درس مهندسی نرم افزار(1)
بخش اول

گردآوری و تدوین:دکتر افشین سلاجقه

مراجع :

1-Software engineering a practitioner approach by pressman 1997-2000

2-The unified software development process by Ivar Jacobson,Grady Booch ,James Rumbaugh.

3- Documents of Rational Rose 2002 or later

4-UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition, Martin Fowler, Addison-Wesley, September 2003

5-Unified Modeling Language User Guide, by Grady Booch, James Rumbaugh, Ivar Jacobson,
The 2nd Edition, Addison-Wesley, May 19, 2005

6- omg.org

7-Software Development for Small Teams: A RUP-Centric Approach, By Gary Pollice, Liz Augustine, Chris Lowe, Jas Madhur, Addison Wesley, January 02,2004

8- The Rational Unified Process Made Easy: A Practitioner's Guide to Rational Unified Process,
By Per Kroll, Philippe Kruchten, Addison-Wesley, April 2003

9-UML Schumes outlines, by Simon bennet ,John Skelton,ken Lumn.

10-Software Architecture in practice by Len Bas,Paul clements,Rick Kazman,Addison Wesley,1998.

11-Refrences about SSADM methodology and Select casetools

مهندسی نرم افزار

مهندسی نرم افزار

- تعریف(1) : بکارگیری اصول و قوانین مهندسی درایجاد نرم افزارهای کارامد ،مطمئن و با کیفیت .

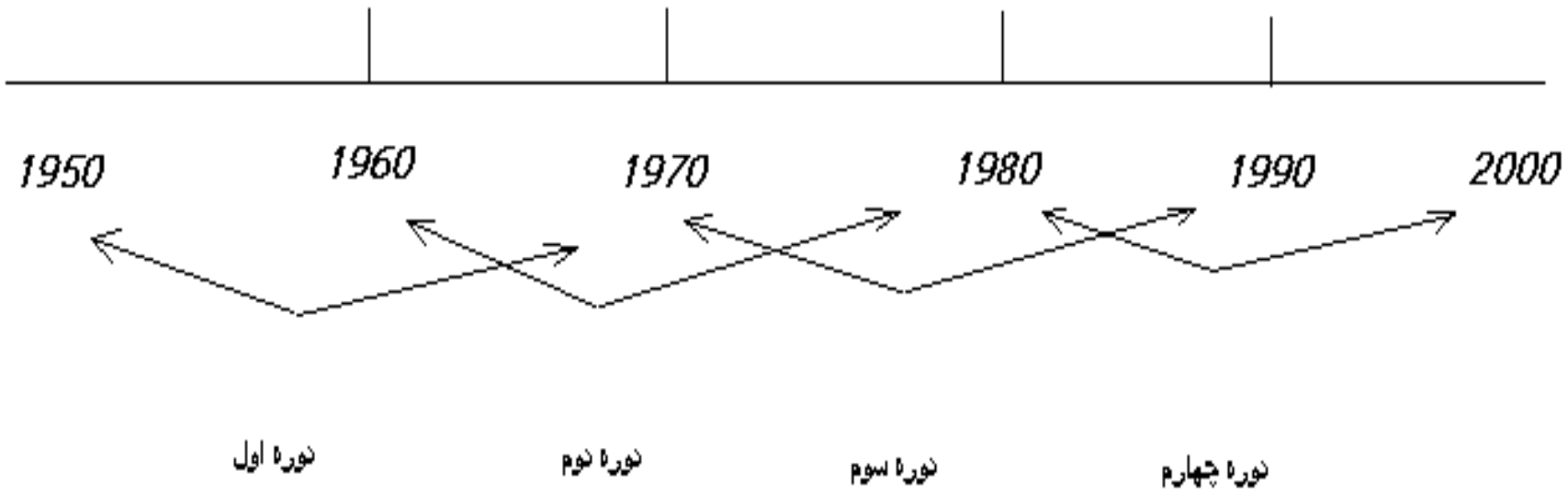
- تعریف(2) : رشته ای است که طراحی و ساخت و نگهداشت نرم افزار های قابل اطمینان را در بر دارد .

بحران نرم افزاری و مهندسی نرم افزاری

1- افزايش زمان و هزينه توليد نرم افزار و غيرقابل پيش بيني بودن آنها .
2- نرم افزارهاي توليدشده همة كارهاي موردنظر را انجام نمي دهند.
3- نرم افزارهاي توليد شده داراي خطا هستند .
4- نگهداري نرم افزارها بخاطر طراحي ضعيفيشان مشكل است.
5- پيچيدگي و قدرت سخت افزارها باعث شده است كه نوشتن نرم افزارهايي كه قادر باشند از اين پتانسيل سخت افزاري استفاده كنند، مشكل گردد.
6- توان ما براي توليد نرم افزارهاي جديد به اندازه تقاضا نيست.

براي مقابله با بحران نرم افزار بود كه مهندسي نرم افزار ايجاد شد.

دوره هاي تكاملي مهندسي نرم افزار



دوره اول

- اين دوره از اوايل دهه 50 ميلادي تا اواسط دهه 60 مي باشد و داراي خصوصيات زير است:

1-*Batch Processing*

در واقع معرف پردازش دسته اي و غير محاوره اي برنامه ها مي باشد، كه در آن برنامه ها در يك زمان دلخواه به داخل كامپيوتر وارد شده و در يك زمان دلخواه ديگر اجرا مي شود .

2-*Limited Distribution(Centralize)*

در اين دوره توزيع شدگي وجود نداشت. امروزه عمده سيستمها داراي ماهيت توزيع شدگي هستند. اما در گذشته mainframe هايي بودند كه باترمينهايي به آنها وصل شده و كار را انجام مي داديم.

3-*In House Software*

در اين دوره كمپاني هايي نبود كه نرم افزار توليد كنند و نرم افزار را منطبق با نياز خاص سازمان توليد كنند (اصطلاحاً خانه هاي نرم افزار وجود نداشت) اگر شركتي نياز به نرم افزار داشت، خودش افرادي را استخدام مي كرده و نرم افزار را توليد مي كرد، در حقيقت اين نرم افزارها، نرم افزارهايي هستند كه در خود سازمان ساخته مي شوند و فقط براي همان سازماني كه ساخته شده اند جوابگو مي باشند. اين نرم افزارها داراي *Document* نمي باشند و اطلاعات آنها فقط در مغز شخص طراح مي باشد .

دوره دوم

اين دوره از اواسط دهه 60 تا اوايل دهه 70 بوده است . خصوصيات اين دوره بشرح زير است :

1-*Real Time System*

اين گونه سيستمها، سيستم هايي هستند كه لازم است در يك زمان مشخص به ما جواب بدهند و اگر جواب ندهند، فاجعه رخ مي دهد.

2-*Data Base*

نرم افزارهاي پايگاه داده ها هستند كه اولين بار در اين زمان پا به عرصه هستی نهادند..

3-Multi User Software

تا قبل از اين دوره عمده نرم افزارها تك كاربره بودند و در اين زمان نرم افزارهاي چندكاربره بوجود آمدند كه مسلماً شيوه طراحي آنها نيز متفاوت بوده است.

4-Product Software

در اين دوره بعلت افزايش استفاده كامپيوتر نسبت به دوره هاي قبلي سازمانهايي به نام خانه نرم افزار ايجاد شد كه براي شركتهاي مختلف نرم افزار توليد مي كردند.

اين نرم افزارها گاها براي بيش از يك سازمان توليد می شد و در خارج از محدوده آن سازمان توليد می گشت.بعلاوه چون از سازماني خارج از سازمان داخلي مي بايست نرم افزار گرفته مي شد ، بايد مسائلی چون زمان و چگونگي اجرا و غيره مشخص مي شد.

اين در حالی بود كه سازمان های توليد كننده نرم افزار تجربه و دانش مديريت و اجرای اين حجم از نرم افزار مورد تقاضا را نداشتند.

اين مواردباعث بوجود آمدن بحران نرم افزار *Software Crisis* گشت.

دوره سوم

اين دوره از اواسط دهه 70 تا اواسط دهه 80 بوده است كه داراي خصوصياتي بشرح زير مي باشد:

1-*Disribiuted Systems*

چون *PC* ها در این زمان حیات پیدا کردند ، پس شبکه نیز به دنبال آن بوجود آمد وعملا *PC* به همراه شبکه در بسیاری از جاها به جاي *Mainframe* مورد استفاده قرار گرفت.این مسئله لزوم ایجاد سیستم های توزیع شده را به دنبال داشت.

2-*Low Cost Hardware* /Customer Impact

در دوره دوم قيمت سخت افزارها بسيار بالا بوده است . بطوريكه فقط شركتهاي بزرگ قادر به خريد اين وسايل بوده اند .

در دوره سوم بعلت بوجود آمدن کامپیوتر های خانگی که قیمتی به مراتب کمتر داشتند و به طبع تعداد بیشتری از افراد قادر به تهیه آن بودند، لذا نیاز به نرم افزار به مراتب بیشتراز قبل احساس شد.

3-*Embedded Intelligence*

این گونه نرم افزارها نرم افزارهايي هستند كه بصورت *Ic Programming* برنامه ريزي شده اند (برنامه از قبل به آنها داده شده است) و با وجود آنها سيستم يك رفتار هوشمند از خود نشان خواهد داد .

- مانند ماشين لباسشويي

دوره چهارم

این دوره از اواسط دهه 80 تا عصر کنونی ادامه پیدا مي کند . از خصوصیات این دوره میتوان به موارد زیر اشاره کرد :

1-*Powerful Desktop Computers*

کامپیوترهاي روميزي در این دوره داراي قدرت بالايي شدند و *P C* هايي به بازار آمدند که قدرت آنها حتی از *main frame* های قبلی نیز بیشتر بود.

2-*Object Oriented*

سیستم هاي شيءگرا هستند که دیدگاه شي گرایي و بکارگیري آن در این دوره بوجود آمد.

3-*AI (Artifical Intelligence)*

مخفف هوش مصنوعي یعني هر سیستمي که به نوعي هوشمندي در آن وجود دارد.

4-*Neural Network*

شبكه عصبي است كه ديدگاه شبكه عصبي در كامپيوتر از شبكه عصبي بدن انسان گرفته شده است،كه لايه هاي مختلف دارد و براساس *data* آموزش مي بيند.

5-*Expert system*

سيستم هاي خبره كه مصارف گوناگوني از جمله در پزشكي دارند ، به اين صورت كه يك پزشك زماني كه يك *expert system* در اختيار داشته باشد، با وارد نمودن علائم بيماري شخص بيمار مي تواند به تمام بيماريهايي كه امكان وقوع آنها با اين علائم وجود دارد دست يابد و حتي علائم ديگري را نيز از آن به نمايش مي گذارد.

6-*Genetic Algorithms*

7-*VRML*

مخفف کلمات *Virtual Reality Modeling Language* مي باشد.

8-*Parallel Processing*

پردازش موازي يعني انجام چندين كار بطور همزمان كه يا چند cpu وجود دارد و يا يك cpu از طريق timesharing كارها را بصورت موازي انجام مي دهد.

يك نرم افزار شامل سه بخش مي باشند :

- برنامه ها و دستورالعمل هايي كه درصورت اجرا اعمال موردنظر ما را انجام مي دهند.

- ساختمان داده هايي كه برنامه را قادر به انجام عمليات لازم روي اطلاعات مي كنند.

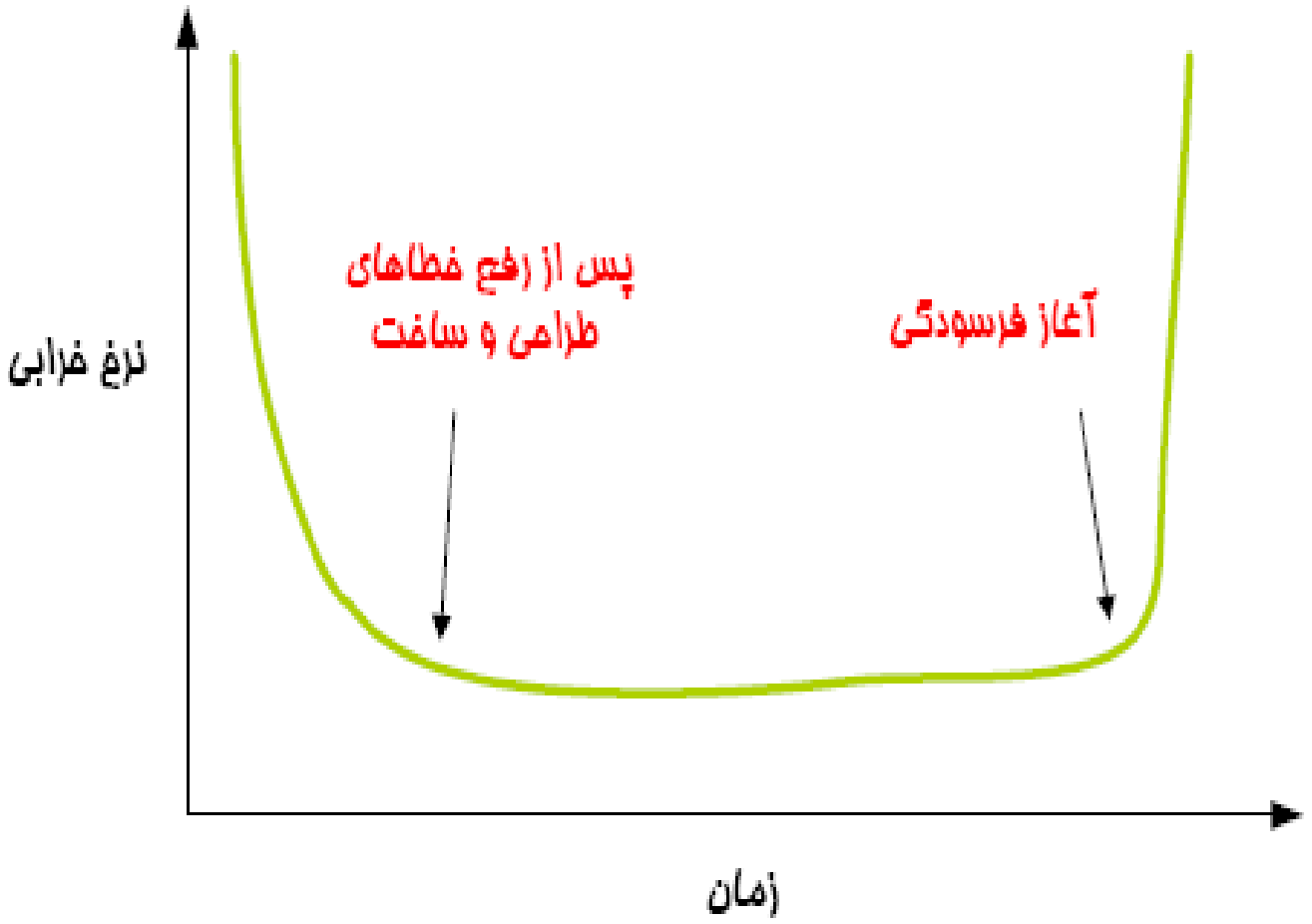
- مستنداتي كه عمل برنامه (*Technical Report*) و چگونگي استفاده از برنامه (*User Guide*)را مشخص مي كنند.

ويژگيهاي نرم افزار

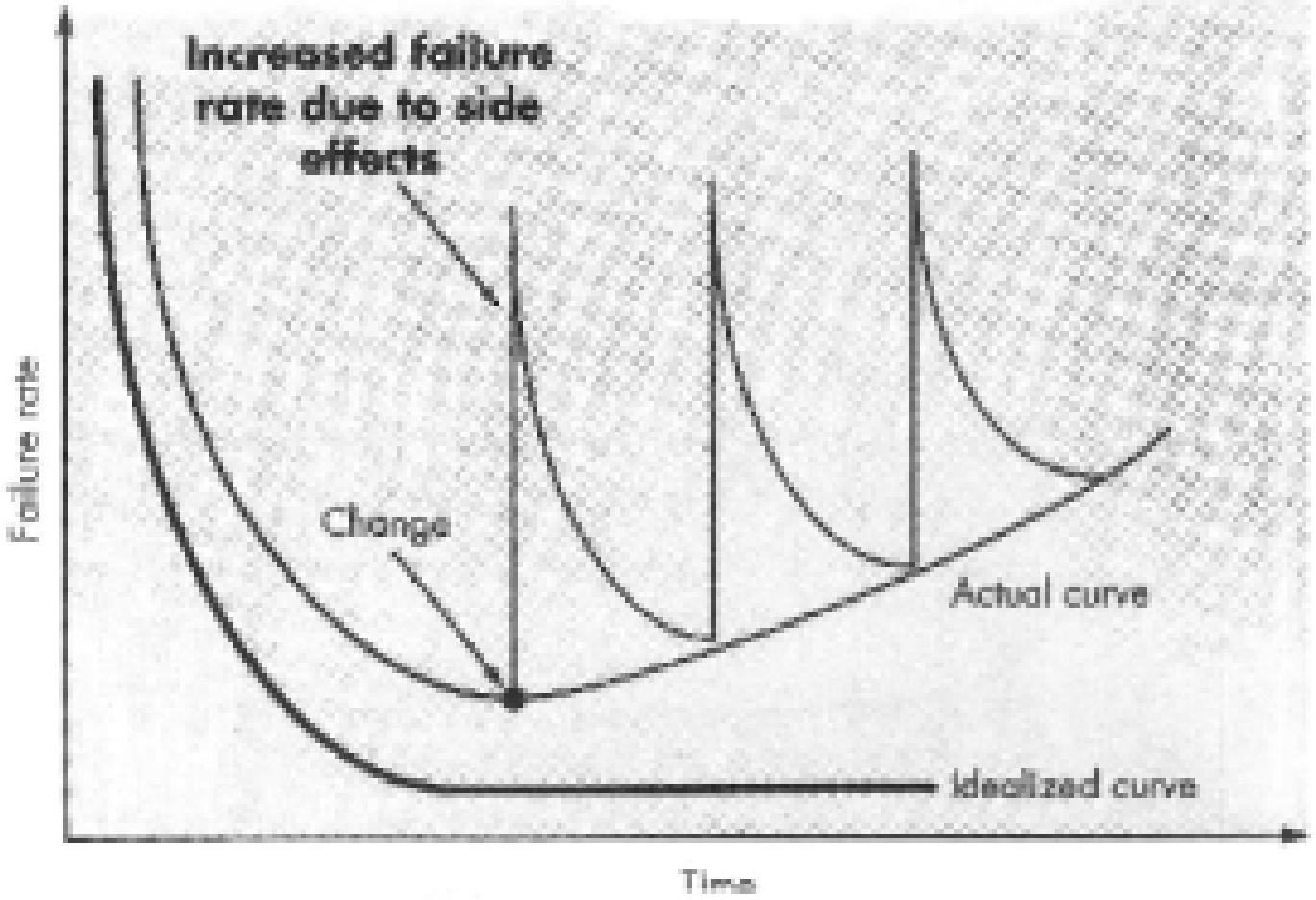
1-*Software is developed or engineering, it is not manufactured*

2-*Software Dose not Wear out (but it deteriorates)*

3-Most Software Are Custom Build Than being Assembled From Existing Components



منحنی نرخ خرابی سخت افزار نسبت به زمان


Increased failure rate due to side effects
Change
Actual curve
Idealized curve
Failure rate
Time

طبقه بندي نرم افزار

1-*System Softwares*

2-Engineesing & Scientific Software

3-Realtime Software

4-PC Software (Personal Computer)

5-Artificial Intelligent Software

6-*Embedded Software*

7-*Business Software*

8-*Web Based Softwares*

System Softwares

- مجموعه اي از نرم افزارها كه براي دادن سرويس به بقيه نرم افزارها از آنها استفاده مي شود ، مثل *Editors* ، كامپايلرها ، سيستم عاملها ، *Loader* و ... كه عمدتاً با داده هاي پيچيده ولي مشخص سر و كار دارند.

خصوصيات آنها

- 1-ارتباط با سخت افزار.
- 2-بكارگيري زياد توسط كاربران.
- 3-اشتراك منابع در اين نرم افزار.
- 4-معمولاً داراي واسطه هاي چندگانه هستند.
- 5-عمليات همزمان و موازي.
- 6-معمولاً داراي ساختمان داده اي پيچيده هستند.
- 7-فرآيند هاي پيچيده در مورد مديريت يا كنترل پروسه ها وجود دارد.

Engineesing & Scientific Software

نرم افزارهاي علمی یا مهندسي مي باشند. نرم افزارهائي مانندAutocad وMATLAB از این دسته می باشند . Casetools ها که نرم افزاري براي توليد نرم افزارهاي ديگرمی باشند نيز از اين دسته اند. ابزار Case tools مثلselect یا Oracle designer .

Realtime Software

نرم افزارهاي بلادرنگ كه بايد در مدت زمان مشخص tبه ما پاسخ دهند.

- در هر سيستم *realtime* سه مؤلفه اصلي وجود دارد :
- 1- جمع آوري كننده اطلاعات (*data Gathering*) : داده ها را جمع آوري مي كند . براي انجام هركار بايد داده هاي محيطي مورد نيازجمع آوري شود.
- 2- كنترل *Output* (*Control Output*) : خروجي را براي ما توليد مي كند و به محيط خارجي پاسخ مي دهد. اگر قرار بود نرم افزار غير *Realtime* باشد، اين دو *Component*كافي بود، اما مؤلفه سومي هم داريم.
- 3- مديريت (*Management*) : مديريت دو قسمت قبل را به عهده دارد و تعيين مي كندكه در مدت زمان مشخص سيستم جوابگو است.

PC Software (Personal Computer)

نرم افزارهايي كه فقط براي *p c* ساخته شده اند و نه براي *mainframe* ها. نرم افزراهايي مانند *اتوماسيون اداری و بازيها از اين دسته اند* .

Artificial Intelligent Software

*نرم افزارهاي هوش مصنوعي، نرم افزارهايي هستند
كه از الگوريتم هاي غيرعددي براي حل مسائل
پيچيده اي استفاده مي شوند كه غالبا از روشهاي
عادي و عددي قابل حل نيستند. مانند نرم افزارهاي
سيستم هاي خبره و يا تشخيص صدا و چهره.*

Embedded Software

این دسته از نرم افزارها در يك *IC* تعبيه شده اند و رفتار منطقي از پيش تعيين شده اي را خواهند داشت. به آنها نرم افزارهاي نهفته يا دروني هم گفته مي شود. در حافظه خواندني سيستم و براي كنترل سيستم از آنها استفاده مي شوند مانند ماشين لباسشوئی و جاروبرقی.

Business Software

عمده صحبتهاي ما در اين قسمت است مثل سيستمهاي حسابداري، انبار ، حقوق و دستمزد و ... كه عمده آنها يك بانك اطلاعاتي دارند و با اجتماع اينها باهم مي توان يك سيستم MIS يا سيستم اطلاعات مديريت ايجاد نمود.

Web Based Software

شامل کلیه نرم افزارهایي می باشد که در محیط web کار مي کنند

ضوابط ارزيابي نرم افزار :

1-عوامل داخلي : عواملي هستند كه صرفاً توسط كسي كه نرم افزار را نوشته و با ديدن كد برنامه قابل تشخيص است.
مهمترين چيزي كه در اينجا وجود دارد *modularity* برنامه است .

2-عوامل خارجي : توسط كسي هم كه نرم افزار را ننوشته و با بررسي نرم افزار قابل تشخيص ميباشد .

correctness

برنامه دقيقاً وظايفي را كه برايش تعريف كرده ايم بدرستي و مطابق با نيازهاي كاربر انجام دهد و تمام نيازهايي كه كاربر از برنامه انتظار دارد، همه را جواب دهد.

Robustness

استحكام برنامه، يعني توانايي برنامه در جوابگويي به كاربر در شرايط غيرعادي.

portability

قابلیت حمل برنامه : برنامه با سخت افزارها و سیستم عاملهاي مختلف قابل اجرا باشد. مثال:

بانک اطلاعاتی *oracle*

زبان برنامه نویسی Java

compatibility

- تطبیق پذیری: براي تركيب و وجود رابطه بين محصولات مختلف نرم افزاري بكار گرفته مي شود. *Compatibility* از سه جنبه حائز اهميت است:
- الف) استاندارد در ساختمان داده هاي استفاده شده.
- ب) استاندارد در واسطه هاي كاربري (*user interface*)
- ج) استاندارد در فايلها.

