بسم الله الرحمن الرحیم

دانشگاه آزاد
دانشکده کامپوتر

مهندسی نرم افزار ۲

گردآوری : مهندس بهروز نیرومندفام

شیوه ارتباط

- رفع اشکال در ساعات کلاس.
- ارتباط دائمی از طریق Computercollege_fam@Yahoo.com

نحوه ارزیابی

آزمون کتبی

میان‌ترم (۲۰)

پایان‌ترم (۴۵)

تمرینات (۸)

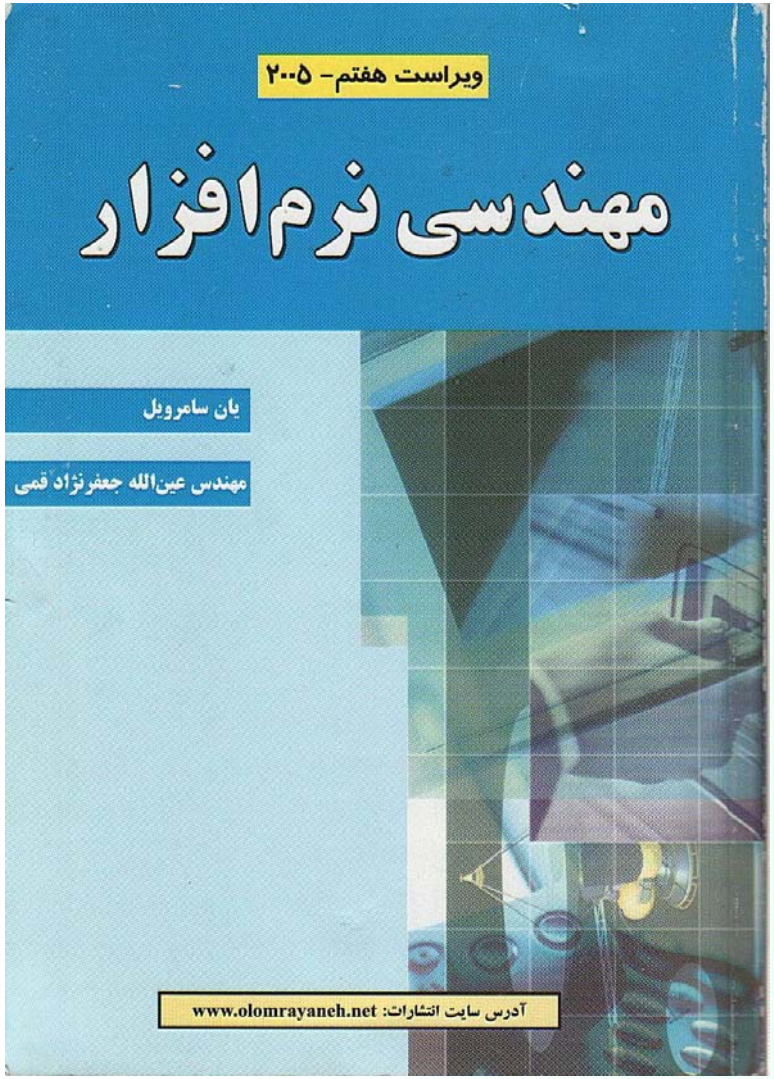
کوئیز و فعالیت کلاسی (۷)

پروژه (۲۰)

حضور در کلاس

حضور در کلاس الزامی است و براساس قوانین آموزشی، غیبت بیش از ۱۶/۳ منجر به اعلام غیبت برای دانشجو خواهد شد.

منابع



- مهندسی نرم افزار

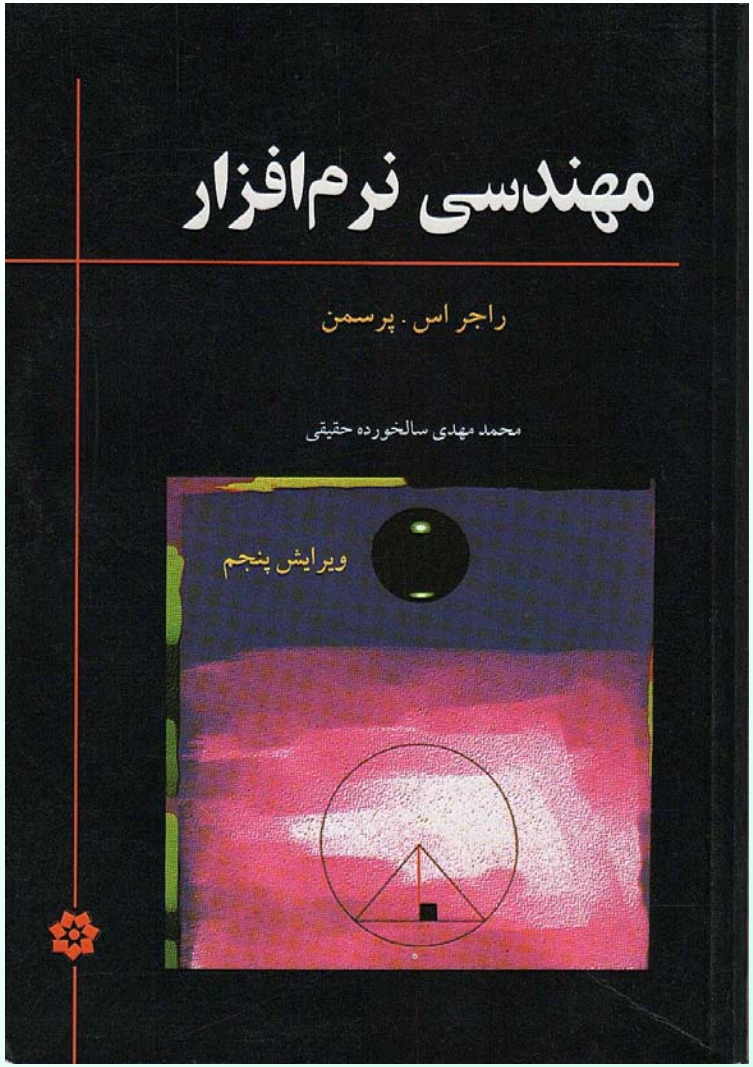
  اثر: یان سامرویل

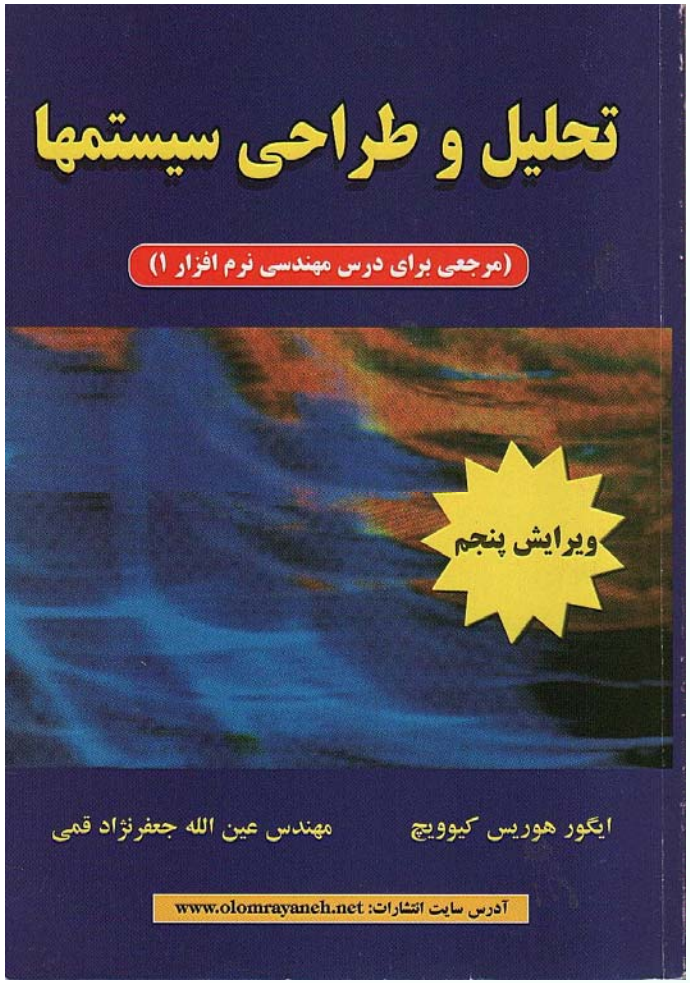
  مترجم: عین الله جعفر نژاد قمی

مهندسی نرم افزار

اثر: راجر اس. پرسمن

مترجم: محمد مهدی سالخورده حقیقی





تحلیل و طراحی سیستم ها

اثر: ایگور هوریس کیوویچ

مترجم: عین الله جعفر نژاد قمی

- *Software Engineering, A Practitioner's Approach*, Roger S. Pressman, Fifth Edition

Email: ComputerCollege_Fam@Yahoo.Com

- بخش اول: محصول و فرآیند

– در این بخش مفاهیم اولیه مهندسی نرم‌افزار و متدولوژی‌های توسعه نرم‌افزار معرفی می‌شوند. مفاهیمی چون نرم‌افزار، نقش نرم‌افزار به‌عنوان محصول، بحران نرم‌افزار، مهندسی نرم‌افزار، متد و متدولوژی، متدولوژی‌های توسعه نرم‌افزار و ... از جمله مفاهیمی هستند که در این بخش مورد بررسی قرار می‌گیرند.

– این بخش شامل دو فصل است

- بخش دوم: مدیریت پروژه‌های نرم‌افزاری
  - در این بخش مفاهیم مدیریت پروژه‌های نرم‌افزاری، فرآیند مدیریت پروژه و متریک‌های پروژه، نحوهٔ ارزیابی پروژه‌های نرم‌افزاری، برنامه‌ریزی و تخصیص منابع به پروژه، مدل‌های تخمین، مدیریت و تحلیل خطر، تضمین کیفیت نرم‌افزار و مدیریت پیکربندی نرم‌افزار و ... مورد بحث قرار می‌گیرند.
  - این بخش شامل هفت فصل است

- بخش سوم: روش‌های متداول مهندسی نرم‌افزار
  - در این بخش مفاهیم مهندسی سیستم، مهندسی نیازمندی‌ها، مفاهیم و اصول تحلیل نیازمندی‌ها، روش‌های مدلسازی، مدل‌سازی داده، مدل‌سازی رفتار و ساختار، مفاهیم و اصول طراحی، مستندسازی طراحی، طراحی معماری، نگاشت نیازمندی‌ها به معماری، طراحی رابط کاربر، طراحی مولفه‌ها، روش‌های آزمایش نرم‌افزار، استراتژی‌های آزمایش، متریک‌های فنی ارزیابی نرم‌افزار و ... مورد بحث قرار می‌گیرند.
  - این بخش شامل ده فصل است

- بخش چهارم: مهندسی نرم‌افزار شی‌گرا

– در این بخش مفاهیم و اصول شی‌گرایی، مدیریت پروژه‌های نرم‌افزاری شی‌گرا، مفاهیم و اصول تحلیل شی‌گرا، شناسایی موارد کاربری، کلاس‌ها و رفتار سیستم، مفاهیم و اصول طراحی شی‌گرا، فرآیند طراحی شی‌گرا، الگوهای طراحی، مفاهیم و اصول آزمایش شی‌گرا، استراتژی‌های آزمایش شی‌گرا، متریک‌های آزمایش شی‌گرا و ... مورد بحث قرار می‌گیرند.

– این بخش شامل پنج فصل است

فصل ۱

محصول

مفهوم نرم‌افزار

خصوصیات نرم‌افزار

تفاوت نرم‌افزار و سخت‌افزار

مفهوم مهندسی نرم‌افزار

افسانه‌های مهندسی نرم‌افزار

چالش‌های توسعه نرم‌افزار

مفهوم نرم‌افزار

- مجموعه‌ای از
  - دستورات برنامه‌نویسی که وقتی اجرا می‌شوند، کارکرد مورد نظر را با کارایی مناسب ارائه می‌دهند.
  - ساختارهای داده‌ای که برای اجرای دستورات و نگهداری اطلاعات مورد نیازند.
  - مستنداتی که عملیات و نحوة استفاده از برنامه را نشان می‌دهند.

خصوصيات نرم‌افزار

- نرم‌افزار مهندسي و توسعه داده مي‌شود
  - نرم‌افزار همانند سخت‌افزار توليد نمي‌شود
  - كيفيت توسط طراحي خوب حاصل مي‌شود
    - در توليد سخت‌افزار مشكلات كيفي در ابتدا مشخص مي‌شوند
    - در توسعه نرم‌افزار برخي مشكلات در ابتدا وجود ندارند!
  - وابستگي به افراد
    - ارتباط بين افراد و كار انجام شده توسط آنها متفاوت است
  - نیازمند ساخت یک «محصول» هستند
    - روش ساخت متفاوت است

هزینه‌های نرم‌افزار بر مهندسی تمرکز دارند، بدین معنی که پروژه‌های مهندسی نرم‌افزار نمی‌توانند همانند پروژه‌های سخت‌افزاری تولید شوند

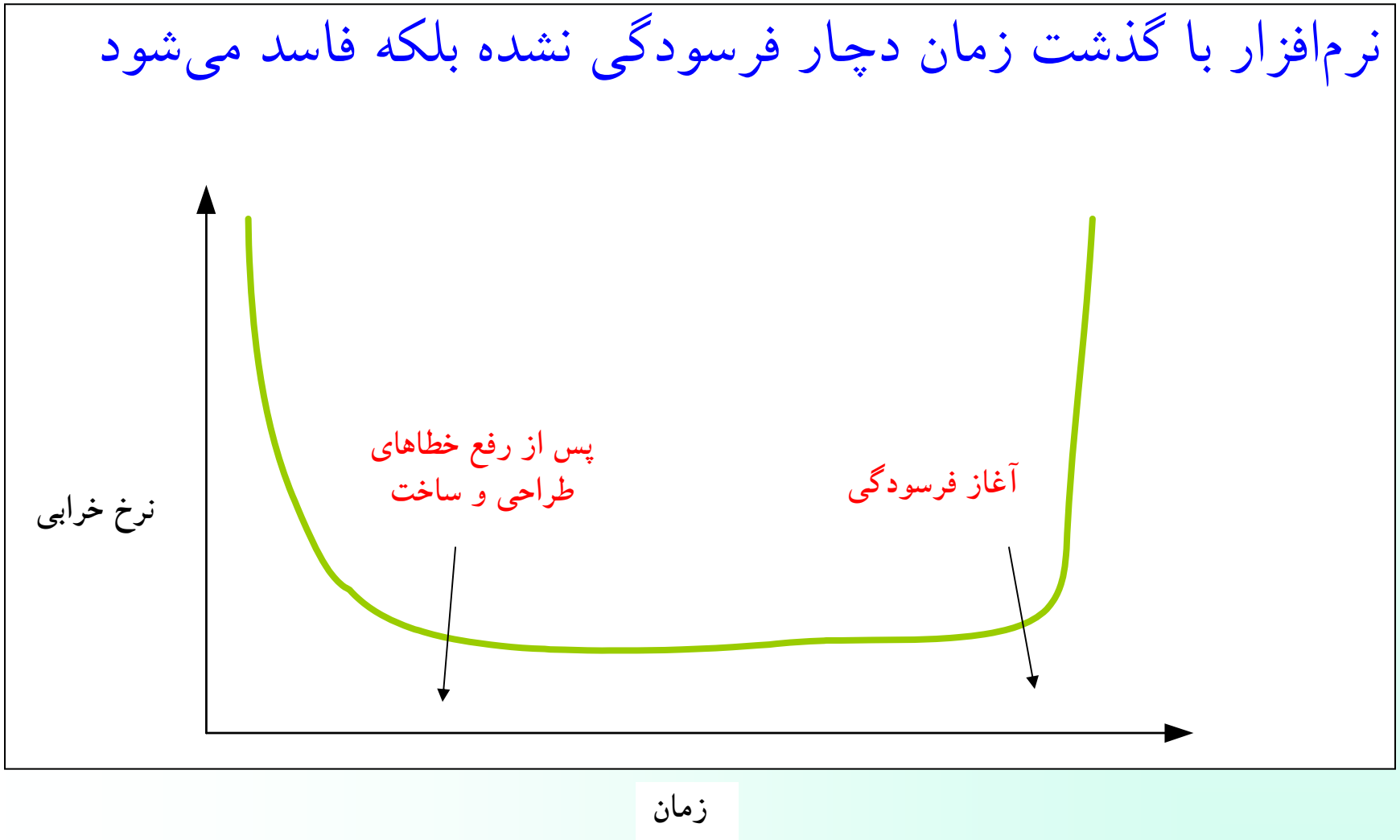
اگر چه صنعت به سمت مونتاژ مولفه‌ها پیش می‌رود، اما اغلب نرم‌افزارها به‌صورت سفارشی توسعه داده می‌شوند

استفاده از مولفه‌های از پیش ساخته شده همانند IC در تولید سخت‌افزار بسیار رواج دارد

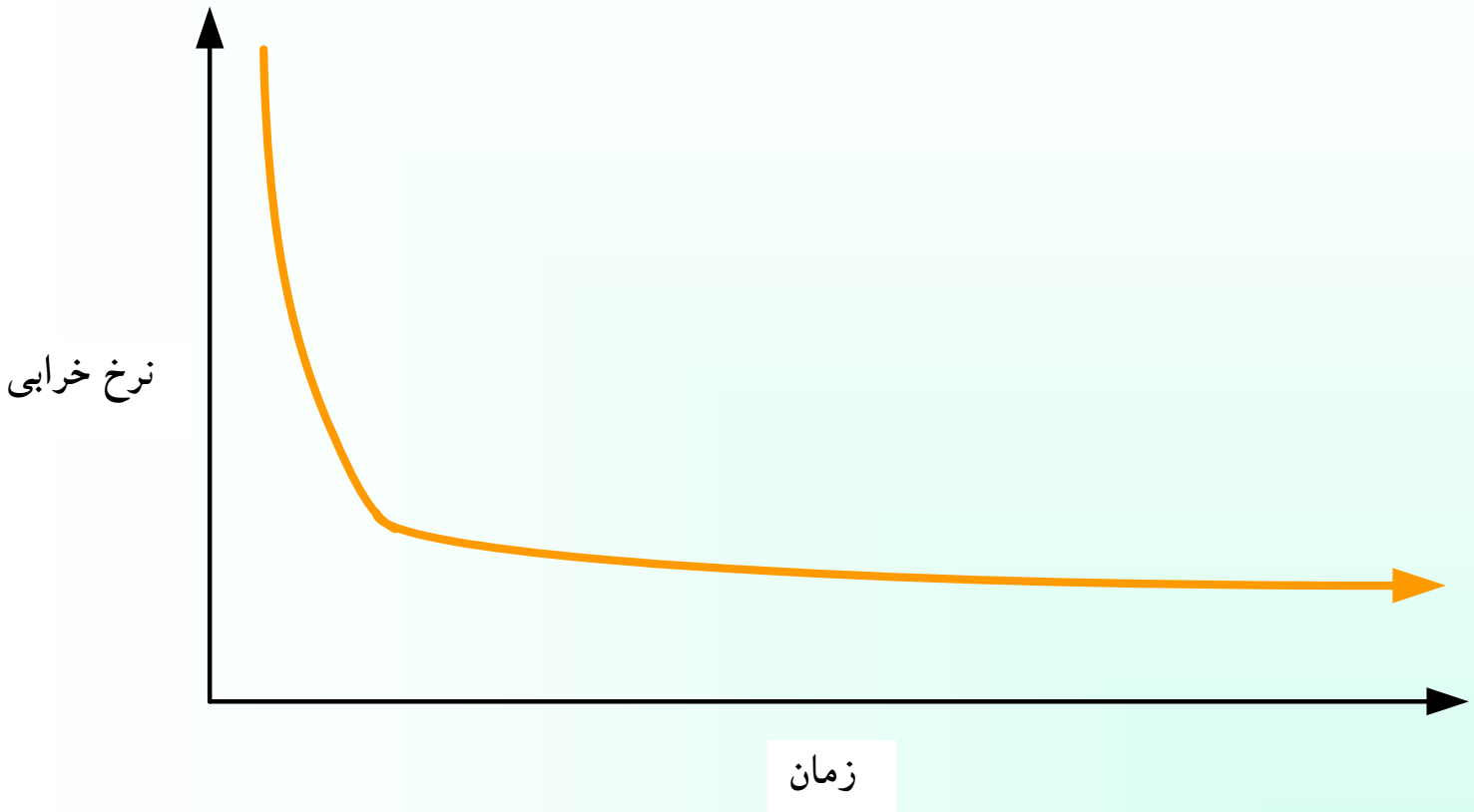
نرم‌افزار پیچیده است

هر نرم‌افزار حالت‌های بی‌شماری دارد که می‌تواند ناشی از رویدادهای درونی یا بیرونی باشد

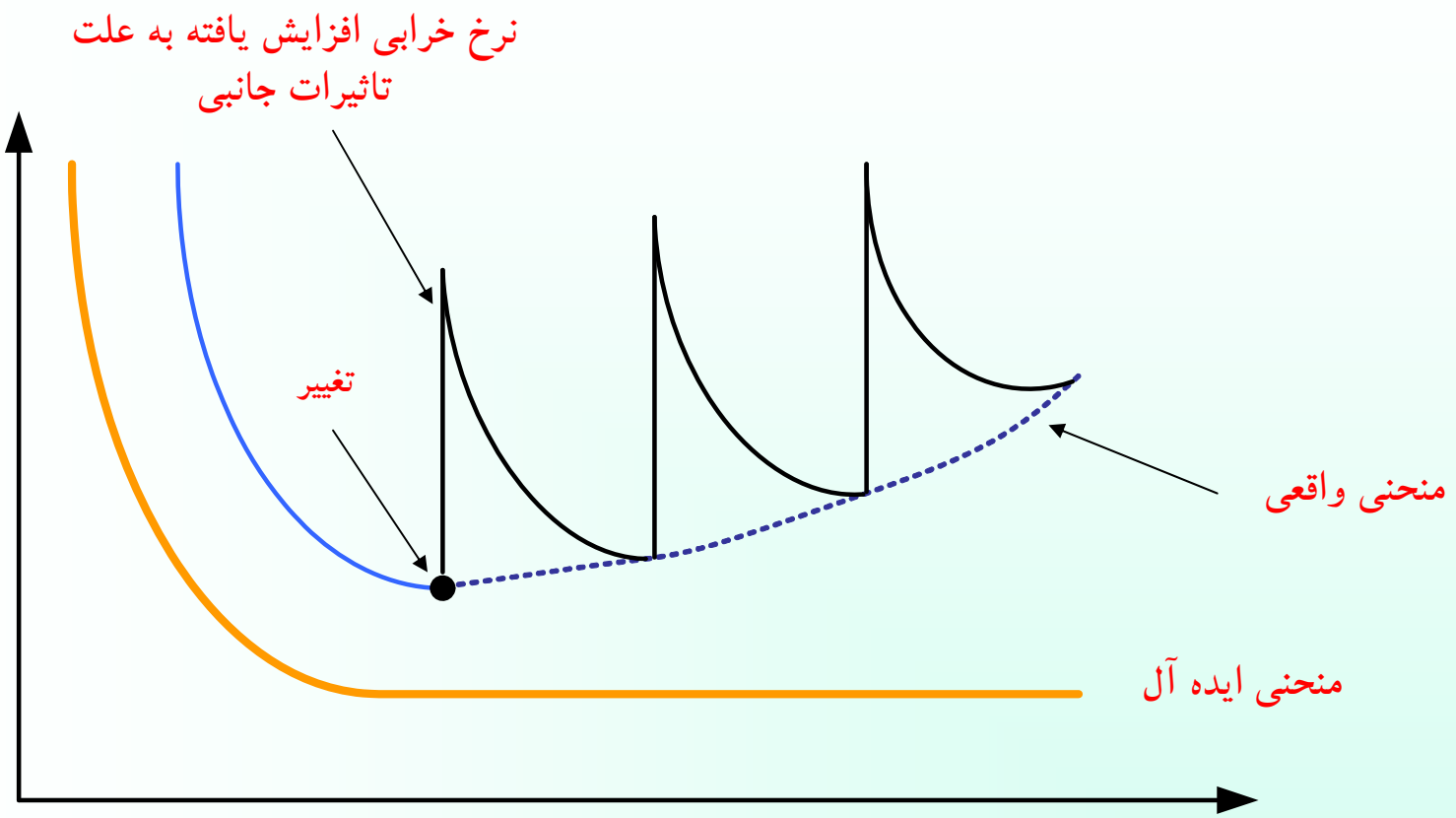
نرم‌افزار با گذشت زمان دچار فرسودگی نشده بلکه فاسد می‌شود



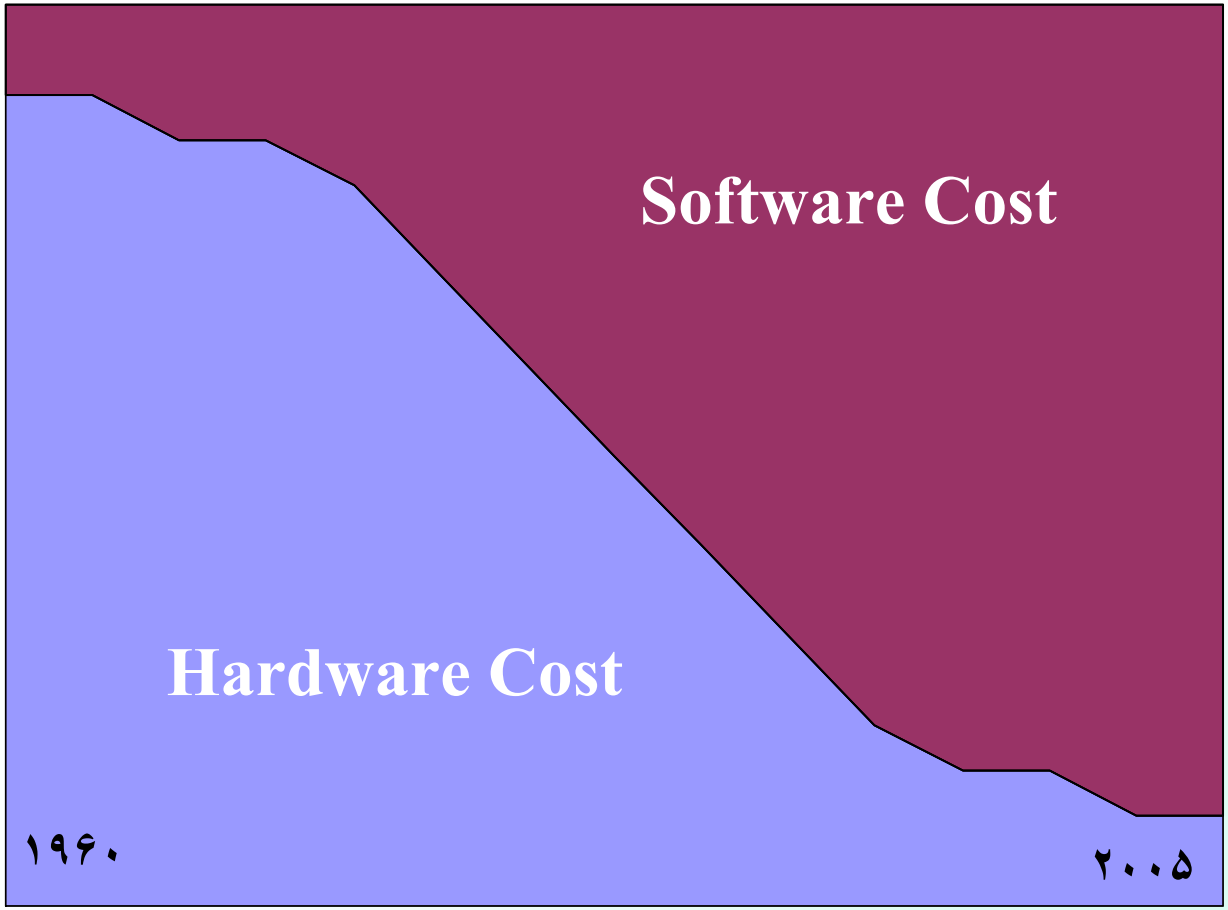
منحنی نرخ خرابی سخت‌افزار نسبت به زمان



منحنی نرخ خرابی ایده‌آل نرم‌افزار نسبت به زمان



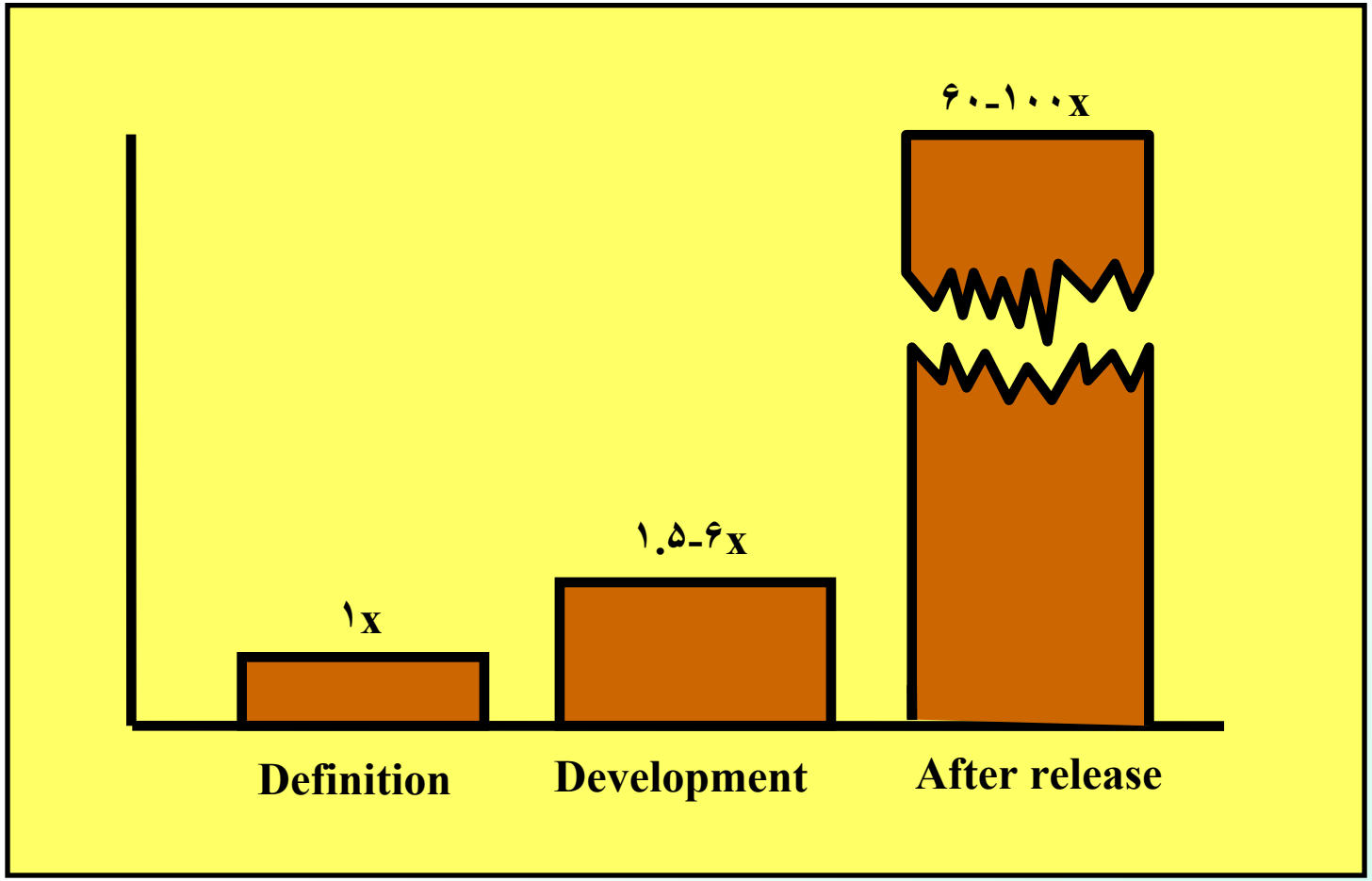
منحنی نرخ خرابی واقعی نرم‌افزار نسبت به زمان



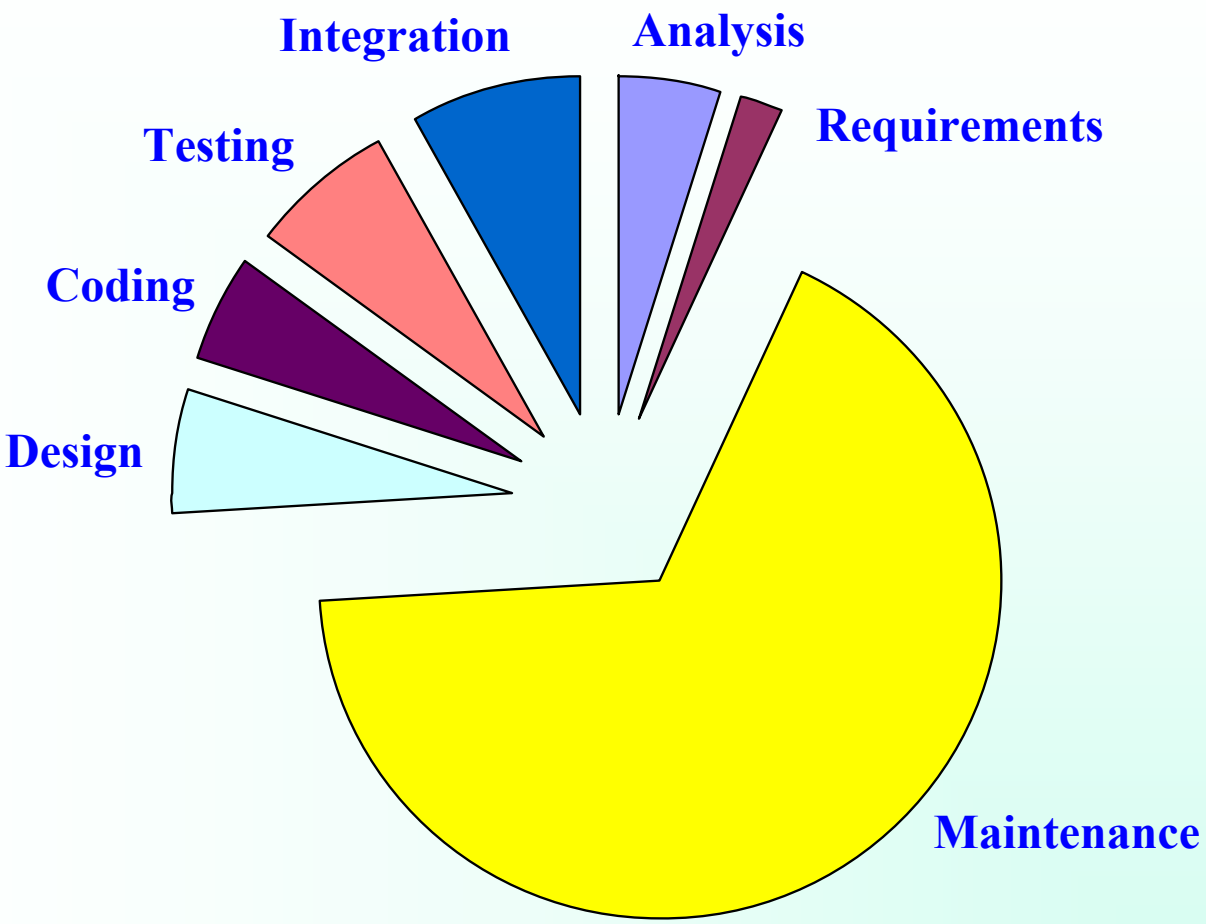
Time

هزینه نرم‌افزار نسبت به سخت‌افزار

هزینه تغییر در زمان‌های متفاوت



Email: ComputerCollege_Fam@Yahoo.Com



هزینه فازهای مختلف توسعه نرم‌افزار

تعریف مهندسی نرم‌افزار

- اصطلاح مهندسی نرم‌افزار برای اولین در دهه ۱۹۶۰ و در کنفرانس مربوط به بحران نرم‌افزار معرفی شد
- تعریف اولیه مهندسی نرم‌افزار

– [مهندسی نرم‌افزار] برقراری و استفاده از مفاهیم مهندسی به‌منظور دستیابی به نرم‌افزارهایی که از نظر اقتصادی به‌صرفه بوده، قابل اعتماد باشند و بتوانند در ماشین‌های واقعی به‌صورتی کارآمد عمل نمایند

تعریف مهندسی نرم‌افزار

- «مهندسی نرم‌افزار نظمی است که هدف آن تولید نرم‌افزاری با کیفیت، در زمان و با بودجه تعیین شده، که نیازهای کاربران را برآورده می‌سازد، است» (S.Schach, ۱۹۹۰)

- «به‌کارگیری روشی سیستماتیک، منظم و قابل اندازه‌گیری برای توسعه، عملیاتی نمودن و نگهداری نرم‌افزار است» (IEEE, ۱۹۹۳)

علل استفاده از مهندسی نرم‌افزار

توسعه نرم‌افزار مناسب با نیازمندی‌ها

کاهش هزینه توسعه

کاهش زمان توسعه

تکرار موفقیت‌های قبلی

درس گرفتن از شکست‌های قبلی

چالش‌های توسعه نرم‌افزار

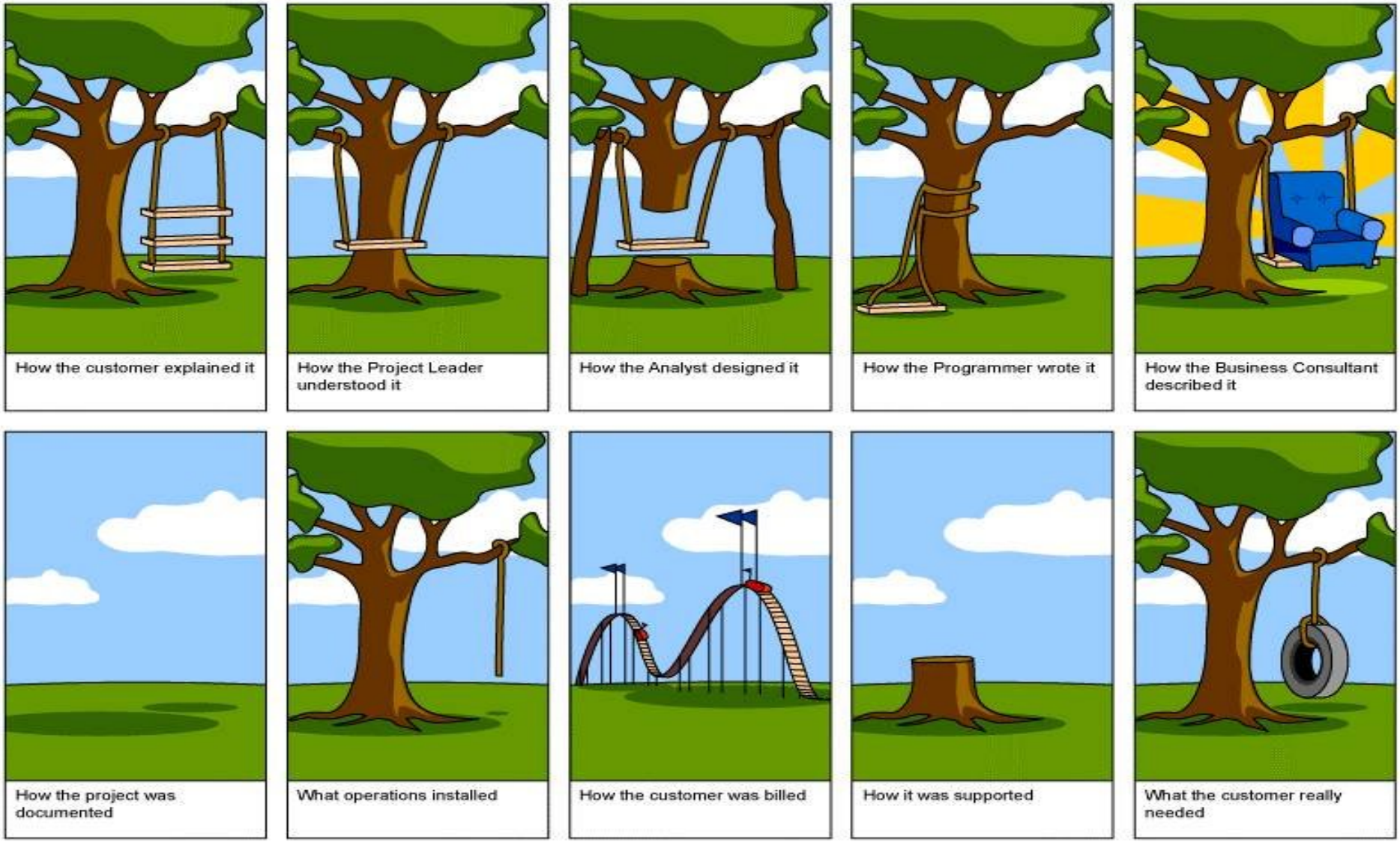
چگونه مطمئن شویم خواسته‌های کاربران را درست فهمیده‌ایم؟

چگونه هزینه و زمان توسعه نرم‌افزار را کنترل نمائیم؟

چگونه از فناوری‌ها و مفاهیم جدید در نرم‌افزار استفاده کنیم؟

چگونه مطمئن شویم که کیفیت نرم‌افزار مناسب است؟

چگونه نرم‌افزار قبلی را ارتقاء دهیم؟


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

تعریف متدولوژی

یک متدولوژی مجموعه ای از روش ها و توصیه ها (Guidelines) می باشد که به همراه راهبرد مشخص و طی مراحل مختلف از توسعه سیستم به کار گرفته می شود.

