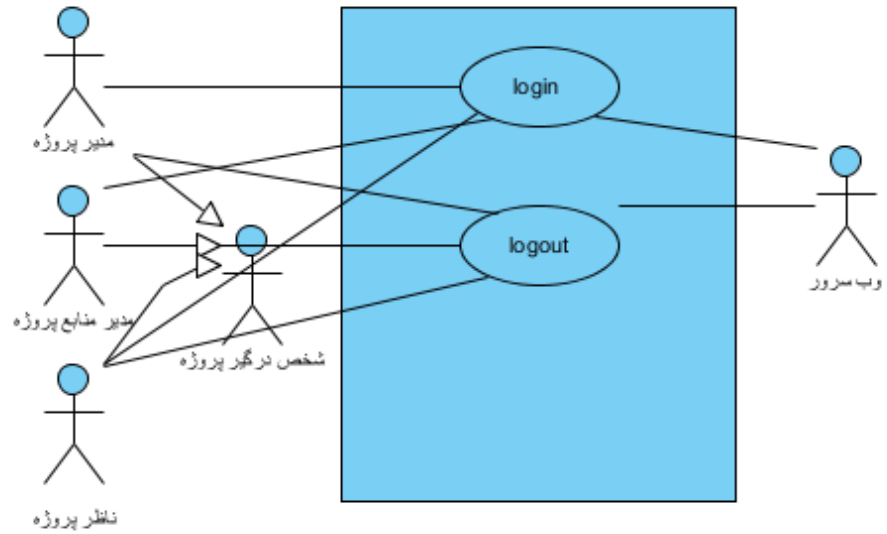
نحوه نمایش: فرزند با یک خط تو پر که به صورت یک مثلث تو خالی به والد متصل شده است.

نکته: ارتباط تعمیم می‌تواند هم در موردU. Cها و هم در مورد actorها وجود دارد.

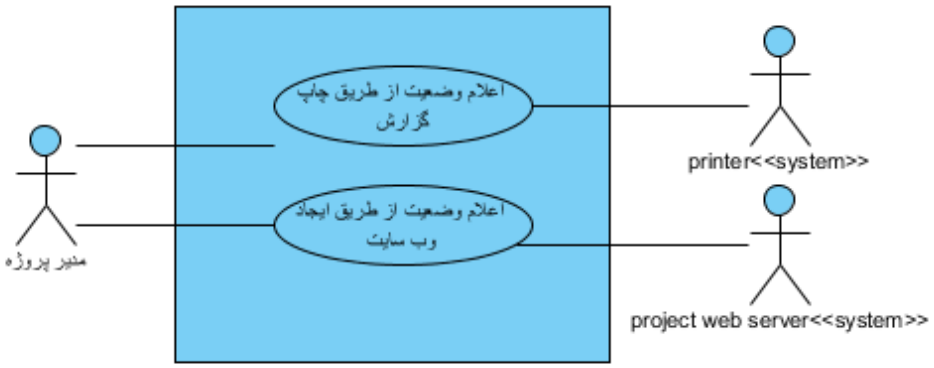
مثال: در حالت عادی:



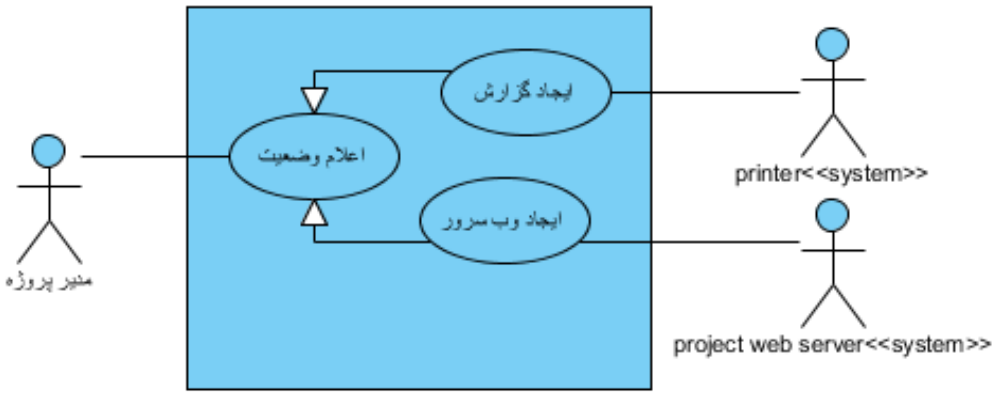
با استفاده از امکان Generalization:



مثال ۲ از Use Case: در حالت عادی:



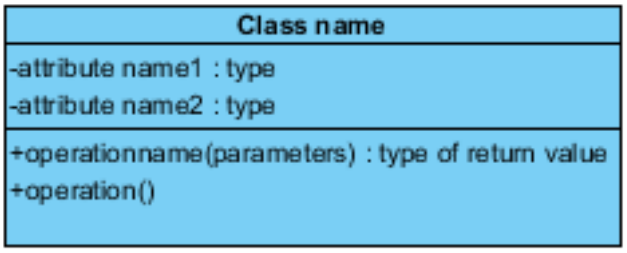
با استفاده از مفهوم تعمیم:



دیاگرام کلاس و شیئ (Class & Object diagrams)

- دیاگرام کلاس مهم‌ترین و عمومی‌ترین دیاگرام در UML است.
- کلاس دو مفهوم یا هدف دارد: دسته‌بندی (category) و قالب (template) دارد.
- دیاگرام کلاس، کلاس‌ها و انواع اشیایی که در سیستم دارید و همین طور روابط بین آن‌ها را نمایش می‌دهد.
- دیاگرام کلاس، یک دیاگرام static به حساب می‌آید؛ یعنی خصوصیت ساختاری سیستم را نمایش می‌دهد.

- **نحوه نمایش** Class:



- **نمایش** Visibility:

کلیک راست روی صفت یا متود و انتخاب یکی از گزینه‌های زیر:

- Public
- Private
- Protected
- Package

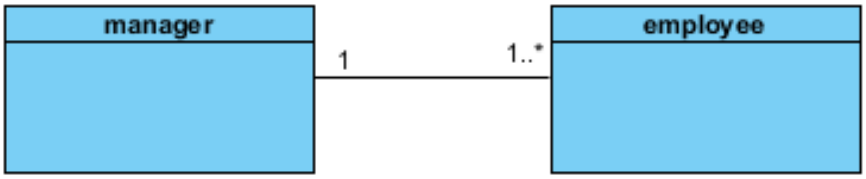
- **نمایش** Associations **(ارتباطات):**

منظور از Associations روابط بین کلاس‌هاست در حقیقت اگر یک ارتباط مفهومی بین کلاس‌ها وجود داشته باشد گفته می‌شود کلاس اول ارتباط دارد با کلاس دوم.

مثال:

مدیر، مدیریت می‌کند کارمندان را

- **نمایش multiplicity (چندگانگی ارتباط):**



برای رابطه الزامیِ یک به یک از علامت 1..x و برای رابطه غیر الزامی یک به یک از علامت 0..x استفاده می‌کنیم.

- **کلاس مجرد[15]:**

افزودن این امکان به زبان است که بتوان قسمتی از یک کلاس را تعریف کرد و تعریف دیگر قسمت‌ها را در کلاس فرزند کامل کرد.

- کلاس‌های مجرد فقط قرار است کلاس‌هایی پایه برای کلاس‌های دیگر باشند نمی‌توان از کلاس‌های مجرد شیئ ایجاد کرد.
- متودها نیز می‌توانند مجرد باشند فقط دقت کنید که متودهای مجرد فقط می‌توانند در یک کلاس مجرد تعریف شوند، نه در یک کلاس معمولی.
- اگر یک کلاس مجرد شامل چند متود مجرد باشد (یعنی لیستی از عملیات که این کلاس پشتیبانی می‌کند) کلاسی که از این کلاس به ارث می‌برد باید تمام متودهای مجرد این کلاس را پیاده‌سازی کند.

- **ساختار کلی تعریف یک کلاس مجرد:**

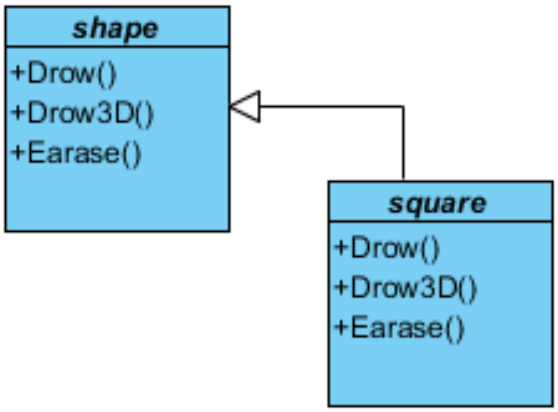
```
[Modifier] abstract class className{

    [Modifier] abstract returnType methodName ([parameters]){
   .
   .
   .
    }

}
```

- **نحوه نمایش کلاس مجرد در UML:**

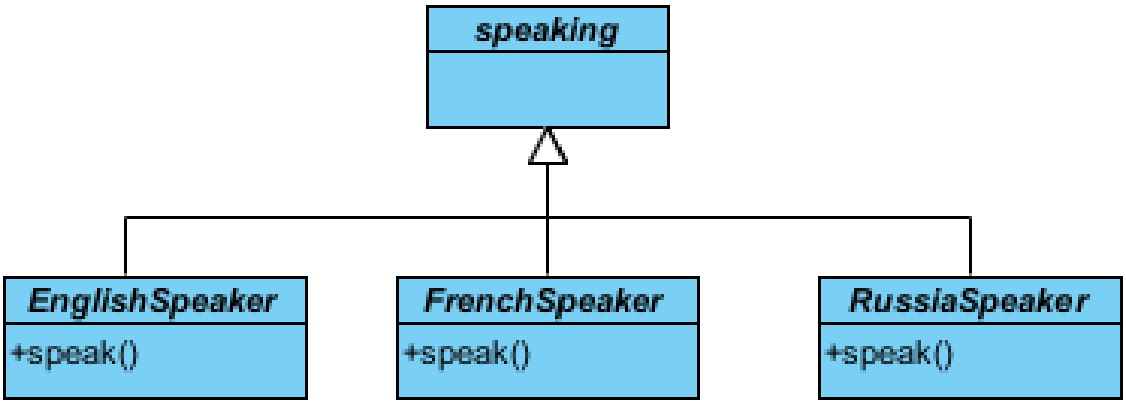
کلاس مجرد نیز مانند یک کلاس ساده نمایش داده می‌شود. با این تفاوت که نام کلاس مجرد به صورت خمیده (Italic) نوشته می‌شود.

مثال:

[15] Abstract class



- معمولاً در کلاس‌های مجرد صفات وجود ندارند.
- اگر یک کلاس از کلاس مجرد به ارث ببرد کلاس فرزند با خط توپر و یک مثلث توپر (در سمت پدر) نمایش داده می‌شود.



■ Interface یا واسط :

تقریباً همان مفهوم کلاس‌های مجرد را دارد با این تفاوت که در یک Interface هرگز پیاده‌سازی نخواهیم داشت یعنی یک یا چندین متود دارد اما بین متودها هیچ کدی را به عنوان تعریف عملکرد متود شاهد نیستیم.

چند نکته)

- واسطها فقط شامل لیستی از اعضای Public است.
- استفاده از Interface مزایای بسیاری دارد که مهم‌ترین مزیت آن دسته بندی عملکردهای مربوط به هر کلاس و تعیین وراثت یک کلاس از واسط مورد نظر است و از قلم نیفتادن پیاده‌سازی عملکردهای (متودهای) مختلف یک کلاس است.
- یک کلاس می‌تواند از چندین واسط به ارث ببرد (فراموش نکرده‌اید که C# یک زبان single inheritance است یعنی به طور مستقیم هر کلاس فقط از یک کلاس به ارث می‌برد).

ساختار کلی تعریف Interface در #C:

```
[Modifier]  interface   Interface name {
        // Methods  and  Structure
}
```

نکته) در ابتدای نام یک Interface معمولاً برای تشخیص اینکه یک واسط است، یکI(حرف بزرگ آی) درج می‌کنند.

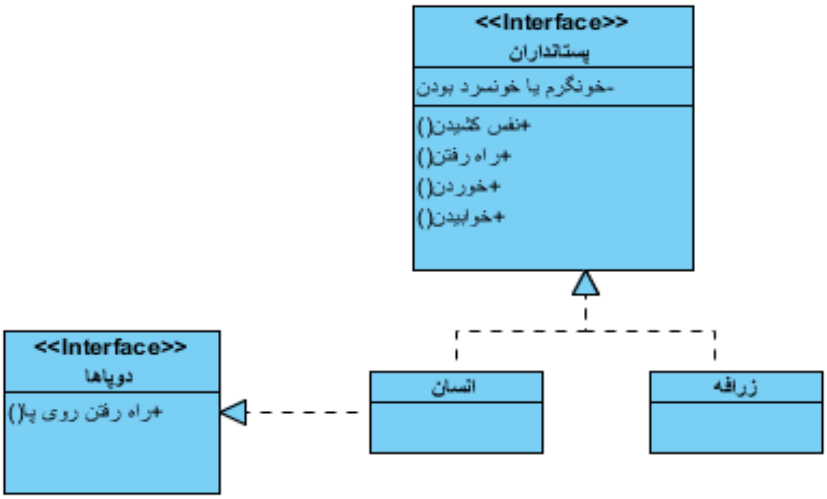
مثال)

```
Public interface Ishape {
        Void Draw();
        Void Draw3D();
}
```

همان‌طور که می‌بینید این واسط شامل دو رفتار یا متود است تصور کنید کلاس‌هایی برای دایره و مربع داشته باشیم درست است که همه این کلاس‌ها دو رفتاری که واسط interface را دارد اما با پیاده‌سازی‌های متفاوت یا Implementationهای متفاوت. پس کلاس‌های دایره و مربع از این واسط به ارث می‌برند اما پیاده‌سازی متودهایشان در خودشان صورت می‌گیرد.

نحوه نمایش Interface در UML:در UML مفهوم Interface زیر مجموعه مفهومی به نام Realization قرار می‌گیرد.

مثال)



چگونه می‌توان کلاس‌ها را از مصاحبه‌های مشتریان استنتاج کرد:

در بحث با مشتریان، مراقب اسامی که آن‌ها برای شرح موجودیت‌ها در کسب و کار خودشان به کار می‌برند باشید آن اسامی در مدل شما "کلاس‌ها" را تشکیل می‌دهند همچنین مراقب افعالی که می‌شنوید باشید زیرا این افعال "عملیات" (operation) آن کلاس‌ها را تشکیل می‌دهند صفات کلاس‌ها در صحبت‌های مشتریان معمولاً در قالب اسامی که مربوط به یک کلاس هستند پدیدار خواهند شد.

مثال) اگر مشتری بگوید:

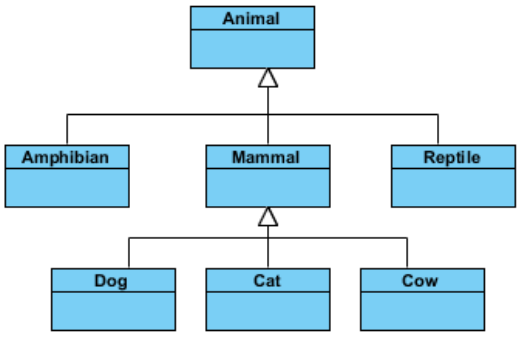
ما در محیطمان یک کارمند داریم که باید اسناد را ثبت کند، این کارمند باید حتماً متأهل باشد.

می‌توان برداشت کرد کارمند و اسناد کلاس بوده، ثبت کردن یک عملیات است و متأهل بودن یا نبودن صفت کامل است.

■ مفهوم وراثت و تعمیم: Generalization and Inheritance

یک کلاس (کلاس فرزند یا زیر کلاس) [sub-class or child class] می‌تواند صفات و عملیات را از دیگری (کلاس والد یا کلاس برتر) [parent class or super class] به ارث ببرد کلاس والد عمومی‌تر است و بعد از آن کلاس فرزند قرار می‌گیرد در عمومیت دادن هرکجا که والد ظاهر شود فرزند هم قادر است باشد دقت کنید که عکس آن درست نیست.

مثال) اگر بدانید که برخی چیزها حیوان هستند انتظار دارید که بخورد، بخوابد، راهی برای متولد شدن داشته باشد... اما تصور کنید که پستانداران، دوزیستان و خزندگان همه حیوان هستند همچنین گاوها، سگ‌ها، گربه‌ها در طبقه پستانداران دسته بندی می‌شوند، شیء گرایی این مورد را وراثت می‌نامد و به صورت زیر در UML نمایش می‌دهد:



نحوه نمایش وراثت در UML: با یک خط که والد را به کلاس فرزند متصل می‌کند و یک مثلث توخالی در سمت والد است.

نحوه یافتن وراثت در صحبت‌های مشتریان:

تحلیل‌گر باید در یابد که چه صفات و عملیاتی از یک کلاس عمومی هستند و شاید برای کلاس‌های متعدد دیگری که صفات و عملیات خود را دارند نیز به کار می‌روند.

✓ تفاوت کلاس مجرد و واسط: Abstract VS Interface

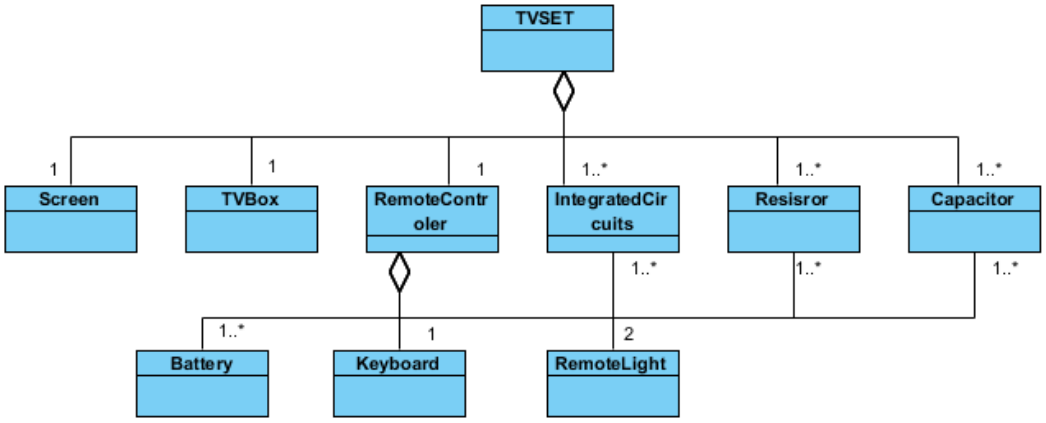
هر دو به هم شبیه‌اند با این تفاوت که تمام امکانات و featureهای Interface از نوع Abstract هستند یعنی پیاده‌سازی نشده اند و باید در فرزند پیاده‌سازی شوند اما کلاس‌های Abstract می‌توانند برخی از عملیات را در خود پیاده‌سازی کنند و پیاده‌سازی برخی دیگر را (که شاید برای فرزندهای مختلف به صورت‌های مختلف پیاده‌سازی شود را به فرزند واگذار کنند).

■ تجمع Aggregation:

گاهی یک کلاس شامل تعدادی کلاس دیگر است که از ترکیب آن‌ها کلاس اول تشکیل می‌شود این مورد نوع ویژه‌ای از ارتباط است که تجمع نامیده می‌شود.

نحوه نمایش تجمع در UML: به صورت یک خط که کلاس کل را به جزء متصل می‌کند با یک لوزی توخالی در سمت کلاس کل.