Efficiency

- معمولاً ازدو جنبه بررسي مي شود. از نظر *resource* هايي كه استفاده مي كند و از نظر زماني كه استفاده ميكند. نرم افزاری كه *time efficient* باشد از زمان كمتر و منابع بيشتري استفاده ميكند و نرم افزاري كه *resource efficient* باشد از منابع كمتر و زمان بيشتري استفاده ميكند.

Reusability

- امكان اينكه بتوان بخشي ازيك نرم افزار و يا تمام نرم افزار را در ساختن نرم افزار ديگري استفاده كرد را *reusability* (قابليت استفاده مجدد)گويند.

فرق بين فرآيند نرم افزار و مهندسي نرم افزار :

- اولی خط مشی ها را مشخص می کند و دومی در این خط مشی ها حرکت میکند و برای این کار از یک سری از ابزارها هم استفاده می شود.
- تفاوت و وجه تمايز مهم يك فرآيند نرم افزار با مهندسي نرم افزار در اين است كه اولي به بيان چارچوب و اصول و روشهاي توليد نرم افزار مي پردازد و دومي با استفاده از اين اصول و روشهاي و علاوه بر آن بكارگيري پاره اي از ابزارها و ادوات به ايجاد و مديريت نرم افزار مي پردازد.

الگوهاي موجود در مهندسي نرم افزار :

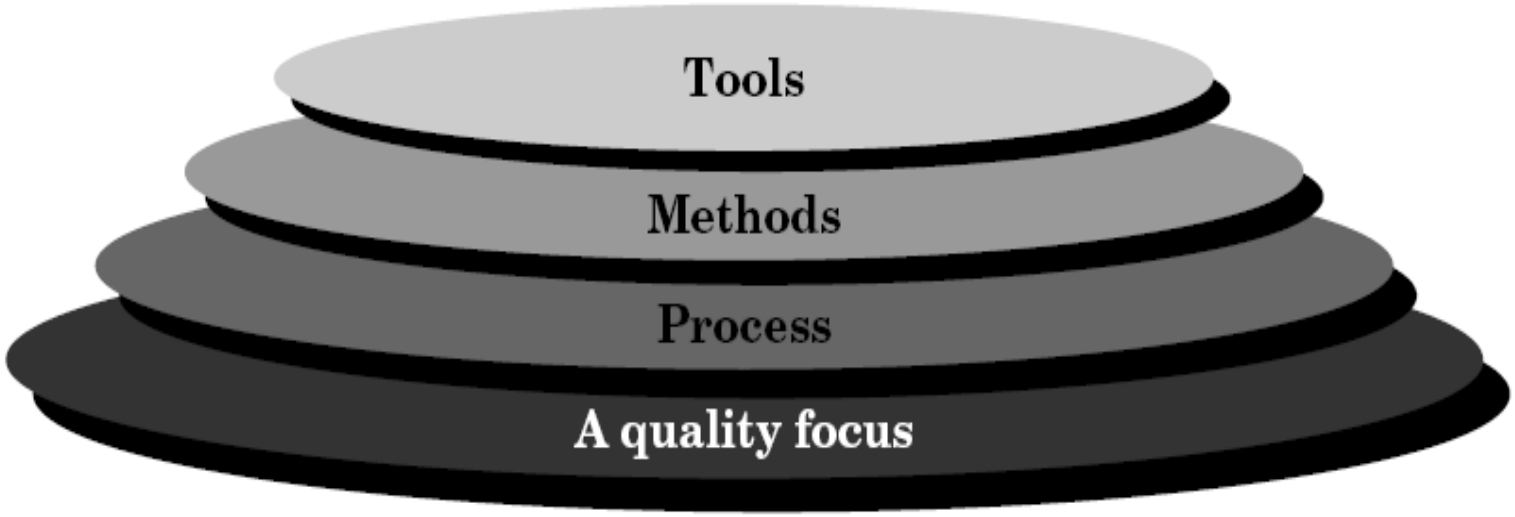


FIGURE 2.1
Software engineering layers

1-روشهاي مهندسي نرم افزار (*method*): روشهاي مهندسي نرم افزار چگونگي ساخت نرم افزار را بصورت تكنيكي بيان مي كند و شامل عملياتي نظير زمانبندي تكميل پروژه و تجزيه و تحليل نيازمنديهاي سيستم و نرم افزار و در نهايت تست نرم افزار و نگهداري برنامه خواهد شد.

2- ابزارهاي مهندسي نرم افزار (*tools*): ابزارهايي هستند که در اين زمينه به ما کمك مي کنند، مانند *case tool* ها.

3- رويه هاي مهندسي نرم افزار (*process*): رويه هاي مهندسي نرم افزار عملاً ابزارها و روشها را به يكديگر پيوند مي زند. عملاً رويه ها ترتيبي را كه روشها لازم است بكار برده شوند را مشخص مي كنند، مستندات و گزارشات و فرمهاي موردنظر و ... را مشخص مي كنند. كنترلهايي بر روي كيفيت پروژه تأليف مي كنند و در نهايت روند پيشرفت پروژه را براي مديريت پروژه مشخص مي كند.

فرایند

- فرایند نرم افزار نقشه راهی (Road Map) است که دو هدف زیر را دنبال می نماید:
  - کیفیت بالا
  - زمانبندی مناسب
- لایه های مهندسی نرم افزار

فرایند (ادامه)

- فرایند چارچوبی برای مجموعه ای از KPA ها(Key Process Area)ایجاد می نماید.
- KPA :
  - ایجاد پایه ای جهت کنترل مدیریتی پروژه های نرم افزاری
  - ایجاد بستری جهت انجام روشهای فنی ، تولید محصولات کاری (مدلها ، مستندات ، گزارشها ، فرمها ، داده ها و غیره) ، مشخص نمودن مراحل ، حصول اطمینان از کیفیت و مدیریت خوب تغییرات

- روشهای مهندسی نرم افزار شیوه های فنی جهت ایجاد نرم افزار را فراهم می نماید . برخی از وظیفه هائی که روشهای مهندسی نرم افزار باید آنها را پوشش دهد عبارت است از :

  تحلیل خواسته ها ، طراحی ، ساخت برنامه ها ، آزمایش و پشتیبانی
- روشها شامل فعالیتهای مدلسازی و سایر فنون توصیفی نیز می گردد.

فرایند (ادامه)

- ابزارهای مهندسی نرم افزار جهت پشتیبانی از فرایندها و روشها مطرح می گردند. زمانی که دارای مجموعه ای از ابزارها باشیم بگونه ای که اطلاعات ایجاد گردیده توسط یک ابزار ، ورودی برای سایر ابزارها بوده و توسط آنها استفاده گردد ، سیستمی برای پشتیبانی توسعه نرم افزار ایجاد می شود که به آن مهندسی نرم افزار به کمک کامپیوتر (CASE) گوئیم.

فعالیتهای مهندسی نرم افزار

- بطور کلی فعالیتهای مربوط به مهندسی نرم افزار در سه فاز زیر دسته بندی می گردد:
  - فاز تعریف
  - فاز توسعه
  - فاز پشتیبانی

    نگهداشت تصحیحی

    نگهداشت تطبیقی

    نگهداشت بهبودی

    نگهداشت پیشگیرانه
- فازهای فوق با یکسري فعالیتهای چتری (Umbrella activities) تکمیل می گردد. مهمترین آنها عبارتند از:

  کنترل و ردیابی نمودن پروژه های نرم افزاری ، تضمین کیفیت نرم افزار ، مدیریت پیکربندی نرم افزار ، تهیه مستندات ، مدیریت قابلیت استفاده مجدد ، سنجش و مدیریت ریسک

فعالیتهای مهندسی نرم افزار

- **Correction.** Even with the best quality assurance activities, it is likely that the customer will uncover defects in the software. *Corrective maintenance* changes the software to correct defects.

- **Adaptation.** Over time, the original environment (e.g., CPU, operating system,business rules, external product characteristics) for which the software was developed is likely to change. *Adaptive maintenance* results in modification to the software to accommodate changes to its external environment.

فعالیتهای مهندسی نرم افزار

- **Enhancement. As software is used, the customer/user will recognize additional functions that will provide benefit. *Perfective maintenance* extends the software beyond its original functional requirements.**

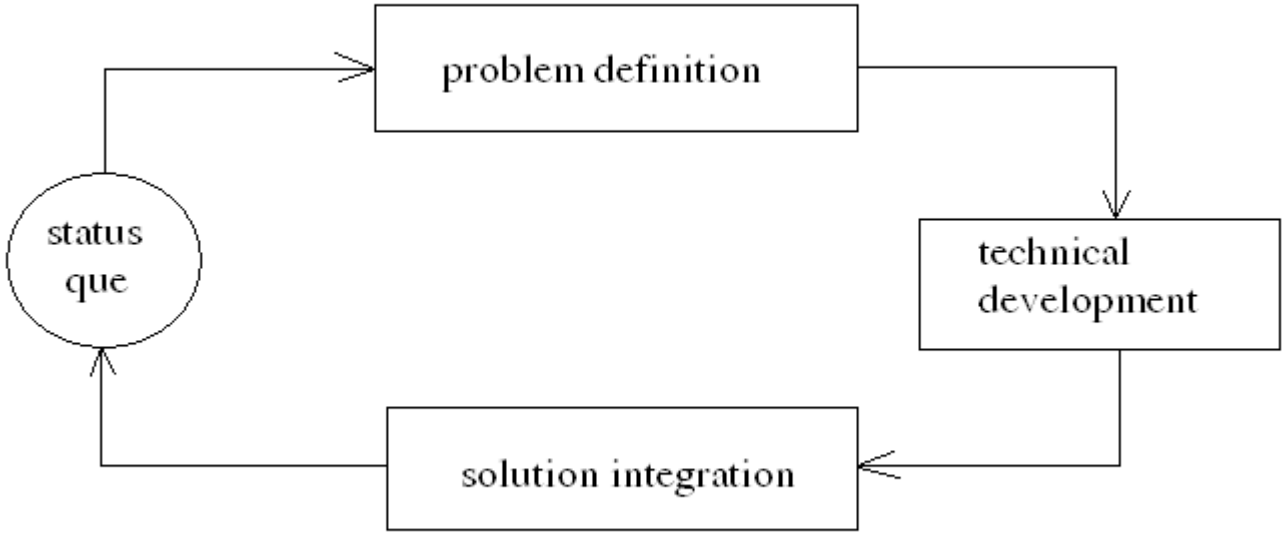
- **Prevention. Computer software deteriorates due to change, and because of this, *preventive maintenance,* often called *software reengineering,* must be conducted to enable the software to serve the needs of its end users. In essence,preventive maintenance makes changes to computer programs so that they can be more easily corrected, adapted, and enhanced.**

مدل فرایند

- تعیین یک راهکار توسعه که شامل لایه های فرایند ، روشها ، ابزار و فازها (تعریف ، توسعه و پشتیبانی) باشد را الگوی مهندسی نرم افزار یا مدل فرایند می نامند.
- مدلهای فرایند نرم افزار عبارتند از :
  - مدل ترتیبی خطی
  - مدل ایجاد نمونه اولیه (Prototyping Model)
  - مدل RAD
  - مدل افزایشی (Incremental Model)
  - مدل حلزونی(Spiral Model)
  - مدل حلزونی برنده - برنده (Win-Win)
  - مدل توسعه همزمان
  - مدل توسعه مبتنی بر مولفه (Component Base Development)
  - مدل روشهای رسمی (Formal Method)
  - تکنیکهای نسل چهارم

رویه حل مسئله

software process model



مزیت رویه حل مسئله

1- فقط مربوط به نرم افزار نیست.

2- در هرزیر قسمت هم مي تواند همین رویه تکرار شود .

- *software process model*هائی که بررسي مي كنيم عبارتنداز :

1)linear sequential model/waterfall model/classic life cycle

2)the prototyping model

3)the RAD model

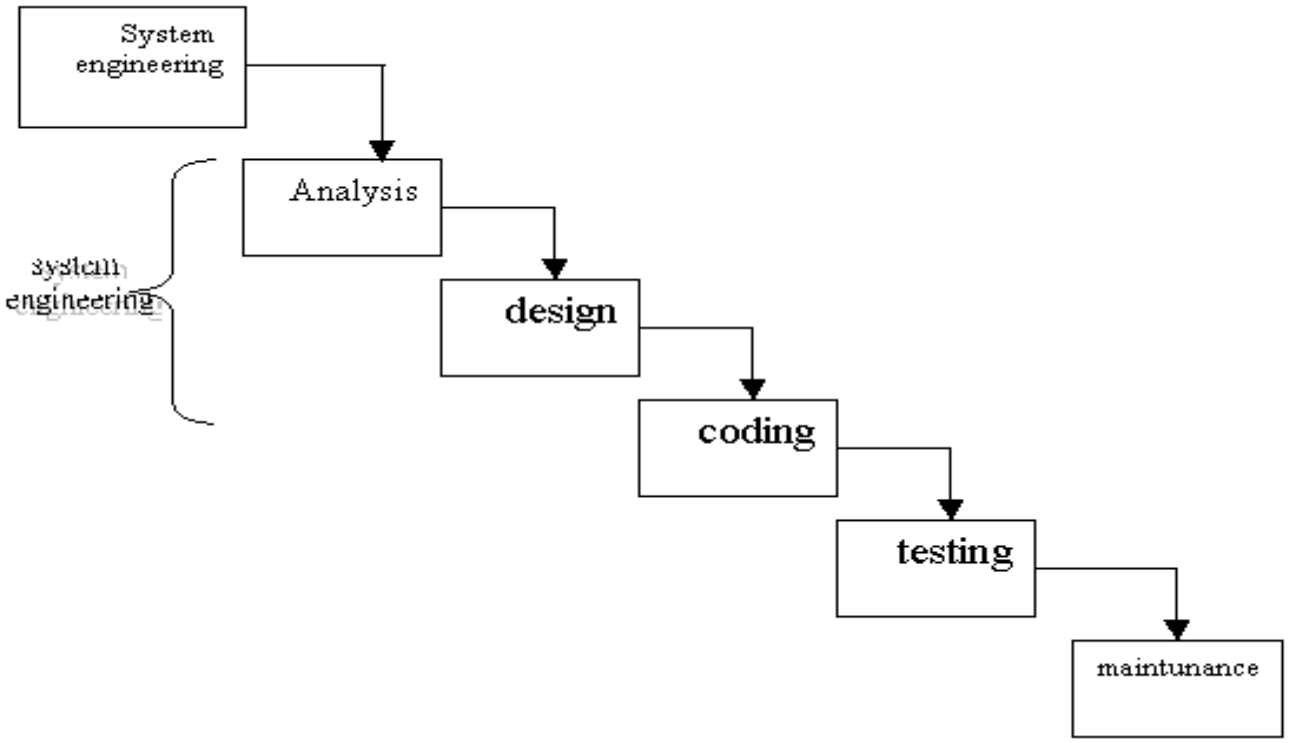
4)the spiral model

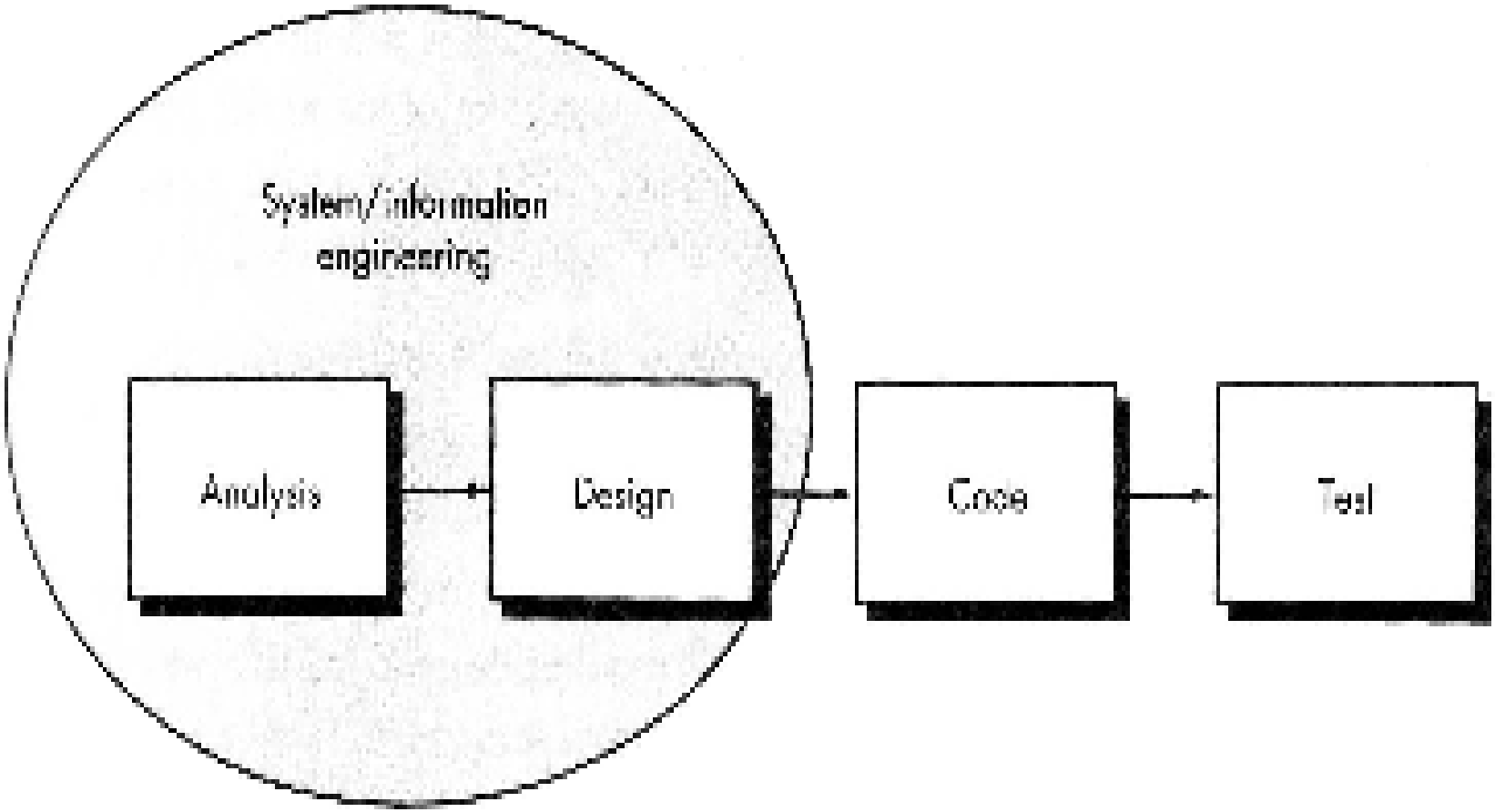
5) component-based development

6)fourth generation techniques(4GT)

7)the formal methods model

1)linear sequential model/ waterfall model/ classic life cycle




System/information engineering
Analysis
Design
Code
Test

System engineering

نرم افزار همواره قسمتي از يك سيستم بزرگتر است لذا كار هميشه ازمشخص كردن كل نيازمنديهاي سيستم آغازمي شود سپس زيرمجموعه اي ازاين نيازمنديها را به نرم افزارنسبت مي دهند.

نرم افزار مرتبط باسخت افزار, مودم, كاربران وكل سيستم است جزئي است كه براي بهتر شدن دقت وسرعت و امنيت كاردر سيستم بكار گرفته می شود.

اين فاز شامل جمع آوري نيازمنديها درسطح سيستم وكمي هم تجزيه وتحليل وجمع آوري نيازمنديها در سطح بالاتر است

Analysis

جمع آوري نيازمنديها مربوط به نرم افزاري كه قرار است توليدشود دراين فاز انجام مي شود .درفازآناليز بايد دامنه اطلاعات (كاراصلي كه بايدانجام شود)، عمليات (ورودي هاوخروجي ها)، واسط ها و پردازش هاي موردنياز شناسائي شود. دراين فاز براي مشخص شدن نيازمنديهاي نرم افزار با مشتري تبادل اطلاعات مي شود و در نهايت عمليات انجام شده *document* مي شود .

Design

فازطراحي يك فرايند چند مرحله اي است شامل

1 - *data structure design*

2- *user interface design*

3- *module design*

4 - *software architecture*

درفاز طراحي نيازمنديها عملا تبديل به نمايشي از نرم افزارمي شود. که دراينجا نمايشي از نرم افزار را قبل اززرسيدن به مرحله کد نويسي بررسي مي کنند. طراحي هم مانند ساير قسمتها در مهندسي نرم افزارلازم است مستند سازي شود.

Coding

. آنچه درطراحي مشخص شده تبديل به فرم قابل فهم براي كامپيوتر مي شود هرچه در طراحي جزئيات كارانجام شده بيشتر باشد مرحله كدنويسي مي تواند بسيار سريعتر و راحت تر انجام گيرد.

Testing

دودسته تست وجود دارد که عبارتند از :

- *white box testing*(1 تست رويه هاي داخلي (تست جعبه سفيد)
  دراين نوع تست، به تست روالهاي منطقي داخل نرم افزار مي پردازند عملا دراين قسمت اطمينان حاصل مي شود که تمامي دستور العمل هاچك شده است. مانندپروسس هاي *debuging* که انجام مي شود.

- *black box testing*(2 تست عملکردسيستم(تست جعبه سياه)
  دراين نوع تست، عمليات خارجي برنامه تست مي شود و عملا در اين قسمت اطمينان حاصل مي شود از اينکه برنامه به ازاء ورودي تعريف شده نتايج مورد نظررا ايجاد کند.

Maintenance

- فازنگهداشت سيستم: نرم افزاريك محصول ديناميكي ا ست ومرتبا درحال تغييرات مي باشد پس نياز است تغييرات اعمال شود تغييرات درنرم افزار درسه محور انجام مي شود :
- 1- ممكن است تشخيص خطاداده شود كه توسط سيستم برطرف مي شود.
- 2- ممكن است ازيك سيستم عامل به سيستم عامل ديگر منتقل شود.
- به عنوان مثال سيستم حسابداري كه روي *Dos* كار مي كرده بخواهندروي ويندوز يا يونيكس كاركند.
- تغييرات مي تواند براي تطابق نرم افزار براساس خواسته هاي كاربران در محيط بيروني باشد فرضا نرم افزار قادر باشد روي يك سيستم عامل ديگر يا يك *platform* سخت افزاري ديگركاركند.
- 3- ممكن است به علت مطرح شدن نيازهاي جديد توسط كاربران باشد.
- به عنوان مثال در نرم افزارسيستم دانشجوي پاره وقت نيزاضافه شود (كه تابه حال وجود نداشته) در واقع نياز جديدي مطرح شده, رويه عملكرد دانشجوي پاره وقت با دانشجوي روزانه متفاوت است و بايد اين تغييرات روي نرم افزار اعمال گردد.

معايب مدل آبشاري

1- در پروژه هاي واقعي جريان كار يك جريان ترتيبي نيست دراغلب اوقات بازگشت به عقب در كار اتفاق مي افتد اين نحوه نگرش كه هرمرحله پس ازپايان مرحله قبلي انجام شود نحوه نگرش درستي نيست بازگشت به عقب جزء ذات كار است.

2- تا زماني كه مرحله *coding* تمام نشود نمونه اي براي ارائه به مشتري وجود ندارد كاربر ناچار است براي مشاهده نمونه تا پايان مرحله *coding* منتظربماند. اين امكان وجوددارد‌كه كاربر پس ازمشاهده *coding* آن را قبول نكندكه دراين زمان تقريباً پروژه به پايان رسيده است و با نپذيرفتن كاربر تمام مراحل انجام شده عملا بي ارزش خواهد بود. واضح است كه اگر يك اشتباه بزرگ تا اين مرحله شناخته نشده باشد در اين مرحله مي تواند منجر به بحران شود.