یک چارچوب است که ترتیب گام به گام راهکارهایی را که به توسعه محصول نهایی کمک می‌کنند، مشخص می‌نماید.

یک متدلوژی دارای ابزار تعریف شده و مدل مفهومی می باشد و از یک گرامر مشخص استفاده می کند.

هر متدولوژی برای انجام کارها از تعدادی ابزار و تکنیک استفاده می‌نماید.

برای مثال مدل شی گرا و یا مدل ساخت یافته در توسعه نرم افزار دو متدولوژی توسعه نرم افزار هستند.

تعریف مدل فرآیند

فرآیند مهندسی نرم افزار مجموعه ای از قدم های قابل پیش بینی برای توسعه نرم افزار را مشخص می کند.

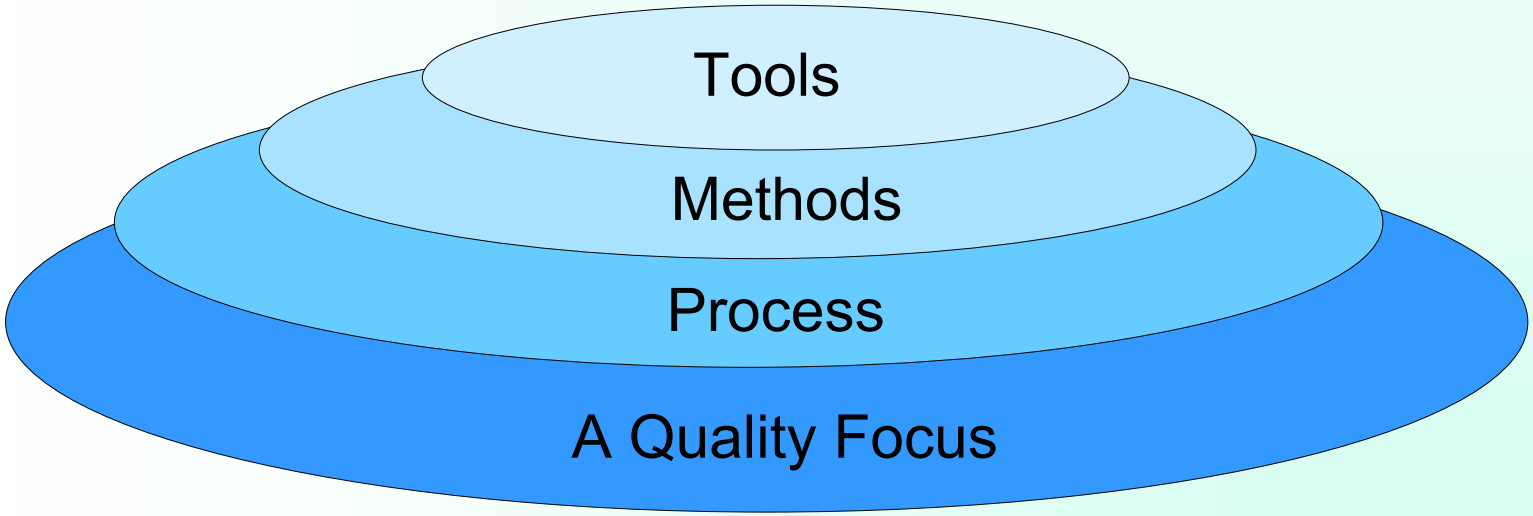
مدل فرآیند نرم افزار قدم ها و استراتژی توسعه نرم افزار فرآیند و روش می باشد.

از مدل‌های معروف می‌توان به مدل‌های
آبشاری،
افزایشی،
و چرخشی
اشاره کرد.

تفاوت های متدولوژی و مدل فرآیند

متدولوژی، روش طی کردن قدم هایی است که مدل فرآیند تعریف می کند.

تکنولوژی مهندسی نرم افزار یک تکنولوژی لایه ای است و متدولوژی بروی لایه فرآیند قرار دارد.



روش‌هایی هستند که اعضای تیم با دنبال کردن آنها اطمینان خواهند داشت کار از ابتدا تا انتها به‌طور کامل و جامع انجام خواهد شد.

ابزارها

برنامه‌های کامپیوتری هستند که استفاده از تکنیک‌های خاصی را آسان می‌نمایند.

مانند ابزار Rational Rose که در تکنیک‌های تحلیل و طراحی بسیار مفید است و یا NUnit مربوط به تکنیک‌های تست نرم‌افزار می‌شود که توسط برنامه‌نویس انجام می‌شود و صدها برنامه کاربردی دیگر.

غیر از ابزارهایی که به ما در استفاده از تکنیک‌ها یاری می‌بخشند، ابزارهایی وجود دارند که اهداف دیگری مانند سرعت بخشیدن به تولید محصول را دنبال می‌کنند

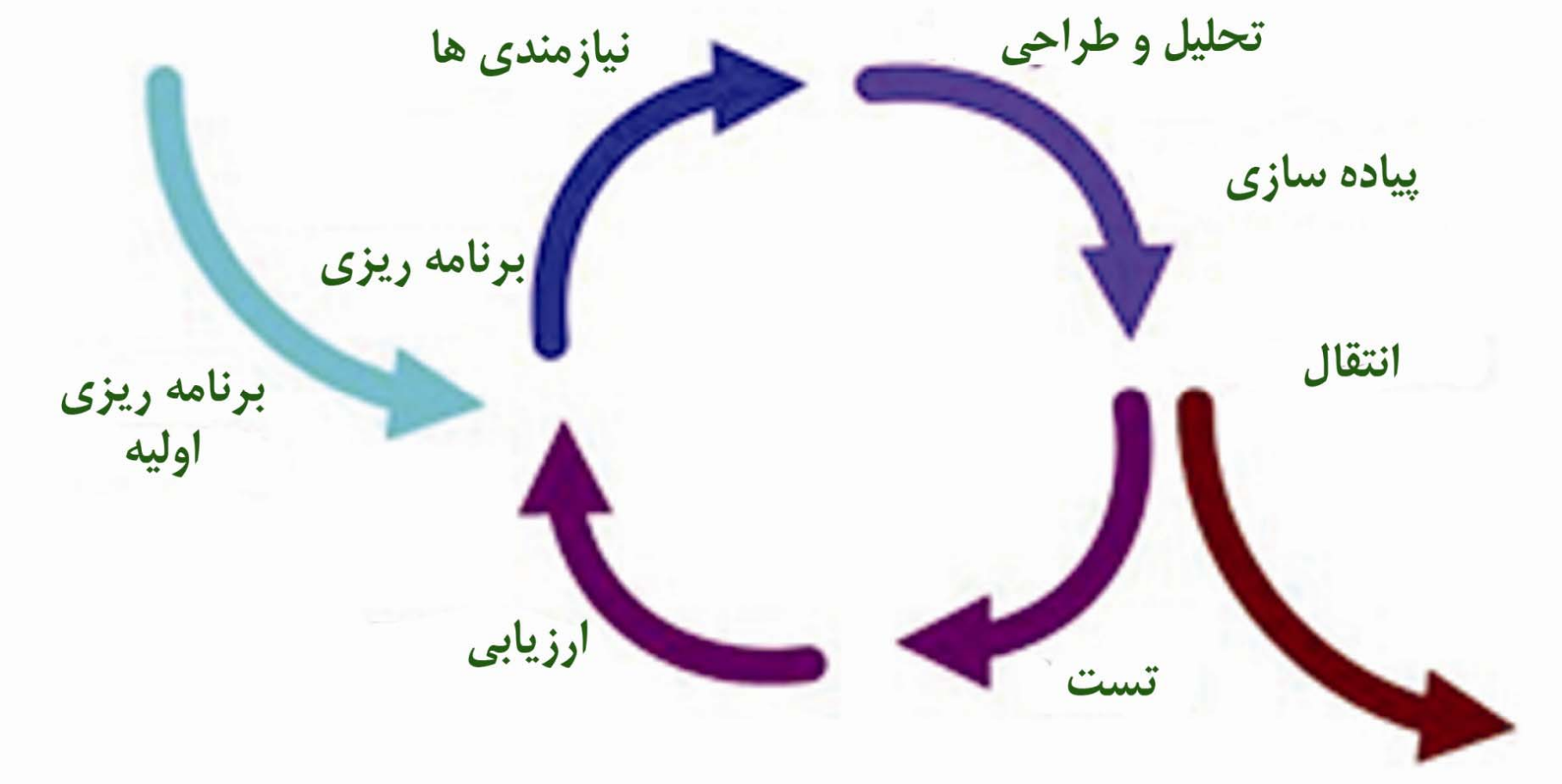
مانند تولید کننده کد Code Generator، ابزار مهندسی معکوس و ...

Software Development Life Cycle (SDLC)

هر فرآیند تولید و توسعه در هر صنعتی دارای یک چرخه‌ی حیات می‌باشد

چرخه‌ی حیات به دوره‌ای گفته می‌شود که یک فعالیت انجام می‌پذیرد

SDLC


نیازمندی ها
تحلیل و طراحی
پیاده سازی
انتقال
تست
ارزیابی
برنامه ریزی
برنامه ریزی
اولیه

این چرخه حیات به مدلی که برای تولید استفاده می شود (آبشاری، افزایشی، چرخشی، ...) بستگی دارد اما ...

در همه متدولوژی ها یک سری فعالیت مشخص همیشه وجود دارند و در متدولوژی ها فقط نحوه، زمان و شخص اجرا کننده آن تغییر می کند:

۱. نیازمندی ها
۲. تحلیل و طراحی
۳. پیاده سازی
۴. تست
۵. مدیریت پروژه و برنامه ریزی

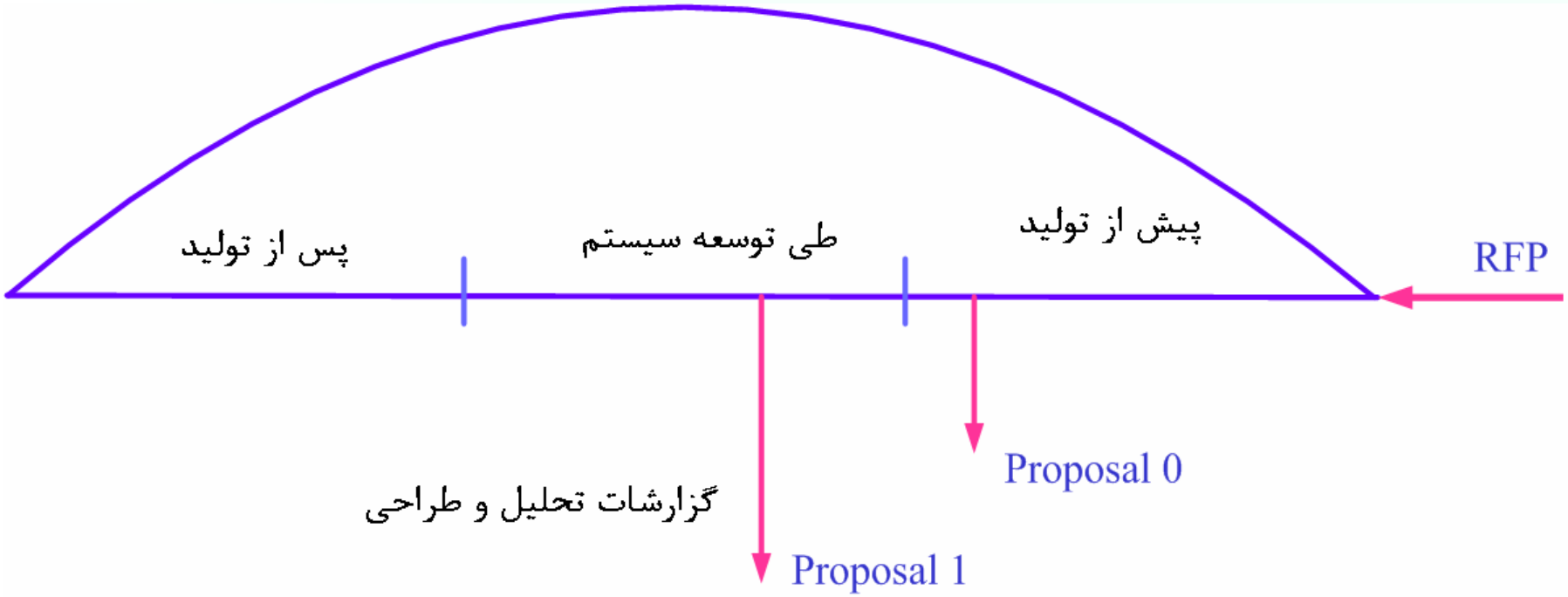
بررسی life cycle توسعه نرم افزار

مراحل مختلف توسعه نرم افزار : شناخت و تجزیه تحلیل ، طراحی ، پیاده سازی، تست و نگهداری می باشد.

Life cycle یک نرم افزار شامل تمام فعالیت های پیش از تولید ، توسعه و دوره اجرای نرم افزار است.

بررسی life cycle (پیش از تولید)

برای شروع پروژه، RFP (Request For Proposal) به تیم نرم افزاری ارائه می شود و تیم توسعه برای دریافت پروژه پیشنهاد خود را در قالب Proposal ارائه می کند.



مفاهیم مدیریت پروژه

فصل ۳ کتاب Pressman

درس مهندسی نرم افزار ۲

Email: ComputerCollege_Fam@Yahoo.Com

بعدهای مدیریت (۴P’s)

افراد (People)

محصول (Product)

فرآیند (Process)

پروژه (Project)

اصول W۵HH

افراد (People)

فرآیند توسعه نرم افزار توسط نقش های زیر اجرا می شود:

مدیر ارشد (Senior manager)

مدیر فنی پروژه (Project technical manager)

کارکنان-سازندگان(Practitioners)

مشتریان

کاربر نهایی

Email: ComputerCollege_Fam@Yahoo.Com

رهبر تیم

وظایف مدیری که رهبری یک پروژه را بر عهده دارد عبارت است از :

- انگیزش
- نظم دهی(سازماندهی)
- ارائه ایده ها و نوآوری
- حل مسائل
- انجام تشخیص های مدیریتی
- کنترل دستیابی به اهداف
- تشکیل و تاثیر گذاری بر تیم

افراد: تیم های نرم افزاری

ساختار مناسب برای یک تیم بستگی به نوع سازمان، تعداد افراد، توانایی های اعضای تیم ها و نوع و سختی کار دارد.

- تیم های دموکراتیک غیر متمرکز (DD)
- تیم های تحت کنترل غیر متمرکز (CD)
- تیم های کنترل شده متمرکز (CC)

فاکتورهای موثر بر ساختار تیم

تیم‌های با کارایی بالا

اعضای تیم باید به یکدیگر اعتماد کنند

توزیع مهارت‌ها باید براساس مسئله باشد

افراد مستقل و تک‌رو باید از تیم کنار گذاشته شوند

- فضای کار آشفته
  - در این فضا، تیم انرژی خود را هدر می‌دهد
  - تمرکز تیم از مسئله اصلی به مسائل پیش‌پا افتاده سوق پیدا می‌کند
- ناامیدی بالا
  - سبب می‌شود اصطکاک بین اعضای تیم بوجود آید
- شیوه ضعیف هدایت یا مدل فرآیند ضعیف
  - سبب عدم اجرای پروژه و عدم دستیابی به اهداف پروژه می‌شود

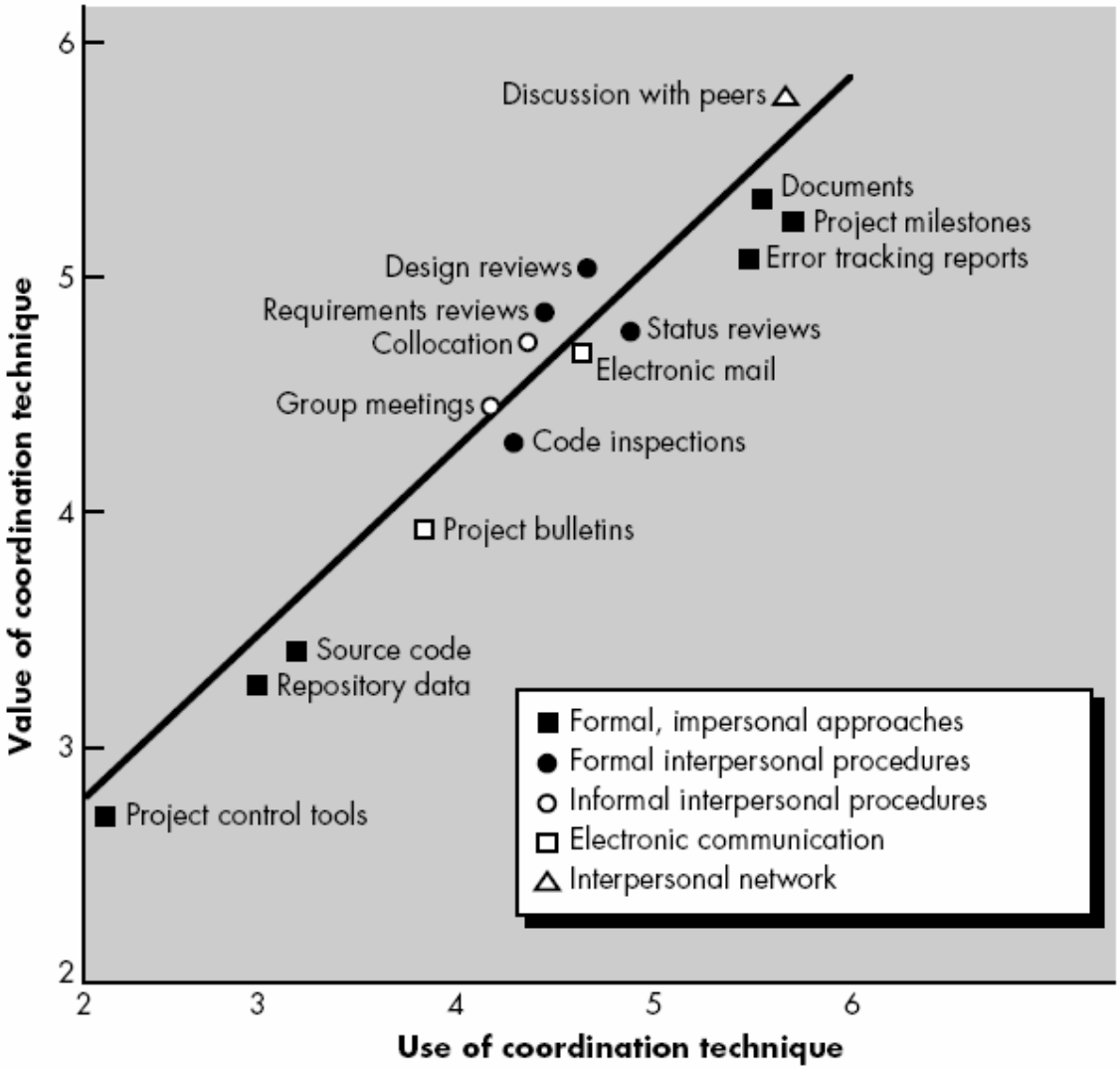
سم تیم‌های پروژه (ادامه)

- عدم تعریف شفاف نقش‌ها
  - سبب ایجاد عدم تعهد و فرافکنی در تیم می‌شود
- شکست مکرر و مدام
  - سبب ایجاد عدم اعتماد و کاهش دلگرمی می‌شود

روش‌های هماهنگی بین تیم

- روش‌های غیرشخصی و رسمی (*Formal, Impersonal*)
  - مستندسازی، فرسنگ‌شمار پروژه، زمانبندی، ابزارهای کنترل پروژه، گزارش‌گیری از مشکلات و رهگیری حل آنها
- روش‌های میان فردی، غیر رسمی (*Informal, Interpersonal*)
  - جلسات گروهی برای رفع مشکلات و حل مسئله
- ارتباط الکترونیکی (*Electronic Communication*)
  - شامل ارسال E-Mail، ویدئو کنفرانس، استفاده از *Sharepoint* و ...
- شبکه میان افراد (*Interpersonal Networking*)
  - جلسات غیررسمی با اعضای تیم و افرادی که تجربه دارند

روش‌های هماهنگی بین تیم (ادامه)



تعیین حیطه محصول

شرحی که محدوده پروژه را نشان می‌دهد

تجزیه مسئله

تجزیه وظیفه‌مندی مسئله به زیرمسائل

۴ نوع دسته بندی سازمان ها:

مدل بسته (Close Paradigm): دارای ساختارهای سنتی (شبیه CC) برای تولید نرم افزارهایی مناسب است که تجربه تولید آنها در گذشته وجود دارد.

مدل تصادفی(Random paradigm): تیم بر افراد و نوآوری آنها تکیه دارد. هنگامی که نیاز به نوآوری وجود دارد از این نوع تیم ها استفاده می شود اما کارایی پایینی دارند.

مدل باز : سعی می کند خصوصیات نوع بسته و نوع تصادفی را دارا باشد، فعالیت ها با همکاری انجام می شوند. برای توسعه سیستم های پیچیده مناسب هستند.

مدل همگام (Synchronous Paradigm): بر تقسیم کار بین افراد تیم تاکید دارد و هر عضو تیم بر روی اندازه کوچکی از کار، فعالیت می کند.

افراد : ارتباط میان اعضای تیم ها

تکنیک های همکاری و ارتباط در پروژه ها :

روش های رسمی و غیر شخصی : شامل: مستندات مهندسی نرم افزار، برنامه های پروژه و گزارش های موجود

رسمی و بین اشخاص: تمرکز بر فعالیت های QA دارد مانند: جلسات بازبینی و مرور

غیر رسمی و بین اشخاص : جلسات گروهی برای بحث و تبادل نظر و حل مسائل

ارتباطات الکترونیکی : مانند پست الکترونیک، بولتن های الکترونیک، ویدئو کنفرانس و ..

شبکه بین اشخاص : شامل بر بحث های غیر رسمی میان افراد گروه و افراد خارج از پروژه

محصول

محدوده نرم افزار (scope): یکی از ابتدایی ترین فعالیت های مدیریت پروژه، تشخیص محدوده محصول است. محدوده با پاسخ به سوالات زیر در رابطه با نرم افزار به دست می آید.

مفهوم

اهداف اطلاعات

وظایف و کارایی

تجزیه مسئله (Problem Decomposing): این فعالیت یک فعالیت پایه در آنالیز سیستم است

فرآیند

تعریف، توسعه و پشتیبانی در توسعه تمامی سیستم های نرم افزار مشترک هستند. مسئله انتخاب یک مدل فرآیند مناسب با پروژه مربوطه است.
مدیر پروژه باید در رابطه با مدل فرآیند توسعه پروژه تصمیم گیری نماید.

۷ سوال که "بوهم" برای یافتن اهداف و خصوصیات پروژه طرح نموده است :

Why is the system being developed ?

چرا سیستم باید تولید شود؟

What will be done , by when?

چه کاری و تا چه زمانی باید انجام شود؟

Who is responsible for a function ?

چه کسی مسئول یک وظیفه است؟

Where are they organizationally located ?

آنها (اعضای تیم توسعه) در چه محلی مستقر و ساماندهی شده اند ؟

How will the job be done technically and managerially ?

کارها از نظر فنی و مدیریتی چگونه انجام خواهند شد؟

How much of each resource is needed ?

چه مقدار از هر منبعی مورد نیاز می باشد ؟

فصل سوم

مفاهیم و اصول شی گرایی

درس مهندسی نرم افزار ۲

مفاهیم کلیدی این فصل

مدل فرآیند شی‌گرا

مفاهیم شی‌گرایی

محصورسازی، وراثت و چندریختی

تعیین عناصر مدل شی

معیارهای انتخاب اشیا

مدل بازگشتی/موازی

معیارهای پروژه‌های شی‌گرا

تاریخچه شی‌گرایی

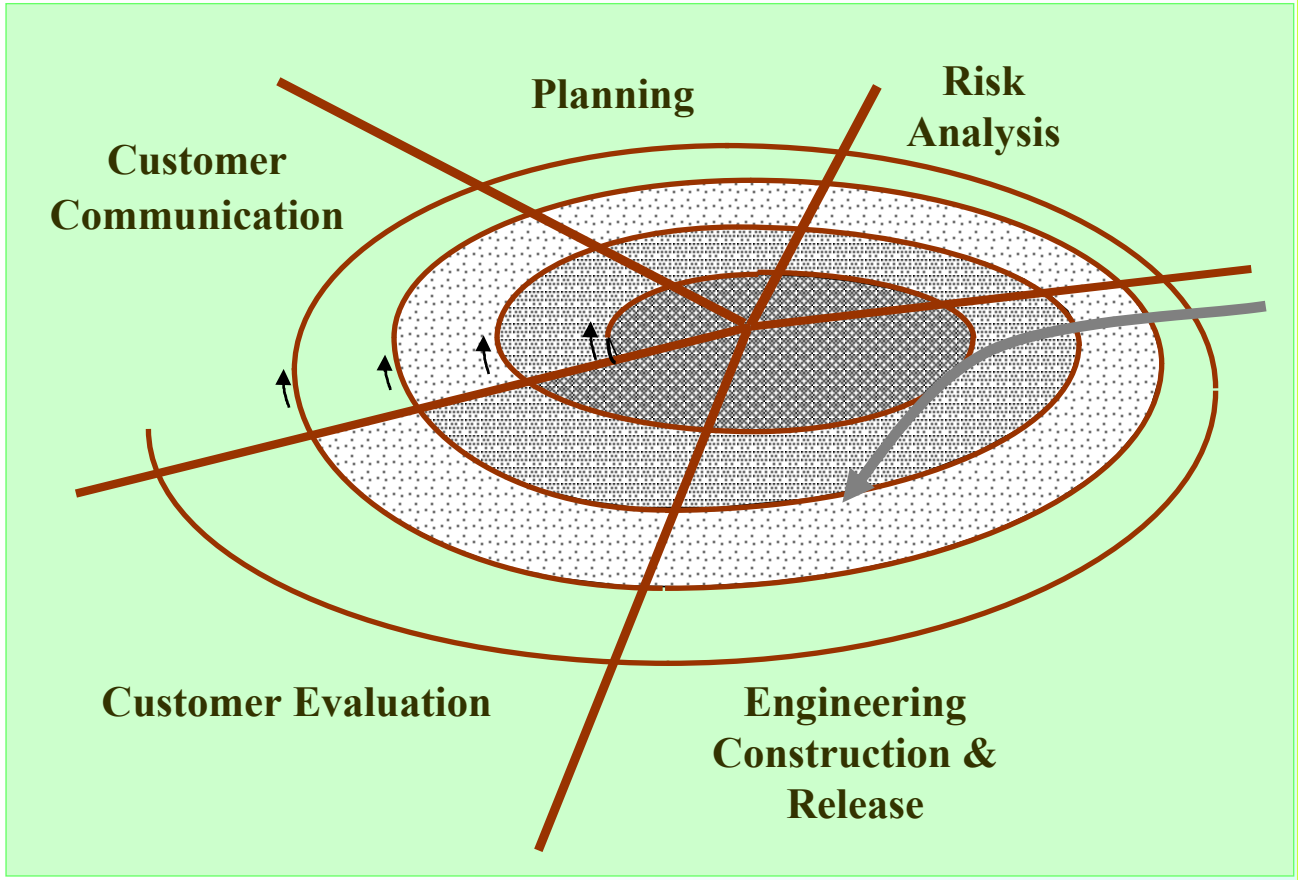
اولین بار در اواخر دهة ۱۹۶۰ برای توسعة نرم‌افزار به‌کار گرفته شد
در دهه ۱۹۹۰ شی‌گرا مورد توجه توسعه‌دهندگان قرار گرفت و جایگزین روش‌های کلاسیک شد

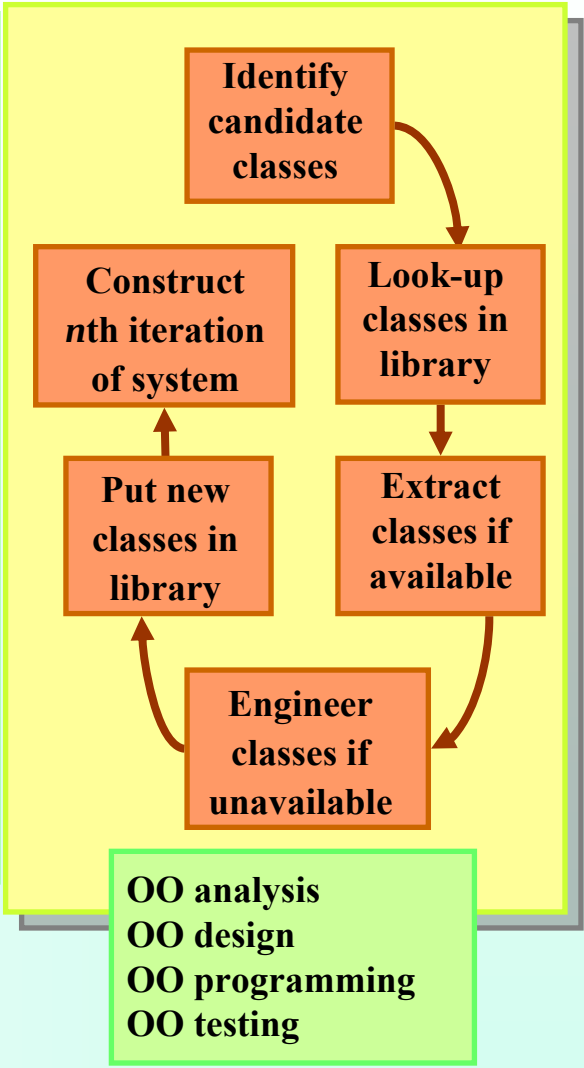
- شی‌گرایی منجر به استفاده مجدد می‌شود
- استفاده مجدد منجر به توسعه سریعتر نرم‌افزارهای با کیفیت بالاتر می‌شود

- نگهدرای نرم‌افزارهای شی‌گرا آسانتر است
- ساختار آن ذاتاً فاقد پیوستگی است

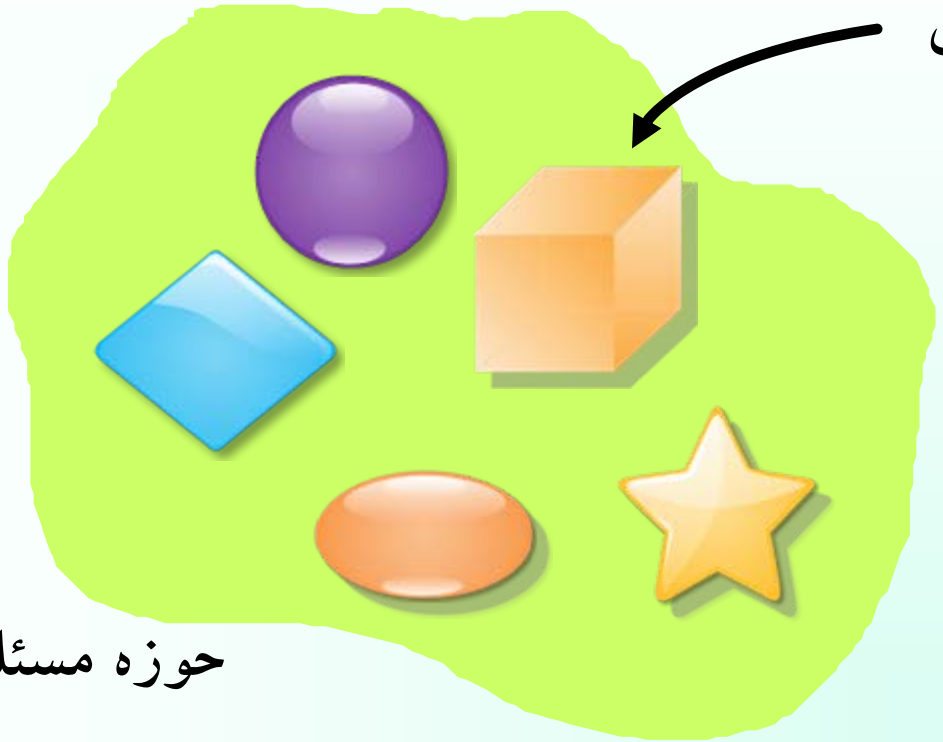
- تطبیق و تغییر دادن اندازة سیستم‌های شی‌گرا آسانتر است

مدل فرآیند شی گرا





مفاهیم شی گرا



مفاهیم شی گرا

Coad and Yourdon شی گرایی را حاصل مفاهیم زیر می‌دانند

object-oriented = objects + classification + inheritance + communication

مفاهیم کلیدی

کلاس‌ها و اشیاء

صفات و عملیات

محصورسازی (*Encapsulation*)، نمونه‌سازی (instantiation)
وراثت (*Inheritance*) و پیمانه ای بودن(Modularity)

Email: ComputerCollege_Fam@Yahoo.Com

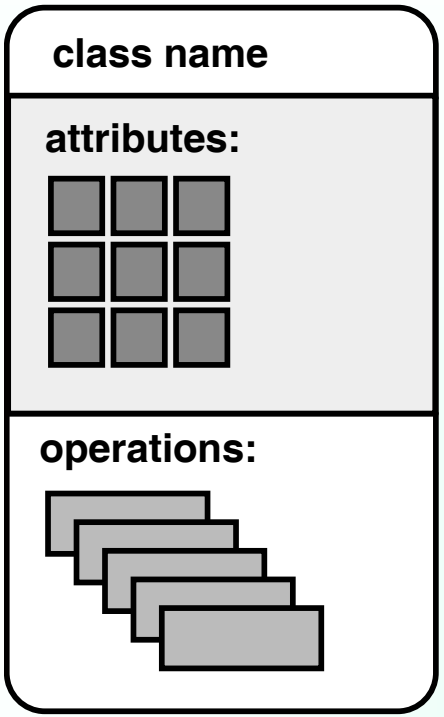
کلاس (Class)

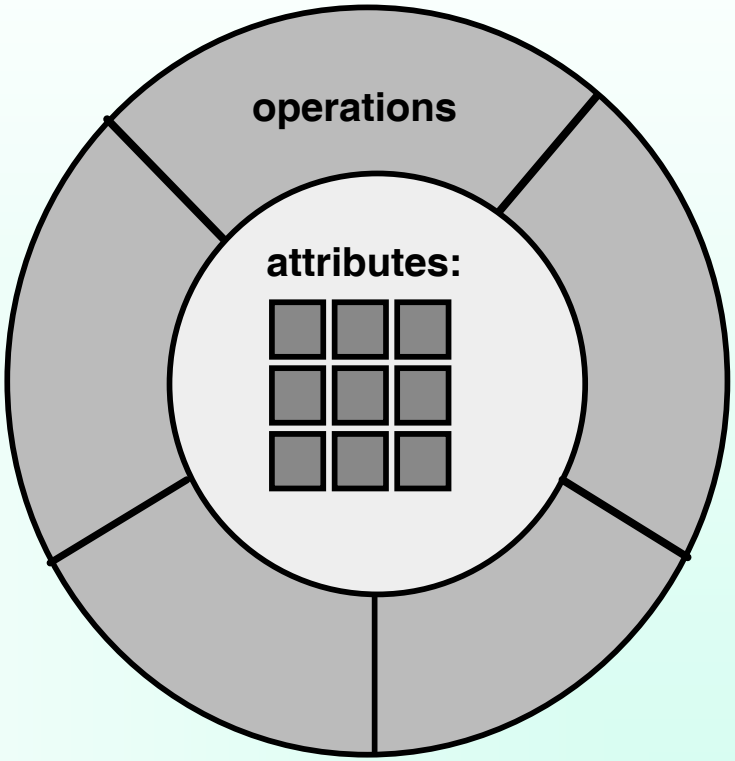
هر کلاس حاوی داده و عملیات برای توصیف رفتار و عملکرد یک موجودیت واقعی است

کلاس، یک توصیف عمومی (نمونه، الگو یا ...) است که مجموعه‌ای از عناصر مشابه را توصیف می‌کند