مثال)

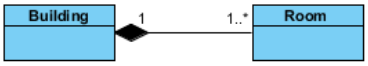


■ ترکیب Composition:

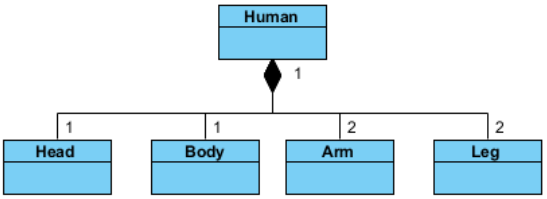
یک Composition نوع قوی از تجمع است هر جزء در Composition می‌تواند فقط به یک کل تعلق داشته باشد.

نحوه نمایش Composition در UML: مانند تجمع در UML نمایش داده می‌شود با این تفاوت که لوزی آن توپر نمایش داده می‌شود.

مثال ۱) یک اتاق فقط متعلق به یک ساختمان است.

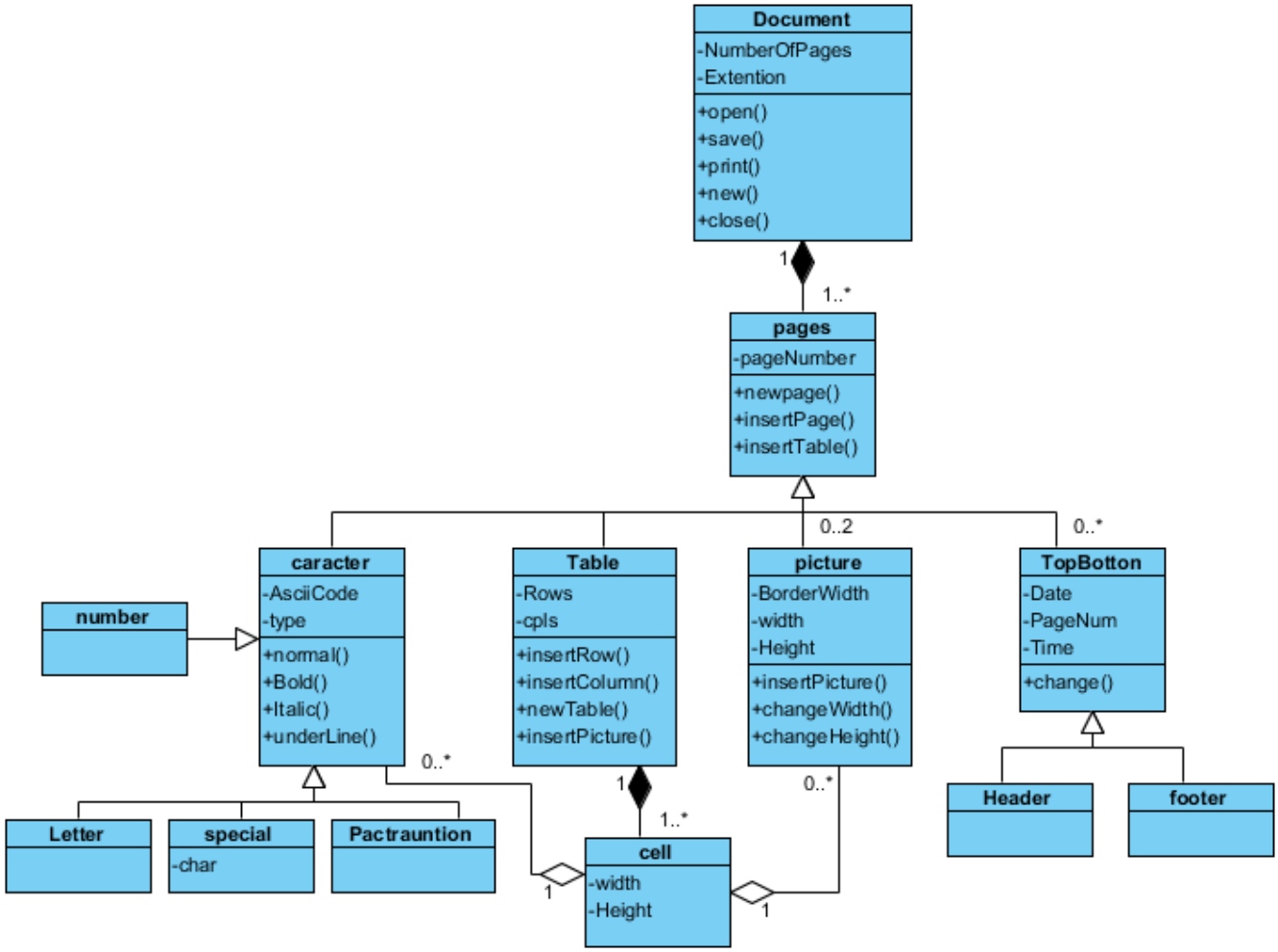


مثال ۲)



■ مصورسازی Visualization:

به ترسیم دیاگرام‌ها و در کل نمایش مسأله در قالب اشکال و روابط و ... در اصطلاح مصورسازی پروژه گفته می‌شود. مثال) نمودار کلاس class Diagram یک Document یا سند رسمی در نرم‌افزار word.

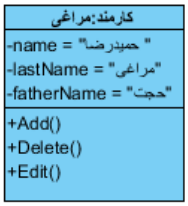


Object Diagram / Instance Diagram:

دیاگرام شیء زمانی کاربرد دارد که فهم دیاگرام کلاس به خاطر انتزاعی بودن بیش از حد آن به سادگی امکان پذیر نباشد با استفاده از دیاگرام شیء چندین نمونه شیء از دنیای واقعی و روابط بین آن‌ها نمایش داده می‌شود تا کلاس‌ها و روابط کلی آن‌ها سریع‌تر و بهتر قابل فهم باشد ضمن اینکه گاهی اوقات با رسم دیاگرام شی، روابط جدیدی بین کلاس‌ها کشف می‌شود.

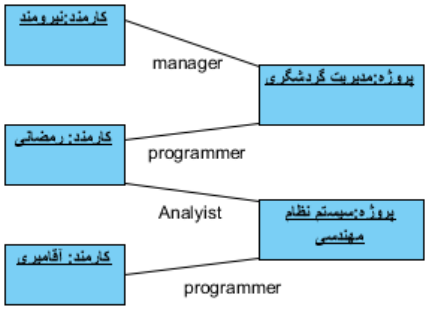
نحوه نمایش Object Diagram در UML:

مثال) نمایش یک نمونه شیء از کلاس کارمند در محیط دانشگاه.



نمایش ارتباط بین اشیاء:

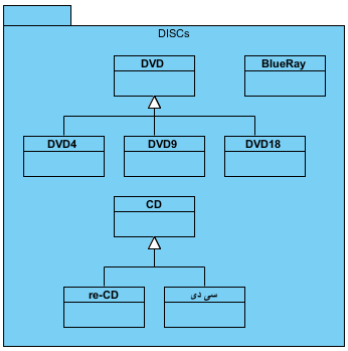
ارتباط بین اشیا با یک خط ساده نمایش داده می‌شود ممکن است روی خط نوع فعالیت (نقش یا role) شیء را بنویسند.



■ بسته (Package):

یک Package شبیه folder در ویندوز است. عناصر UML (مثل کلاس‌ها یا حتی U.Cها) را در سطح بالاتری در یک گروه قرار می‌دهیم که به این گروه Package گفته می‌شود . گاهی اوقات دیاگرام‌های کلاس بسیار گسترده می‌شوند و فهم آن‌ها مشکل. Packageها پیچیدگی دیاگرام کلاس را ساده‌تر می‌کنند پس کاربرد دیاگرام Package زمانی است که پروژه‌های بسیار پیچیده داریم با صدها کلاس.

مثال) یک Package برای دیسک‌ها در یک فروشگاه را رسم کنید.



Packageها نیز مثل کلاس‌ها روابط "وابستگی" و "تجمع" را بین خود دارند.

قبل از بررسی دیاگرام‌های پویای UML باید با مفهوم پردازش و فرآیند آشنا شویم و بتوانیم فرآیند را مدل‌سازی کنیم.

مدل‌سازی فرآیند

(مدل‌سازی گردش اطلاعات سیستم)

- هدف از مدل‌سازی فرآیند، شناخت و شفاف کردن گردش یا جریان اطلاعات در سیستم فعلی (سنتی) و نهایتاً در سیستم جدید (اتوماسیون) می‌باشد.
  اینکه چه پردازش‌هایی صورت می‌گیرد و ورودی و خروجی هر پردازش چیست؟
  چگونه اطلاعات در سیستم جریان می‌یابد؟
  مبدأ آن کجاست و نهایتاً به کجا می‌رود؟
- نشان داده می‌شود که در سیستم یا سازمان چه عملیات یا پردازش‌هایی صورت می‌گیرد و اطلاعات و مخازن ورودی و خروجی لازم برای هر پردازش (process) چیست.
- نتیجه مدل‌سازی فرآیند مدل فرآیند (process model) است که در رویکرد ساخت یافته با استفاده از "نمودار گردش داده‌ها" یا DFD نمایش داده می‌شود.

نمودار گردش داده‌ها Data Flow Diagram:

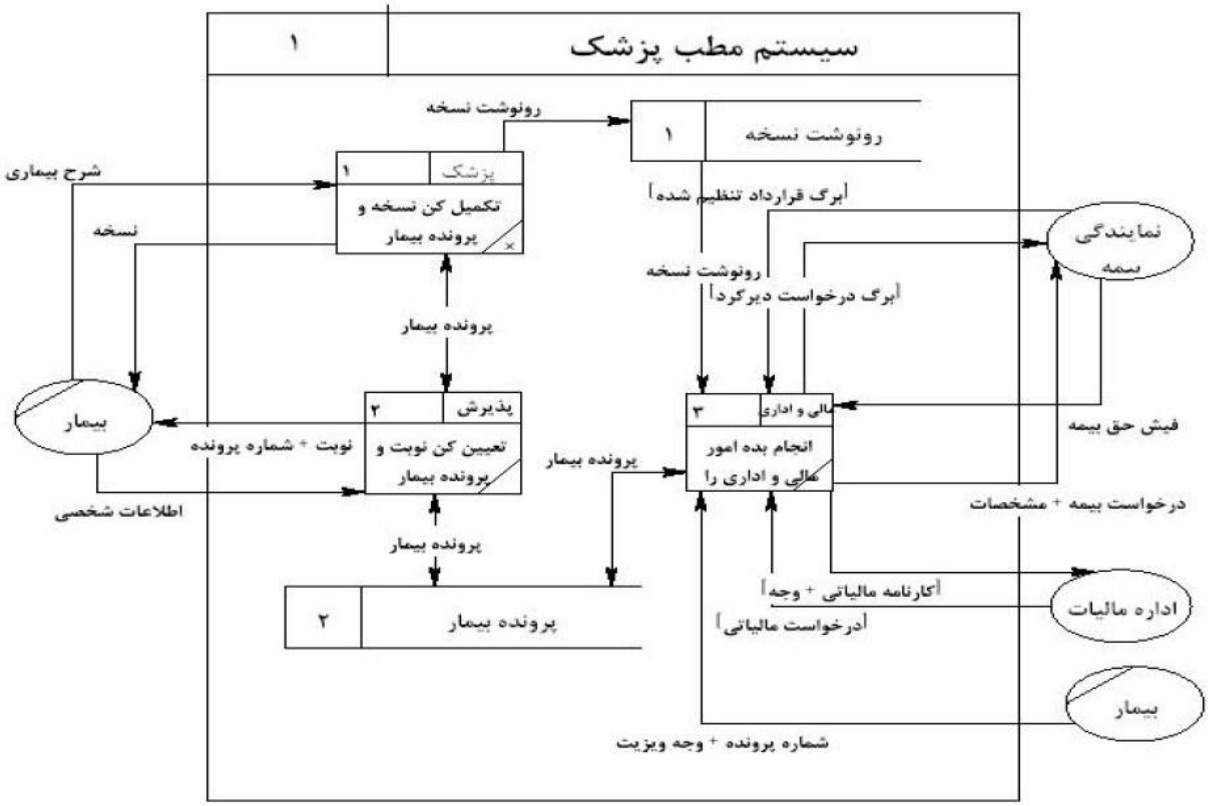
در نمودار گردش داده‌ها از چهار نماد یا symbol برای نمایش سیستم (از نظر عملیات و پردازش) استفاده می‌شود که عبارتند از:

- موجودیت خارجی (External Entity)
- پردازش یا پردازه (Process)
- جریان یا گردش داده (Data Flow:DF)
- انباره یا مخزن داده (Data Store:DS)

توسط این چهار عنصر می‌توان هر سیستمی را در هر سطحی از جزئیات نمایش دهیم. هدف از نمودارهای گردش داده‌ها فراهم کردن یک پل مفهومی بین کاربران و توسعه دهندگان سیستمی است این نمودار دارای ویژگی‌های زیر است:

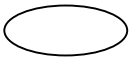
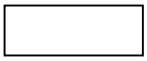
۱- گرافیکی و بصری‌اند و می‌دانیم که یک تصویر بهتر از هزاران کلمه است.
۲- امکان نمایش ماهیت و منطق سیستم را فراهم می‌کند این که سیستم چه(what) کارهایی انجام می‌دهد.
۳- به صورت سلسله مراتبی، جزئیات سیستم را در سطوح مختلف نمایش می‌دهد.
۴- با استفاده از آن‌ها می‌توان موافقت کاربر را نسبت به درک عملکرد سیستم راحت‌تر جلب کرد.

نمودار گردش داده‌های مطب پزشکی



توضیح عناصر مدل FD:

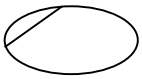
• **موجودیت خارجی:** موجودیتی است که بیرون از محدودیت سیستم قرار دارد و منبع داده‌های ورودی به سیستم و یا مقصد داده‌های خروجی از سیستم است. معمولاً به صورت یک بیضی یا مستطیل

نکاتی در مورد موجودیت خارجی:

۱- با نام مناسب نام‌گذاری کنید.

۲- به منظور جلوگیری از برخورد خطوط و خوانایی هر چه بیشتر نمودار و دسته بندی موضوعی Data Flowهای بین موجودیت خارجی و سیستم، موجودیت خارجی در نمودار می‌تواند به هر تعدادی تکرار شود کنار عنصر موجودیت خارجی تکرار شده یک خط مورب بکشید که نشان دهید این موجودیت تکرار شده است.



۳- مرز سیستم را مشخص می‌کنند و خارج زا سیستمی هستند که مطالعه می‌کنید.

۴- موجودیت خارجی می‌تواند یک انسان، سیستمی دیگر، یک بخش یا سازمان دیگری باشد.

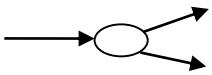
۵- موجودیت‌های خارجی بهتر است کدگذاری شوند برای هر موجودیت خارجی یک کد منحصر به فرد در نظر گرفته شود.

• جریان داده Data Flow:

به صورت یک فلش یا پیکان نمایش داده می‌شود که مشخص کننده حرکت داده‌ها در سیستم است در حقیقت منبع و مقصد اطلاعاتی را که جا به جا می‌شوند را نشان می‌دهد.

نکاتی در مورد Data Flow:

- اگر لازم شد یک سری داده به چندین مقصد مجزا ارسال شوند می‌توان به صورت "زیر انشعاب" ایجاد کرد.

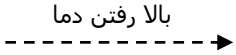


- اگر چندین Data Flow یک مقصد یکسان داشتند می‌توان به صورت زیر پیوند ایجاد کرد.

- باید فقط نمایش گر "داده" باشند نه کنترل
- همیشه باید دارای نام باشند و اینکه کدشان از روی منبع و مقصد جریان داده بدست آید.
  به طور مثال) کد 1.1-M1 یعنی جریان داده از فرآیند 1.1 به مخزن یا موجودیت خارجی M1
- نام جریان داده باید یک اسم (Noun) باشد و کنار Data Flow حتماً باید نوشته شود.

فرق Data Flow باControl Data :

جریان داده، داده‌ها را بین فرآیندها یا بین فرآیندها و مخزن داده‌ها حرکت می‌دهد و ماهیت داده‌ای را که جا به جا می‌شود را نمایش می‌دهد، در حالی که جریان کنترلی سیگنالی است که یک دستور یا فرمان با خودش حمل می‌کند یا مشخص می‌کند چه event (رویدادی) رخ داده است. جریان کنترلی به صورت زیر نمایش داده می‌شود.



• مخزن یا انبار داده:

محلی برای نگهداری اطلاعات می‌باشد، داده‌هایی را نمایش می‌دهد که در حال حرکت نیستند برای مثال فرم‌های دستی یا کامپیوتری، کارت‌های شاخص، دیسک مغناطیسی، حافظه انسان، دفتر و ...

نکاتی در مورد Data Store:

- با یک نام مناسب نام‌گذاری شوند.
- باید کدگذاری شوند اگر دستی است از حرف M(Manual) اگر دیجیتالی است از حرف D(Digital) و اگر موقتی است از حرف T(Temporary) و به دنبال آن یک عدد استفاده شود.
- به منظور خواناتر شدن نمودارها و جلوگیری از برخورد خطوط DSها می‌توانند به هر تعدادی در نمودار تکرار شوند. DS تکراری به صورت زیر نمایش داده می‌شود.

نام	کد

• پردازه یا Process:

یک فعالیت یا پردازشی است که داده‌های ورودی‌اش را به داده‌های خروجی تبدیل می‌کند، پس پردازه یا فرآیند، داده‌ها را تبدیل، دست‌کاری، ترکیب، مرتب و ...

نام پردازه باید توصیف کننده اهم وظایف پردازه باشد. این نام می‌تواند نام بخش، مکان، واحد، شخص مسئول یا پست سازمانی باشد.

نحوه نمایش پردازه

نام پردازه	کد پردازه
خلاصه عملکرد پردازه	

نکاتی در مورد پردازه‌ها یا Process:

۱- تبدیل یا تغییر داده‌ها را نشان می‌دهد همه فرآیندها باید دارای ورودی و خروجی باشند.
۲- حتماً دارای نام باشند و با کمک یک فعل مناسب و یک مفعول عملکردشان توصیف می‌شود.
۳- فعل استفاده شده نشان دهنده وظیفه‌ای است که پردازه به عهده دارد. پردازه‌ها باید کد گذاری شوند و بر اساس این کد سطوح جزئیات نمایش داده می‌شود.

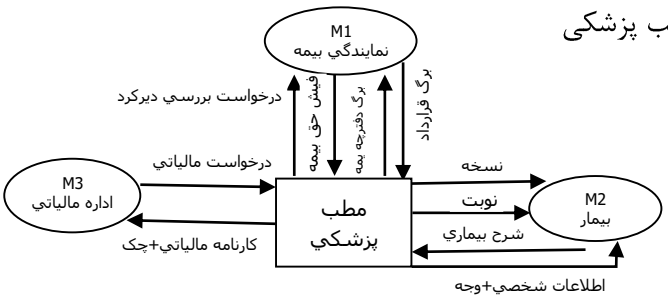
مثلاً پردازه‌های 1.4 , 1.3 , 1.2 , 1.1 شکسته می‌شود در کنار پردازه‌هایی که DFD سطوح پایین‌تر آن‌ها رسم می‌شود یک مثلث توخالی و در کنار پردازه‌هایی که بیش از این DFD سطوح بعدی آن‌ها رسم نمی‌شود یک مثلث شامل × رسم می‌شود.