معايب مدل آبشاري (ادامه)

3- اغلب درشروع كار بيان همه نيازمنديها ازسوي كاربر امكان پذير نيست ولي مدل آبشاري به اين موضوع نياز دارد. باعدم قطعيت موجود در شروع بسياري ازپروژه ها با اين روش به مشكل برخورد خواهند كرد.

4- با توجه به اينكه درفرايند ساخت نرم افزار غالباً تيم تجزيه وتحليل و طراحي از تيم مربوط به كدنويسي مجزا مي باشند. در اينجا لازم است تا تجزيه وتحليل وطراحي تمام شود تا كدنويسي آغازگردد. لذا در هربخش اين پروژه تنها تعدادي از اعضاي تيم فعال هستند و تعدادي ديگر بيكار مي باشندكه اين مسئله ازنظر مديريت نيروي انساني نامطلوب است .

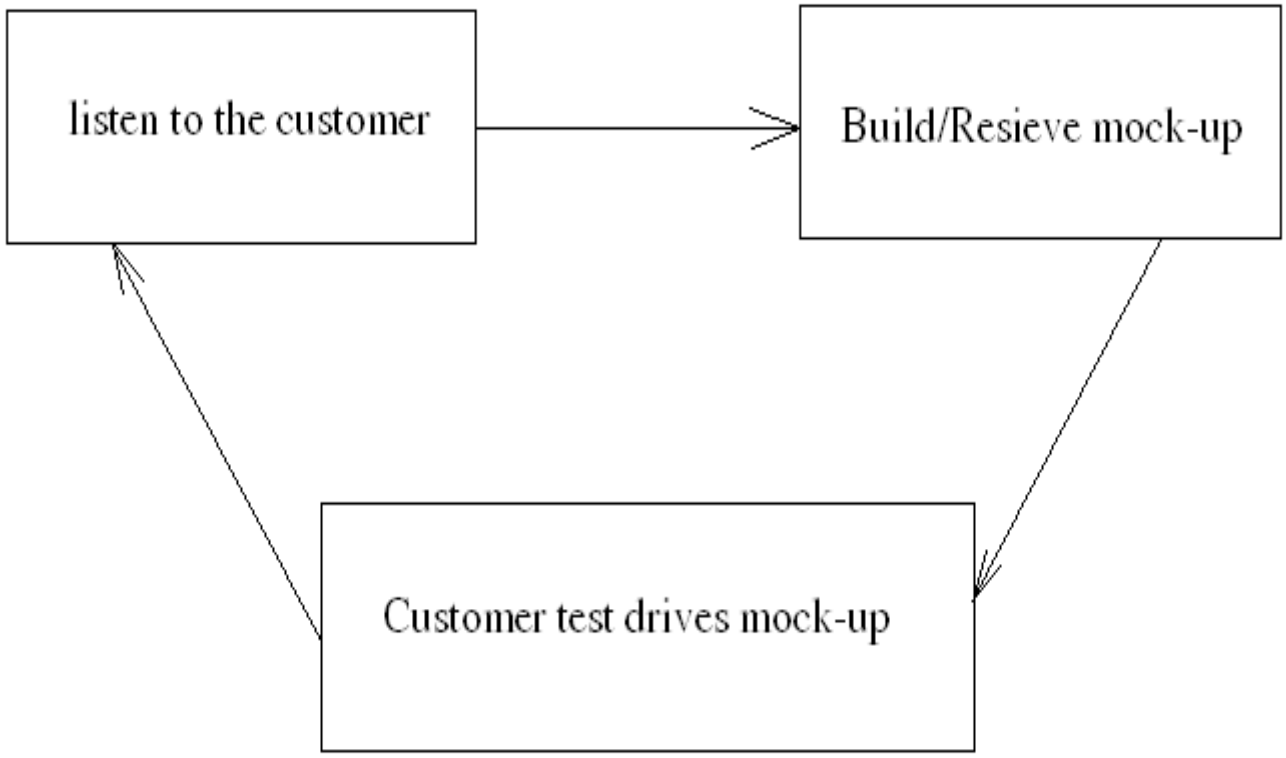
مزاياي مدل آبشاري

1- مرحله به مرحله مي باشد.

2- مهندسي ساز مي باشد.

3- در همه قسمتهاي آن *document* تهيه مي شود.

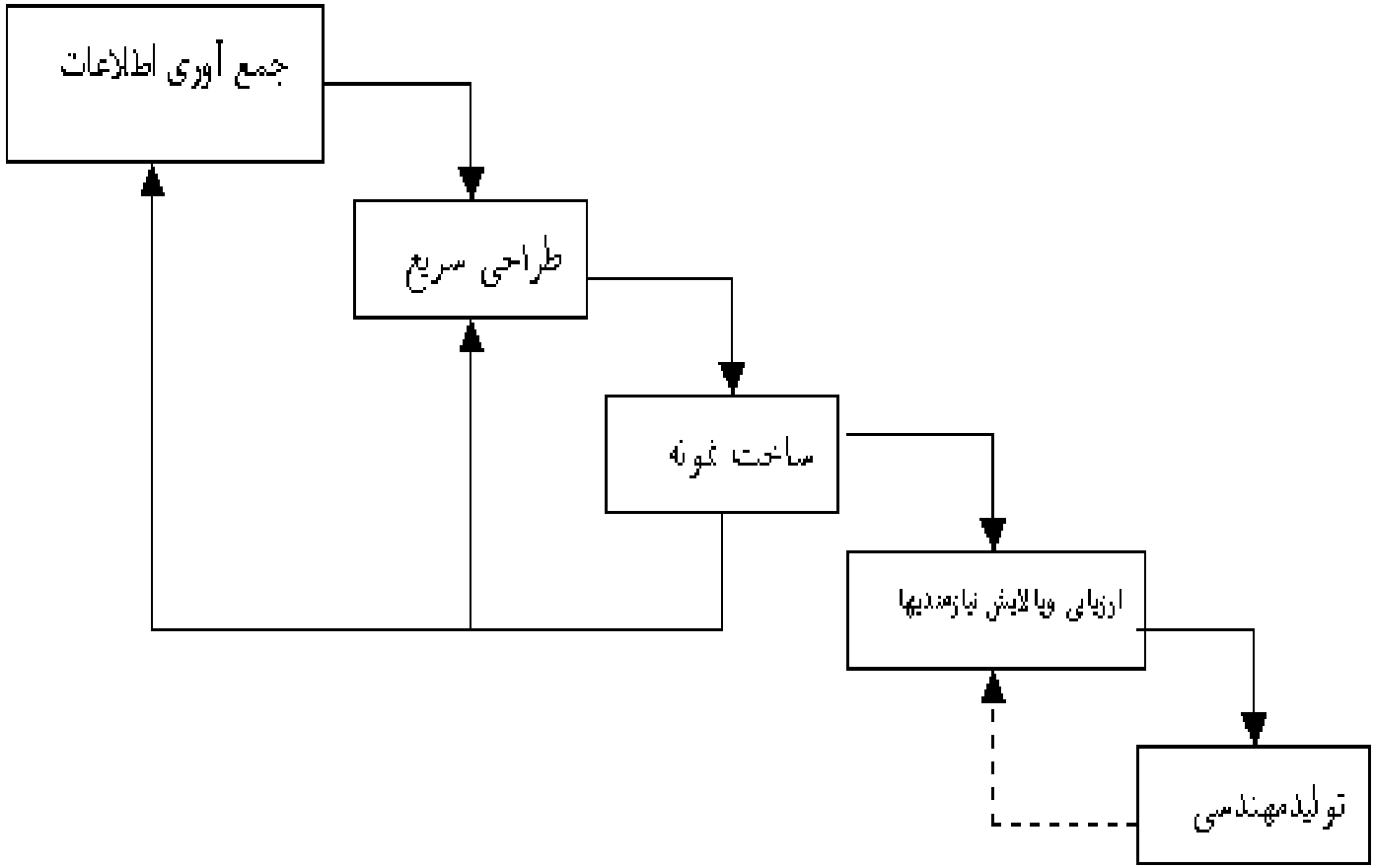
2)the prototyping model



- 1-*(listen to customer)* به صحبتهاي مشتري براي پي بردن به نيازهايش گوش مي دهند.

- 2-*(build/revise mock up)* براساس آنچه مشتري بيان نموده مدلي كه قراراست ساخته شود را مي سازيم .

- 3-*(customer test drives mock-up)* آنچه ساخته شده مورد تست وبررسي قرار مي گيرد و به مشتري ارائه مي شود اگر مورد تائيد مشتري دراين *step* بود كارتمام مي شود در غير اين صورت اين سيكل دوباره تكرار مي شود.


جمع آوری اطلاعات
طراحی سریع
ساخت نمونه
ارزیابی وپالایش نیازمندیها
تولیدمهندسی

- درحالت كلي *prototype* رامي توان به سه شكل زير پياده سازي نمود :

thrown away-1 (دور انداختني)

incremental-2 (افزايشي)

Evolutionary-3 (تكامل تدريجي)

- درحالت كلي *prototype* مي تواند به يکی از سه شکل زير ارائه شود :

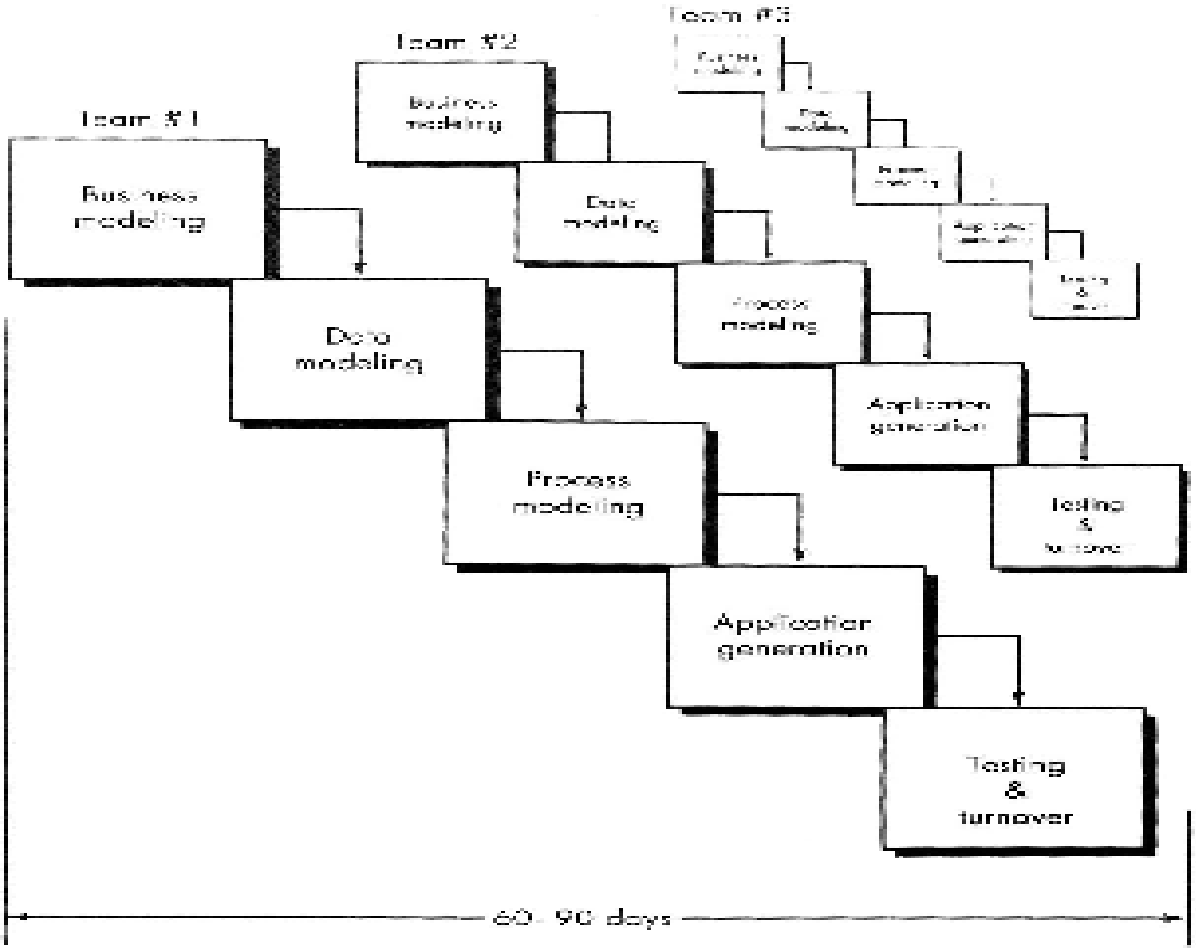
1. *paper prototype*
2. *working prototype*
3. *Existing prototype*

معايب بكارگيري *prototype*

1- مشتري به محض رسيدن *prototype* مخصوصاً اگر روي كامپيوتر باشد تصورمي كند سيستم آماده شده چون درك درستي ازقضيه ندارد.

2- امكان دارد در هنگام ساختن *prototype* از ابزار راحت استفاده شود يا از الگوريتم نامناسب استفاده شود. امكان دارد از ابزاري كه درتهيه *prototype* استفاده مي شود از ويژگي خاصي برخوردار باشدكه درتوليد نهايي نتوان ازآن ويژگي استفاده نمود .

3)the RAD model (*RAPID APPLICATION DEVELOPMENT*)



Bussiness modeling

دراینجا *business* مربوطه یا کارمربوطه ازدیدگاه افراد متخصص در حرفه مورد نظر مورد بررسی قرار گرفته و مدل مربوطه اصلاح می گردد.

Data modeling

سیستم اززاویه داده هایي که درآن وجوددارد و ارتباط بین داده هاي مختلف مدل مي شود. درسیستم هاي مکانیزه مهمترین فاز مي باشد. در این مرحله نمودار ERD یاLDS استخراج مي شود.

Process modeling

پردازش هايي كه قراراست درسيستم انجام شودمدل مي شودسيستم ازديدگاه پردازشي مدل مي شود. دراينجا مشخص مي شود كه چطورمي توان از *data* به *information* رسيد.

Application generation

دراينجا باداشتن مدل داده ومدل پردازشي سيستم *application* توليد مي شود. اجراي برنامه دراين مرحله مي باشد.

Testing & turnover

تستهاي مربوط روي محصول انجام مي شود سپس به مشتري تحويل داده مي شود.

شرایط مورد نیاز جهت بکارگیری مدل RAD

1. برای پروژه های بزرگ ولی با قابلیت اندازه گیری،این روش نیاز به نیروی انسانی کافی جهت تشکیل تیم های RAD به تعداد کافی خواهد داشت.

2. در این روش لازم است تا تولید کنندگان و مشتریانی داشته باشیم که توانائی لازم جهت درک و انجام کار در یک بازه کوتاه زمانی را داشته باشند در غیر این صورت بکار گیری این روش نمی تواند موفقیت آمیز باشد.

مواردي كه نمي توان ازمدل RAD استفاده كرد:

1. استفاده ازاين مدل براي *application* هاي بزرگي كه شناخت درستي نسبت به آنها موجودنباشد *(management Risk)* مناسب نيست چون نمي توان دقيقاً زمان انجام پروژه راتخمين زد. يا در مورد *application* هايي كه ازتكنولوژي جديداستفاده مي كنند نيزاستفاده از اين روش ريسك بالايي خواهد داشت.دركل مواقعي كه سيستم *high Risk (technical Risk, management Risk)* باشد،توصيه نمی شود.
2. - اگردرتوليدسيستم *performance* حائزاهميت است و اين *performance* ازطريق *optimize* كردن *interface*بدست آيد.
3. اگر يك *application* نتواند ماجولار شود اين روش، روش خوبي نيست و نميتواند بكارگرفته شود.

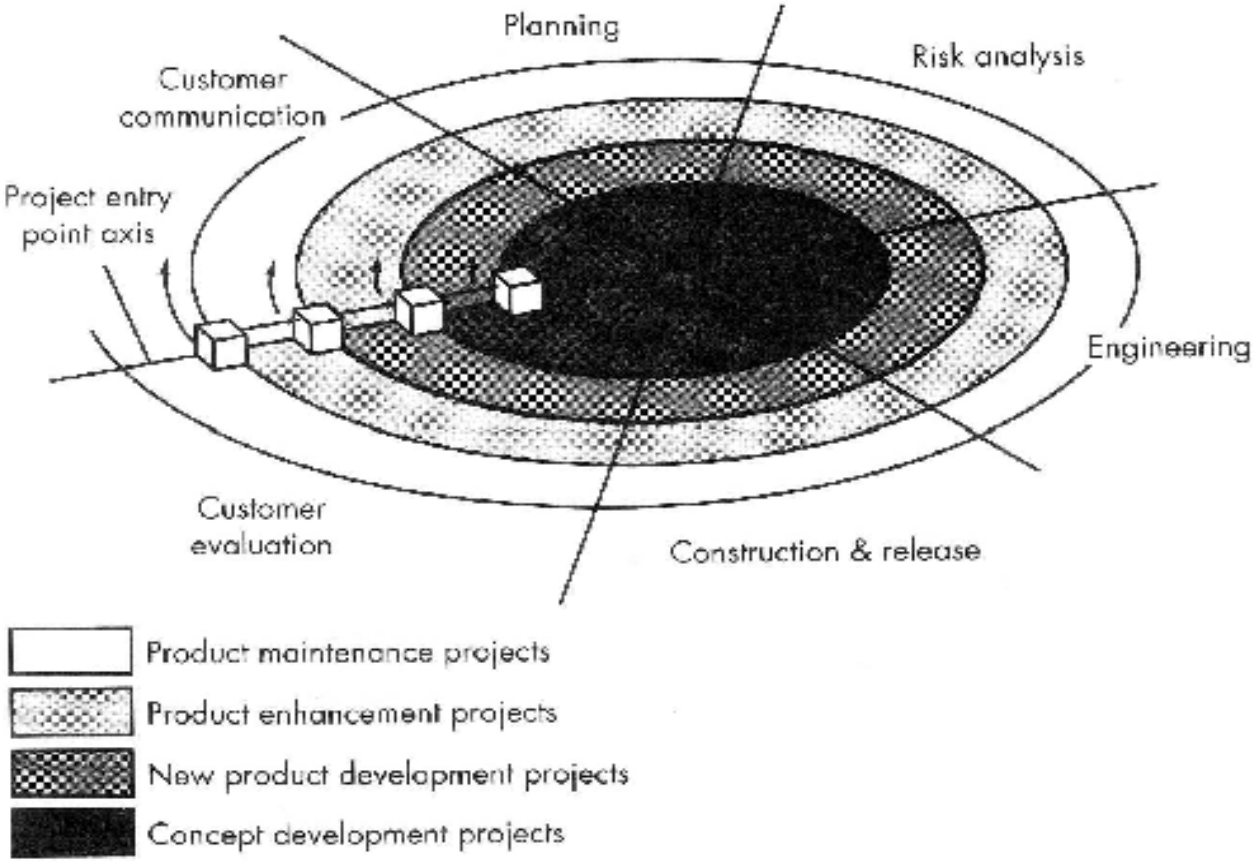
مواردي كه نمي توان ازمدل RAD استفاده كرد:

1- استفاده ازاين مدل براي *application* هاي بزرگي كه شناخت درستي نسبت به آنها موجودنباشد *(management Risk)* امكان پذيرنيست .

2- اگردرتوليدسيستم *performance* حائزاهميت است و اين *performance* ازطريق *optomize* كردن *interface* ها می باشد.

3- اگر يك *application* نتواند Modularizeشود اين روش، روش خوبي نيست يعني نمي توان به راحتي آن را به قسمتهايي تكه تكه نمود.

4)the spiral model



Customer communication

برقراري ارتباط بامشتري :بامشتري ارتباط برقرارميشودونيازمنديهاي مشتري استخراج مي شود.

Planning

برنامه ريزي: برنامه ريزي هاي مورد نياز براي توليدسيستم موردنظر مشتري دراين قسمت انجام مي شود.

تعیین اهداف ، آلترناتیوها و محدودیتها تعیین منابع و ایجاد زمانبندی

Risk analysis

دراينجاريسك ناشي ازمسائل فني ومديريتي سيستم را استخراج مي كنند.در انتهاي كارمعمولا مطالب ريسك درقالب *Risk profile* (پرونده ريسك)مطرح مي شود.

تحلیل آلترناتیوها ، شناسائی ریسکها و راهکارهای مقابله با آنها

Engineering

عملا همان قسمتهاي آناليز و طراحي می باشد که در اينجا با عنوان Engineering از آن ياد شده است.

Construction & Release

در اينجا :سيستم موردنظرساخته شده و سپس يك سري تست برروي آن انجام مي شود و دراختيارمشتري قرارمي گيرد(release) .

Customer evaluation

ارزيابي توسط مشتري :مشتري سيستم توليدشده رامورد ارزيابي قرارمي دهد و نظرات خود را در مورد آن بيان مي كند.

- این مدل رامي توان به قسمتهاي زير تقسيم نمود:

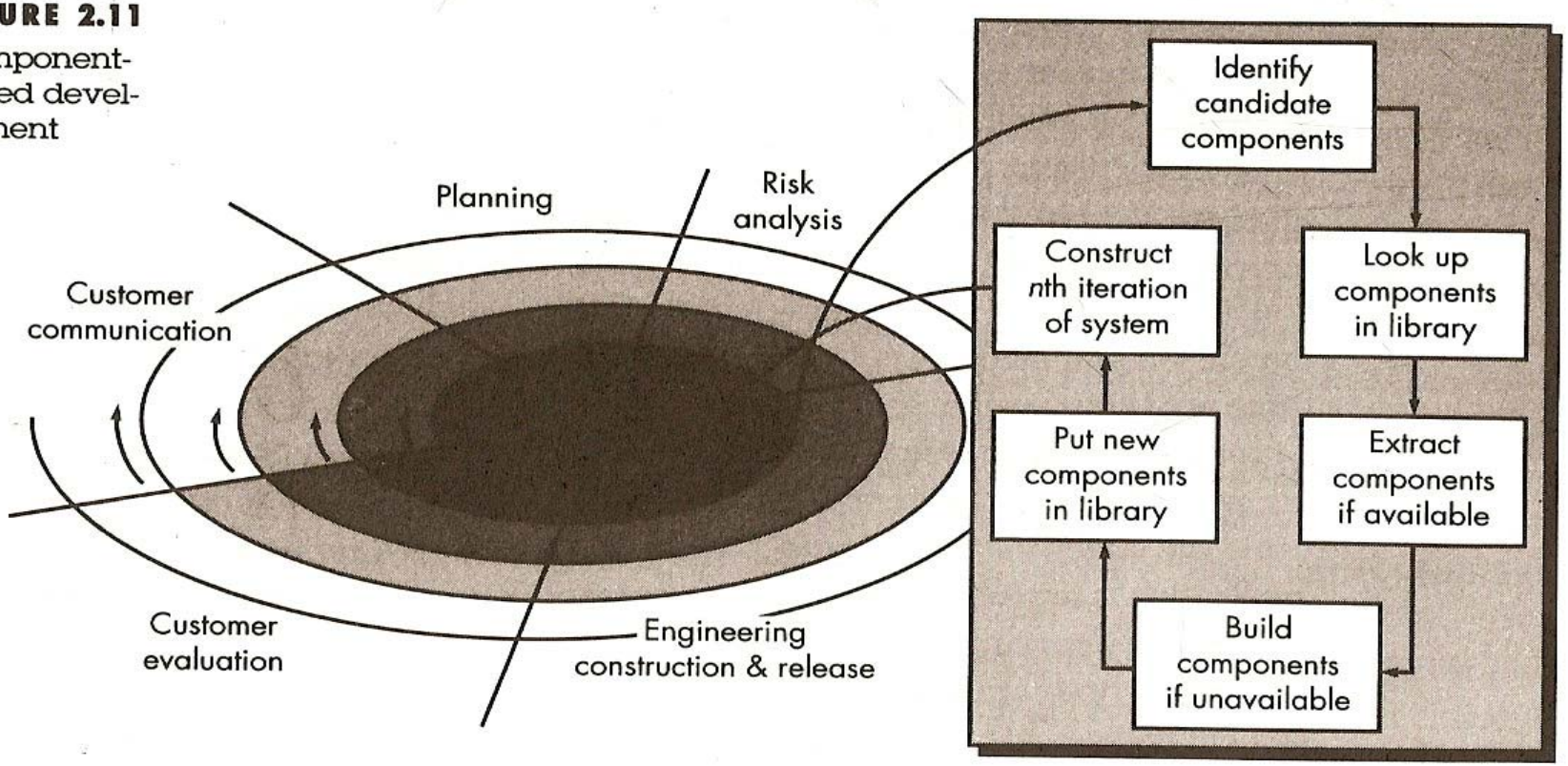
- *concept development phase*
- *new product development phase*
- *product enhancement phase*
- *product maintenance phase*

5)component-based development

این مدل مشابه مدل *spiral* است .دوفاز *engineering*
و *construction & release* از مدل *spiral* به عنوان يك فاز
در اين مدل درنظرگرفته مي شوند

5)component-based development

FIGURE 2.11
Component-based development



12 This is a simplified description of class definition. For a more detailed discussion, see Chapter 20.

مدل توسعه مبتنی بر مولفه (Component Base Development)

انجمن QSM associates در گزارشی ضمن تاکید بر قابلیت استفاده مجدد از نرم افزار برخی از مزایای آن را به شرح زیر بیان می نماید:

مونتاژ مولفه ها :

- باعث کاهش 70 درصدی زمان توسعه سیتم می گردد .
- موجب کاهش 84 درصدی هزینه های تولید سیتم می گردد.
- سبب افزایش ضریب بهره وری می گردد.