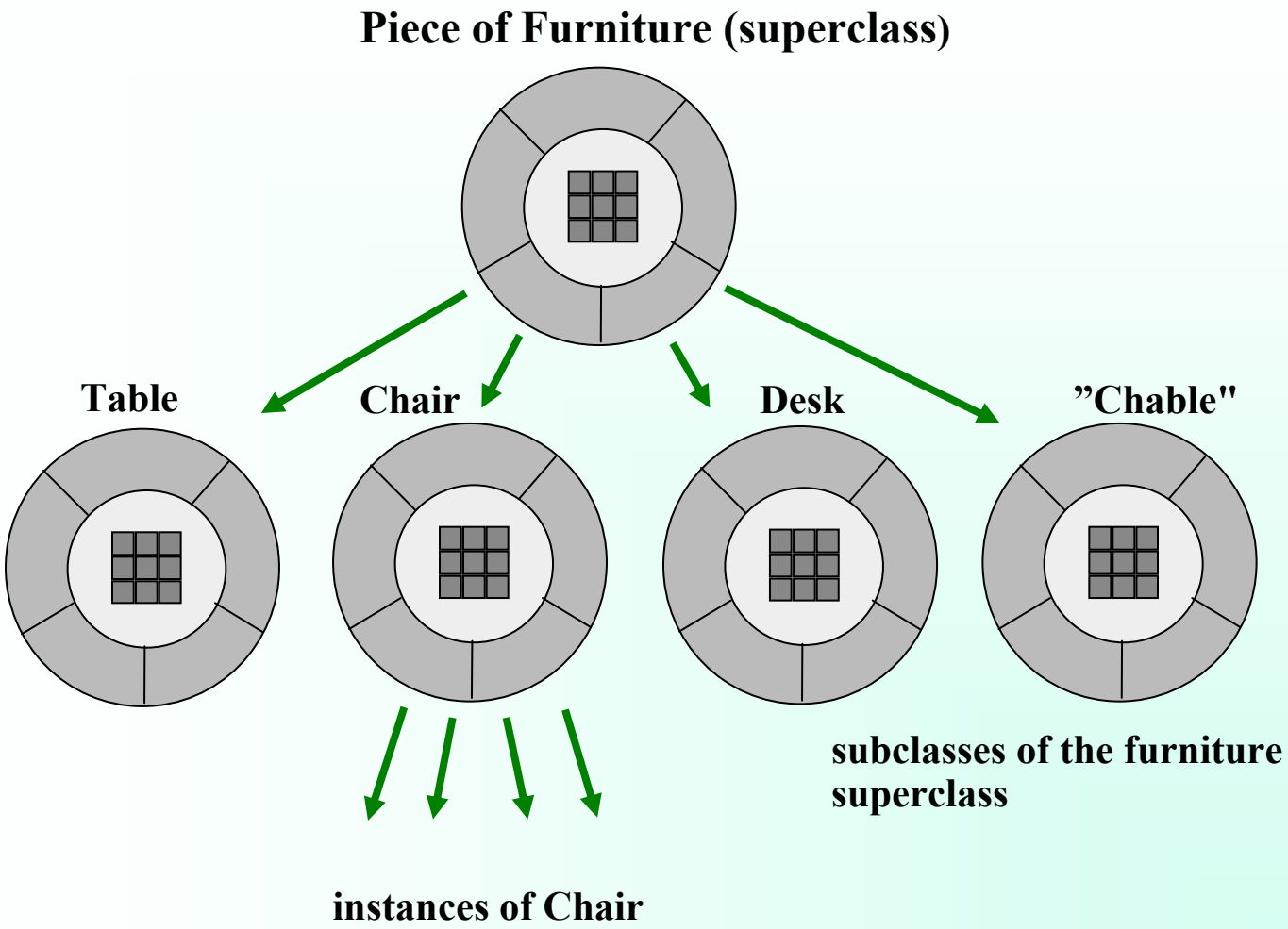
ابرکلاس (*Superclass*) سلسله مراتبی از کلاس‌ها ایجاد می‌کند
همینکه کلاسی از اقلام تعریف شدند، نمونه خاصی (*instance*) از کلاس می‌تواند مشخص شود

نمایش کلاس





سلسله مراتب کلاس

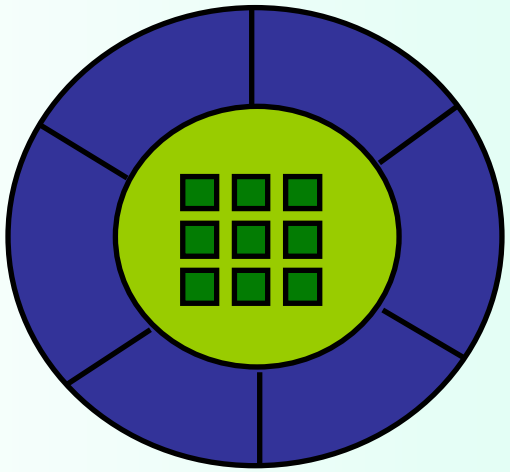


Email: ComputerCollege_Fam@Yahoo.Com

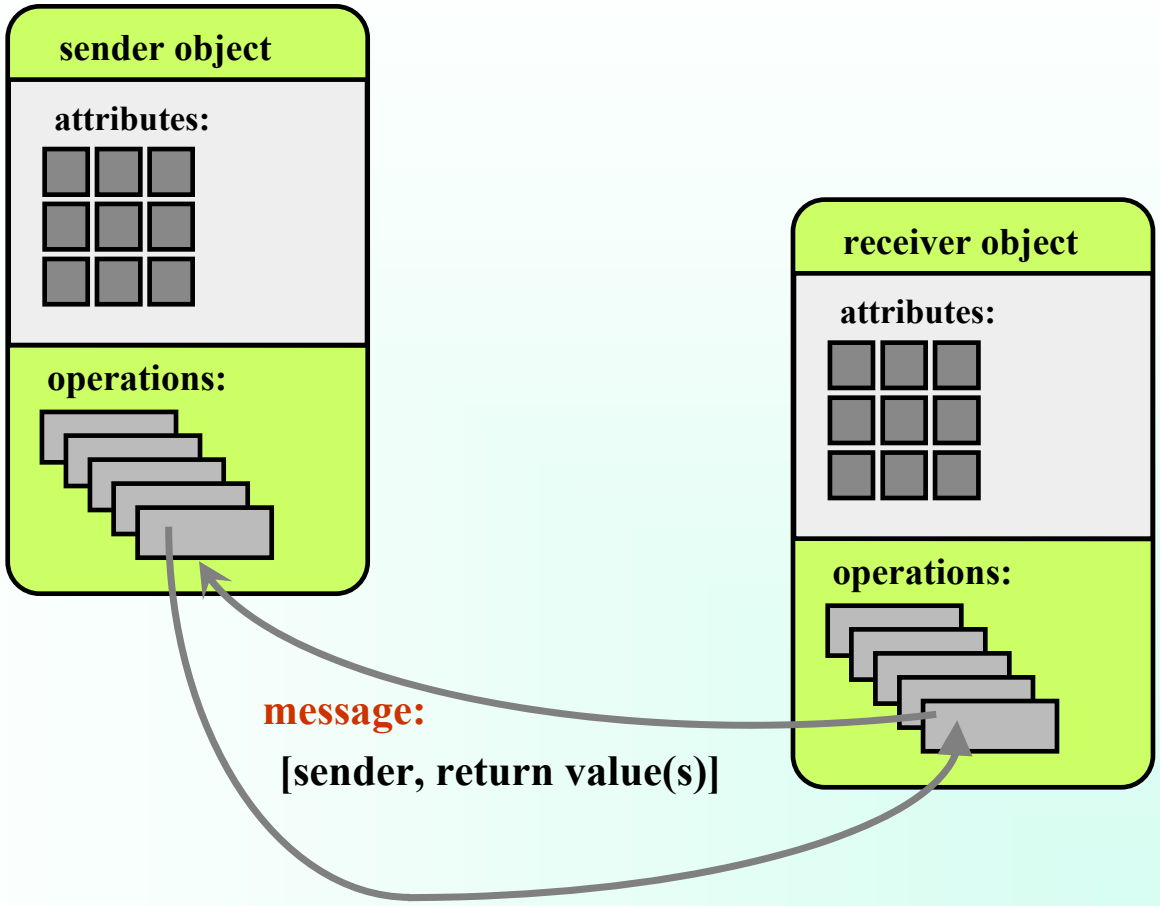
یک رویه محصور شده که در یک کلاس قرار دارد

بر روی یک یا بیشتر قلم داده‌ای کلاس، عملیات خود را انجام می‌دهد

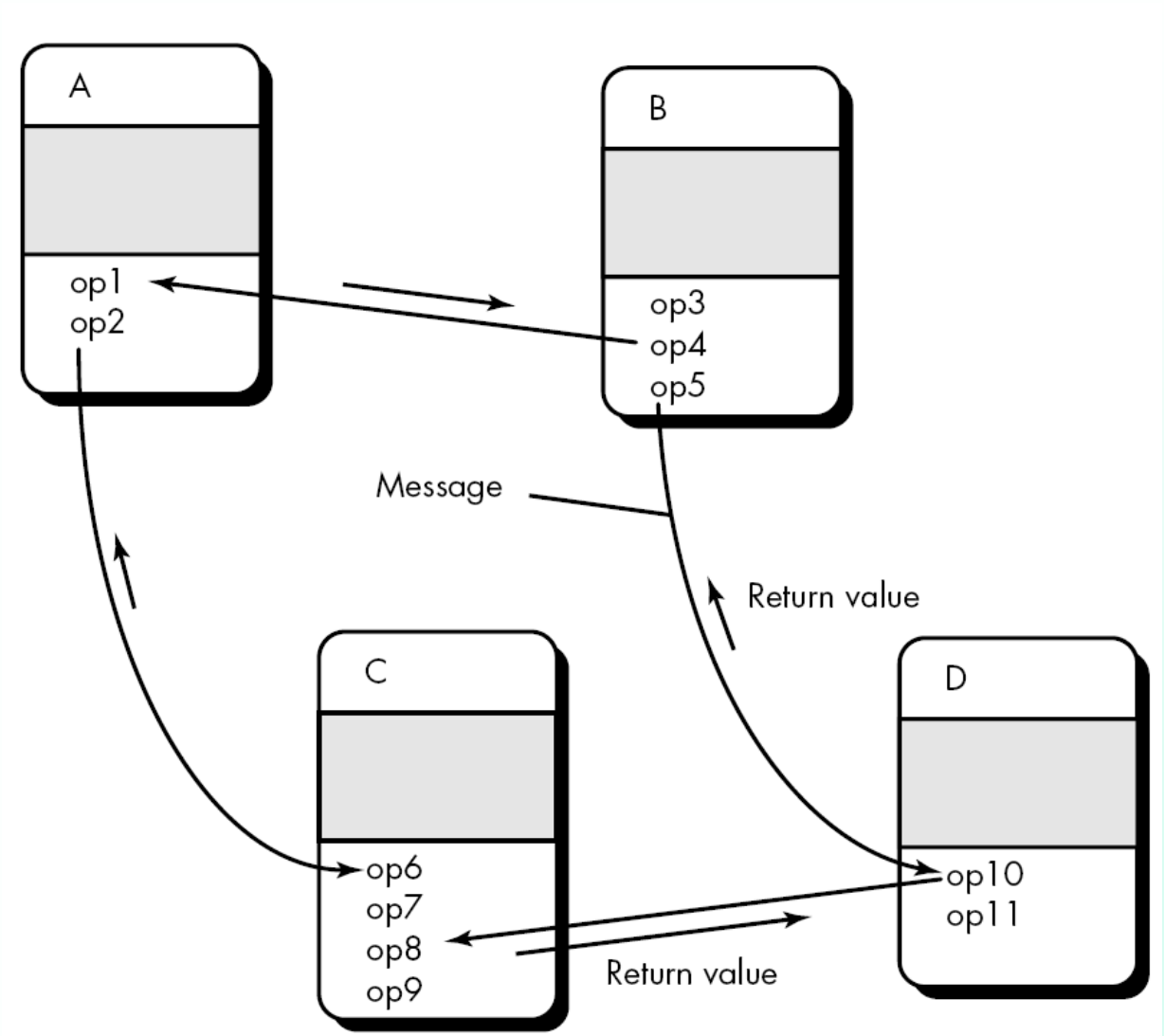
هر متد توسط ارسال پیام (Message Passing) فراخوانی می‌شود



پیام‌ها (Messages)

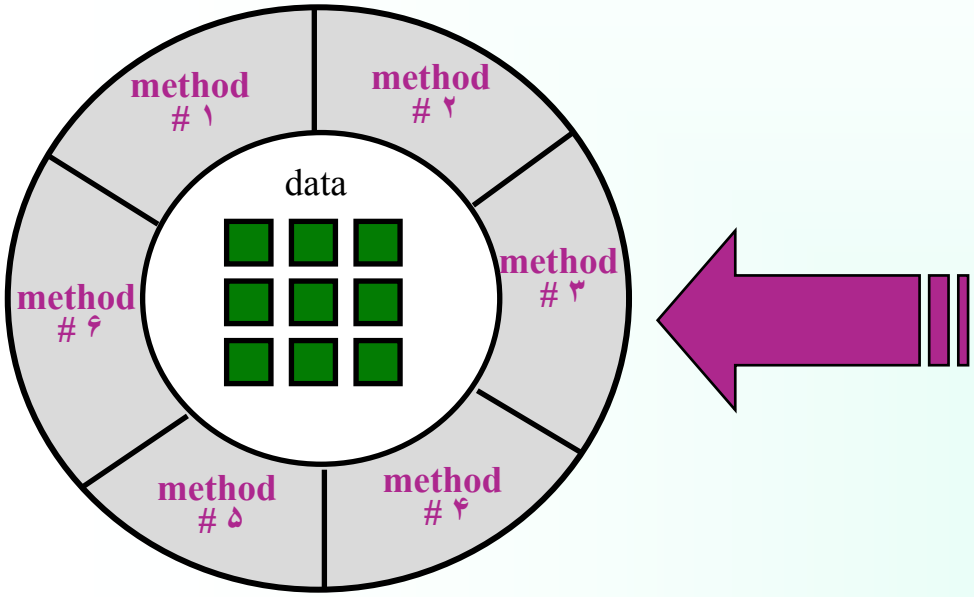


ارسال پیام


A
op1
op2
B
op3
op4
op5
Message
Return value
C
op6
op7
op8
op9
D
op10
op11
Return value

محصورسازی (Encapsulation)

هر شی؛ داده و رویه‌های مورد نیاز برای اصلاح داده‌ها را محصور می‌نماید



محصورسازی سبب پنهان‌سازی داده‌ها می‌شود

مزایای محصورسازی

جزییات پیاده‌سازی داده‌ها و رویه‌ها از دنیای خارج پنهان می‌مانند (پنهان‌سازی اطلاعات)

سبب کاهش انتشار اثرات جانبی در هنگام تغییرات می‌شود

ساختارهای داده و عملیات (که آنها را تغییر می‌دهند) در یک موجودیت مجزا به نام کلاس با یکدیگر ادغام می‌شوند

سبب تسهیل استفاده مجدد از مولفه می‌شود

رابط‌های بین اشیای محصور شده ساده‌تر می‌شوند

شی فرستنده پیام نیاز نیست که به جزییات داخلی ساختارهای داده توجه کند و اتصال (Coupling) سیستم کاهش یابد

وراثت (Inheritance)

مهندس نرم‌افزار برای ایجاد یک کلاس جدید، گزینه‌های طراحی زیر را دارد

- کلاس از ابتدا طراحی و ساخته می‌شود
  وراثت مورد استفاده قرار نگرفته است

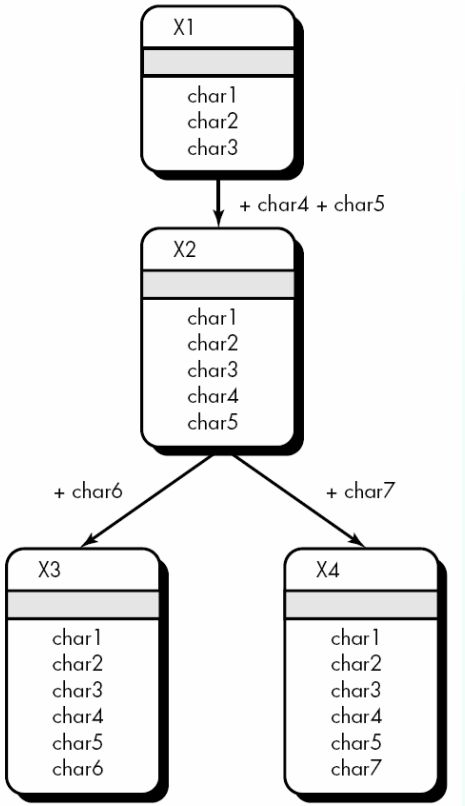
- سلسله مراتب کلاس می‌تواند جستجو شود و مشخص شود که آیا کلاس بالاتری در سلسله مراتب (Superclass) وجود دارد که دارای بیشترین خصوصیات و عملیات مورد نیاز باشد
  کلاس جدید از کلاس یافت شده، ارث می‌گیرد و در صورت وجود عملیات و خصوصیات بیشتر، به کلاس جدید افزوده می‌شوند

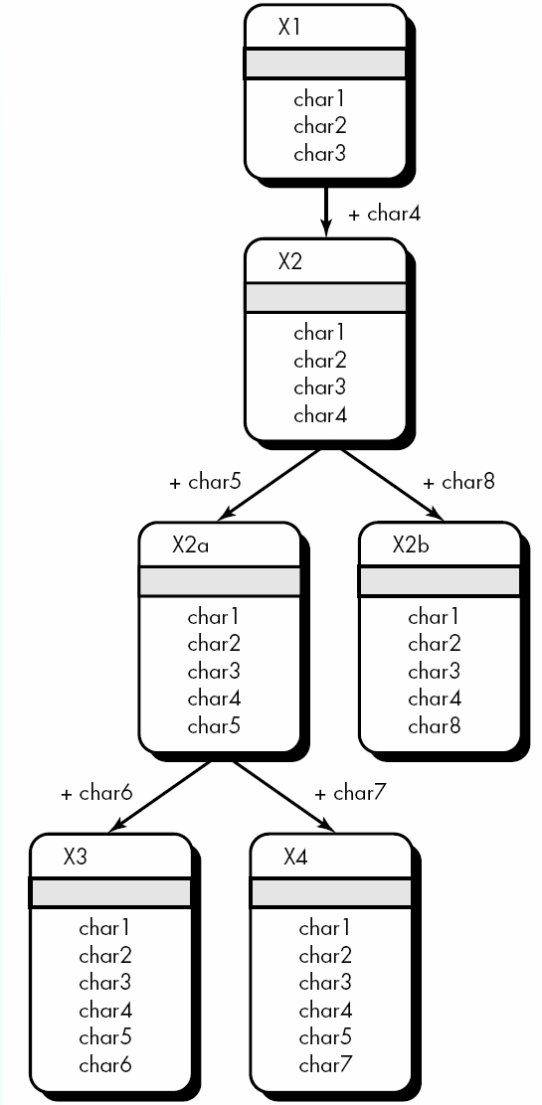
گزینه‌های طراحی (ادامه)

- سلسله مراتب کلاس مجددا سازماندهی شود تا خصوصیات و عملیات مورد نیاز بتوانند توسط کلاس جدید به ارث برده شوند

- خصوصیات یک کلاس موجود می‌توانند باطل شوند و نسخه های خصوصی از داده و عملیات برای کلاس جدید پیاده سازی شوند

وراثت (Inheritance)





چندریختی (Polymorphism)

با استفاده از چندریختی، چندین عملیات می‌توانند از یک نام استفاده کنند

سبب کاهش تعداد خطوط برنامه و تسهیل اعمال تغییرات می‌شود

روش سنتی

```
case of graphtype:
        if graphtype = linegraph then DrawLineGraph (data);
        if graphtype = piechart then DrawPieChart (data);
        if graphtype = histogram then DrawHisto (data);
        if graphtype = kiviat then DrawKiviat (data);
end case;
```

چندریختی

```
graphtype draw
```


Email: ComputerCollege_Fam@Yahoo.Com

تعیین عناصر مدل شی

مراحل تعیین عناصر مدل شی:

تعیین کلاس‌ها و اشیا

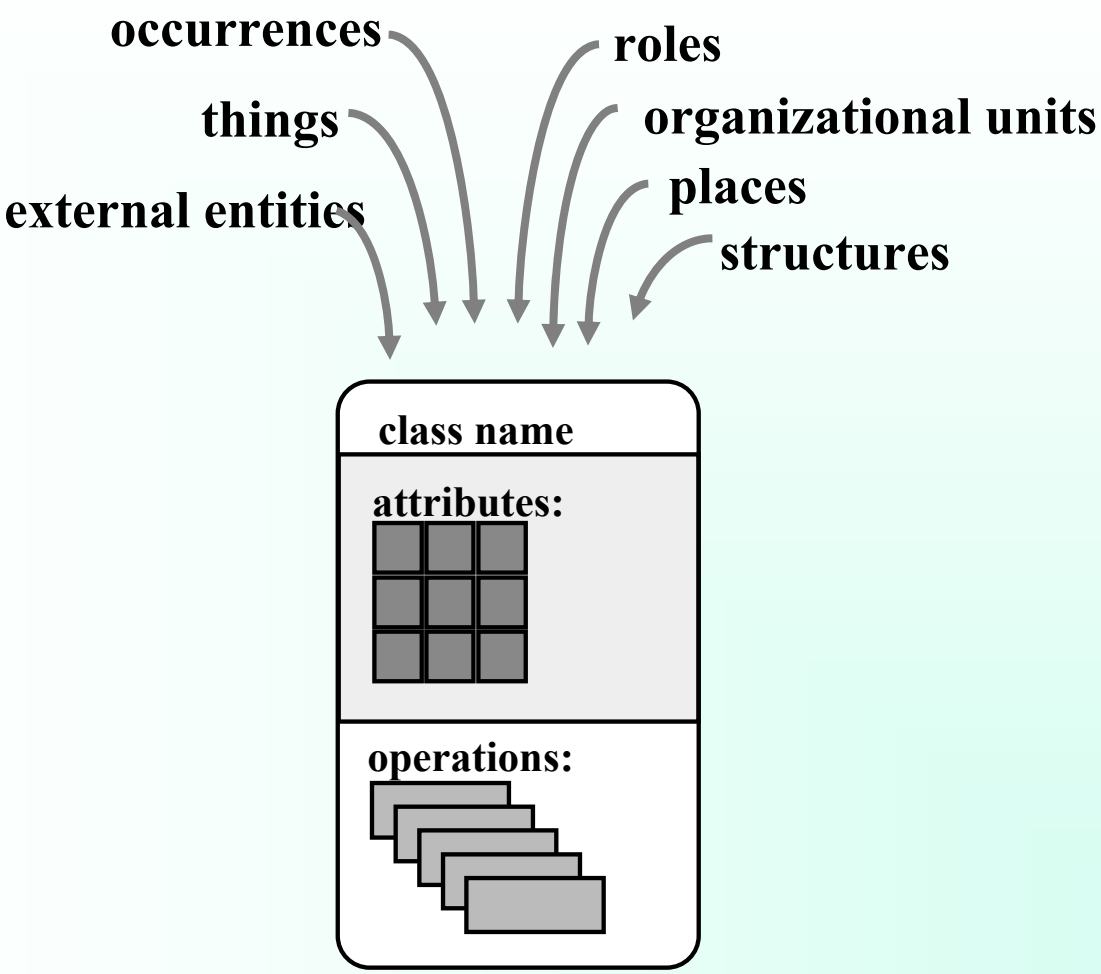
تعیین صفات

تعریف عملیات

خاتمه تعریف شی

Email: ComputerCollege_Fam@Yahoo.Com

انواع کلاس


occurrences
things
external entities
roles
organizational units
places
structures
class name
attributes:
operations:

انواع اشیا

- موجودیت‌های خارجی (External Entities)

همانند سیستم‌های دیگر، ابزارها، افراد و ... که اطلاعات مورد استفاده سیستم کامپیوتری را تولید یا مصرف می‌کنند

- چیزها (Things)

همانند گزارشات، صفحات نمایش، نامه‌ها، علائم الکترونیکی و ... که بخشی از دامنه اطلاعاتی مسئله محسوب می‌شوند

- اتفاقات یا رویدادها (Occurrences or events)

همانند دریافت کارت، اتمام کار و ... که در حیطه عملکرد سیستم رخ می‌دهند

تعیین کلاس‌ها و اشیا

- نقش‌ها (Roles)

همانند مدیر، فروشنده و ... که افراد در حال تعامل با سیستم بر عهده دارند

- واحدهای سازمانی (Organizational units)

همانند بخش، گروه، تیم و ... که با یک کاربرد خاص سروکار دارند

- مکان‌ها (Places)

همانند سالن، کلاس و ...که حیطه مسئله و وظیفه کلی سیستم را مشخص می‌کند

- ساختارها (Structures)

همانند حسگرها، کامپیوترها و ... که کلاسی از اشیا را تعریف می‌کنند

معیارهای انتخاب اشیاء

خصوصیات گزینش اشیا (Coad and Yourdon):

- اطلاعات نگهداری شده (Retained information)

  اطلاعات آن برای انجام عملیات سیستم مورد نیاز باشد

- سرویس‌های مورد نیاز (Needed services)

  شی بالقوه باید مجموعه‌ای از عملیات قابل شناسایی داشته باشد که می‌توانند صفات آنرا به طریقی تغییر دهند

- صفات چندگانه (Multiple attributes)

  یک شی با یک صفت ممکن است در طول طراحی مفید باشد ولی بهتر است در هنگام تحلیل به عنوان صفتی از شی دیگر در نظر گرفته شود

Email: ComputerCollege_Fam@Yahoo.Com

– صفات مشترک (*Common attributes*)

- مجموعه‌ای از صفات می‌توانند برای یک شی بالقوه تعریف شوند که برای تمام نمونه‌های شی بکار می‌روند

– عملیات مشترک (*Common operations*)

- مجموعه‌ای از عملیاتی که می‌توانند برای تمام نمونه‌های شی بکار می‌روند

– نیازمندی‌های ضروری (*Essential requirements*)

- موجودیت‌های خارجی که در فضای مسئله ظاهر می‌شوند و اطلاعات ضروری برای کارکرد هر راه‌حل سیستم را تولید یا مصرف می‌کنند

نمونه‌ای از یک شی

Class name

Object *System*

System ID
Verification Phone Number
System Status
Sensor Table
- **Sensor Type**
- **Sensor Number**
- **Alarm Threshold**

Alarm delay time
Telephone Number (s)
Master Password
Temporary Password
Number of tries

attributes

Program
Display
Reset
Query
Modify
Call

operations

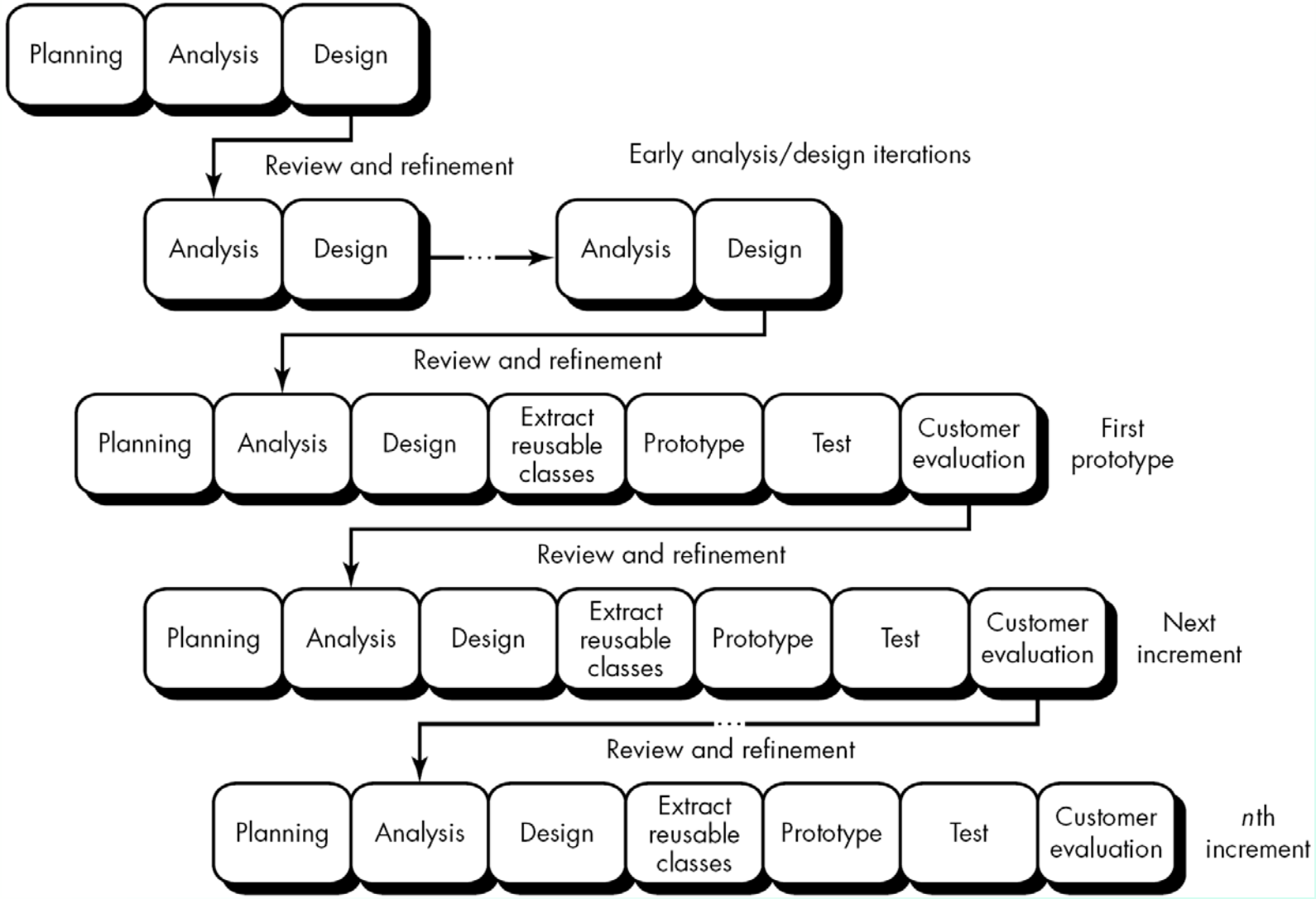
مدل بازگشتی/موازی (Recursive/Parallel)

- انجام تحلیل کافی برای جداسازی کلاس‌ها و ارتباطات اصلی مسئله
- انجام اندکی طراحی برای تعیین اینکه آیا کلاس‌ها و ارتباطات عملاً قابل پیاده‌سازی هستند
- استخراج اشیای قابل استفاده مجدد از یک کتابخانه برای ساخت یک نمونه اولیه
- اجرای چند آزمون برای کشف خطاها در نمونه اولیه
- دریافت نظرات مشتری در مورد نمونه اولیه

مدل بازگشتی/موازی (Recursive/Parallel)

- اصلاح مدل تحلیل براساس آنچه که از نمونه اولیه، انجام طراحی و نظرات مشتری فرا گرفتید
- پالایش طراحی به منظور انجام تغییرات
- برنامه‌نویسی اشیای خاص (که در کتابخانه موجود نیستند)
- مونتاژ یک نمونه جدید با استفاده از اشیای کتابخانه و اشیای جدیدی که ایجاد نمودید
- اجرای چند آزمون برای کشف خطاها در نمونه
- دریافت نظرات مشتری در مورد نمونه

نمونه فرآیند برای پروژه شی گرا



معیارهای پروژه‌های شی‌گرا

معیارهای پیشنهادی پروژه‌های شی‌گرا (*Lorenz and Kidd*)

تعداد متون سناریو (scenario scripts)

تعداد کلاس‌های کلیدی (key classes)

تعداد کلاس‌های پشتیبان (*support classes*)

تعداد متوسط کلاس‌های پشتیبان به ازای هر کلاس کلیدی

تعداد زیرسیستم‌ها

روش‌های آزمایش نرم‌افزار

درس مهندسی نرم افزار ۲

Email: ComputerCollege_Fam@Yahoo.Com

مفاهیم کلیدی

- آزمایش نرم‌افزار
- چه کسانی نرم‌افزار را آزمایش می‌کنند؟
- اصول آزمایش
- قابلیت آزمایش
- طراحی داده‌های آزمایش (*test case*)
- آزمایش جعبه سفید
- آزمایش جعبه سیاه

آزمایش نرم‌افزار

آزمون نرم افزار چیست؟

هدف طراحی یک سری موارد آزمون است که یافتن خطاها را به گونه ای مناسب پوشش دهد. این تکنیک ها راهنمای سیتماتیکی برای طراحی آزمون هایی می کنند که:

۱)منطق درونی اجزای نرم افزاری را بررسی کرده

۲)حوزه ورودی و خروجی برنامه را برای مشخص کردن خطاهای عملکرد برنامه می آزماید.

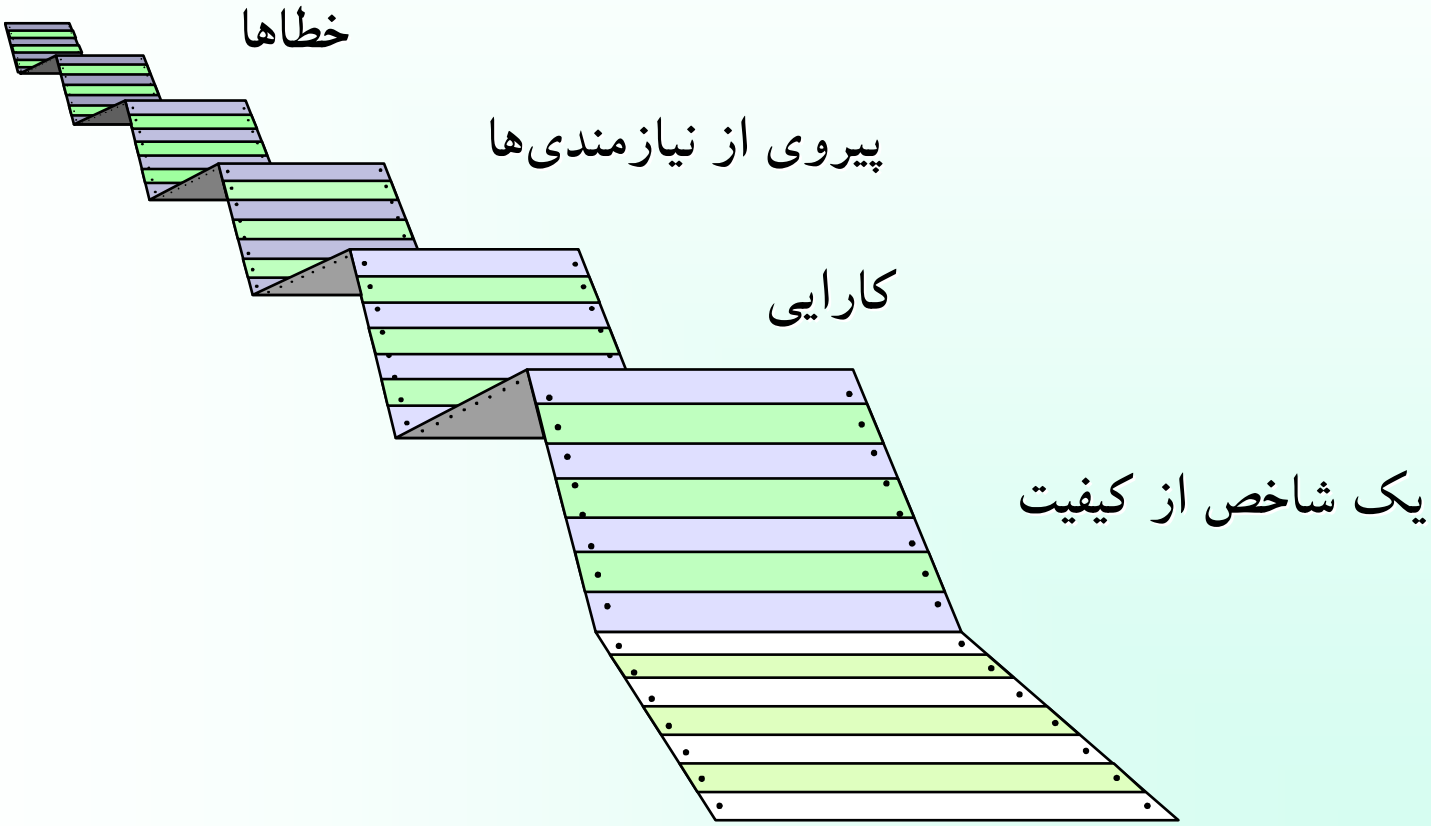
اهداف آزمایش نرم‌افزار (Glen Myers)

آزمایش، فرآیند اجرای برنامه با هدف یافتن خطا است

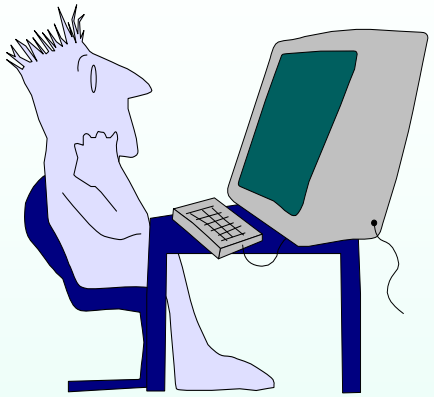
یک داده آزمایش (test case) خوب، نمونه‌ای است که با احتمال بالایی خطاها را بیابد

آزمایش موفق، آزمایشی است که خطاهای تاکنون یافت نشده را بیابد

آزمایش نرم‌افزار چه چیزی را نشان می‌دهد؟

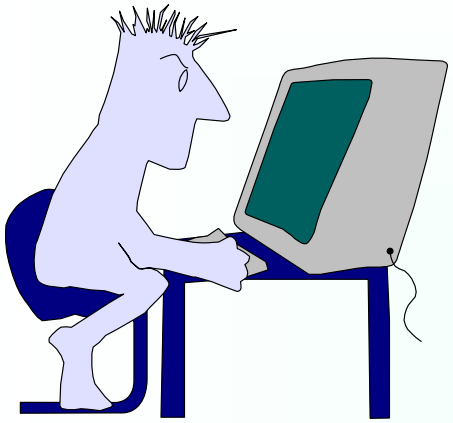


چه کسی نرم‌افزار را آزمایش می‌کند



توسعه دهنده

سیستم را درک می‌کند،
اما آن را بتدریج آزمایش می‌کند
و هدفش تحویل نرم‌افزار است



آزمایش‌کننده مستقل

باید عملکرد سیستم را فراگیرد
اما تلاش می‌کند آن را از کار بیندازد
و هدفش کیفیت است

و در نهایت کاربر نهایی سیستم را آزمایش می‌کند...

اصول آزمایش نرم‌افزار (Davis)

- تمام آزمایشات باید به نیازمندی‌های مشتری قابل ردگیری باشند
- آزمایشات باید مدتی طولانی قبل از شروع، برنامه‌ریزی شوند
- آزمایش باید با اجزاء کوچک شروع شود و به سمت آزمایش کل سیستم پیش رود
- آزمایش کامل و جامع امکان‌پذیر نیست
- به منظور تاثیرگذاری بیشتر، آزمایش باید توسط تیم مستقلی هدایت شود
- اصل pareto برای آزمایش نرم‌افزار به کار رود: اصل pareto می‌گوید" ۸۰ درصد خطاها به خاطر ۲۰ درصد مؤلفه‌ها می‌باشد." بنابراین ۲۰ درصد مؤلفه‌ها شناسایی و کاملاً مورد آزمایش قرار می‌گیرد.

