مهندسی نرم افزار به کمک کامپیوتر

دکتر رویا گلچای

چکیده تحلیلی تدریس استاد و مستندات مرتبط

بابک آشفته یزدی

بخش اول : مقدمه

1
- نقش دو گانه نرم افزار
- نرم افزار چیست؟
- نکاتی مهم در مورد نرم افزار
- عوامل فساد پذیری نرم افزار
- میراث نرم افزار
- تقسیم بندی نرم افزارها

2
- ویژه گی نرم افزارهای تحت وب
- تعریف مهندسی نرم افزار
- ساختار لایه بندی نرم افزار
- چارچوب فرآیند های نرم افزار
- فعالیتهای چارچوب – اصلی
- فعالیتهای پوششی یا چتری
- پیشنهاد آقای پولیا

بخش دوم مدل های فرآیند نرم افزار

3
- دیدگاه های عمومی مهندسی نرم افزار
- روش خطی
- روش خطی با جریان فرآیند تکراری
- روش دورانی یا تکاملی

4
- روش موازی
- روش خطی
- مشخص نمودن وظایف
- الگوهای فرآیند
- الگوهای مورد استفاده در فاز تولید نرم افزار
- الگوهای بهینه سازی و توسعه و ارزیابی کیفیت

5
- مدل های تجویزی
- مدل آبشاری
- مدل وی V

6
- مدل افزایشی
- مدل های تکاملی
- مدل نمونه سازی
- مدل مارپیچی ، حلزونی

7
- مدل هم روند
- مدل توسعه مبتنی بر اجزاء
- مدل شیوه های رسمی
- مدل روش جنبه گرا

8
- مدل فرآیندهای یکپارچه
- فازهای مدل فرآیند یکپارچه
- فرآیند تولید نرم افزار بصورت فردی

9
- فرآیند تولید نرم افزار بصورت تیمی

بخش سوم توسعه سریع نرم افزار

9
- مهندسی نرم افزار سریع
- ارزشیابی های مهم درمهندسی نرم افزار سریع
- مفهوم چابکی در تولید نرم افزار
- چابکی و تغییرات هزینه

10
- فرآیند چابکی
- اصول چابکی
- عوامل انسانی اصول چابکی
- مدل های استاندارد توسعه نرم افزار سریع
- برنامه نویسی اغراطی

11
- روش توسعه مبتنی بر همکاری
- فازهای روش توسعه مبتنی بر همکاری

12
- روش توسعه سیستمهای پویا
- اصول روش توسعه سیستمهای پویا

13
- روش سریع اسکرام

14
- روش توسعه کریستالی
- روش ویژه گی محور

15
- روش مدل سازی سریع
- نکات برجسته روش های توسعه سریع نرم افزار

بخش چهارم CASE Tools

16
- تاریخچه CASE
- مزایای CASE
- معایب CASE

17
- دسته بندی CASE
- ابزارها
- میزکار
- محیط های کاری

18
- اجزاء CASE
- مخزن اطلاعات
- ابزار طراحی نمودار
- ابزار تولید گزارش و تجزیه تحلیل
- برخی ابزارهای دیگر

19
- تکنولوژی های CASE
- دسته بندی CASE از دیدگاه تکنولوژی
- تقسیم بندی CASE از دیدگاه کاربردی
- تقسیم بندی CASE از دیدگاه فرآیند
- تقسیم بندی CASE از دیدگاه تجمیع
- چرخه حیات CASE Tools
- فاز خریداری CASE
- فاز همگن نمودن ابزار خریداری شده با شرکت
- فاز مقدماتی CASE
- فاز عملیاتی و تکاملی CASE
- بودجه بندی CASE
- منسوخ شدگی CASE

20
- شی گرائی
- محاسن شی گرائی
- ویژه گیهای شی گرائی

بخش پنجم UML

21
- تعریف UML و SSADM
- جدول مقایسه UML وSSADM

22

محاسن UML

تاریخچه UML

عناصر UML

نمودارهای ساختاری Structural Diagrams

نمودارهای رفتاری Behavior Diagrams

نمودارهای برهم کنش Intractional Diagrams

23

نمودارهای رفتاری Structural Diagrams

نمودار از دید کاربر Usecase Diagram

Usecase

Actor

انواع ارتباط بین Actor و Usecase

24

فرایند توسعه ساخت محصول Development Process

نمودار از دید کاربر Usecase Diagram

ارتباط Communication

ارتباط Include

ارتباط Extend

ارتباط Generalization

25

اجزاء نمودار Usecase Diagram

نمودار کلاس Class Diagram

انواع کلاس ها Stereotype

26

مثال انتقاب واحد

Usecase Diagram

نمودار توالی Sequence Diagram

27

مثال انتقاب واحد – نمودار همکاری Collaboration Diagram

نمودار کلاس Class Diagram

ارتباط تناظر Association

ارتباط وابستگی Dependency

28

ارتباط تعمیم Generalization

ارتباط تجمیع Aggregation

ارتباط ترکیب Composition

29

مثالهایی از دیاگرام کلاس

نمودار حالت State Diagram

نقش دوگانه نرم افزار

– نرم افزار امروزه تکنولوژی مهمی در عرصه جهانی میباشدو بخش جدایی ناپذیر از علوم مهندسی و تجارت میباشد و کلا در هر سیستمی ورود پیدا کرده است.

1. خود میتواند یک محصول باشد.(دارای پتانسیل محاسباتی سخت افزاری مانند تولید ، مدیریت ، تغییر ، نمایش اطلاعات ، ...)
2. وسیله ای که یک محصول را به ما تحویل میدهد (سیستم عامل ، نرم افزارهایی که نرم افزار دیگری را کنترل و یا ایجاد مینماید)

نرم افزار چیست؟

Instruction
Data Structures
Documentation

1. دستورالعمل هایی (برنامه هایی کامپیوتری) که اجرای آن منجر به خروجی مورد نظر (فعالیت ها ، عملیات ، کارایی) میگردد.
2. ساختمان داده ای که به کمک آن داده ها بگونه مناسبی دستکاری میگردد.
3. مستنداتی که عملکرد و استفاده از برنامه را توصیف میکند.

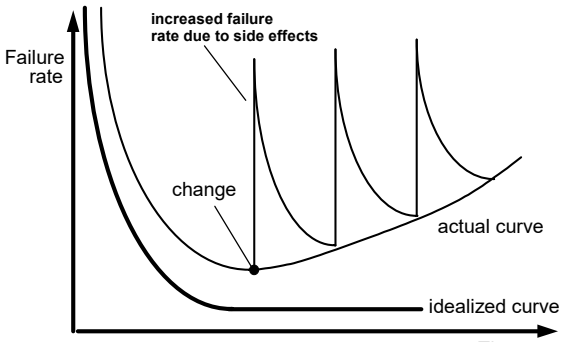
– این مجموعه منجر به تولید نرم افزار به دو منظور 1 مشتری خاص 2 بازار عمومی میگردد.

چند نکته

1. نرم افزار مهندسی شده و توسعه داده میشود ولی ساخته نمیشود.
   (نرم افزار یک واهد منطقی و غیر قابل لمس است)
2. نگهداری از نرم افزار خیلی پیچیده است و معمولا سفارشی ساخته میشود و باعث میشود که همراه با پیچیده گیهای زیادی باشد.

(در مورد سخت افزار ، وقتی یک قطعه بصورت استاندارد ساخته میشود ، این قطعه به دفعات توسط مهندسین الکترونیک و مکانیک به کار گرفته میشود ولی نرم افزار ها معمولا بصورت سفارشی ساخته میشوند و حالت ایده آل بدین صورت خواهد بود که مولفه های نرم فزاری هم طوری ساخته شوند که در برنامه های مختلف قابل استفاده باشند. بطور مثال یک رابط گرافیکی کاربر که شامل یک سری منو ، پنجره و امکاناتی از این قبیل ، به گونه ای ساخته میشود که بتوان آن را در برنامه های مختلف مورد استفاده قرار داد و نیاز به کد کردن مجدد وجود نداشته باشد.)

3. نرم افزار فرسوده نمیشود و در واقع فاسد میشود.
   (یک قطعه سخت افزاری در طول زمان به دلایل مختلفی از جمله حرارت ، رطوبت و غیره فرسوده میشود و در نهایت از بین میرودولی نرم افزار جنبه فرسودگی به این شکل ندارد و بیشتر جنبه فساد پذیری آن مطرح میباشد که منبعث از دوعلت زیر میباشد)



منحنی فرسودگی و فساد نرم افزار در طول زمان

عوامل فساد پذیری نرم افزار

1. تغییر خواسته کاربران (تغییر و اصلاح نیاز ذاتی نرم افزار میباشد)
2. تغییر تکنولوژی (نرم افزار باید با فناوری های جدید سازگاری داشته باشد)

– ایجاد تغییرات مداوم باعث پیچیده و سخت شدن بروزرسانی نرم افزار میگردد.

میراث نرم افزار

بدین مفهوم که نرم افزار ازبین نمیرود.

آگر فرض را بر نرم افزاری قدیمی بگذاریم که در یک سیستم کار میکند ولی بروز نمیباشد.

1. – تا زمانیکه نرم افزار کار میکند و خروجی مناسبی که برای آن طراحی شده است را تولید میکند را نباید مورد دستکاری قرار داد.
2. – ارزیابی نرم افزار (۱– برطرف کردن نیاز کاربر ۲–کارکرد بدون مشکل ۳– سازگی اصلاح ، تغییرات و توسعه)
   – نرم افزارهای قدیمی عمدتا قابلیت توسعه را ندارند و دارای کدهای ترکیبی و مستندات ضعیف میباشند.
3. – انجام مهندسی مجدد بعلت ویژگی فساد پذیری ذاتی نرم افزار و استفاده از متدولوژی جدید برای تکامل نرم افزار
   – تغییر سیستم نرم افزار قدیمی بطوریکه از آن بتوان نرم افزار جدیدی ساخت.

تقسیم بندی نرم افزارها

1. سیستمی – نرم افزاری که سرویس برای بقیه نرم افزار ها را فراهم میکند.
2. کاربردی – برنامه های مستقل برای حل نیازهای شغلی مختلف
3. مهندسی یا علمی – یک سری الگوریتم که بر روی اعداد اجرا شده و خروجی تولید میکند. (بیولوژی ، ستاره شناسی ، ....)
4. جاسازی شده – نرم افزاری کوچک در داخل یک سیستم سخت افزاری
5. خط تولید – نرم افزارهای مخصوص خط تولید
6. تحت وب – نرم افزارهای تحت وب
7. هوش مصنوعی – از الگوریتم های غیر عددی برای حل مسائل استفاده میکند.

New Categories

- Open world computing
- Ubiquitous computing
- Netsourcing
- Data mining
- Grid computing
- Cognitive machines
- Software for nanotechnologies

ویژه گیهای نرم افزارهای تحت وب

- **Network intensiveness.** – امکان سرویس به اجتماع بزرگی از شبکه کاربران
- **Concurrency.** – همزمانی کارکرد تعداد زیادی از کاربران
- **Unpredictable load.** – پشتیبانی از بار افزایشی پیش بینی نشده کاربران
- **Performance.** – در صورت بالا بودن زمان انتظار سرویس سمت سرور ، کاربر میتواند فعالیت سمت کابر را انجام دهد
- **Availability** – دسترسی همیشگی کاربران ۱۰۰٪ بصورت ۲۴/۷/۳۶۵
- **Data Driven** – سهولت استفاده از Hypermedia (Text,Graphic,Audio,Video,...)
- **Content sensitive** – کیفیت و زیبایی در تولید محتوا
- **Continuous evolution** – پیوستگی تکامل در نرم افزارهای تحت وب
- **Immediacy** – سریع بودن روند تولید نرم افزار برای نمایش و عرضه به بازار
- **Security** – ایجاد امنیت بعلت دسترسی کاربران از طریق شبکه وب
- **Aesthetics** – زیبایی و راحتی در تولید محصول

تعریف مهندسی نرم افزار

– در تعریف مهندسی نرم افزار باید توجه داشته باشیم که برنامه نویسی بخشی از مهندسی نرم افزار می باشد.

از دیدگاه IEEE به کارگیری روشی سیستماتیک ، منظم و قابل اندازه گیری برای توسعه ، تولید و عملیاتی نمودن نرم افزار میباشد.

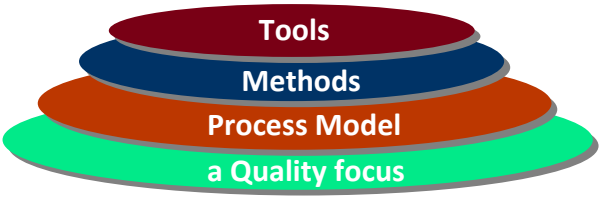
(به کارگیری اصول مهندسی در تولید نرم افزار)

– شرکتهای کامپیوتری با استفاده از مجموعه ای از تکنیکها و قواعد معتبر مهندسی ، یک نرم افزار قابل اطمینان و مقرون به صرفه را تولید مینمایند.

ساختار لایه بندی مهندسی نرم افزار

Software Engimeering Layer

– مهندسی نرم افزار ساختار لایه لایه دارد و زیرساختی از فعالیتهای مرتبط به هم میباشد.



① کیفیت Quality
– مانند هر سیستم دیگری مبنا بر کیفیت میباشد.

② فرآیند Process Model
– فرآیند روش ها و ابزار ها به هم متصل مینماید.
– کار سازماندهی شده و فعالیت ها کنترل میگردد.
– عدم بهم ریختگی در تولید نرم افزار . نظم و ترتیب
– پایه و بیس مهندسی نرم افزار با فرآیند تعریف میشود و شامل همه کارهایی است که باید انجام شود تا یک نرم افزار تولید گردد . فرآیند مشخص کننده روش های تولید نرم افزار میباشد.

③ متدها Method
– چگونگی ساخت نرم افزار بصورت تکنیکی را بیان میکند (۱- تحلیل نیازمندی ۲- طراحی ۳-مدل سازی ۴- ساخت ۵- نگهداری و پشتیبانی)

④ ابزارها Tools
– بصورت اتوماتیک برای تولید فرآیند و متد از ابزارها استفاده میکنیم و در حقیقت سیستمی برای حمایت از توسعه نرم افزار میباشد.