روال تهیه DFDها:

۱- شناسایی و تهیه لیستی از موجودیت‌های خارجی که ورودی‌های سیستم را فراهم می‌کنند و یا خروجی سیستم را دریافت می‌کنند.
۲- شناسایی و تهیه لیست داده‌های ورودی از موجودیت‌های خارجی به سیستم و خروجی‌ها از سیستم به موجودیت خارجی.
۳- ایجاد نمودار متن Context Diagram به طوری که کل سیستم به صورت یک باکس در مرکز و موجودیت‌های خارجی ارسال و دریافت کننده اطلاعات در اطراف آن.
۴- شناسایی عملیات کسب و کار در محدوده سیستم (بر اساس عملیاتی که در سیستم انجام می‌شود سیستم باید به تعدادی زیر سیستم یا زیر پردازه شکسته می‌شود)
۵- شناسایی ورودی‌ها و خروجی‌های هر یک از زیر پردازش‌ها.
۶- ردیابی و یادداشت اینکه چه اتفاقاتی رخ می‌دهد. این کار برای هر Data Flow باید انجام شود (حرکت داده‌ها، ذخیره داده‌ها، تبدیل یا پردازش داده‌ها)
۷- زیر نمودارها را برای تشکیل یک نمودار کامل به یکدیگر متصل کنیم.
۸- دقت کنید همه فلش‌ها (DF)ها دارای منبع و مقصد باشند.
۹- دقت کنید که DSهایی که خروجی دارند، دارای ورودی باشند.
۱۰-نمودار را جهت ساده سازی مجدداً رسم کنید.
۱۱-نمودار را مرور کنید.
۱۲-نمودار را گسترش داده و در صورت لزوم مراحل فوق را تکرار کنید.

نمودار متن:

نمونه‌ای از نمودار متن برای مطب پزشکی



توضیح نمودار متن:

در نمودار متن یک Black Box (جعبه سیاه) برای کل سیستم در نظر می‌گیریم و در کنار این Box موجودیت‌های خارجی را نمایش می‌دهیم در این مرحله به داخل این باکس کاری نداریم و فعلاً نمی‌خواهیم بدانیم که درون آن چه می‌گذرد بلکه تمرکز خود را روی موجودیت‌های خارجی و ارتباطاتی که با سیستممان چه داده‌هایی رد و بدل می‌شود. در نمودار متن ارتباط بین موجودیت‌های

خارجی در نظر گرفته نمی‌شود چون بیرون سیستم هستند(اگر هم خواستیم ارتباطی بین آن‌ها برقرار کنیم به صورت خط چین نمایش می‌دهیم).

نمودار متن مرز یک سیستم را نمایش می‌دهد. اگر مرز سیستم به خوبی تشخیص داده نشود آنگاه در رسم نمودار متن با این شکل مواجه می‌شویم که کدام موجودیت خارجی است و کدام موجودیت داخلی یا ممکن است برخی از موجودیت‌های خارجی از قلم بیفتد.

توضیح DFD سطح یک:

در سطح یک سعی می‌شود قدری از جزئیات سیستم شفاف شود برای این کار سیستم را به تعدادی زیر سیستم تجزیه می‌کنیم اینکه سیستممان را به چند زیر سیستم بشکنیم به شناخت و تحلیل ما و وضعیت موجود بر می‌گرداند. برای رسم DFD دقیق‌تر و بهتر باید ببینیم مسؤولیت‌های کلان سیستم چیست و در کجا انجام شود؟ و از همه مهم‌تر اهم وظایف سیستم چیست؟ که در کل تشخیص عملکرد جهت تجزیه و شکستن آن به درک سیستم فعلی و همچنین تجزیه زیاد بر می‌گردد.

چند مثال)

در یک دانشکده ممکن است سه مسئولیت کلان عبارت باشند از:

۱- امور دانشجویی
۲- امور اساتید
۳- امور دروس

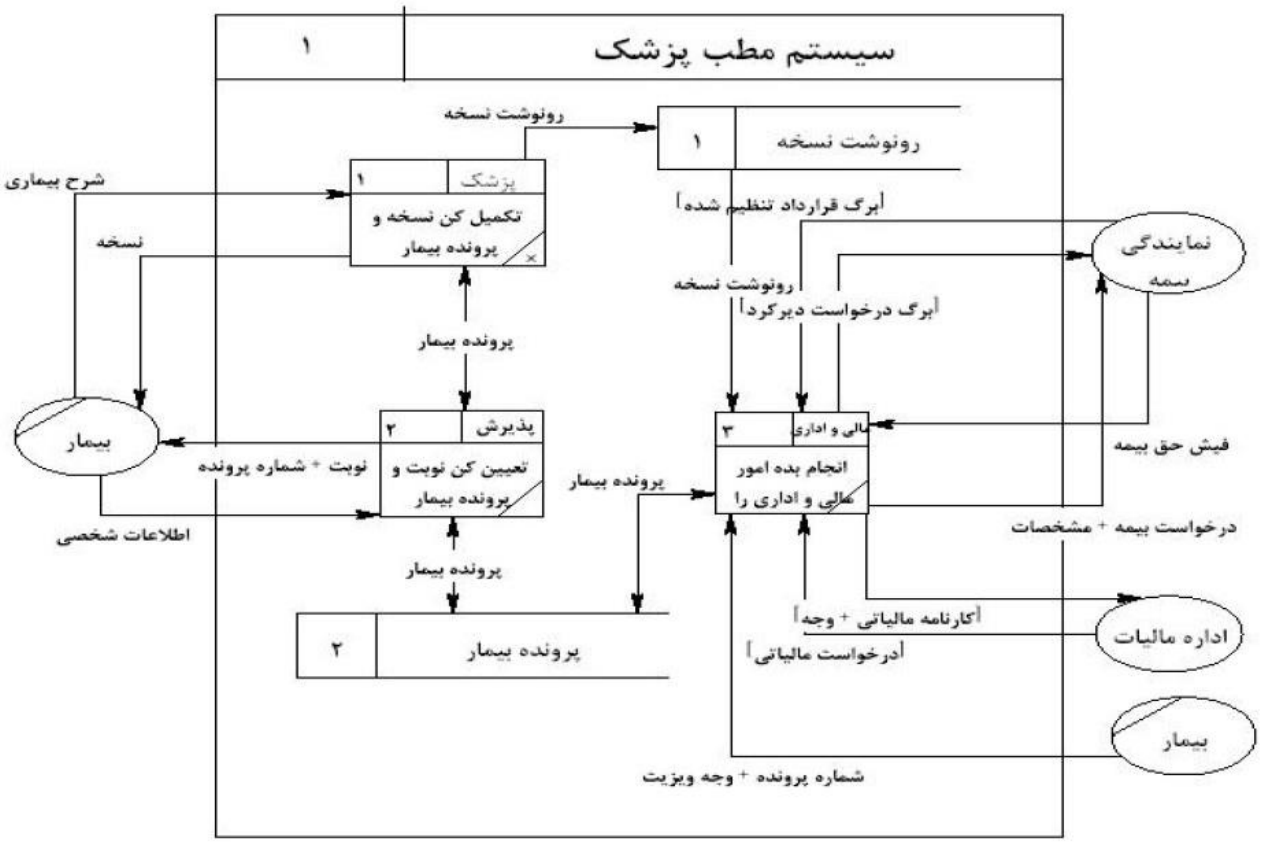
در کتابخانه می‌توان دو مسئولیت کلان زیر را متصور شد:

۱- امور اعضا
۲- امور کتاب‌ها

در یک فروشگاه می‌توان سه مسئولیت کلان زیر را متصور شد:

۱- امور فروش
۲- امور خرید
۳- امور مدیریتی

DFD سطح یک مطب پزشکی:



نکاتی در مورد شکستن پردازه‌ها:

- همان‌طور که مشاهده می‌کنید سیستم را بر اساس تقسیم وظایف به سه زیر سیستم یا در اصطلاح سه پردازه شکسته‌ایم. در شکستن یا تجزیه همیشه دو سؤال مهم مطرح است:

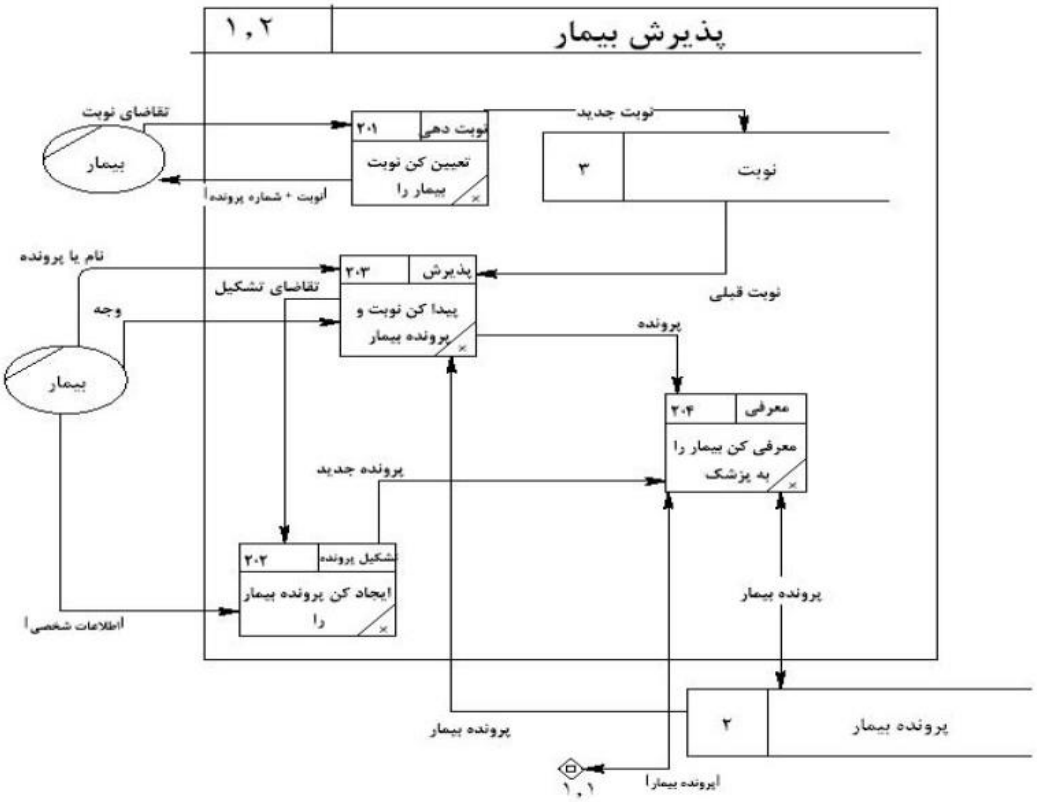
۱- آیا یک پردازه باید شکسته شود؟ و در صورت تجزیه به چند زیر پردازه باید بشکند؟

۲- عملیات شکستن پردازه‌ها تا کی باید ادامه یابد.

جواب سؤال‌ها:

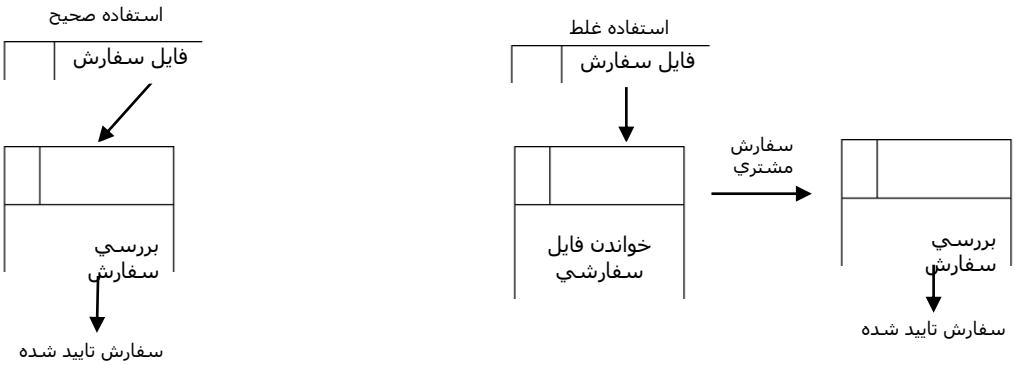
- در صورتی که یک پردازه باید شکسته شود که تمام جزئیات کافی درباره سیستم و وظایف آن را مستند نکرده‌ایم.
- تا زمانی باید یک پردازه شکسته شود که به یک تصمیم گیری، مصاحبه یا عملیات منفرد (update، حذف، افزودن و ...) برسیم. ضمن اینکه شرح پردازه به قدر کافی کوچک شده باشد.
- در سطح یک باید سیستم را به پردازه‌های سنگین و کلی بشکنیم.
- حتی الامکان پردازه‌ها باید هم وزن باشند یعنی پردازه‌های در یک سطح باید با هم توازن داشته باشند.

DFD سطح دو پذیرش بیمار:

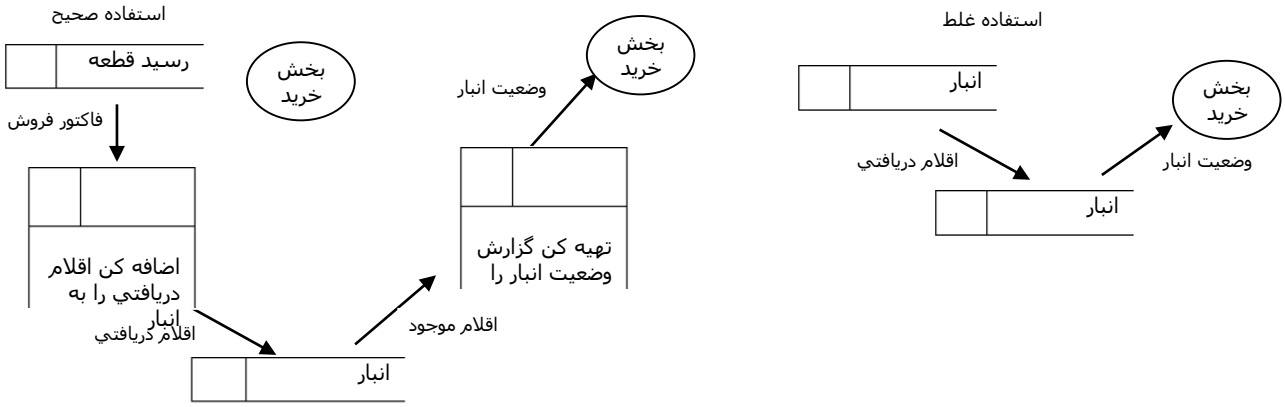


نکاتی در مورد DFDها:

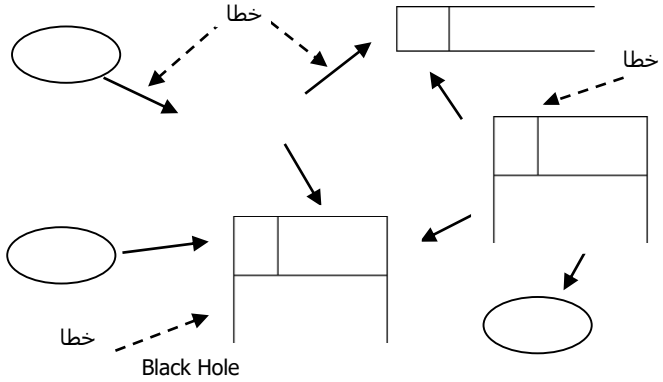
از پردازه‌ها برای نمایش پردازش یا تبدیل داده‌ها استفاده کنید جایی که داده‌ها پردازش یا تبدیل نمی‌شوند لازم نیست پردازه‌ای استفاده شود. مثل خواندن رکوردی از فایل



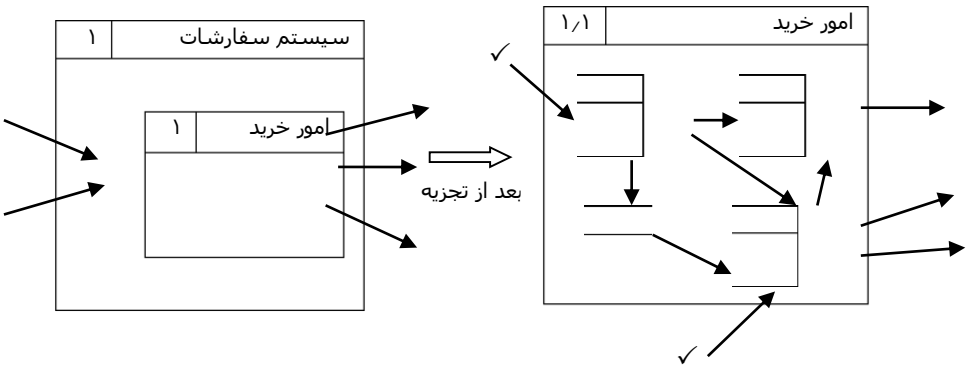
۱- جریان داده یا DF باید از یک پردازه شروع شود یا به یک پردازه ختم شود یعنی حداقل یک طرف DF باید پردازه باشد.



۲- از پردازه‌ای که ورودی دارد ولی خروجی ندارد(Black Hole)[چاله سیاه] و همچنین پردازشی که خروجی دارد ولی ورودی ندارد(Miracle). همچنین پردازه‌ای که نمی‌تواند از روی ورودی‌هایش خروجی‌های مشخص شده را تولید کند یعنی داده ورودی برای تولید خروجی‌ها کافی نیست (Gray Hole)اجتناب شود.



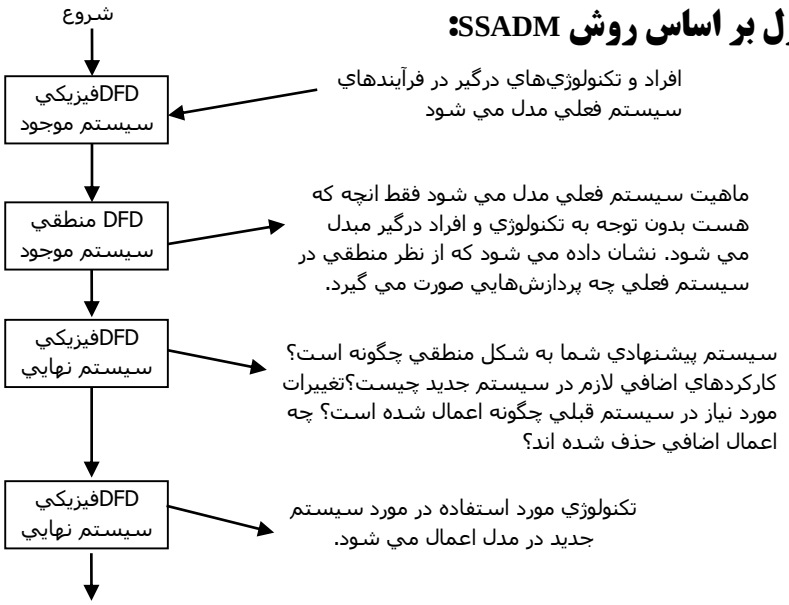
باید خاصیت Level Balancing وجود داشته باشد یعنی ورودی و خروجی یک پردازه قبل و بعد از تجزیه معین هم باشد. همچنین نام آن‌ها نیز یکی باشد.



مقایسه DFD با فلوچارت:

- DFD یک فلوچارت نیست.
- استفاده از فلوچارت به منظور توصیف برنامه یا درک توالی فعالیت‌های یک پردازه می‌تواند مفید باشد.
- در فلوچارت خطوط می‌تواند بدون برچسب باشد ولی در DFD جریان داده بدون نام معنی ندارد.
- در فلوچارت منطق شرطی داریم ولی در DFD نه.
- در فلوچارت ترتیب مهم است اما در DFD نه.

انواع DFDهای قابل اصول بر اساس روش SSADM:



DFD فیزیکی:

مشخص می‌کند که سیستم چه کاری انجام می‌دهد و چگونه پیاده‌سازی می‌شود در مدل طراحي و پياده سازي دا DFD فیزیکی سیستم موجود را می‌کشیم تا وضعیت فعلی سیستم را درک کنیم و بتوانیم تعیین کنیم چه افراد و تکنولوژی‌هایی داده‌ها را پردازش می‌کنند.