قابلیت آزمایش

- عملیاتی بودن (Operability)

  نرم‌افزار هرچه بهتر کارکند، با کارایی بالاتری می‌تواند آزمایش شود

- قابلیت مشاهده (Observability)

  آنچه را می‌توانید ببینید، می‌توانید آزمایش کنید

- قابلیت کنترل (Controllability)

  هر چه نرم‌افزار بیشتر قابل کنترل باشد، آزمایش بیشتر به طرز خودکار و بهینه قابل انجام است

- پایداری (Stability)

  هر چه تغییرات کمتر باشد، انحراف از آزمایش کمتر است

قابلیت آزمایش

- سادگی (Simplicity)

هر چه موارد برای آزمایش کمتر باشد، آزمایش با سرعت بیشتری انجام می‌گیرد

- قابلیت تجزیه‌پذیری (Decomposability)

با کنترل نمودن محدودهٔ آزمایش، با سرعت بیشتری مسایل تجزیه می‌شوند و آزمایشات هوشمندانه‌تری انجام می‌شود

- قابلیت فهم (Understandability)

هر چه اطلاعات بیشتری در اختیار داشته باشد، آزمایش هوشمندانه‌تری انجام می‌شود

آزمایش خوب

- با احتمال بالا، خطاها را نشان می‌دهد
- تکراری نیست
- جامعیت خوب و مناسب دارد
- نه بسیار ساده و نه بسیار پیچیده است

اصول و مبانی آزمون

در واقع آزمون مرحله ای از فرآیند نرم افزاری است که می توان آن را بیشتر ویران کننده دانست تا سازنده.

آزمون مستلزم این است که تولید کننده نکات و عبارات از پیش شناخته شده ای را که در مورد درست بودن نرم افزار تازه تولید دارد, دور ریخته و بر برخورد عقایدی که بعد از برملا شدن خطاها بوجود می آید, غلبه کند.

چرا انجام این امر از اهمیت برخوردار است؟

به منظور یافتن بیشترین مقدار احتمالی خطا, آزمون ها باید به صورت نظام مند صورت گرفته و موارد آزمون با استفاده از فنون اصولی طراحی شوند.

مراحل کار چیست؟

نرم افزار از دو دیدگاه مختلف آزموده می شود:

۱)منطق درونی برنامه با استفاده از آزمون White Box و فنون آن, اجرا می شود.

۲)مقتضیات نرم افزاری با استفاده از فنون طراحی مورد آزمون Black Box آزموده می شوند. در هر دو مورد, هدف یافتن حداکثر خطا با حداقل تلاش و زمان است.

محصول کار چیست؟ مجموعه ای از موارد آزمون, که برای اجرای مقتضیات برونی و منطق درونی طراحی شده اند, طرح و ثبت شده است.

طراحی موارد آزمون

هر محصول مهندسی ساز را می توان به یکی از دو روش زیر امتحان نمود:

با آگاهی از کارکرد خاصی که این محصول برای آن تولیدشده, آزمون هایی را می توان انجام داد که هر کارکرد را از نظر عملی بودن کاملاً تشریح کند.

با آگاهی از کارهای صورت گرفته داخلی در هر محصول می توان آزمون هایی را انجام داد که از جور شدن همه کارها اطمینان یابیم, یعنی عملیات داخلی طبق مشخصات بوده و همه اجزای درونی به اندازه کافی به کار گرفته شده اند.

آزمون جعبه سیاه اشاره دارد به آزمونی که بر رابط نرم افزاری صورت می گیرد.

آزمون جعبه سفید درمورد بررسی دقیق جزییات رویه ای کارصورت می گیرد.

آزمون جعبه سفید

یک روش طراحی مورد آزمونی است که از ساختار کنترل طراحی رویه برای بدست آوردن موارد آزمون استفاده می کند.

با استفاده از روش های آزمون جعبه سفید مهندس نرم افزار می تواند موارد آزمونی را بدست آورد که:

۱)تضمین کنند که همه مسیرهای مستقل داخل یک پیمانه حداقل یکبار به کار گرفته شده اند.
۲)همه تصمیمات منطقی را در مورد طرفین درست و غلط آنها اجرا کند.
۳)همه لوپ ها را در سرحدات آنها و در داخل سرحدات عملیات آنها اجرا کند.
۴)تمام ساختارهای اطلاعاتی داخلی را برای تضمین اعتبارشان اجرا سازد.

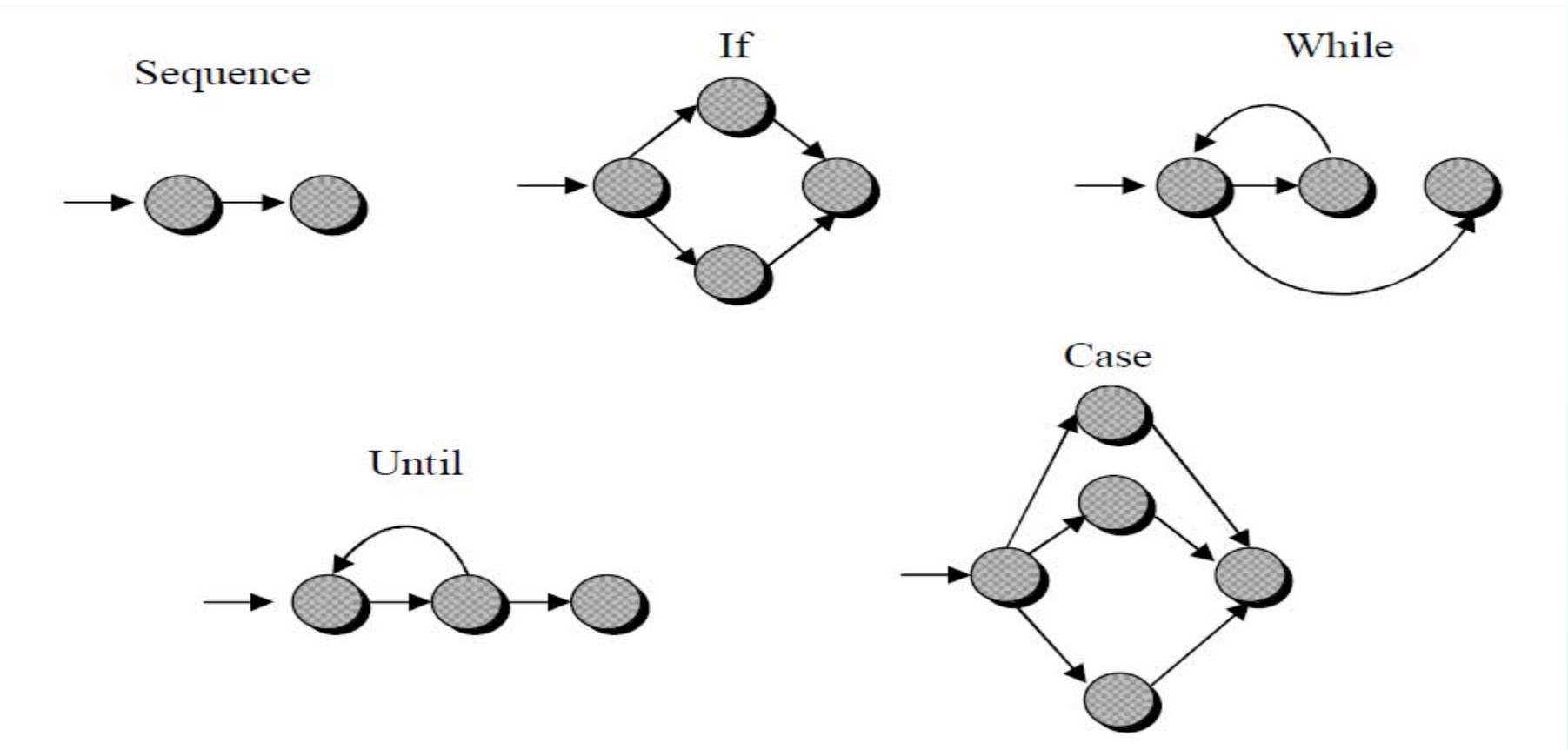
آزمون مسیر پایه

آزمون مسیر پایه یک تکنیک آزمون جعبه سفید است.

این روش, طراح مورد آزمونی را قادر می سازد تا یک ارزیابی پیچیده منطقی را از طرح رویه داشته و از این ارزیابی به عنوان راهنمایی برای تعریف مجموعه مقدماتی مسیرهای اجرایی استفاده کند.

گراف جریان

گراف جریان با استفاده از نشانه گذاری آمده در شکل زیر, جریان منطقی کنترل را مشخص می کند.



استخراج مسیرهای پایه

- مقدار محاسبه شده برای پیچیدگی چرخشی، تعداد مسیرهای مستقل را در مجموعة پایه برنامه مشخص می کند

۱. حد بالایی برای تعداد آزمایشاتی که باید برای اطمینان از اجرای حداقل یک بار هر یک از دستورات انجام شوند

۲. مسیر مستقل؛ هر مسیری در برنامه که حداقل یک مجموعة جدید از دستورات پردازشی یا شرط جدیدی را بیان نماید

- پس از استخراج مسیرهای پایه، داده آزمایش برای هر مسیر ایجاد می‌شود

نمونه از استخراج مسیرهای پایه

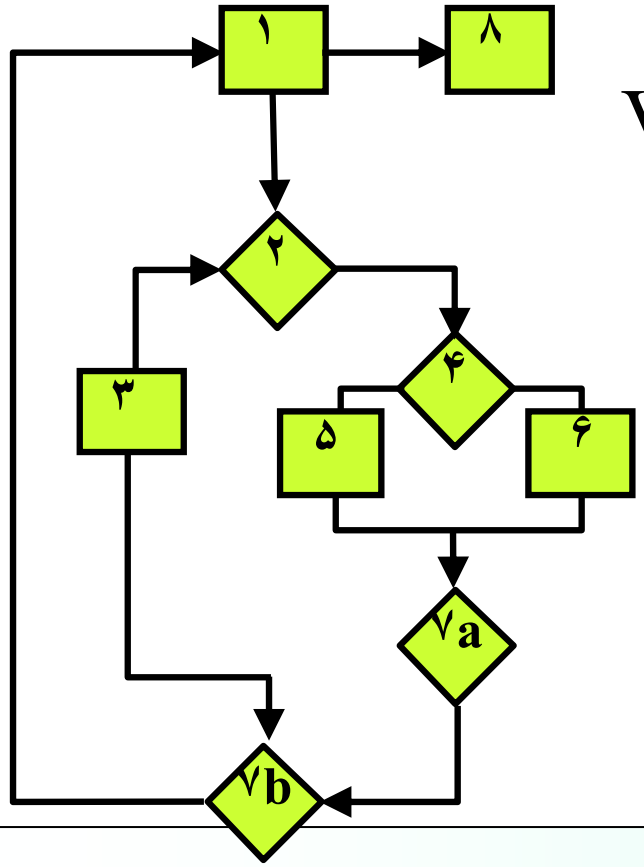
Example PDL

```
        procedure  sort
۱ :             do while records remain
                        read record ;
۲ :                     if  record field۱ = ۰
۳ :                             then  process record ;
                                      store in buffer ;
                                       increment counter ;
۴ :                           elseif  record field۲ = ۰
۵ :                                   then reset counter ;
۶ :                                   else  process  record ;
                                            store in file ;
۷a:                           endif
                        endif
۷b:             enddo
۸ :     end
```


Email: ComputerCollege_Fam@Yahoo.Com

نمونه از استخراج مسیرهای پایه (ادامه)

- مقدار پیچیدگی چرخشی



$$V(G) = ۱۱ - ۹ + ۲ = ۳ + ۱ = ۴$$

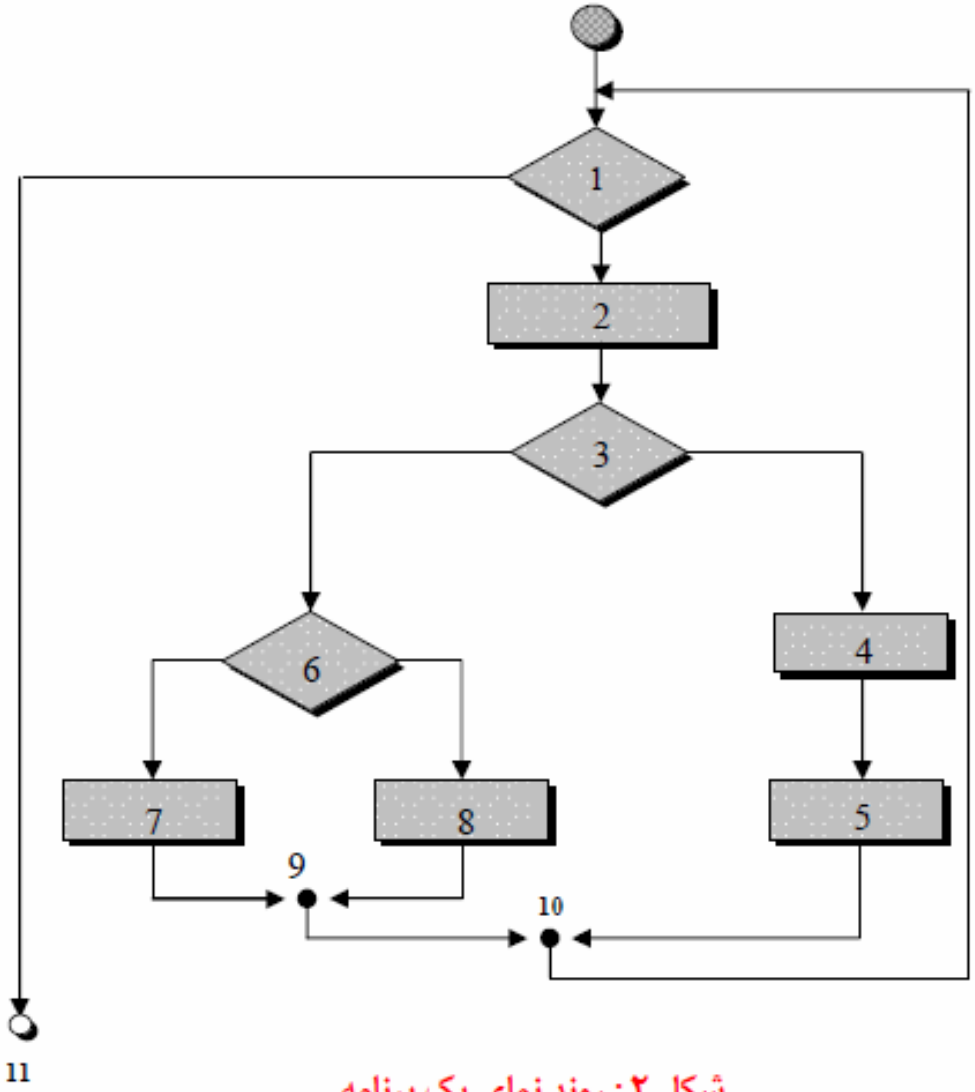
Path ۱: ۱,۸

Path ۲: ۱,۲,۳,۷b,۱,۸

Path ۳: ۱,۲,۴,۶,۷a,۷b,۱,۸

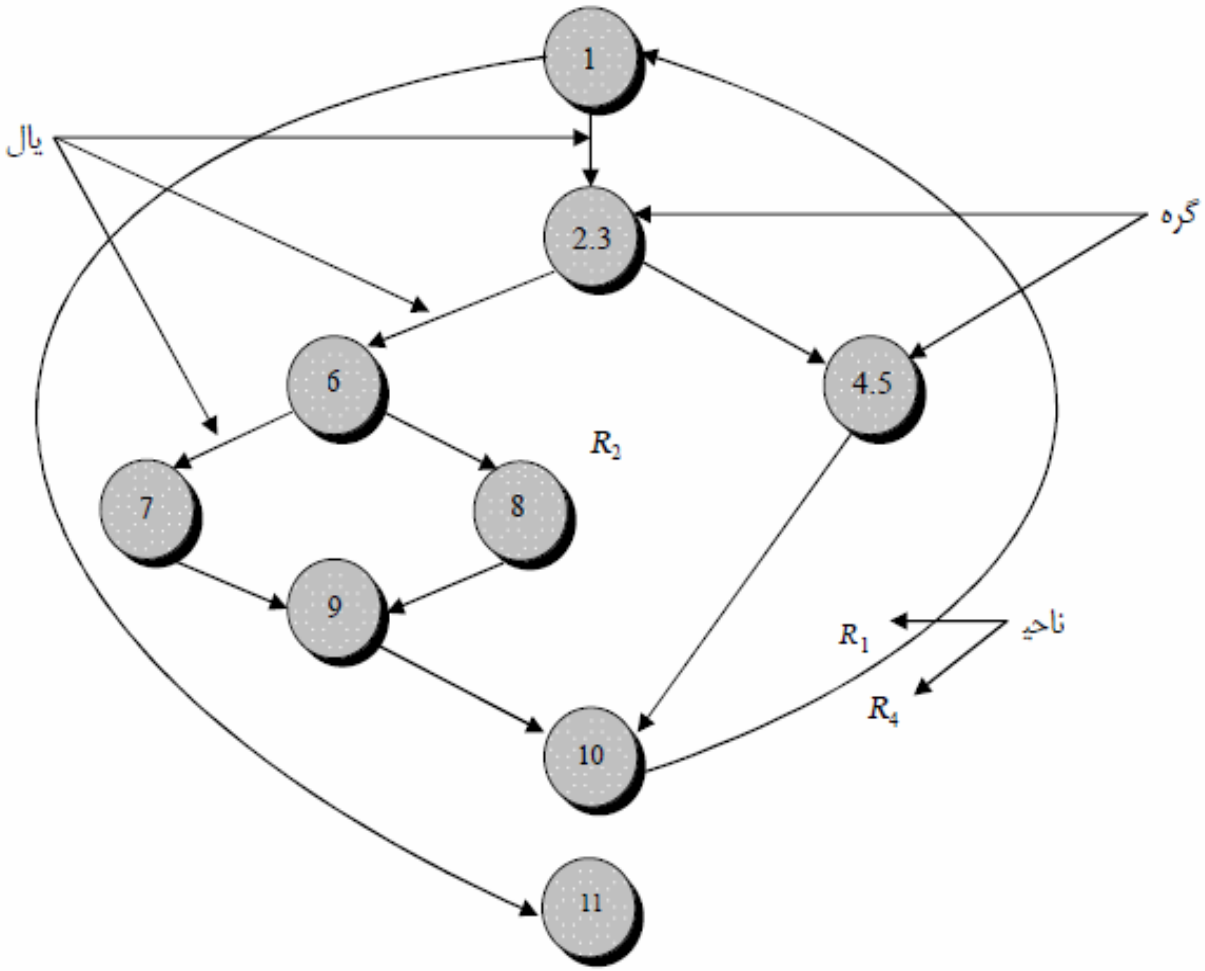
Path ۴: ۱,۲,۴,۵,۷a,۷b,۱,۸

نمونه از استخراج مسیرهای پایه



شکل ۲ : روند نمای یک برنامه

نمونه از استخراج مسیرهای پایه



شکل ۳ : گراف جریان مساله فوق

نمونه از استخراج مسیرهای پایه

پیچیدگی سیکلوماتیک به یکی از سه شکل زیر محاسبه می شود:

۱) تعداد مناطق (ناحیه های) نمودار جریان که با پیچیدگی سیکلوماتیک ارتباط دارند.

۲) پیچیدگی سیکلوماتیک, V(G) برای گراف جریان G بصورت زیر تعریف شده:

$$V(G) = E - N + ۲$$

که در آن E تعداد لبه های نمودار جریان و N تعداد گره ها می باشد.

۳) V(G) برای نموار جریان (G) بصورت زیر تعریف می شود:

$$V(G) = P + ۱$$

که در آن P تعداد گره های گزاره در نمودار جریان G است.

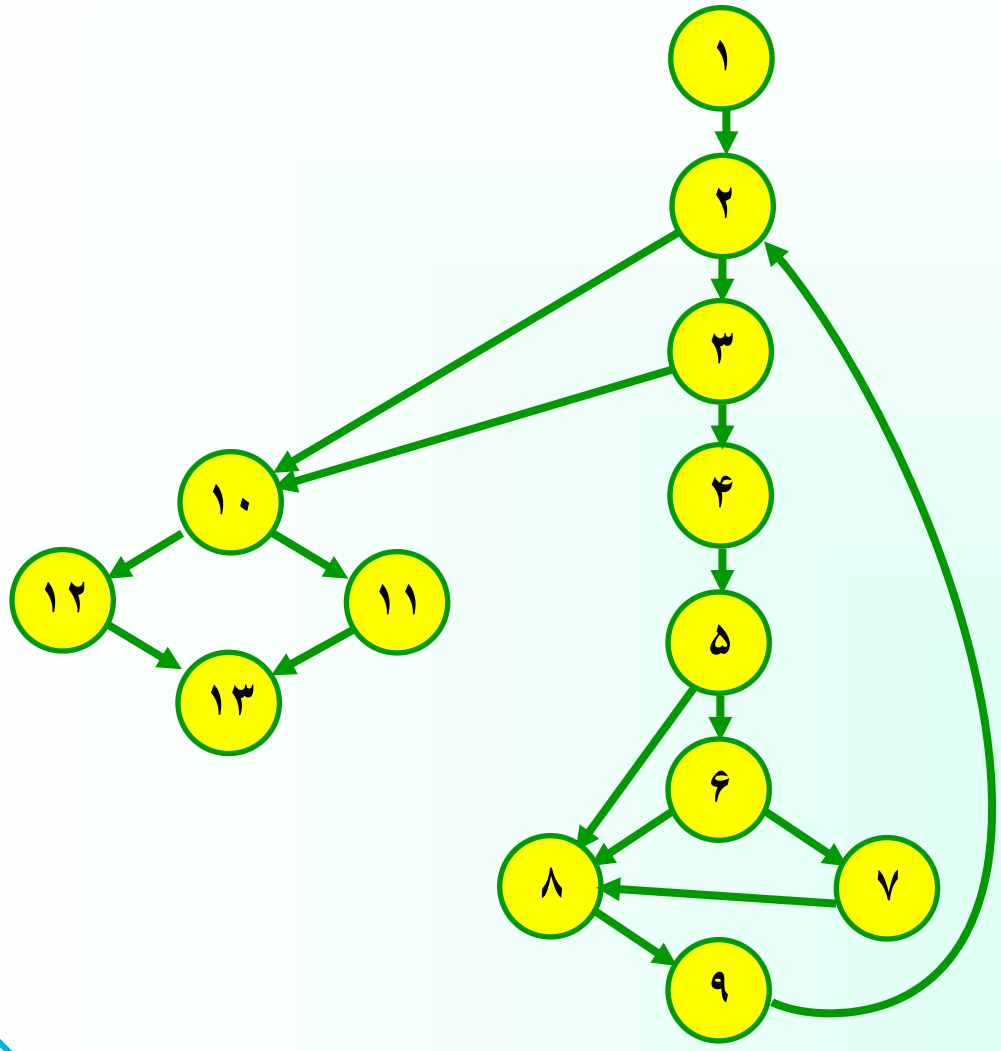
Email: ComputerCollege_Fam@Yahoo.Com

نمونه دیگر از استخراج مسیرهای پایه

```
PROCEDURE average;

    *   This procedure computes the average of 100 or fewer
        numbers that lie between bounding values; it also computes the
        sum and the total number valid.

    INTERFACE RETURNS average, total.input, total.valid;
    INTERFACE ACCEPTS value, minimum, maximum;

    TYPE value[1:100] IS SCALAR ARRAY;
    TYPE average, total.input, total.valid;
        minimum, maximum, sum IS SCALAR;
    TYPE i IS INTEGER;
    i = 1;
    total.input = total.valid = 0;
    sum = 0;
    DO WHILE value[i] <> -999 AND total.input < 100
        increment total.input by 1;
        IF value[i] > = minimum AND value[i] < = maximum
            THEN increment total.valid by 1;
                 sum = s sum + value[i]
            ELSE skip
        ENDIF
        increment i by 1;
    ENDDO
    IF total.valid > 0
        THEN average = sum / total.valid;
        ELSE average = -999;
    ENDIF
END average
```


Email: ComputerCollege_Fam@Yahoo.Com

نمونه دیگر از استخراج مسیرهای پایه (ادامه)



نمونه دیگر از استخراج مسیرهای پایه (ادامه)

V(G) = ۶ regions , V(G) = ۱۷ edges – ۱۳ nodes + ۲ = ۶ ,

V(G) = ۵ predicates nodes + ۱ = ۶

path 1:	1-2-10-11-13
path 2:	1-2-10-12-13
path 3:	1-2-3-10-11-13
path 4:	1-2-3-4-5-8-9-2-. . .
path 5:	1-2-3-4-5-6-8-9-2-. . .
path 6:	1-2-3-4-5-6-7-8-9-2-. . .

Email: ComputerCollege_Fam@Yahoo.Com

آزمون شرط (آزمون وضعیت)

یک روش طراحی مورد آزمون است که شرایط منطقی موجود در پیمانه برنامه را می آزماید.

یک وضعیت ساده, یک متغیر بولین یا عبارت رابطه ای است که احتمالاً با یک اپراتور NOT همراه است. عبارت ربطی شکل زیر را به خود می گیرد:

E۱ <Relational-Operator> E۲

که در آن E۱ و E۲ عبارات جبری و قسمت عملگر رابطه ای یکی از موارد زیر است: “>“ , “≥“ , “≠“ , “<“ , “≤“ . وضعیت مرکب متشکل از دو یا چند وضعیت ساده, اپراتورهای بولین و پرانتزهاست.

فرض می کنیم که اپراتورهای بولین که در وضعیت مرکب مجازند شامل (“|“) OR, (&) AND, (“¬“) NOT باشد. شرط بدون عبارات ربطی را عبارت بولین می نامند.

Email: ComputerCollege_Fam@Yahoo.Com

آزمون شرط (آزمون وضعیت)

بنابر این انواع خطاهای یک شرط شامل موارد زیر هستند:

- خطای عملگر بولین (نادرست/ از قلم افتاده/ اپراتورهای اضافی بولین)
- خطای متغیر بولین
- خطای پرانتز بولین
- خطای اپراتور رابطه ای
- خطای عبارت محاسباتی

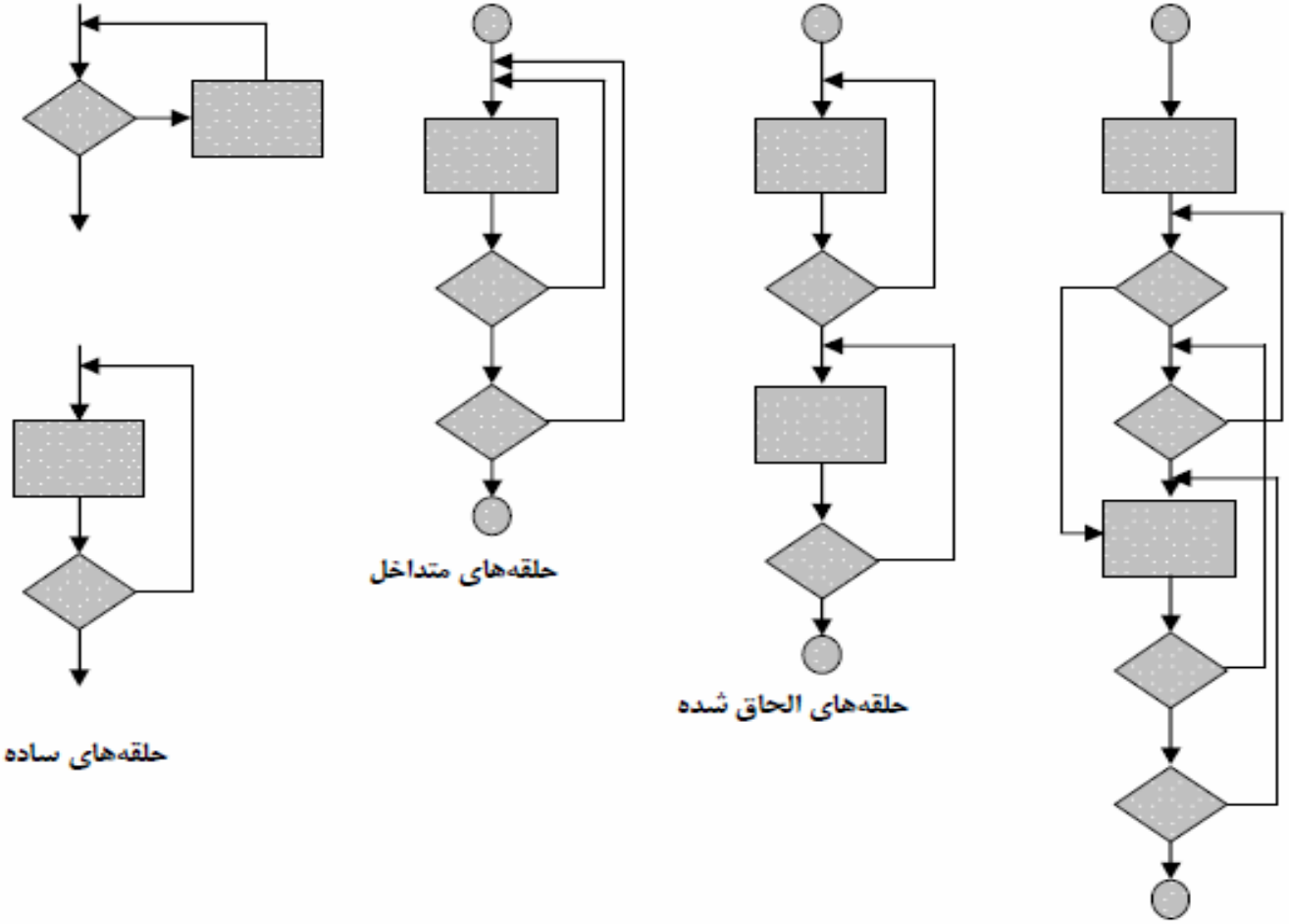
آزمون جریان داده

این روش مسیرهای آزمونی یک برنامه را طبق محل تعاریف و کاربرد متغیرها در برنامه، انتخاب می کند.

یک شیوه ساده آزمون جریان داده ها عبارتست از درخواست این که هر زنجیره تعریف-کاربرد (یا زنجیره DU) حداقل یکبار پوشش داده می شود. به این راهبرد، راهبر آزمون DU می گویند.

آزمون حلقه

آزمون حلقه, یک تکنیک آزمون جعبه سفید است که منحصراً روی اعتبار ساختمان های حلقه متمرکز می شود. می توان چهار کلاس از حلقه ها را تعیین نمود:



آزمون جعبه سیاه(آزمون رفتاری)

آزمون جعبه سیاه، مهندس نرم افزار را قادر می سازد مجموعه ای از وضعیت های ورودی را بدست آورد که به طور کامل همه نیازمندی های کارکردی را برای برنامه اجرا خواهند کرد.

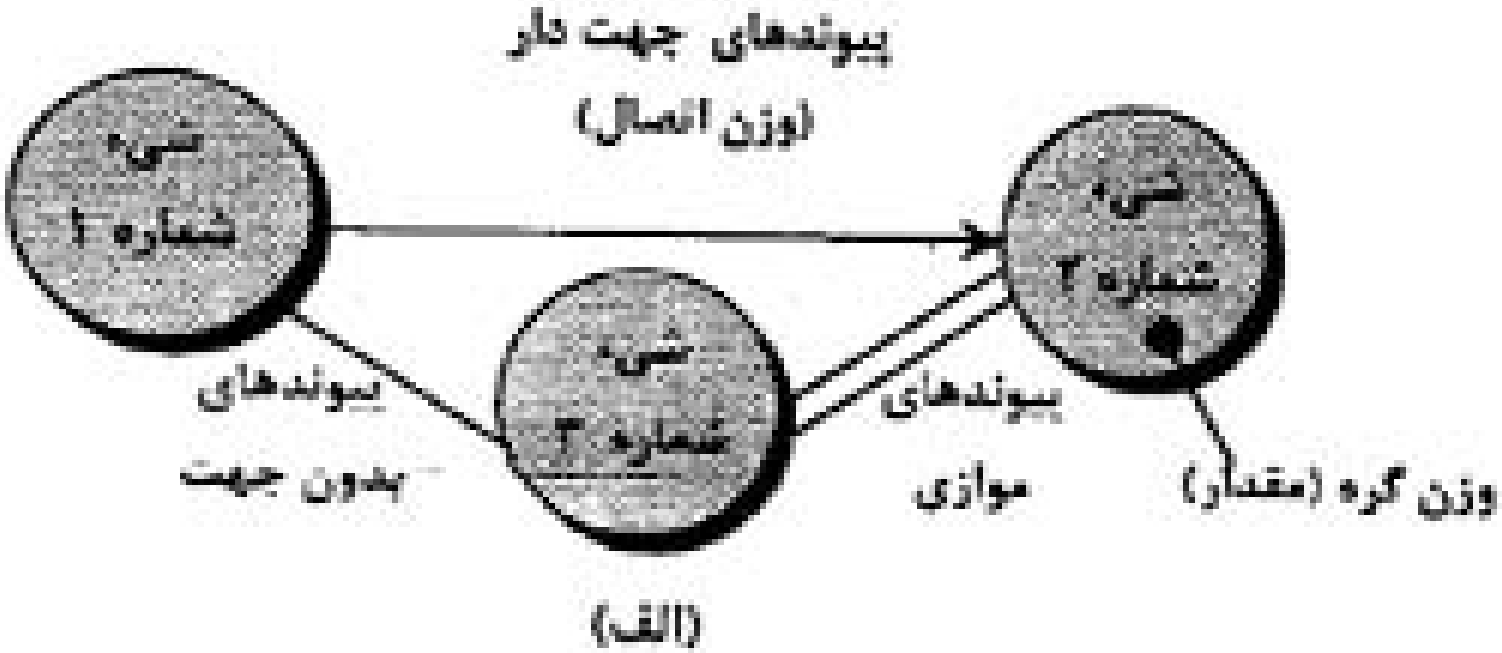
آزمون جعبه سیاه سعی دارد خطاهایی را در گروه های زیر پیدا کند:

- کارکردهای(توابع) غلط یا حذف شده
- خطاهای رابط ها(واسط ها)
- خطاهای ساختاهای داده ای یا دسترسی به پایگاه های داده ای بیرونی
- خطاهای رفتاری یا عملکردی
- خطاهای شروع و خاتمه.

شیوه های آزمون مبتنی بر گراف

- اولین مرحله در آزمون جعبه سیاه عبارتست از شناخت اشیایی که در نرم افزار مدل سازی شده و ارتباطاتی که این اشیاء را به هم مرتبط می کند.

- آزمون نرم افزار با ایجاد نموداری از اشیای مهم و ارتباطاتشان آغاز شده و سپس یک سری آزمون تعبیه می شود که نمودار را تحت پوشش قرار می دهد به طوری که هر شی و رابطه آن به اجرا درآمده و خطاها مشخص می شوند.


پیوندهای جهت دار
(وزن اتصال)
شیء
شماره ۱
شیء
شماره ۲
شیء
شماره ۳
پیوندهای
بدون جهت
پیوندهای
موازی
وزن گره (مقدار)
(الف)

تجزیه هم ارزی

- تقسیم و تجزیه هم ارزی یک روش از آزمون جعبه سیاه است که قلمرو ورودی برنامه را در گروه های مختلفی از داده ها تقسیم می کند که از آنها موارد آزمون بدست می آیند.

- طراحی مورد آزمون برای تقسیم هم ارزی بر اساس ارزیابی گروه های هم ارز برای یک شرط ورودی است.

تحلیل مقادیر ارزی

- بنا به دلایلی که کاملاً روشن نیست, بیشتر خطاها در سرحدات دامنه ورودی رخ می دهند تا در مرکز. به همین دلایل است که تحلیل مقدار سرحد(BVA-Boundary Value Analysis) به عنوان یک تکنیک آزمونی اراده شده است.
- به جای اینکه منحصراً بر شرایط ورودی متمرکز شویم, BVA موارد آزمونی بدست می آورد که از دامنه خروجی نیز هستند.

- رهنمودهای مربوط به **BVA**:

۱. اگر یک شرط ورودی، طیفی را مشخص کند که بوسیله مقادیر a و b محدود شده اند, موارد آزمونی باید طراحی شوند که به ترتیب بالا و پایین مقادیر a و b باشند.

۲. اگر یک شرط ورودی چند مقدار را مشخص کند, موارد آزمونی باید ارائه شوند که حداقل و حداکثر ارقام را آزمایش نمایند. مقادیری که درست بالا و پایین مقادیر حداقل و حداکثر هستند نیز, آزمون می شوند.

۳. رهنمودهای ۱ و ۲ شرایط ورودی را در شرایط خروجی بکار گیرید.

۴. اگر ساختار داده ای برنامه داخلی سرحدات را مشخص نموده باشد، مطمئن باشد که یک مورد آزمونی برای آزمون ساختار داده ای در سرحد آن طراحی می شود.

مقایسه خروجی ها

با اجراهای متعددی از یک مشخصه یکسان, موارد آزمونی با استفاده از فنون دیگر جعبه سیاه طراحی می شوند که به عنوان ورودی برای هر نسخه از نرم افزار هستند.

اگر خروجی هر نسخه یکسان باشد, فرض می شود که همه اجراها درست هستند.

اگر خروجی مختلف بود, هر برنامه مورد بررسی قرار می گیرد تا معلوم شود آیا نقضی در یک یا چند نسخه وجود دارد که مسئول این اختلاف است یا خیر.

در اکثر موارد, مقایسه خروجی ها را می توان با ابزار خودکار انجام داد.

آزمون رابط های گرافیکی کاربر (GUI): به خاطر اجزای قابل استفاده مجدد که به عنوان بخشی از محیط های تولیدی GUIمهیا شده اند، ایجاد رابط کاربر زمان کمتری گرفته و دقیق تر نیز هست.
اما در عین حال پیچیدگی GUI نیز افزایش یافته که منجر به مشکلات بیشتری در طراحی و اجرای موارد آزمون شده است.
از آن جا که بیشتر GUIهای مدرن ظاهر و حالات یکسانی دارند، یک سری آزمون های استاندارد ارائه شده اند. گراف های مدل سازی حالت محدود, ممکن است برای بدست آوردن یک سری آزمون ها استفاده شوند که اشیای برنامه ای و داده ای خاصی را که مربوط به GUI هستند, مورد نظر قرار می دهد.

آزمون معماری های خادم /مخدوم(C/S)

ماهیت توزیع شده ی محیط های C/S، موضوعات عملکردی مربوط به پردازش تراکنش ها، حضور بالقوه یک سری پایگاه های سخت افزاری مختلف، پیچیدگی های ارتباط شبکه ای, نیاز خدامات به کاربران متعدد از یک پایگاه داده ای مرکزی و نیازمندی های هماهنگی که برروی خادم اعمال شده, همگی کار آزمون معماریC/S و نرم افزاری که در آن قرار گرفته را نسبت به برنامه های متکی (Standalone) مشکل تر می سازند.

آزمون مستندات و تسهیلات کمکی(راهنما)

این آزمون می تواند در دو مرحله صورت گیرد:

- اولین مرحله، بازنگری و بازرسی است که اسناد را از نظر ویرایشی بازبینی می کند.
- دومین مرحله، آزمون زنده است که از مستندات در ارتباط با استفاده از برنامه واقعی استفاده می کند.

می توان یک راهبرد چهار مرحله ای را پیشنهاد نمود:

۱) آزمون وظیفه: اولین مرحله در آزمون نرم افزار بدون وقفه عبارتست از آزمون هر وظیفه بطور مستقل, یعنی آزمون های جعبه سفید و سیاه طراحی شده و برای همان وظیفه به اجرا درمی آیند.