چارچوب های فرآیند نرم افزار

A Process Framework

① Framework activities فعالیت های چارچوب (اصلی)

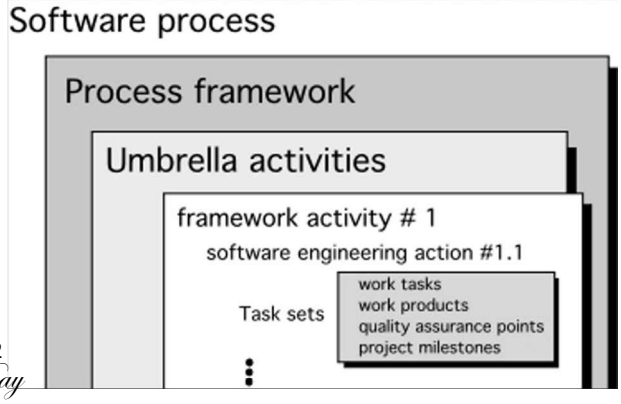
① Communication ارتباط ، تحلیل
– صحبت با مسئولین پروژه جهت مشخص کردن نیازمندی ها ، اهداف و درک مسئله

② Planning طراحی
– برنامه ریزی و تعیین نقشه مسیر (مشخص نمودن طرح پروژه ، منابع موجود ،زمانبندی، توصیف کارها و ریسک ها)

③ Modeling مدل سازی
۱-تحلیل و شناخت بهتر نیازمندیهای نرم افزار ۲- ایجاد یک طرح مناسب برای پوشش نیازمندیهای شناخته شده

④ Construction تولید
۱-کدنویسی نرم افزار ۲- آزمایش نرم افزار

⑤ Deployment استقرار
– نصب و استقرار نرم افزار، نرم افزار تحویل مشتری شده و ارزیابی گردیده و بازخورد آن مشخص میگردد.



② Umbrella Activities فعالیت های پوششی یا چتری

– برای پشتیبانی و کامل نمودن فعالیتهای چارچوب استفاده میشود.

– مدیریت ریسک – تضمین کیفیت – مدیریت پیکره بندی نرم افزار – مدیریت استفاده مجدد – مدیریت پروژه نرم افزار و .....

Process Framework

پیشنهاد آقای پولیا
Polya suggests

– در این دیدگاه هر موقع که نیاز به ایجاد نرم افزار باشد چهار فعالیت بهتر است انجام گیرد.

1 – درک مسئله
2 – مدل نمودن و طراحی
3 – کدنویسی
4 – ارزیابی و آزمایش

طرح مسئله:
– پیدا نمودن مسئول درست و انتظارات او از برنامه چیست ؟ و چه مجهول هایی وجود دارد ؟
– آیا امکان تفکیک مسئله به قطعات کوچکتر وجود دارد ؟ و چگونه میشود این مسئله را مدل نمود ؟

– آیا قبلا مسئله مشابهی وجود داشته است ؟ چگونه مدل شده است؟ آیا امکان استفاده مجدد وجود دارد؟

– ارزیابی و آزمایش از طریق بازخورد کاربر و درصورت نیاز بازبینی مسئله و تغییرات در کد

دیدگاه های عمومی مرتبط با مهندسی نرم افزار

– در واقع همیشه یک سری شعار در طرح مسئله وجود دارد (باورهای اشتباه) و یک سری هم مطالبی وجود دارد که در واقع واقعیت میباشد و از دیدگاه کاربران و مدیران و دیدگاه های دیگر این مسائل قابل بررسی میباشد.

از دیدگاه مدیریتی :

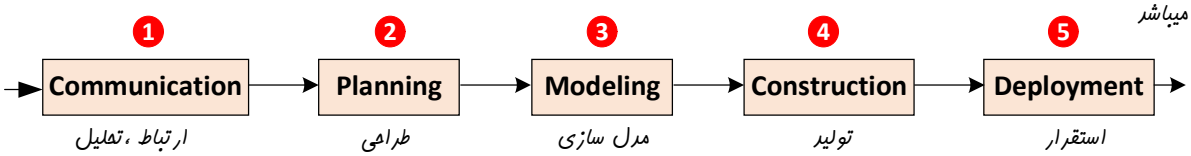
1 – یکی از باورهای غلطی که ممکن است مدیران داشته باشند اینست که اگر یک کتاب و قاعده کلی برای تولید نرم افزار وجود داشته باشد نیاز اطلاعاتی Developer ها برطرف میگردد و نرم افزار با کیفیتی تولید خواهد شد در این مورد باید گفت که
۱– آیا اساسا چنین کتاب استانداردی وجود دارد ؟
۲– از کجا مشخص میگردد که بروز میباشد و با نیازهای در حال تغییر منطبق است ؟
۳– در صورتیکه با نیازهای روز هماهنگ باشد چه کسی آن را میسنجد و اندازه گیری مینماید؟

2 – یکی از مشکلاتی که در بحران نرم افزار وجود دارد اینست که عمدتا نرم افزارها با تاخیر تولید میگردند یک باور غلط دیگر اینست که اگر تولید نرم افزار به تاخیر میافتد این مشکل را با جذب نیروی متخصص جبران نمائیم. لذا با توجه به اینکه ممکن است افراد مناسبی بکار گرفته نشوند و ضمن اینکه با فرض متخصص بودن چون از ابتدای پروژه در جریان کار نبوده اند و نیاز به زمان دارند تا به درک صحیحی از مسئله و طراحی آن پی ببرند که همین فرایند زمان بر خواهد بود. و این امکان وجود دارد که منجر به کوتاه شدن زمان پروژه نگردد و حتی برعکس زمان افزایش یابد.

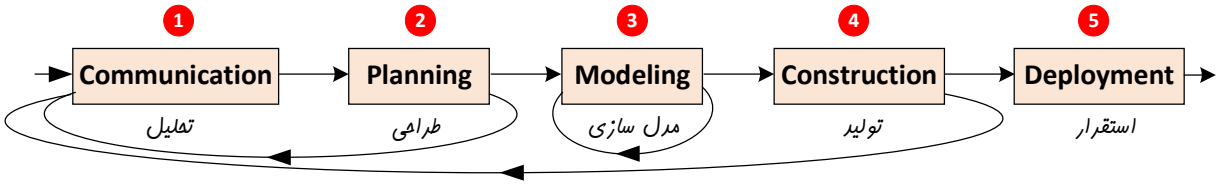
3 – برون سپاری پروژه نرم افزاری نیز میتواند باور غلطی باشد . از این لحاظ که باید برای برون سپاری نیازمند تسلط کامل در مدیریت و کنترل پروژه نرم افزاری میباشد.

روش خطی 1
Linear Process Flow

– ترتیب اجرای فرآیند فعالیت های چهارچوب میتواند متفاوت باشد
– ساده ترین مدل اجرایی میباشد

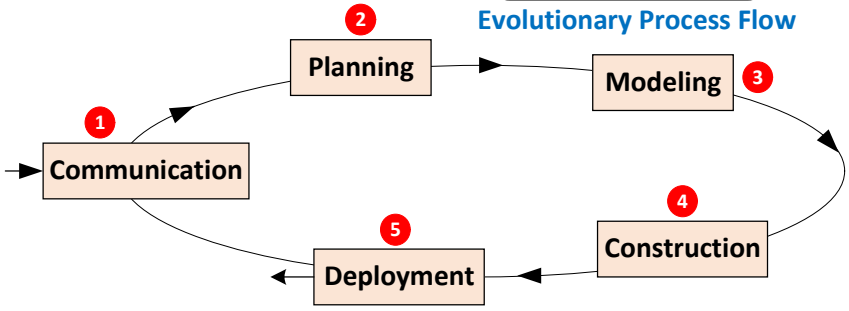


روش خطی با جریان فرایند تکراری 2
Intractive Process Flow



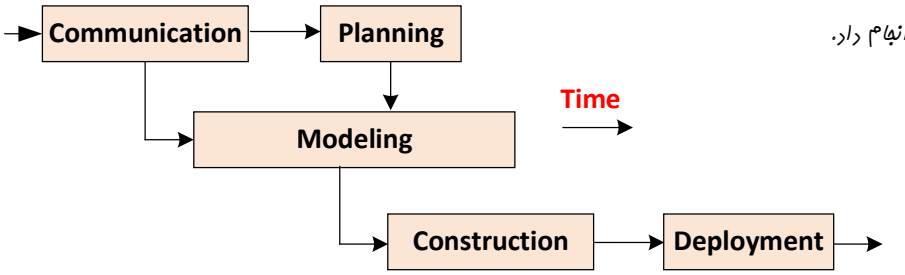
– در این مدل یک یا چند فعالیت امکان تکرار قبل از رفتن به مرحله بعدی را دارند.
– بطور مثال میتوان بعد از مرحله ارتبط و بعد از فعالیت طراحی مجددا جهت مطمئن شدن از خواسته های مشتری مجددا با او ارتباط گرفت

روش دورانی یا تکاملی 3
Evolutionary Process Flow



– بدین مفهوم که پس از یکبار انجام فعالیتها و تولید اولین نسخه برنامه ، مجددا جهت کامل شدن نرم افزار برنرگیه راهکار اهتیلاعخ ،و نسخه بعدی نرم افزار تولید میگردد و در صورت نیاز همین رویه ادامه پیدا میکند.

روش موازی
Parallel Process Flow



– بدین مفهوم که چند فعالیت را میتوان بطور موازی با هم انجام داد.

بطور مثال در زمانیکه قسمتی از یک نرم افزار تولید میشود ، نیازمندیهای قسمتی دیگر از نرم افزار تحلیل میگردد.

مشخص نمودن وظایف
Identifying a Task Set

– باید مشخص شود در چارچوب فعالیت های نرم افزاری چه کارهایی (وظایفی) Task باید انجام گیرد تا منجر به حل مسئله بصورت مهندسی گردد.

1. تهیه لیستی از فعالیت هایی که باید انجام پذیرد.
2. تهیه لیستی از محصولاتی که باید تولید شود.
3. تهیه لیستی از فیلترهای تضمین کیفیتی که باید بکار بسته شوند.

– این درست است که ما یک الگوی کاملی برای فعالیتها تهیه نموده ایم ولی با توجه به شرایط مختلف ممکن است Task Set ها تغییر کنند.

– بطور مثال فرض میکنیم که میخواهیم نیازمندیهای یک مجموعه را تهیه نمائیم . اگر این مجموعه خیلی کوچک باشد ما ابتدا مسئولین پروژه را لیست میکنیم ، سپس لیستی از نیازمندیهای آنان را مشخص نموده و پس اولویت بندی کار را شروع مینمائیم.

– ولی اگر نیازمندیها را برای یک پروژه بزرگ و پیچیده انجام دهیم بعد از لیست کردن مسئولین پروژه اولویت های نیازمندیها زمانبندی شده و سپس مجددا جهت تایید مسولین پروژه با آنها جلسه گذاشته شده و در نهایت بر اساس اولویت نیازمندیها مشخص میگردد در نسخه اول و نسخه های بعدی نرم افزار، کدام اولویت ها مد نظر قرار خواهد گرفت . ضمنا ریسکها نیز باید مورد بررسی قرار گیرد.

– در واقع در هردو مثال بالا نیازمندیهای یک سازمان مورد بررسی قرار گرفت ولی با توجه به حجم پروژه و پیچیده گیهای آن ما TaskSet های متفاوتی خواهیم داشت.

الگوی های فرآیند
Process Patterns

– جهت ساده تر شدن فرایند از یک سری الگو استفاده میشود

– الگوی فرآیند شامل یک سری از Activity , Task ,Action,Product هایی است که همه با هم مرتبط هستند

– اگر یک راه حل اثبات شده ای برای حل مسائل وجود داشته باشد و بتوان از آن استفاده نمود کمک بزرگی خواهد بود

– این الگو یا به نوعی Template مد نظر در مورد یک مشکل بحث میکند و راه حل های اثبات شده ای را برای حل مسئله ارائه میدهد.

– الگوها میتوانند به صورت های مختلفی و فاز های مختلفی ارائه میشود مانند Stage Patterns, Task Patterns, Phase Patterns

– اگر بخواهیم الگوها را تقسیم بندی نمائیم در حالت کلی دو دسته الگو را میتوان تعریف نمود.

1 الگوهای مورد استفاده در فاز تولید نرم افزار
Process Pattern Types

1. **Stage patterns** → – موارد مربوط به فعالیتهای چهارچوب (Framework Activities) را مدل میکند.
2. **Task patterns** → – موارد مربوط به (Work Task , Action) را مدل میکند.
3. **Phase patterns** → – موارد مربوط به یک رشته از فعالیتهای چهارچوب که شامل یک فاز کامل میشود را مدل میکند.

– با توجه به نیاز میتوان از هر کدام از الگوها استفاده نمود.

2 الگو های مربوط به بهینه سازی و توسعه و ارزیابی کیفیت فرایند نرم افزار
Process Assessment and Improvement

– توسط این الگوها کیفیت نرم افزار تست میگردد

– فرایندها وقتی تعریف میگردند به هیچ عنوان مشخص نمیکنند که نرم افزار سرموقع معین آماده گردد یا هر نیازی که مشتری دارد را برآورده نماید به همین دلیل میتوان از الگو های ارزیابی استفاده نمود تا مطمئن شویم در شرایط بحرانی چه کاری باید انجام داد

1. **Standard CMMI Assessment Method for Process Improvement (SCAMPI)**
   initiating, diagnosing, establishing, acting and learning.
   – مدل ارزیابی فرآیند که ۵ مرحله آماده سازی ، تشخیص ، ایجاد ، عمل و آموزش را ارزیابی کرده و سپس نتیجه گیری مینماید.

2 **CMM-Based Appraisal for Internal Process Improvement (CBA IPI)** – استانداردی که شامل تکنیکهای تشخیص برای ارزیابی یک نرم افزار سازمانی میگردد.

3 **SPICE—The SPICE (ISO/IEC15504)** – شامل استاندارد تعریف یک سری از نیازهای ارزیابی کارائی نرم افزار برای استاندارد سازی و کمک به توسعه نرم افزار میگردد.

4 **ISO 9001:2000 for Software** – یک استاندارد عمومی کاربردی برای بهبود کیفیت کلی محصولات نرم افزاری یک سازمان میباشد.