6) *fourth generation techniques(4GT)*

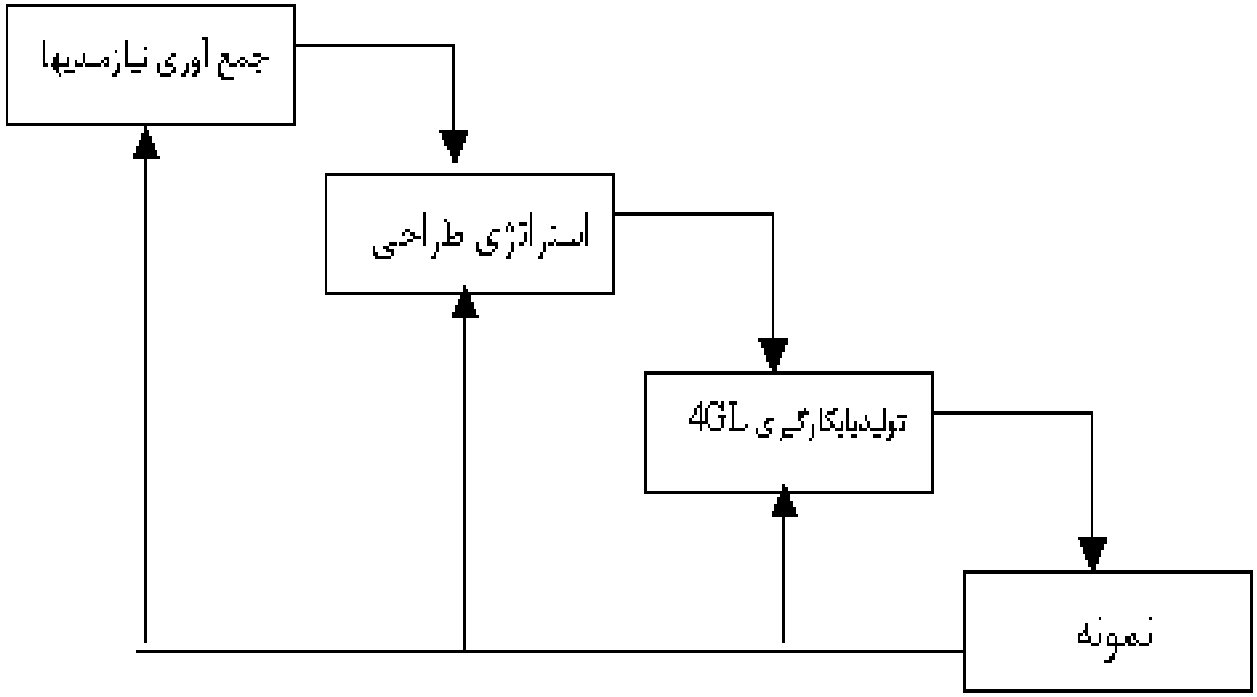
- دسته نسبتا بزرگي از نرم افزارها هستند خصوصيتي كه همه آنها دارند اين است كه مي توان صفات يا ويژگيهاي نرم افزار را درسطح نسبتا بالايي مشخص نمود و سيستم براساس آن ويژگي ها اقدام به توليد نرم افزارمي نمايد به اين دسته تكنيكهاي نسل چهارم گويند نمونه اي ازآنها ابزارهاي ويژوال هستند اين ابزارها بعلاوه يك سري امكانات *component base* را نيز دارا هستند.

6) *fourth generation techniques(4GT)*

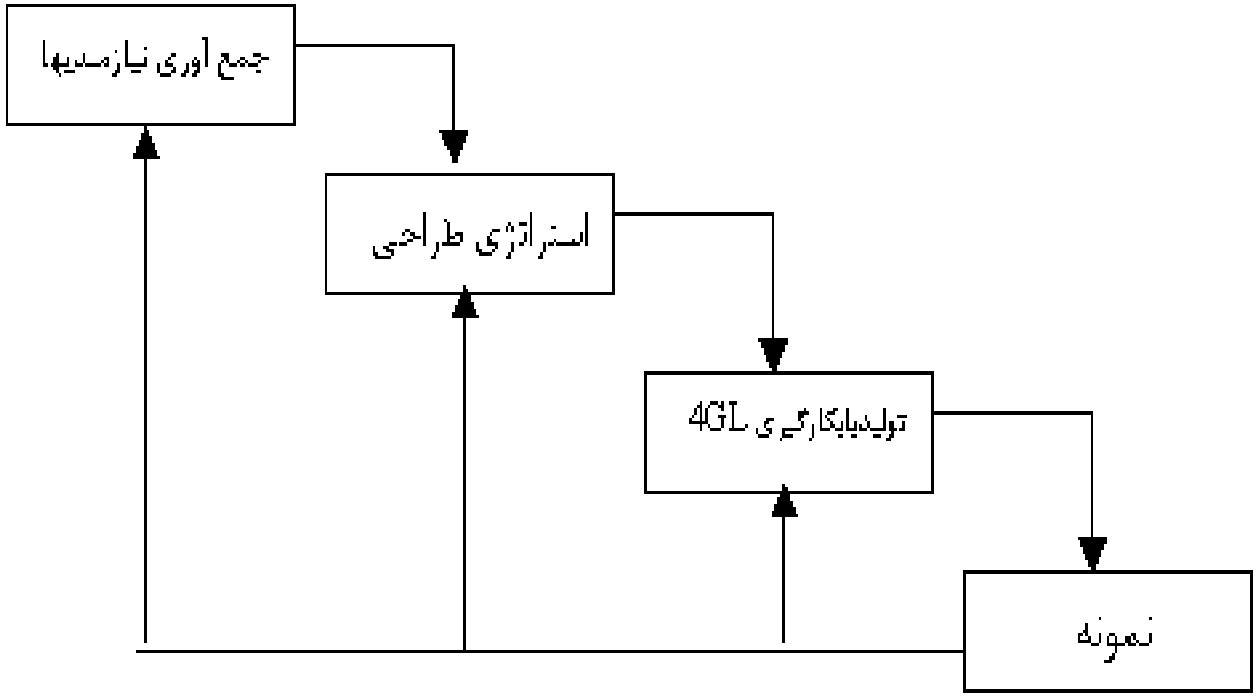
همه نرم افزارهايي كه با الگوي *4GT* كارمي كنند ويژگيهاي زير رادارا هستند:

1-زبانهاي غيررويه اي جهت پرس وجو از پايگاه دادها
2-امكان توليد *Form Generator*
3-امكان توليد *Report Generator*
4-امكانات گرافيكي درسطح بالا
5-امكان توليدكد

مراحل ساخت 4GT



مراحل ساخت 4GT



fourth generation techniques(4GT)

- استفاده از *4GL* يك سري موافق ومخالف دارد:
- نظرمخالفين: اعتقاد دارند زبانهاي *4GL* چندان هم ساده تر از زبانهاي عادي توليد نرم افزارنيستند وكدي كه توليدمي كنندمعلوم نيست كه كارا باشد معتقدند ابزاري كه كد توليدمي كند چون خودش كد را توليدكرده كارايي چنداني ندارد و نگهداري از اين نرم افزارها ممكن است راحت نباشد.
- نظرموافقين: استفاده از اين ابزارهاكاهش زمان توليد و افزايش كارايي توليد نرم افزار را بدنبال دارد.

fourth generation techniques(4GT)

- وضعيت فعلي نرم افزارهايي كه ماهيت *4 GL* دارند:
- *4GL* براي سيستم هاي اطلاعاتي به كارمي رود كه پايگاه اطلاعاتي از آنها پشتيباني مي كنند و نهايتا آماري كه بدست آمده نشان مي دهد
- 1-استفاده از *4GL* كاهش قابل ملاحظه اي را در رابطه با زمان توليد نرم افزار در پي داشته است.
- 2-كاهش هزينه توليدنرم افزار را در پي داشته است.

استفاده از این ابزارها در پروژه های بزرگ زمان تحلیل و طراحی را کاهش نداده اما زمان کد نویسی را کاهش داده است.

- 3-بكارگيري *4GL* خصوصا زماني كه باروشهاي *component* *Assembly* ادغام شود بازدهي بسيارخوبي راخواهدداشت.

7)the formal methods

- روشهاي فرمال امكاني جهت *program Verification* را دارا مي باشند.براي اينكه بتوان صحت برنامه رااثبات نمود
- نوع خاصي ازاين روش *c lean room software engineering* مي باشد.در اين روش بايد ضمانتي وجودداشته باشد براي اينكه نرم افزاري كه قراراست توليدشودبدون خطاباشد.
- دربعضي ازسيستم ها توليدخطابحراني است مانند سيستم هواپيماكه كوچكترين خطائي غيرقابل جبران است دراين گونه موارد می توان ازروش *clean room engineering* استفاده نمود.
- روشهاي فرمال درمحيط هاي عملياتي به خوبي قابل استفاده نيستند و فقط دربرخي مواردبراي سيستم هاي حساس ازاين روش استفاده شده است.
- *Z-language* و *VDM* دونمونه زباني هستندكه براي مدل سازي با روشهاي فرمال بوجودآمده اند.

معايب روش فرمال

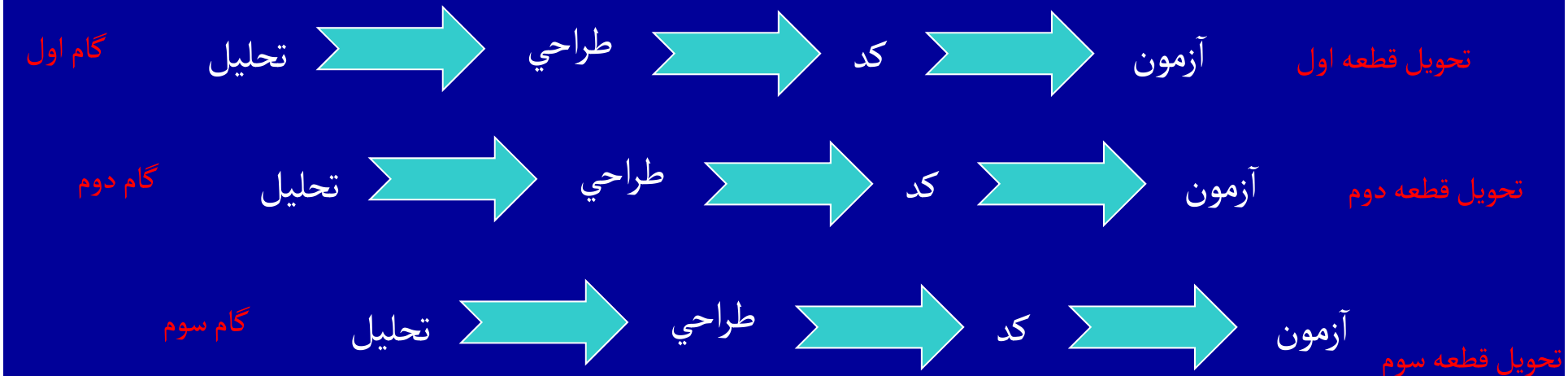
1. پياده سازي سيستم به اين روش دردنياي واقعي بسيارمشكل، پرهزينه وحتي درمواردي غيرممكن مي باشد.

2. براى اينكه سازمانها قادر به استفاده از اين روشها شوند ،زمان زيادي احتياج است تامهندسين نرم افزارآموزش هاي لازم رابيننددوهمين طورهزينه زيادي را هم درپي خواهد داشت.

3. باابزارهايي مانندتئوري مجموعه هاوازاين قبيل خصوصيات رياضي نمي توان به راحتي بامشتري ارتباط برقرارنمود.

مدل افزایشی

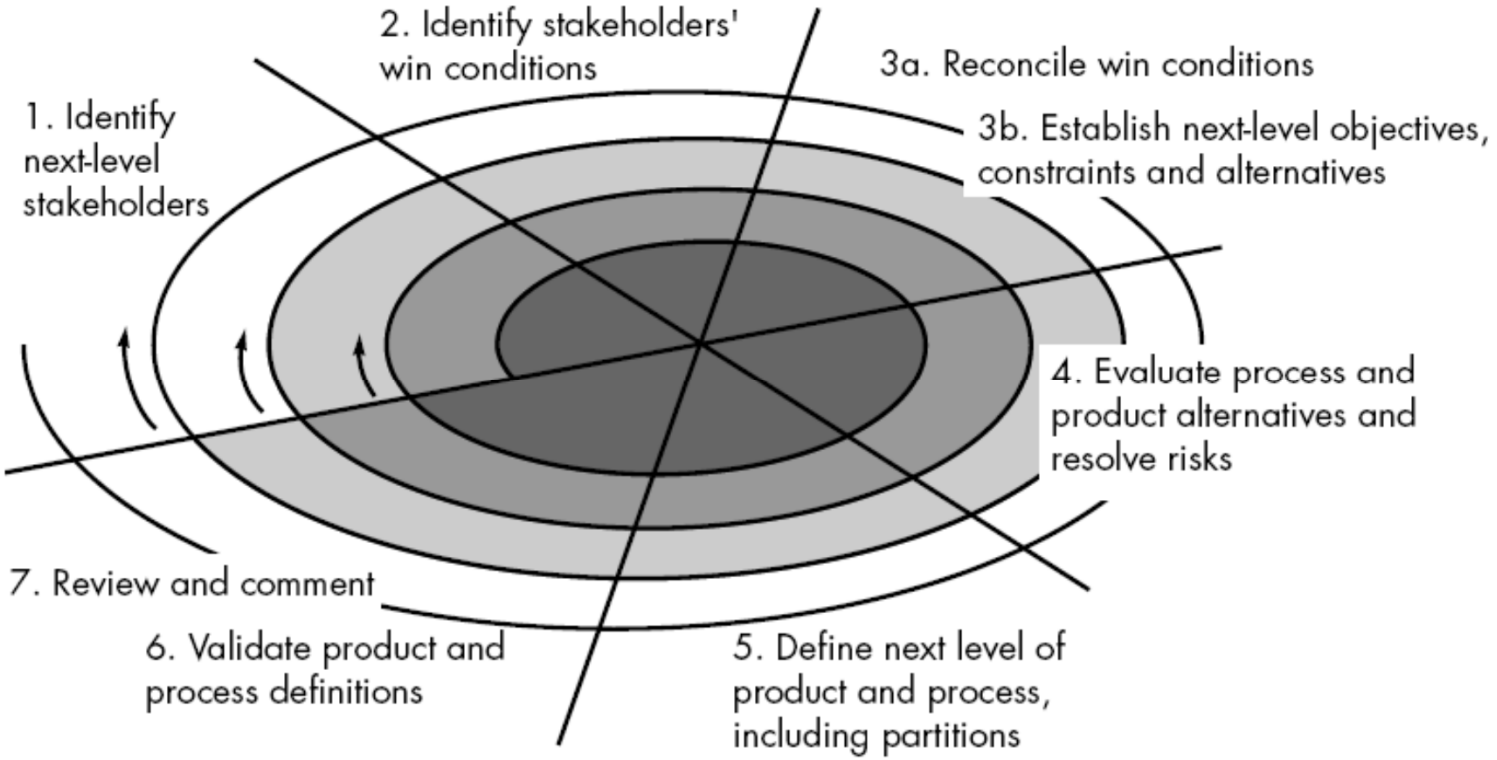
- ترکیب مدل خطی و مدل ساخت نمونه اولیه
- در انتهاي هر ترتيب خطی یک محصول از نرم افزار ارائه می گردد.اولین محصول با نام محصول هسته ای (Core Product) به نیازمندیهای پایه ای پرداخته و پس از بازنگری توسط کاربر اصلاح و بهینه می گردد.



مدل حلزونی برنده برنده(Win-Win)

- در بخش تعامل با مشتری نیازمندیها از سوی مشتری می بایست مشخص شوند .جهت این موضوع لازم است مشتری به یک موازنه (trade off) بین نیازمندیهای خود و تیم توسعه برسد. به عبارت دیگر موازنه ای بین عملکرد ، قابلیتهای سیستم و کارائی از طرفی و هزینه و زمان از سوی دیگر برقرار نماید. در این شرایط تلاش می گردد اکثر نیازمندیهای مشتری در مقابل زمان و قیمت مناسب جهت تیم توسعه دهنده نرم افزار فراهم گردد(برد-برد). در مدل مذکور به جای بخش تعامل با مشتری و تعیین نیازمندیها قسمتهای زیر جایگزین می گردد:
  - شناسائی واگذارنده و تعیین شرایط برد او
  - مذاکره جهت حصول به توافق (در راستای قاعده برد – برد)

مدل حلزونی برنده برنده(Win-Win)



مدل حلزونی برنده برنده(Win-Win)

Rather than a single customer communication activity, the following activities are defined:

1. Identification of the system or subsystem's key "stakeholders."
2. Determination of the stakeholders' "win onditions."
3. Negotiation of the stakeholders' win conditions to reconcile them into a set of win-win conditions for all concerned (including the software project team).

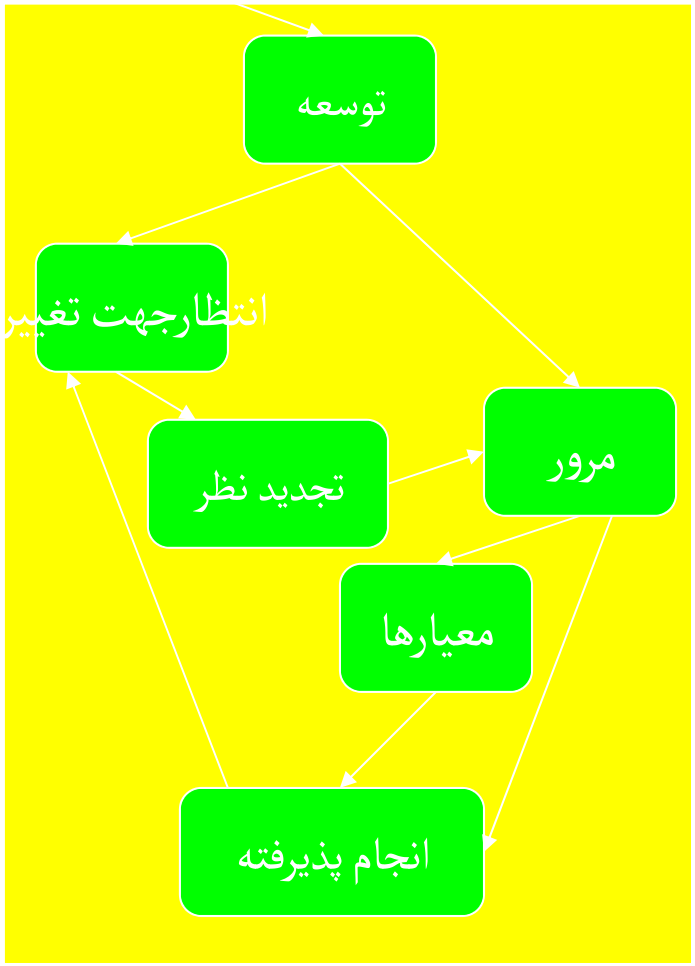
مدل توسعه همزمان (مهندسی همزمانی)

- در این مدل هر فعالیت دارای چندین حالت می باشد که با تعریف مجموعه ای از رخدادها شاهد گذار از حالتی به حالت دیگر خواهیم بود. به بیان دیگر تمامی فعالیتها بصورت همزمان وجود دارند لیکن در حالتهای متفاوتی قرار می گیرند.
- Project managers who track project status in terms of the major phases [of the classic life cycle] have no idea of the status of their projects.

مدل توسعه همزمان (مهندسی همزمانی)

- although . . . [a large] project is in the coding phase, there are personnel on the project involved in activities typically associated with many phases of development simultaneously. For example,. . . personnel are writing requirements, designing, coding, testing, and integration testing [all at the same time].

مدل توسعه همزمان (مهندسی همزمانی)



- به عنوان مثال بعد از اتمام فعالیت تعامل با کاربر این فعالیت در حالت هیچ(none) رفته و فعالیت تحلیل از حالت هیچ به حالت توسعه وارد می گردد و حالتهای بعدی خود را طی می نماید. چنانچه کاربر احتیاج به انجام تغییرات در نیازمندیهای خود داشته باشد فعالیت تحلیل به حالت انتظار جهت تغییر می رود.

مدل توسعه همزمان (مهندسی همزمانی)

- The concurrent process model defines a series of events that will trigger transitions from state to state for each of the software engineering activities.
- For example, during early stages of design, an inconsistency in the analysis model is uncovered.This generates the event *analysis model correction* which will trigger the *analysis* activity from the **done** state into the **awaiting changes** state.

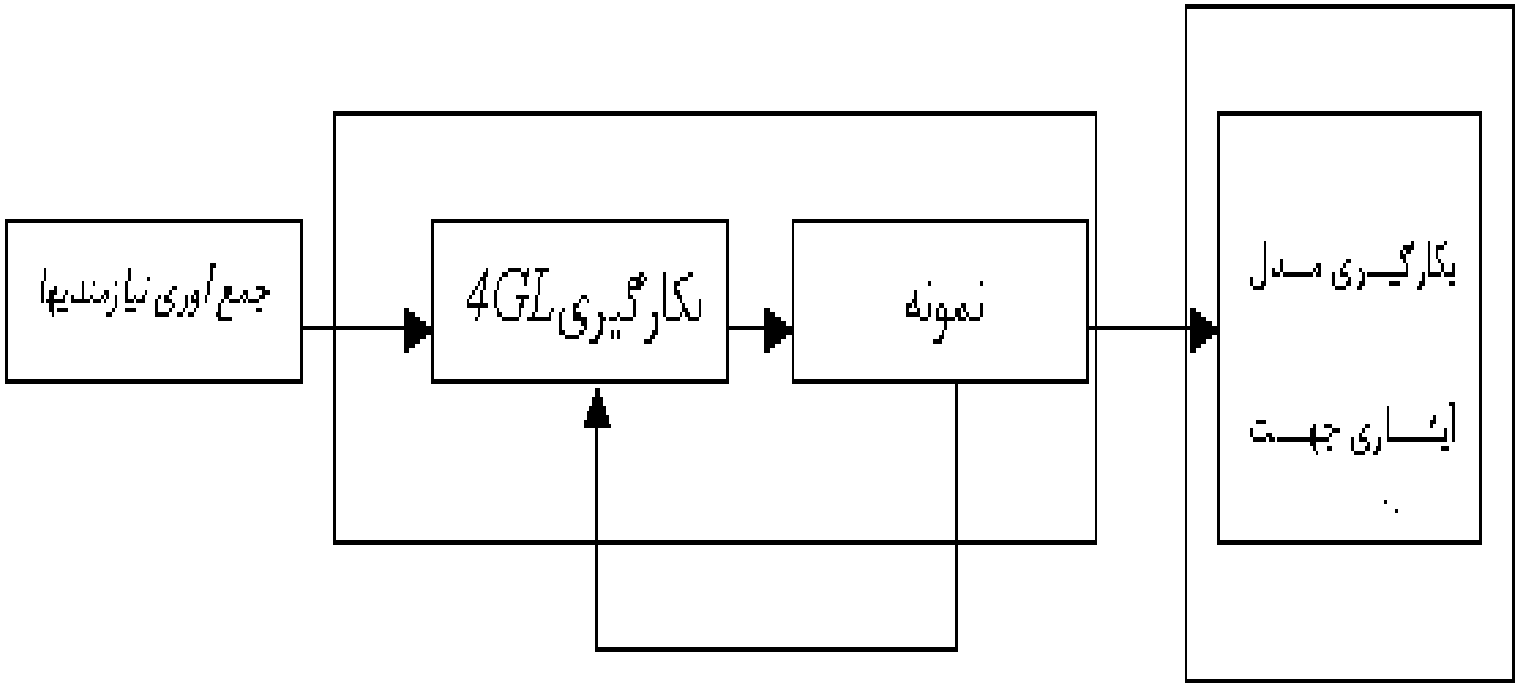
مدل توسعه همزمان (مهندسی همزمانی)

- In reality, the concurrent process model is applicable to all types of software development confining software engineering activities to a sequence of events, it defines a network of activities. Each activity on the network exists simultaneously with other activities.