۲) آزمون رفتاری: با استفاده از مدل های سیستمی که توسط ابزارهای CASE ایجاد شده اند، می توان کارکرد سیستم بدون وقفه را شبیه سازی نموده و عملکرد آن را در اثر حوادث خارجی بررسی نمود.

۳) آزمون بين وظايف: كارهاي ناهمگام كه از نظر برقراري ارتباط با يكديگر شناسايي شده اند با پردازش و ارزيابي مختلف داده ها صورت مي گيرند كه نهايتاً منجر به اين امر مي شود كه آيا خطاهاي ناهمگام بين وظيفه اي رخ مي دهند يا خير؟

۴) آزمون سيستم: نرم افزار و سخت ابزار با هم تلفيق مي شوند و يک سري آزمون هاي سيستم روي آنها صورت مي گيرد تا خطاهاي موجود در رابط نرم افزار/ سخت افزاري مشخص شود.

- آزمایش آلفا در سایت توسعه دهنده (توسط مشتری) انجام می شود.

نرم افزار با تنظیمات معمول استفاده می شود و توسعه دهنده بر آن نظارت دارد و خطاها را ثبت می نماید. آزمایش های آلفا در **محیطی کنترل شده** انجام می شوند.

- آزمایش بتا توسط کاربر نهایی نرم افزار انجام می شود

برخلاف آزمایش آلفا، توسعه دهنده عموماً حضور ندارد.

آزمایش بتا، بکارگیری زندة نرم افزار در محیطی است که توسط توسعه دهنده قابل کنترل نیست .

مشتری تمام مشکلات را (واقعی یا خیالی) که در طول آزمایش بتا شناسایی می شوند ثبت می کند و این گزارشات را به توسعه دهنده در بازه های زمانی منظم تحویل می دهد. در نتیجة مشکلات گزارش شده در ضمن آزمایش ها ی بتا، مهندسین نرم افزار اصلاحات را انجام می دهند و برای انتشار محصول نر مافزار به مشتری آماده می شوند.

معیارهای اندازه‌گیری و سنجش نرم‌افزار

درس مهندسی نرم‌افزار ۲

مطالب این فصل

- اندازه‌گیری
- معیار، شاخص و اندازه
- نقش معیارهای سنجش
- معیارهای فرآیند
- معیارهای پروژه
- ارزش تابعی
- معیارهای کیفیت نرم‌افزار
- فرآیند جمع‌آوری معیارها

کیفیت نرم افزار

- چه چیزی تفاوت بین دو نرم‌افزاری که وظیفه و عملکرد یکسانی را انجام می دهند مشخص می کند؟
- چرا یک نرم افزار در عمل از دیگری موفق تر است؟

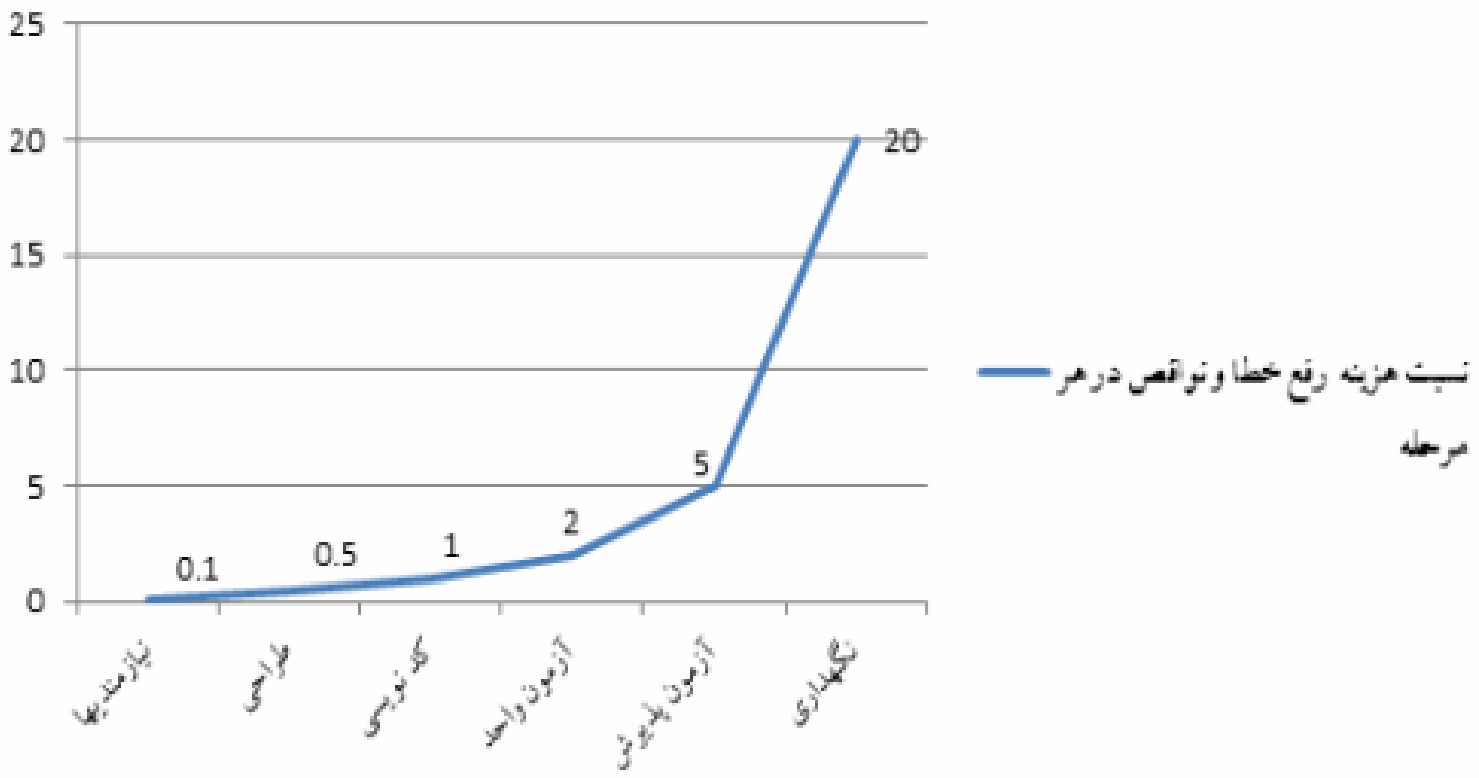
- تفاوت بین دو نرم افزار را توانایی نرم‌افزارها در برآورده ساختن نیازهای مورد انتظار تعیین می‌کند.
- نرم‌افزارها دارای دو نوع نیاز هستند، نیازهای عملیاتی و غیرعملیاتی.

- نیازهای عملیاتی، توانایی‌های سیستم در انجام وظایف مختلف را توصیف می‌کنند.
- و نیازمندی‌های غیرعملیاتی سیستم، ویژگی‌های کیفی سیستم می‌باشند نیازهایی هستند که بطور مشخص با عملیات سیستم در ارتباط نیستند

تعریف کیفیت نرم‌افزار

کیفیت نرم افزار شامل همه ویژگی‌ها و مشخصات مهم یک محصول است که نیازمندی‌های صریح و ضمنی را برآورده سازد. به بیان دیگر، کیفیت همه ویژگی‌های یک محصول نرم افزاری است که نتیجه آن توانایی برآورده کردن نیازهای خواسته شده است.

نسبت هزینه رفع خطا و نواقص در هر مرحله



کیفیت داخلی

خصوصیاتی است که متناسب با مشخصات ثابت و ایستای کد برنامه تعریف شده و توسط برنامه نویس سیستم اندازه‌گیری می‌شود.

کیفیت خارجی

خصوصیاتی است که متناسب با مشخصات پویای کد برنامه در زمان اجرا تعریف شده و توسط کاربر سیستم اندازه‌گیری می‌شود.

کیفیت استفاده

خصوصیاتی است که متناسب با دیدگاه کاربر از کیفیت سیستم در زمان استفاده از آن است. کیفیت استفاده برحسب اینکه نرم‌افزار تا چه اندازه پاسخگوی نیازهای کاربر در محیط اجرای سیستم باشد، اندازه‌گیری می‌شود..

اندازه‌گیری پایه مهندسی است و مهندسی نرم‌افزار نیز از این قاعده مستثنی نیست Lord Kelvin می گوید:

وقتی که بتوانید آن چه را که از آن سخن می‌گویید اندازه‌گیری کرده و آن را در قالب اعداد بیان کنید، چیزی در مورد آن می‌دانید، ولی وقتی نتوانید آن را اندازه‌گیری کنید و نتوانید آن را در قالب اعداد بیان کنید، آگاهی شما ضعیف بوده و رضایت‌بخش نیست؛ در واقع ممکن است این امر نشاندهنده سرآغاز آگاهی شما از موضوع باشد، ولی در ذهن شما نقصان وجود دارد، تا به سطح دانش مورد نظر برسید.

در طول سال‌های گذشته جامعه مهندسی نرم‌افزار گفته Kelvin را مورد بحث قرار داده و شروع به پذیرش آن نمودند.

Measure ،Metric و indicator

- هر چند ترم های measure (اندازه) و metric (معیار) غالبا به جای یکدیگر به کار می روند باید توجه داشت که با یکدیگر تفاوت دارند.

- **Measure** یک شاخص کمی از اندازه، مقدار، ابعاد و ظرفیت یک صفت از یک محصول یا فرآنید را تعیین می کند.

- **Metric** یک اندازه کمی از درجه ای به یک فرآیند، مولفه، سیستم برای یک صفت مشخص می باشد.

- **Indicator** نوعی نشانه و شاخص برای شناخت روند حرکت تیم و پروژه می باشد.

مثال:

- ۵۰۰ نفر ساعت کار کرده است به ازای هر نفرساعت ٦ خطا(متریک) : تیم ۱
- ۱۰۰۰ نفر ساعت کارکرده است به ازای هر نفرساعت ۸ خطا(متریک) : تیم ۲
- نتیجه (شاخص) :تیم ۱ بهتر بوده

چهار دلیل برای اندازه‌گیری:

توصیف: برای درک فرآیند، محصول، منابع و محیط

ارزیابی: برای تعیین وضعیت مطابق با برنامه

پیش‌بینی: ابتدا پیش‌بینی می‌کنیم و سپس برنامه‌ریزی می‌کنیم

بهبود: کیفیت و کارایی فرآیند

فرآیند
معیارهای فرآیند
معیارهای محصول
اندازه‌گیری
معیارهای پروژه
محصول
چگونگی اندازه‌گیری
• اندازه کد
• وظایف

نقش معیارهای سنجش

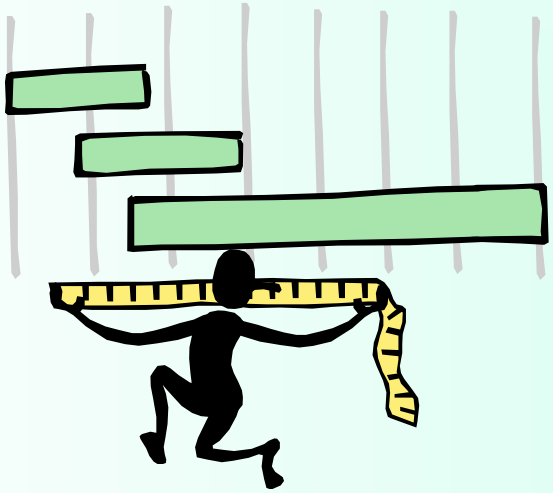
اغلب معیارهای سنجش هم در حوزه پروژه و هم در فرآیند نرم‌افزار بکار می‌روند
معیارهای سنجش فرآیند در طول پروژه‌های مختلف و در طی یک دوره زمانی طولانی گردآوری می‌شوند

هدف از تهیه معیارهای سنجش فراهم آوردن شاخص‌هایی است که منجر به بهبود فرآیند نرم‌افزار در بلند مدت شود

شاخص‌های فرآیند

به سازمان توسعه‌دهنده نرم‌افزار کمک می‌کنند تا درجه تاثیر فرآیند جاری سنجیده شود

به مدیران و توسعه‌دهندگان کمک می‌کند تا بفهمند چه روشی اجرایی است و چه روشی اجرایی نیست.



نقش معیارهای سنجش

- **شاخص‌های پروژه** به مدیر پروژه کمک می‌کنند تا:

۱. وضعیت یک پروژه در حال جریان را ارزیابی نماید

۲. ریسک‌های بالقوه را پیگیری کند

۳. زمینه‌های مشکل را پیش از آنکه «حاد» شوند، مشخص نماید

۴. جریان کار یا فعالیت‌ها را تعدیل نماید

۵. قابلیت گروه پروژه را در مورد کنترل کیفیت محصولات کاری مهندسی نرم افزار مورد ارزیابی قرار دهد

تنها راه بهبود هر فرآیند:

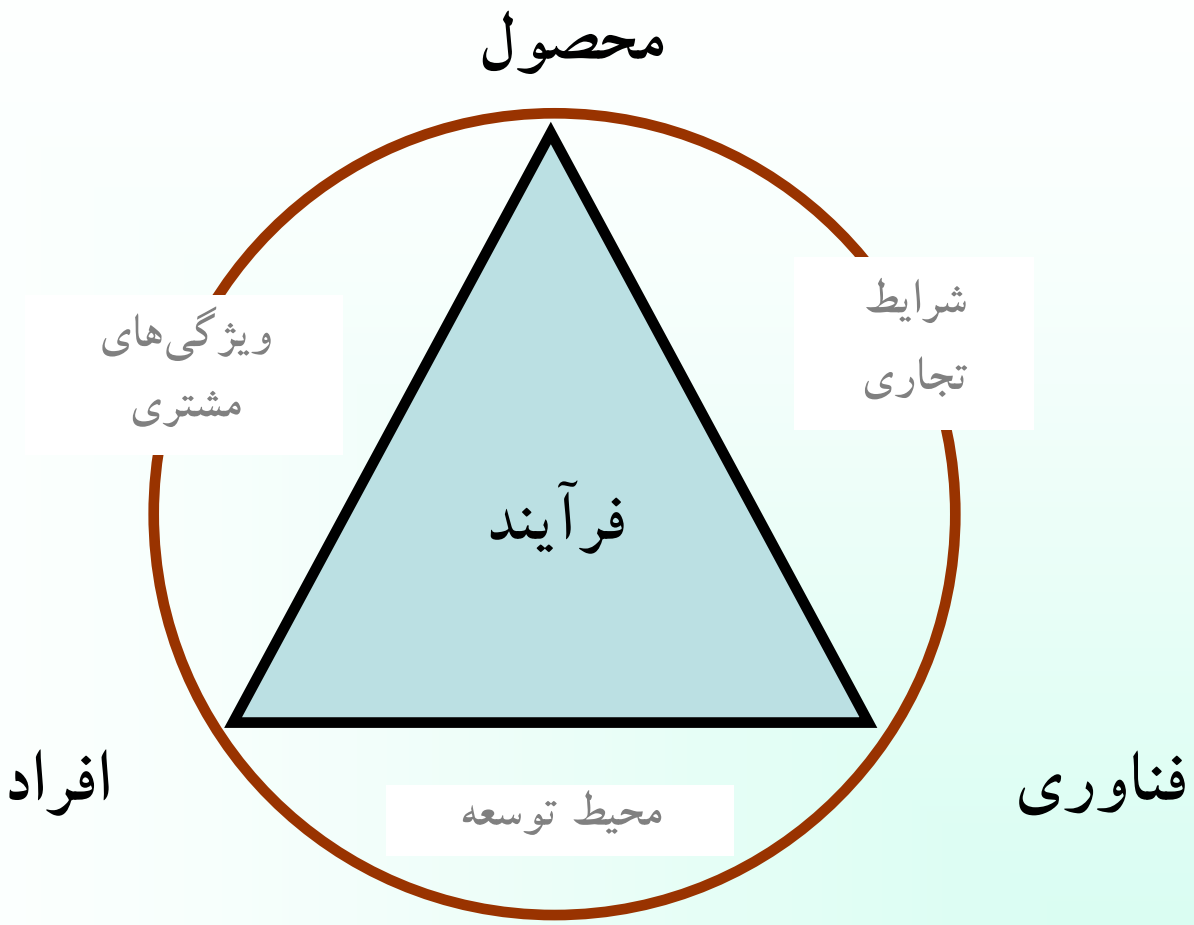
اندازه‌گیری ویژگی‌های خاص فرآیند

تهیه مجموعه‌ای از معیارهای سنجش معنی‌دار مبتنی بر این صفات

استفاده از معیارها جهت استخراج شاخص‌هایی که منجر به استقرار یک استراتژی برای بهبود فرآیند شود

در واقع، فرآیند تنها یکی از «عوامل قابل کنترل در بهبود کیفیت نرم‌افزار و کارایی سازمانی است»

معیارهای فرآیند



دیگر معیارهای بهبود کیفیت نرم‌افزار و کارایی سازمانی

معیارهای فرآیند

تاثیرگذاری فرآیند نرم‌افزار غیرمستقیم اندازه‌گیری می‌شود

مجموعه‌ای از معیارها، براساس خروجی‌هایی که می‌توان از فرآیند بدست آورد، ایجاد می‌شوند

نمونه‌ای از خروجی‌ها

۱. اندازه‌گیری تعداد خطاها قبل از تحویل نرم‌افزار به مشتری
۲. نقص‌های گزارش شده توسط کاربر
۳. محصولات کاری تحویل شده (بهره‌وری)
۴. نیروی انسانی بکار گرفته شده
۵. زمان صرف شده
۶. مطابقت زمانبندی

معیارهای فرآیند می‌توانند با اندازه‌گیری خصوصیات برخی فعالیت‌های خاص مهندسی نرم‌افزار حاصل شوند

معیارهای فرآیند

- دو نوع استفاده از داده‌های مختلف فرآیند (Grady)
  - معیارهای خصوصی (Private Metrics) برای یافتن شاخص‌های افزایش کارایی فرد
    - نرخ خرابی (توسط فرد)
    - نرخ خرابی (توسط واحد)
    - خطاهای پیدا شده در طول توسعه نرم‌افزار
  - معیارهای عمومی (Public Metrics) برای یافتن شاخص‌های افزایش کارایی گروه
    - نرخ های عیب سطح پروژه
    - نیروی کاری صرف شده
    - تقویم زمانی
    - داده های مربوطه

معیارهای فرآیند

- تقسیم معیارها به صورت خصوصی و عمومی مشکل ایجاد می‌کند
- Grady رسومی را برای استفاده از معیارهای سنجش نرم‌افزار پیشنهاد می‌کند که هم برای مدیران و هم توسعه‌دهندگان مناسب است
  - در موقع تفسیر داده‌های سنجش توجه به حساسیت متداول و سازمانی داشته باشید
  - هرگز از معیارهای سنجش برای تهدید افراد یا گروه‌ها استفاده نکنید
  - هرگز اجازه ندهید یکی از معیارها جایگزین تمامی معیارهای دیگر گردد
  - و ...

- مدلی برای اندازه‌گیری پروژه
  - ورودی‌ها: اندازه‌گیری منابع مورد نیاز برای انجام کار(۱۰ نفر برنامه نویس ، ۳ تا کامپیوتر)
  - خروجی‌ها: اندازه‌گیری تحویل دادنی‌ها یا محصولات کاری ایجاد شده در طول فرآیند مهندسی(۵ تا فرم، ۲ تاجدول)
  - نتایج: اندازه‌گیری شاخص‌های موثر بر تحویل دادنی‌ها(۸خطا در قسمت نمایش)

معیارهای سنجش پروژه

- در مقابل معیارهای سنجش فرآیند که برای مقاصد راهبردی بکار می‌روند، معیارهای سنجش پروژه فنی هستند
- برای تطبیق جریان کار پروژه و فعالیت های فنی بکار می‌روند
- کاربرد معیارهای سنجش پروژه
  - کاهش زمان توسعه با انجام تطبیق‌ها لازم برای کاهش تاخیرها و کاهش مشکلات و خطرات
  - ارزیابی کیفیت محصول براساس روند جاری فعالیت‌ها و تغییر فعالیت‌ها برای افزایش کیفیت

- در دنیای واقعی اندازه‌گیری به دو دسته عمده تقسیم می‌شود
  - اندازه‌گیری مستقیم
    - برای فرآیند دربرگیرنده سنجش هزینه و کار است
    - برای محصول
      - میزان خطوط برنامه
      - زمان اجرا
      - میزان حافظه
      - خطای گزارش شده در زمان معین
  - اندازه‌گیری غیرمستقیم
    - کیفیت نرم‌افزار
      - کارایی
      - قابلیت اصلاح و ...

نرمال‌سازی معیارها

در صورتی‌که معیارها نرمال شود، می‌توان معیارهای نرم‌افزاری ایجاد نمود که قابل مقایسه برای سازمان‌های متفاوت هستند

دو روش نرمال‌سازی:

- Size-Oriented
- Function-Oriented

معیارهای سنجش مبتنی بر اندازه

با نرمال‌سازی اندازه‌های مقداری کیفیت و یا بهره وری با استفاده از «اندازه» نرم‌افزار تولید شده، حاصل می‌شوند

- **متریک های مبتنی بر اندازه**

متریک های ساده مبتنی بر اندازه برای هر پروژه

۱. خطاها در هر هزار خط برنامه (KLOC)
۲. تعداد عیب ها در هر هزار خط برنامه (KLOC)
۳. هزینه هر خط برنامه (LOC) (بر حسب دلار یا واحد پول)
۴. تعداد صفحات مستندات به ازای در هر هزار خط برنامه (KLOC)
۵. تعداد خطاها به ازای هر نفر – ماه
۶. تعداد خطوط برنامه (LOC) به ازای هر نفر – ماه
۷. هزینه هر صفحه از مستندات (بر حسب دلار یا واحد پول)

اعتقاد بر این است که این معیارها، معیارهای وابسته به زبان برنامه‌نویسی هستند

معیارهای سنجش مبتنی بر اندازه (ادامه)

Project	LOC	Effort	$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
•	•	•	•	•	•		
•	•	•	•	•	•		
•	•	•	•	•	•		

معیارهای سنجش مبتنی بر عملکرد

- وظیفه‌مندی ارائه شده توسط نرم‌افزار به‌عنوان مقدار نرمال‌سازی بکار می‌رود
- وظیفه‌مندی را نمی‌توان به صورت مستقیم ارزیابی نمود
  - باید به صورت غیرمستقیم از معیارهای مستقیم استخراج شود
- این روش اولین بار توسط Albrecht با معرفی «ارزش تابعی» (function point) پیشنهاد شد

ارزش تابعی

- ارزش تابعی با استفاده از یک رابطه تجربی مبتنی بر معیارهای قابل اندازه‌گیری (مستقیم) دامنه اطلاعات نرم‌افزار و ارزیابی‌های مربوط به پیچیدگی نرم‌افزار حاصل می‌شود
- معیارهای قابل اندازه‌گیری
  - تعداد ورودی کاربران (Number of user inputs)
  - تعداد خروجی‌های کاربر (Number of user outputs)
  - تعداد درخواست‌های کاربران (Number of user inquiries)
  - تعداد فایل‌ها (Number of files)
  - تعداد واسط‌های خارجی (Number of external interfaces)

- تعداد ورودی کاربران
  - هر ورودی کاربر که داده‌هایی با کاربرد متمایز را برای نرم‌افزار فراهم می‌آورد
  - ورودی با درخواست تفاوت دارد
- تعداد خروجی های کاربر
  - هر خروجی کاربر که اطلاعاتی کاربردی را برای کاربر فراهم می‌کند
  - گزارشات، خروجی صفحه نمایش، پیام‌های خطا و ... به‌عنوان خروجی محسوب می‌شوند
  - اقلام داده‌ای مجزا که در یک گزارش قرار دارند، مجزا محاسبه نمی‌شوند

- تعداد درخواست‌های کاربران
  - درخواست به‌عنوان ورودی لحظه‌ای (on-line) تعریف می‌شود که حاصل آن تولید پاسخ(هایی) توسط نرم‌افزار به شکل خروجی لحظه‌ای است
  - هر درخواست، مجزا شمارش می‌شود
- تعداد فایل‌ها
  - هر فایل اصلی منطقی شمارش می‌شود
  - فایل منطقی یک گروه‌بندی منطقی از داده‌هاست که می‌تواند بخشی از یک پایگاه داده بزرگ یا یک فایل باشد

ارزش تابعی (ادامه)

- تعداد واسط‌های خارجی
  - تمام واسط‌های قابل خواندن توسط دستگاه (نظیر فایل‌های داده‌ای روی ابزار ذخیره‌سازی) که برای ارسال اطلاعات به یک سیستم دیگر مورد استفاده هستند، شمارش می‌شوند
- پس از شمارش این پارامترها، مقداری به‌عنوان پیچیدگی به هر یک منتسب می‌گردد

محاسبه ارزش تابعی

معیارهای اندازه‌گیری	تعداد		فاکتورهای وزن‌دهی از ساده تا پیچیده				
تعداد ورودی‌های کاربران		X	۳	۴	۶	=	
تعداد خروجی‌های کاربر		X	۴	۵	۷	=	
تعداد پرس و جوهای کاربران		X	۳	۴	۶	=	
تعداد فایل‌ها		X	۷	۱۰	۱۵	=	
تعداد واسط‌های خارجی		X	۵	۷	۱۰	=	
مجموع معیارها							

Email: ComputerCollege_Fam@Yahoo.Com

محاسبه ارزش تابعی

هر یک از پارامترهای ورودی را بشمارید

به هر شمارش یک مقدار وزنی نسبت دهید و «مجموع معیارها» را محاسبه کنید

مقادیر فاکتورهای خارجی **Fi** را محاسبه کنید

ارزش تابعی برابر است با

$$FP = C \times [۰.۶۵ + ۰.۰۱ \times \sum_{i=۱}^{۱۴} (F_i)]$$

Email: ComputerCollege_Fam@Yahoo.Com

محاسبه ارزش تابعی

مقادیر Fiها با پاسخ (مقدار ۰- ۵) به سوالات ذیل به دست می‌آید

۱- آیا سیستم نیاز به داشتن نسخه پشتیبان و بازیابی دارد؟

۲- آیا ارتباطات داده‌ای مورد نیاز است؟

۳- آیا توابع پردازشی توزیع شده وجود دارد؟

۴- آیا کارایی عامل حساسی است؟

۵- آیا سیستم در محیط عملیاتی موجود و دشوار اجرا خواهد شد؟

۶- آیا سیستم نیاز به داده ورودی لحظه‌ای دارد؟

۷- آیا داده ورودی لحظه‌ای نیاز دارد تا تراکنش ورودی بر روی صفحات چندگانه یا توسط عملیات چندگانه ساخته شود؟

مقادیر (ادامه)

۸- آیا فایل اصلی لحظه‌ای بهنگام می‌شود؟

۹- آیا ورودی‌ها، خروجی‌ها، فایل‌ها و درخواست‌ها پیچیده هستند؟

۱۰ - آیا پردازش داخلی پیچیده است؟

۱۱ - آیا کد بگونه‌ای طراحی می‌شود که قابل استفاده مجدد باشد؟

۱۲ - آیا تغییرات و نصب در طراحی در نظر گرفته شده است؟

۱۳ - آیا سیستم طوری طراحی شده است که بتوان آن را چند بار در سازمان‌های متفاوت نصب کرد؟

۱۴ - آیا نرم افزار طوری طراحی شده است که امکان تغییرات را بدهد و کاربر به سادگی از آن استفاده نماید؟

نرمال‌سازی ارزش تابعی

- پس از محاسبه مقدار ارزش تابعی، همانند روش LOC نرمال می‌شود:

۱. نسبت خطاها بر FP

۲. نسبت خرابی بر FP

۳. نسبت هزینه بر FP

۴. نسبت تعداد صفحات مستندات بر FP

۵. نسبت FP بر نفر - ماه

دلیل انتخاب ارزش تابعی

- عدم وابستگی به زبان برنامه‌نویسی
- از معیارهای ساده استفاده می‌کند که می‌توانند قبل از انجام فرآیند توسعه نرم‌افزار تعیین شوند
- پیاده‌سازی‌های ابتکاری که سبب کاهش LOC می‌شوند منجر بر جریمه نمی‌شوند
- ارزیابی میزان تاثیر مولفه‌های قابل استفاده مجدد را ممکن می‌سازد

تطبیق دو روش متریک(رابطه بین کارکردگرا و اندازه گرا)

معیارهای LOC و FP برای بدست آوردن معیارهای قابلیت تولید استفاده می شوند.

زبان برنامه سازی	میانگین تعداد خطوط به ازای هر امتیازکارکردی LOC / FP
Assembly Language	۳۲۰
C	۱۲۸
COBOL	۱۰۶
FORTRAN	۱۰۶
Pascal	۹۰
C++	۶۴
Ada ۹۵	۵۳
Visual Basic	۳۲
Smaltalk	۲۲
Powerbuilder (code generator)	۱۶
SQL	۱۲

- صحت (Correctness)
  - برنامه باید صحیح عمل کند و گرنه برای کاربر ارزشی نخواهد داشت
  - درستی عملکرد درجه‌ای است که در آن نرم‌افزار عملکرد مورد نیاز خود را بروز می‌دهد
  - از معمول‌ترین معیارهای درستی عملکرد می‌توان به تعداد خرابی در هر KLOC اشاره نمود
  - خرابی (شکست) بعنوان عدم تطابق مشاهده شده با نیازها تعریف می‌گردد

- قابلیت نگهداری (Maintainability)
  - نگهداری امکانی است که توسط آن در صورت بروز خطا می‌توان یک برنامه را اصلاح نمود و یا در صورت تغییر محیط آن را با محیط تطبیق داد، و یا در صورتی که مشتری خواهان تغییری در نیازهای نرم‌افزار باشد امکاناتی به آن اضافه کرد
- جامعیت (Integrity)
  - قابلیت یک سیستم را در مقابله با دستکاری‌های تصادفی و یا عمدی در راستای ایمنی آن اندازه گیری می‌کند

معیارهای کیفیت نرم‌افزار

برای اندازه گیری جامعیت دو ویژگی باید بررسی شود

- **تهدید:** بروز یک حمله از نوعی خاص و در محدوده زمانی مشخص
- **امنیت:** احتمال دفع نوعی خاص از حمله

- قابلیت استفاده (Usability)
  - برنامه باید User Friendly باشد
  - از طریق ۴ ویژگی این کار اندازه گیری می گردد

۱) مهارت فیزیکی یا ذهنی مورد نیاز برای یادگیری نرم افزار

۲) مدت زمان مورد نیاز برای کسب کارایی بالا در استفاده از سیستم

۳) خالص افزایش بهره وری در مقایسه با سیستم قبلی

۴) ارزیابی شهودی (گاهی این معیار از طریق پرسشنامه از نظرسنجی کاربران نسبت به سیستم بدست می‌آید)

فاکتورهای کیفیت ایزو ۹۱۲۶:

۱ــ قابلیت کارکردی

۲ــ قابلیت اطمینان

۳ــ قابلیت به کار گرفته شدن

۴ــ کار آیی

۵ــ قابلیت نگهداری شدن

۶ــ قابلیت حمل شدن

معیارهای کیفیت نرم‌افزار

- یکی از معیارهای کیفی که در هر دو سطح پروژه و فرآیند سودمند است

معیار قابلیت پالایش(فیلتر کردن) فعالیت‌های تضمین کیفیت و کنترل می‌باشد

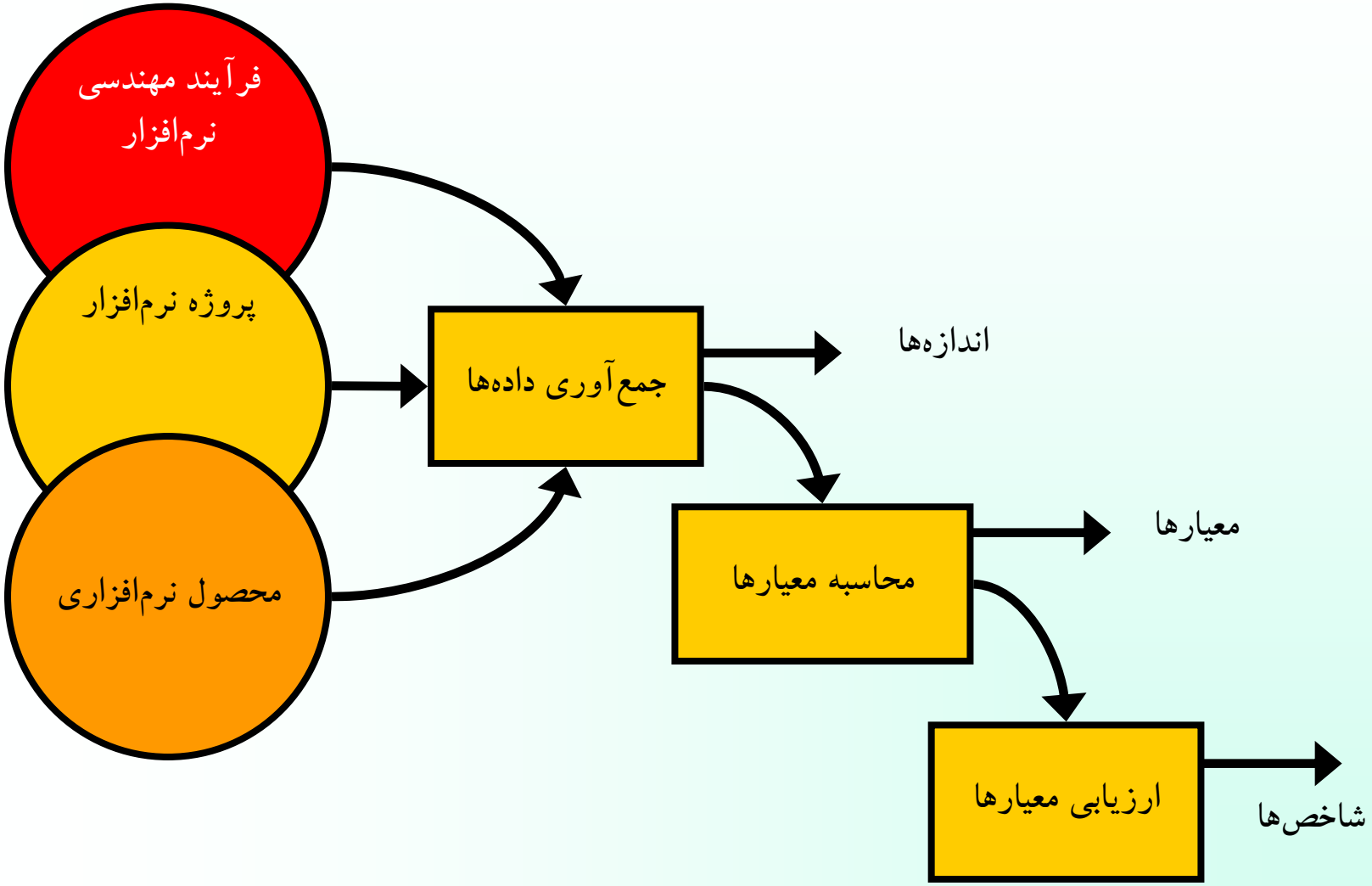
DRE = (Errors) / (Errors + Defects)

که Erorrتعداد خطاهایی است که قبل از تحویل نرم افزار به کاربر نهایی مشاهده شده

Defects خطاهایی است که بعد از تحویل یافت شده اند و در حالت عادی بزرگتر از ۰ است

Email: ComputerCollege_Fam@Yahoo.Com

فرآیند جمع‌آوری معیارها



UML

Systems Analysis and Design With UML
Component Object Modeling (COM)
Object Modeling Technique (OMT)
An Object-Oriented Approach
Unified Modelling Language
Rational Rose

شی و شی گرایی:

مجموعه ای از تفکرات و دید های قبلی است.
این مفاهیم چیزی جز مفاهیم طبیعی نیستند.
یک شی هر چیزی می تواند باشد.

یک شی نمونه ای از یک کلاس است .
نکته مهم : وقتی که یک شی را ایجاد می کنید ، محیط عملیاتی ، تعیین کننده صفات و عملیاتی است که یک شی می تواند به خود بگیرد .

استفاده از مفاهیم شی گرا ⟵ درک نیازهای کاربران

این نیازها را به تصویر بکشید ⟵ درک آن ها سریع تر و آسان تر

چرا از شی گرائی استفاده می کنیم؟

شی گرایی می تواند دارای منافع زیادی باشد.
متدولوژی شی گرا توانایی حل تمام مسائل را ندارد.

منافع این متدولوژی :

افزایش موفقیت :

موفقیت تیم پروژه ⟷ کاهش زمان تحلیل ، طراحی و برنامه نویسی
⇦ برنامه زمان بندی پروژه را به صورت چشمگیری کاهش می دهد.

توسعه سریع و کاهش هزینه :

مفهوم دوباره استفاده نمودن ⟷ بهبود زمان توسعه سیستم
عمل توسعه سیستم از طریق استفاده مجدد آنها باعث بالا رفتن بهره وری و بهبود کیفیت خواهد شد و بطور کلی به کاهش هزینه ها منجر گردد.

تاریخچه :

مطرح شدن دیدگاه شی گرایی (Object Oriented) از اواسط دهه ۱۹۷۰ تا اواخر دهه ۱۹۸۰.
افزایش تعداد متدولوژی ها در طول ۵ سال یعنی ۱۹۸۹ تا ۱۹۹۴.
دوران جنگ متدولوژی ها: تکثر متدولوژی ها و زبانهای شی گرایی و رقابت بین آن ها

از جمله متدولوژی های پرکاربرد :

Booch، OOSE ،OMT ،Fusion ،Coad-Yourdan ،Shlayer-Mellor ,...

مشکلات حاصل از فراوانی و اشباع متدولوژی ها و روش های شی گرایی و نیز نبودن یک زبان مدلسازی استاندارد :

پیامد ۱ :کاربران خسته !
پیامد ۲ :کاربران زده شده از دنیای شی گرایی !
پیامد ۳ : عدم وجود یک زبان استاندارد، برای فروشندگان محصولات نرم افزاری !

تاریخچه : (ادامه ...)

اولین تلاش های استانداردسازی از اکتبر ۱۹۹۴ آغاز شد.
Rumbaurgh صاحب متدولوژی OMT
Booch در شرکت Rational
و اولین محصول ترکیبی شان به نام "روش یکنواخت" را ارائه دادند.

در سال ۱۹۹۵: با پیوستن Jacobson،

روش یکنواخت + روش **OOSE** = **UML** نسخهٔ ۰,۹ در سال ۱۹۹۶

- ✓ استقبال شدید شرکت ها از این محصول
- ✓ تبلیغات گسترده شرکت Rational

⇦ پذیرفته شدن نسخهٔ UML۱,۰ به عنوان زبان مدلسازی استاندارد از سوی OMG(Object Management Group)

UML چیست ؟

UML : زبانی استاندارد به منظور مشخص نمودن، پیش بینی، ایجاد و مستندسازی تولیدات نرم افزاری.

مجموعه ای است از بهترین امکانات مهندسی به منظور استفاده در مدلسازی سیستم های بزرگ .

UML یک متدولوژی برای ساخت نرم افزار است.