مدل های تجویزی
Prescriptive Models

– در ارتباط با الگوهای تولید نرم افزار از یک سری مدل های تجویزی استفاده میشود تا از بی نظمی و بوم ریختگی که در تولید نرم افزار بوجود می آید جلوگیری نماید . و مراحل تولید نرم افزار را ساختار سیستماتیک و منظم مینماید.

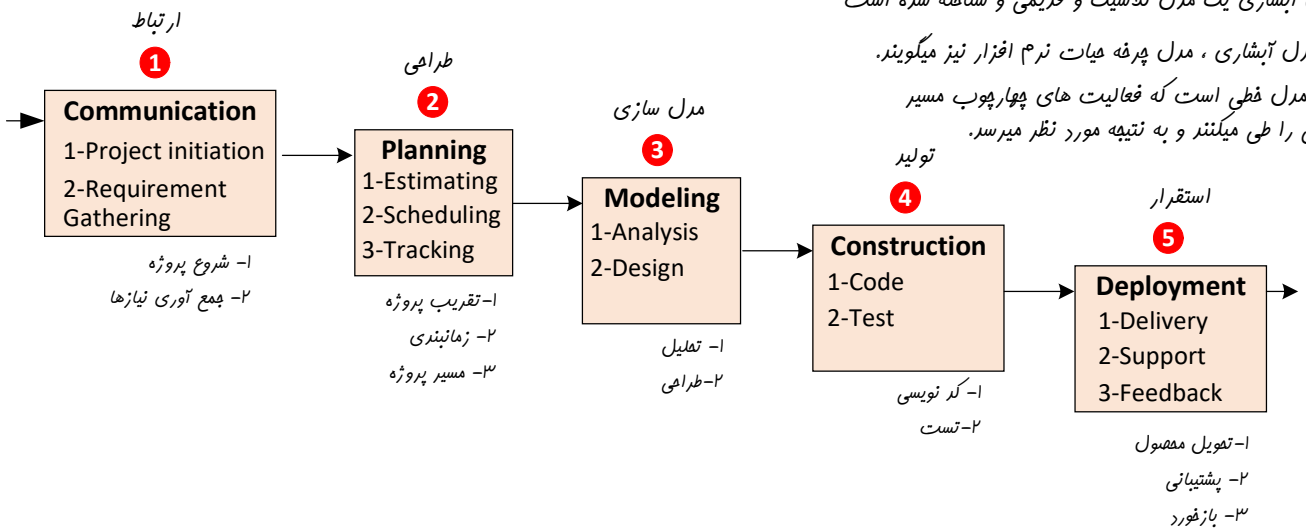
– مدل های تجویزی مناسب با تغییرات جدید در تولید نرم افزار میباشد و میتواند در کیفیت تولید نرم افزار موثر باشد هرچند که امکان استفاده از مدل های قدیمی هم با توجه به شرایطی نیز امکان پذیر میباشد.

1 مدل آبشاری
Waterfall Model

– مدل آبشاری یک مدل کلاسیک و قدیمی و شناخته شده است

– به مدل آبشاری ، مدل چرخه حیات نرم افزار نیز میگویند.

– یک مدل خطی است که فعالیت های چهارچوب مسیر مشخصی را طی میکنند و به نتیجه مورد نظر میرسد.



– از مشکلات مدل آبشاری میتوان به این نکات اشاره نمود

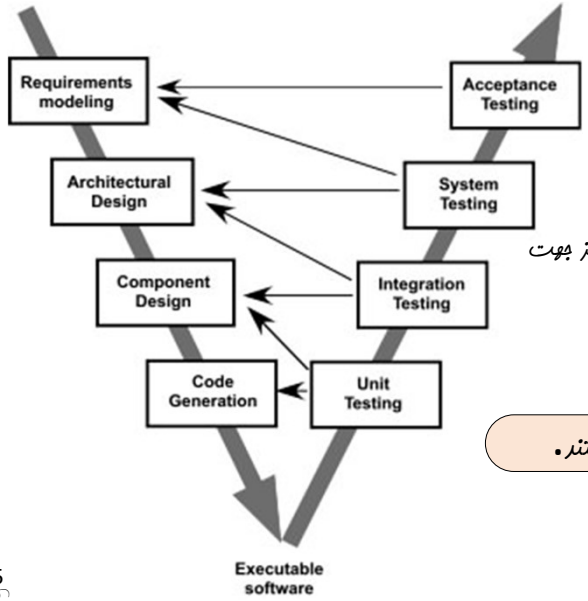
۱- که بندرت این امکان وجود دارد که مسیر تولید نرم افزار بصورت خطی و به ترتیب جلو رود و معمولا مواردی پیش می آید که مجبوریم به عقب برگردیم و تغییراتی ایجاد نمائیم (نیازمندی جدیدی اضافه ، حذف یا تغییر داده شود)

۲- معمولا مشتریان در شروع کار نیازمندی های دقیق خود را نمیتوانند بطور کامل و جامع بیان کنند .

۳- مشتری باید صبر کند تا نرم افزار به فاز آخر برسد و نتیجه ای از کار را مشاهده کند که در اینحالت اگر زمانبندی درست انجام نشود مشکلات زیادی بوجود خواهد آمد.

۴- هر فاز باید کامل شود تا فاز جدیدی شروع گردد و در جایی کاربرد دارد که نیازمندی ها بطور کامل مشخص گردد.

2 مدل وی V
The V-Model



– نوعی از مدل آبشاری میباشد که فعالیتهای تضمین کیفیت در نظر گرفته میشود

– هر کدام از فعالیتهای چهارچوب که انجام میشود درکنار آن فعالیتهای تضمین و کنترل کیفیت نیز جهت بازبینی و اعتبارسنجی انجام میگردد.

مدل آبشاری و مدل **V** بعلت عدم سازگاری با تغییرات ، خیلی کاربردی نیستند.

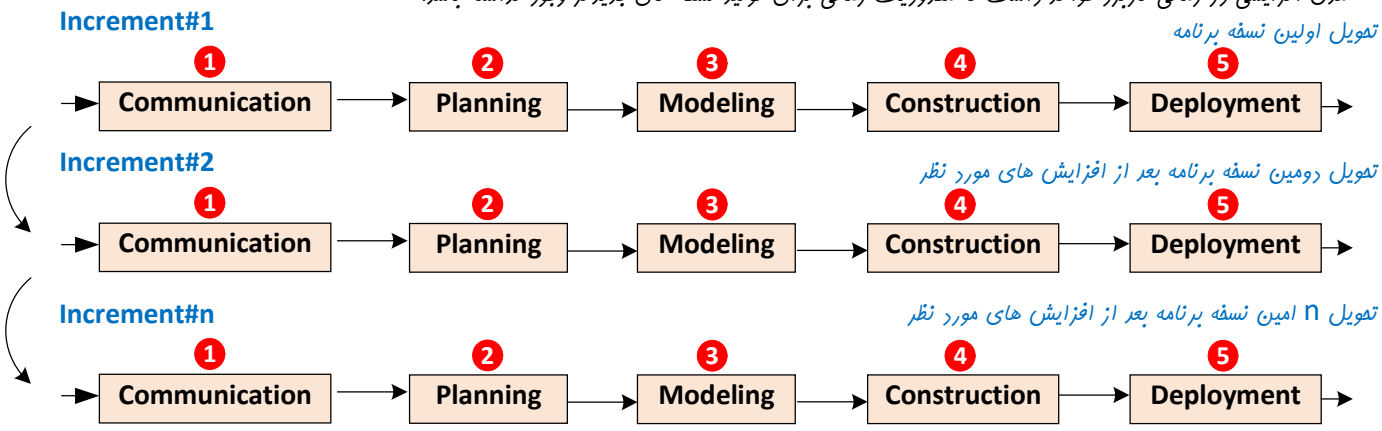
3 مدل افزایشی
The Incremental Model

– همانند مدل آبشاری هر فاز بطور کامل انجام گردیده و افزایش های مورد نظر و تغییرات اعمال میگردد ، بدین ترتیب در اتمام هر فاز نسخه مشخصی از نرم افزار تولید میگردد.

– بطور مثال فرض میکنیم قصد تولید یک برنامه پردازش کلمات داریم در فاز اول مدیریت فایل و ویرایش را انجام میدهیم ، پس از انجام ۵ مرحله ، اولین خروجی که نسخه اول نرم افزار میباشد را تولید میکنیم .در نسخه بعدی بطور مثال ویرایش های پیچیده تر را اضافه میکنیم و همین طور این روند میتواند ادامه پیدا کند.

– در مدل افزایشی ابتدا یک مدل پایه تولید می گردد و با توجه به نیازمندیهای جدید نسخه های جدید تر عرضه میگردد.

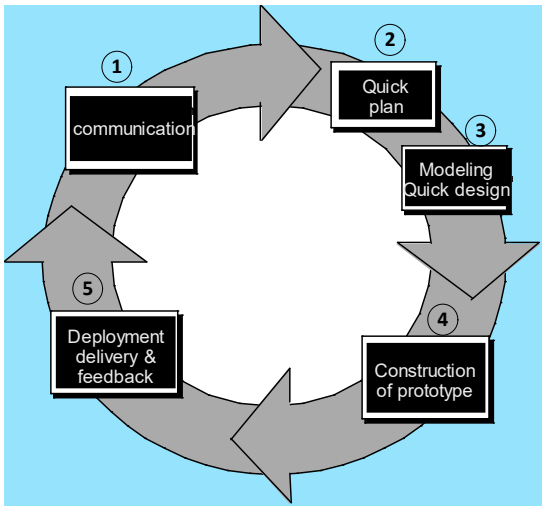
– مدل افزایشی در زمانی کاربرد خواهد داشت که محدودیت زمانی برای تولید نسخه های جدیدتر وجود نداشته باشد.



مدل های تکاملی
Evolutionary Models

– این مدل ها مرتبط با سیستمهای پیچیده میباشد . سیستمهای پیچیده در طول زمان تکمیل میشوند و یک مسیر مستقیم برای تولید نرم افزار مد نظر قرار نمیگیرد.

1 مدل نمونه سازی
Prototyping Model



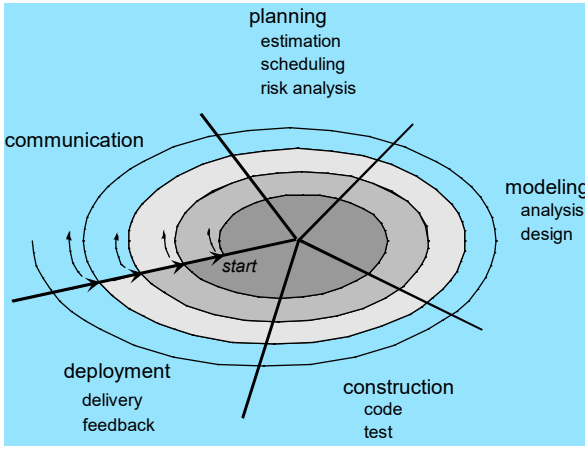
– در این روش ابتدا نیازمندیها جمع آوری شده سپس یک طرح سریع مستند میگردد بعد مدل سریع ساخته میشود و یک نمونه برای ارائه به مشتری ساخته شده و پس از تحویل به مشتری بازخوردهای آن مشخص شده و پس از اعمال تغیرات در نیازمندیها ازسوی مشتری مجددا به چرخه فعالیت های چهارچوب برگشته و نسخه بعدی تولید میگردد و در صورت نیاز مشتری این سیکل تکرار میگردد تا چرخه تکاملی نرم افزار کامل گردد.

– از مشکلات این مدل میتوان به

۱– مشتری پس از تولید نمونه اولیه انتظار دارد با کمی تغییرات بهترین کارائی را نرم افزار داشته باشد ، در صورتیکه این یک نمونه اولیه است و زمان زیادی طول میکشد تا نرم افزار از هر حیث کامل باشد .

۲–چون نمونه اولیه سریع تولید میشود ممکن است دارای مشکلات ساختاری همچون الگوریتم نامناسب و یا بیس نرم افزاری مناسبی بکار گرفته نشده باشد.

2 مدل مارپیچی ، حلزونی
The Spiral Model



– در این مدل از ویژه گی تکرار پذیری مدل نمونه سازی به همراه کنترل سیستماتیکی که در مدل آبشاری وجود دارد ترکیب نموده و نرم افزار بصورت یک سری از نسخه های افزایشی تولید میشود.

– در اینحالت نسخه اولیه ممکن است بر روی کاغذ مستند گردد یا بصورت نمونه اولیه تولید گردد ، ولی در تکرارای بعدی نرم افزار پیچیده تر ، کامل تر و مهندسی شده تر میشود.

– در این مدل هر کدام از قطاع ها به فعالیت های چهارچوب اختصاص داده شده و در جهت عقربه ساعت و بر اساس تکرار به کامل تر شدن و توسعه نرم افزار میپردازیم

– خاصیت توسعه پذیری و تکامل در طول زمان بر اساس تغییرات و نیاز به روزشدن نرم افزار و لحاظ نمودن و کم کردن ریسکهای احتمالی در این مدل در نظر گرفته میشود.

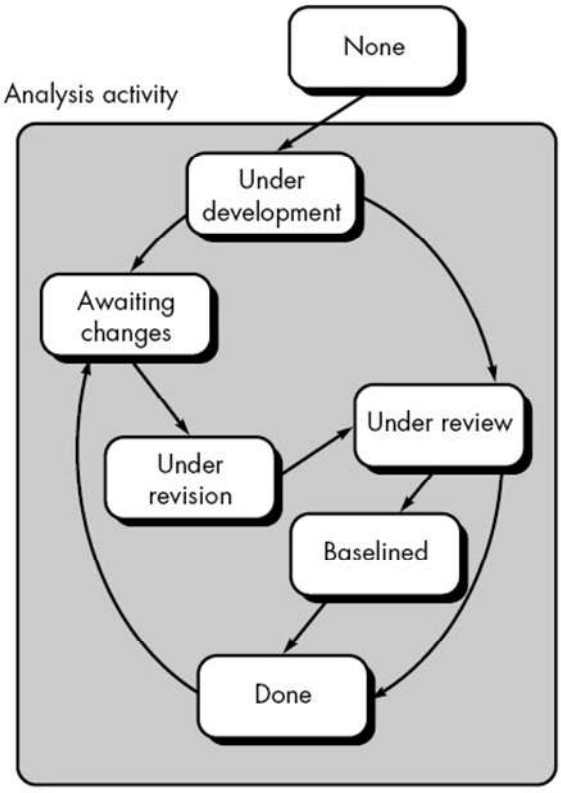
– از معایب این مدل میتوان به

۱– سخت بودن متقاعد کردن مشتری برای استفاده از این مدل اشاره نمود

۲– در صورتیکه نتوانیم ریسکها را مدیریت و کنترل کنیم دچار مشکلات زیادی خواهیم شد.