تركيب الگوها

- درمتدلوژيهاي معتبرامروزي معمولاازيكي ازاين الگوهابه صورت كامل استفاده نمي شودبلكه دربسياري ازمواردازتركيب يك ياچندالگواستفاده مي شودبه اين ترتيب هركدام نقاط ضعف ديگري راپوشش مي دهند.

مثال)



نمونه سازی

Analysis modeling

روش هاي آناليز و طراحي نرم افزار به دو دسته كلي تقسيم مي شود :

- روش هاي *structural*
- روش هاي *object oriented*

- اهداف آناليز

1-شرح نيازمنديهاي سيستم و مشتري

2-پايه گذاري بنيادي جهت توليد نرم افزار

3-تعريف مجموعه اي از نيازمنديها كه با ايجاد نرم افزار به آنها پاسخ داده مي شود

روش هاي *structural* و طراحي سيستم :

اولين بار شخصي به نام *doglas.rose* اين روش را بنيانگذاري نمودو پس از آن *demarco* پيشنهاد هاي زير را مطرح نمود:

- محصول توليد شده براي آناليز بايد قابل پشتيباني باشد.
- مسائل با اندازه و حجم بزرگ بايد به مسائل كوچكتر شكسته شوند.
- در كاري كه انجام مي دهيم بايد بتوان بين بخش هاي *logical* و *physical* تفكيك قائل شد.
- هر كجا لازم است بايد از نمادهاي گرافيكي استفاده نمود.

پس حداقل نياز داريم به:

- چيزی كه به ما كمک كند تا نيازمنديهايمان را تکه تکه نماييم و هر قسمت را قبل از تغيير مستند نمائيم.
- روشی جهت دنبال كردن و ارزيابی Interfaceها.
- و در نهايت نياز به ابزاري جديد جهت شرح منطق و سياست داريم كه اين ابزار از نوشتار معمولي بهتر باشد(مثلاً *data flow diagram*) .

تاریخچه ای از برخی از افراد که در این زمینه کار کرده اند:

- Douglas Ross
- Demarco
- Gane and Sarson
- Ward and Mellor
- Hatley and Pirbhai

روش هاي *structural* و طراحي سيستم :

بخش هاي يك مدل آناليز :

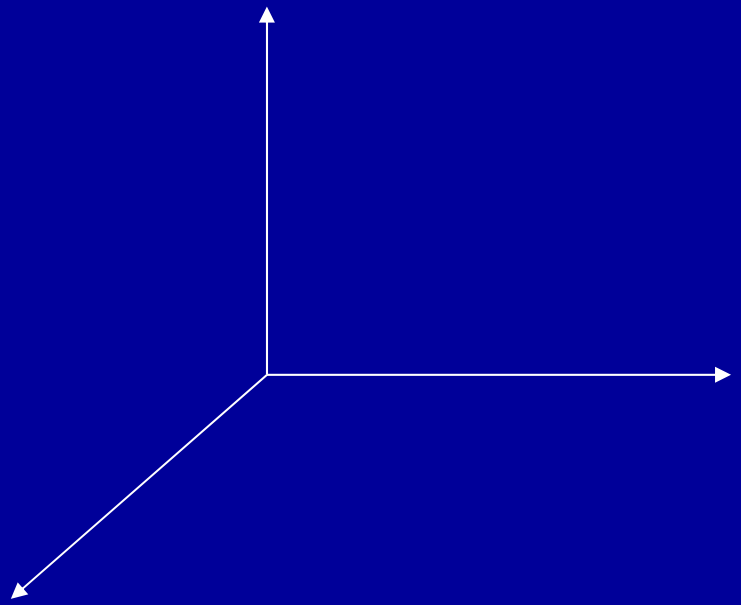
- وقتي مي خواهيم يك سيستم را مدل كنيم معمولاً از چند زاويه مختلف به آن نگاه مي كنيم و تركيب اين زاويه ها يك مدل واقعي براي ذهن ما ايجاد مي كند. در تجزيه و تحليل نرم افزار هم به همين طريق عمل مي كنيم.در حالت كلي براي انجام آناليز و طراحي سيستم هاي نرم افزاري از سه ديدگاه اساسي به سيستم نگاه مي كنيم . به عبارتي مي توان گفت، سه عينك مختلف به چشمان خود مي زنيم كه هر كدام سيستم را از يك جنبه خاص به ما نشان مي دهد.
- اين سه ديدگاه عبارتند از :

1-مدل سازي سيستم از ديدگاه ساختاري

2-مدل سازي سيستم از ديدگاه رفتاري

3-مدل سازي سيستم از ديدگاه رفتار پوياي سيستم

مدل سازي از ديدگاه رفتار پوياي سيستم



مدل سازی از ديدگاه ساختاری

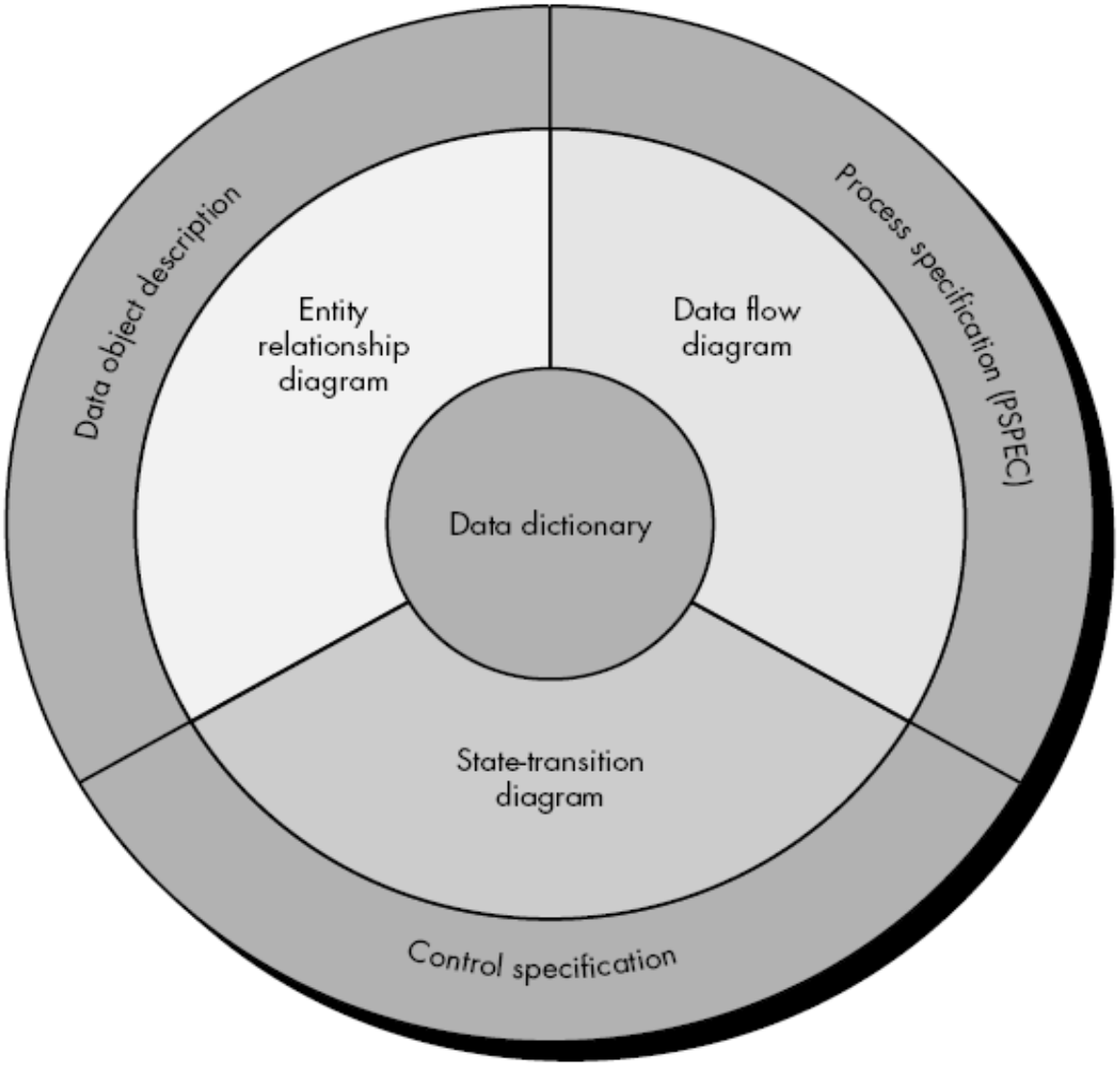
مدل سازي ازديدگاه رفتاري

1- در اين ديدگاه ساختار و استخوان بندي اطلاعات مورد نياز در سيستم مشخص مي گردد، نظير موجوديتها و ارتباط آنها با يكديگر. به عنوان مثال در ديدگاه ساختاري مي توان به نمودار *ERD* در ديدگاه *structural* و نمودار *class diagram* در ديدگاه *object oriented* اشاره نمود .

2- در این دیدگاه فرآیند کار یا نحوه عملیات انجام شده در سیستم مدل مي گردد. به عنوان نمونه اي در این زمینه مي توان به نمودار *DFD* در دیدگاه *structural* و به نمودار *use case diagram* و Activity Diagramدر دیدگاه *object oriented* اشاره نمود .

3- در اين ديدگاه مشخص مي گردد كه هر فرآيند در سيستم چه زماني يا به واسطة چه رخدادي فعال مي گردد بعنوان نمونه مي توان به نمودارهاي *ELH* و *ECD*در متدلوژي *SSADM* ، *triggre* ها در *process model* در *designer2000* و *collaboration diagram* ، *state diagram* ، *sequence* *Diagram*و *activity diagram*(تا حدودی) در ديدگاه *object oriented* اشاره نمود.

The elements of the Analysis Model



در این نمودار:

دیدگاه ساختاري با *(Entity relationship diagram ERD)*

دیدگاه رفتاري با *Data flow diagram (DFD)*

ودیدگاه رفتار پوياي سيستم با *State transition (STD)* *diagram* نشان داده شده است .

Data dictionary

در مرکز سیستم *Data dictionary* یا کاتالوگ سیستم *Data* مشاهده مي شود که در این قسمت مشخصات تمامي *objecte* ها و ارتباط آنها و بطور کلي تمامي اطلاعات مربوط به سیستم نگهداري مي شود .

Data Flow Diagram

در Data Flow Modeling(DFM)به نمایش نحوه انتقال داد ه ها در سیستم می پردازیم.هدف اصلی از رسم دیاگرام DFD مشخص نمودن Function ها SubFunctionها درسیستم می باشد.

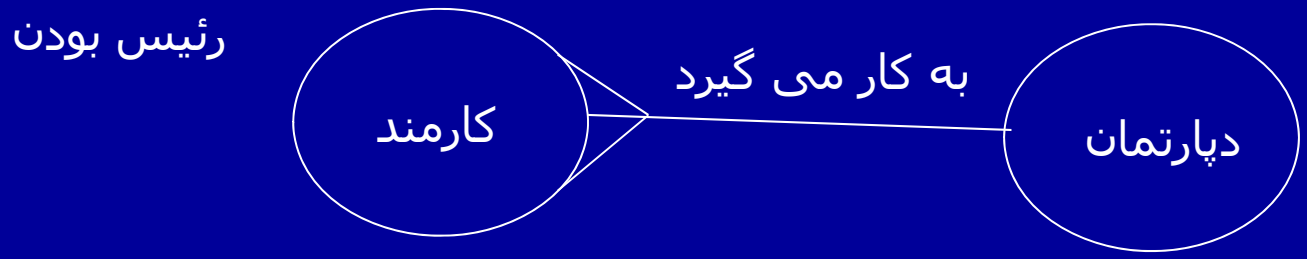
State transition diagram

در مدل سازي سيستم از ديدگاه رخدادها از *STD* استفاده مي شود.STD بيانگر چگونگی رفتار سيستم در مقابل رخدادهای خارجی می باشد. در SSADM برای اين کار از دياگرام ELH استفاده می شود.

Data modeling

در *Data modeling* يا مدل سازي سيستم از ديدگاه داده اي ، به رسم يکی ازمهم ترين ديا‌گرام ها يعني *ERD* مي پردازيم . در *ERD* موجوديت ها يا *entity* ها و ارتباط آنها با يکديگر مشخص خواهد شد.

مثال :



Emp (detail)

Emp#	ename	Sal	Dept #
100	Ali	3000	10
200	Reza	2100	20
300	Mina	2400	20
400	Leyla	2600	30

Dept (master)

Dept #	dname	Loc
10	Com	Teh
20	Math	Tab
30	Phys	Shir
40	Shimi	Teh

نکته: *ERD* صرفاً بر روي داده تمرکز مي کند و در نهايت به شکل شبکه اي از داده هاي مرتبط به هم نمايش داده خواهد شد. *ERD*براي رسم *DFD* نيزبسيار مفيد خواهد بود و لازم به ذکر است که حداقل در مدل *structural* رسم *ERD* به صورت مستقل و بدون توجه به پردازشي که قرار است روي آن انجام شود صورت مي گيرد.

Data modeling

در *Data modeling* به سوالات زیر پاسخ داده خواهد شد:

- چه هستند entity های اولیه
- از چه Attributeهائی تشکیل شده اند.
- کجا قرار گرفته اند؟
- چه ارتباطی با یکدیگر دارند.

Attribute

Attribute ها به بيان خصوصيت هاي *data object* ها مي پردازند و به سه دسته زير تقسيم مي شوند :

1- *attribute* هايي كه براي نامگذاري *instance* هستند .

2- *attribute* هايي كه براي شرح *instance* هستند.

3- *attribute* هايي كه براي ارتباط يك *instance* با *instance* ديگر بكار مي روند.

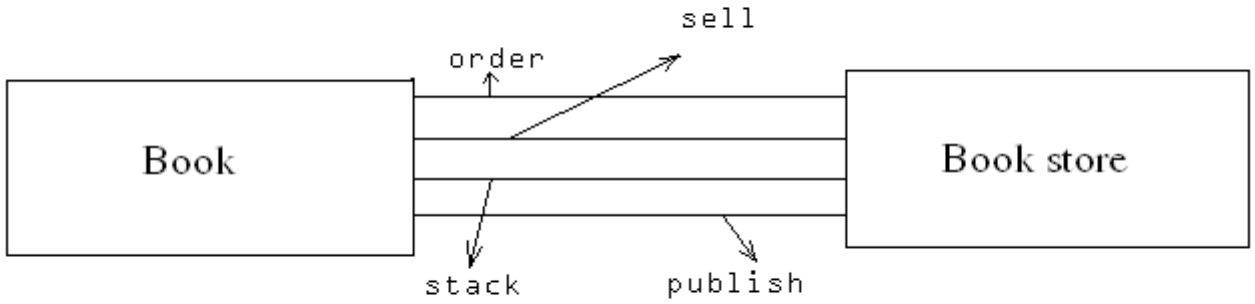
Data objects

Data object در حقیقت همان *entity* یا موجودیت است. به عبارت دیگر *Data object* نمایشي از تمام اطلاعات *composite* یا مرکب است که نرم افزار به آن نیاز دارد .

- *Instance* به معناي نمونه اي از يك *data object* است .

Relation

بين دو موجوديت ممكن است ارتباط هاي گوناگوني برقرار باشد مثلاً بين دو موجوديت كتاب و كتاب فروشي ارتباط هاي زير برقرار است:



Cardinality and modality

Cardinality به معني چندي ارتباط است . يعني يك رخداد از يك *object* با چند رخداد از *object* ديگر در ارتباط است

Cardinality سه حالت دارد :

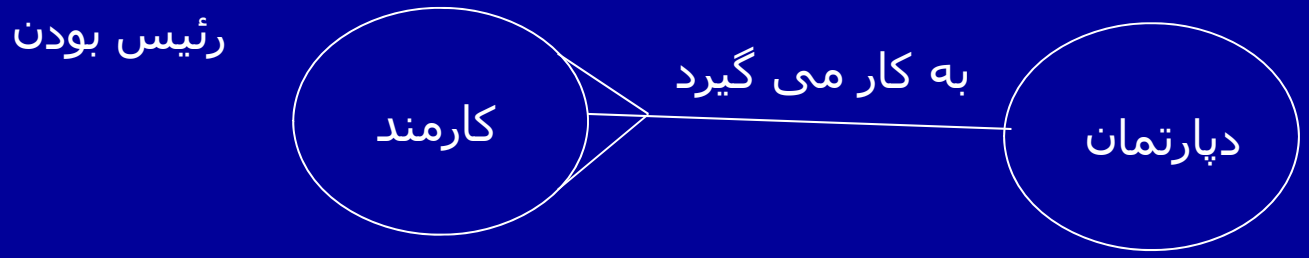
1 - يك به يك 1 : 1

2 - يك به چند 1 : n

3 - چند به چند n : n

1 : n　　1 : 1　　n : n

مثال :



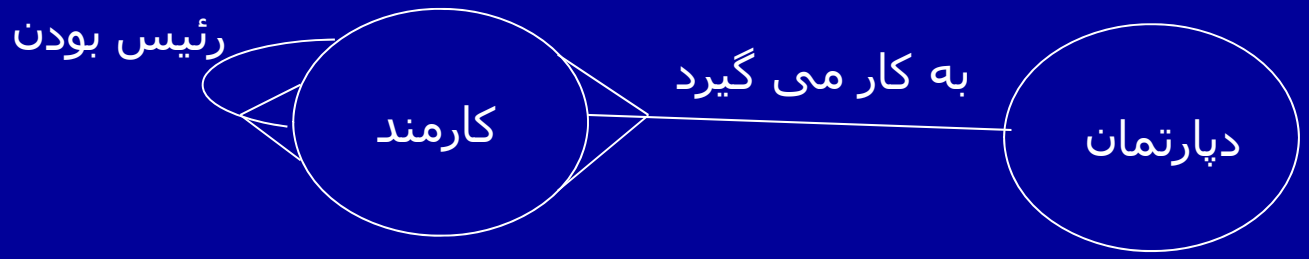
Emp (detail)

Emp#	ename	Sal	Dept #
100	Ali	3000	10
200	Reza	2100	20
300	Mina	2400	20
400	Leyla	2600	30

Dept (master)

Dept #	dname	Loc
10	Com	Teh
20	Math	Tab
30	Phys	Shir
40	Shimi	Teh

مثال :



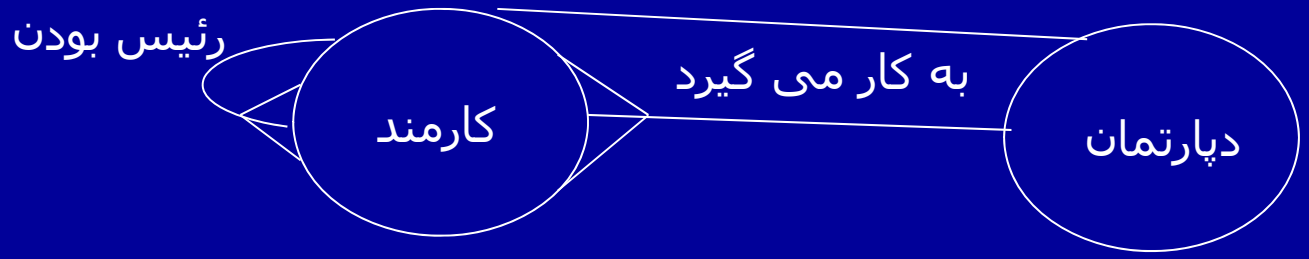
Emp (detail)

Emp#	ename	Sal	Dept #	Mgr#
100	Ali	3000	10	Null
200	Reza	2100	20	100
300	Mina	2400	20	100
400	Leyla	2600	30	300

Dept (master)

Dept #	dname	Loc
10	Com	Teh
20	Math	Tab
30	Phys	Shir
40	Shimi	Teh

مثال :



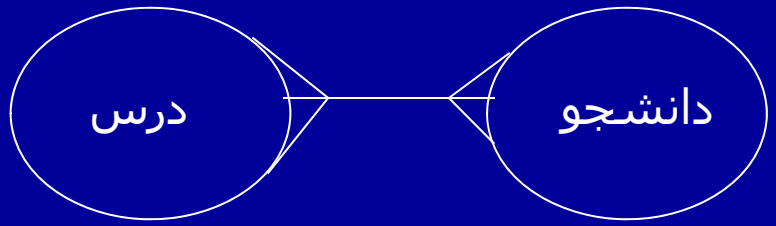
Emp (detail)

Emp#	ename	Sal	Dept #	Mgr#
100	Ali	3000	10	Null
200	Reza	2100	20	100
300	Mina	2400	20	100
400	Leyla	2600	30	300

Dept (master)

Dept #	dname	Loc	Mgr#
10	Com	Teh	100
20	Math	Tab	200
30	Phys	Shir	300
40	Shimi	Teh	400

مثال :

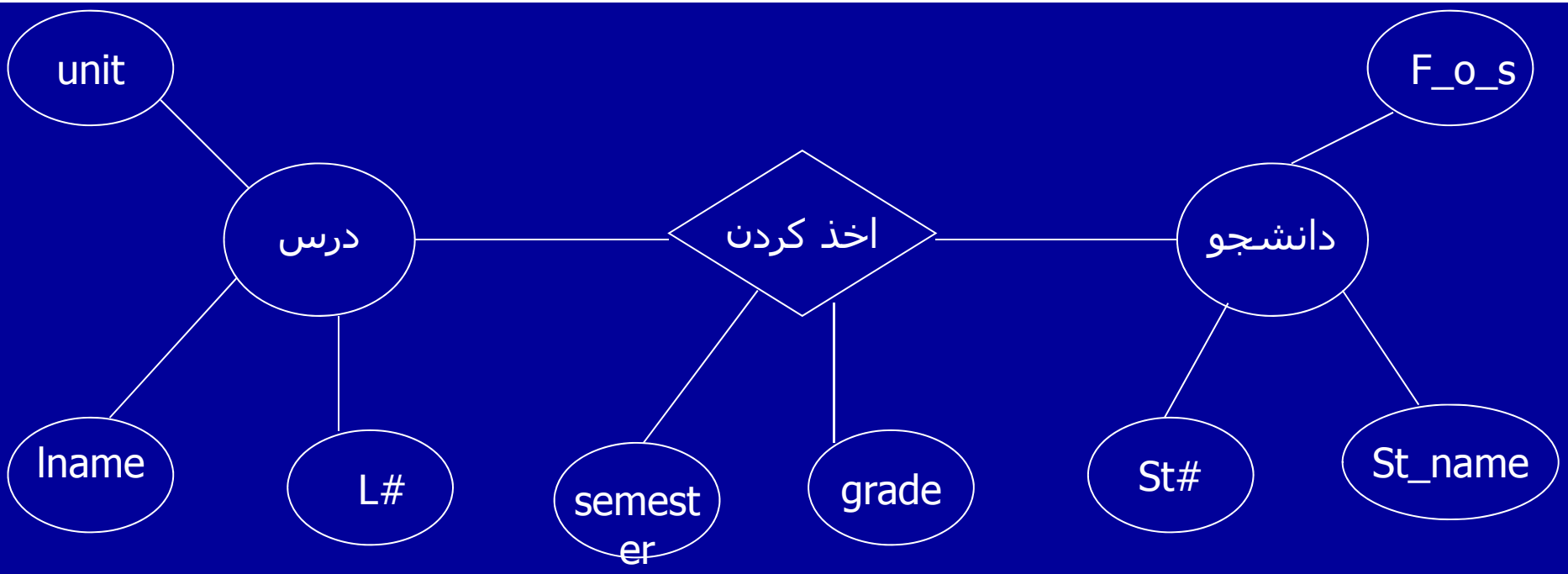


Lesson

L#	Lname	unit
10	DB	3
20	chemistry	2
30	S_e	3

st

st#	St_name	F_o_s
100	Ali	com
200	Mina	‘’
300	Leyla	‘’
400	Amir	physics



Lesson

L#	lnam e	unit
10	DB	3
20	che mist ry	2
30	S_e	3

Lesson_st

L#	St#	semester	grad e
10	100	82-1	15
20	200	83-1	14
30	100	82-1	7
30	300	82-2	18

st

St#	stna me	F_O_ S
100	Ali	com
200	Mina	,,
300	Leyla	,,
400	Amir	physi cs

مثال:حالتی که بین دو موجودیت رابطه 1:1 برقرار است و.....