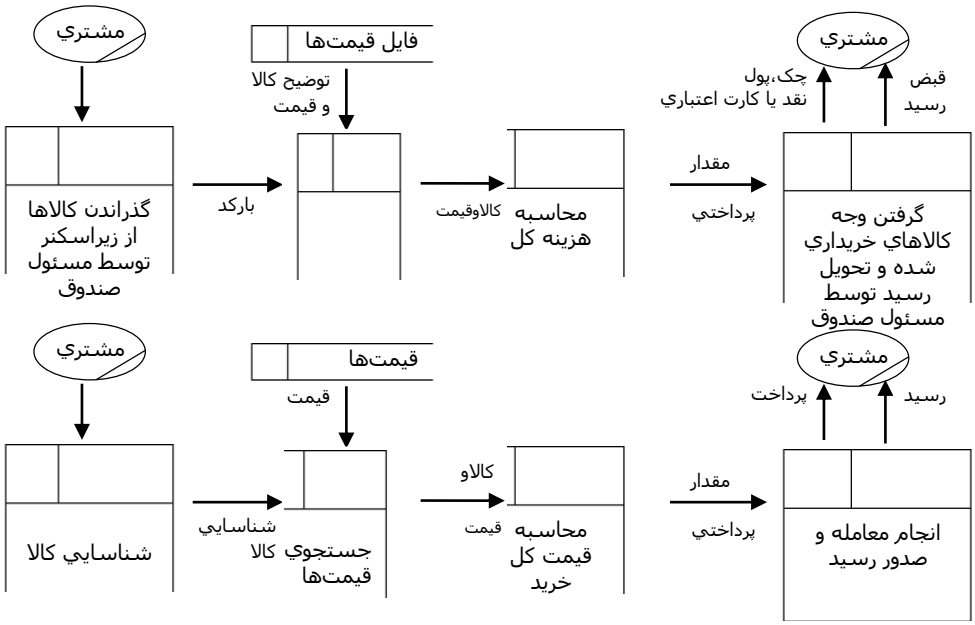
نکته)

- وقت خود را زیاد صرف کشیدن DFD فیزیکی نکنید چون وضعیت سیستم موجود را نشان می‌دهد و این سیستم باید تغییر کند. البته اگر زیاد وقت گذاشتید سیستم موجود را بهتر درک کرده‌اید.
- در DFD فیزیکی، نام پردازه برابر با نام واحد یا بخش،نام مسئول و یا نام ماشین خاصی که پردازش را انجام می‌دهد قرار داده می‌شود.

DFD منطقی:

DFD منطقی مشخص می‌کند سیستم چه کاری انجام می‌دهد به هیچ وجه به مسائل پیاده‌سازی و تکنولوژی توجه نمی‌شود. فقط ماهیت سیستم مدنظر است کاری به تکنولوژی‌های درگیر نداریم و خلاصه در رسم DFD منطقی تمام جنبه‌های فیزیکی سیستم را کنار بگذاریم.

مثال) DFD فیزیکی وDFD منطقی زیر سیستم صندوق



شرح عناصر DFD:

برای شرح عناصر DFDباید از قبل آن‌ها را کدگذاری کنیم و برای هر عنصر یعنی برای مؤلفه یک شماره یا یک کد یکتا در نظر بگیریم که این کد یک راه دسترسی سریع به آن عنصر است.

آرم پروژه	عنوان فرم: کد پروژه: نام پروژه:	کد فرم: صفحه از	نام تهیه کنندگان: تاریخ تولید:

- **شرح موجودیت‌های خارجی:**

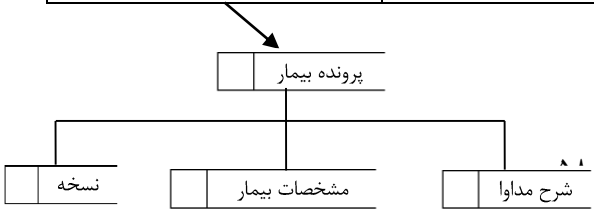
آرم پروژه	عنوان فرم:شرح موجودیت‌های خارجی کد پروژه: نام پروژه:سیستم مطب	کد فرم:C5 صفحه 1 از 3	نام تهیه کنندگان: تاریخ تولید:

کد	نام موجودیت خارجی	توضیح
E1	نماینده بیمه	یک سازمان،اداره‌ای است که طرف حساب مطب است
E2	بیمار	درخواست درمان را به بیمار می‌دهد
E3	اداره مالیات	مطب، مالیات درآمدهای خود را محاسبه کرده و به حساب اداره مالیات واریز می‌کند

- **شرح مخزن یا انباره داده:**

آرم پروژه	عنوان فرم:شرح انبار داده کد پروژه: نام پروژه:سیستم مطب	کد فرم:C6 صفحه 2 از 3	نام تهیه کنندگان: تاریخ تولید:

کد	نام	فیلدهای اطلاعاتی	توضیح
M1	رونوشت نسخه	شماره نسخه،نام نسخه کننده،شماره بیمه،کد پزشک،شرح مداوا،تاریخ و...	_____
M2	مشخصات بیمار	نام و نام خانوادگی، تاریخ تولد، شماره شناسنامه	_____
M3	شرح مداوا	کد دارو، نام دارو، میزان مصرف	_____
M4	پرونده بیمار	_____	مشتمل است برM1,M2,M3



•شرح DFیا Data Flow:

آرم پروژه	عنوان فرم:شرح انبار داده کد پروژه: نام پروژه:سیستم مطب	کد فرم:C6 صفحه 2 از 3	نام تهیه کنندگان: تاریخ تولید:

از	به	نام	فیلدهای اطلاعاتی	توضیح
1.1	E2	نسخه	ــــــــــ	رجوع شود به M1
E2	1.2	اطلاعات کلی	نام و نام خانوادگی، شماره شناسنامه	رجوع شود به M1
.				

نکته) برای توصیف DFها در سطوح مختلف دو راه وجود دارد:

۱- در همان سطح بالا DF را شرح داده و اگر در سطوح بعدی هم بود در شرح آن بنویسیم رجوع شود به سطح Xرجوع به بالاترین سطح) یعنی در حقیقت در سطوح بالا نیز DFها را شرح دهید.

اشکال: حجم سند شرح DF زیاد می‌شود.

۲- DFها را فقط در بالاترین سطح شرح دهیم بنابراین حجم مستندات شرح DFها کم می‌شود ولی دسترسی به آن‌ها کمی سخت.

• شرح پردازه‌های جزئی:

پردازه‌های پایین‌ترین سطح که به آن‌ها پردازه‌های جزئی (Partial Process) گفته می‌شود باید توصیف شود و منطق آن‌ها بیان شود. باید نشان دهیم در داخل آن پردازه چه کارهایی انجام می‌شود؟ چگونه ورودی‌های پردازه جزئی به خروجی‌ها تبدیل می‌شوند.

اهداف:

۱- کاهش ابهام در پردازه‌ها: وقتی قرار است منطق پردازه‌ها را در قالب شبه کد Pseudo codeبیان کنیم اگر از پس این کار برنیاییم و دچار ابهام شویم این نشان می‌دهد که شناخت ما هنوز کافی نیست و باید از روش‌های جمع آوری اطلاعات مانند مصاحبه و سایر روش‌ها هر چیزی را که کم داریم را بدست آوریم و نقاط مبهم را برطرف کنیم.

۲- بدست آوردن توضیح دقیق از آنچه که باید انجام شود.

۳- اعتبار سنجی طراحی: طراحی منعکس کننده دقیق آنچه که مشخص کرده‌ایم است.

تکنیک‌های موجود برای شرح پردازه‌های جزئی:

۱- انگلیسی ساخت یافته structural English

۲- جدول تصمیم گیری Decision Table

۳- درخت تصمیم Decision Tree

بسته به تجربه موقعیت شما، منطق فرآیند و پیچیدگی آن از ترکیبی از این تکنیک‌ها استفاده کنید.

• توضیح انگلیسی ساخت یافته:

از ساختمان موجود در برنامه نویسی ساخت یافته برای توصیف منطق یک فرآیند استفاده می‌کنیم. انگلیسی ساخت یافته تکنیکی است برای ارتباط بین تحلیل‌گر، کاربر نهایی و طراح.

در انگلیسی ساخت یافته به جزئیات و تکنیک‌های کلی (مثل باز و بسته کردن فایل‌ها و مقداردهی اولیه متغیرها و...) نمی‌پردازیم.

قوانین انگلیسی ساخت یافته :

۱- فقط از افعال امری استفاده می‌شود(چاپ کن یا print)

۲- فقط از اسامی و اصطلاحات تعریف شده در فرهنگ داده‌ای پروژه استفاده می‌شود(مثل printf در Dictionary زبان c که غیر از آن قابل استفاده نیست)

۳- از صفات و قیدهای تعریف نشده (مثل "خوب") استفاده نمی‌شود مگر اینکه در Dictionary داده‌ای پروژه Data Dictionary گفته باشیم که مثلاً خوب معادل ۷۵٪ است.

۴- از کلماتی که جریان طبیعی را از بین ببرد استفاده نمی‌شود (مثل دستور goto در زبان c)

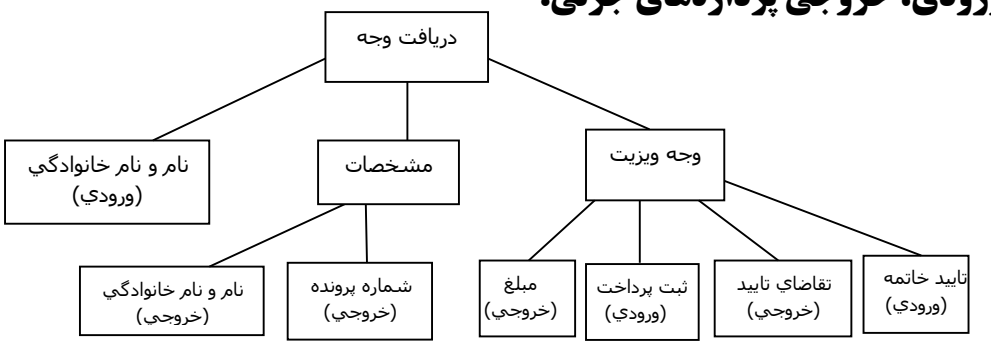
۵- از ساختارهای معتبر منطقی استفاده می‌شود مثل ساختار شرطی(if,switch)، ساختارهای تکرار(for,while,...) و ساختارهای توالی.

۶- دقت کنید که قوانین خیلی مشخص و محکمی وجود ندارد. زبان بر اساس قوانین مورد نظر خود تعریف کرده و در پروژه استفاده کنید.

مثال)

آرم پروژه	عنوان فرم:شرح پردازه‌های جزئی نام پروژه:سیستم مطب		کد فرم:C8 صفحه 3 از 20	نام تهیه کنندگان: تاریخ تولید:
کد پردازه	نام پردازه	جریان داده ورودی	جریان داده خروجی	شرح مختصر	منطق پردازه
1.4	دریافت وجه	۱.نام و شماره پرونده از استور لیست مراجعه کنندگان ۲.نام مراجعه کننده از مسئول صندوق	علامت پرداخت در ds دیتا استور لیست مراجعه کنندگان ثبت وجه، شماره پرونده و تاریخ	وظیفه پرداخت وجه ویزیت بعدی و ثبت در دفتر اسناد و زدن علامت در لیست مراجعه کنندگان را به عهده دارد.	- بخوان نام و نام خانوادگی مراجعه کننده را از ورودی - جستجو کن نام و ... را در لیست مراجعه کنندگان - نمایش بده مبلغ ویزیت را - بخوان علامت پرداخت مبلغ را از ورودی - ثبت کن شماره پرونده، تاریخ، مبلغ و وجه دریافتی را از ردیف دفتر اسناد حسابداری

ساختار ورودی، خروجی پردازه‌های جزئی:

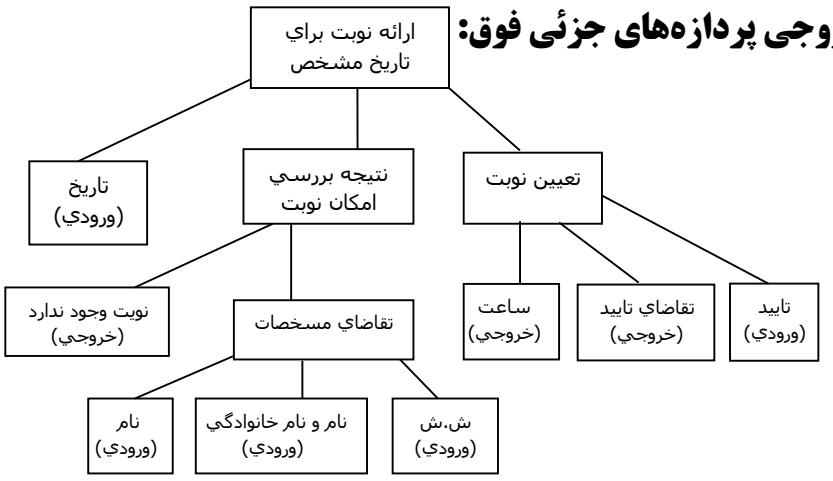


مثالی از شرح پردازه‌های جزئی:

آرم پروژه	عنوان فرم:شرح پردازه‌های جزئی نام پروژه:سیستم مطب	کد فرم:C8 صفحه 3 از 20	نام تهیه کنندگان: تاریخ تولید:

کد پردازه	نام پردازه	جریان داده ورودی	جریان داده خروجی	شرح مختصر	منطق پردازه
1.2-2.2	ارائه نوبت برای تاریخ مشخص			وظیفه ارائه نوبت در تاریخ مشخص را بر عهده دارد	-دریافت کن تاریخ را از ورودی -جستجو کن در فایل نوبت برای جای خالی در این تاریخ -اگر جای خالی پیدا نشد آنگاه نمایش بده "جای خالی وجود ندارد" در غیر این صورت دریافت کن نام و نام خانوادگی و ش.ش را از ورودی -جستجو کن در فایل نوبت با استفاده از نام،نام خانوادگی ش.ش -اگر بیمار پیدا شد آنگاه اگر بیمار در نوبت قبلی مراجعه نکرده باشد آنگاه کسب تکلیف کن -در غیر این صورت اگر بیمار نوبت آتی دارد آنگاه حذف کن نوبت آتی را در غیر این صورت اعلام کن وجود نوبت و ساعت ممکن را -دریافت کن تایید و ثبت نوبت را از ورودی -ثبت کن نوبت را در فایل نوبت

ساختار ورودی خروجی پردازه‌های جزئی فوق:



اگر منطق پردازه‌ها پیچیده باشد و شامل چندین if تودرتو باشد بهتر است منطق آن‌را با کمک جدول تصمیم یا درخت تصمیم بیان کنیم.

• جدول تصمیم و درخت تصمیم:

جدول تصمیم از دو بخش تشکیل شده است:

۱- بخش شرط یا شرط‌ها(شامل شرط‌هایی است که در منطق فرآیند وجود دارد)

۲- بخش اعمال (شامل اعمالی است که در صورت برقراری شرط‌ها باید انجام شود)

شرط ۱	✓		
شرط ۲		✓	
شرط n	✓		
عمل ۱		✓	
عمل ۲		✓	
عمل n			

مثال) خروج از خانه در یک روز بارانی و یا سرد

شرط‌ها:

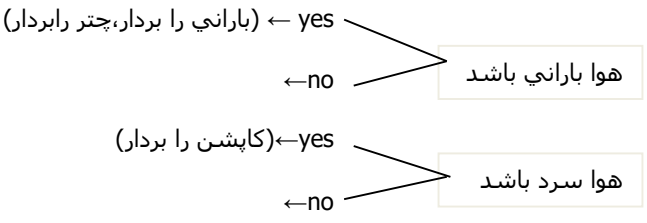
- هوا بارانی باشد یا نباشد
- هوا سرد باشد یا نباشد

اعمال:

- بارانی را بردار.
- کاپشن را بردار.
- چتر را بردار.
- برو بیرون.

- جدول تصمیم مثال فوق

هوا بارانی باشد	Y	Y	N	N
هوا سرد باشد	Y	N	Y	N
بارانی را بردار	✓	✓	×	×
کاپشن را بردار	✓	×	✓	×
چتر را بردار	✓	✓	×	×
برو بیرون	✓	✓	✓	✓



در این حالت با توجه به تعدد شروط درخت تصمیم مناسب نیست.

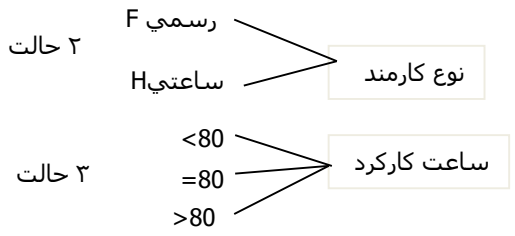
مثال)

- اگر کارمند رسمی باشد و ساعت کاری وی کمتر از ۸۰ساعت باشد حقوق پایه وی را محاسبه کن و گزارش غیبت را رد کن.
- اگر ساعت کارکرد وی برابر ۸۰ باشد آنگاه حقوق پایه‌ی وی را محاسبه کن.
- اگر ساعت کاری وی از ۸۰بیشتر باشد حقوق پایه را محاسبه کن و اضافه کاری رد کن.

- اگر کارمند ساعتی باشد و ساعت کاری وی کمتر از ۸۰ساعت باشد دستمزد ساعتی را محاسبه کن و گزارش غیبت را رد کن.
- اگر ساعت کارکرد وی مساوی ۸۰یا بیشتر از ۸۰ باشد دستمزد ساعتی وی را محاسبه کن.

شرط‌ها:

	کارمند	F	F	F	H	H	H
شرایط	ساعت کارکرد	80<	80=	80>	80<	80=	80>
اعمال	حقوق پایه را محاسبه کن	✓	✓	✓	×	×	×
	اضافه کاری رد کن	✓	×	×	×	×	×
	گزارش غیبت رد کن	×	×	✓	×	×	✓
	دستمزد ساعتی را محاسبه کن	×	×	×	✓	✓	✓



2×3=6 حالت

از جدول تصمیم برای بیان مشخصات پردازه‌ها در سیستم‌های بانکی زیاد استفاده می‌شود.

نکته) اگر ده شرط داشته باشیم و هر شرط در حالت داشته باشد 2^{10} یعنی ۱۰۲۴ ستون برای جدول خواهیم داشت. یعنی وضعیت به توان شرط (وضعیت)شرط

مثال)اگر هوا بارانی باشد. چتر برداشته می‌شود و اگر نباشد چتر لازم نیست.

هوا بارانی باشد	Y	N
چتر را بردار	✓	×

بنابراین ممکن است گاهی جدول تصمیم بسیار بزرگ شود در نتیجه نیاز داریم که این جدول بهینه شود.

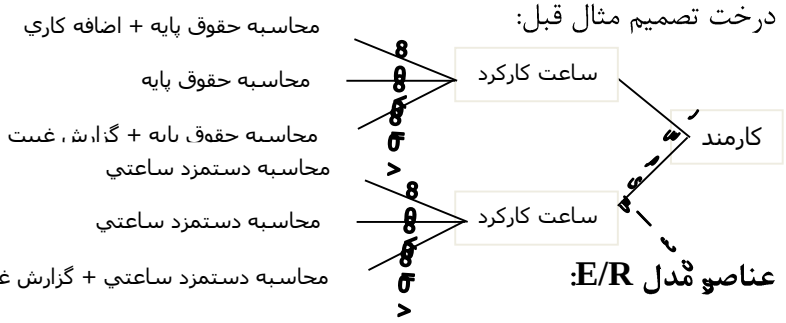
راه‌های بهینه کردن اندازه جدول:

۱- حالت‌های بی تفاوت را ادغام کنیم.

۲- ممکن است بعضی از قوانین (ستون‌ها) اصلاً پیش نیاید و بتوانیم از جدول آن‌ها را حذف کنیم.

H	H	F	F	F	کارمند
80>	80<=	80>	80=	80<	ساعت کارکرد

درخت تصمیم مثال قبل:



عناصر مدل E/R:

در مدل E/RساختارDatabaseتوسط عناصر زیر به صورت گرافیکی نمایش داده می‌شود.