–از این مدل در چرخه تولید نرم افزار زیاد استفاده نمیشود.

3 مدل هم روند
Concurrent Model



– در این مدل شمایی (دید انتزاعی) از فعالیت ها یا وظایف و وضعیت خاص هر کدام از آنها (درحال توسعه بودن ، ارزیابی ، بازبینی ، منتظر تغییرات) مشاهده میگردد.

–در یک نگاه State دقیق هر یک از فعالیت های چارچوب مشخص میگردد.

– در شکل روبرو وقتی وارد یک فعالیت میشویم مسیر بدین شکل خواهد بود.
۱– ابتدا غیر فعال
۲–در حال توسعه
۳– سپس اگر نیاز به تغییراتی باشد به حالت منتظر تغییرات و بعد بازبینی و بعد ارزیابی خواهد رفت.
۴–در صورت عدم نیاز به تغییرات به حالت ارزیابی میرود.
۵–حالت انجام تغییرات و انجام شده.

مدل هم روند ، مدل مناسبی در انواع توسعه های نرم افزاری میباشد.

– در این مدل براحتی مشخص میگردد که در آن واحد چند فعالیت یا وظیفه در حال انجام میباشد و هر کدام از آنها در چه وضعیتی میباشد.

دیگر مدل های فرآیند
Other Process Models

1 مدل توسعه مبتنی بر اجزاء
Component based development

– روشی است که از اجزاء تکراری مختلف برای پیشبرد و تکامل نرم افزار آن استفاده میکند و فیلی از ویژه گیهای مدل حلزونی را در بردارد و از نظر ماهیت دارای ساختار تکاملی و تکرار شونده دارد. بدین صورت که از یک سری اجزاء Component استفاده شده و روند تکاملی آن بصورت نسخه های جدیدتر ادامه میابد.
– از مزیت های این مدل میتوان به
۱– کاهش زمان توسعه نرم افزار بعلت اینکه معمولا از اجزاعی که از قبل وجود داشته استفاده میکنیم و در صورت نیاز اجزاء جدیدی تولید میشود.
۲–باعث کاهش پیچیدگی های نرم افزار و جلوگیری از سفارشی سازی بودن آن

2 مدل شیوه های رسمی
Formal methods

– شامل یک مجموعه از فعالیت های نرم افزار میگردد که بصورت ریاضی تعریف شده است.
– مهندسی نرم افزار با استفاده از یک سری علائم ریاضی سیستم را تعریف ، پیاده سازی و بررسی مینماید.
– از این روش بعلت استفاده از فرمولهای ریاضی و دقیق و تمیز بودن و عدم ابهام و ناسازگاری با نام Clean Process نیز یاد میگردد

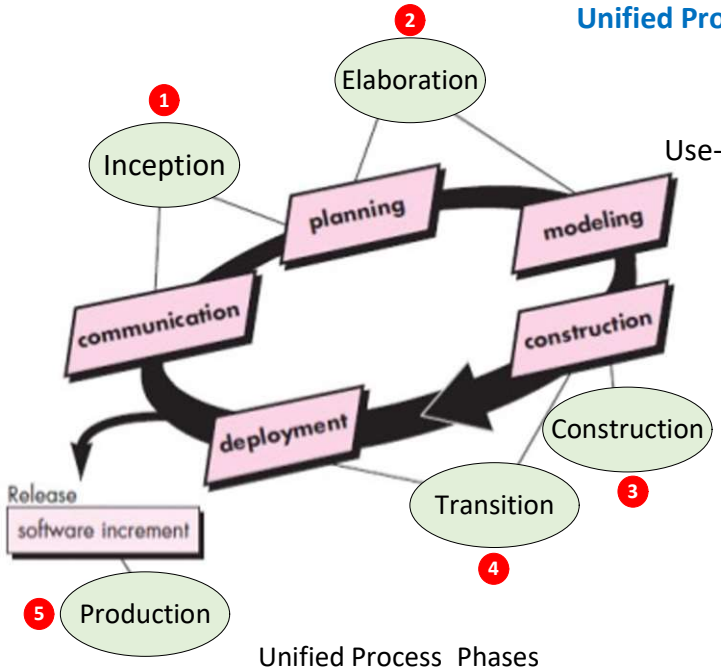
از معایب این روش ریاضی میتوان به موارد زیر اشاره نمود.
۱–مدل های ریاضی وقت گیر و پرهزینه هستند.
۲–همه تولید کنندگان نرم افزار اطلاعات کامل ندارند و پیاده سازی این روش نیاز به آموزش دارد که هزینه بر میباشد.
۳–بعلت عدم ارتباط کامل و دوسویه منظم با مشتری دقیقا مشخص نیست که آیا همه نیازمندیهای مشتری پوشش داده شده است یا خیر.

3 مدل روش جنبه گرا
AOSD (Aspect Oriented)

– در زمان توسعه نرم افزاری ،به خاطر نیاز کاربر و شرایط محیط سیستمهای نرم افزاری به تدریج پیچیده تر میشوند . از همین رو میتوان از مدل هایی استفاده نمود که این پیچیدگیها را نیز در نظر بگیرد بطور مثال تمرکز بر روی خصوصیتهای سطح بالای سیستم یا مشکلاتی که بر روی عملکردها یا وظایف تاثیر میگذارند.

– در مدل های جنبه گرا مشکلات را بصورت گروه ها تعریف مینمایند بطوریکه بر روی معماری نرم افزار تاثیر خواهند داشت و بر اساس اهمیت تاثیر گزاری گروه بندی و طبقه بندی میگردد.

4 مدل فرآیندهای یکپارچه
Unified Process

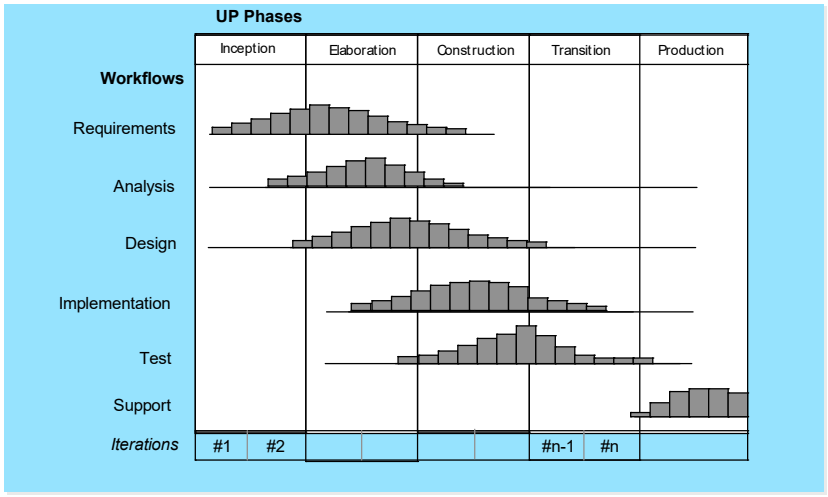


- این مدل ها اهمیت ارتباط با مشتری را از دید کاربر Use-case driven بررسی مینمایند.و از یک مدل برنامه سازی یکپارچه UML استفاده میشود.
- با توجه به نیازهای کاربر کل برنامه مدل میگردد

- زبان UML حاوی یک سری علائم قدرتمندی است که در جهت مدل سازی و توسعه سیستمهای شی گرائی میباشد.

- زبان UML استفاده وسیعی در مدل سازی دارد

Unified Process Phases



فازهای مدل فرآیندهای یکپارچه
Unified Process Phases

1. **inception** شروع به کار
2. **Elaboration** پیچیدگی و جزئیات
3. **Construction** ساخت
4. **Transition** انتقال
5. **Production** تولید

فرآیند تولید نرم افزار

1 فرآیند نرم افزار بصورت فردی
Personal Software Process (PSP)

- همه کارها توسط شخص تولید کننده نرم افزار انجام میگردد و شامل ۵ فعالیت میشود.

1 Planning برنامه ریزی

- تضمین منابع و کاری که باید انجام گردد، ارزیابی خرابی و ریسک ، توسعه آتی و به طور کل همه معیارهای مستند گردیده و زمانبندی پروژه انجام میگردد.

2 High-level Design طراحی سطح بالا

- مولفه جزء هایی که باید ساخته شود و نمونه های اولیه زمانی را مشخص ، مستند و پیگیری میکنیم .

3 High-level Design Review مرور طراحی سطح بالا

- طراحی سطح بالا را برای خطاهای احتمالی مرور مجدد میکنیم تا متوجه شویم آیا به هدف طراحی که مد نظر بوده خواهیم رسید؟

4 Development توسعه

- کامپوننت هایی که در طراحی سطح بالا مدل شده اند را تولید میکنیم. کد نویسی

5 Postmortem پس از وقوع

- اندازه گیری بهینه بودن فرآیند و خطاهای آن از طریق تحلیل آماری . تشخیص به هنگام خطاها بسیار با اهمیت میباشد . پس باید معیار های مهمی تعریف و اندازه گیری شود تا پیش بینی خطای دقیق تر و زمانبندی پروژه بهتر حاصل گردد.

- روش PSP کاربرد زیادی در صنعت ندارد ولی برای استفاده شخصی بسیار مناسب است،
- از مشکلات این روش آموزشهای خیلی طولانی و پرهزینه بودن تربیت فرد متخصص است

1 فرآیند نرم افزار بصورت تیمی

Team Software Process (TSP)

– با توجه به اینکه بسیاری از پروژه های نرم افزاری بصورت تیمی میباشد از روش TSP استفاده میگردد.

– در این روش تیم به صورت خود هدایتی Autonomy سازماندهی میشود بطوریکه بتوانیم با کیفیت بالا چرخه تولید نرم افزار انجام گیرد و شامل اهداف زیر است.

– تیم بصورت خودکار کارهای تیمی را طراحی میکنند و در تیمهای یکپارچه تولید نرم افزار تا ۲۰ مهندس نیز به کار گرفته میشود.

– به مدیران نشان داده میشود که چطور انگیزش در این تیم ها را ایجاد کنند و به آنها کمک نمایند تا بیشترین کارائی را داشته باشند.

– برای مدل های کنترل کیفیت تولید نرم افزار بطور مثال از استاندارد CMMI-5 استفاده شده تا کیفیت و سرعت تولید نرم افزار بالا برود.

– تهیه نمودن یک راهنمای بهبود یافته سطح بالا برای سازمان ها

– ساده تر شدن آموزش دانشگاهی و رتبه های مهارت های صنعتی

– تیمی که بصورت خود هدایتی TSP رفتار میکند ، دارای یک درک تیمی از مولفه ها و اهداف میباشد و نقش ها و وظایف هریک از افراد تعریف شده ، فرآیندهای مناسبی که باید انجام شود را مشخص نموده ، بطور مداوم احتمال ریسکهایی که بوجود خواهد آمد را مشخص ، کنترل و مدیریت میکند و گزارش میدهد. در این روش تیم میتواند با نیازها سازگار و منطبق شوند ، زمانبندی را خود اندازه گیری ، تحلیل و کنترل کنند تا به هدف تولید نرم افزار برسند.

– این روش بیشترین استفاده را در صنعت دارد.

مهندسی نرم افزار سریع

Agile Software Development

– در محیط تجاری مدرن فعلی و سیستمهای کامپیوتری و محصولات نرم افزاری به سرعت در حال پیشرفت و تغییر هستند ، در صورت استفاده از اصول اصلی و اولیه توسعه نرم افزار سرعت پیاده سازی کند میگردد و تا زمان تحویل سیستم کلی تغییرات اتفاق خواهد افتاد . به همین علت مهندسی نرم افزار سریع مفهوم پیدا میکند.

– فعالیت ها بر طبق روال های قبلی مهندسی نرم افزار میباشد (فعالیت های چارچوب) ولی سعی بر این است که بصورت مجموعه کارهای حداقلی تغییر شکل داده شود و تیم نرم افزاری به سمت ساخت و تحویل نرم افزار بصورت سریع و هماهنگ پیش رود.

ارزشیابی های مهم در مهندسی نرم افزار سریع

– پارامترهای مهم بر اساس افراد تیم و تعامل بین آنها میباشد.

– ارزشیابی نرم افزار و مستندات مرتبط با آن

– همکاری کاربر با کسی که در حال مذاکره برای تحلیل نیاز های نرم افزار میباشد.