چون عامل ارتباطی دارد

Emp(master)

Emp#	ename	sal
100	Ali	3000
200	Reza	4000
300	Mina	4500
400	Leyla	5000

Couple(detail)

emp#	C_name	C_job
100	Tina	clerk
200	Leyla	teacher
400	Amir	Engineer

Emp#	ename	Sal	C_name	C_job
100	Ali	3000	Tina	clerk
200	Reza	4000	Leyla	teacher
300	Mina	4500	Null	Null
400	Leyla	5000	Amir	Enginieer

نکته : در مثال قبل اگر تعداد زیادی از کارمندان مجرد هستند بهتر است از حالت اول استفاده کنیم .

نکته : ارتباط عادی مورد قبول بین دو موجودیت در پایگاه داده ها ارتباط از نوع یک به چند می باشد . چنانچه بین دو موجودیت ارتباط از نوع چند به چند باشد بنا به مشکلاتی که پیش می آید لازم است که آن ارتباط چند به چند به دو ارتباط یک به چند تبدیل گردد . از طریق موجودیت واسط به گونه ای که توضیح داده شد .

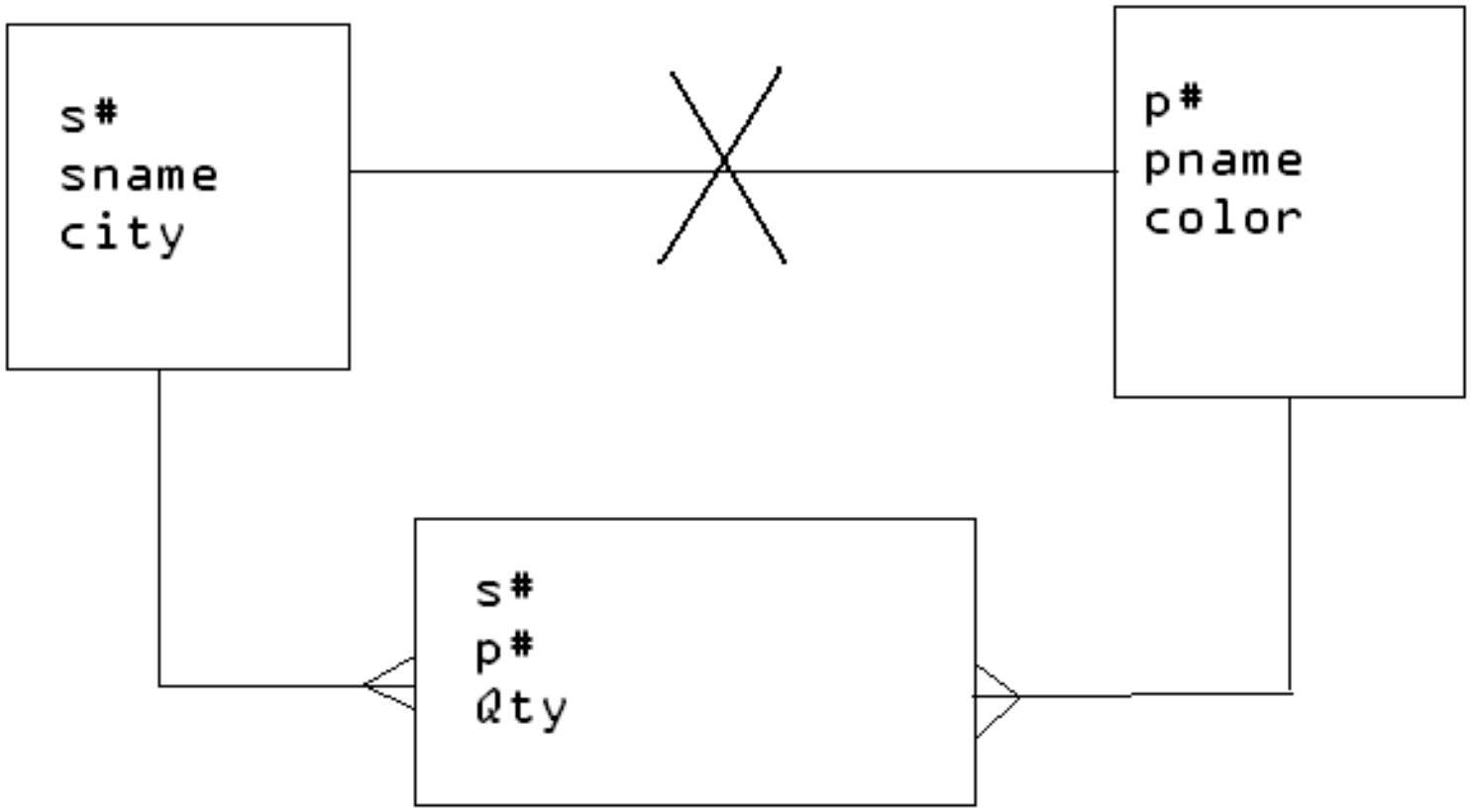
در ارتباط یک به یک نیز همانگونه که ذکر شد در بیشتر مواقع اگر ارتباط یک به یک وجود داشته باشد به احتمال زیاد هر دو آنها می توانند یک موجودیت باشد . ولی گاهی ممکن است بنا به دلایلی مثلا صرفه جویی در حافظه یا افزایش سرعت یک موجودیت را دو موجودیت کنیم . ولی در حالت کلی بجز موارد استثناء اگر بین دو موجودیت ارتباط یک به یک باشد می توان آنها را یک موجودیت در نظر گرفت .

نکته : همیشه در ارتباطات 1 :n سمت یک را موجودیت master و سمت چند ارتباط را موجودیت detail می نامیم .

نکته : ممکن است یک موجودیت با خودش هم ارتباط داشته باشد که به آن رابطه گوش فیلی نیز گویند .

نکته : اگر در بین دو موجودیت هم رابطه 1:n و هم رابطه 1:1 باشد آن دو را نمی توان یک موجودیت دانست .

مثال:

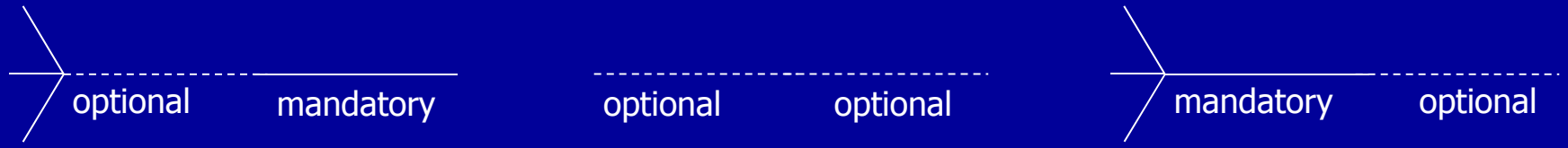


Modality

Modality در ارتباط بين دو موجوديت(يا يک موجوديت و خودش) بارز است (اگر 1 باشد *mandatory* و اگر 0 باشد *optional* است) يعني ارتباط اجباري باشد يا اختياري .

اختياري يا اجباري بودن سمت *detail* حائز اهميت است و بر توليد کد تاثير مي گذارد.(درمثال کارمند و شرکت، کارمند قسمت *detail* است) .

رابطه mandatory و optional :



مثال :

Emp (detail)

Emp#	ename	Sal	Dept #	Mgr#
100	Ali	3000	10	Null
200	Reza	2100	20	100
300	Mina	2400	20	100
400	Leyla	2600	30	300

رئیس داشتن

رئیس دپارتمان بودن

به کار می گیرد

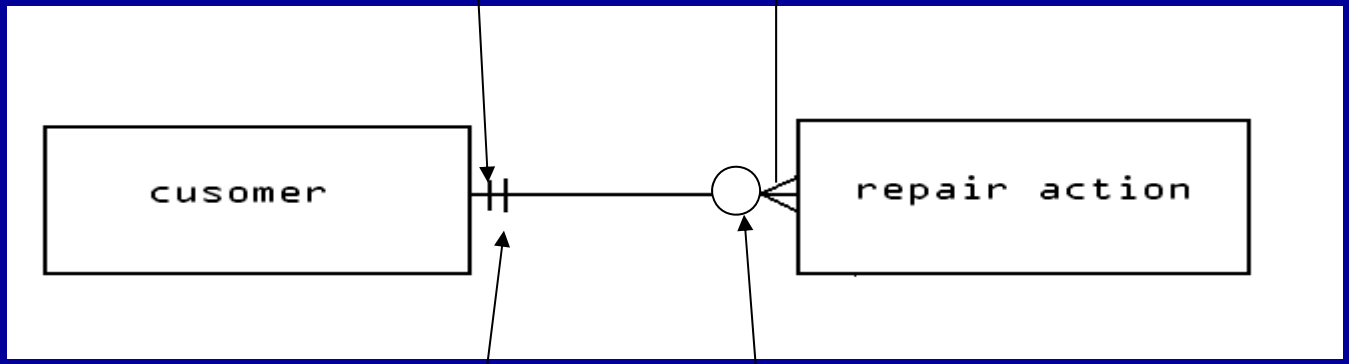
Dept (master)

Dept #	dname	Loc	Mgr#
10	Com	Teh	100
20	Math	Tab	200
30	Phys	Shir	300
40	Shimi	Teh	400

(مثال)

Cardinality:Implies that there may be many repair actions

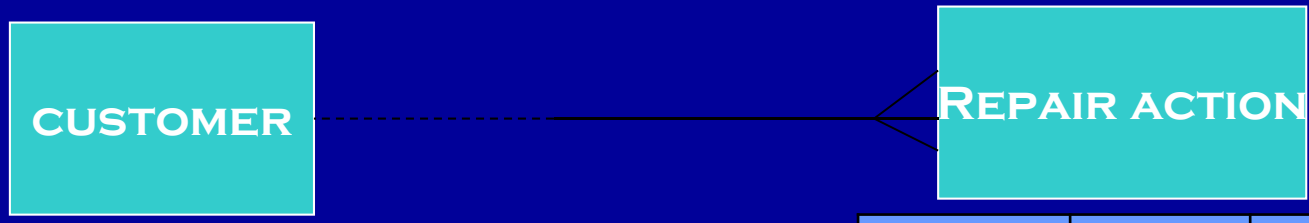
Cardinality:implies that a single customer awaits repair action(S)



Modality:mandatory implies that In order to have a repair Action(S) We must have a customer

Modality:Optional implies that There may be a situation in which A repair action is not necessary

مثال)

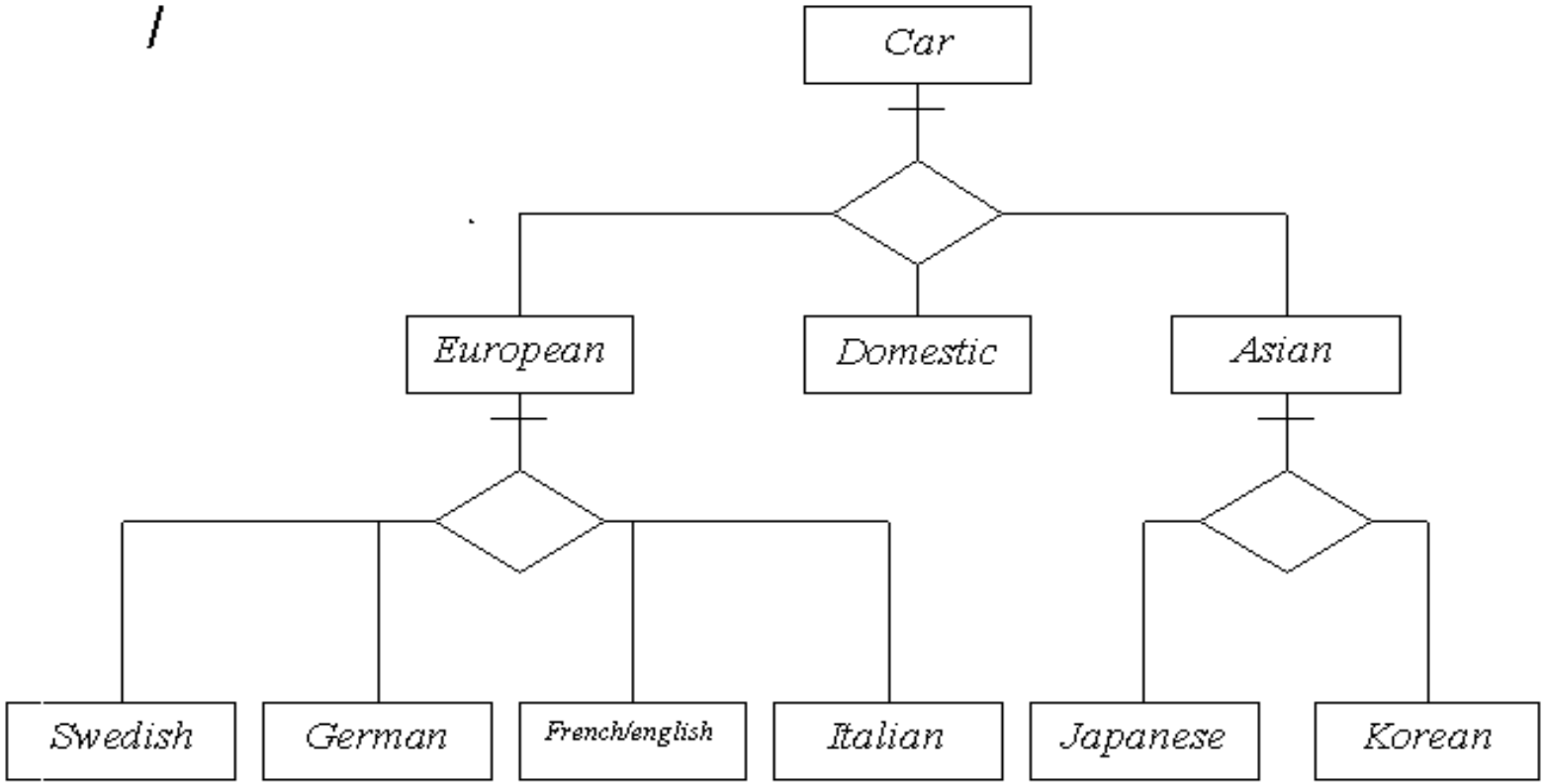


id	name	address	tel
100	Ali	tehran ….	2345677
200	Reza	Karaj …	43543098
300	Mina	Shemiran…	2243456

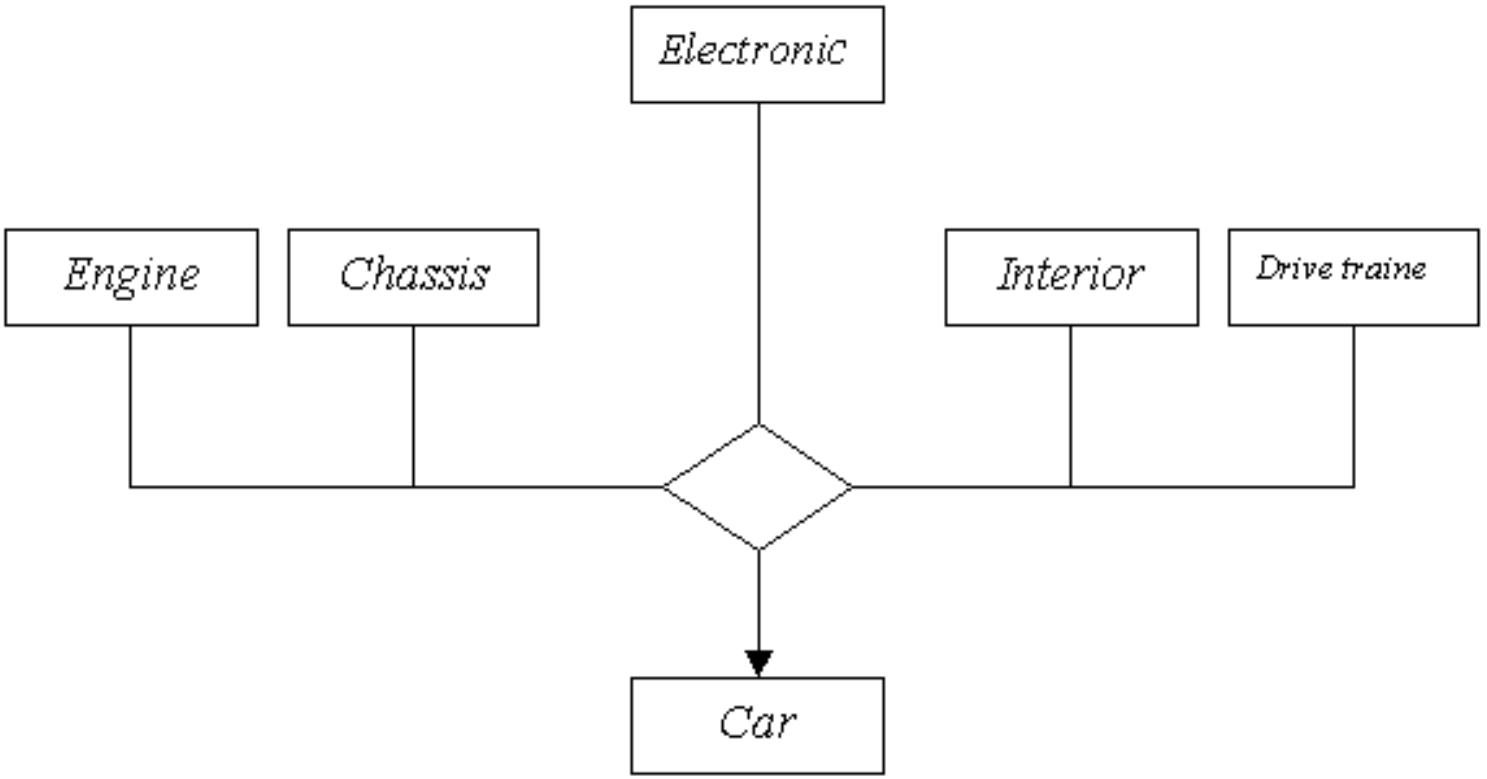
Id	Customer	Time	Service.e
100	100	شنبه 28/8/82	آب گرمکن
120	200	شنبه 28/8/82	آب گرمکن
150	100	شنبه 28/8/82	تعمیر کولر

نمودار سلسله مراتبي *data object* ها

مثال :

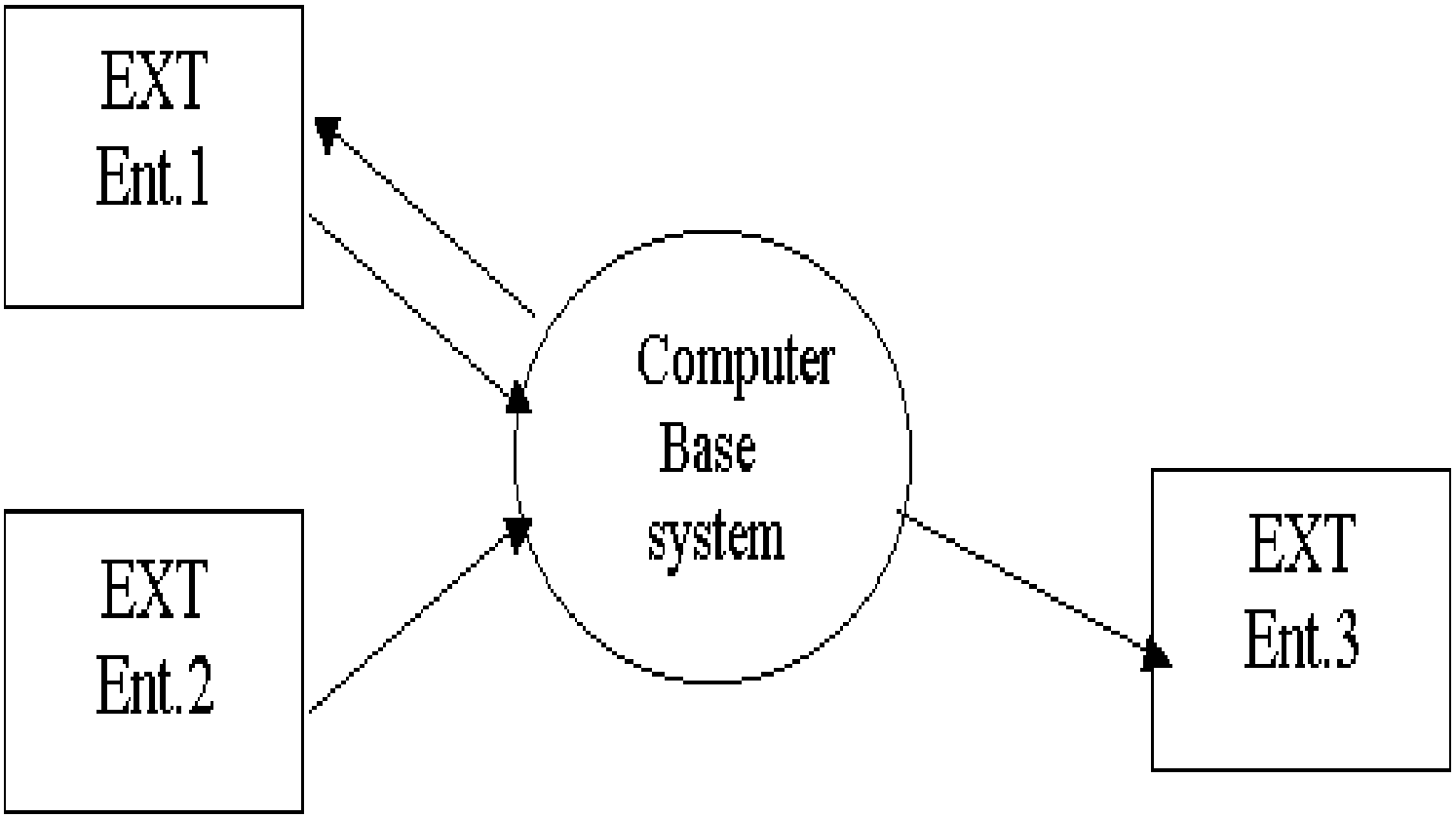


associating data object

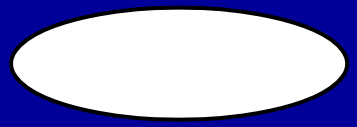
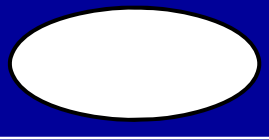
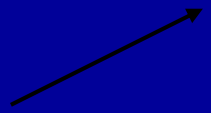
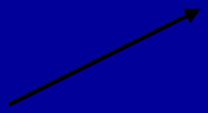
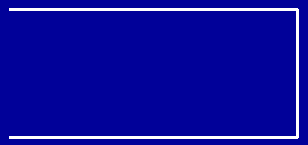
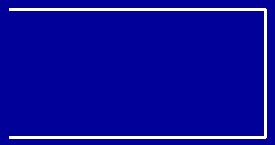


Data flow diagram (DFD)

- *Dfd* براي مدل سازي سيستم از ديدگاه *f unctional* استفاده مي شود. مهم ترين محصولي كه در *dfm* مطرح است نمودار *dfd* مي باشد. *Dfd* عملاً يك نماد گرافيكي است كه جريان اطلاعات را از ورودي به خروجي نشان مي دهد، عملاً از آن جهت *f unction modeling* در سيستم استفاده مي شود.
- در *dfd* ، به اولين سطح ، *dfd* سطح صفر يا *context diagram* مي گويند، در *dfd* سطح صفر، كل سيستم را به صورت يك *bubble* در مركز در نظر مي گيريم كه موجوديتهاي خارجي اطلاعاتي به آن مي دهند يا از آن مي گيرند


EXT
Ent.1
EXT
Ent.2
Computer
Base
system
EXT
Ent.3

استانداردهای رسم dfd

Pressman	Designer2000/ SSADM	Name
		External entity
		Process
		Data flow/ Material flow
		Store/data store store/manual store store/transient store

- *Ext.entity*: يك توليد كننده يا مصرف كننده اطلاعات است كه خارج از محدوده سيستمي كه هدف مدل سازي آن است قرار گرفته است.