۱- موجودیت (Entity) ۲- صفات (Attributes) ۳- روابط (Relation ship)

تعریف موجودیت:

an entity represents a real-world object or concepts such as an employee or a project

یک موجودیت نمادی از یک شیء یا مفهوم در دنیای واقعی است مثل کارمند یا پروژه در هر محیط عملیاتی مجموعه‌ای از موجودیت‌ها وجود دارد که مدل ساز باید این مجموعه را به درستی تشخیص دهد.

مثال) محیط عملیاتی:دانشگاه

چند موجودیت در این دانشگاه :دانشجو،کارمندان،استاد،کلاس‌ها،درس،ترم،...راهنمای عملی تشخیص موجودیت‌های محیط عملی:

۱.یک موجودیت معمولاً چندین نمونه در محیط دارد.

۲.کاربرد مجموعه‌ای از اطلاعات در مورد آن می‌خواهد .

۳.معمولاً بیش از یک صفت دارد.

مثال۲) سیستم یک آموزشکده‌ی زبان

برخی از موجودیت‌ها:زبان آموز،اساتید،کلاس‌ها،کارمندان،ترم...

توجه کنید که در این سیستم خود آموزشکده یک موجودیت نیست چون بند اول نقض می‌شود.اما اگر بخواهید برای کل آموزشکده‌های زبان سراسر کشور برنامه نویس آنگاه آموزشکده یک موجودیت است.

انواع موجودیت:

۱.موجودیت قوی یا مستقل (strong Entity)

۲.موجودیت ضعیف یا وابسته (weak Entity)

تعریف موجودیت قوی:موجودیتی است که وجود آن وابسته به وجود موجودیت دیگری نیست.

مثلاً در سیستم دانشگاه یک "دانشجو" موجودیتی قوی است حتی اگر هیچ درس در یک‌ترم اخذ نکند با زهم دانشجو است. "استاد" نیز یک موجودیت قوی است حتی اگر در یک ترم درس نداشته باشد.

تعریف موجودیت ضعیف: موجودیتی است که وجودش وابسته وجود موجود دیگری است. مانند موجودیتی به نام "گواهی‌نامه"(و مشخصات آن)که وابسته به "شخص" است اگر آن شخص حذف شود گواهی نامه ی او دیگر مطرح نیست یا مثل موجودیت "والدین دانشجو" که اگر خود شخص حذف شود والدین دیگر مطرح نیست.

تعریف صفت:هر موجودیت دارای مشخصاتی است که آن موجودیت را بیشتر توصیف می‌کنداین مشخصات در اصطلاح Attribute نامیده می‌شود.

An attribute represents some property of interest that further describes an entity.

مثال: موجودیتی به نام دانشجو دارای صفاتی مانند شماره دانشجویی، نام و نام خانوادگی، نام پدر، تاریخ تولد، تلفن و آدرس و.....

مجموعه‌ای از اطلاعاتی در مورد هر یک از نمونه‌های Ei $Ei\rightarrow \{Ai\}_{i=1,2,...}$

هر صفت از بخش‌های زیر تشکیل شده:

۱-نام ۲- معنا ۳- نوع(عددی،رشته‌ای و.....) ۴- طول مقدار(مثلاً رشته ۲۰کاراکتری) ۵- میدان(دامنه):مقادیر مجاز

مثال:نمره۱- نام: Grade ۲- معنا:نمره دانشجو ۳- نوع:Decimal ۴- طول مقدار:برای نوع دسیمال طول معنا ندارد. ۵- دامنه:0-20

انواع صفت:

۱- صفت می‌تواند شناسه باشد یا نباشد: صفت شناسه عامل جداسازی یک نمونه از نمونه‌ی دیگر است. مثلاً شماره دانشجویی موجودیت"دانشجو"

صفت شناسه دارای دو ویژگی زیراست:

۱-۱- یکتایی مقدار دارد. ۱-۲- حتی الامکان طول کوتاه‌تر و ترجیحاً از صفات طبیعی است مثل کد ملی

۲- صفت ممکن است ساده باشد یا مرکب:صفت مرکب صفتی است که از تعدادی صفت ساده تشکیل شده باشد مانند"آدرس"دانشجو که شامل"نام استان"،"نام شهر"،"نام خیابان"،نام کوچه"و"پلاک"

۳- صفت ممکن است تک مقداری یا چند مقداری باشد. مثلاً تلفن اساتید که ممکن است تا سه شماره

۴- صفت ممکن است واقعی یا مجازی باشد (مشتق شده، محاسبه شده)

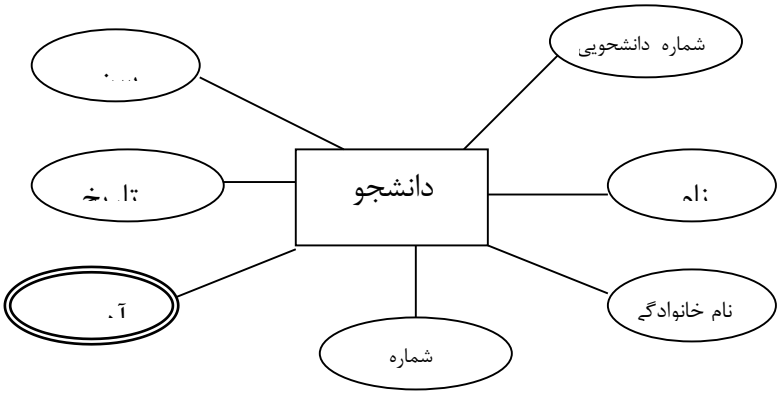
صفت واقعی مقدار ذخیره شده درDatabase است مثل نام دانشجو: نام استاد

صفت مجازی مقدار ذخیره شده درDatabase ندارد بلکه در پردازش از روی مقادیر ذخیره شده به دست می‌آید. مثل سن(از تفریق تاریخ تولد از تاریخ سال جاری)،معدل

دیاگرامE/R (E/R Diagram) :یک دیاگرامE/Rگرافی است که نماد موجودیت‌ها،صفات و روابط است.

Symbol	meaning
	موجودیت
	صفت
	رابطه
	صفت مجازي
	موجودیت ضعیف
	صفت چند مقداري

مثال) نمودار نمایش دانشجو در محیط عملیاتی دانشگاه را رسم کنید.



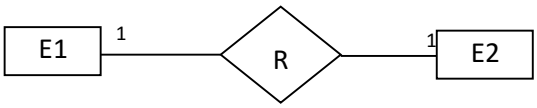
رابطه‌ها Relation ship**:**

Relationships are connections among two or more entity sets.

رابطه‌ها ارتباط بین دو یا چند موجودیت هستند.

به طور مثال درس و دانشجو دو موجودیت هستند،رابطه"اخذ کردن"دو موجودیت را به هم متصل می‌کنند. رابطه:دانشجو اخذ می کند درس را.

انواع رابطه بین دو موجودیت:



۱- رابطه one-one(یک به یک)

اگر یک نمونه ازموجودیتE1بایک موجودیت به نامE2رابطه داشته باشد و بالعکس آن گاه رابطه بینE1وE2یک به یک است.

رابطه"مدیریت"بین"استاد"و"دانشکده"

یعنی یک نمونه از دانشکده فقط توسط یک استاد مدیریت می‌شود و یک استاد فقط می‌تواند یک دانشکده را مدیریت کند.

۲- رابطهmany-one(یک به چند)یا(1بهN):اگر یک نمونه از موجودیتE1باNنمونه ازموجودیتE2درارتباط باشد در حالی که یک نمونه از موجودیت E2فقط با یک نمونه ازموجودیتE1درارتباط باشد در این صورت رابطه1بهNبین E1وE2برقراراست.

مثال:رابطه بین دانشگاه و دانشجو در سیستم کل دانشگاه‌های کشور.

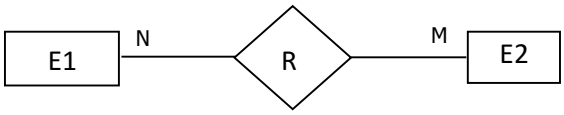
هر دانشجو فقط در یک دانشگاه می‌تواند تحصیل کند اما هر دانشگاه می‌تواند با چندین دانشجو رابطه داشته باشد.

مثال۲:رابطه بین شخص و استان

هر شخص می‌تواند در یک استان سکونت کند اما هر استان می‌تواند با چندین شخص ارتباط داشته باشد.

نکته) رابطه یک به یک و1بهNمعمولادریک سیستم خیلی کم دیده می‌شود.

۳- رابطهmany-many(چند به چند)یا(Nبه‌M):



اگر یک نمونه از موجودیتE1باMنمونه ازموجودیتE2ویک نمونه ازموجودیتE2باNنمونه ازموجودیتE1درارتباط باشد بین E1و E2 رابطه Nبه‌M برقراراست.

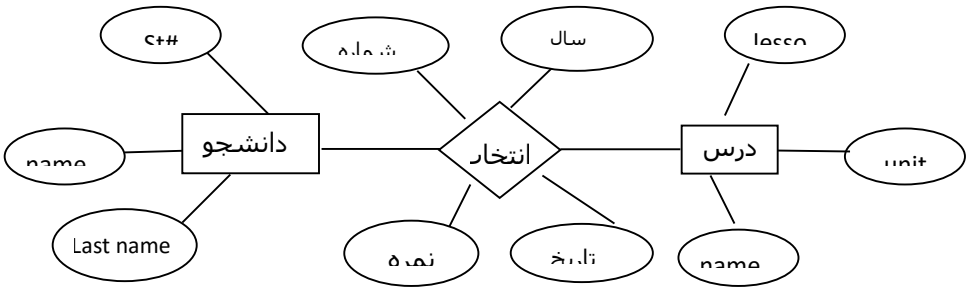
مثل رابطه بین دانشجو و درس(اخذ کردن)

یک دانشجو چندین درس را اخذ می‌کند و یک درس نیز توسط چندین دانشجو اخذ می‌شود.

نکته) در محیط‌های عملیاتی بیش از همه این رابطه به چشم می‌خورد.

● ارتباط نیز می‌تواند صفت یا صفاتی داشته باشد. از این نگاه ارتباط نیز نوعی موجودیت است.

مثال:



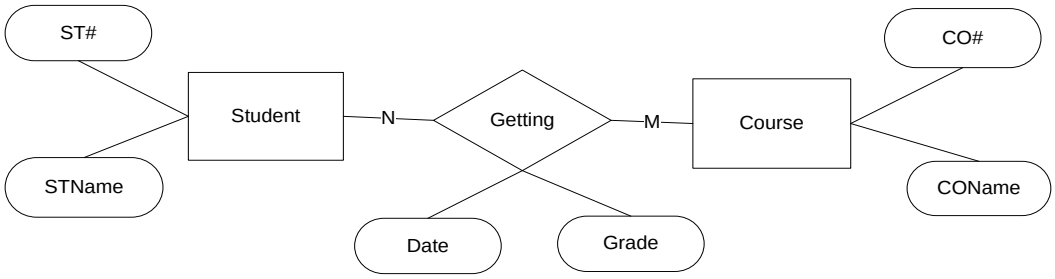
● مشارکت یک نوع موجودیت در یک رابطه ممکن است الزامی(کامل)باشد یا نباشد. مشارکت الزامی یا کامل زمانی است که تمام نمونه‌های موجودیت در آن ارتباط شرکت داشته باشند. ارتباط الزامی را به صورت═══ نشان می‌دهیم. در مثال قبل مشارکت الزامی نداریم.

یک مثال از مشارکت الزامی:

هر گروه آموزشی حتماً باید توسط یک استاد مدیریت شود اما الزاماً هر استاد مدیریت یک گروه آموزشی را بر عهده ندارد.

تبدیل مدل E/R به مدل رابطه‌ای:

حالت اول: دو موجودیت مستقل و ارتباط چند به چند



Student ($\underset{C.K}{\underline{ST\#}}$, STName , …)

Course ($\underset{C.K}{\underline{CO\#}}$, COName , …)

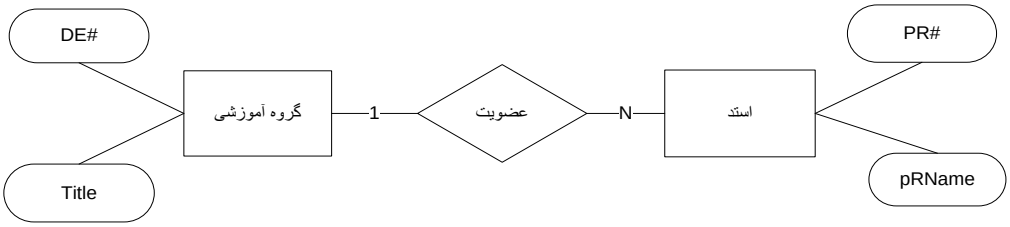
S-C ($\underset{F.K}{\underline{ST\#}}$, $\underset{F.K}{\underline{CO\#}}$, Date , Grade , …)

نکته: در این حالت تعداد رابطه‌ها برابر است با $n+1$ که n تعداد موجودیت‌هاست.

نکته: در نمایش ارتباط بین ۲ موجودیت، کلید کاندید رابطه حداقل از ترکیب کلیدهای خارجی بدست می‌آید. یعنی ممکن است صفت یا صفاتی از ارتباط هم در کلید کاندید به کار آید.

۱. دو موجودیت مستقل و ارتباط ۱ به N

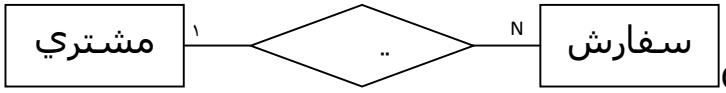
در این حالت موجودیت طرف یک را با یک رابطه و موجودیت طرف N و ارتباط را با هم با یک رابطه‌ی دیگر نمایش می‌دهیم.



DEPT ($\underset{C.K}{\underline{DE\#}}$, Title , …)

Prof ($\underset{C.K}{\underline{PR\#}}$, PRName , ... , $\underline{DE\#}$)

در رابطه‌ی دوم کلید کاندید رابطه‌ی اول به عنوان کلید خارجی حضور دارد و ارتباط مورد نظر را برقرار می‌کند.

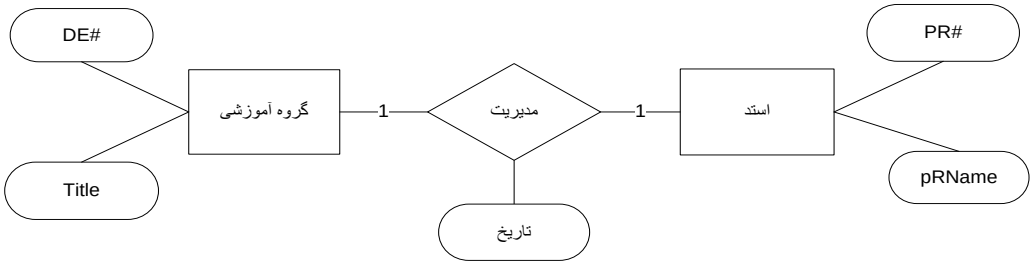


Customer ($\underline{CO\#}$, CName , Address,….)

Order ($\underset{C.K}{\underline{OR\#}}$, Date , … , $\underline{CO\#}$, تعداد محصول)

F.K کلید خارجي

حالت سوم: دو نوع موجودیت مستقل در ارتباط یک به یک



برای موجودیتی که سمت یک ارتباط است یک رابطه تعریف می‌کنیم و برای هر موجودیتی که در سمت مشارکت الزامی رابطه است یک رابطه‌ی دیگر تعریف خواهیم کرد. کلید کاندید رابطه‌ی اول در این رابطه کلید خارجی است.

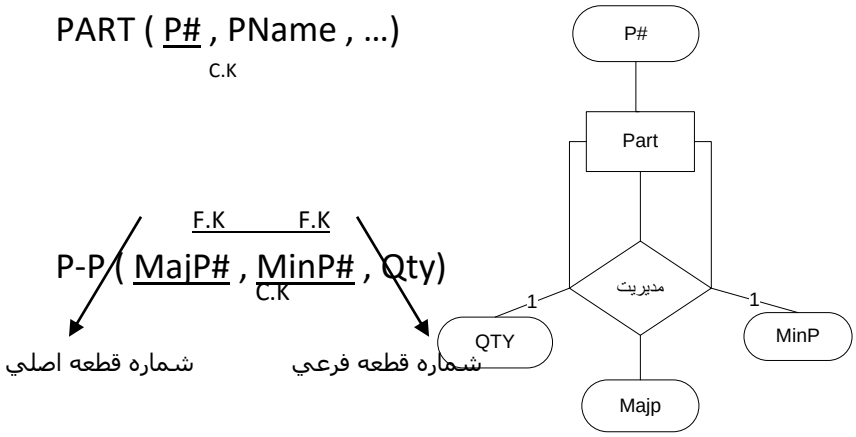
Prof (PR# , PRName , …)
C.K

DEPT (DE# , Title , … , PR#)
C.K F.K

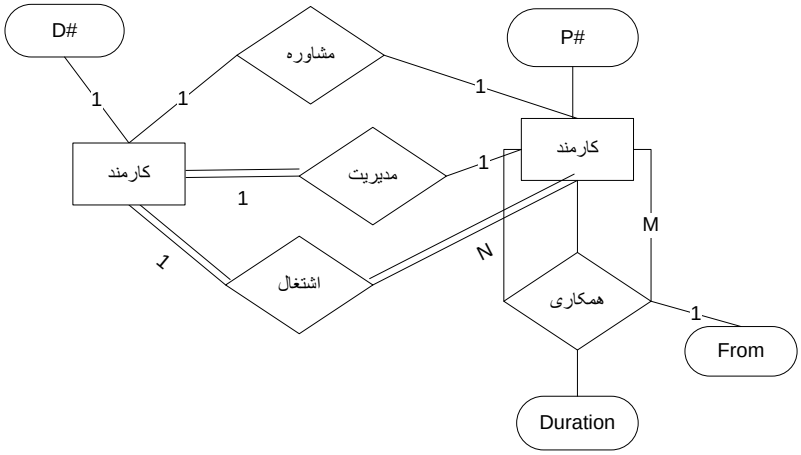
نکته) اگر رابطه‌ی یک به یک و یک به N ، خود نیز صفاتی داشت، صفات را به رابطه‌ای که شامل کلید خارجی است اضافه می‌کنیم.

حالت چهارم: یک نوع موجودیت داریم که با خودش رابطه‌ی چند به چند دارد.

مثال)مسأله رابطه‌ی قطعه و تولید کننده



با توجه به نمودار ER زیر، پایگاه داده‌های مربوط را طراحی کنید و سایر صفات هر ارتباط و هر موجودیت را متناسب انتخاب کنید.

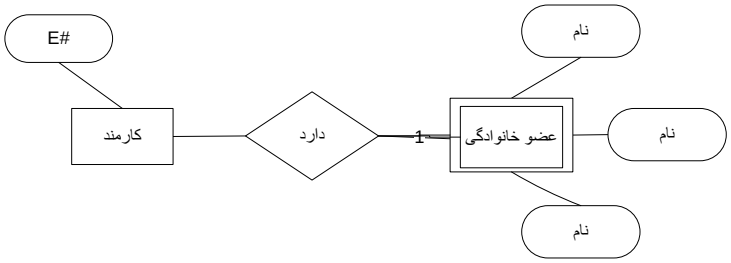


DEPT (D#, DName, Room#, …, E#)
C.K

EMPLOYEE(E#,EName,ELName,EFatherName, EAddress, ETel,…,D#)
C.K F.K

اگر رابطه یک به یک بود و یک طرف الزامی بود باید کلید کاندید موجودیت‌ها الزامی نیست را در موجودیتی که الزامی است بنویسیم که کلید خارجی می‌شود.