– هماهنگی سریع با تغییرات برنامه ریزی و طراحی

– مهندسی نرم افزار سریع چیزی فراتر از صرفا یک پاسخ سریع به تغییرات میباشد و موضوعاتی دیگری همچون رابطه سریع بین افراد یک تیم ، تحویل سریع نرم افزار به صورت نسخه های کوچک از برنامه با فواصل زمانی کوتاه میباشد

مفهوم چابکی در تولید نرم افزار

Agility Concept

– پاسخ سریع به تغییرات ، ارتباط مداوم با کاربر همانند اعضاء تیم . به نوعی وارد کردن مشتری به تیم جهت تست نسخه های سریع و فید بک مداوم ، آماده سازی تیم برای کار مداوم در چرخه تولیدوتحویل افزایشی نرم افزار

چابکی و تغییرات هزینه

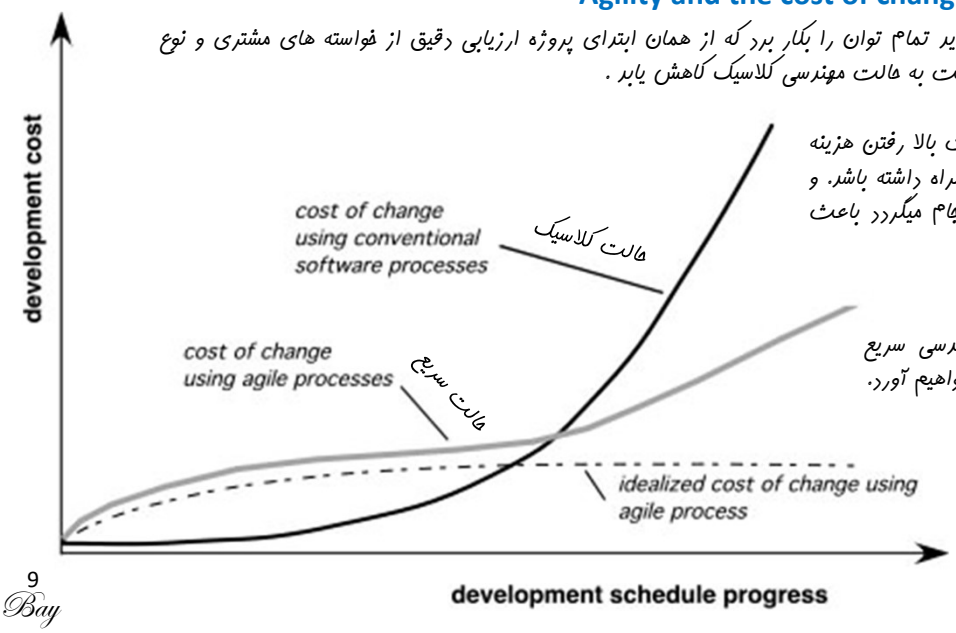
Agility and the cost of change

– نکته مهم اینست که برای مهندسی نرم افزار سریع باید تمام توان را بکار برد که از همان ابتدای پروژه ارزیابی دقیق از خواسته های مشتری و نوع معماری بدست آوریم تا در طول پروژه هزینه ها نیز نسبت به حالت مهندسی کلاسیک کاهش یابد .

– تغییرات مهم از سوی مشتری در انتهای پروژه با عث بالا رفتن هزینه پروژه و حتی ممکن است تغییر در مدل سازی را به همراه داشته باشد. و در حالت سریع که تغییرات در فواصل زمانی کوتاه انجام میگردد باعث یکنواختی هزینه ها میشود.

– اگر طراحی خوبی انجام گردد ، هزینه های مهندسی سریع یکنواخت خواهد شد و حالت ایده آل را به دست خواهیم آورد.

– تحویل نسخه های سریع با فواصل زمانی کوتاه در طراحی و تولید و پایین آوردن هزینه ها بسیار موثر است.



فرآیند چابکی
An Agile Process

– در یک فرآیند سریع
۱– نیازمندیها بصورت سریع توسط مشتری توصیف میگردد
۲– طرح ها با مدت زمن حیات کوتاه مشخص میگردند
۳– نرم افزار بصورت تکرار های مختلف و با تاکید بر فعالیت هایی که در هر دوره باید انجام گردد توسعه افزایشی میابد
۴– هماهنگی و سازگاری در تغییرات به دست می آید.

اصول چابکی
Agility Principle

– یک سری اصول کلی برای پیاده سازی مهندسی نرم افزار سریع تروین شده اند .

۱– اولویت اول راضی نگه داشتن مشتری است و در این حین تکه های کوچک نرم افزاری جهت تست و بازخورد مشتری سریع و پیوسته آماده میشود.
۲– همواره به نیازمندیهای که در هر مرحله ای تغییر میکنند خوش آمد گفته میشود چون بعنوان مزایایی برای رقابت در جذب مشتری محسوب میگردد.
۳– نرم افزار هایی که بصورت پیوسته تحویل داده میشود بین دو هفته تا دو ماه یکبار میباشد.
۴– تیم تولید کننده نرم افزار و مشتری باید هر روز با هم در ارتباط باشند .
۵– پروژه های بر اساس نیاز کاربر تعریف شده و در حقیقت کاربر محیطی را برای تست و اعلام نیاز خود دارد.
۶–موثرترین روش دریافت اطلاعات برای تیم توسعه دهنده نرم افزار مصاحبه های رو در رو میباشد.
۷–معیار سنجش پیشرفت کار خوب ، نرم افزار تولید شده در حال کار و عملیاتی میباشد.
۸– باید یک جریان توسعه از ابتدا تا انتهای پروژه بصورت ثابت و رو به جلو حفظ گردد .
۹– نیاز به تکنیک و طراحی خوب برای تیم توسعه الزامی است .
۱۰ – ساده سازی مقدار کار انجام نشده از هنر های چابکی میباشد.
۱۱– بهترین معماری ، طراحی ، تحلیل نیازمندی و ساختارمندی توسط تیم های خود سازمانده انجام خواهد گرفت.
۱۲–در بازهای منظمی که نسخه های مختلف نرم افزار تولید میگردد کیفیت نرم افزار سنجیده شده و رفتار موثر تیم، هماهنگ و منطبق با خواسته های مشتری میگردد.

عوامل انسانی اصول چابکی
Human Factors

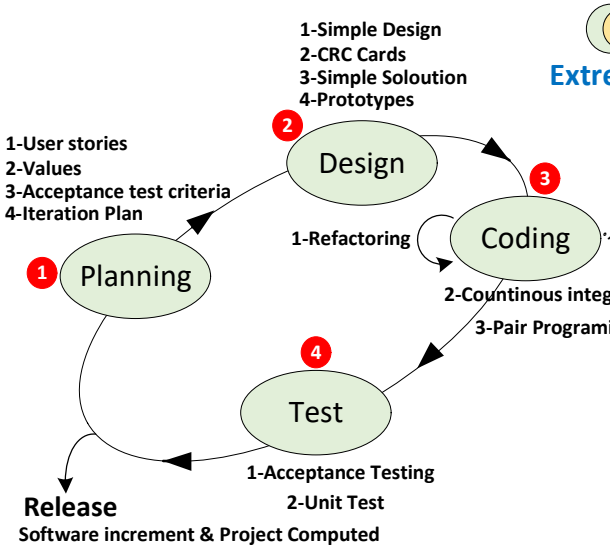
– توسعه سریع نرم افزار ، خیلی زیاد متکی به مهارتهای فردی متخصصین تیم خود سازمان ده میباشد

– ویژه گیهای فردی و توانایی افراد تیم خود سازمان ده به شرح ذیل است.

1. **Competence.** – داشتن استعداد ذاتی، مهارت و شایستگی در رقابت
2. **Common focus.** – داشتن تمرکز مشترک در هدف نهایی
3. **Collaboration.** – داشتن همکاری و روحیه تیمی
4. **Decision-making ability.** – داشتن قابلیت و توانایی تصمیم سازی در شرایط مختلف تکنیکی توسط تیم خودسازمان ده
5. **Fuzzy problem-solving ability.** – دارای قابلیت حل نمودن فازی مسائل از جمله ، تحلیل سریع خواهد شد
6. **Mutual trust and respect.** – برقراری اعتماد و ملاحظه در بین افراد تیم
7. **Self-organization.** – مدیریت خود سازمانده تیم و ایجاد انگیزش و دلگرمی بیشتر در افراد تیم در جهت توسعه نرم افزار

مدل های استاندارد توسعه سریع نرم افزار

۱ برنامه نویسی اغراطی
Extreme Programming (XP)



۱ Planning برنامه ریزی

– برنامه نویسی مفرط یا اغراطی قدیمی ترین میباشد و بصورت گسترده در حال استفاده میباشد و عمومیت بالا دربین دیگر مدل ها دارد.
– این نوع برنامه نویسی بر پایه شی گرائی بنا نهاده شده و دارای چهار قانون و زمینه فعالیتی میباشد.
– با تولید نیاز های کاربر شروع شده
– هر نیازکاربر به سرعت ارزیابی و هزینه آن(زمان مورد نیاز برای هر نیاز) به آن نسبت داده میشود
– اگر زمان به بیش از سه هفته ارزیابی شود باید نیاز مشتری به قطعات کوچکتر تقسیم گردد.
– نیاز هایی به گروه های مختلفی برای توسعه افزایشی و تحلیل موثر تقسیم بندی میگردد
– هر کدام از نیازها که دارای ارزش بیشتر و یا ریسک بیشتر هستند در ابتدای جدول زمانبندی قرار میگیرند. و توافق بر زمانبندی نیاز ها حاصل میگردد.
– بعد از انجام کار اول صحیح بودن زمانبندی و یا نیاز به تغییر زمانبندی برای ادامه توسعه نیازها در نظر گرفته میشود.

② Design طراحی

- اولین اصل ، اصل ساده نگاه داشتن طراحی میباشد تا پیچیده نمودن آن
- استفاده از کارتهای CRC که برای مشخص کردن و سازماندهی کردن کلاس های شی گرائی و توسعه افزایشی جاری استفاده میگردد
- برای طراحی های پیچیده از راه حل Spike استفاده میشود بدین مفهوم که یک نمونه اولیه Prototype با طراحی ساده تر استفاده میکنیم.
- ابتدا یک پلایش اولیه و تکراری از طراحی درونی برنامه انجام میدهیم Refactoring

③ Coding کدنویسی

- پس از انجام طراحی اولیه و مشخص نمودن مسیر کار وارد مرحله کد نویسی میشویم.
- هر کدی که آماده میشود باید دارای ساختار Unit test باشد که بتواند به راحتی مورد ارزیابی و آزمایش قرار گیرد.
- برنامه نویسی زوجی Pair Programming توصیه به این دارد که هر قسمت از برنامه توسط دو نفر کدنویسی گردد که باعث بالاتر رفتن کیفیت کار و کنترل بهتر کد نویسی و جزئیات و استانداردهای آن میگردد

④ Test آزمایش

- کلیه Unit test ها بطور روزانه اجرا میشوند و با توجه به اینکه حاوی انتظارات مشتری و نیاز های او میباشد نیز مورد پذیرش مشتری قرار خواهد گرفت.

- یک روش **XP** بصورت صنعتی نیز وجود دارد که یک تکامل ذاتی نسبت به نسخه پیشین میباشد و بر اساس ایده های مشتری محور ، آزمایش محور و حداقل سازی توسعه پیدا میکند و تفاوت اصلی آن با نسخه اولیه اینست که مدیریت و نقش توسعه یافته برای مشتری کمی متفاوت است

- تغییر پیاپی نیازمندی های مشتری میتواند از مشکلات کنترلی **XP** باشد.
- تناقض در نیازمندی ها نیز بعلت بیشتر غیر رسمی بودن ارتباط با مشتری وقتی پروژه بزرگ میباشد نیز از مشکلات کنترلی **XP** میباشد.

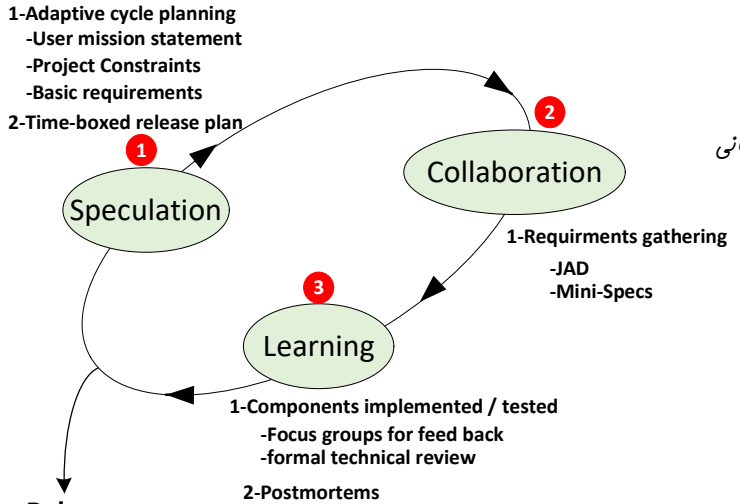
② روش توسعه مبتنی بر همکاری

Adaptive Software Development (ASP)

-تکنیکی برای ایجاد سیستمها و نرم افزارهای پیچیده با تمرکز بر همکاری افراد و خودسازماندهی تیمی میباشد.

Mission-driven planning	- طرح ریزی توسعه نرم افزار به شکل ماموریت
Component-based focus	- بر اجزاء شی گرائی تمرکز دارد.
Uses time-boxing	- تخصیص دقیق فواصل زمانی
Explicit consideration of risks	- تصور واضح از ریسک ها
Emphasizes collaboration for requirements gathering	- تاکید بر همکاری در جمع آوری نیازها
Emphasizes "learning" throughout the process	- تاکید بر درک کلیه فرآیندها

فازهای یک دوره ASP



① Speculation تعمق

- برای کاری که باید انجام گردد برنامه ریزی مینمائیم
- گرفتن نیاز ها و مشخص نمودن محدودیتها و نیازمندیهای پایه بر اساس توالی زمانی
- مشخص شدن یک طرح اولیه

② Collaboration همکاری

- براساس متدولوژی همکاری مداوم و منظم مشتری و کاربر با تیم طراحی
- یکپارچه سازی Joint مجموع طرح های اولیه

③ Learning یادگیری

- مولفه های پیاده سازی شده آزمایش میشوند
- گروه های مختلف بازخوردهای خود را جهت تمرکز دقیق تر به اشتراک میگذارند.
- تکنیکهای رسمی نیز در کار در نظر گرفته میشود.
- تشخیص به هنگام خطا ها و ریسک ها و به اشتراک گذاری دانش بدست آمده
- انتقال مهارت ها اعضای تیم در قالب همکاری فی مابین

- انتشار نسخه جدید و آمده سازی برای سیکل نسخه های جدیدتر نرم افزار

روش توسعه سیستمهای پویا 3

Dynamic Systems Development Method (DSDM)

- این روش یک چاپوبی را برای ایجاد و حفظ سیستمهایی را فراهم می آورد که با محدودیت های زمانی فشرده مواجه هستند.

- یکی از اصولی که در این روش بکار گرفته میشود اصل پرتو یا همان ۸۰-۲۰ میباشد.