- *Process* : يك انتقال دهنده اطلاعات يا يك *function* مي باشد كه در محدوده سيستمي كه هدف مكانيزه كردن آن است قرار گرفته است مثل پروسة ثبت نام يا ثبت مشخصات دانشجو (هر *process* قابل مكانيزه كردن نيست مثل اخذ برگه انتخاب واحد كه يك *process* است اما در دنياي واقعي نمي توان آن را مكانيزه نمود).

- ***flow*** *:* عملاً به شكل يك پيكان جهت دار مي باشد كه جهت آن، جهت انتقال اطلاعات را در سيستم مشخص مي كند.اگر از آن جهت انتقال داده ای در سيستم استفاده شود (در انتقال بين بخش هاي مختلف سيستم) ازآن بعنوان *data flow نامبرده می شود* و اگر از آن جهت انتقال چيزی بجزداده در سيستم استفاده شود(مثل پرونده كاغذي دانشجو يا يك چك) ازآن بعنوان *Material flow نامبرده می شود*.

- ***Store*** *:* يك ا نباره از داده ها مي باشد كه توسط يك يا چند *process*از سيستم مورد استفاده قرار مي گيرد ممكن است به سادگي يك *b uffer* و يا به پيچيدگي يك *data base*باشد. Store ممكن است به اشكال زير وجود داشته باشد:

- *Data store* : محلي است كه داده هاي به فرمت مكانيزه در آن ذخيره مي شود. مثل اطلاعات يك دانشجو زماني كه در يك سيستم مكا نيزه ذخيره مي گردد.

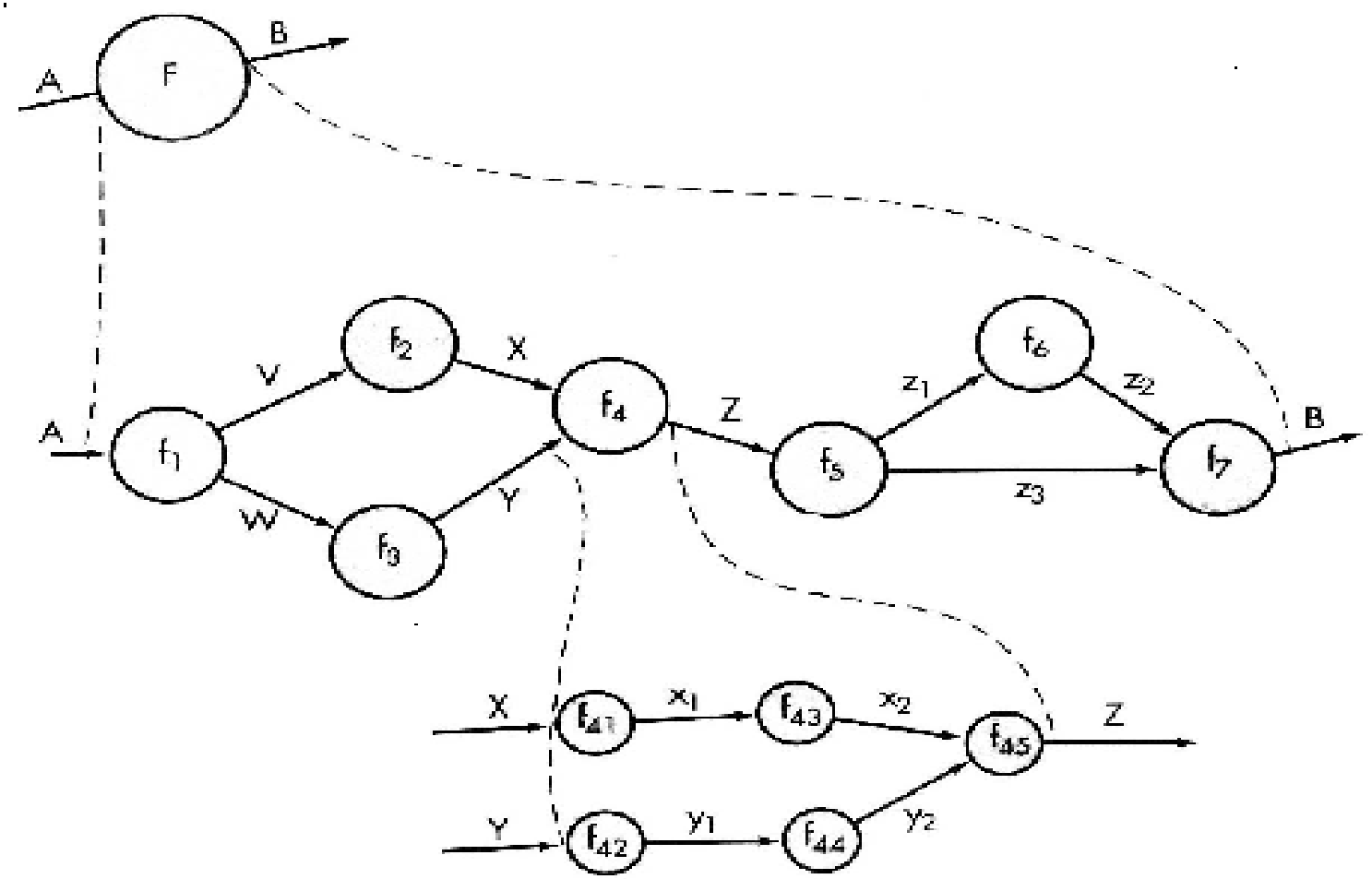
- *Manual store* :شامل *item* هايي است كه بصورت *manual* يا دستي ذخيره مي شود مثل برگه تعهدي كه از دانشجو اخذ مي گردد.

Transient store : *Store* هاي موقتي هستند كه ممكن است به فرم *Transient data store* يا *Transient manual store* باشد مثلاً محلي كه جهت *sort* كردن داده ها مورد استفاده قرارمي گيرد

يك *Transient data store* است وبه عنوان مثالي براي *Transient manual store* مي توان به جعبه اي كه كنار كارمند بانك قرار گرفته و فيش ها و چك ها را موقتاً در آن قرار مي دهد اشاره نمود

- از ويژگي هاي *dfd* اين است كه به صورت پله به پله جلو مي رود و هر سطح نسبت به سطح قبل كامل تر عمل مي كند. هميشه در سطح اول سيستم را بسيار كلان مي بينيم و در سطوح بعدي كه سيستم شكسته مي شود اطلاعات جزئي تري از سيستم براي ما مشخص مي شود. پروسس هاي آخرين سطح براي تبديل شدن به يك *f unction* كانديد مي شوند.

مثال)

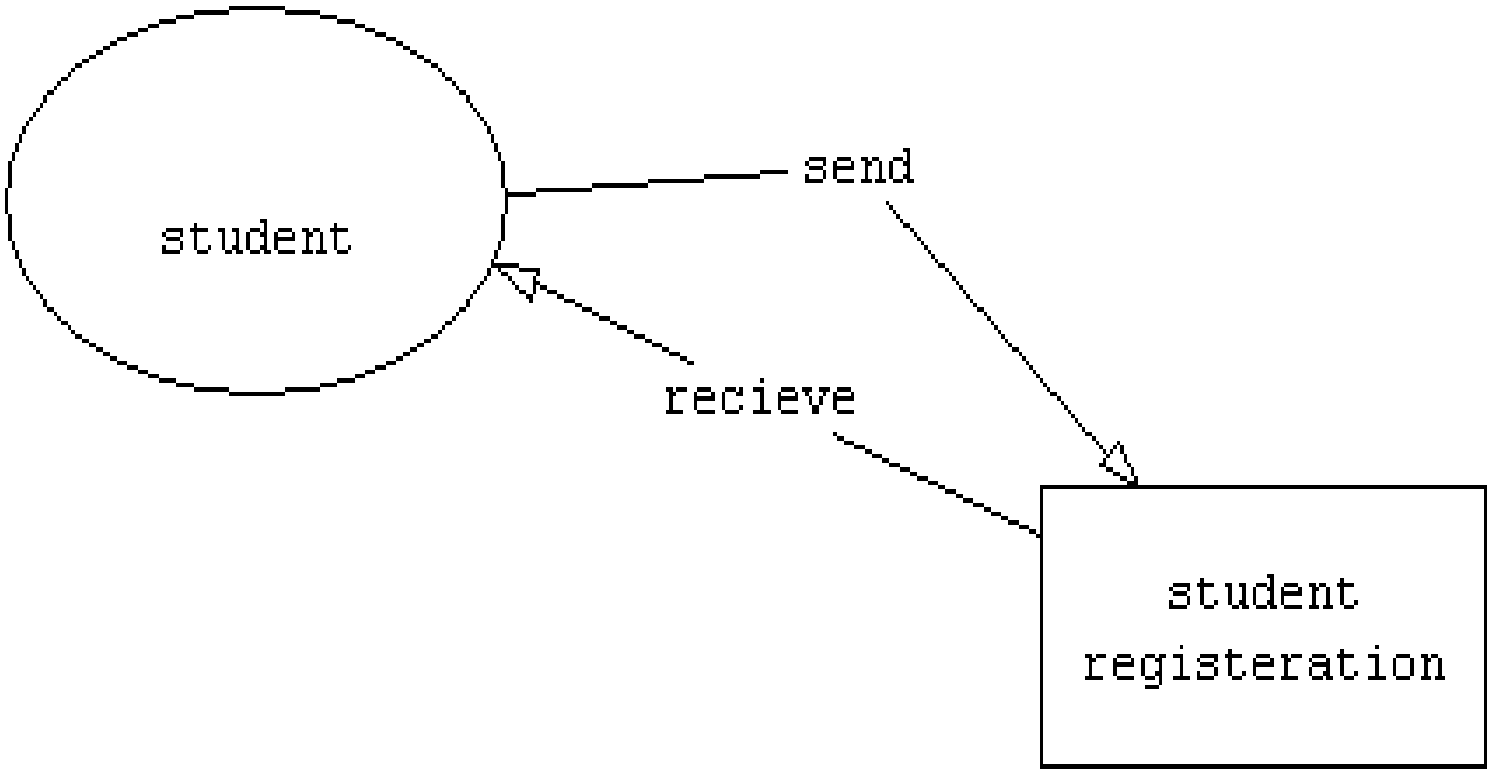

A
F
B
A
f1
V
W
f2
f3
X
Y
f4
Z
f5
z1
z2
z3
f6
f7
B
X
f41
x1
f43
x2
f45
Z
Y
f42
y1
f44
y2

نکات موجود در شکل

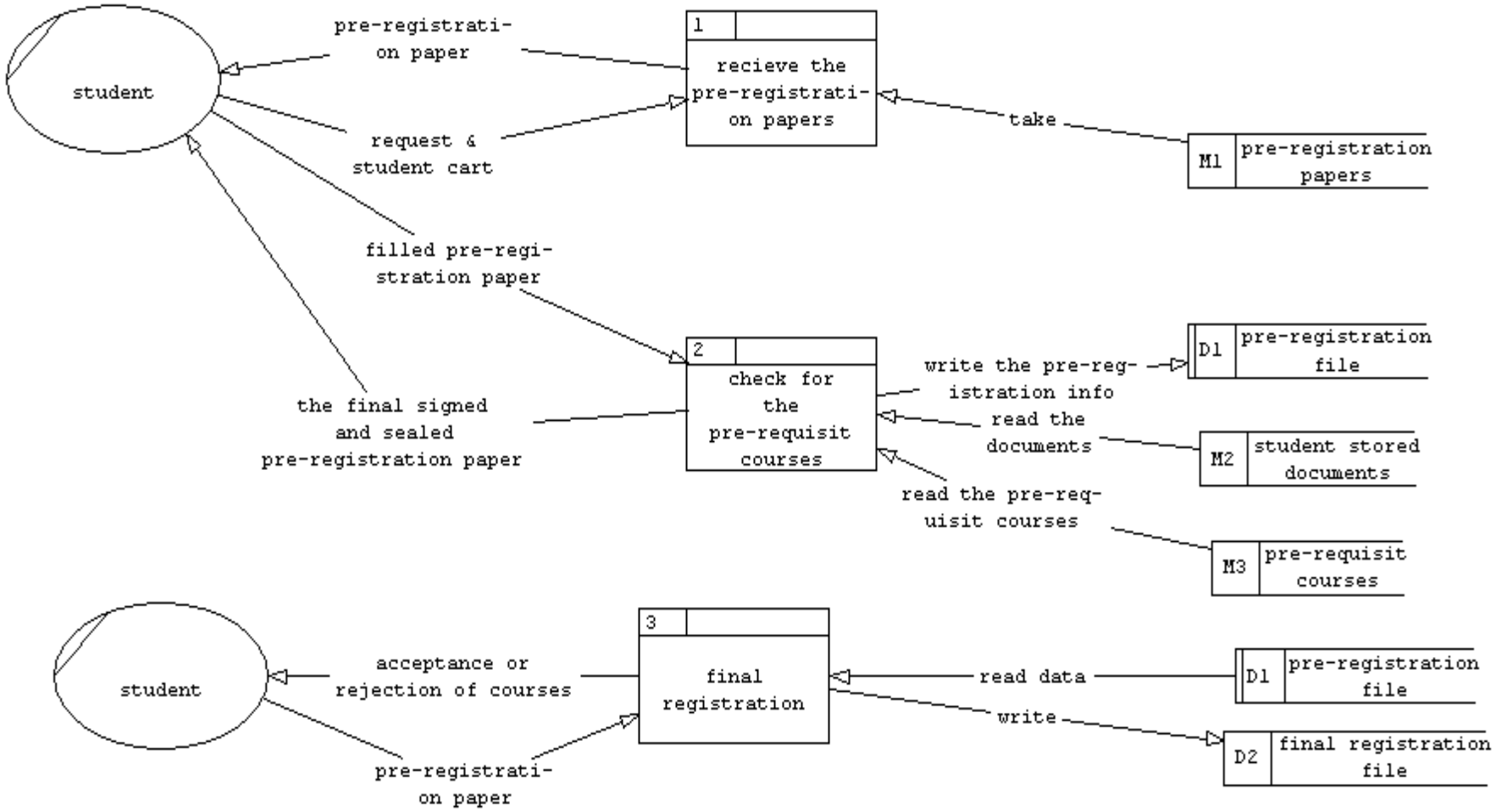
- تمام *data flow* ها داراي نام هستند.
- اگر *data flow* ها در مرتبه بالاتر هستند در مرتبه پايين تر هم هستند.
- در سطح بالا همه سيستم را در *bubble* قرار داديم و در سطح بعد به *f1* و *f2*... شكسته مي شود.
- ورودي و خروجي ها كاملاً حفظ شده است از پروسس هاي سطح اول *f4* شكسته شده است اما ورودي و خروجي هاي *x* و *y* و *z* هنوز وجود دارد.

مثالی از رسم **DFD** با استانداردهای **SSADM**برای **سیستم آموزش دانشگاه**

- DFD سطح صفر:



- DFD سطح یک:

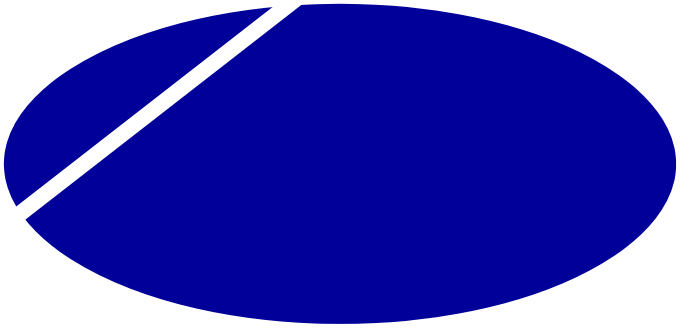

student
pre-registrati-
on paper
request &
student cart
1
recieve the
pre-registrati-
on papers
take
M1
pre-registration
papers
filled pre-regi-
stration paper
the final signed
and sealed
pre-registration paper
2
check for
the
pre-requisit
courses
write the pre-reg-
istration info
read the
documents
read the pre-req-
uisit courses
D1
pre-registration
file
M2
student stored
documents
M3
pre-requisit
courses
student
acceptance or
rejection of courses
pre-registrati-
on paper
3
final
registration
read data
write
D1
pre-registration
file
D2
final registration
file

نکات مهم در رسم نمودار *dfd* در *ssadm* :

- معمولاً *dfd* را در سطوح بالاتر بصورت كلي بيان مي كنيم و بتدريج در سطوح پايين تر شكسته مي شود بنا براين در ترسيم *d fd* بايد دقت كنيم كه پروسس هاي كلي را ابتدا بيان كرده وآنگاه مي توانيم در صورت نياز براي هر يك از پروسس هاي ديگر *d fd* يا *dfd* هايي در سطوح ديگر رسم كنيم. ديدگاه *dfd* ديدگاه سلسله مراتبي است زيرا در هر سطح جزئيات بيشتري را به تصوير مي كشد. لازم به ذكر است كه فلسفة رسم *dfd* دستيابي به *function* ها مي باشد يعني *dfd* را بايد آنقدر شكاند تا در آخرين سطح آن عملاً هرپروسس كانديد ايجاد يك يا چند *function* مشخص باشد.

نكات مهم در رسم نمودار *dfd* در *ssadm* (ادامه) :

- در رسم *dfd* چنانچه يك *external entity* را بيش از يكبار بياوريم (به جهت خوانايي بيشتر *dfd*) مطابق شكل يك خط مورب در گوشة سمت چپ *external entity* ها قرار داده مي شود.



نکات مهم در رسم نمودار *dfd* در *ssadm* :(ادامه)

- در رسم ا نباره هاي داده چنا نچه يك ا نباره بيش از يك بار استفاده شود شكل آن ا نباره بصورت زير خواهد بود:

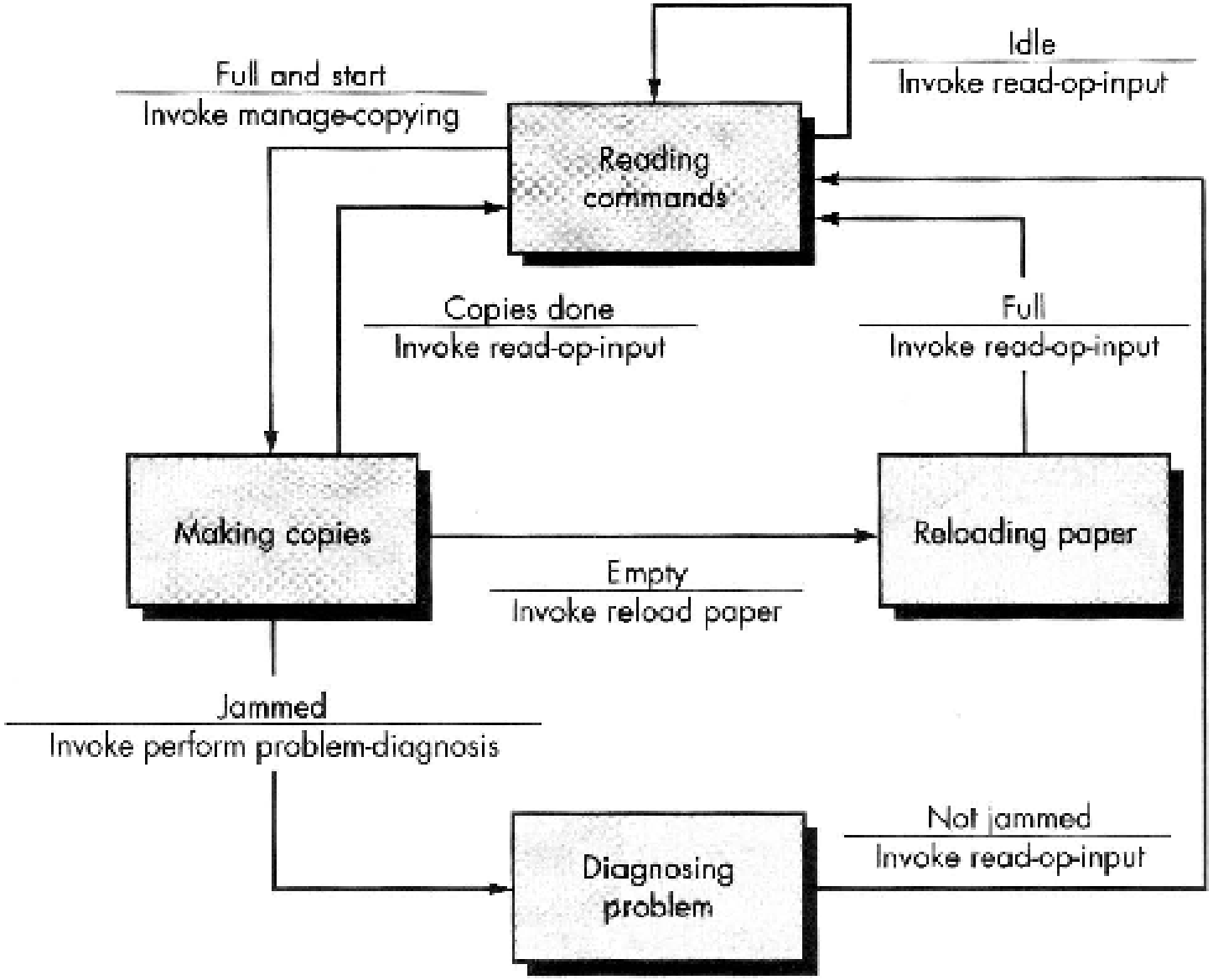
- در رسم *dfd* آخرين سطح پروسس (بالاترين سطح جزئيات) را با يك ستاره درگوشة آن مشخص مي كنيم

نكاتي راجع به نحوة رسم *f low* ها :

- امكان ارتباط مستقيم بين دو *data store* ميسر نيست.
- امكان ارتباط بين يك *data store* و يك *external entity* بطور مستقيم ميسر نيست.
- ارتباط بين دو پروسس هر چند كه امكان پذير است اما پيشنهاد نمي شود و در بيشتر مواقع اگر ارتباطي بين دو پروسس وجود داشته باشد، اين ارتباط غالباً ترتيب اجراي اعمال را مشخص مي كند.