حالت پنجم: یک موجودیت داریم که با خودش ارتباط یک به N دارد این حالت با یک رابطه طراحی می‌شود.

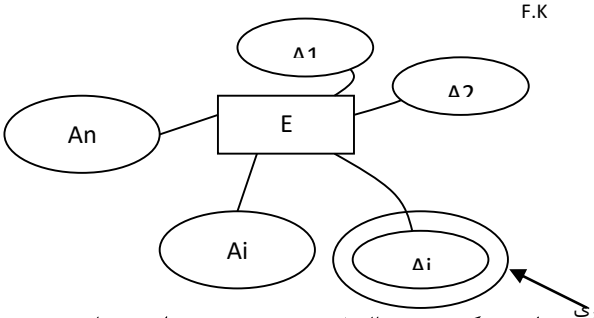


EMP(E#,BName,ELName,…,EManager#)
C.K F.K

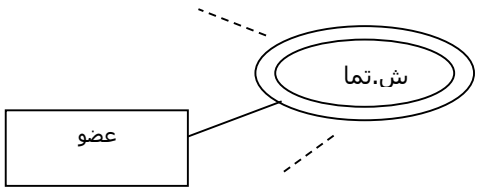
حالت ششم: ارتباط موجودیت قوی با موجودیت ضعیف: معمولاً ارتباط موجودیت قوی با موجودیت ضعیف ۱ به N است.در این حالت برای موجودیت قوی یک رابطه و برای موجودیت ضعیف و ارتباط آن (که می‌تواند صفت هم داشته باشد) یک رابطه در نظر می‌گیریم.

EMPL(E#,EName,ELName,Address,…)
C.K
MEM(MName,BirthDay,Gender,…,E#)
F.K

حالت هفتم: وجود صفت چند مقداری (اعم از ساده و مرکب)

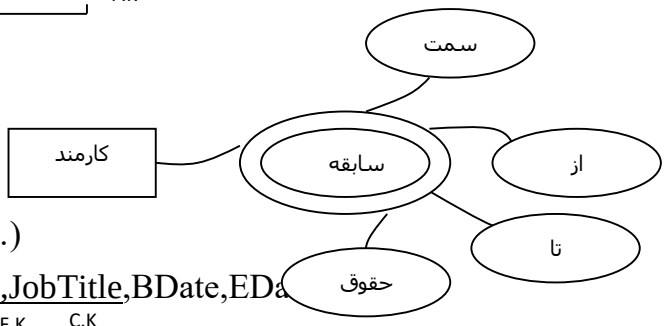


در این حالت دو رابطه یکی برای موجودیت و یکی برای صفت چند مقداری در نظر می‌گیریم. مثال۱: صفت چند مقداری ساده.



MEM(M#,MName,…)
C.K
Tel(TelNumber,Code,…,M#)
F.K
C.K

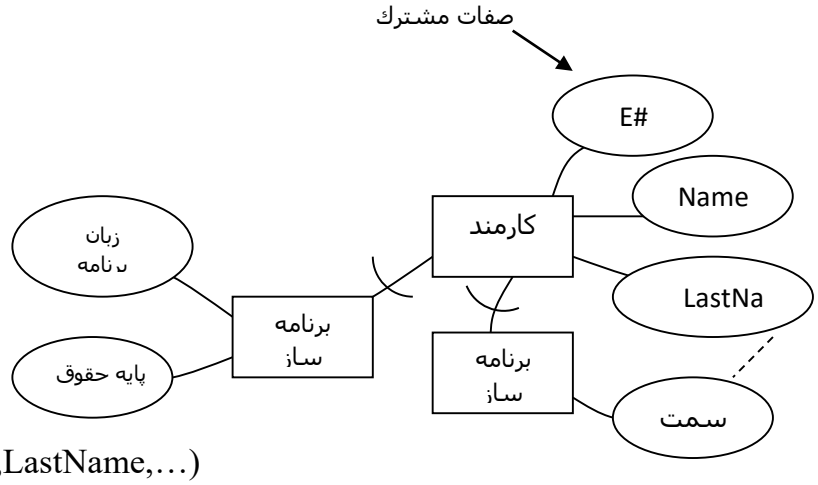
مثال ۲: صفت مرکب



EMP(E#,EName,…)
C.K
EMpjobHistory(E#,JobTitle,BDate,EDa
F.K C.K

اگر تعداد صفات چند مقداری یک موجودیت n باشد. N+1 رابطه باید طراحی کرد.

حالت هشتم: رابطه‌ی IS-A: اگر یک نمونه موجودیت n زیر موجودیت داشته باشد آنگاه N+1 رابطه طراحی می‌شود.



EMP(E#,Name,LastName,…)
C.K

Noprm(Title,…,E#)
F.K

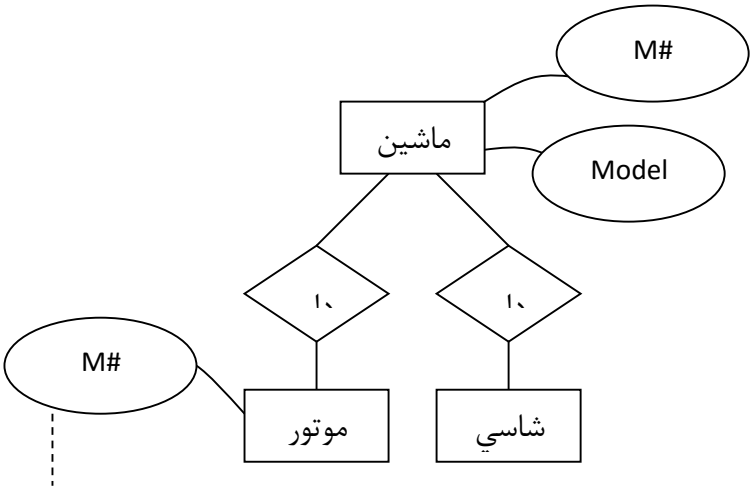
PRM(PRLanguage,Fee,…,E#)
F.K

حالت نهم:IS-A-PART-OF: در این حالت شیء جزء صفات کل را به ارث نمی برد برای طراحی روابط برای موجودیت کل یک رابطه و برای موجودیت جز یک رابطه‌ی دیگر طراحی می‌کنیم.

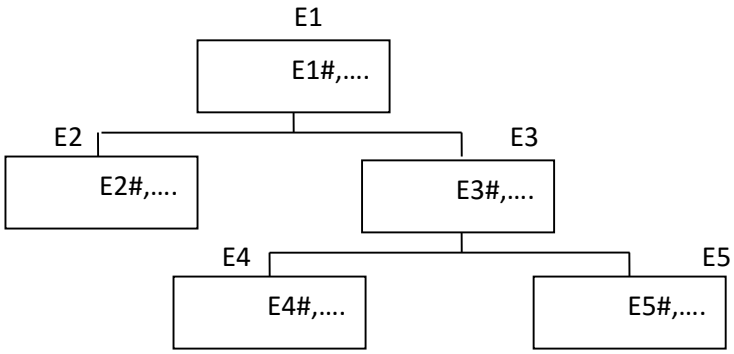
Car(M#,Modle,…)
C.K

Motor(M#,MT#,…)
C.K

Shasi(M#,SH#,…)
C.K



حالت دهم: ارتباط سلسله مراتبی بین انواع موجودیت:



E1(E1#,…)
C.K

E2(E2#,E1#,…)
F.K
C.K

E3(E3#,E1#,…)
F.K
C.K

E4(E4#,E3#,E1#,…)
F.K F.K
C.K

در این حالت برای هر موجودیت یک رابطه تعیین می‌کنیم.کلید رابطه، ترکیب شناسه‌ی موجودیت با شناسه‌ی موجودیت پدر آن موجودیت در نظر گرفته می‌شود.

مدل‌سازی داده‌ای Data Modeling:

عناصر مدل داده‌ای: موجودیت‌ها،صفات و روابط مدل داده‌ای در رابطه با پردازش داده‌ها در سیستم به سؤالات زیر پاسخ می‌دهد:

۱) چه موجودیت‌هایی(شیء داده‌ای) توسط سیستم پردازش می‌شود؟
۲) چه ویژگی‌هایی موجودیت را توصیف می‌کنند؟
۳) روابط بین موجودیت‌ها چیست؟

ERD(Entity Relationship Diagram) دیاگرام رابطه موجودیت:

- بازنمایی گرافیکی از مدل داده‌ای شامل موجودیت‌ها، ویژگی‌ها و روابط بین موجودیت‌ها.

نکته) نمودارهای ER جریان داده‌ها یا پردازش ورودی داده‌ها را نشان نمی‌دهد بلکه داده‌هایی که ذخیره می‌شوند را نشان می‌دهند.

سه روش برای تهیه یا استخراج مدل رابطه‌ای:

۱) روش بالا به پایین
۲) روش پایین به بالا
۳) استفاده از DFDها

- **توضیح روش بالا به پایین:**

با درک طبیعت سازمانی که سیستم در آن پیاده‌سازی می‌شود موجودیت‌ها استخراج می‌شوند. با کمک این قبیل پرسش‌ها می‌توان به موجودیت‌های سیستم دست یافت:

۱- چه افرا، مکان‌ها و چیزهایی در سازمان درباره آن‌ها داده جمع آوری می‌شود.
۲- چند نمونه از آن موجودیت یا شیء داده‌ای وجود دارد (می‌دانید که یک موجودیت دارای بیش از یک نمونه است)
۳- چه مشخصه‌هایی (ویژگی‌هایی) هر شیء و یا موجودیت را توصیف می‌کند (معمولاً یک موجودیت دارای چندین ویژگی است معمولاً موجودیت تک ویژگی نداریم)
۴- بر چه مبنایی اشیا، انتخاب، مرتب یا گروه بندی می‌شوند.
۵- چه مشخصاتی از هر شیء آن شیء را از دیگر اشیا همان نوع به شکل منحصر به فردی متمایز می‌کند.
۶- هر چیزی که می‌خواهیم راجع به آن اطلاعاتی را ذخیره یا نگهداری کنیم و برای نگهداری آن نیاز به حافظه داریم.

- **توضیح روش پایین به بالا:**

در این روش با مطالعه اسنادی که توسط سیستم پردازش می‌شود مثل فرم‌ها، گزارش‌ها، فرم‌ها و گزارشات کامپیوتری و... موجودیت‌ها استخراج می‌شوند.

- **توضیح روش استفاده از DFD:**

در این روش به سراغ تمامی دیتا استورهای DFDها می‌رویم. هر DS ساده خود یک موجودیت می‌باشد و از روی DSها لیست موجودیت‌ها را استخراج می‌کنیم سپس از روی آن‌ها صفات هر موجودیت و روابط بین موجودیت‌ها را تعیین می‌کنیم. بنابراین باید یک سازگاری بین مدل داده‌ای و مدل فرآیندی وجود داشته باشد و محدودیت‌های زیر در این مدل‌ها باید اعمال شده باشد.

۱) برای هر موجودیت در ERD باید یک DS در DFD وجود داشته باشد.
۲) برای هر DS در DFD باید یک موجودیت در ERD داشته باشیم. اگر محدودیت یک و دو برقرار نباشد یا DFD ناقص است یا ERD.

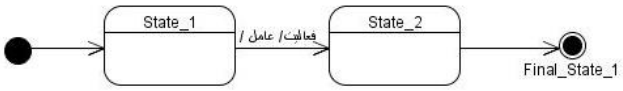
۳) برای هر ERD در DFD باید یک سری پردازه داشته باشیم که روی آن‌ها عملیات درج، حذف و بروز رسانی را انجام دهند.

نمودار حالت State Diagram:

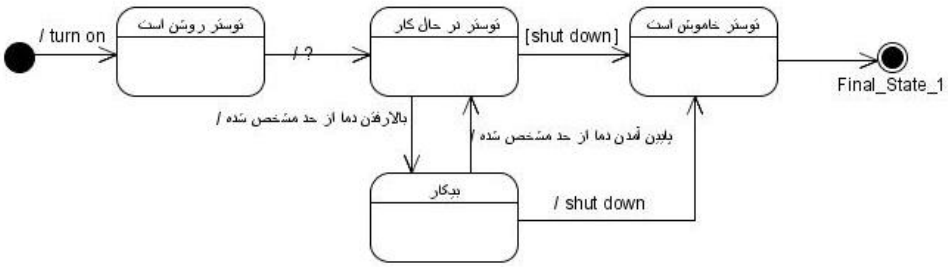
نام‌های دیگر : State Machine Diagram و State Chart Diagram

نمودار حالت رفتار(Behavior) سیستم را نمایش می‌دهد می‌توان توسط این دیاگرام عملکرد یک کلاس یا زیر سیستم یا کل سیستم را نمایش داد. در این دیاگرام این طور تصور می‌شود که سیستم توسط برخی Actionها که توسط Triggerها یا عامل‌ها اتفاق می‌افتند در Stateها یا حالات مختلف قرار می‌گیرند.

نحوه نمایش نمودار حالت:با یک مستطیل با گوشه‌های نرم نشان می‌دهند که نام وضعیت داخل آن نوشته می‌شود و یک پیکان که انتقال یا Transition را نشان می‌دهد به وضعیت بعد متصل می‌شود. شروع نمودار حالت با دایره توپر و حالت نهایی با ضربدر یا به صورت مشخص می‌شود.



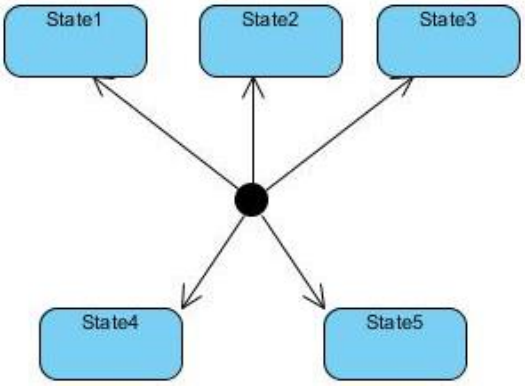
مثال) حالت توسعه نیافته (اولیه) پروسه تولد نان برشته.



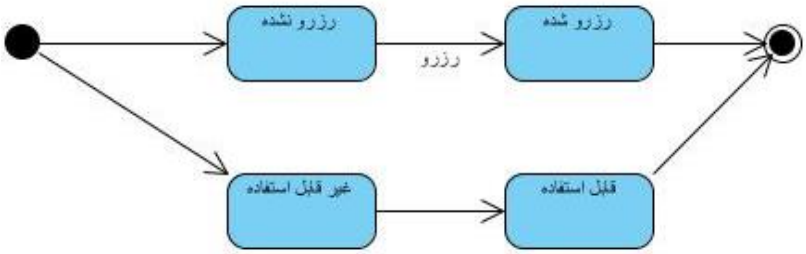
برخی دیگر از اجزای نمودار حالت:

• انشعاب Junction:

برای ادغام چند حالت به یک حالت یا تجزیه یک حالت به چند حالت.

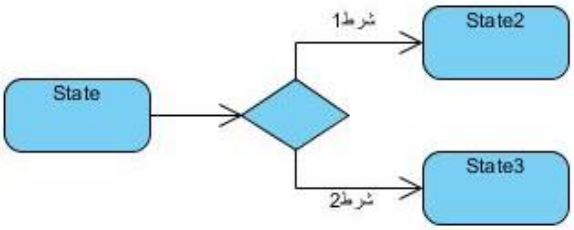


تمرین) نمودار مربوط به یک اتاق در سیستم هتل را رسم کنید.

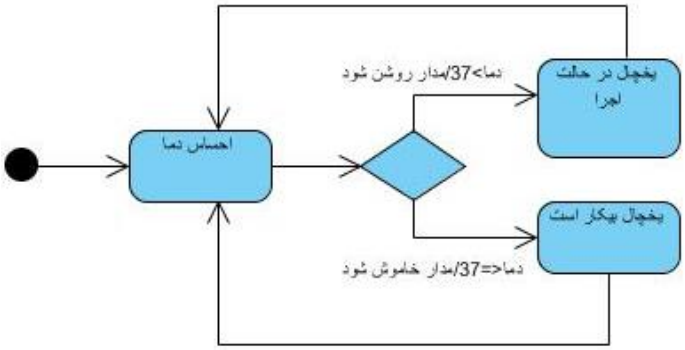


- **انتخاب یا Choice:**

شکل کلی :



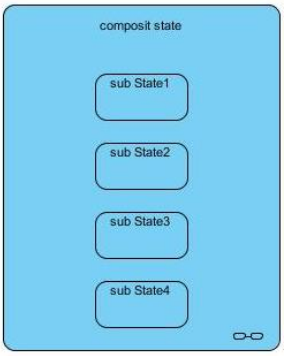
مثال) نمودار مربوط به نظارت بر دما در یخچال



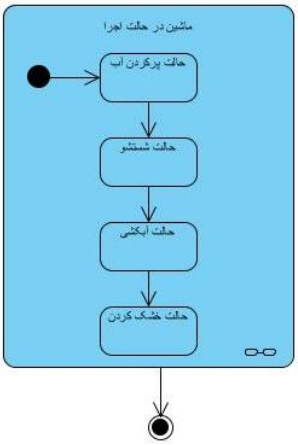
اگر قرار شد از یک حالت با توجه به یک شرط به دو یا چند حالت دیگر منتقل شویم (transition) از امکان choice (انتخاب) در UML استفاده می‌کنیم.

- **Composite state:**

یک حالت ممکن است دارای چندین حالت دیگر باشد به حالت کامل‌تر Composite state و به حالت‌های زیر مجموعه sub state گفته می‌شود.



مثال) نمودار حالت، حالت اجرا در ماشین لباس‌شویی

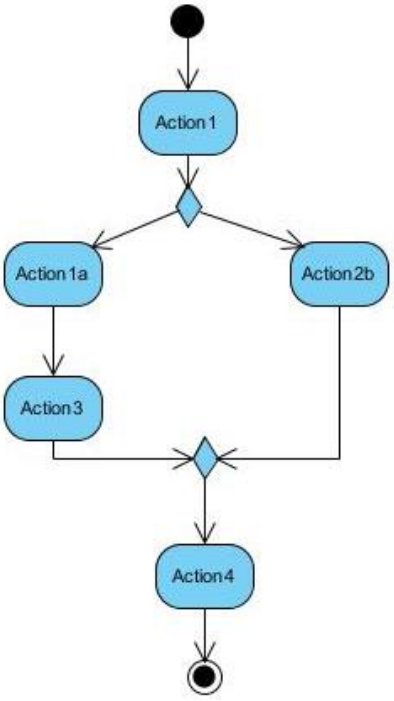


• نمودار فعالیت Activity Diagram :

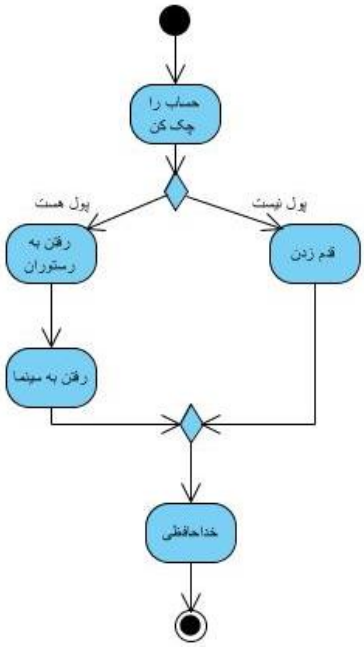
این دیاگرام روال پردازش یا عملیات در سیستم را به تصویر می‌کشد. دیاگرام فعالیت بسیار شبیه فلوچارت در برنامه نویسی است. می‌توان از این دیاگرام برای تشریح روال پردازش در یک کسب و کار Business Process یا software Process یا آنچه در یک یا چند Use Case اتفاق می‌افتد استفاده کرد.

نحوه نمایش: هر Action یا عملیات با یک مستطیل با گوشه‌های نرم نمایش داده می‌شود و پیکان‌ها مسیر جریان را مشخص می‌کنند.