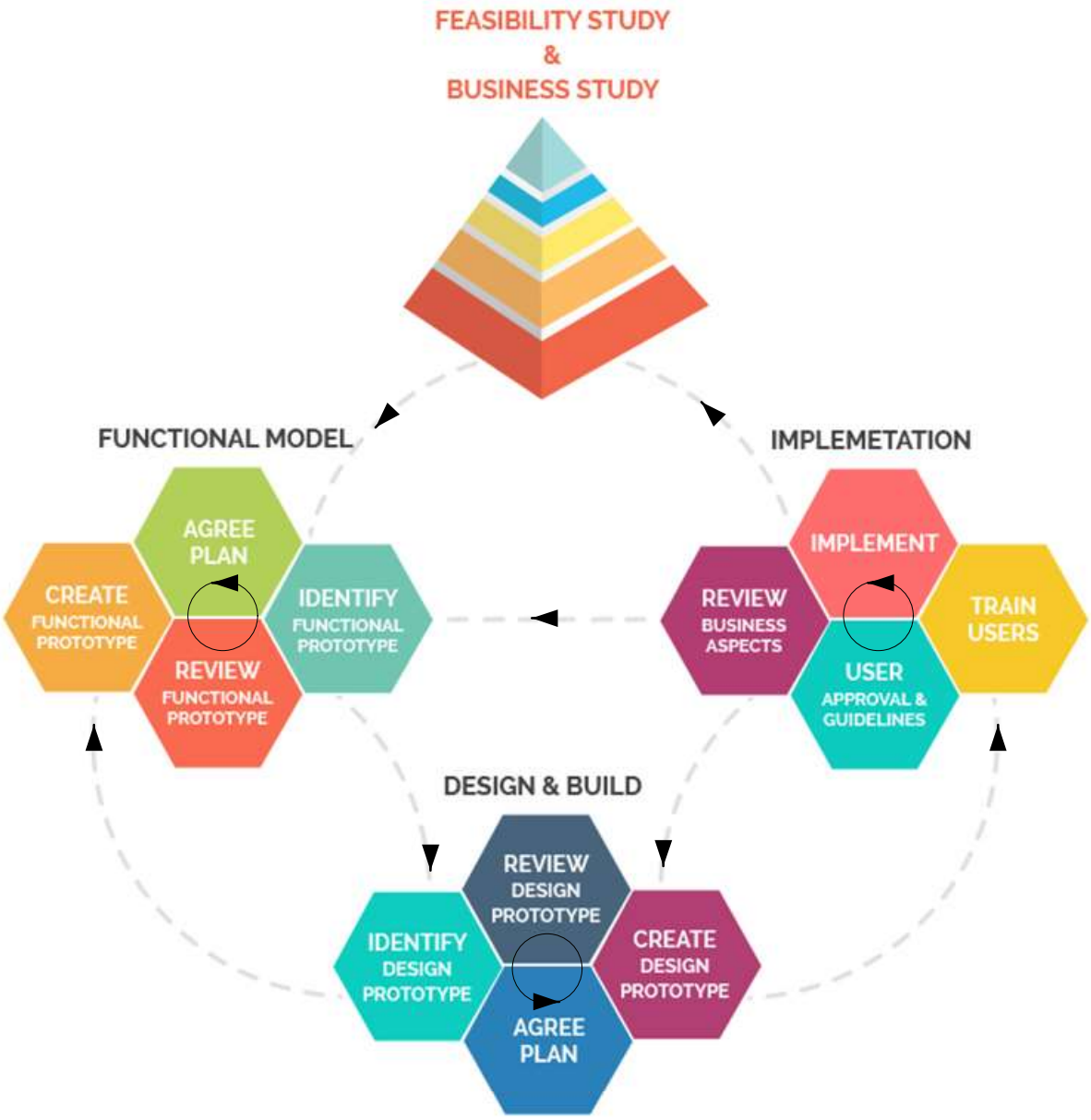
اصل پرتو

- برین مضمون که ۸۰ درصد کاری که باید در هر افزایش Increment انجام گیرد باید در ۲۰ درصد زمان آن کار انجام گردد.
- اگر نتوان این اصل را در ۲۰ درصد زمان انجام داد ، قسمت هایی که باعث بالا بردن زمان میشوند به افزایش بعدی انتقال داده میشود.

- در خیلی از موارد این روش DSDM شبیه روش های XP,ASD میباشد و یکی از تفاوتها ، اضافه شدن اصل پرتو میباشد.

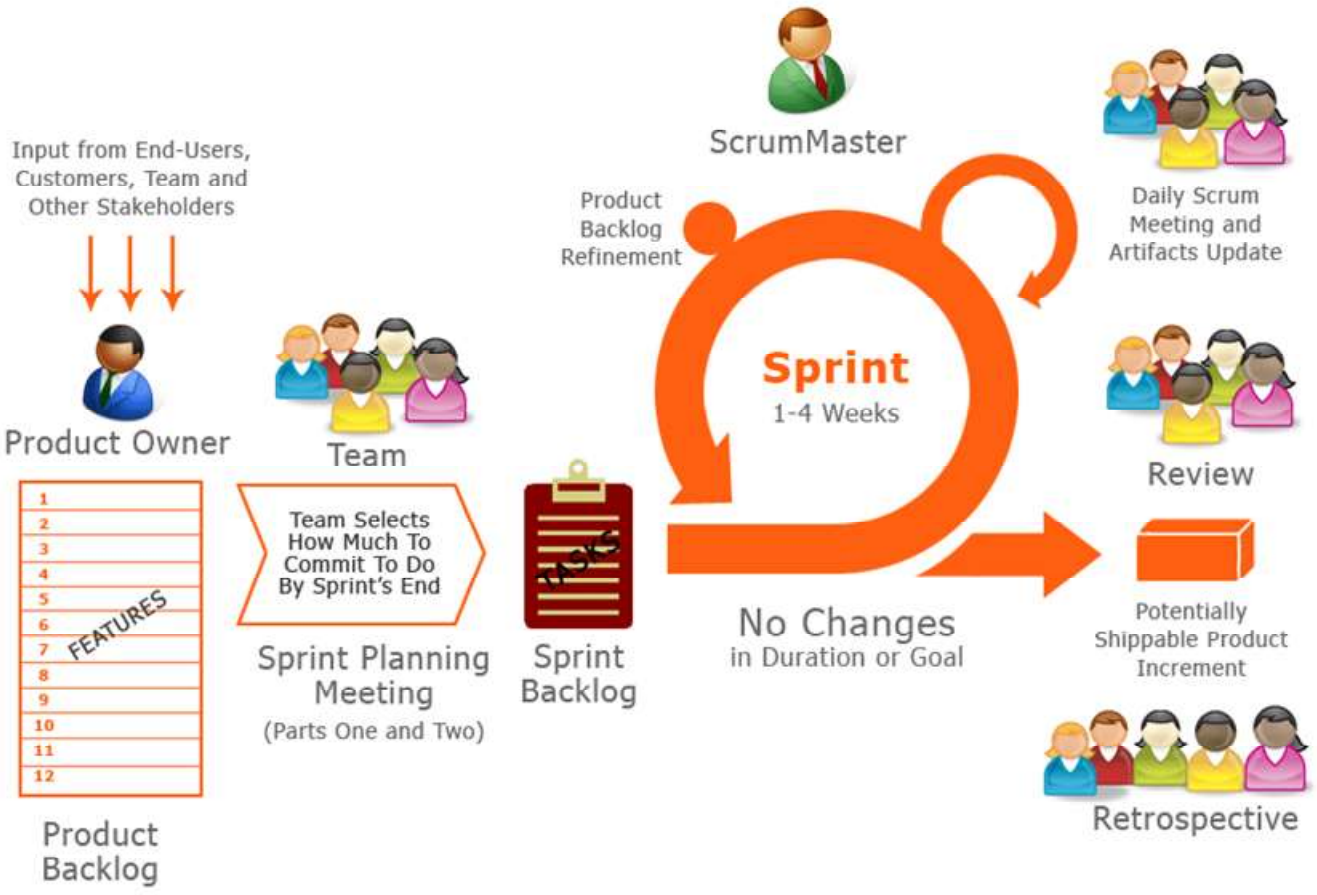
اصول روش DSDM

۱- کاربر بصورت جدی درگیر فرایند است
۲- اعضای تیم دارای قدرت تصمیم گیری بالایی هستند.
۳- نسخه های افزایشی برنامه در فواصل زمانی کوتاه باید آماده شوند.
۴-معیارهای سنجش برنامه ، افزایش های است که برای توسعه نرم افزار و رسیدن به هدف مشخصی دنبال شده است.
۵- روش ها تکراری و نسخه به نسخه ضروری میباشد.
۶- تغییرات در مراحل توسعه نرم افزار برگشت پذیر هستند.
۷- نیازمندی ها در سطح بالایی قابل انجام هستند .
۸- عملیات تست در چرخه فرایند در حال انجام شدن میباشد.



روش سریع اسکرام 4
Scrum Developement

- **Scrum** مربوط به یک واژه ای در ورزش راگبی میشود و نشان دهنده همکاری تیمی با سرعت بالا میباشد .
- توسعه کار در قالب یکسری **Packet** تقسیم بندی میشود
- تست و مستند سازی در هنگام ساخت محصول انجام میگردد.
- رویدادهای کاری سریع **Work occurs sprint** نیز براساس نیز بر اساس **Backlog** هایی که در زمان نیازمندی مشخص شده است جمع آوری میشود.
- ملاقات های تیمی و گروهی خیلی کوتاه هستند و بعضی اوقات بصورت ایستاده انجام میگردد.
- دموها به مشتریان بر اساس **Time box** های ازپیش تعیین شده برای تحویل به مشتری و گرفتن بازخورد مشخص میگردند.



- بطور مثال تکه های کوچک نرم افزاری توسط تیم اولویت بندی میشود و پیش نیاز های آن توسط هر تیم آماده میگردد ، جلسات روزانه ۱۵ دقیقه ای جهت مشخص نمودن پیشرفت کار و مشکلات تشکیل میشود
- در اینحالت چند مسئله کوچک بر اساس نیازمندیها مشخص میگردد .
- تیم دارای یک پشتیبانی اصلی میباشد **Scrum master** که شامل لیستی الویت بندی شده از نیازمندیهای پروژه بر اساس اولویت مشتری میباشد .
- لیست پشتیبان میتواند مرتبا در حال تغییر باشد.
- بین اعضاء مسابقه سرعت برای انجام موارد لیست پشتیبان بر اساس بازه زمانی مشخص شده وجود دارد.

روش توسعه کریستالی 5

Crystal Software Development

		Crystal Methodologies				
		Clear	Yellow	Orange	Red	Maroon
Criticality of the Project	Life (L)	L6	L20	L40	L80	L200
	Essential Money (E)	E6	E20	E40	E80	E200
	Discretionary Money (D)	D6	D20	D40	D80	D200
	Comfort (C)	C6	C20	C40	C80	C200
		1 to 6	7 to 20	21 to 40	41 to 80	81 to 200
		Number of People involved in the Project				

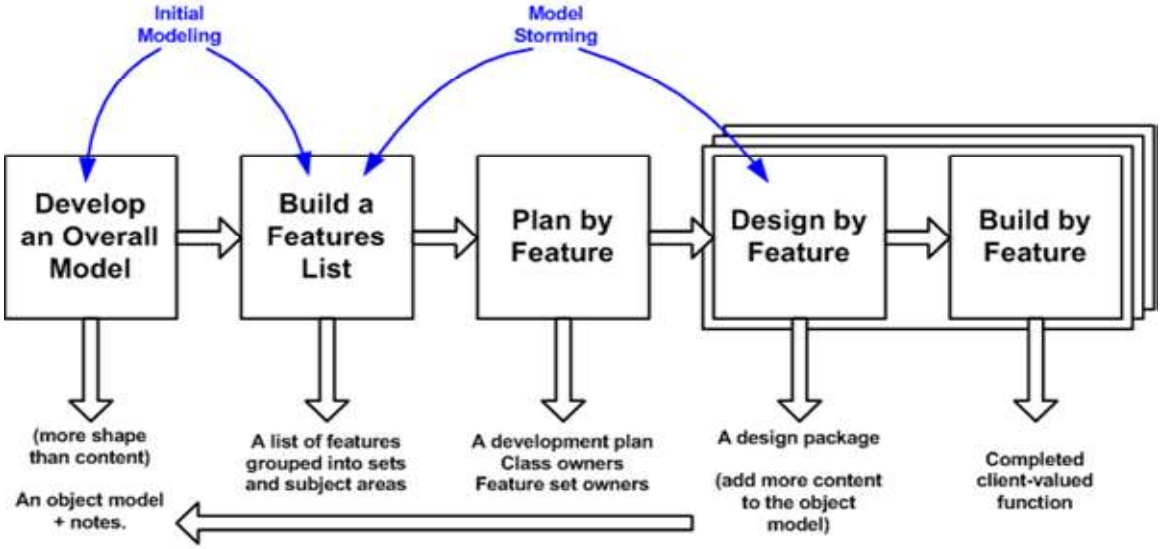
- در این روش تمرکز بر روی قابلیت مانور بر اساس خصوصیات مسائل میباشد. و دارای دو هدف است.

۱- هدف اصلی آن تحویل مفید و کارکرد خوب نرم افزار میباشد.
۲- هدف دوم آمادگی برای تحویل نسخه های بعدی نرم افزار است.

- تاکید بر مکالمه های رو در رو و دریافت بازخورد جلسات کارگروهی **Reflection Workshops** جریان کاری تیم را مرور میکند.