UML یک ابزار ویژوال بوده که از انواع متفاوتی از نمودارها استفاده می کند و هریک از نمودارهای آن امکان مشاهده یک سیستم نرم افزاری را از دیدگاه های متفاوت و با توجه به درجات متفاوت تجرید (Abstraction) در اختیار پیاده کنندگان قرار می دهد.

UML چیست ؟(ادامه ...)

UML مکانیزمی برای استفاده برنامه نویسان نرم افزار

- ✓ درستی دریافت درخواست مشتری
- ✓ جلوگیری از ابهام و دوباره کاری در نوشتن برنامه

مساله دیگر اینکه، UMLیک زبان برنامه نویسی بصری (visual) نیست.

مهندسی رو به جلو: نگاشت از مدل های UML به کد زبان های برنامه نویسی

آیا عکس این عمل نیز ممکن است؟

مهندسی معکوس: بدست آوردن مدل های UML از کد یک برنامه زبانی شی گرا.

مزیت استفاده از UML تفکر مبتنی بر برنامه نویسی شی گراست.

مدلسازی :

تعریف: شبیه سازی یک محیط با اندازه های متفاوت از محیط واقعی و احتمالا مواد و مصالحی متمایز از جنس مواد ومصالح محیط مدل شده.

اهداف :

- ✓ شناخت (Exploration) : هدف شناخت محیط مورد مدل می باشد.
- ✓ تبیین(Specification) : معرفی و ارائه خصوصیات موجودیت واقعی یک مدل.

نکته : متدولوژی ساخت مدل.

ساده یا پیچیده !

روش های استاندارد زیادی مثل : USDP ، RUP ، SSAPM که روش های تولید نرم افزار است.

مدلسازی :

چرا مدلسازی می کنیم؟

ضروری بودن مدلسازی یک سیستم نرم افزاری با قدرت صنعتی قبل از ساخت یا نوسازی.

ضروری بودن مدل های خوب، برای ارتباط افراد در گروه های پروژه با یکدیگر و نیز اطمینان از قوت معماری.

دلیل ایجاد مدل هایی از سیستم های پیچیده؟

پیچیدگی سیستم ها و اهمیت تکنیک های مدلسازی خوب.

وظیفه UML :

با وجود عوامل متعدد موفقیت پروژه ولی داشتن یک زبان استاندارد مدلسازی یکی از عوامل ضروری است و این همان چیزی است که UMLفراهم می کند.

سیستم های نرم افزاری و متدولوژی:

تعریف: متدولوژی یا فراروش، مجموعه ایست همگرا و هدف مدار از مفاهیم، عقاید و ارزش ها و اصولی که به وسیله منابعی در جهت حل مسایل گروهی به کار گرفته می شود.

چرخه ی حیات طراحی وتولید سیستم های نرم افزاری :

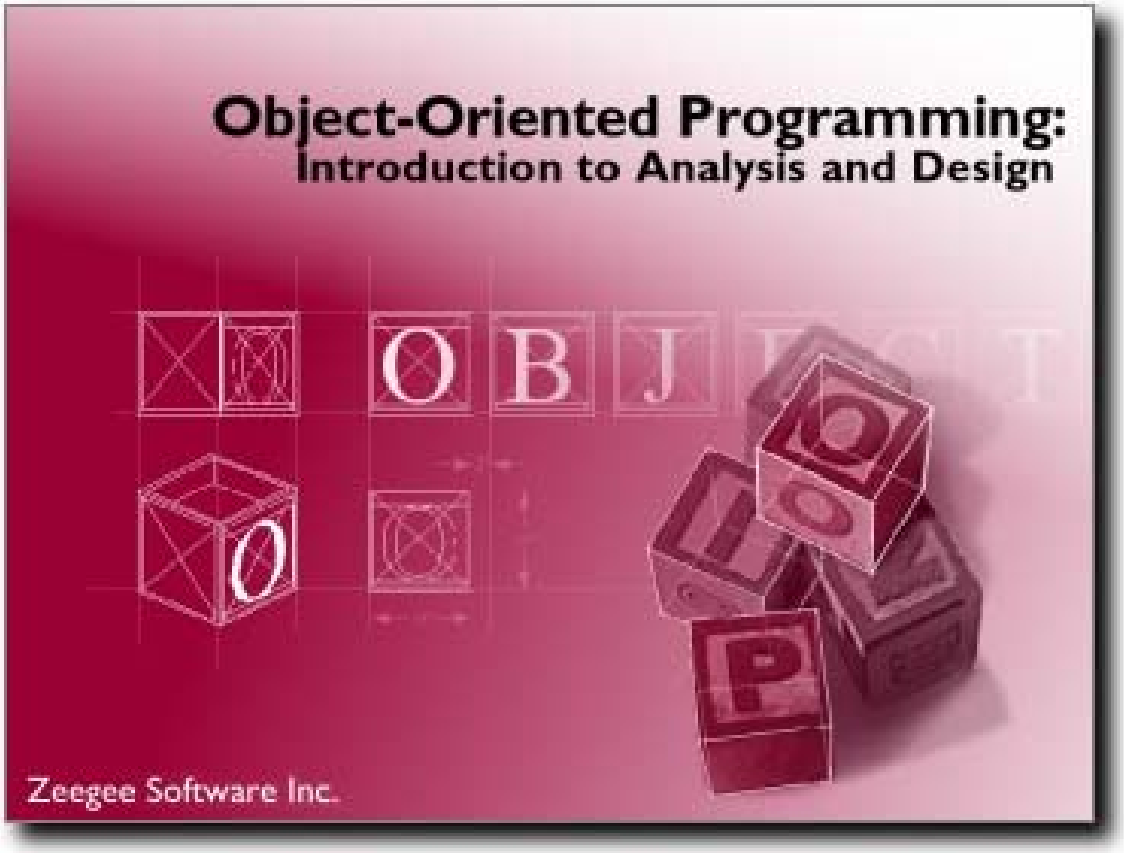
- ✓ فعالیت جمع آوری نیازمندی ها و مشخص کردن آن
- ✓ فعالیت تحلیل نیازمندی ها برای درک بهترآنها
- ✓ فعالیت طراحی برای اینکه مشخص شود که سیستم چگونه نیازمندی ها را برآورده می کند
- ✓ فعالیت ساخت سیستم
- ✓ آزمایش سیستم
- ✓ فعالیت تحویل سیستم

متدولوژی های مختلفی برای انجام این فعالیت ها وجود دارد.

هدف←ما متد شی گراست **UML**

روند کار در متد شی گرا:

OOA → OOD → OOP



آنالیز شی گرا (OOA): یک متدولوژی برای تجزیه و تحلیل فرایند ساخت نرم افزار محور آنالیز سیستم: هر چیز در فرایند پیاده سازی نرم افزار به منزله کلاس در نظر گرفته خواهد شد تاکید و سرو کار داشتن با سوالاتی که با What شروع می شوند.

طراحی شی گرا (OOD): تاکید و سرو کار داشتن با سوالاتی که با How شروع می شوند.

در طراحی شی گرا تاکید بر پیاده سازی کلاس ها و صفات وخصایصی است که به منزله هسته یک کلاس مطرح می گردد.

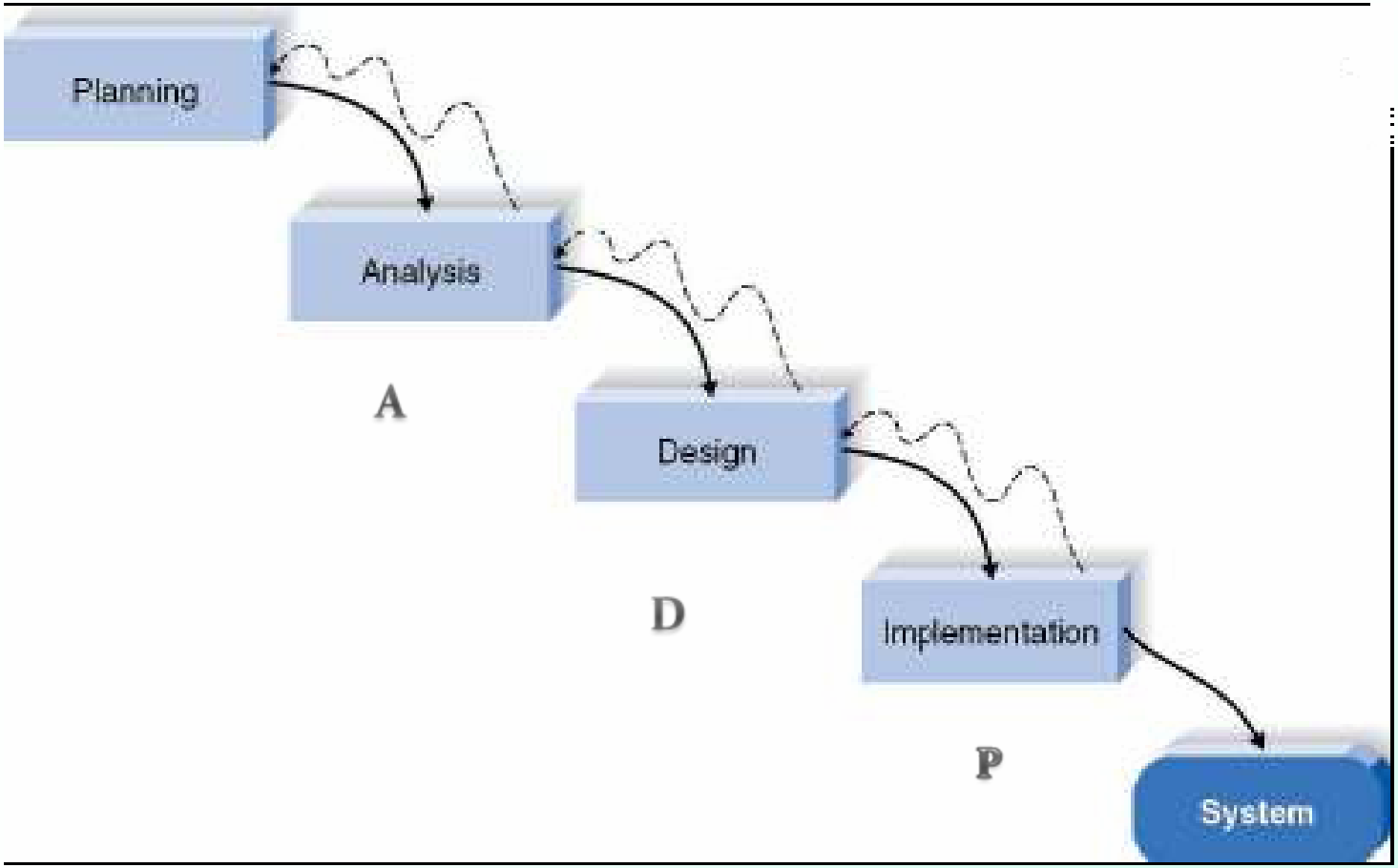
OOP = لینک هایی که با کلاس ها سروکار دارند + OOD + OOA

Email: ComputerCollege_Fam@Yahoo.Com

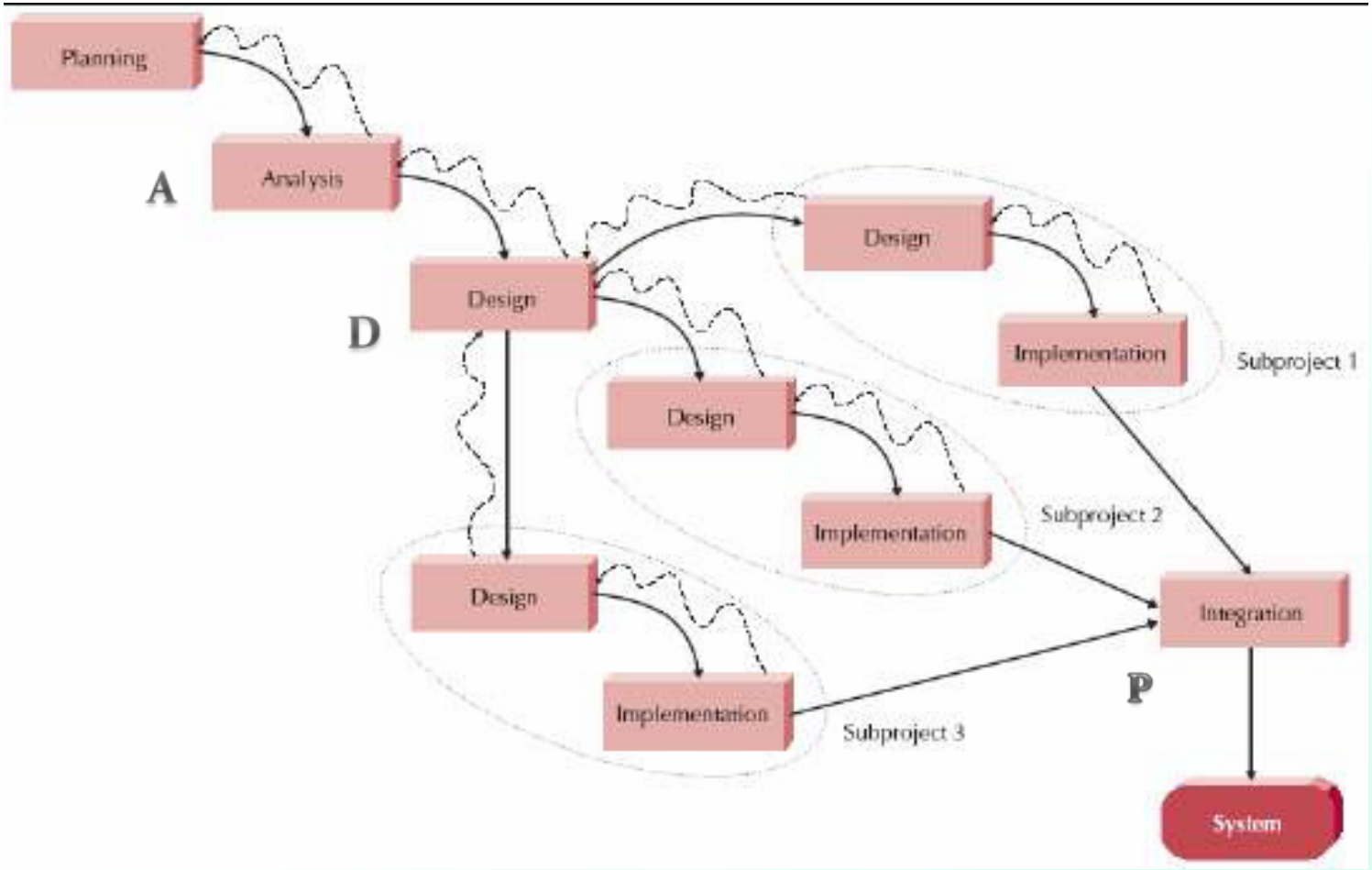
مصنوعات (deliverable) تولید شده درهر فاز پروژه:

Process	Product
Planning	Project Plan
Analysis	System Proposal
Design	System Specification
Implementation	New System and Maintenance Plan

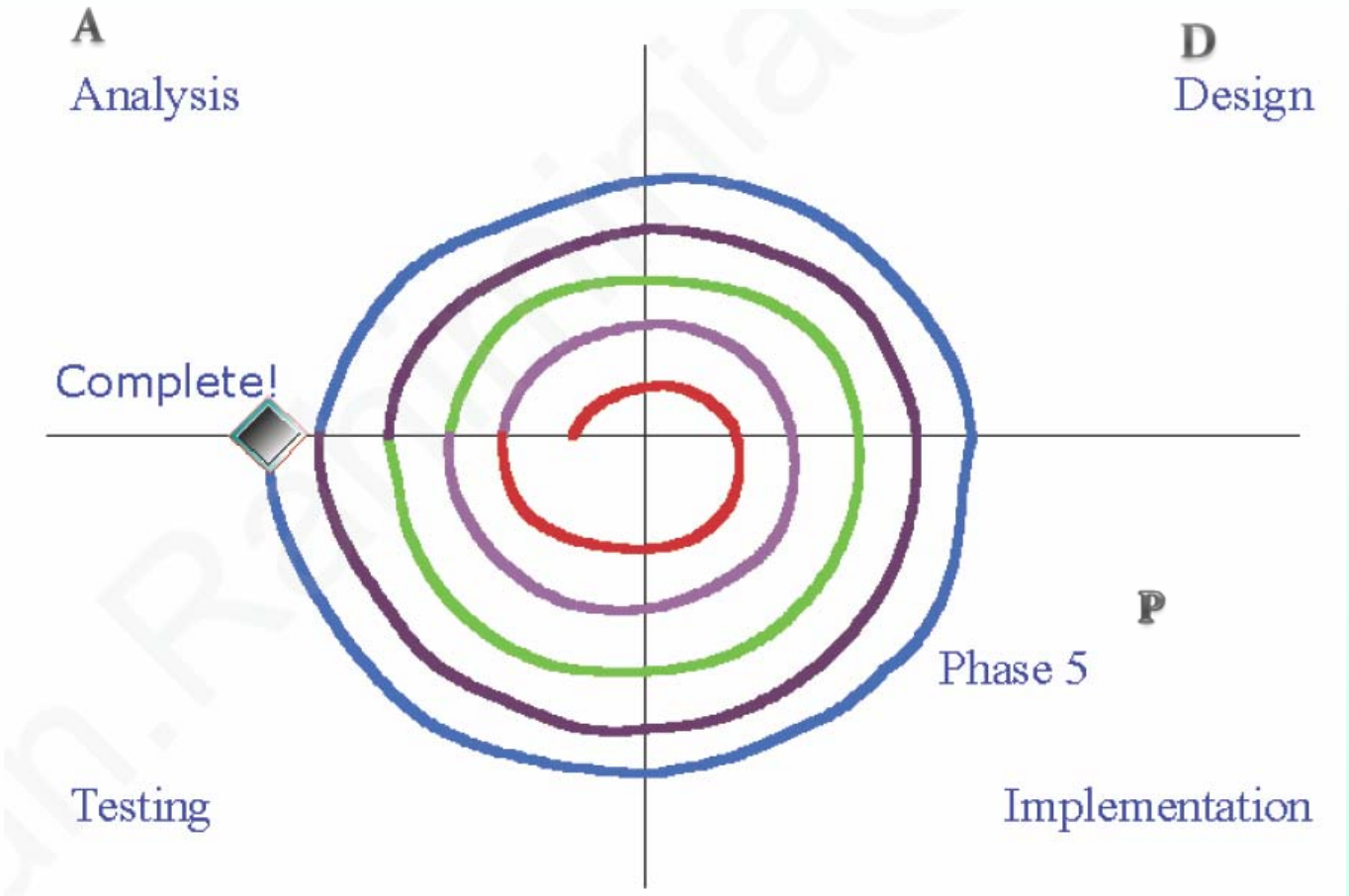
متدلوژی آبشاری(متد ساخت یافته):



متدلوژی موازی (متد ساخت یافته) :



متدلوژی حلزونی (متد ساخت یافته) :



مفاهیم متد شی گرا :

شی(Object) :

به هرمفهوم یا هرچیز قابل درکی که بشود با خصوصیات و رفتار مستقل آن را از یک محیط باز شناسی کرد.

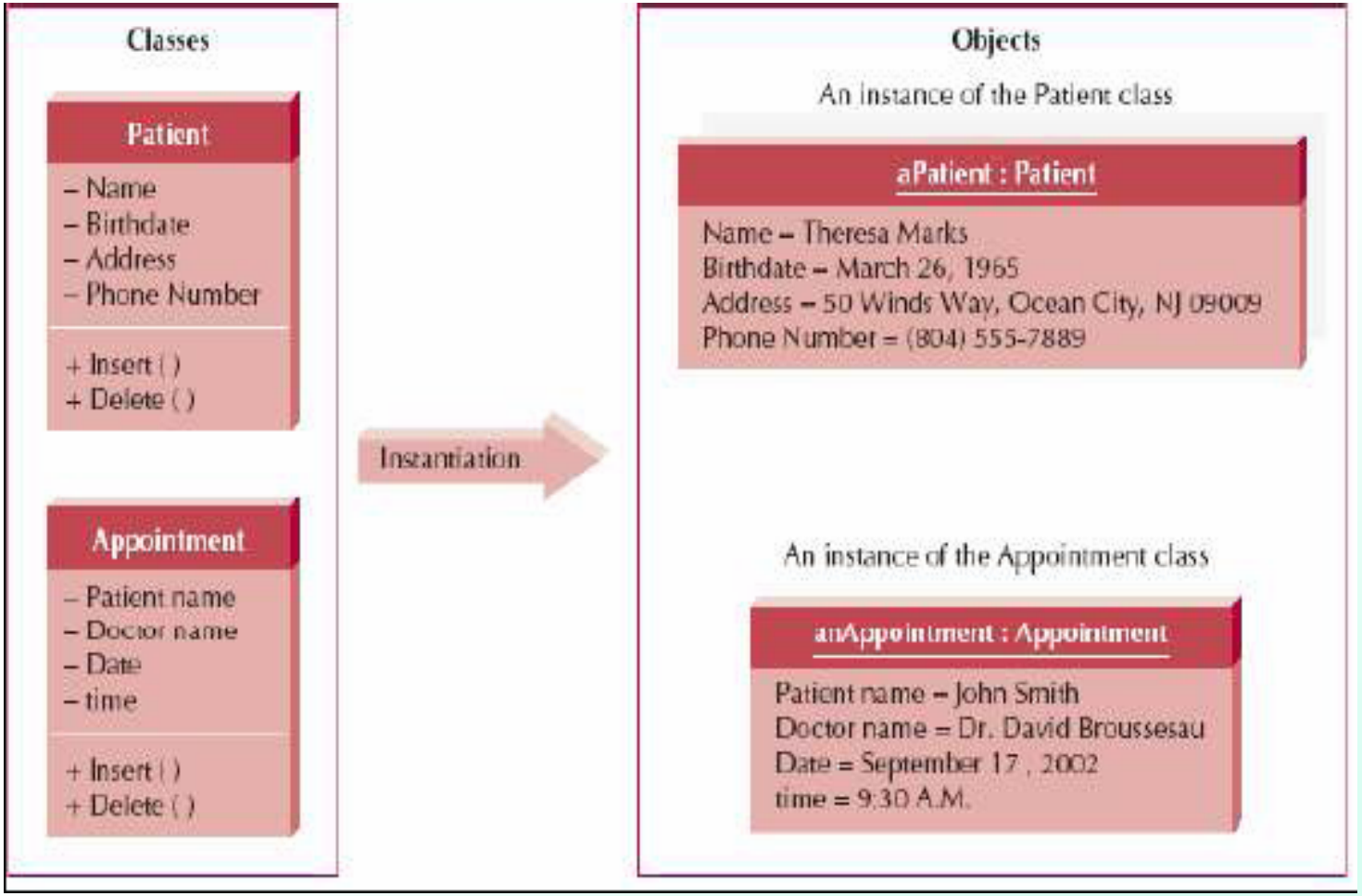
کلاس(Class) :

مفهومی انتزاعی برای دسته بندی اشیا
نشان دهنده خصوصیات و رفتار گروه خاصی از اشیا.
خصوصیات و رفتارها مشترک هستند.

صفت(Attribute) :

هر شی یکسری خصوصیات دارد که به آنها صفت گفته می شود که در واقع یک مقدار یا ارزش مشخصی برای آن به ازای هر شی می تواند وجود داشته باشد.

: Class & Object



روش(Method) :

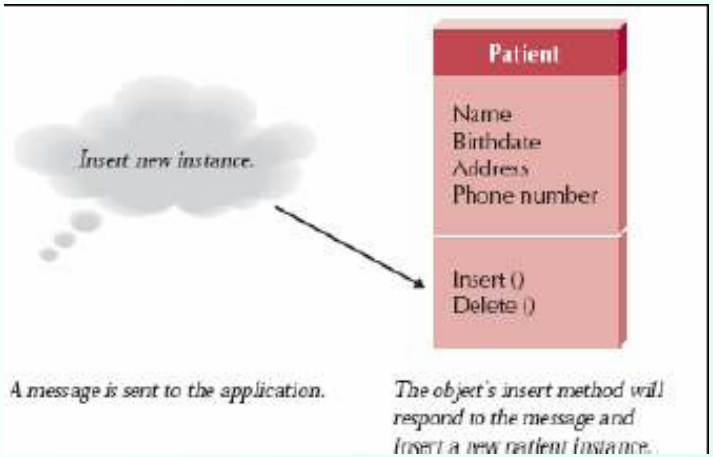
هر شی یکسری رفتار دارد که به آنها متد گفته می شود در واقع پاسخ هایی است که آن شی در قابل تحریکات محیط از خود صادر می کند.

ارسال پیام(Message Sending):

در یک سیستم ، اشیا با هم کار می کنند.

فرستادن پیام به یکدیگر.

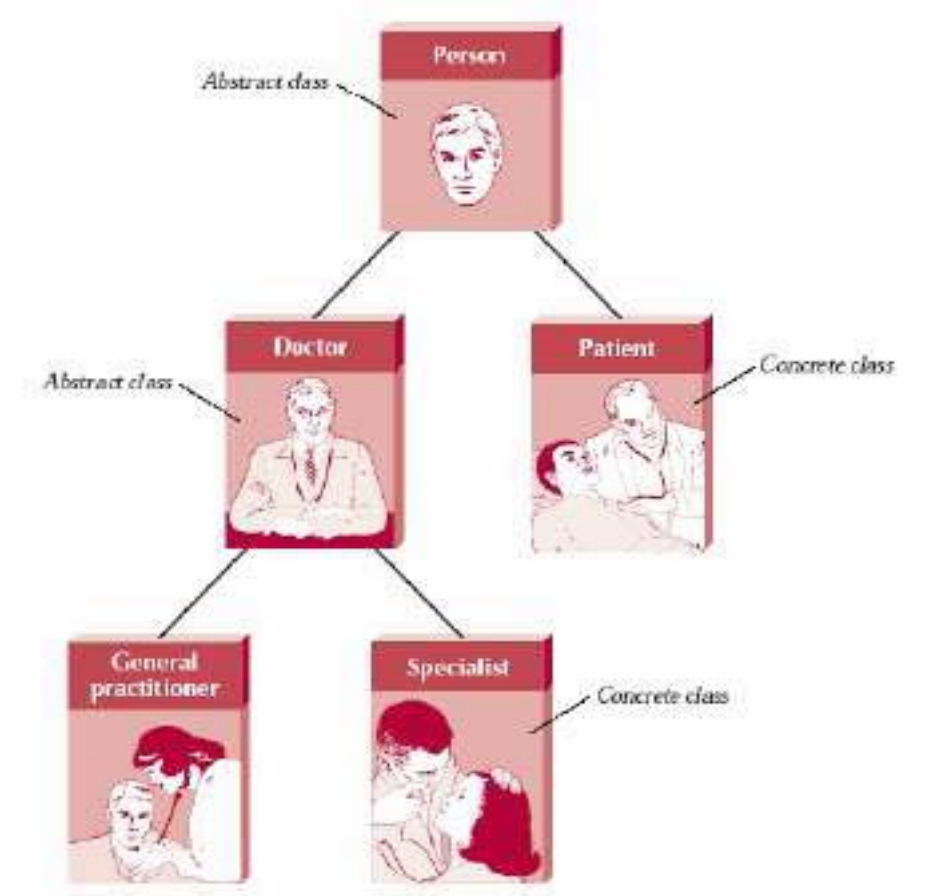
یک شی پیامی را برای اجرای عملیات به شی دیگر می فرستد، شی گیرنده پیام آن عملیات را اجرامی کند.



میراث(Inheritance)

میراث یکی از جنبه های مهم در مفاهیم شی گرایی است .

یک شی صفات وعملیات کلاس خودش وهمچنین کلاس دیگری را می تواند به ارث ببرد.



سناریو(Scenario):

یک مجموعه پشت سر هم یا متوالی می باشد که منجر به انجام کار خاصی می گردد.

تجرید(Abstraction):

صفات و عملیات یک شی را آنقدر از صافی بگذرانیم تا مجموعه ای که مورد نیاز است باقی بماند .

دید محدودی است که فقط جنبه های خاص مورد نظر در داخل آن آورده می شوند.

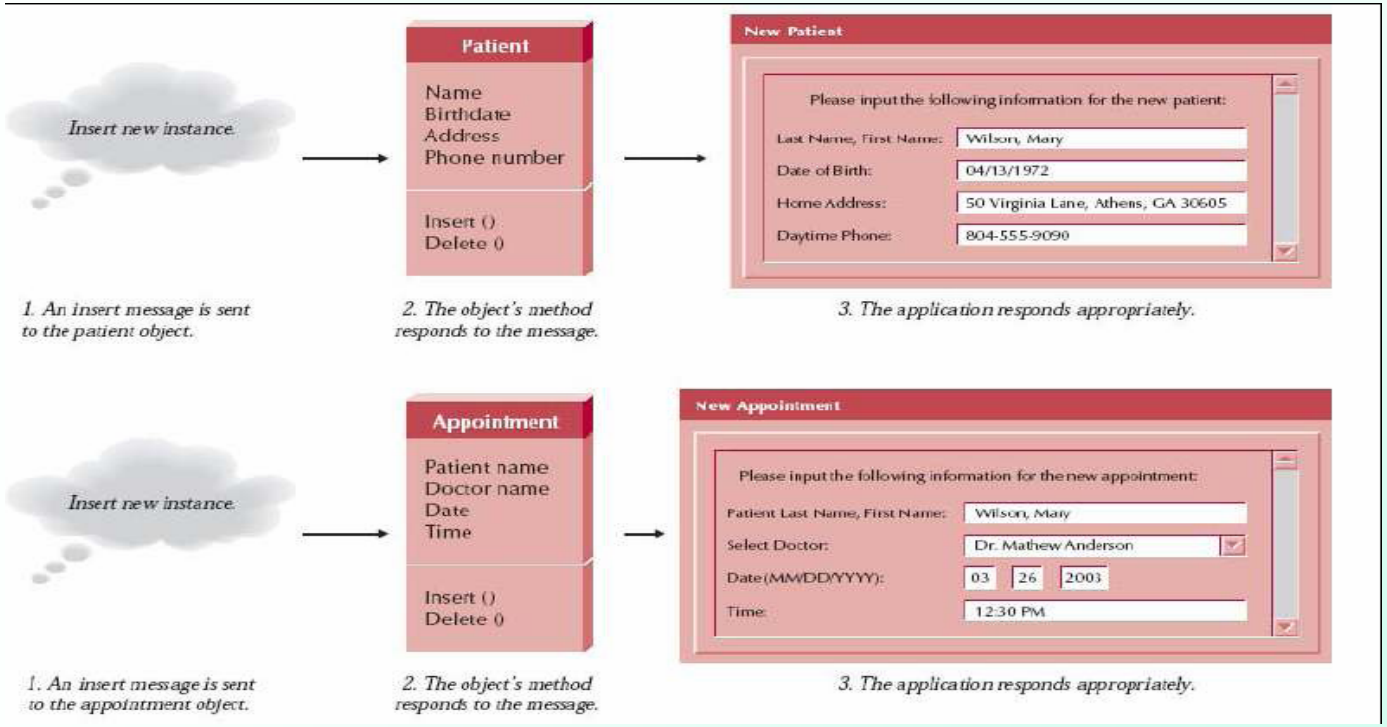
این یک دید کلی بدون نگاه کردن به جزئیات می باشد .

چند شکلی(Polymorphism)

عملیات ها می توانند نام مشابهی در کلاس های متفاوت داشته باشند و هر کلاس عملیات را به شکلی مختلف انجام دهد.

(مثال)

بطور کلی می توان گفت که چند شکلی به معنای یک چیز بودن و چند شکل داشتن است .



پنهان سازی(Encapsulation)

اشیا عملیات داخلی خود را از دید بینندگان خارج از دنیای خود پنهان می سازند.
پنهان سازی: هر شی بطور مستقل دارای داده ها و فرآیند های مورد نیاز خود است.

شی ماهیتی تقریبا مستقل از اصل و محیط اطراف خود دارد.

نمونه سازی(Instantiation) :

به معنای ایجاد یک شی از یک کلاس است یعنی نمونه ای از آن کلاس که همان شی می باشد تولید می گردد .

مسئولیت(Responsibility) :

مسئولیت چیزی است که به شی اختصاص داده می شود و سه جنبه دارد :

- ✓ آنچه که شی راجع به خودش می داند.
- ✓ کسانی که شی را می شناسند با آن ارتباط دارند.

تناظر یا ارتباط (Associations)

اشیا در بعضی حالات به یکدیگر وابسته هستند.

مثال:

تناظر یک ارتباط ساختاری بین دو شی است .

یک شی از یک کلاس ممکن است با چند شی از کلاس دیگر متناظر باشد .

تجمع (Aggregation)

نشان دادن وابستگی بین اجزا و کل.

تجمع نوع دیگر از تناظر یا ارتباط بین اشیا می باشد.

ترکیب نوع دیگری از تجمع است که درگیر یک ارتباط قوی بین شی تجمع با اشیای جزءاش می باشد

ابزارهای موجود برای طراحی شی گرا :

Microsoft Visio

UML star

Enterprise Architect

VP Suite Windows

Rational Rose

……

Email: ComputerCollege_Fam@Yahoo.Com

Rational Rose

Rational Object Oriented Software Engineering

چرا Rational ؟

- ✓ مبتنی بودن بر تصویر منجر به کیفیت بالای نرم افزار می شود.
- ✓ با استفاده از زبان استاندارد متداول (UML) اعضای تیم می توانند ارتباط موثرتری با هم داشته باشند.
- ✓ امکانات مهندسی معکوس، توسعه دهندگان را قادر به استفاده از سیستم های شی گرای قبلی می سازد.
- ✓ مدل ها و کد، درطول چرخه توسعه هم گام می مانند.

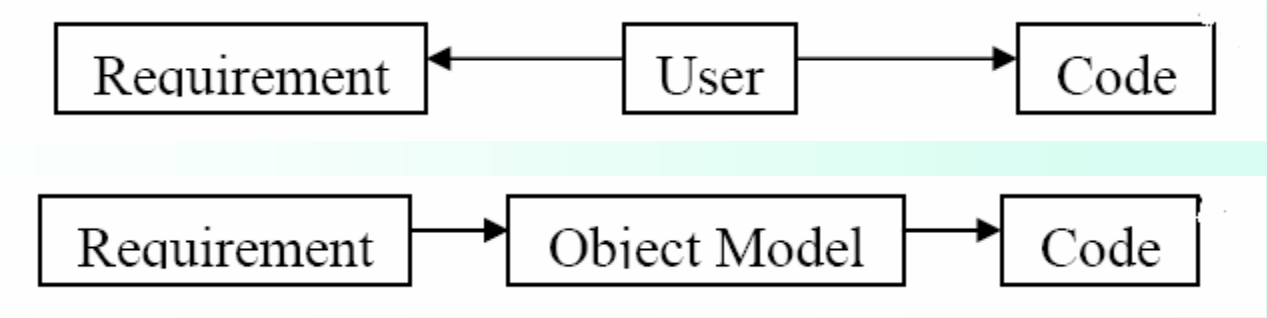
نسخه های مختلف نرم افزار Rose :

Rose Modeler : فقط اجازه ایجاد مدل را می دهد و تولید کد و مهندسی معکوس منتفی است !

Rose Professional : فقط اجازه تولید کد را می دهد.

Rose Enterprise : ایجاد مدل، تولید کد و مهندسی معکوس امکان پذیر است.

طرح جدید و قدیم :



معرفی محیط رشنال رز :

Rational Rose - (untitled)

File Edit View Format Browse Repor: Query Tools Add-Ins Window Help

(untitled)
- Use Case View
- Logical View
- Component View
- Deployment View
- Model Properties

Class Diagram: Logical View / Main

Create New Model

Rose 2000

New | Existing | Recent

jdk-116, jdk-12, jenterprise, jfc-11, Make New Framework, oracle8-dat..., rational unified process, VB5 Standard, VB6 Standard, VC6 MFC 6.0

OK | Cancel | Details >> | Help

Don't show this dialog in the future

start | Adobe Reader - [rati... | rational - Microsoft W... | Rational Rose - (untitl... | EN | 02:04 ب.ظ

Email: ComputerCollege_Fam@Yahoo.Com

معرفی محیط رشنال رز :

عناصر اصلی رشنال رز عبارتند از:

Standard toolbar : برای تمام دیاگرام ها مشترک است و در قسمت بالای پنجره واقع است.

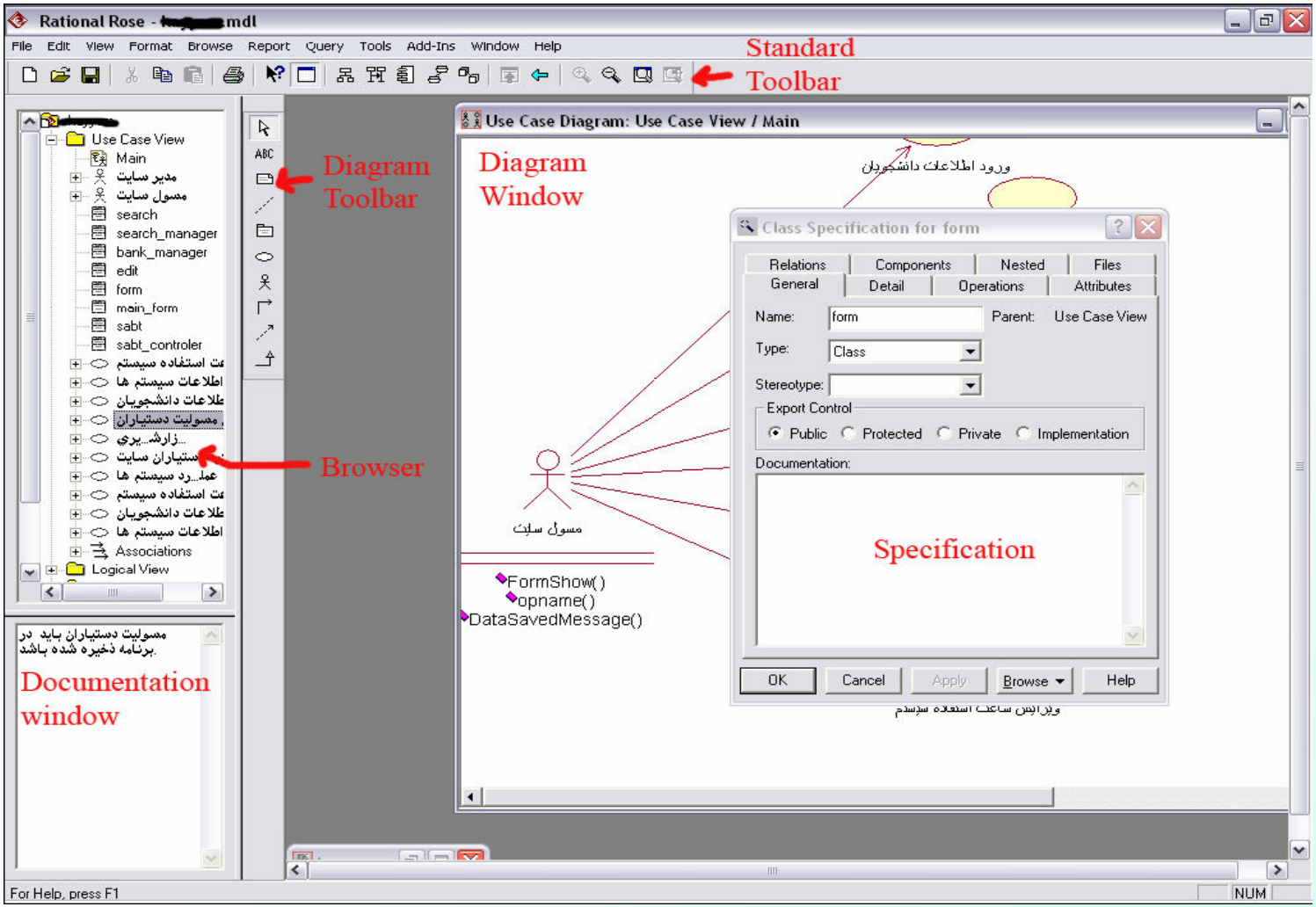
Diagram toolbar : که وابسته به پنجره ی دیاگرام فعال است و در سمت چپ پنجره ی دیاگرام واقع است.