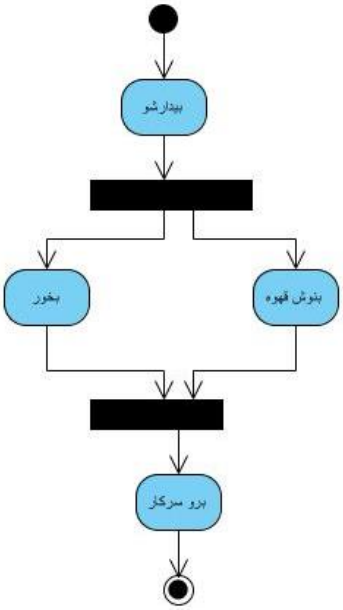
یک نمونه از شکل کلی نمودار فعالیت



مثال)



- **مسیرهای هم‌زمان:**

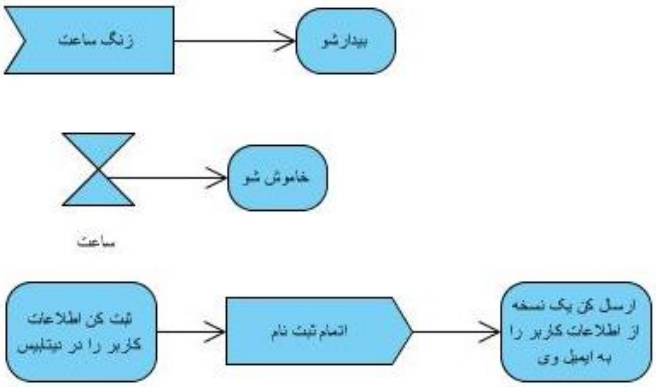


- **سیگنال Signal:**

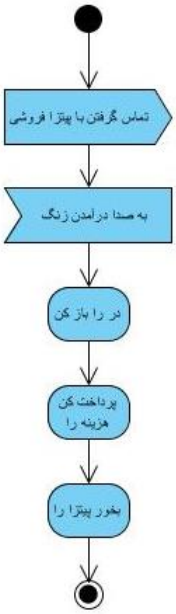
دو نوع Signal در UML داریم: ۱- Input signal ۲- Output signal

گاهی اوقات یک فعالیت یک پیغام ارسال یا دریافت می‌کند که به این پیغام در اصطلاح signal گفته می‌شود.

مثال) فعالیت بیدار شدن با سیگنال زنگ ساعت اتفاق بیفتد یا در برخی دستگاه‌ها توقف یا آغاز ساعت با سیگنالی از طرف یک ساعت اتفاق می‌افتد.چند مثال)



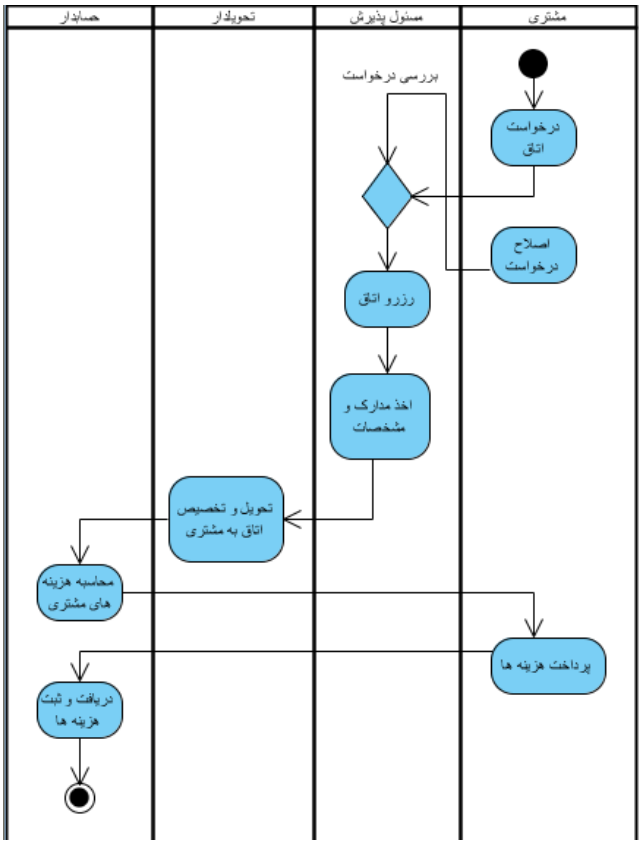
مثال) نمودار فعالیت تقاضای پیتزا



- **خط شنا Swim Lane:**

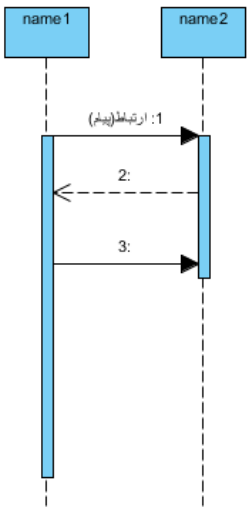
تا اینجا مشکل اصلی Activity Diagram این است که نمی‌توان فهمید چه کسی (who) مسئول انجام چه کاری (what) است. برای رفع این مشکل از خطوط شنا استفاده می‌شود.

مثال) Activity Diagram مربوط به رزرو اتاق در سیستم هتل

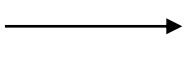


• نمودار توالی(Sequence Diagram):

نمودار توالی نشان می‌دهد که اشیاء با یکدیگر در طول زمان چگونه ارتباط دارند، نمودار حالت، حالات مختلف یک شیء را بررسی می‌کرد اما نمودار توالی روی ارتباط بین اشیاء در طول زمان تأکید دارد. منظور از زمان ترتیب اجرای عملیات است که در مدت اجرای آن‌ها، نمودار توالی یک نمودار دو بعدی است (2-Dimentional)که بعد عمومی زمان را نمایش می‌دهد و بعد افقی اشیا یا شرکت کنندگان (Participants) مختلف را نمایش می‌دهد.



انواع پیام:

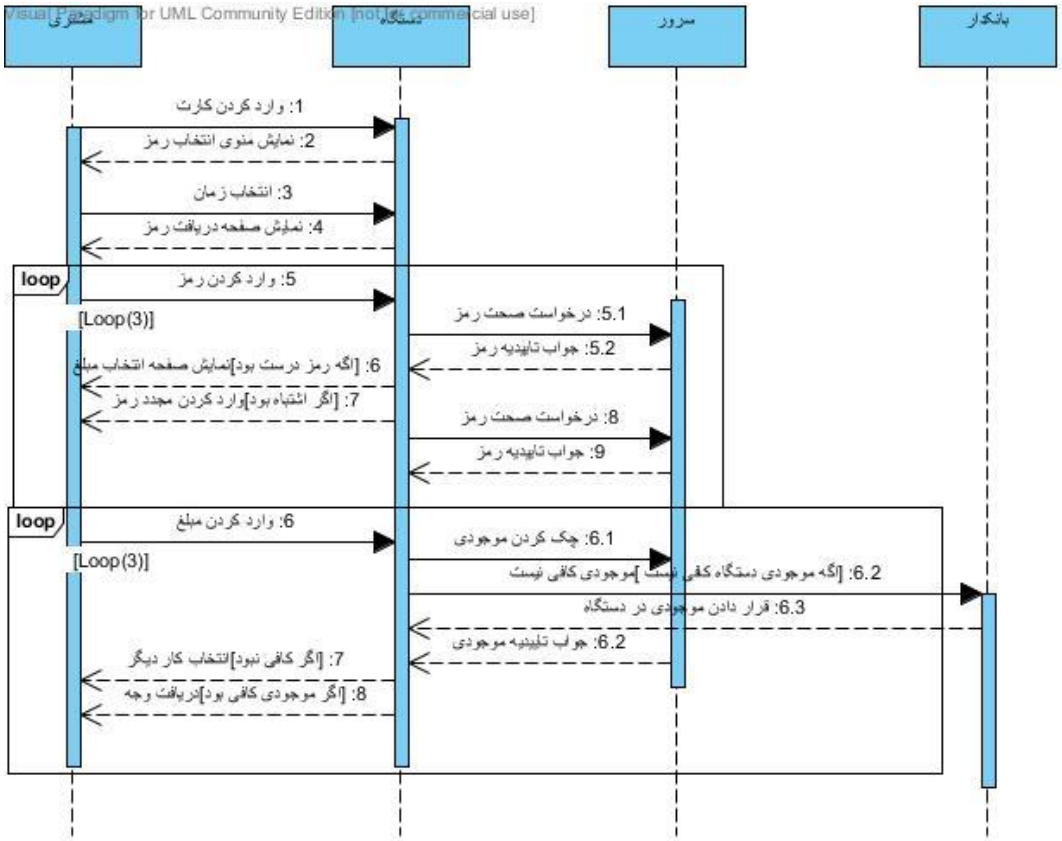


Synchronou همزمان
message

Return
برگشت
message

هرگاه یک شیء یک "پیام همزمان"به شیء دیگر ارسال کرد منتظر "پیام برگشت" می‌ماند پس هر پیام همزمان اجباراً یک پیام برگشت نیز خواهد داشت.

مثال) نمودار ترتیب برداشت وجه درATM



- **دیاگرام ارتباط یا تعامل Communication Diagram**:

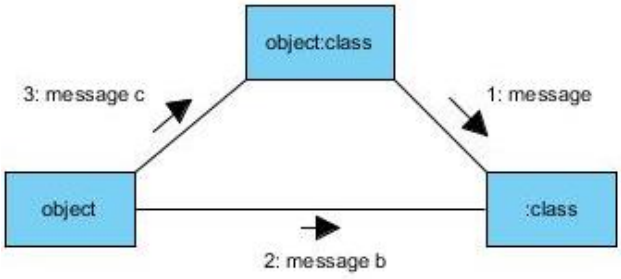
شامل سه عنصر است:

۱- Objects (اشیاء)
۲- Links یک خط ساده
۳- Messages پیکان‌های توپر روی لینک‌ها

این دیاگرام ارتباطات بین اشیاء (objects) یا شرکت کنندگان (Participant) را در سیستم به تصویر می‌کشد.

تأکید اصلی نمودار توالی بر روی زمان است در حالی که تأکید اصلی نمودار تعامل بر روی پیوند بین اشیاء است(Links).

در نمودار توالی شماره گذاری اجباری نیست چرا که ترتیب فعالیت‌ها از روی نمودار برداشت می‌شود اما درنمودار تعامل شماره گذاری اجباری است. یک نمونه از نمونه کلی دیاگرام ارتباطات



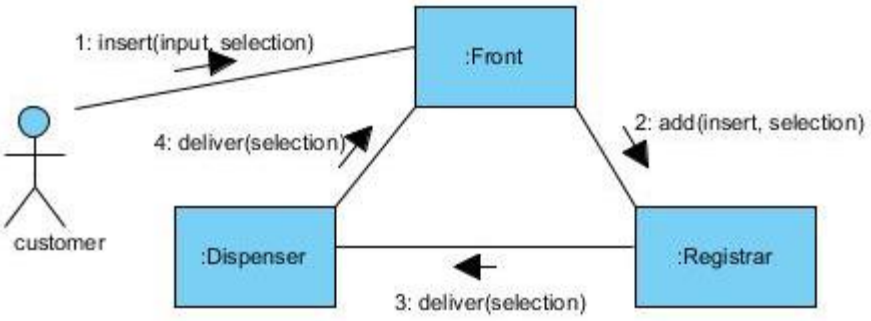
مثال) سیستم سلف-سرویس

سناریو:

۱- مشتری پول را در دستگاه می‌اندازد و یک یا چند محصول ارائه شده در دستگاه را انتخاب می‌کند.

۱- هنگامی که ثبت کننده پول (Registrar)، پول را دریافت می‌کند(در نظر بگیرید مشتری مقدار پول را درست قرار می‌دهد و همیشه یک محصول انتخاب شده وجود دارد) محصول انتخاب شده به توزیع کننده (Dispenser) تحویل داده می‌شود.

۲- توزیع کننده محصول را به جلو (Front) دستگاه تحویل می‌دهد و مشتری آن محصول را می‌گیرد.

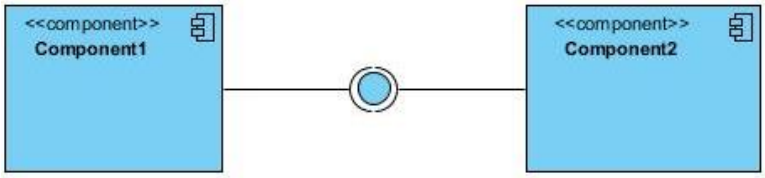


- **Component Diagram:**

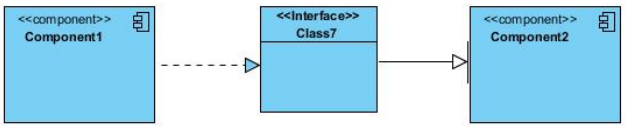
- A Physical view of your system. یک نمای فیزیکی از سیستم شما.

سه جزء اصلی این دیاگرام component(اجزا)، Interface(رابط‌ها)و ارتباط بین اجزا و روابط است. یک Component یک ماژول یا یک جزء قابل استفاده مجدد(Reusable) از سیستم است.

یک نمونه کلی از نحوه نمایش Component Diagram

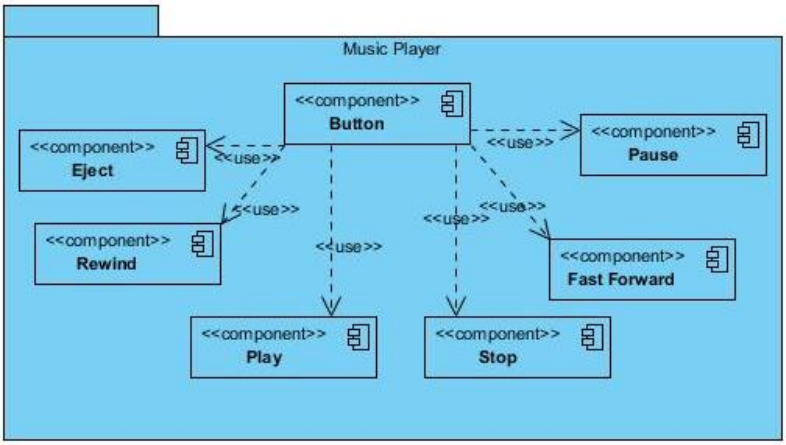


نوع دوم نمایش Interface:



مثال) فرض کنید که احتیاج داریم برای اجرای آهنگی از روی یک CD یک نرم‌افزار بسازیم برای این کار یک زبان برنامه نویسی ویژوال مثل ویژوال C#وVB یا Delphi می‌تواند مورد استفاده قرار گیرد. اگر این زبان از کنترل‌های مالتی مدیا پشتیبانی می‌کند می‌توانیم به سادگی از اجزای آن هرجا که لازم باشد استفاده کنیم یک سری Component طراحی و استفاده کنیم. یک ماکت گرافیکی از این برنامه می‌تواند به صورت زیر باشد .

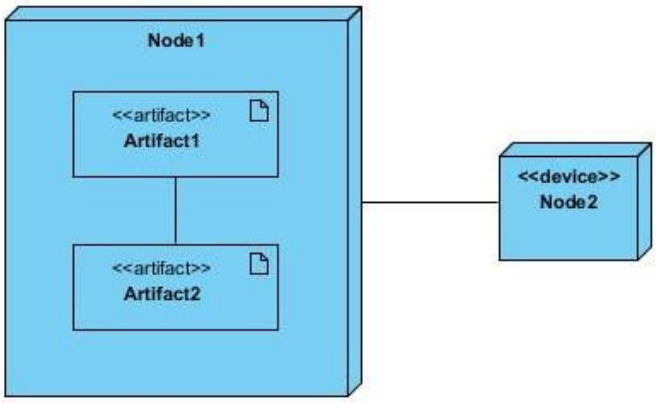
همان‌طور که می‌بینید این اجرا کننده آهنگ تعدادی کنترل نیاز دارد که هر یک با یک دکمه قابل طراحی است اگر به دکمه‌ها به عنوان اجزاء جدا از هم نگاه کنیم می‌توانیم یک دیاگرام اجزا به صورت زیر استخراج کنیم.



- **دیاگرام استقرار Deployment Diagram:**

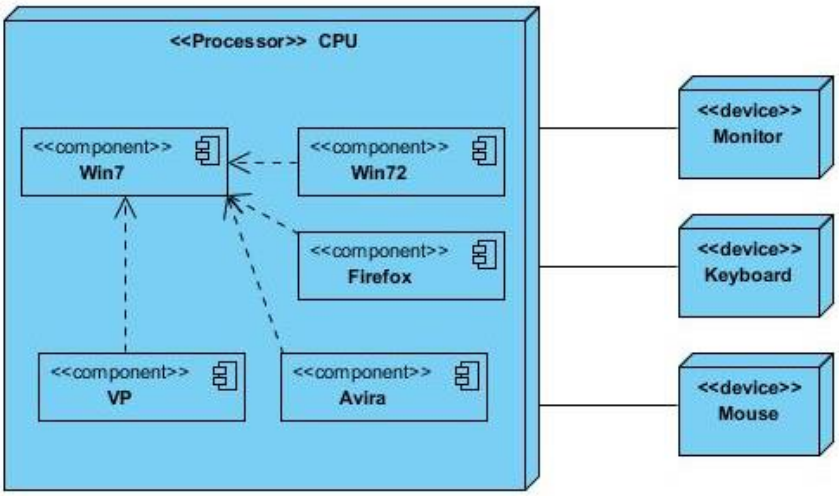
این نمودار نسبت بین اجزاء نرم‌افزاری سیستم و سخت افزاری که آن‌را اجرا خواهد کرد را نمایش می‌دهد .

مثال کلی از نحوه نمایش :



مثال) یک نمودار استقرار برای سیستم کامپیوتر خانگی

این سیستم از اجرای سخت افزاری cpu,monitor,mouse,keyboard و اجزای نرم‌افزاری windows7 , office 2010 و مرورگر وب fire fox و آنتی ویروس avira و visual paradigm داریم تشکیل شده است. دیاگرام استقرار این سیستم به شکل زیر است:


<<Processor>> CPU
<<component>>
Win7
<<component>>
Win72
<<component>>
Firefox
<<component>>
VP
<<component>>
Avira
<<device>>
Monitor
<<device>>
Keyboard
<<device>>
Mouse

RUP

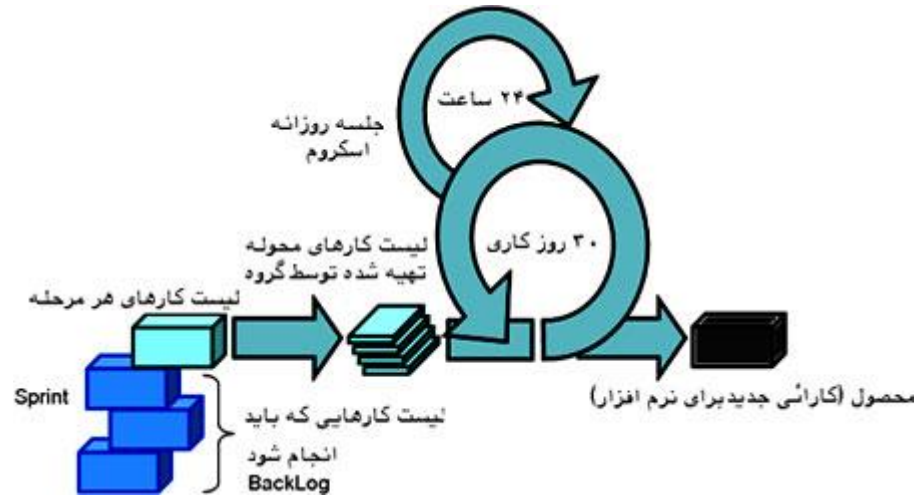
فرآیند تولید نرم‌افزار

پیروی از یک رویه منظم تولید نرم‌افزار به تولید کنندگان نرم‌افزار کمک می‌کند امور مربوط به تولید نرم‌افزار را منظم و پروژه را در حداقل زمان ممکن و با کارایی بالایی انجام دهند. در حقیقت یک رویه یا Process از مراحل مختلفی تشکیل شده است. این مراحل فعالیت‌های مربوط به رویه را مشخص می‌نمایند و تعیین می‌کنند که این فعالیت‌ها باید چگونه انجام شوند. پیروی از این مراحل به اعضای پروژه دریابند یاری می‌رساند که چه کاری را چه موقع و چگونه انجام دهند همچنین این کار میان اعضای گروه نیز هماهنگی به وجود می‌آورد. از آن جایی که منابع موجود و نیازهای کاربران هر نرم‌افزار با دیگری تفاوت دارد، فرایند تولید نرم‌افزارهای گوناگون نیز متفاوت است.