روش ویژه گی محور 6

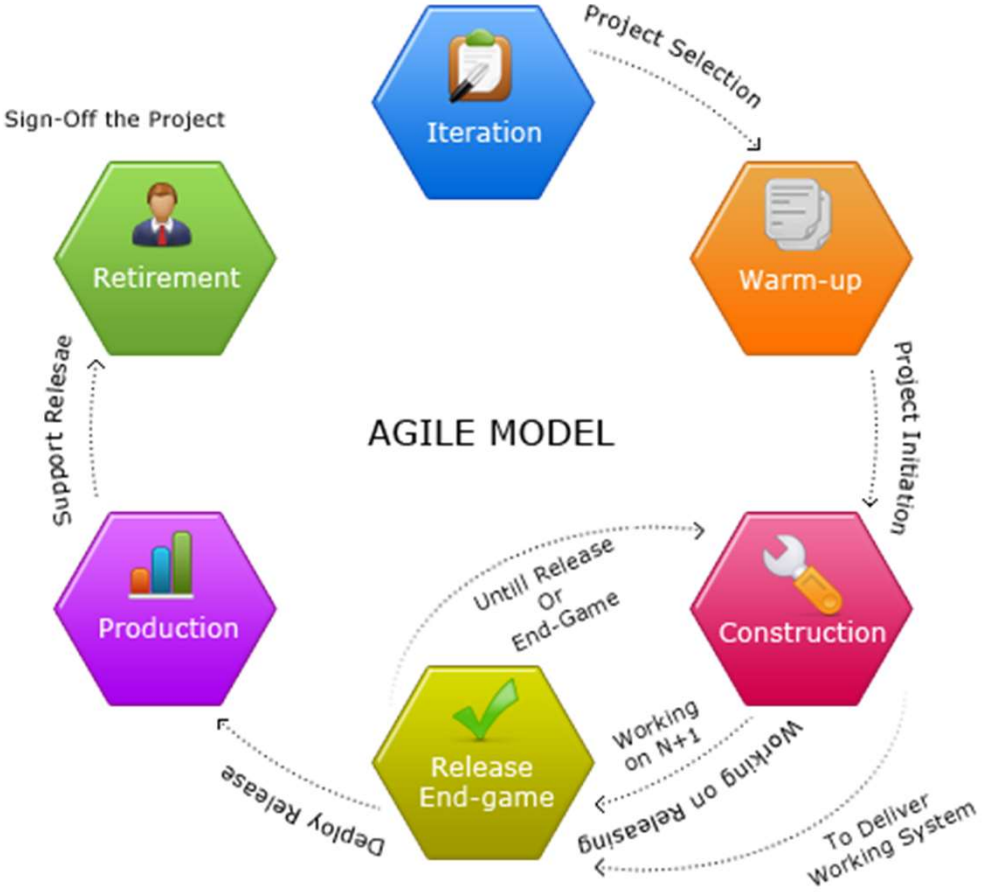
Feature Driven Development (FDD)



- این مدل از روش های شی گرائی استفاده میکند و برای قطعه بندی پروژه ها به قطعات کوچک مورد استفاده است .
- درحقیقت پیچیدگی یک مسئله میتواند به ویژگی های مختلف تجزیه گردد.
- تاکید بر **Feature** و ویژگی به مفهوم اینکه یک ویژه گی که حدود دو هفته یا زود تر بر اساس خواست مشتری آماده گردد.
- ویژه گی ها دارای یک **Feature template** میباشد . **<action> the <result> <by | for | of | to> a(n) <object>**
- یک لیست ویژه گی تولید شده و طراحی ویژه گی به آن اضافه میگردد.
- در این روش طراحی و ساخت بصورت همزمان انجام میگردد.
- این روش بر روی فعالیت های تضمین کیفیت نیز تاکید دارد
- کاربر خیلی راحت میتواند قطعات کوچک را توصیف کند و مورد آزمایش قرار دهد
- یک نیاز مهم کاربر طی دو هفته انجام گردیده و پس از تحویل به مشتری نیازمندی دیگری در زمانبندی قرار میگیرد. و درنهایت نیازها با هم یکپارچه میشوند. **Merge**
- بعلت ساده شدن مسائل ، فعالیت های چارچوب نیز به سرعت انجام میگیرد.
- در این نوع روش راهنمایی ها و تکنیکهای مدیریتی بیشتر مد نظر است .
- سنجش روند توسعه پروژه و زمانبندی فعالیت ها از نکات مدیریتی نیز میباشد.

روش مدل سازی سریع 7

Agile Modeling



- برای سیستمهای بحرانی و یا پیچیده تجاری کوچک و بزرگ بر اساس یک مدل سریع استفاده میگردد.
- مدل سازی باید به گونه ای انجام گردد که برای همه افراد قابل درک باشد و مسئله به شکل قابل موثری تقسیم بندی میشود و کیفیت نیز باید تضمین گردد.
- این روش بر مبنای مستند سازی موثر سیستمها میباشد

- در این روش یک هدف کلی داریم
- استفاده از مدل های چند گانه
- باید آینده نگری وجود داشته باشد بدین معنی که مدلی ارزشمند است که بتوان در بلند مدت از آن استفاده نمود
- هر محصولی باید بتواند تغییرات را نیز پشتیبانی کند.
- محتوا مهمتر از ارائه میباشد بدین معنی که اگر یک محتوی عالی با کد کمی بد ارائه شود بهتر از محتوی متوسط با کد عالی میباشد.
- بهتر است از مدل ها و ابزارهای شناخته شده استفاده نمود.
- سازگاری محلی به معنی سازگاری با نیاز های تیمی مهم میباشد.

مرور کوتاه بر روش های سریع توسعه نرم افزار

- فلسفه مهم بودن سرعت در تولید نرم افزار
- تاکید بر تحویل سریع نرم فزار
- ارتباط و همکاری مدام کاربر با تیم تولید نرم افزار
- تیم ها بصورت خود سازمانده میباشند
- تطابق پذیری با تغییرات آتی مهم وضروری میباشد.

-از مدل های سریع ذکر شده میتوان ۱- به روش **XP** اشاره نمود که گسترده ترین میزان استفاده رادارد ۲-روش **ASP** با تاکید بر همکاری تیمی ۳- روش **DSDM** که حاوی سه ویژگی و قانون پرتو در چرخه حیات خود میباشد. ۴- روش **Scrum** با تاکید بر مجموعه ای از **Process pattern** ها ۴- روش کریستال که از خانواده های مدل فرایندی و قابلیت مانور بر روی خصوصیات استفاده میکند ۵- روش ویژه گی محور ۶- روش مدل سازی سریع

تاریخچه CASE

CASE (computer-aided software engineering)

- CASE از دهه ۷۰ ،زمانیکه شرکتها شروع به قرض گرفتن ایده هایشان از فرآیند ساخت سفت افزار و بکار بستن آن در توسعه نرم افزار نمودند نشات میگیرد.
- در سال ۱۹۶۸ در دانشگاه میشیگان پروژه ای تعریف میگردد به نام Information System Design by Optimization System : ISDOS
- پروژه ISDOS باعث میشود علاقه زیادی به سمت استفاده از ابزارهای کامپیوتری در کمک کردن به تحلیلگران در فرایندهای پیچیده و توسعه سیستمها بوجود می آید.
- پروژه ISDOS با استفاده از ابزار Problem Statement Language/ Problem Statement Analyzer (PSL/PSA) که خود بعنوان شروع یک ابزار Case میباشد.
- مقارن با ISDOS موضوع توسعه Data dictionary در Database ها نیز گسترش میابد و استفاده از Active dictionary بطور مثال در جهت استفاده مستمر از یک ویژه گی در آینده . اگر چه ممکن است رابط گرافیکی وجود نرشته باشد ولی بین متا دیتا ها رابطه وجود دارد.
- در سال ۱۹۸۲ یک شرکت کامپیوتری ابزار CASE را با ارائه نرم افزار Graphic CASE در باز بسط میدهد که بصورت ذاتیGrahic CASE و Text editor را با هم تجمیع مینماید .
- در اوائل دهه ۹۰ شرکت IBM اتحادی را برای فروشندگان نرم افزار فراهم میکند
  ۱- استفاده از IBM Repository بر اساس دیتابیس DB2
  ۲-استفاده از Mainframe و OS2
- در جهت روند تکاملی CASE tools استفاده از مترها و ارزیابی شی گرایی مطرح میگردد که با پشتیبانی شرکت OMG : Object Management Group و ارائه UML تکامل خود را طی میکند.

> - CASE فرایند توسعه نرم افزار با استفاده از یک پشتیبانی خودکار و اتوماتیک میباشد.و CASE Tools ها ابزارهایی هستند که به فرآیند تولید نرم افزار کمک میکنند و از آن پشتیبانی مینمایند.

مزایای CASE

- باعث افزایش کیفیت کار میگردد.
- باعث نظم در امور و هماهنگی مناسب تر بین اعضاء تیم میگردد.
- کلیه اطلاعات بصورت گرافیکی نمایش داده میشود که در مجموع به درک ساده تر فرآیند می انجامد.
- همه اطلاعات بصورت مرکزی در یکجا جمع آوری میگردد
- مجموع همه موارد گفته شده منجر به بهبود کیفیت و بهره وری در توسعه نرم افزار میگردد.
- تولید کنندگان CASE انتظارات بیشتر از چیزی که در عمل ثابت شد را داشتند و به دو علت به پیشبینی های اولیه دست نیافتند.
  ۱-مسائل موجود در توسعه نرم افزار مانند مسائل مدیریتی قابلیت اتوماتیک انجام شدن را زیاد ندارند .
  ۲- سیستمهای CASE خیلی هم یکپارچه نمیباشد و به مسائلی از قبیل هزینه و ... برمیگردد.
- ابزار Case باعث کاهش زمان و تلاشی که انجام میدهیم میشود. و کارها خیلی سریعتر و کاملتر انجام میشوند
- امکان استفاده مجدد از Component ها در مدل ها میسر میگردد
- کاهش یافتن هزینه های نگهداری نرم افزار

معایب CASE

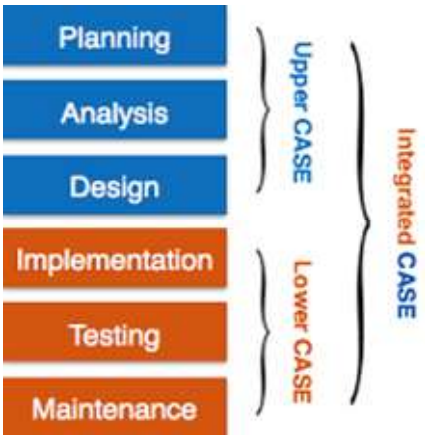
- محدودیت در انعطاف پذیری مستندات بطوریکه ممکن است Template ای که توسط CASE فراهم میگردد با نیازها و Template های آن سازمان همخوانی نداشته باشد و امکان انعطاف پذیری مناسب نیز در جهت تغییرات میسر نباشد.
- استفاده از همه عناصر در یک CASE مانند UML حتی با رعایت همه قواعد مرتبط ، بصورت صد در صدی ، تامین کننده همه نیازهای مورد انتظار نیست.
- هزینه خرید و آموزش CASE Tools ها باید مد نظر قرار داد. (حدود ۵ الی ۱۵۰۰۰ دلار در سال هزینه خرید و پشتیبانی)

CASE Tools دسته بندی

Tools 1 **Workbench** 2 **Environment** 3

1 ابزارها
Tools

- ابزارها بیشتر به فاز تحلیل نیاز ها باز میگردند .
- امکاناتی هستند که فرایندهای منفرد Individual را پشتیبانی میکنند
- بطور مثال وقتی یک برنامه را میخواهیم کامپایل کنیم یا مقایسه کردن نتایج تست را انجام دهیم از یک وظیفه در چرخه حیات تولید نرم افزار پشتیبانی میشود.



CASE Tools

1 Upper CASE Tools
Front-End Case Tools

- به توسعه دهنده در مشخص نمودن ، تحلیل و طراحی Work Flow کمک میکند
- در این زمینه به Dataflow diagram,structure charts , decision table , decision tree میتوان اشاره نمود.

2 Lower CASE Tools
Back-End Case Tools

- مرتبط با پشتیبانی از فرآیندهای Implementation ,test و نگهداری جریان داده میگردند .
- شامل فعالیت های پایه ای که به سیستم نزدیک تر است را انجام میدهند.

3 Integrated CASE Tools

- شامل پشتیبانی از تمام چرخه حیات نرم افزار و کارائی بیشتراز طریق Merge ابزارهای مختلف به روش های گوناگون که حاوی ۵ سطح مختلف میگردد.

Integrated CASE Tools

1 Platform integration

- سطح اول شامل تجمیع Platform ها میباشد. بدین شکل که ابزار ها بر روی یک Platform سفت افزاری یا نرم افزاری اجرا میشوند . بطور مثال استفاده از یک قابلیت خاص بر روی سیستمهای عامل Linux و Windows

2 Data integration

- از یک مدل به اشتراک گذاشته شده و بر اساس یک فرمت و ساختار داده مشترک داده استفاده میشود. .
- بعضی اوقات یک مخزن داده Data Repository نیز برای استفاده مشترک داده وجود دارد.

3 Presentation integration

- از یک رابط کاربری و یک اینترفیس مشترک برای بکارگیری از ابزار استفاده میشود.

4 Control integration

- از یک شیوه رشته کنترلی مشترک استفاده میگردد.

5 Process integration

- ابزار ها از یک راهنمایی و خط مشی که Process model ارائه نموده است استفاده میکنند و تجمیع میگردند.

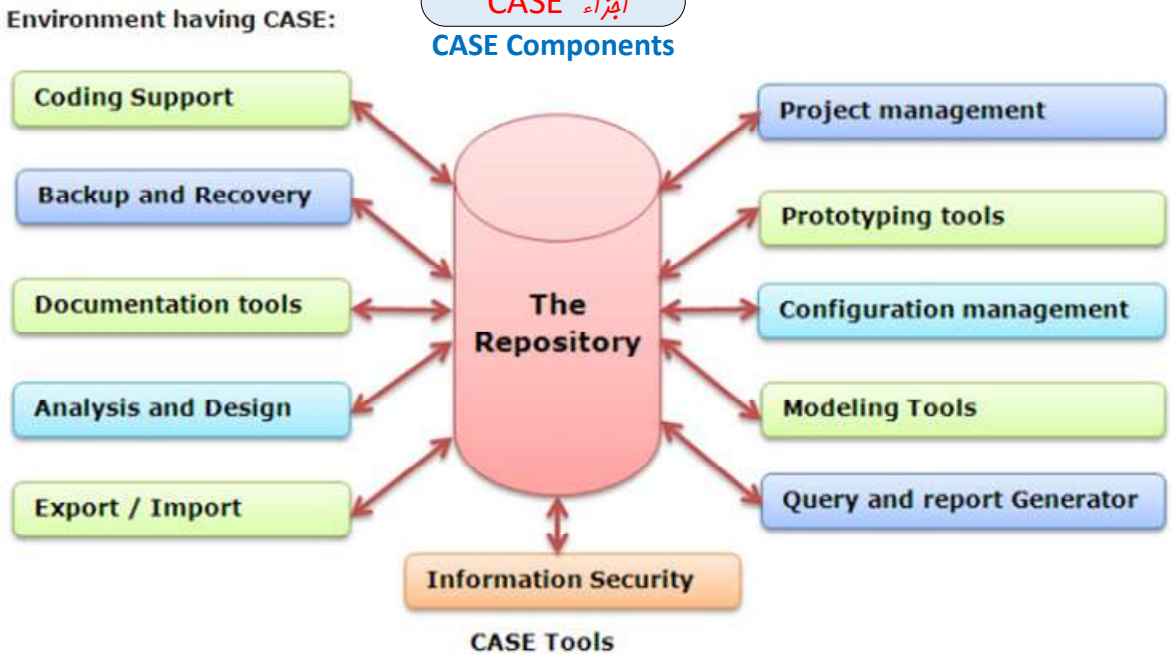
2 میز کار
Workbench

- میزکارها به غیر از فاز تحلیل نیاز ها شامل فاز طراحی نیز میگردند.
- حالتی که چند ابزار با تمرکز بر روی یک بخش خاصی باهم ترکیب شده و تشکیل یک میز کار را میدهند.
- مجموعه ای از ابزار ها که میتوانند جریان فرآیند ، نیازمندیها و طراحی را پشتیبانی کنند .
- ابزار Software Architect,Power builder نمونه هایی از میز کار هستند که چند قسمت از فرایند تولید نرم افزار را پشتیبانی میکنند.