Browser : به شما اجازه می دهد تا بصورت یک ساختار درختی دیاگرام های موجود و عناصر مدلهایتان را مشاهده کنید.

Diagram window : ساخت و ویرایش دیاگرام ها در این قسمت صورت می پذیرد.

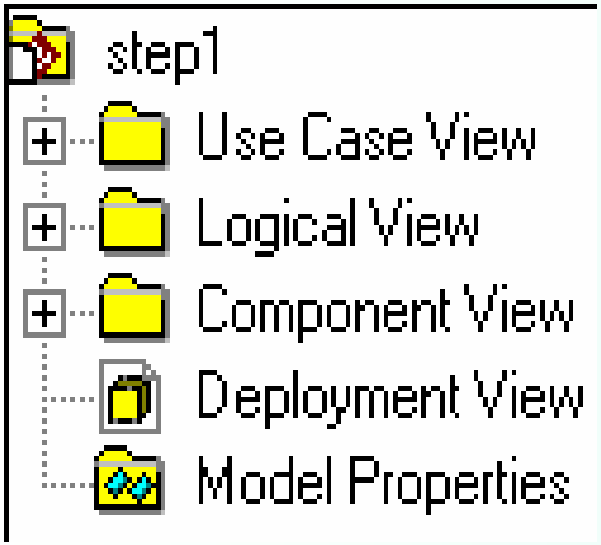
Documentation window : به شما اجازه می دهد تا به مدلهایتان مستندات لازم را نیز اضافه نمایید . می توانید مستنداتتان را در این قسمت یا در قسمت specification ویرایش نمایید.

Specification: محیط ویرایشی برای اضافه کردن مستندات به مدل.



معرفی نماهای رشنال رز:

رشنال رز نماهای زیر را برای یک پروژه فراهم می آورد.



Usecase view –

Logical view –

Component view –

Deployment view -

چهار دید متفاوت در طراحی نرم افزار

UseCase View

دیاگرام کاربرد (UseCase Diagram)
دیاگرام توالی (Sequence Diagram)
دیاگرام همکاری (Collaboration Diagram)
دیاگرام فعالیت (Activity Diagram)

Logical View

دیاگرام کلاس (Class Diagram)
دیاگرام حالت (Statechart Diagram)

Component View

دیاگرام اجزا (Component Diagram)

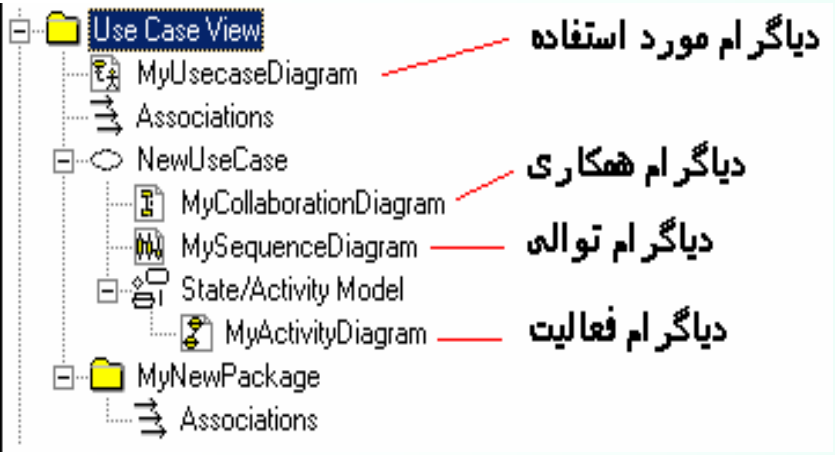
Deployment View

دیاگرام پیاده سازی (Deployment Diagram)

نمای مورد استفاده ی سیستم (usecase view):

این نما تشریح رفتار سیستم از دیدگاه کاربر است.
فعل وانفعالات متقابل بازیگرها و موردهای استفاده نمایش داده می شود.
در این نما چهار دیاگرام زیر وجود دارند :

- دیاگرام های موردهای استفاده **(usecase diagrams)**
- دیاگرام های توالی **(sequence diagrams)**
- دیاگرام های همکاری **(collaboration diagrams)**
- دیاگرام های فعالیت **(activity diagrams)**

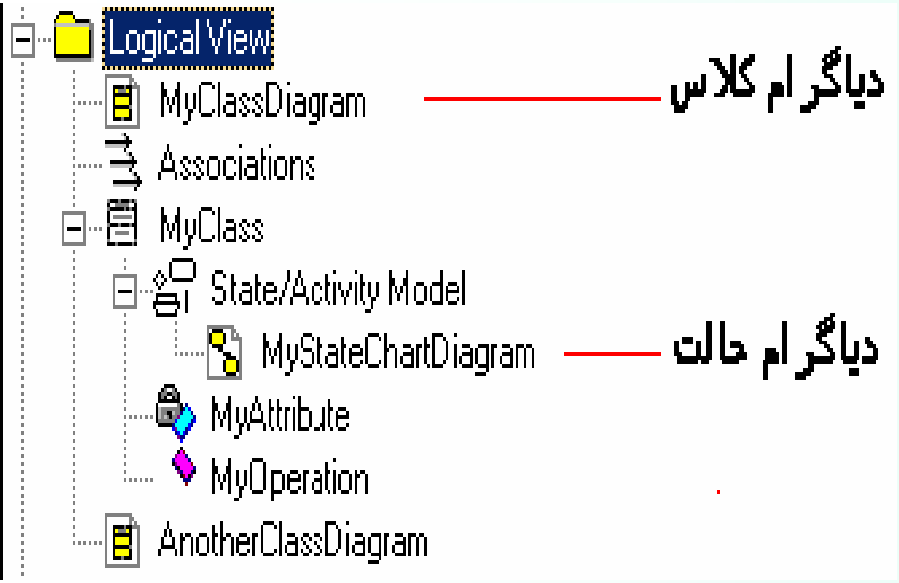


نمای منطقی سیستم(logical view):

این نما شامل نیازمندی های عملیاتی سیستم می باشد که به کلاس ها وارتباط بین آنها می پردازد.
این نما شامل دو دیاگرام زیرمی باشد:

-دیاگرام های کلاس ها (class diagrams)

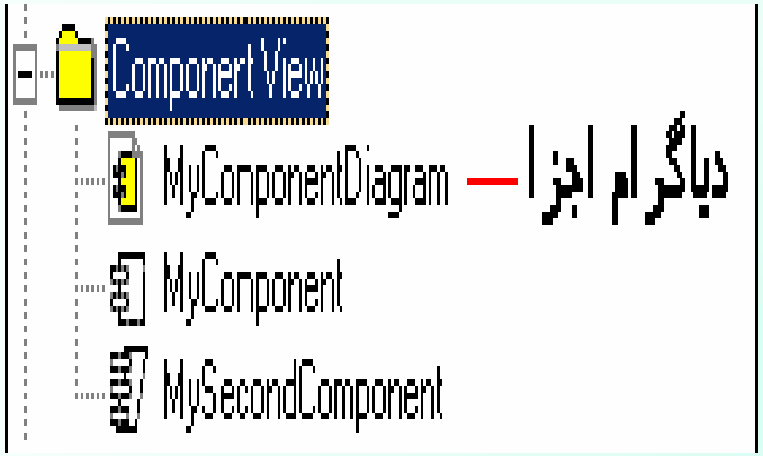
-دیاگرام های حالت (statechart diagrams)



نمای اجزای سیستم (component view):

این نما به سازمان سیستم می پردازد و اطلاعاتی در باره ی نرم افزار، اجزا قابل اجرا و کتابخانه های سیستم دارد.

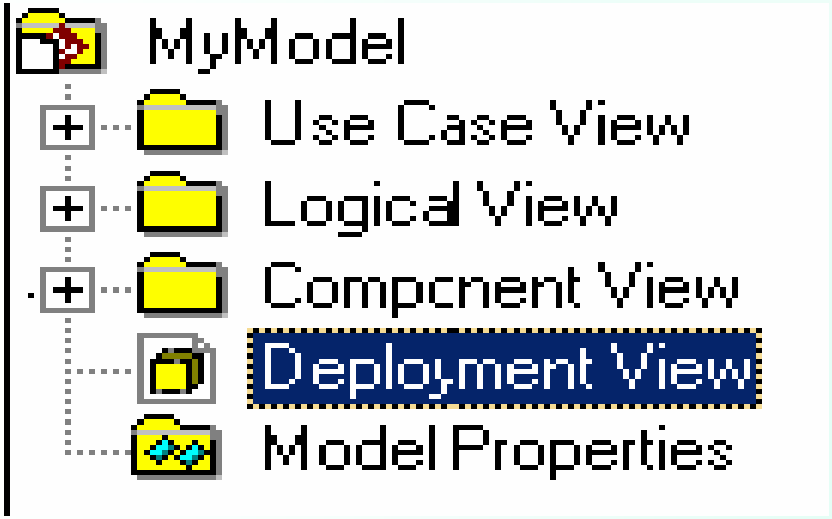
تنها دیاگرام موجود در این قسمت دیاگرام اجزا(component diagram) می باشد.



نمای پیاده سازی سیستم(deployment view):

این قسمت شامل نگاشتی از فرایندهای موجود با سخت افزار سیستم می باشد .

این نما فقط شامل یک دیاگرام (deployment diagram) می باشد.



معرفی دیاگرام های رشنال رز:

یک دیاگرام یک نمایش گرافیکی از عناصر سیستم می باشد.

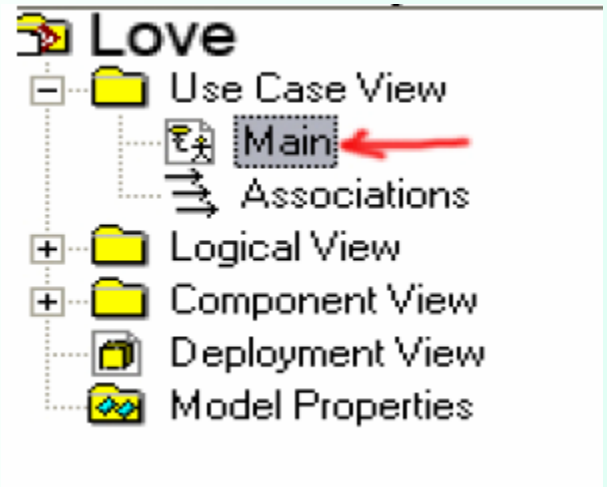
رشنال رز دیاگرام های زیر را دارد

دیاگرام مورد استفاده (usecase diagram)
دیاگرام کلاس (class diagram)
دیاگرام توالی (sequence diagram)
دیاگرام همکاری (collaboration diagram)
دیاگرام فعالیت (activity diagram)
دیاگرام حالت (statechart diagram)
دیاگرام اجزا (component diagram)
دیاگرام پیاده سازی (deployment diagram)

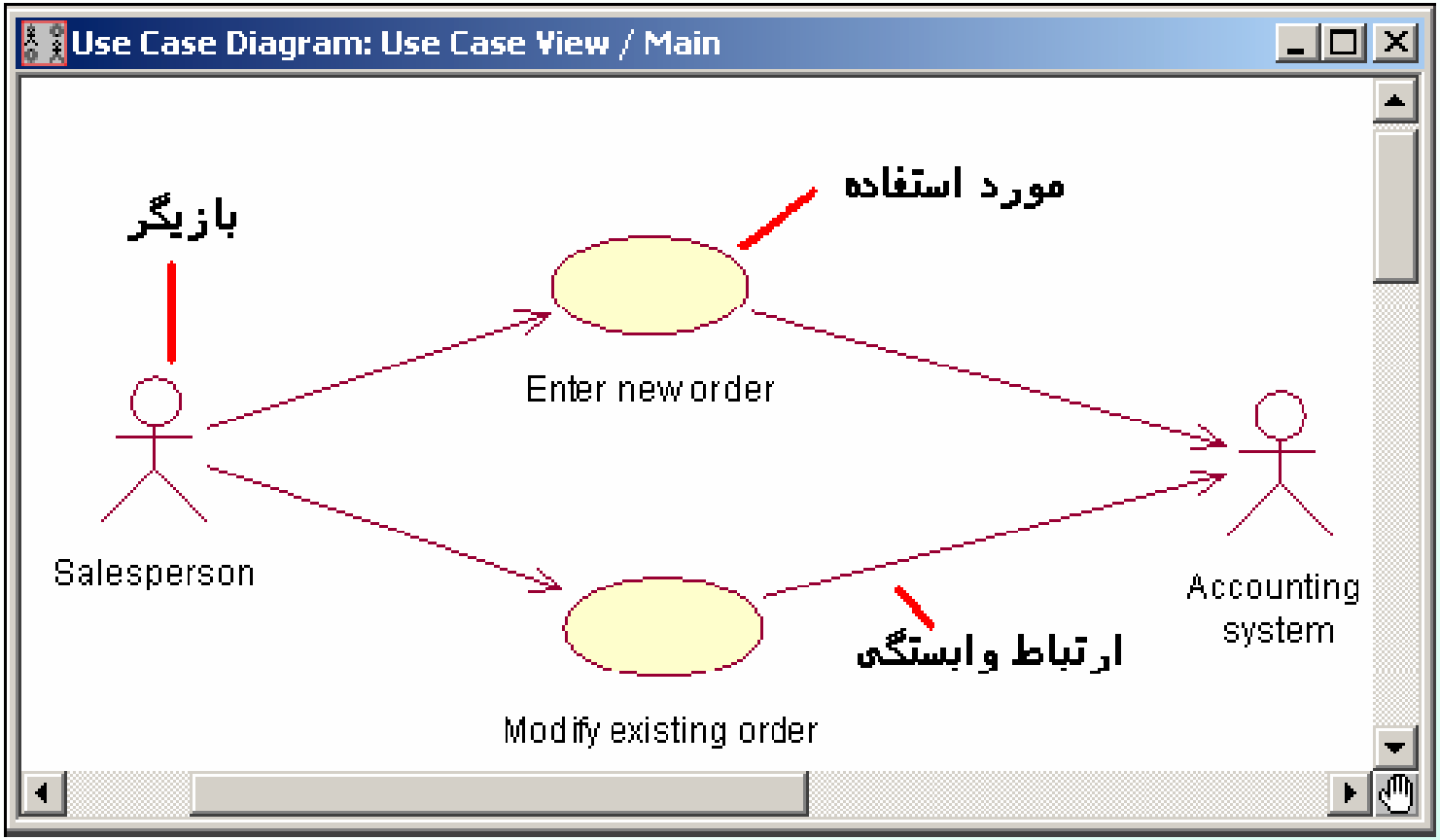
دیاگرام موردهای استفاده (usecase diagram) :

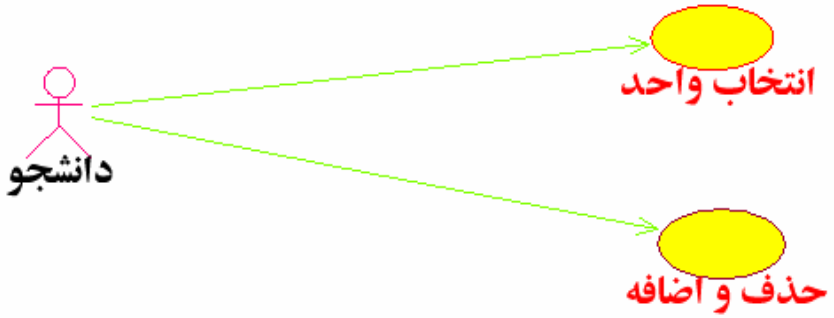
یک usecase رفتار سیستم را توصیف می کند، که شامل تقابل بین سیستم و بازیگران می باشد. نحوه ی برخورد آن با دنیای بیرون را مشخص می کنند.

هدف : بدست آوردن نیازمندی ها و نشان دادن چگونگی کارکرد سیستم



usecase diagram:




انتخاب واحد
دانشجو
حذف و اضافه

اجزائی که در یک Use case مطرح می شود

سناریو

جریان اصلی usecase(main flow) و جریان فرعی (alternative flow)

شرایطی که باید ایجاد شود تا usecase فعال شود(precondition)

دانشجو جهت انتخاب رشته رمز عبور خود را به سیستم می دهد

شرایطی که بعد از اتمام کار usecase ایجاد می شود (postcondition)
به دانشجو یک برگه تایید انتخاب واحد صادر می شود

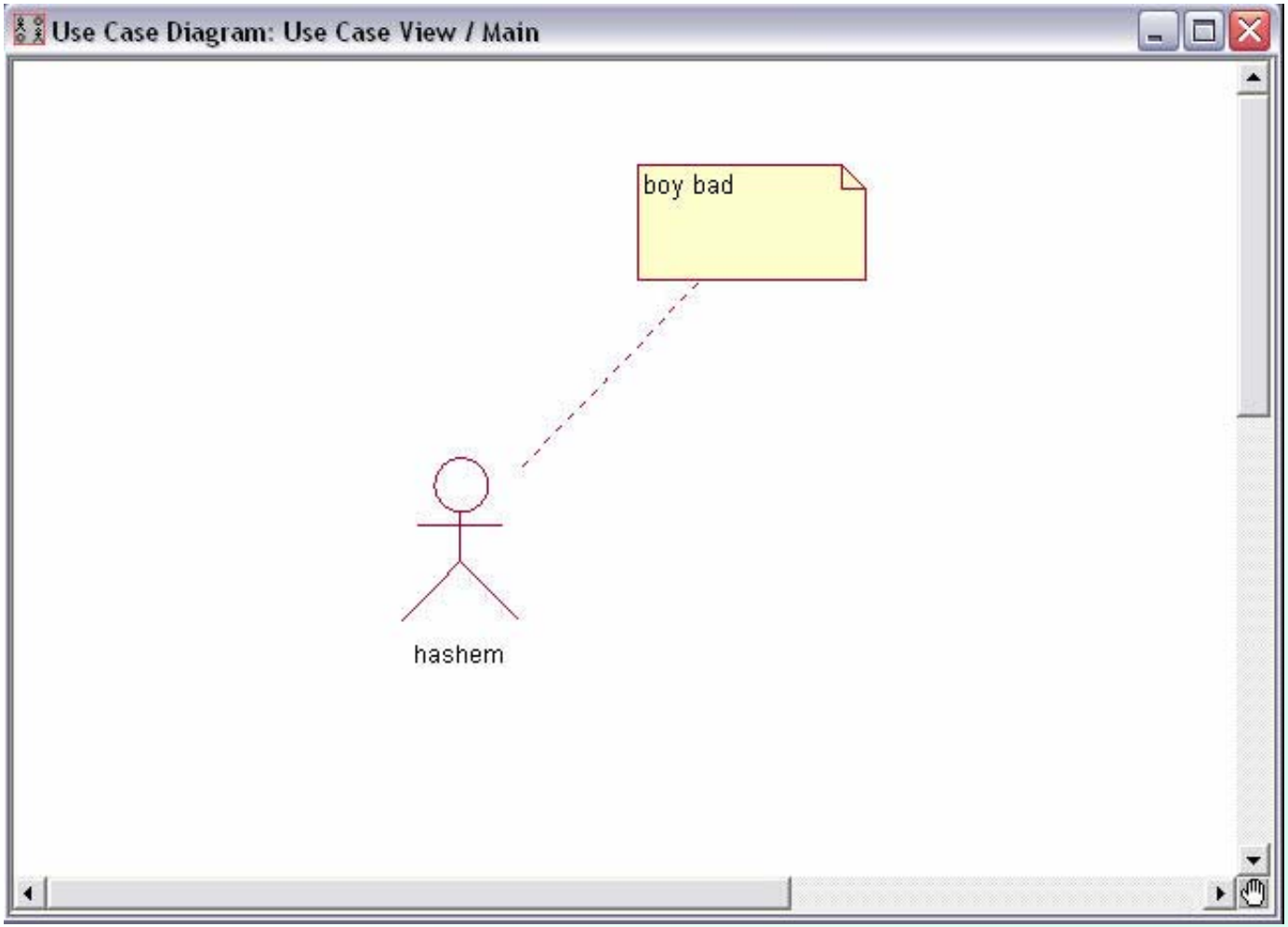
هدف usecase
ثبت دروس انتخابی دانشجو در سیستم

Diagram Toolbar

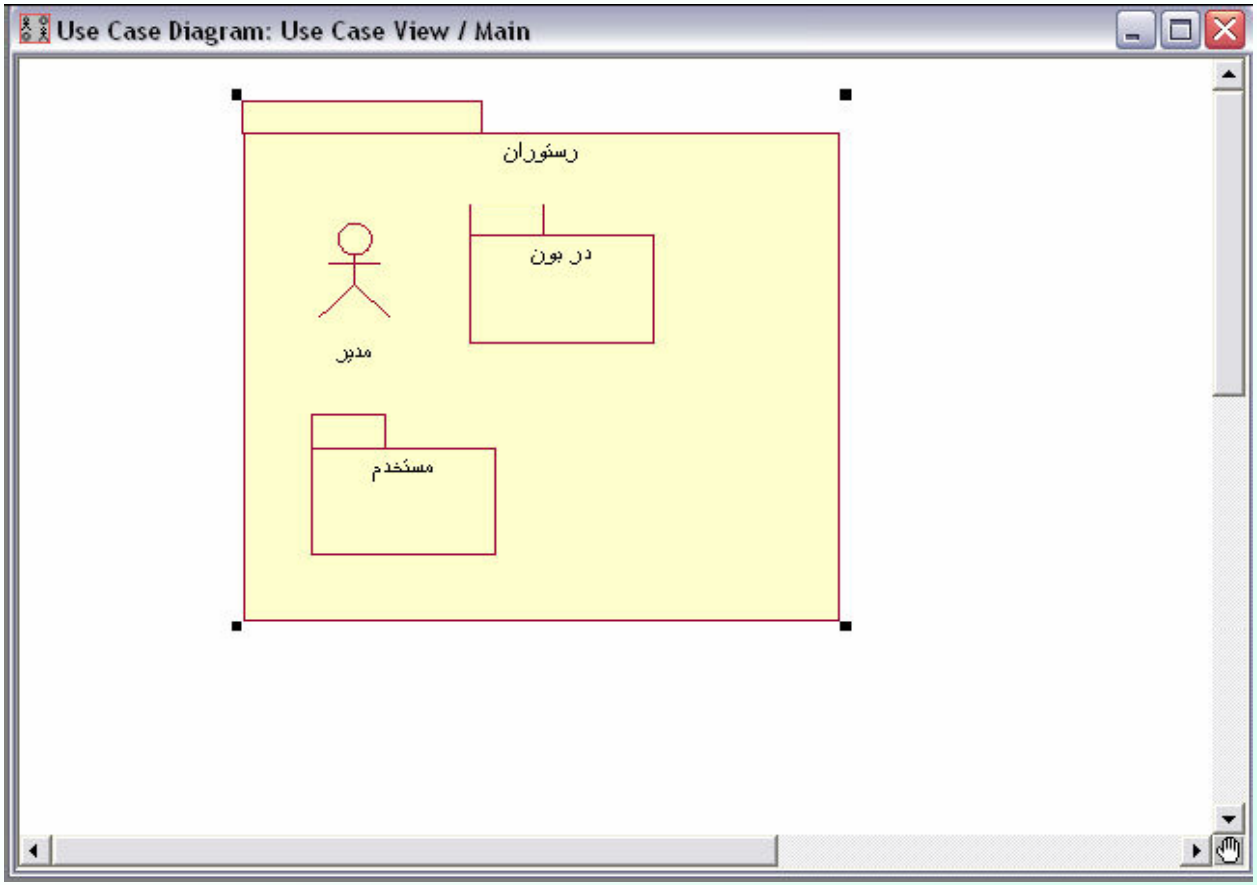
	برای انتخاب کردن شکل ها ی دیاگرام		مورد استفاده – کاری که انجام می شه
ABC	برای نوشتن متن یا هر Text دیگری		کاربر

	نوشتن توضیحات تو دیاگرام		ارتباط مستقل
	خط ارتباط بین توضیحات و هر شکل دیگر در دیاگرام		ارتباط از نوع وابسته
	بسته که زیر سیستم های ما که می تونه شامل کلاس ها و دیگر اجزا این زیر سیستم باشه را مشخص می کنه		ارتباط از نوع تعمیم یافته

usecase diagram

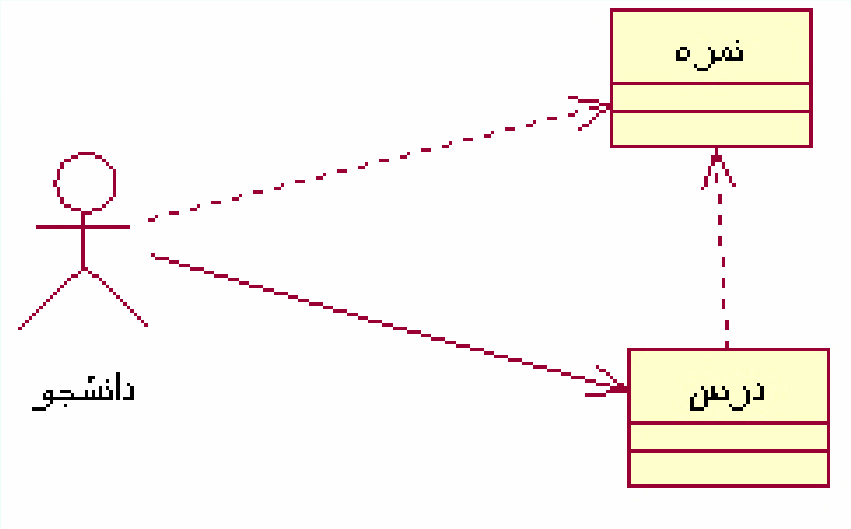


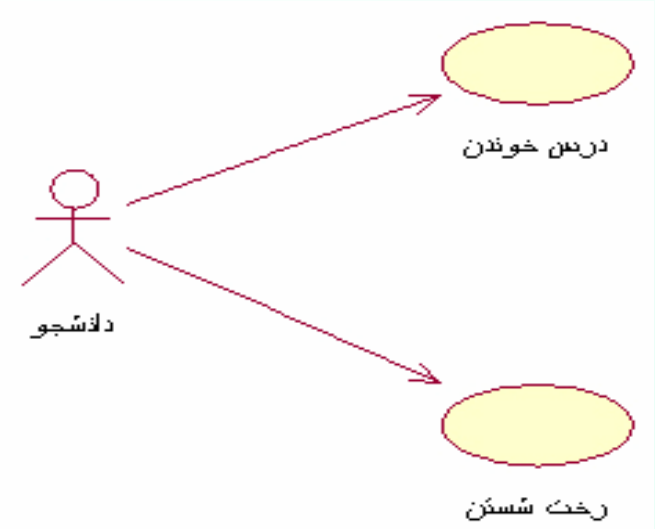
usecase diagram



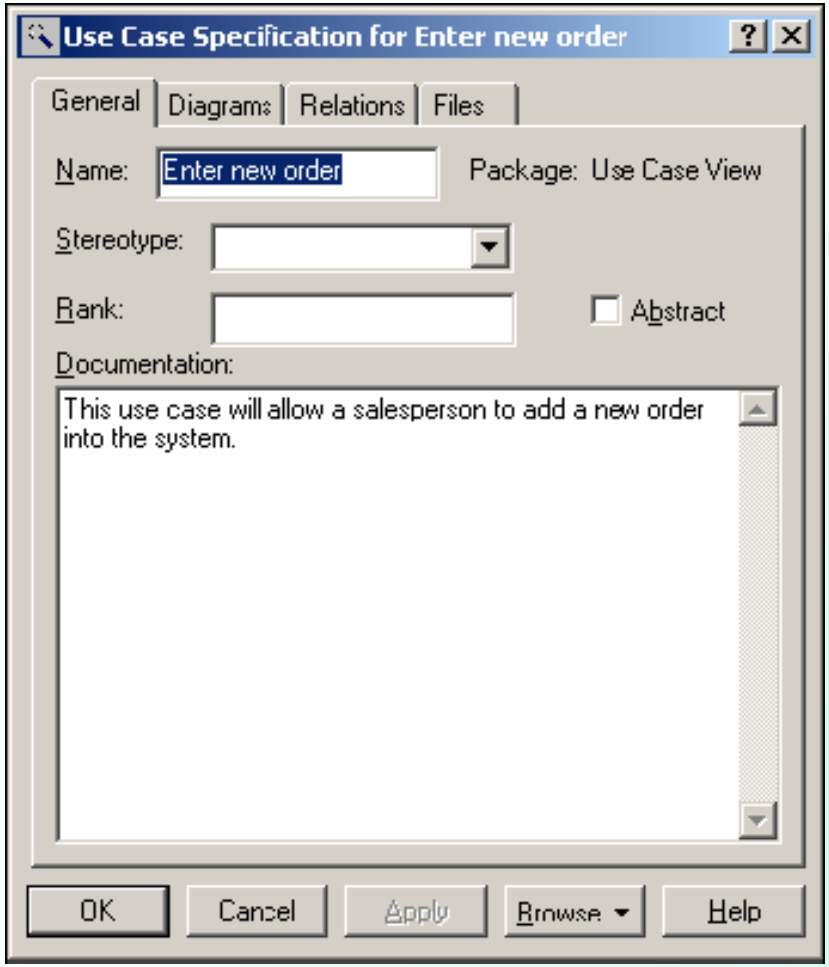
انواع ارتباطات:

وابستگی





usecase specification



دیاگرام کلاس ها (class diagram):

- این دیاگرام به شما کمک می کند تا نمای ساختاری یک سیستم را بصورت بصری (visual) در آورد.
- جزییات هر کلاس و ارتباطات بین آنها را نشان می دهد.
- پایه و اساس دیاگرام های اجزا و پیاده سازی می باشد.

در یک دیاگرام، کلاس با کلاس های با سه نوع متفاوت stereotype زیر سروکار داریم:

- boundary : اجزای لازم برای برقراری ارتباط سیستم با یک بازیگر را در خود دارند .
- control : این کلاس ها معمولا اشیا دیگر و رفتارهای تعبیه شده در یک usecase را کنترل می کنند.
- entity : این کلاس ها اطلاعاتی را که باید توسط سیستم ذخیره گردند را در خود نگهداری می کنند .

ابزارهاو مشخصات کلاس :

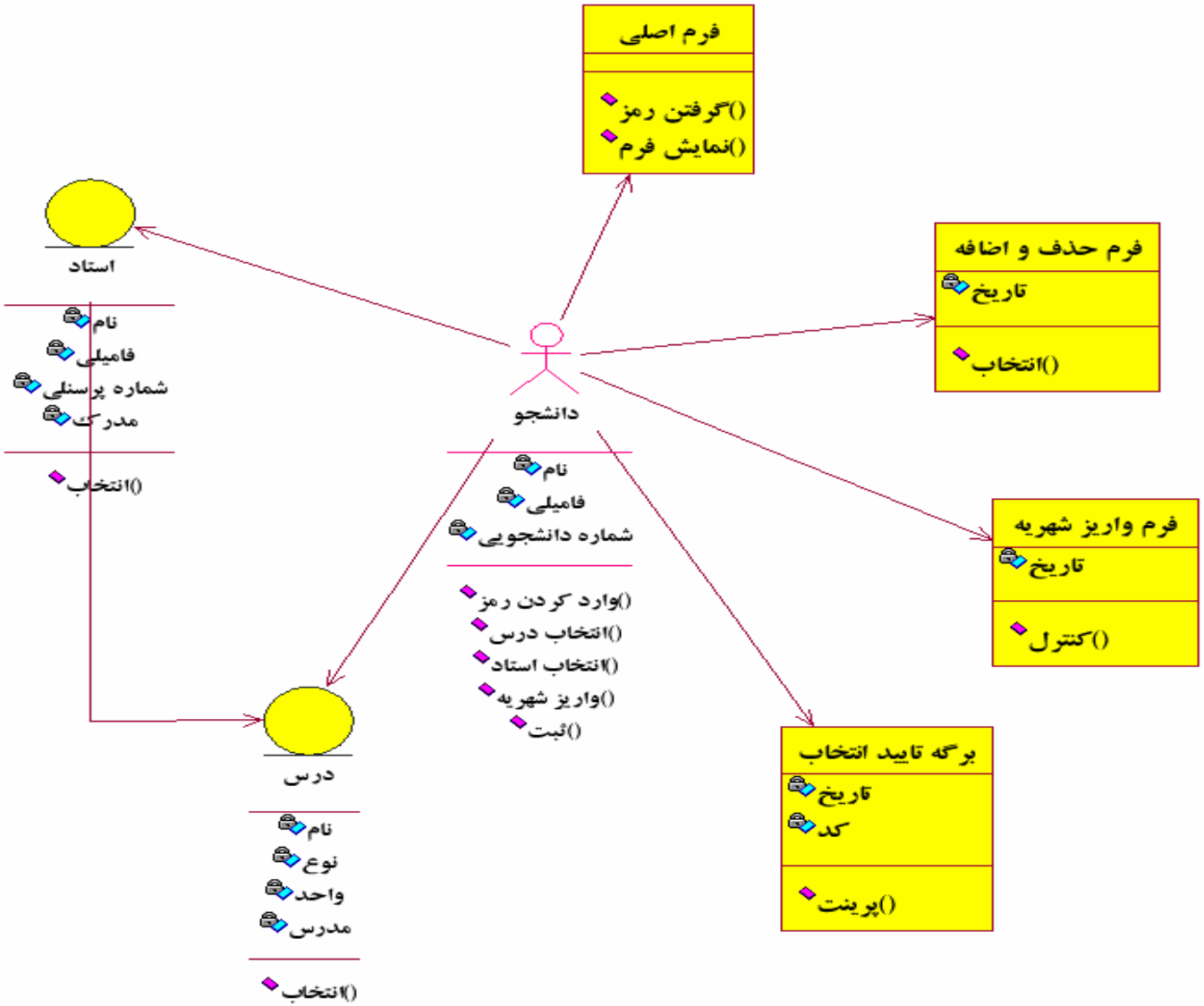
Public و

Protected و

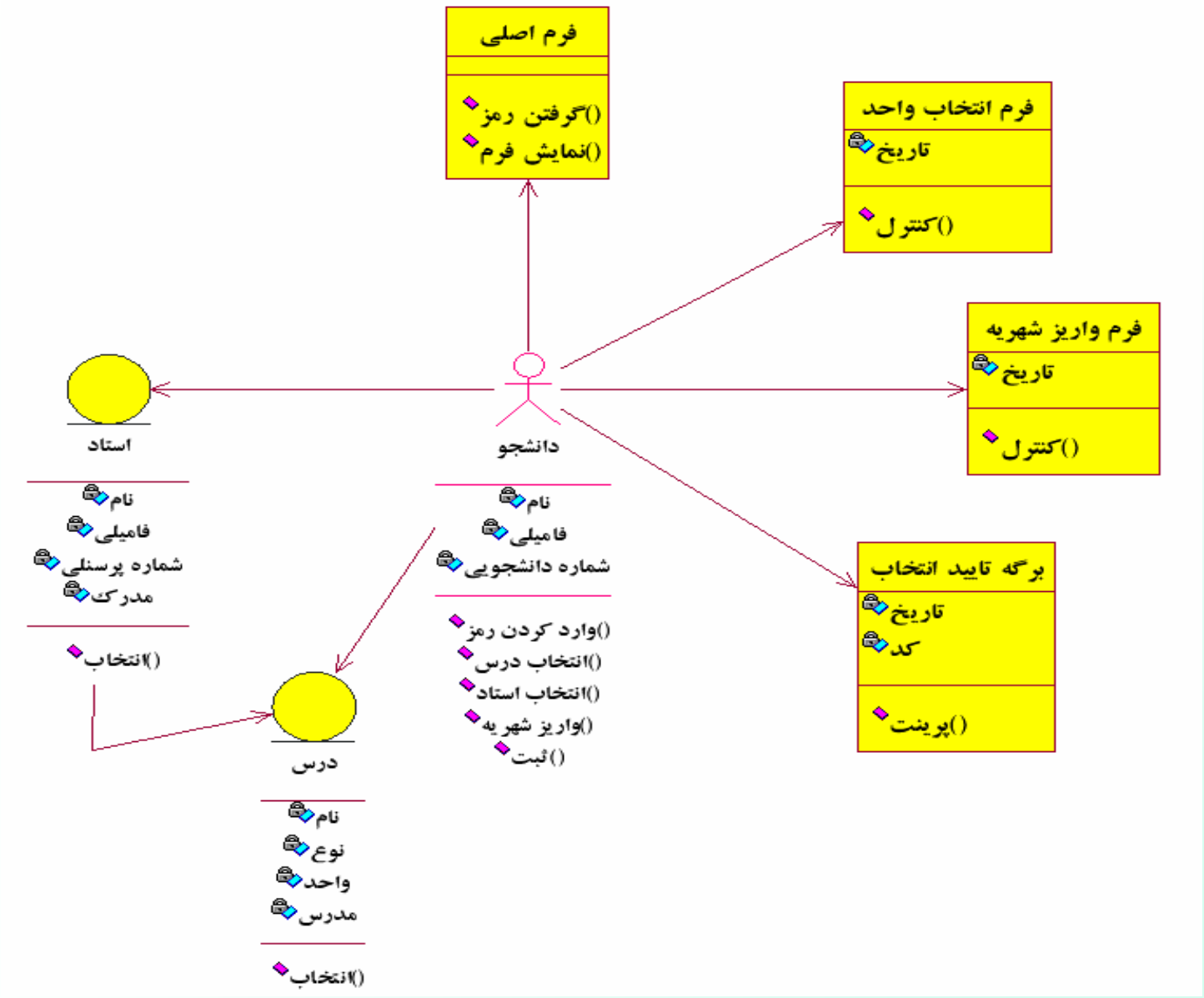
Private و

	برای نمایش کلاس		ارتباط بین ها کلاس ها یا association یا هر دوي آن ها
	Interface (رابط کاربری – همون فرم)		رابط بین interface و کلاس یا اجزا و interface