انجمن IEEE رویه یا فرایند تولید نرم‌افزار را این گونه تعریف می‌کند: رویه تولید نرم‌افزار در واقع شامل مراحلی مانند جمع‌آوری نیازهای کاربران ، طراحی سیستم با استفاده از تحلیل این نیازها و نوشتن کدهای نرم‌افزار با استفاده از طرح نرم‌افزار است. همچنین بر این باور است که از آن جایی که کیفیت و بهره‌وری نیروی کار با کیفیت روند تولید نرم‌افزار ارتباط مستقیم دارد، طراحی و مدیریت رویه تولید نرم‌افزار از اهمیت شایانی برخوردار است.

برای طراحی یک رویه تولید نرم‌افزار می‌توان از روش‌های متفاوتی استفاده نمود و از آن جایی که هر پروژه نرم‌افزاری با دیگر پروژه‌ها متفاوت است، می‌توان گفت که رویه تولید آن پروژه نیز با دیگر پروژه‌ها تفاوت دارد. در واقع می‌توان گفت: انتخاب این روش‌ها رابطه مستقیمی با اندازه گروه در پروژه دارد و نرم‌افزارهای بزرگ و کوچک نیاز به رویه‌های تولید متفاوت دارند.

روش Scrum



امروزه یکی از روش‌های تولید نرم‌افزار که به خصوص برای پروژه‌های نرم‌افزاری کوچک مورد استفاده قرار می‌گیرد و توسط بسیاری از اساتید و صاحب‌نظران مورد تأیید قرار گرفته است، روش Scrum است. با استفاده از این روش که روشی به اصطلاح (iterative تکراری یا چرخشی) می‌باشد، می‌توان نرم‌افزارهای کوچک را با کیفیت بالا تهیه نمود. در این روش که به روش هوشمند یا Agile نیز مشهور است، مدیریت قوی تولید نرم‌افزار وجود دارد که به برنامه‌نویسان اجازه می‌دهد با استفاده از آن در پروژه‌ها به سرعت نرم‌افزار مورد نظر را تهیه نمایند. اسم Scrum در حقیقت از بازی راگبی گرفته شده است (در بازی راگبی Scrum تیمی متشکل از هشت نفر است که با همکاری بسیار نزدیک با یکدیگر بازی می‌کنند).

روش XP

اشتباه نکنید! منظور از روش اکس‌پی، ویندوز اکس‌پی نیست. اکس‌پی مخفف Extreme Programming یا برنامه‌نویسی سریع می‌باشد که مانند Scrum روشی هوشمند در تولید نرم‌افزار است. در اکس‌پی دو برنامه‌نویس کار را انجام می‌دهند و قبل از اتمام

برنامه آن را چندین بار امتحان می‌کنند. اکس‌پی در حقیقت روشی از تولید نرم‌افزار است که بر اساس آسانی، ارتباط، واکنش و تصمیم‌گیری سریع استوار است.

روش RUP یا Rational Unified Process

در این بخش یکی از معروف‌ترین رویه‌های تولید نرم‌افزار که توسط شرکت آی‌بی‌ام طراحی گردیده است را معرفی می‌کنیم. این روش با نام (onal Unified ProcessRati) RUP در بسیاری از پروژه‌های نرم‌افزاری به کار گرفته می‌شود.

در حقیقت هدف اصلی RUP مطمئن شدن از این موضوع مهم است که آیا نرم‌افزار تولیدشده نیازهای کاربرانش را به صورت کامل، با کیفیت بالا، در زمان معین و با بودجه مشخص برآورده کرده است یا خیر.

مطابق تحقیقات انجام شده، از آن جایی که RUP به تمامی اعضای تیم، اطلاعاتی مشترک می‌دهد و تمامی اعضای گروه با یک زبان مشترک با هم مرتبط هستند، بازده کاری گروه را بالا می‌برد.

به گزارش رویتر در سال ۲۰۰۱ میلادی بیش از ششصد هزار شرکت تولید کننده نرم‌افزار، از ابزارهای شرکت Rational استفاده می‌کرده‌اند که این تعداد کماکان هم در حال افزایش است. این متدلوژی، برای انواع پروژه‌های نرم‌افزاری در دامنه‌های مختلف (مانند سیستم‌های اطلاعاتی، سیستم‌های صنعتی، سیستم‌های بلادرنگ، سیستم‌های تعبیه شده، ارتباطات راه دور، سیستم‌های نظامی و ...) و در اندازه‌های متفاوت، از پروژه‌های بسیار کوچک (یک نفر در یک هفته) تا پروژه‌های بسیار بزرگ (چند صد نفر تولید کننده با پراکندگی جغرافیایی)، کاربرد دارد.

RUP دارای سه جزء اصلی است. جزء اول از مجموع راه‌حل‌های خوب که در رویه می‌تواند مورد استفاده قرار گیرد تشکیل شده است. جزء دوم همان مراحل تهیه نرم‌افزار است و جزء آخر قسمت‌های تشکیل‌دهنده این رویه می‌باشد.

RUP شش راه حل خوب که می‌تواند در مراحل مختلف این رویه به ما کمک کند را معرفی کرده است که عبارتند از:

۱- استفاده از USE CASEها که می‌توانند در جمع آوری نیازهای کاربران مفید باشند.

۲- استفاده از معماری نرم‌افزار قابل استفاده مجدد (component reuse)

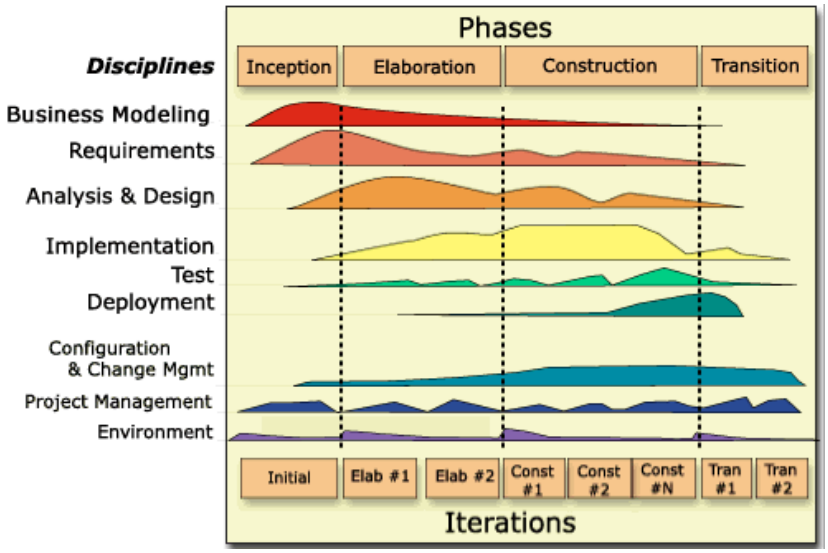
۳- استفاده از روش‌های تکمیلی و Iterative برای کنترل بهتر و آسان پروژه نرم‌افزاری

۴- استفاده از نمودارهای UML

۵- کنترل تغییرات در نرم‌افزار

۶- کنترل کیفیت نرم‌افزار با توجه به درخواست‌های اولیه کاربران

شکل ۳ رویه RUP را نمایش می‌دهد. همان‌طور که در این شکل مشخص شده است چرخه تولید نرم‌افزار به چهار قسمت اصلی تقسیم شده است:



الف: Inception phase یا مرحله آغازین:

- بدست آوردن محدوده نرم‌افزاری پروژه و محدودیت‌های آن که شامل یک دید عملیاتی، معیار پذیرش و اینکه چه چیز باید در محصول باشد و چه چیز نباید باشد، می‌شود

- مشخص کردن Case-Useهای اساسی سیستم، سناریوهای اصلی عملیات که مسائل مربوط به طراحی اصلی را ایجاد می‌کند.

- نمایش و شاید توضیح حداقل یک معماری کاندیدا برای بعضی سناریوهای اصلی

- برآورد هزینه و زمان کلی برای کل پروژه

ب: Elaboration phase یا مرحله مقدماتی:

- اطمینان از اینکه معماری، نیازمندی‌ها و طرح‌ها به اندازه‌ی کافی پایدارند و ریسک‌ها به اندازه‌ی کافی کاهش یافته‌اند بطوریکه بتوان هزینه و زمان‌بندی لازم برای تکمیل تولید را پیش‌بینی کرد. برای اکثر پروژه‌ها، گذر از این مرحله‌ی مهم مانند انتقال از یک عملیات سبک و سریع و با ریسک پایین به یک عملیات با هزینه و ریسک بالا همراه با اجبار سازمانی است.

- بیان همه‌ی ریسک‌های پروژه که از نظر ساختاری اهمیت دارند.

- ایجاد یک معماری پایه، مشتق شده از سناریوهای مهم که از لحاظ ساختاری اهمیت دارند، که این معماری ریسک‌های فنی عمده پروژه را نیز مشخص می‌کند.

- تولید یک نمونه‌ی اولیه‌ی تکاملی از مؤلفه‌های با کیفیت تولیدی خوب، و همچنین یک یا چند نمونه‌ی اولیه‌ی اکتشافی و نمونه‌های اولیه‌ی غیر قابل استفاده جهت کاهش ریسک‌های خاص مانند :

سازش‌های مربوط به نیازمندی‌ها یا طراحی

استفاده‌ی مجدد از مؤلفه‌ها

عملی بودن محصول یا توضیحات برای سرمایه گذاران، مشتریان و کاربران نهایی

- توضیح اینکه معماری پایه از نیازمندی‌های سیستم با هزینه‌ی منطقی و در زمان منطقی پشتیبانی می‌کند

- ایجاد یک محیط پشتیبانی کننده

ج: Construction phase یا مرحله ساخت و توسعه:

- کمینه کردن هزینه‌های تولید با بهینه‌سازی منابع و پرهیز از دور انداختن و دوباره‌کاری غیر ضروری
- دست‌یابی هرچه سریع‌تر به کیفیت کافی
- دست‌یابی هر جه سریع‌تر به ویرایش‌های مفید (آلفا، بتا و سایر نسخه‌های تست)
- کامل کردن تحلیل، طراحی، تولید و تست کارآیی مورد نیاز
- تولید تکراری و گام به گام یک محصول کامل که آماده‌ی انتقال به محیط کاربران باشد
- تصمیم در مورد اینکه آیا نرم‌افزار، سایت‌ها و کاربران همه برای استقرار طرح آمادگی دارند
- دست‌یابی به میزانی از موازی سازی در کار تیم‌های تولید.

د: Transition phase یا مرحله تغییرات:

- تست بتا برای تشخیص اعتبار سیستم جدید با توجه به انتظارات کاربر
- تست بتا و عملیات موازی همراه با یک سیستم قدیمی که در حال جایگزینی می‌باشد.
- تبدیل پایگاه‌های داده‌ی عملیاتی
- آموزش کاربران و نگهداری کنندگان
- بازاریابی، توزیع و فروش برای نخستین انتشار محصول
- تنظیم فعالیت‌ها از قبیل رفع اشکال، افزایش کارایی و قابلیت استفاده
- ارزیابی خط مبناهای استقرار در مقایسه با تصویر کلی و معیار قابلیت قابل قبول برای محصول
- دست‌یابی به موافقت ذی‌نفع در مورد اینکه خط مبناهای استقرار کامل می‌باشند
- دست‌یابی به موافقت ذی‌نفع در مورد اینکه خط مبناهای استقرار با معیار ارزیابی تصویر کلی سازگارند.

همان‌طور که در این قسمت ذکر شد، روش RUP روشی انعطاف پذیر، قابل تغییر و پیشرفته است که می‌تواند در صورت استفاده صحیح، باعث افزایش کارایی و کیفیت نرم‌افزار تولیدی گردد. اما آیا RUP می‌تواند رویه خوبی برای تولید نرم‌افزارهای کوچک باشد؟ در جواب باید گفت که RUP را طوری طراحی کرده‌اند که بتواند برای انواع پروژه‌های نرم‌افزاری در هر اندازه مفید باشد و از آن جایی که از ابزارهای خوبی مثل UML نیز استفاده می‌کند، UML) در گروه‌های کوچک که نرم‌افزارهای کوچک طراحی می‌کنند ابزار مدلی خوبی است) می‌تواند باعث همکاری و هماهنگی بیشتر گروه گردد.

دیسیپلین‌های RUP

دیسیپلین مجموعه‌ای از کارهای به هم مرتبطی است که برای انجام جنبه خاصی از یک پروژه انجام می‌شوند. متدلوژی RUP دارای ۶ دیسیپلین اصلی (مربوط به تولید محصول) و ۳ دیسیپلین کمکی (مربوط به تیم و محیط تولید) است که در ادامه به ترتیب معرفی خواهند شد.

دیسیپلین‌های اصلی:

Business Modeling (مدل‌سازی کسب و کار)

Requirements (نیازمندی‌ها)

Analysis & Design (تحلیل و طراحی)

Implementation (پیاده‌سازی)

Test (آزمون)

Deployment (استقرار)

دیسیپلین‌های کمکی:

Environment (محیط)

Project Management (مدیریت پروژه)

Configuration & Change Management (مدیریت پیکربندی و تغییرات)

تشریح دیسیپلین‌های اصلی:

۱- Business Modeling (مدل‌سازی کسب و کار)

- شناخت ساختار و دینامیک‌های سازمانی که در آن یک سیستم باید استقرار یابد(سازمان هدف).
- شناخت مشکلات فعلی در سازمان هدف و تشخیص پتانسیل‌های بهبود
- تضمین اینکه مشتری، کاربر نهایی و تولید کنندگان یک شناخت مشترک از سازمان هدف دارند.
- هدایت نیازمندی‌های سیستم که برای حمایت از سازمان هدف مورد نیازند.
- دیسیپلین مدل‌سازی کسب و کار توضیح می‌دهد که برای رسیدن به این هدف چگونه می‌توان یک تصویر کلی از سازمان را تولید نمود، و بر اساس این تصویر کلی فرآیندها، نقش‌ها و مسؤولیت‌های آن سازمان را در یک مدل case-Use کسب و کار و یک مدل شیء کسب و کار تعریف کرد.

۲- Requirements (نیازمندی‌ها)

- تشخیص و نگهداری موارد توافق با مشتری‌ها و سایر ذینفعان در مورد کارهایی که سیستم باید انجام دهد.
- فراهم آوردن شناخت بهتر از نیازمندی‌های سیستم برای تولید کنندگان سیستم
- تعریف مرزهای و حدود سیستم
- فراهم کردن یک پایه برای طرح ریزی مفاهیم تکنیکی تکرارها
- فراهم کردن یک پایه برای تخمین مخارج و زمان تولید سیستم
- تعریف یک واسط کاربر برای سیستم با تمرکز بر روی نیازها و اهداف کاربران

۳- Analysis & Design (تحلیل و طراحی)

- تبدیل نیازمندی‌ها به طراحی سیستم که قرار است به وجود آید.
- پیدایش یک معماری مستحکم برای سیستم
- سازگار ساختن طراحی برای هماهنگ شدن با محیط پیاده‌سازی و طراحی آن برای کارایی بهتر

۴- Implementation (پیاده‌سازی)

- تعریف ساختمان کدها برای کد نویسی، بر حسب زیر مجموعه‌ای از مجموعه‌های پیاده‌سازی سازمان یافته در لایه‌ها
- پیاده‌سازی کلاس‌ها و اشیاء به وسیله مؤلفه‌ها (فایل‌های منبع، باینری‌ها، فایل‌های اجرایی و ...)
- تست اجزاء تولید شده به عنوان واحدها
- مجتمع‌سازی نتایج تولید شده توسط پیاده سازان فردی (یا تیم‌ها) به صورت یک سیستم قابل اجرا.

۵- Test (آزمون)

- یافتن و مستند کردن نقایص در کیفیت نرم‌افزار
- آگاهی دادن در مورد کیفیت نرم‌افزار بررسی شده
- اثبات اعتبار فرضیاتی که در طراحی و مشخصات نیازمندی‌ها ساخته شدند، از طریق نمایش‌های واقعی

- تصدیق عملکردهای محصول نرم‌افزار همان‌طور که طراحی شده است.
- تصدیق اینکه نیازمندی‌ها به درستی پیاده‌سازی شده‌اند.

۶- Deployment (**استقرار**)

- نصب اختصاصی
- آماده فروش کردن محصول نهایی
- دست‌یابی به نرم‌افزار از طریق اینترنت

تشریح دیسیپلین‌های کمکی:

۷- Environment (**محیط**)

- دیسیپلین محیط بر فعالیت‌هایی که برای پیکربندی فرآیند برای یک پروژه لازم و ضروری‌اند، متمرکز می‌شود. این دیسیپلین فعالیت‌های مورد نیاز برای تولید رهنمودهایی که در جهت پشتیبانی از یک پروژه لازم می‌باشند را توضیح می‌دهد. هدف فعالیت‌هایی محیطی فراهم آوردن محیط تولید (هم فرآیندها و هم ابزاری که تیم تولید را پشتیبانی می‌کنند) برای سازمان تولید کننده نرم‌افزار می‌باشد. جعبه ابزار مهندس فرآیند پشتیبانی ابزاری را برای پیکربندی یک فرآیند فراهم می‌کند. این مورد شامل ابزارها و نمونه‌هایی برای ایجاد سایت‌های وب پروژه و سازمان بر اساس RUP می‌شود.

۸- Project Management (**مدیریت پروژه**)

- فراهم کردن یک چارچوب برای مدیریت پروژه‌های صرفاً نرم‌افزاری
- فراهم کردن رهنمودهای عملی برای طرح‌ریزی، تعیین نیروی انسانی، اجرا و نظارت بر پروژه‌ها
- فراهم کردن یک چارچوب برای مدیریت ریسک
- با این وجود، این دیسیپلین از RUP برای پوشش دادن همه‌ی جنبه‌های مدیریت پروژه نیست. برای مثال این دیسیپلین موارد زیر را پوشش نمی‌دهد :
  - مدیریت افراد: استخدام، آموزش، رهبری
  - مدیریت بودجه: تعیین، تخصیص و غیره
  - مدیریت قراردادها: با پشتیبانی کنندگان و مشتریان
- این دیسیپلین به طور عمده روی جنبه‌های مهم یک فرآیند تکراری تمرکز می‌کند که عبارتند از :
  - مدیریت ریسک
  - طرح ریزی برای یک پروژه‌ی تکراری، از طریق چرخه‌ی حیات و برای یک تکرار به خصوص
  - نظارت بر پیشرفت یک پروژه‌ی تکراری و متریک‌ها

۹- Configuration & Change Management (**مدیریت پیکربندی و تغییرات**)

- تشخیص موارد پیکربندی
- محدود کردن تغییرات آن موارد
- رسیدگی به تغییراتی که برای آن موارد ساخته شده
- تعریف و مدیریت پیکربندی آن موارد