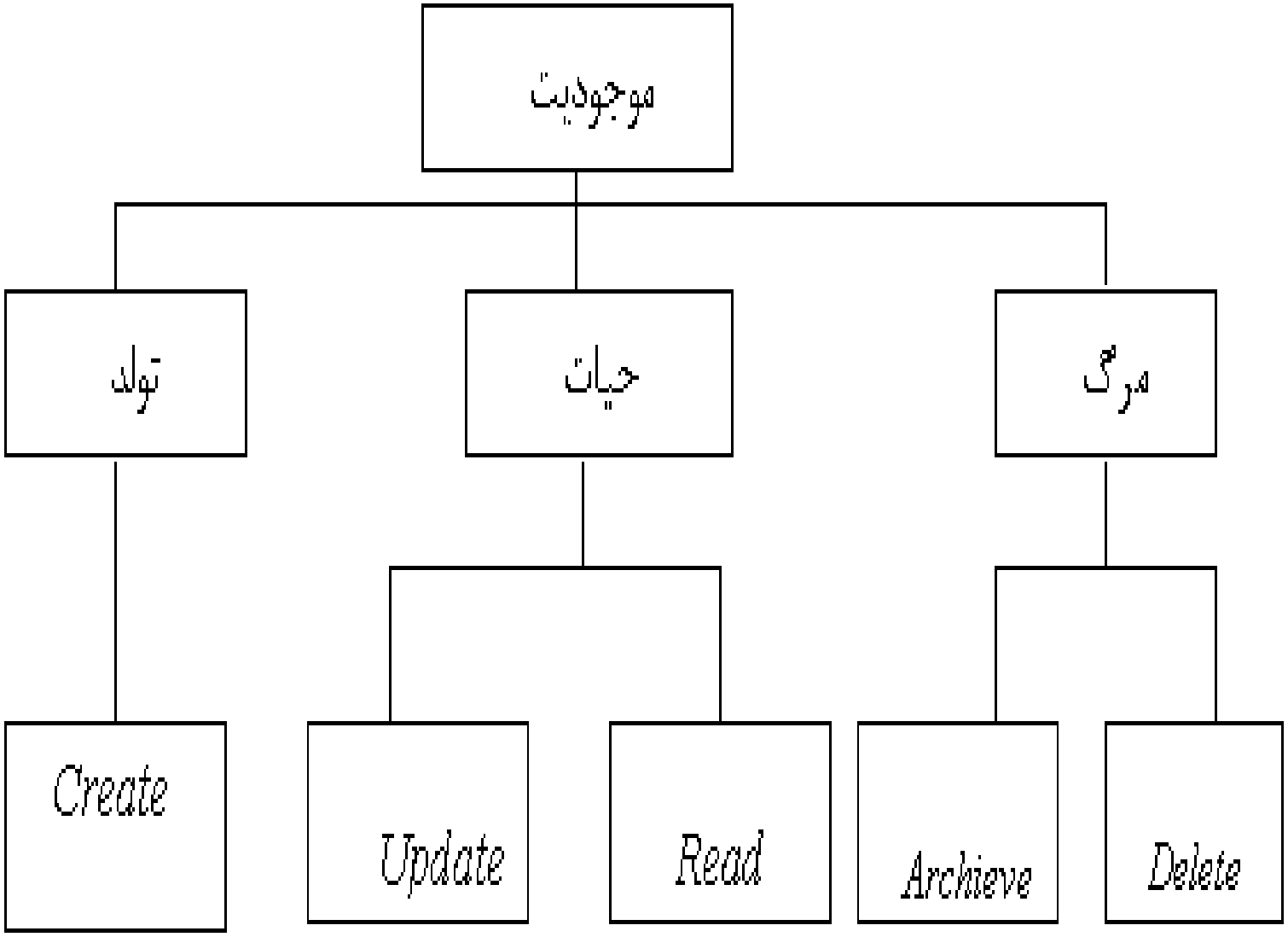
State transition diagram

- در مدل سازي سيستم ها همواره لازم است تا رخدادهاي سيستم و اينكه هر رخدادي چه تاثيري بر روي سيستم مي گذارد مشخص گردد. در كتاب پرسمن براي اين كار نمودار *State transition diagram* را پيشنهاد نموده است.


Idle
Invoke read-op-input
Full and start
Invoke manage-copying
Reading commands
Copies done
Invoke read-op-input
Full
Invoke read-op-input
Making copies
Reloading paper
Empty
Invoke reload paper
Jammed
Invoke perform problem-diagnosis
Diagnosing problem
Not jammed
Invoke read-op-input

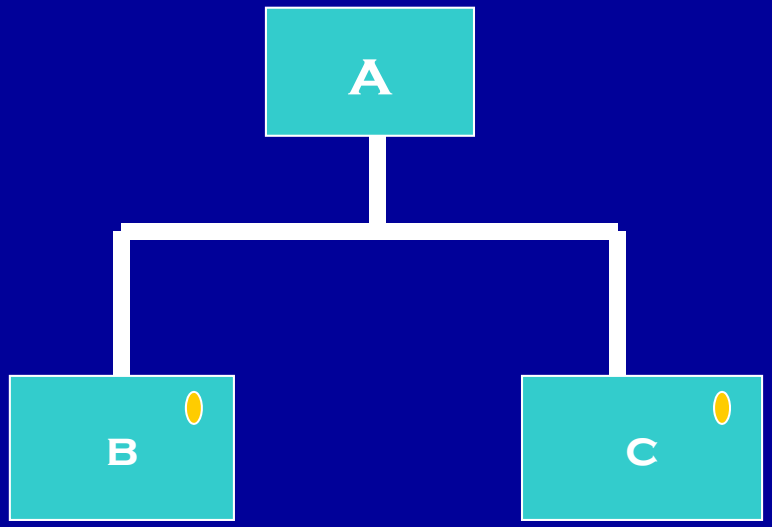
ELH(Entity Life History)

- در *ELH* بررسي مي كنيم كه چه رخدادهايي براي هر موجوديتي اتفاق مي افتد كه معمولاً اين رخدادها را در 3 بخش تولد موجوديت، دوره حيات موجوديت و دوران مرگ موجوديت مورد بررسي قرار مي دهيم:

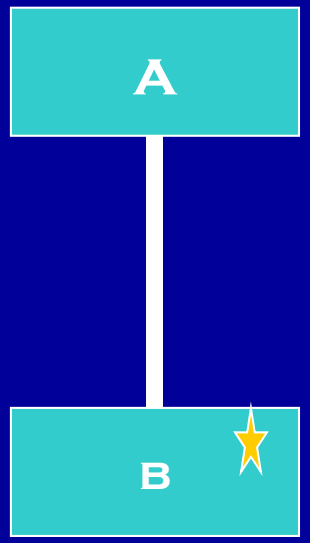

موجودیت
تولد
حیات
مرگ
Create
Update
Read
Archieve
Delete

Jackson notation :

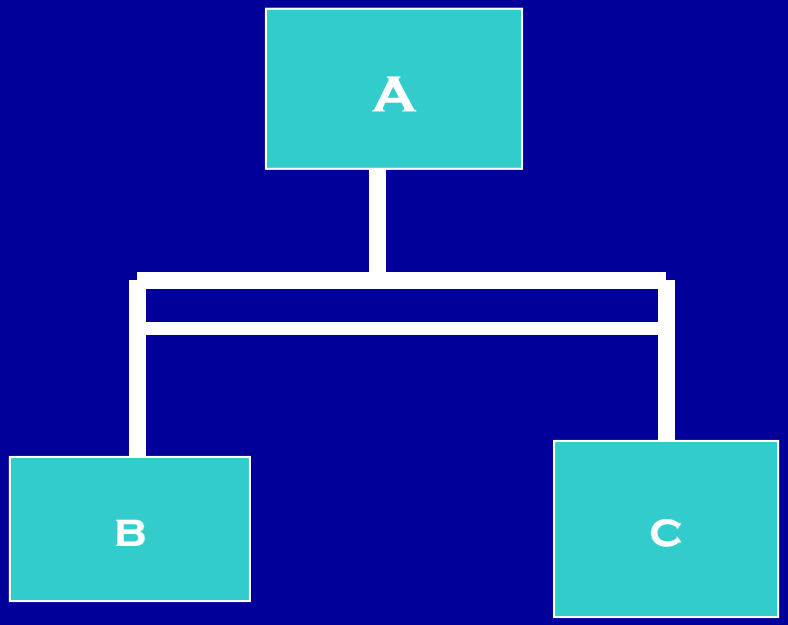
1- سمبل دایره بیانگر این است که *A* یا شامل *B* است، یا شامل *C* (*XOR*) .



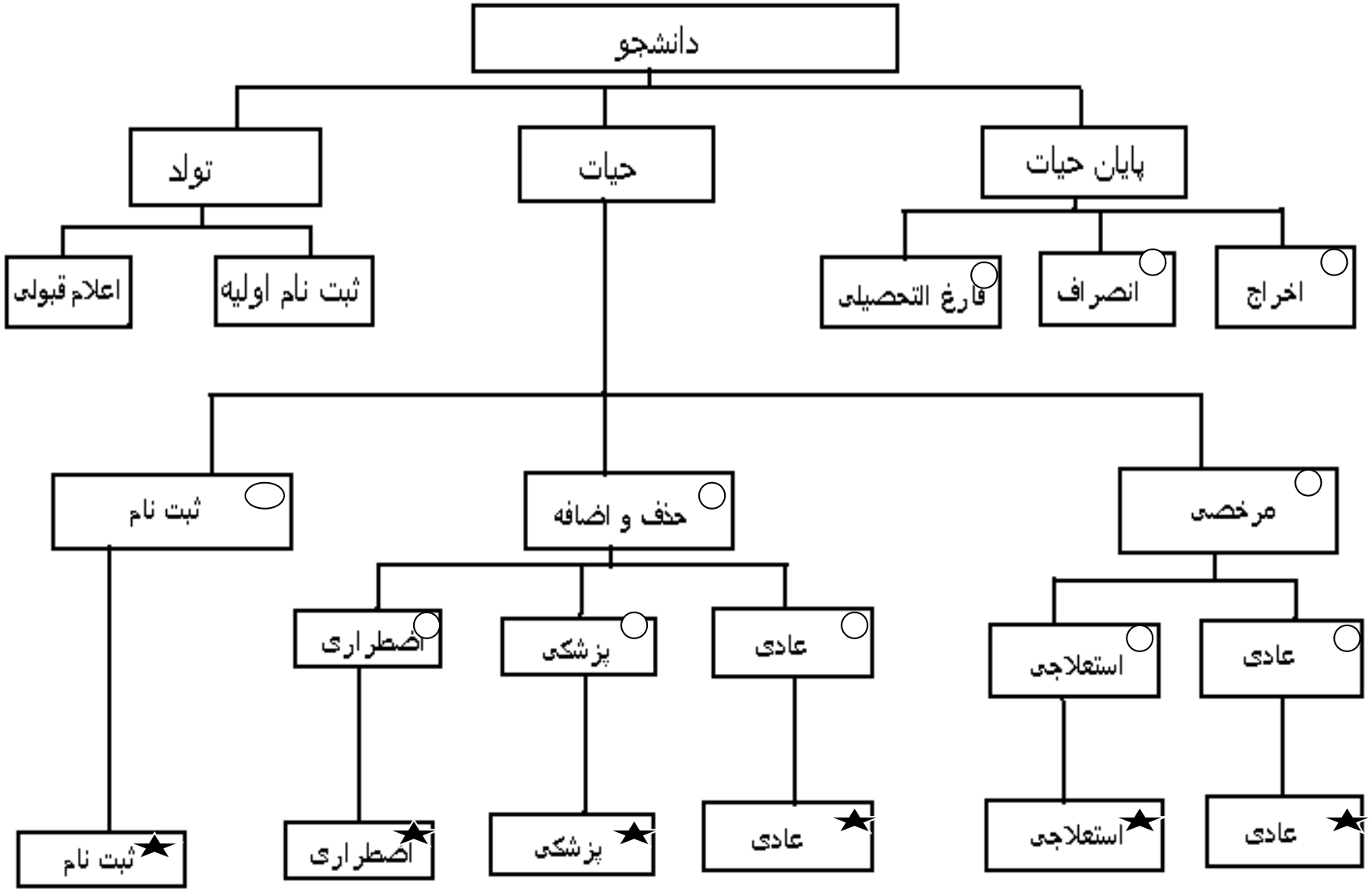
Iteration -2 به اين معناست كه *A* مي تواند شامل 0 تا يكي يا بيشتر از *B* باشد.



3- زماني كه *Sequance* (ترتيب) *event* ها غير قابل پيش گويي باشد، از ساختار موازي مطابق شكل استفاده مي شود. در اينجا *B*، *event* اصلي است و *C*، *event* غير اصلي است كه زمان رخداد آن مشخص نيست.



- عملیات *CRUD* : عمل *Create* در دوران تولد است. در دوران حیات یا *Read* داریم یا *Update* و در دوران مرگ *Delete* یا آرشیو داریم.


دانشجو
تولد
حیات
پایان حیات
اعلام قبولی
ثبت نام اولیه
فارغ التحصیلی
انصراف
اخراج
ثبت نام
حذف و اضافه
مرخصی
اضطراری
پزشکی
عادی
استعلاجی
عادی
ثبت نام
اضطراری
پزشکی
عادی
استعلاجی
عادی

- *Effect*: زماني كه يك *event* به وقوع مي پيوندد، يك يا چند موجوديت را به روز مي رساند كه در اين حالت به آن *Ef fect* مي گويند.

- معمولاً در رسم *ELH* سطوح آخرمان را تبديل به *Effect* مي كنيم.

مقدمه ای بر متدلوژی:

تعريف اوليه از متدلوژي

منظور از متدلوژي به كارگيري مجموعه اي از روش ها، گامها و توصيه ها، براساس پاره اي از نمادها *(notation)* ها جهت انجام يك يا چند فاز از مدل چرخه حيات نرم افزار مي باشد.

تعريف *Methode*(روش):

فرآيندي منظم است كه با استفاده از مجموعه نماد گذاري هاي خوش تعريف، مجموعه اي از مدلها را ايجاد مي كند كه هر كدام بخشي از سيستم نرم افزاري در حال توليد يا توسعه را توصيف مي كند.

تعريف *Methodology*

متدلوژي مجموعه اي از روشها (*Methode* ها) مي باشد که در تمام چرخه حيات سيستم نرم افزاري(يا حداقل بعضي از فازهاي آن) اعمال شده و بر يك نوع نگرش كلي درباره جهان نرم افزار متكي مي باشد.

اهميت متدلوژي

به نظر بيشتر متخصصين به كارگيري يك متدلوژي مدون در توليد نرم افزار قادر خواهد بود تا به اندازه قابل توجهي مشكلات بيان شده در توليد نرم افزار را برطرف نمايد.

ويژگيهاي يك متدلوژي مطلوب

- ارائه تعاريف از مفاهيم اوليه به كار رفته در متدلوژي.
- داراي يك مدل فرايند توليد *Software Process Model* مي باشد.
- داراي يك شيوه علامت گذاري استاندارد باشد.
- وجود ابزار اتوماتيك براي كمك به توليد و اجراي مدلهاي مبتني بر متدلوژي *Case Tools*

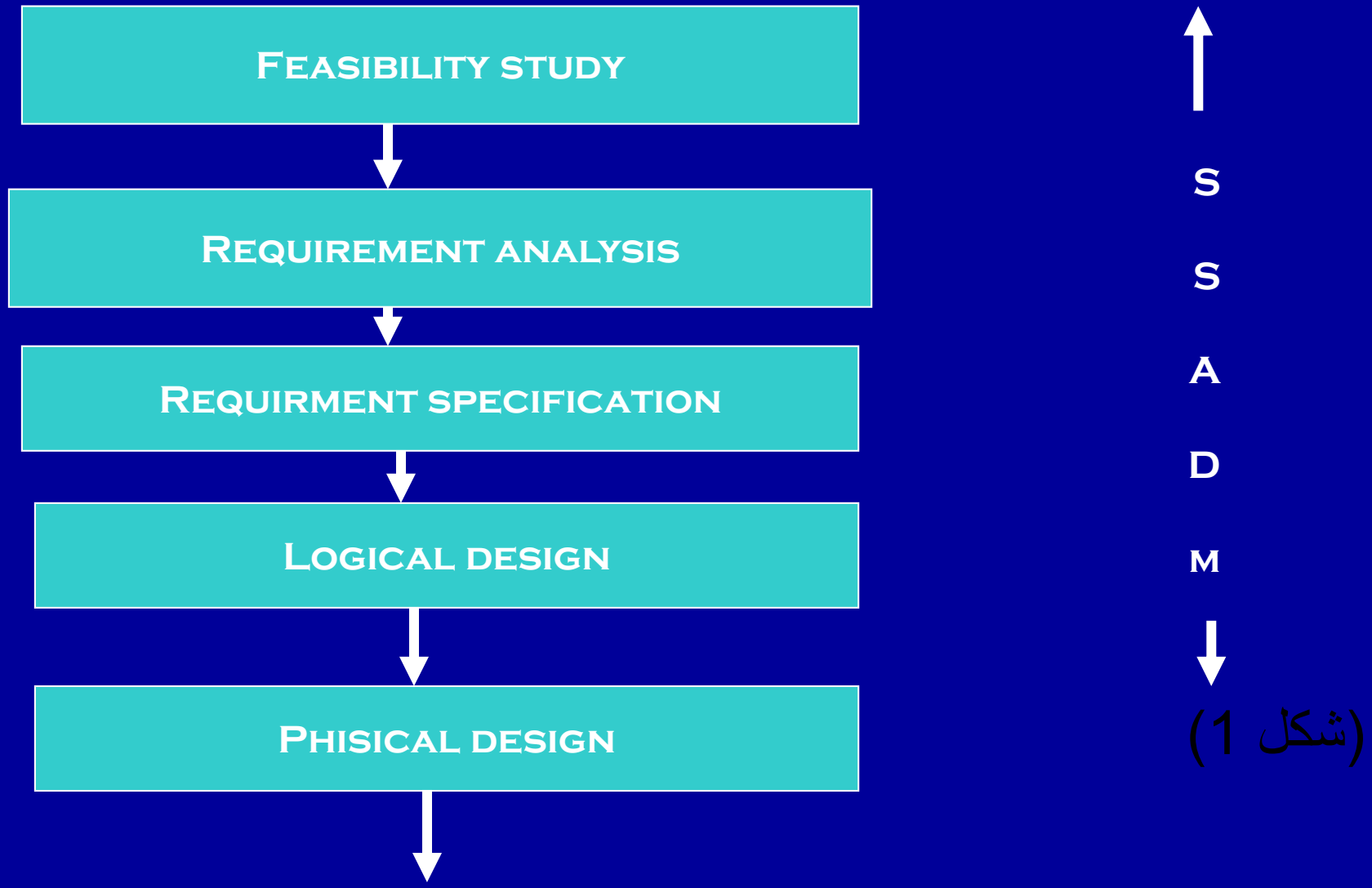
ويژگيهاي يك متدلوژي مطلوب(ادامه)

- متدلوژي مطلوب بايد معيارهايي براي ارزيابي نتايج حاصل از بكارگيري متدلوژي ارائه دهد.
- معرفي تكنيك هايي براي پياده سازي متدلوژي كه توانايي كنترل پيچيدگي هاي سيستم كنوني را دارا باشد.
- داشتن يك مدل زير بنايي (مدل معماري).

دو دیدگاه اساسي در متدلوژي داريم، نمونه اي ازمتدولوژی های موجود در این دو دیدگاه:

Object orienter	structural
booch	ssadm
omt	Ie(information engineering)
Usdp(rup)	yourdon
	jackson
	cdm

مدل فرایند نرم افزار در SSADM :



(شکل 1)

متدولوژی ماهیتا Tailor Madeاست

زماني كه هدف ايجاد يك پروژه نرم افزاري است و به دنبال يك متدلوژي هستيم، لازم است دقت شود كه انتخاب يك متدلوژي لزوما به اين معنا نيست كه بايد تمام گام هاي آن را به صورت مو به مو و گام به گام اجرا نمود و كليه فرآورده های آن را بصورت تک به تک توليد نمود، بلكه اصطلاحاً گفته مي شود كه متدلوژي *tailor made* است، يعني براساس شرايط پروژه خاص متخصصين فني اين قضيه لازم است تصميم بگيرند كه كدام قسمت يك متدلوژي و آن هم به چه شكلي و تا چه حدي لازم است در پروژه مذكور به كار گرفته شود.

در اين راه ممكن است گاهي در انجام يك پروژه تركيبي از متدلوژي ها نيز به كار گرفته شود. به عنوان مثال ممكن است از فرم هاي استاندارد متدلوژي *CDM* براي بخش تست در متدلوژي *RUP* استفاده نمود و يا مثلاً توليد كنندگاني را كه با بكارگيري متدلوژي *RUP* قصد دارند تا سيستمي را توليد كنند، ملزم كنيم كه نمودار *P M* را نيز رسم كنند.

ويژگي هاي *CASE TOOLS* ها:

. به طور كلي تمامي ابزارهاي *C ase tools* داراي امكانات زير مي باشد:

1) *Diagramming tools*
2) *Automatic Generation of first cut*
3) *Diagram Validation*
4) *Report Generation*
5) *Code Generator*

EAP (Enquary Access Path)

*EAP*مسير دستيابي به داده هاي استعلام ميباشد تكنيك *EAP* براي محرز ساختن اين قضيه در متدلوژي *SSADM* گنجانده شده است، بدين ترتيب كه در اينجا به ازاي هر *enquary* يك *EAP* رسم مي كنيم .

BPM(Bussiness Process Modeling)

در متدلوژي هاي جديد مرحله اي به نام *BPM* وجود دارد، اين مرحله عملاً مقدمه معماري نرم افزار مي باشد، زيرا در معماري نرم افزار شناخت فرايندها از اهميت ويژه اي برخوردار است.

فرايند (*Process*)

- فرايند مرحله اي سلسله مداومي از تغييرات است كه منجر به نتيجه خاصي مي شود، فرايند از وضعيتي به وضعيت ديگر منتقل مي شود و بايد داراي نتيجه باشد.
- وقتي يك زنجيره مداوم از اعمال و تغييرات را دنبال مي كنيم كه در يك *business* اتفاق مي افتد، عملاً به مدل سازي آن *Bussiness* مي پردازيم .

ويژگي هاي فرايند

- داراي يك سري *state* مي باشد و با يكسري فعاليت ها از يك *state* به *state* ديگر منتقل مي شود.
- يك سري *event* (رخدادها) ممكن است منجر به اين شده كه از يك *state* به *state* ديگر برويم و گاهي اوقات براي فعال شدن يك *state* نياز به بروز يك *event* مي باشد.
- يك سري *Egent* (نماينده) موجود مي باشد كه غالباً مامور انجام هر يك از مراحل فرايند مي باشد .

- تفاوت بين *Process* و *Procedure*: در انجام *Process* ممكن است سليقه به خرج داده شود يعني در عمل اجراي يك *Process* ممكن است به شكل هاي گوناگوني انجام شود، اما در مورد *Procedure* چنين نيست و اجراي يك *Procedure* غالباً به صورت دقيقاً مشخص شده صورت مي گيرد.

Model چيست؟

منظور از مدل نمايش ساده شده از يك پديده مي باشد، جهت انجام مدل سازي غالباً از *abstraction* استفاده مي كنيم. (منظور از *abstraction* آن است كه براي انجام كارها صرفاً به كليات پرداخته و از پرداختن به جزئيات خودداري مي شود.)

چرا فرايند را مدل مي كنيم؟

مدل سازي فرايندها قابليت فهم مسئله را افزايش مي دهد و عملاً به ما كمك مي كند تا درك دقيق تري از پيچيدگي سيستم ها را بدست آوريم.

بررسي كلي برخي از اطلاعات موجود در كاتالوگ سيستم:

1- *Elemantry Process Description* (شرح پروسس هاي اوليه) آنهايي كه در سطح آخر هستند و اينكه در مواقع تشكيل *function* ما را مي دهند كه توضيحات كاملي را در اينجا مي دهيم كه ديگر احتياجي نباشد در سطوح بالاتر توضيحات بدهيم.

2 - External Entity Description :

External Entity همیشه چیزي مثل دانشجو نیست که خیلي گویا باشد، مثلاً مرکز خدمات آموزشي که مي توانیم شرح دهیم که منظورمان: دانشگاهها، آموزش و پرورش، مؤسسات آموزشي و ... مي باشد.

3- *Flow Description* اينكه *flow* هاي ما از كجا به كجا چه اطلاعاتي را منتقل مي كنند، اينكه *material* است يا اينكه فقط *Sequance* را مشخص مي كند.

پایان