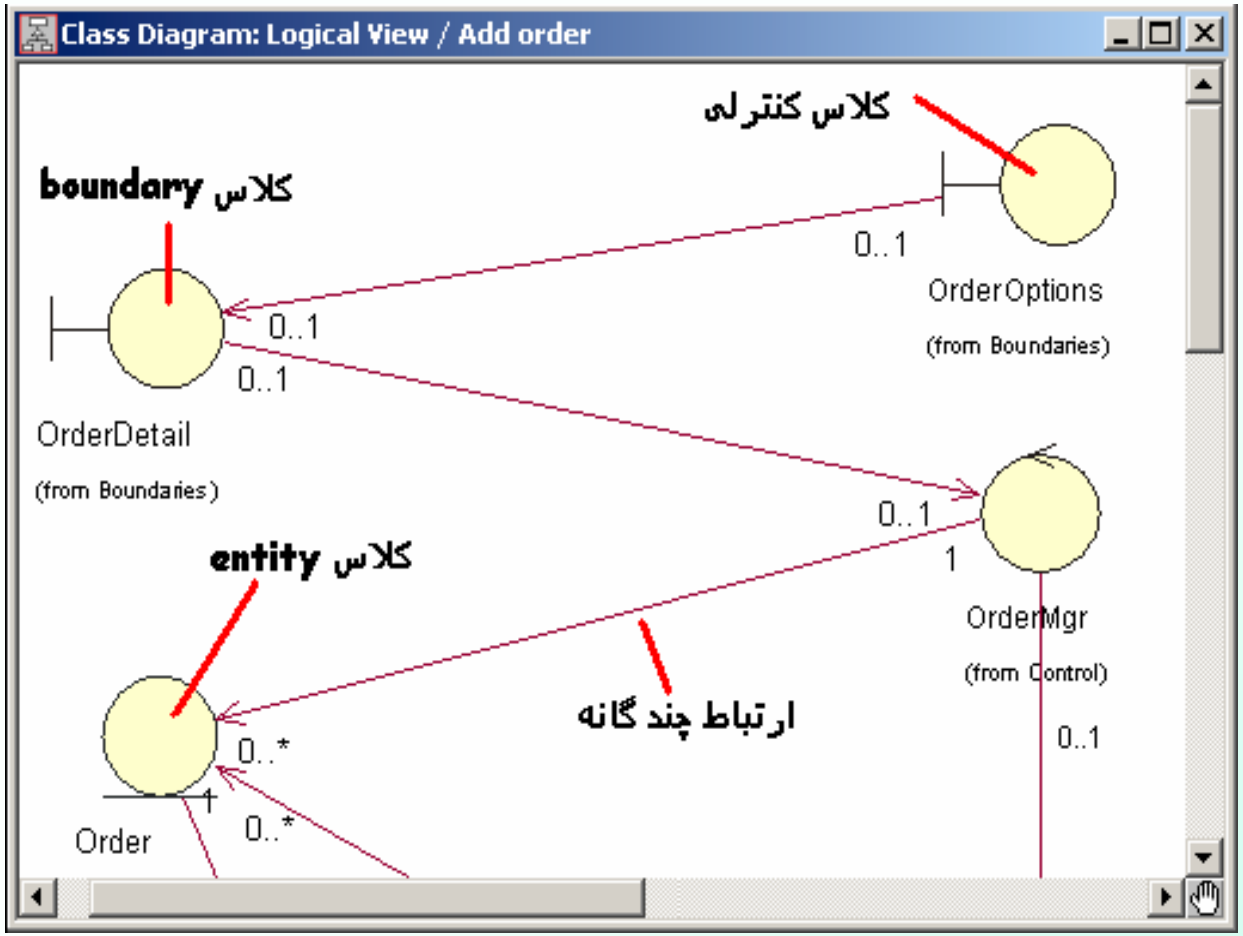
دیاگرام کلاس انتخاب واحد



دیاگرام کلاس حذف و اضافه



class diagram



دیاگرام توالی (Sequence Diagram)

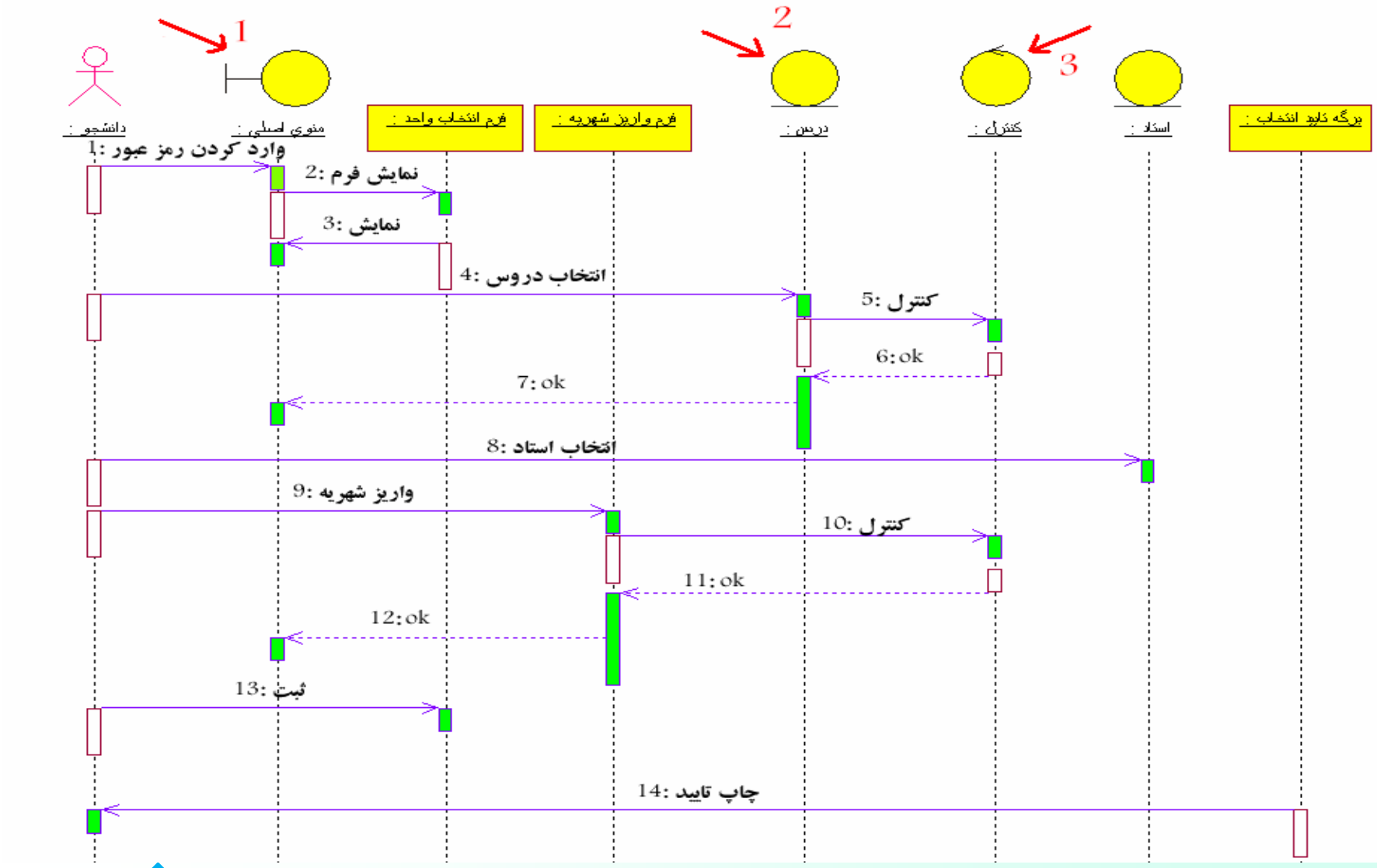
۱. مشخص کردن توالی رویداد ها در یک عمل خاص

۲. مورد استفاده در مراحل تحلیل و طراحی برای فهم نحوه ی عملکرد سیستم

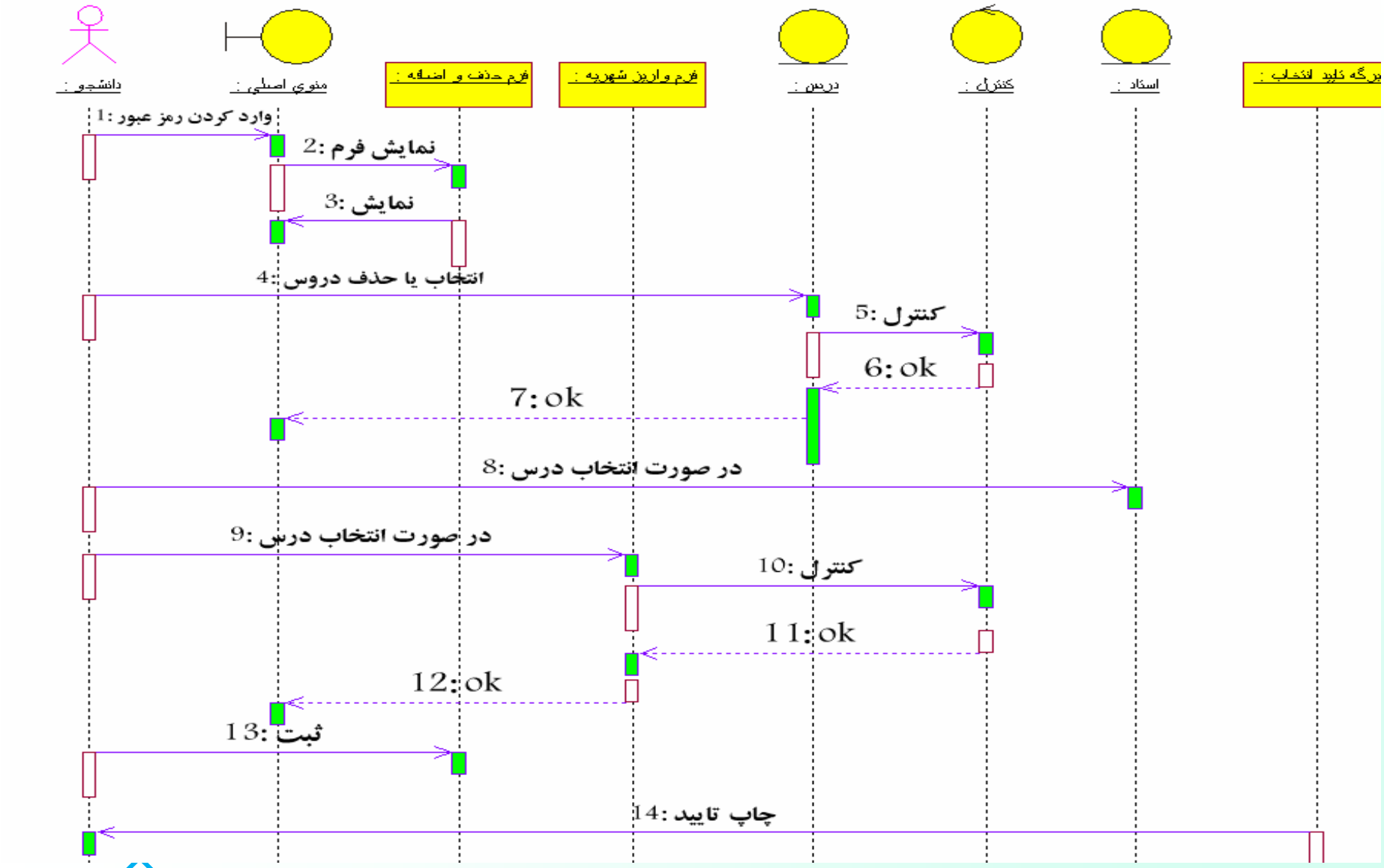
۳. مثال : مرخصی استحقاقی

پرکردن فرم ← تایید دانشکده ← پر کردن فرم ← پر کردن کارت مرخصی سالیانه

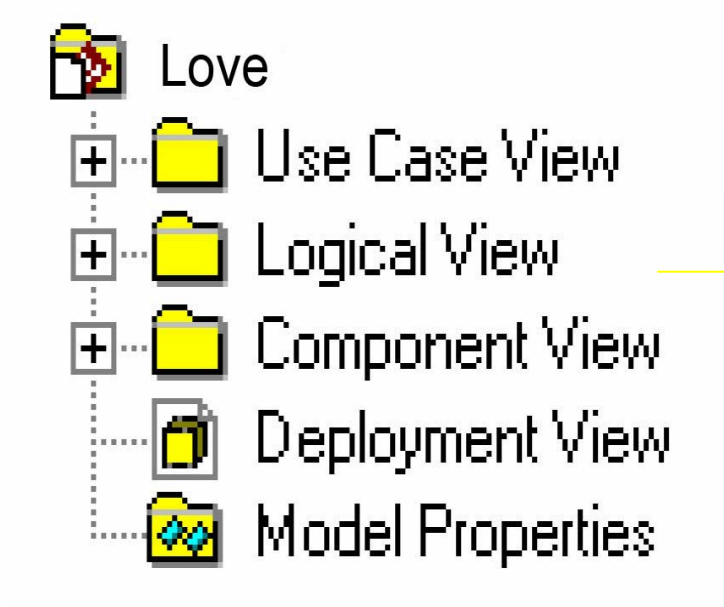
دیاگرام توالی انتخاب واحد



دیاگرام توالی حذف و اضافه



دیاگرام توالی در Rational Rose



دیاگرام همکاری (Collaboration Diagram)

مشخص کردن ارتباط بین اشیا

تاکید دیاگرام بر ارتباط بین اشیا

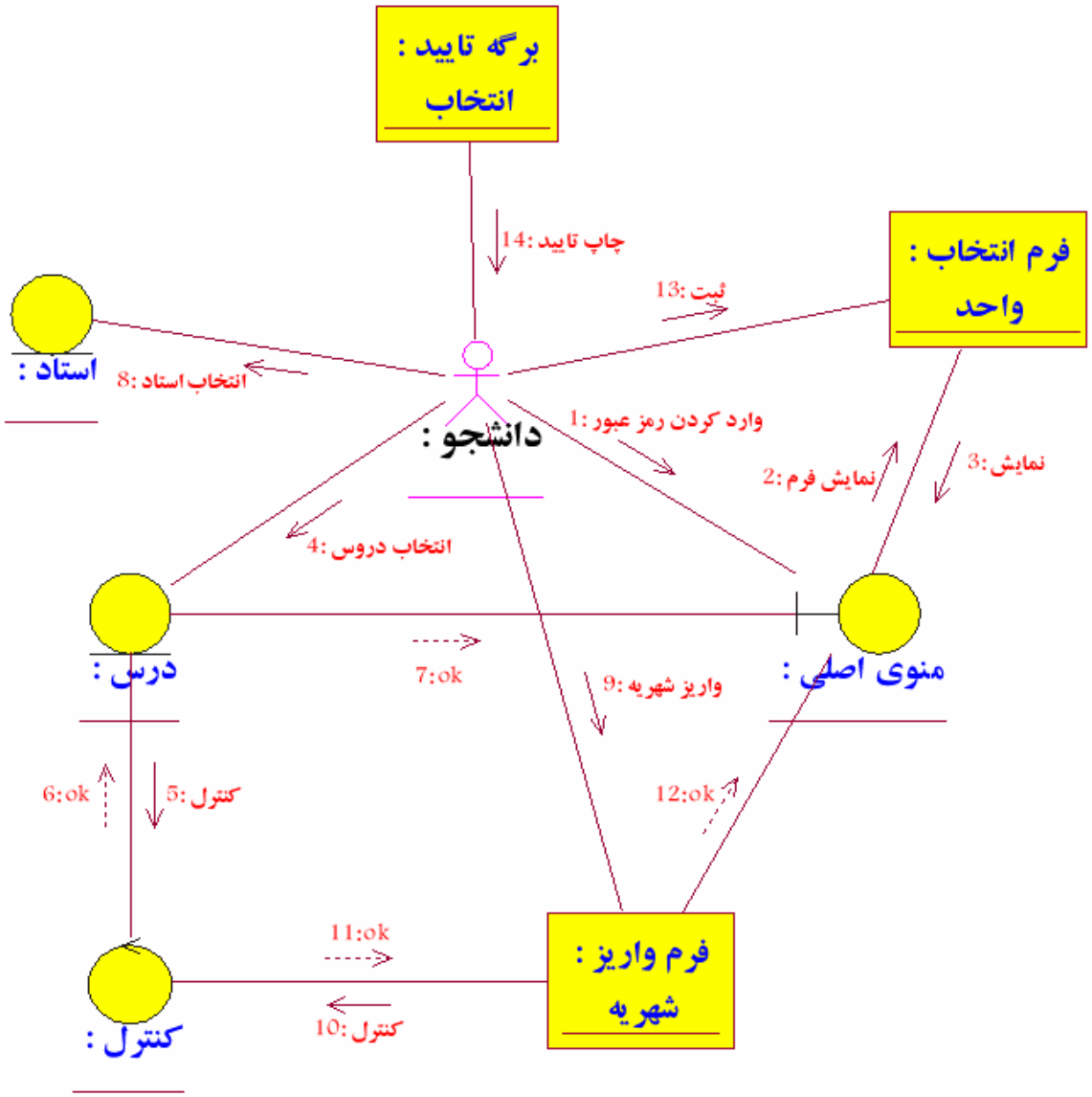
مورد استفاده در درک و فهم چگونگی رفتار سیستم و اتخاذ هرگونه تصمیم درباره آن

دیاگرام همکاری از روی دیاگرام توالی ساخته می شود.

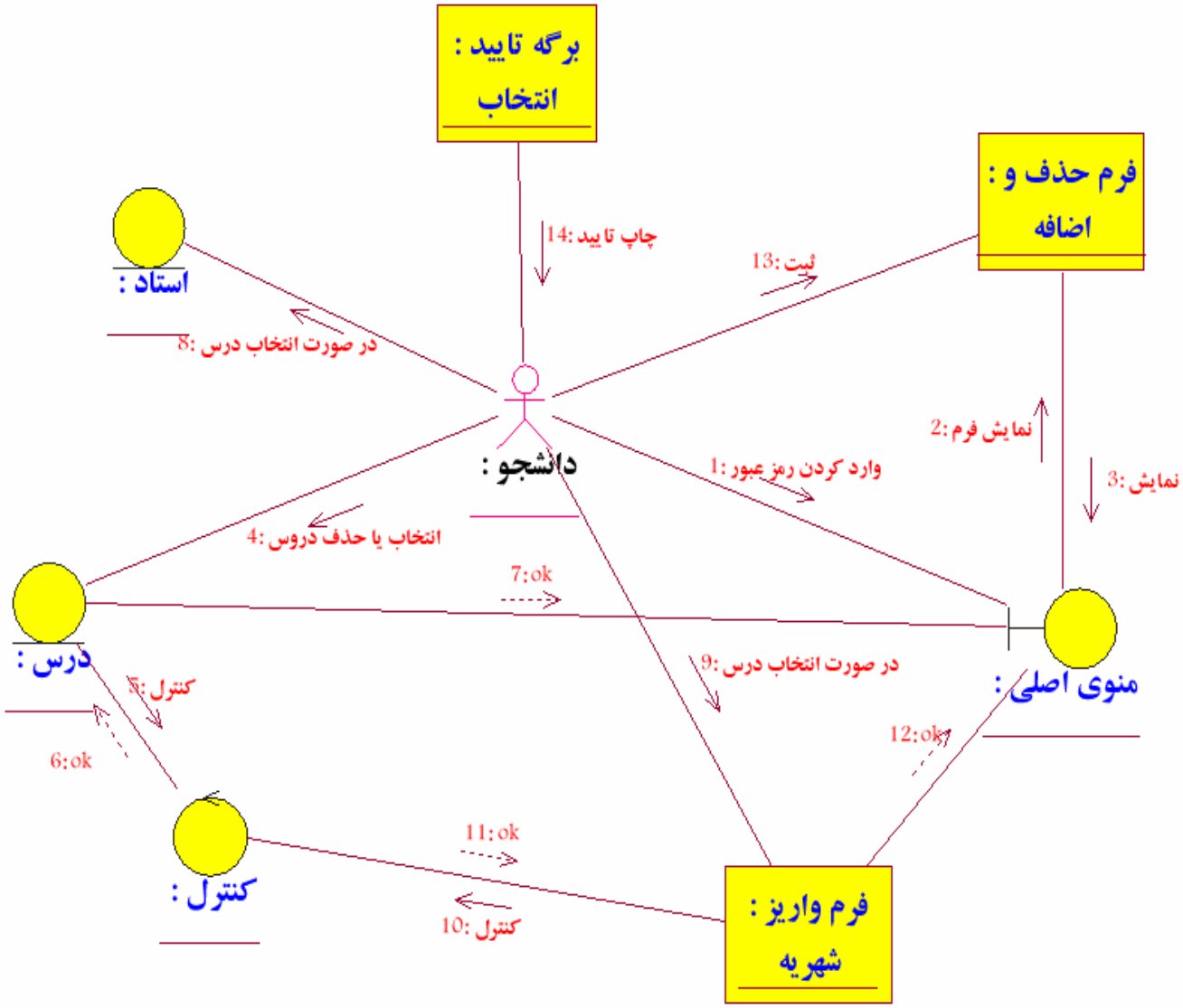
مثال :

شیء کارمند از طریق درخواست بازنشستگی با شیء بازنشستگی در ارتباط است . . .

دیاگرام همکاری (collaboration diagram)

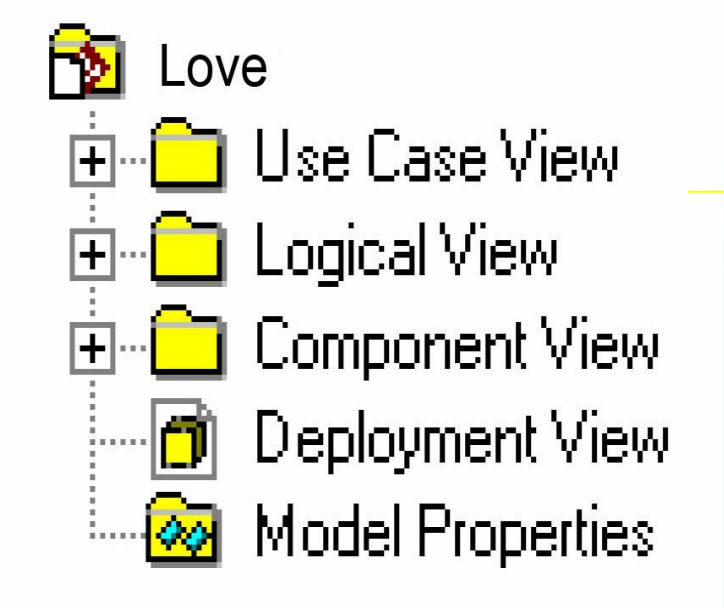


دیاگرام همکاری حذف و اضافه



Email: ComputerCollege_Fam@Yahoo.Com

دیاگرام همکاری در Rational Rose



دیاگرام (Statechart Diagram) حالت

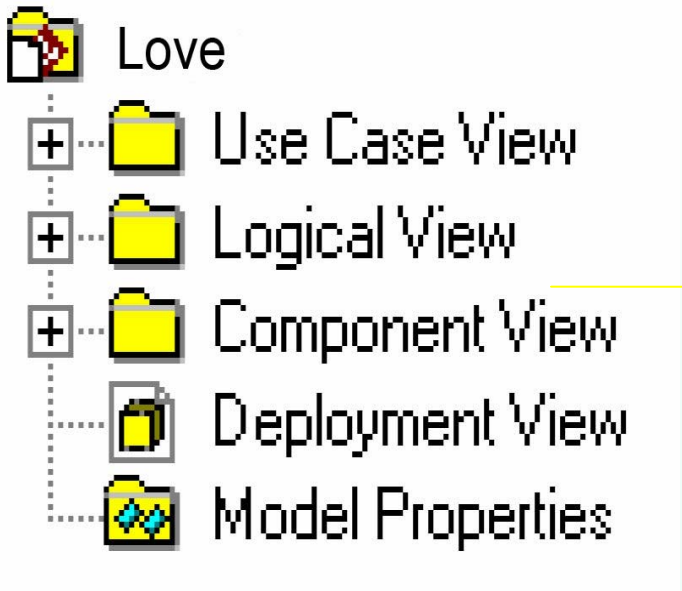
حالت های ممکن یک رویداد

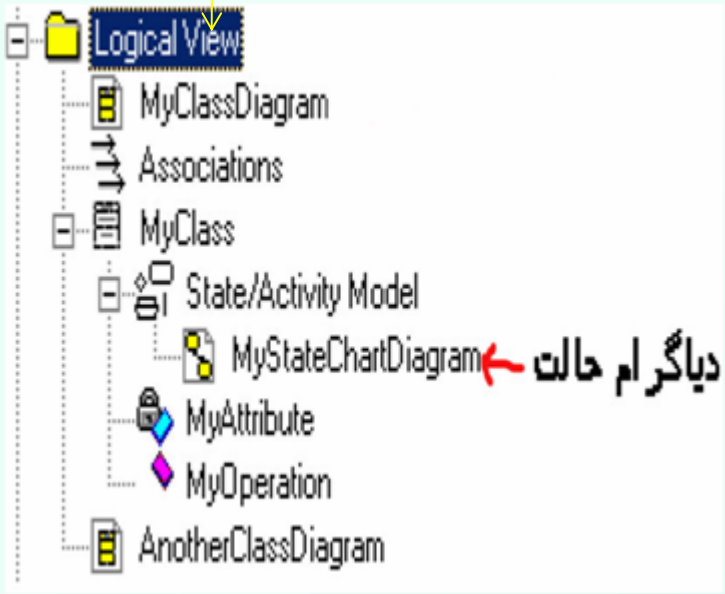
↓

رویداد هایی که موجب انتقال از یک حالت یا فعالیت به دیگری می شود

دیاگرام حالت مانند فلوچارت عمل می کند

دیاگرام حالت در Rational Rose





دیاگرام فعالیت (Activity Diagram)

شباهت زیادی با فلوچارت ها دارند.

مدل سازی فرایند های سیستم اطلاعاتی مستقل از اشیاء

فعالیت ها و روابط مابین آنها را در طول یک فرایند به تصویر می کشد.

کاربرد: توصیف رفتارهای موازی و چگونگی نشان دادن عکس العمل در مقابل وضعیت چندگانه

اجزای دیاگرام فعالیت :

Actions and Activities : کارهایی هستند که به دلیل خاصی انجام می شوند

– تفاوت Action و Activity

Object Node: اشیایی که توسط Action ها و Activity ها تغییر می یابند

Flows:

جریان های کنترلی (Control Flows): توالی اجرای فعالیت ها را نشان می دهد

– جریان های اشیاء (Object Flows) :نشان دهنده ی مسیرتغییراتی که بر روی اشیاء در طول اجرای فرایندها رخ می دهد

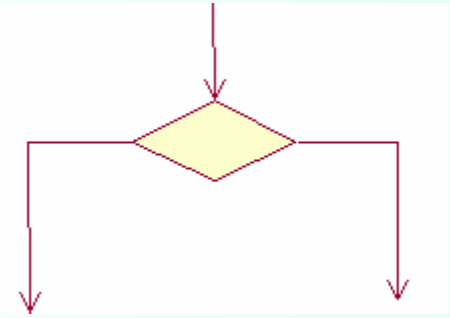
: Control Nodes

Initial Node: نقطه ی شروع فرایند

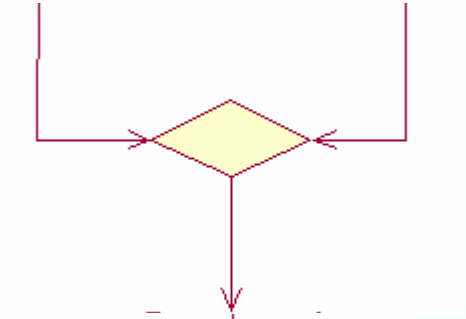
Final-Activity Node: نقطه ی پایان فرایند

Final-Flow Node : پایان یک Control Flow یا Object Flow خاص

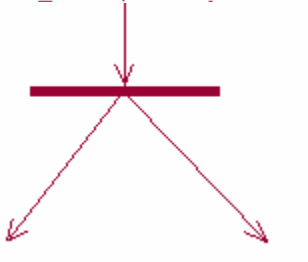
– Decision Node: دنبال کردن فعالیت ها، طبق مسیری که شرط گفته شده معین می کند



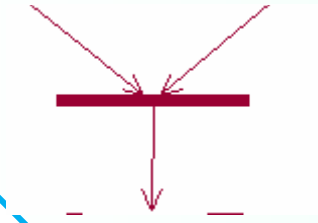
- Merge Node: نقطه ی شروع یک فعالیت یکسان بس از طی مسیر های متفاوت



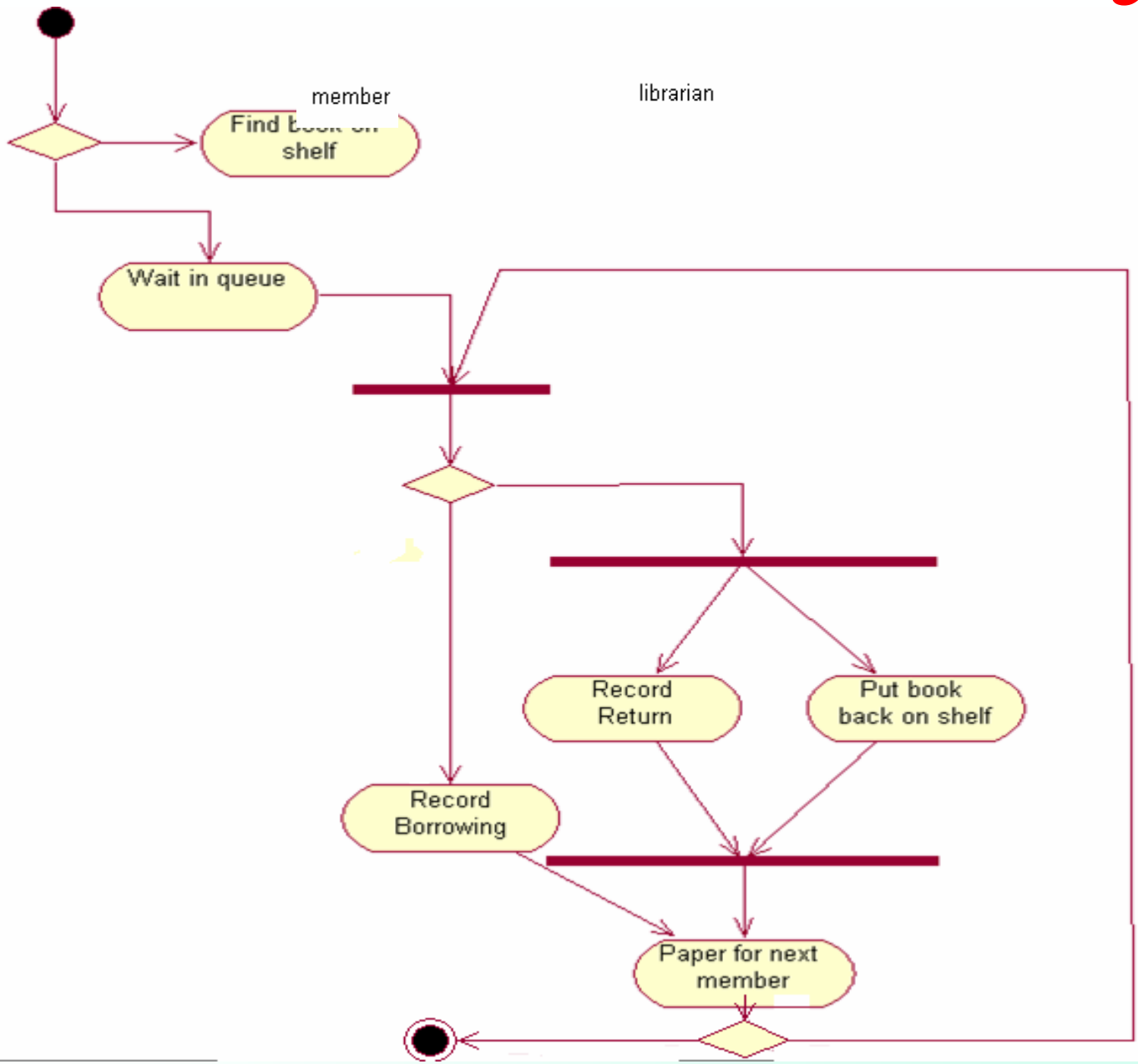
- Fork Node: نقطه ی شروع فعالیت همزمان



- Join Node: نقطه ی شروع فعالیت بعدی پس از چند فعالیت همزمان



معرفی محیط رشنال رز



دیاگرام اجزا (Component Diagram)

ساختار پیاده سازی سیستم را بیان می کند

یک جزء وابستگی زیادی با یک زیر سیستم دارد

نشان دهنده ی وابستگی بین قسمت های مختلف کد

سه گونه مختلف اجزاء:

- **کد منبع**: وابسته به هر جزئی که باید در زمان کامپایل در دسترس باشد

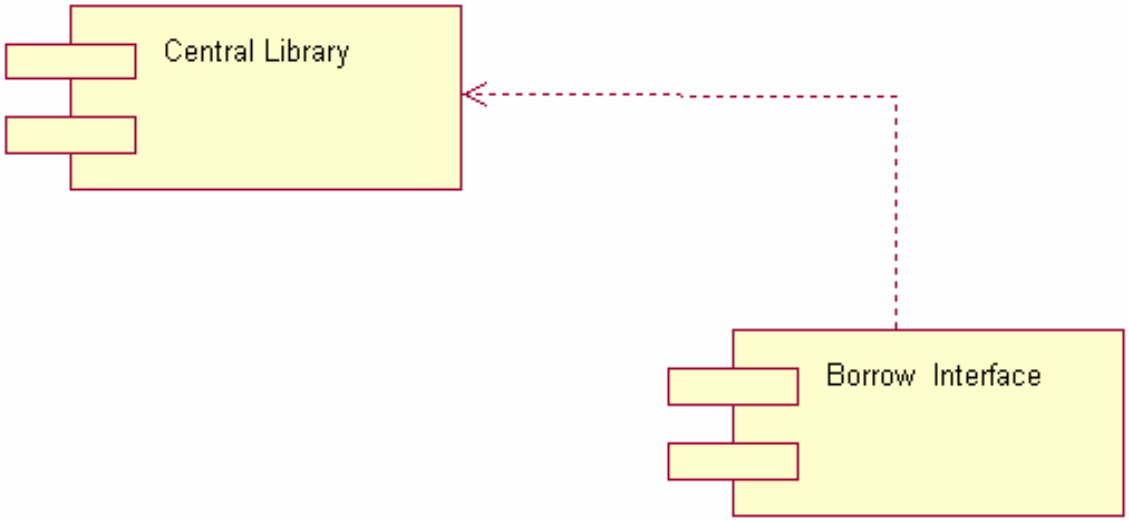
مثال :یک فایل شامل کد یک کلاس

- **کد باینری شیء**: مرتبط به هر شیئی است که به صورت یک برنامه ی قابل اجراست

مثال :یک کلاس کتابخانه

- **برنامه ی کاربردی قابل اجرا**: وابسته به دیگر برنامه های قابل اجرا برای فعل و انفعال با آن در زمان اجرا

مثال: مشتری یا سرویسی در برنامه ی کاربردی



دیاگرام اجزا در سیستم کتابخانه

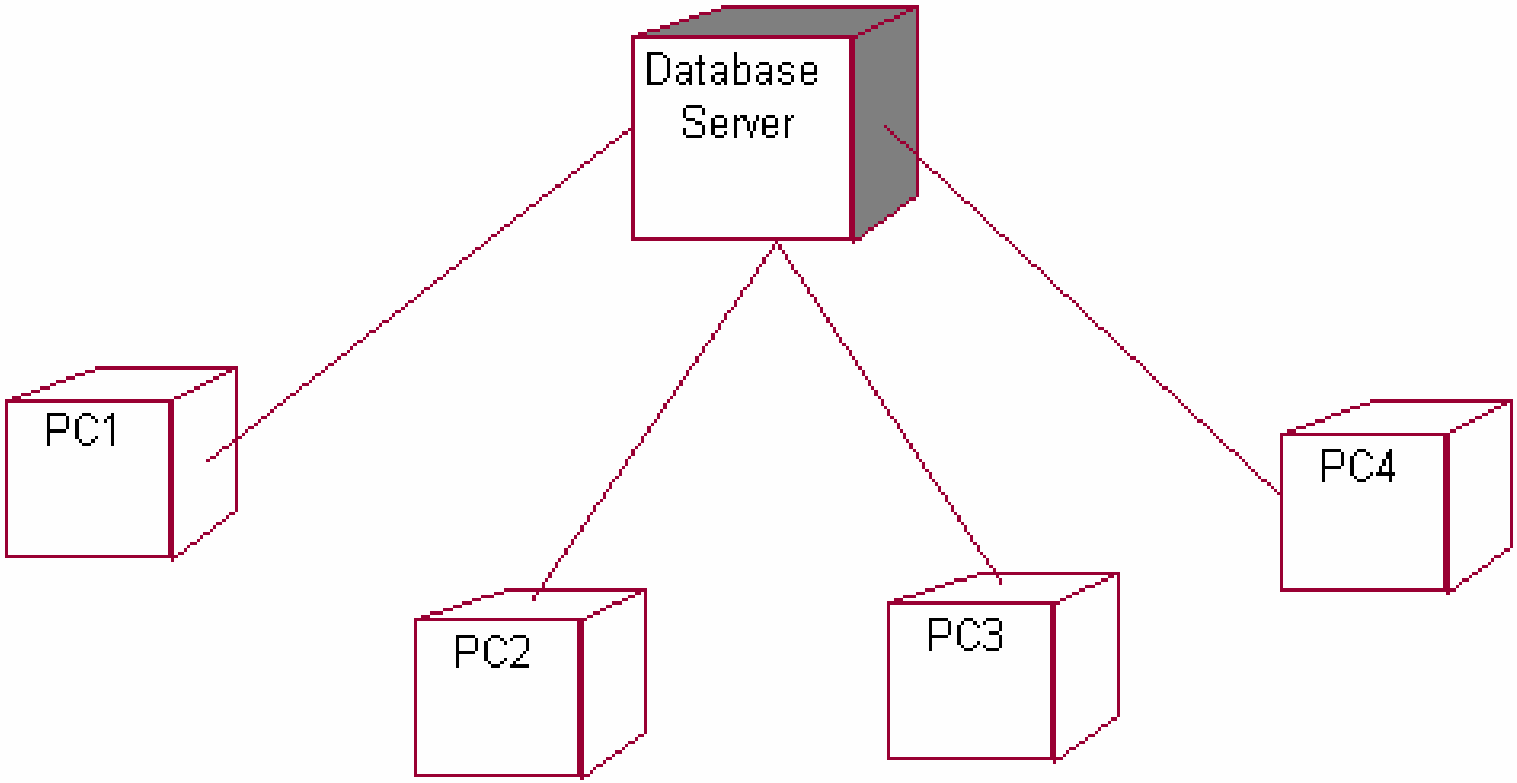
دیاگرام توسعه (Deployment Diagram)

نحوه ی گسترش یافتن سیستم را روی یک ساختار سخت افزاری خاص بیان می کند

بیان کننده ی پیوند های ارتباطی فیزیکی بین اجزای سخت افزاری

مفهوم نود ها به همراه ارتباطات بین آنها

هر مدل شامل فقط یک دیاگرام پیاده سازی است



دیاگرام توسعه برای کتابخانه