3 محیط های کاری
Environment

- در اینحالت چرخه کامل حیات نرم افزار یا حداقل بخش بزرگی از آن پشتیبانی میگردد.
- مجموعه ای از چند Workbench میزکار که با یکدیگر ترکیب شده اند.
- ممکن است در یک محیط کاری هم Oracle وجود داشته باشد و برای پروژه خاصی هم از DB2 استفاده شود ، در این حالت میتوان از Integrated development Environment استفاده نمود که حاوی ابزار نقاشی و تهیه گزارش و غیره را دارد مانند Rational Rose
- یک ابزار میتواند زیر مجموعه ای از چند ابزار که همان میز کار است باشد یا زیرمجموعه ای از یک IDE که مجموعه ای از چند میز کار است باشد.

اجزاء CASE
CASE Components



1 مخزن اطلاعات
CASE Repository

- اصلی ترین جزء CASE Repository میباشد که مخزن اطلاعات میباشد و هسته اصلی و مرکزی CASE را تشکیل میدهد.
- یک پایگاه داده مرکزی که اجازه میدهد اطلاعاتی که از آن در چرخه حیات نرم افزار استفاده میشود در یکجا مجتمع گردند .
- از CASE Repository با نام های Information Repository و Data Repository نیز یاد میگردد.
- از جنبه تجمیع اطلاعات سازمان در یکجا و فراهم شدن امکان کنترل و مدیریت آن با نام Information Repository نیز شناخته میشود.
- از جنبه ماهیت پایگاه داده (ذخیره ، بازیابی ، پردازش و تصحیح داده ها) با نام Data Repository نیز شناخته میشود.
- فازهای اولیه که در مخزن داده جمع میشوند بصورت Diagram,Dataflow و Prototype هایی هستند که در فاز طراحی و تحلیل نیازمندیها تولید میگردند و این گونه اطلاعات بسیار ارزشمند هستند و باعث ساده تر شدن مدیریت پروژه و استفاده مجدد از آن میگردند.

2 ابزار طراحی نمودار
Diagram Tools

- باعث نمایش سیستم در سطح گرافیکی میگردد و یک سطح بلاتری از پردازش ها مشخص میشود
- باعث بالا رفتن کیفیت و درک بهتر از سیستم در حالت دیداری میگردد.

3 ابزار تولید گزارش و تجزیه تحلیل
Report & Analytic Tools

- ابزارهای گزارش و تجزیه تحلیل باعث بالا رفتن دقت ، کیفیت و تصحیح مسیر چرخه حیات میگردند.
- بعنوان یک ابزار کنترلی ساختار داده و کاربرد استفاده میشود.
- از ابزار Document Generation نیز برای تولید مستندات باکیفیت و استاندارد شده استفاده میگردد که حدود ۸۰ درصد نگهداری های سیستم راحت تر میشود

برخی ابزارهای دیگر

CASE Drawing, Design,Analysis ,Database Generator, Code Genarator, Verision Control , Backup & Security

- از مجموعه ابزارهای میز کار که بخشی از چرخه حیات را کنترل میکنند میتوان به **....,** Code Generator , Data Dictionary اشاره نمود. که این نشان دهنده همپوشانی ابزارها در در سه سطح ابزاری ، میزکار و محیط میباشد.

تکنولوژی های CASE

- از دیدگاه تکنولوژی های CASE نیز سه دسته بندی برای بهینه نمودن توسعه نرم افزار متصور است.

دسته بندی CASE از دیدگاه تکنولوژی

CASE Technology & Categories

- این نوع تکنولوژی از فرآیند توسعه نرم افزار حمایت میکند و شامل ابزار های توسعه نرم افزار میباشد.
- این نوع تکنولوژی مرتبط با بخش مدیریت فرآیند هستند و مدل سازی مدیریت فرآیند را پشتیبانی میکنند.
- این نوع تکنولوژی شامل مولد های ابزار CASE هستند

تقسیم بندی CASE از دیدگاه کاربردی

- در اینحالت ابزارها براساس عملیاتی Function که انجام میدهند طبقه بندی میگردد.

تقسیم بندی CASE از دیدگاه از نظر فرآیند

- در اینحالت ابزارها براساس فرآیندی Process که انجام میدهند طبقه بندی میگردد.

تقسیم بندی CASE از دیدگاه از نظر تجمیع

- در اینحالت ابزارها براساس تجمیع Integrated که شده اند طبقه بندی میگردد.

- این امکان وجود دارد که به صورت سفارشی تقسیم بندی انجتم میگردد مثلا بر اساس Editing Tools , Planning Tools ,... که از نقطه نظرعملیاتی تقسیم بندی میگردد نمونه های دیگری هم امکان پذیر است مانند Prototype Tools یا Activity Tools

چرخه حیات CASE Tools

CASE Tools Life Cycle

- چرخه حیات یک CASE Tools شامل شش CASE Life Cycle Model میگردد .

1 فاز خریداری و تهیه Case Tools

Purchase

- بر اساس استانداردهای موجود در شرکت ابزارهای پشتیبانی از استانداردهای سفت افزاری موجود شناسایی میگردد
- پیش بینی از سفت افزار و آینده نگری در زمینه CASE Tools منجر به تهیه یک Environment میشود که با سفت افزار هماهنگ باشد
- ابزاری که خریداری میگردد باید با برنامه های کاربردی و نرم افزار های موجود در آن شرکت نیز سازگاری داشته باشد.
- ابزاری که خریداری میشود باید بتواند در ازای راحت تر کردن توسعه نرم افزاربتواند در حد معقولی از امنیت داده موجود در شرکت پشتیبانی نماید.

2 اندازه نمودن و همگن نمودن ابزار خریداری شده با شرکت

Tailoring

- نصب و راه اندازی نرم افزار (ابزار) خریداری شده بر اساس Platform شرکت
- انجام تنظیمات Configuration و سفارشی سازی Customize بر اساس تجمیع ابزارهای مورد نظر
- تهیه مستندات Documentation جامع و دقیق از کاری که تا این مرحله انجام شده بعنوان پیش نیاز استفاده از آن در مراحل دیگر.

3 فاز مقدماتی

Introduction

- نیازمندی ها و تغییرات و هماهنگیهای مورد نیاز برای کارائی سیستم توسعه نرم افزار مشخص میگردد بعنوان مثال آموزش افراد ، معرفی سیستم به کاربران .
- ممکن است پذیرش یک ابزار جدید در شرکت مقاوتهایی را بوجود بیاود و یا تعارضاتی از این قبیل که این نوع ابزارها بیشتر بدرد استفاده مدیریتی میخورند.

4 فاز عملیاتی و تکاملی

Operation

- سیستم از این به بعد وارد فاز تکاملی میشود و مجموعه تغییرات و نیازمندیها تا این مرحله کامل شده است ، هرچند که فرآیند تغییرات همواره میتواندبرای تصحیح نیازهای قبلی یا یک نیاز جدید وجود داشته باشد.

5 بودجه بندی

Budget

- با توجه به گران بودن هزینه ای ابزارها و پشتیبانی از آن ها بودجه بندی میتواند حائز اهمیت باشد.

6 منسوخ شدگی

Extinct

- یک ابزار با توجه پیشرفت علم و نیازهای فرآیند تولید نرم افزار پس از یک مدتی دچار کهنگی و منسوخ شدن میشود.
- در حالت منسوخ شدگی راه کارهای متعددی از قبیل بروز رسانی و تغییر پلت فرم ابزارها و یا انتقال به محیط دیگر و انجام تغییرات وجود دارند.

شی گرائی

Object oriented

– در برنامه نویسی شی گرا **Object oriented programing** داده ها در حد امکان در قالب شی قرار میگیرند و توابعی که در بیرون این محیط هستند قالبا توانایی تغییر شی را ندارند.
– مفاهیم شی گرایی به سالهای ۱۹۸۰ بر میگردد و از ایده آن از دیده شدن اشیاء توسط افراد گرفته شده است. بطور مثال وقتی از صندلی صحبت میشود ما یک دید کلی از صندلی داریم و فقط ویژه گیهای آن مانند رنگ ، جنس و .... ممکن است تغییر کند. در واقع یک کلاسی به نام صندلی در ذهن ما وجود دارد و با همان مفهوم شی گرائی یک صندلی نیز خود زیر مجموعه گروهی به نام اسباب و اثاثیه میباشد که دارای یک سری ویژه گیهای مشترک هستند

– در مثال بالا وقتی از صندلی صحبت میکنیم ، میتواند دارای یک سری ویژه گیهای منحصر بفرد نیز باشد و ضمن اینکه امکان دستکاری یک شی نیز وجود دارد مثلا بطور تلویحی خرید و فروش صندلی و تغییر و جابجایی آن . پس کلاس صندلی مجموعه ای است از خود صندلی بعلاوه صفت های آن و کارهایی که میتوان بر روی آن انجام داد و همه این موارد دسته بندی و کپسوله گردیده و تولید یک شی میشود.

– در مثالی دیگر فرض میکنیم میخواهیم یک تاریخی را نگه داریم که دارای روز ، ماه و سال میباشد که فضای کمی را در حافظه اشغال خواهد نمود ، اگر بخواهیم **N** تاریخ را در حافظه نگه داریم به **N** متغییر با اسامی جدید و فضای حافظه زیادی نیاز خواهیم داشت. بنابراین از کلاس تاریخ استفاده میکنیم که میتواند تعداد زیادی تاریخ را در زیر مجموعه خود داشته باشد.

محاسن شی گرائی

۱– کم شدن پیچیدگی ۲– هزینه کم تر ۳– امکان دسترسی سریعتر به داده ها ۴– پایین آمدن خطا نسبت به سایر روش ها ۵– ساده تر شدن مدیریت تغییرات در سیستم ۶– قابلیت استفاده مجدد در سیستم از طریق کلاس

ویژه گیهای شی گرائی

– یک کلاس جدید از کلاس قبلی دارای کلیه خواص عمومی و توابع کلاس قبلی بعلاوه و یژه گیهای کلاس جدید میباشد. به این خاصیت ارث بری میگویند. و اگر تغییری در توابع یا خاصیت های های یک کلاس بالاتر ایجاد گردد بطور اتوماتیک به تمام کلاسهای زیر مجموعه انتقال پیدا میکند.

① Inheritance

– وراثت یا ارث بری از مفاهیم اساسی برنامه نویسی شیءگراست. هر شیء یک نمونه از یک کلاس است و هر کلاس می‌تواند از کلاس یا کلاسهای دیگری مشتق شده باشد)خواص متدها یا رویدادهای کلاس‌های دیگر را به ارث ببرد(. در یک مثال ساده می‌توان اتومبیلی را در نظر گرفت که برای جلوگیری از بازنویسی خواص عمومی اتومبیل شامل: چهار چرخ، متدهای حرکت چرخ، متد چرخاندن فرمان، فرمان، بدنه، در و غیره، می‌توان یک کلاس پایه از اتومبیل ایجاد کرد سپس مثلاً برای اتومبیل سیتروئن مدل **C5** یک کلاس جدید ایجاد کرده که خواص، متدها و رویدادهای عمومی اتومبیل را داشته باشد و فقط برای خواص، متدها و رویدادهای جدید این اتومبیل کد نوشته شود. این ویژگی باعث صرفه‌جویی در نوشتن کد و تا حدودی تضمین صحت کد موجود می‌شود. به عنوان مثال اگر کلاس پایه مشکلی داشته باشد فقط کافی است کلاس پایه تغییر داده شود و در تمامی کلاس‌هایی که از این کلاس پایه ویژگی‌ای ا به ارث برده‌اند این تغییر اعمال خواهد شد.

– کپسوله کردن یعنی بعضی از خصوصیات یا رفتار یک شی از دیگر اجزاء میتواند مخفی گردد. بدین معنی که اشیاء بدون اطلاع از عملکرد یک دیگر با هم میتوانند کار کنند و تعامل داشته باشند مثلا در شماره حساب بانکی یک فرد دیگر افراد با داشتن شماره حساب به جزئیات حساب دسترسی ندارند.

② Encapsulation

کپسوله سازی، مخفی‌سازی ، یا لفافه‌بندی، به این مفهوم اشاره دارد که باید بعضی خصوصیات یا رفتارهای شیء را از دید دیگران پنهان کرد. فرض کنید شما به عنوان یک راننده می‌خواهید اتومبیل شخصی تان را روشن کنید و به محل کارتان عزیمت کنید سوییچ را بر می‌دارید، استارت می‌زنید و با فشار آوردن به پدال‌های گاز و ترمز و چرخش فرمان و ... به محل کارتان خواهید رسید. در طول این مسیر در اتومبیل شما به عنوان یک شیء اعمال مختلفی در حال انجام بود. مثلاً لنت ترمز به دیسک چرخ برخورد می‌کرد و باعث کم شدن سرعت می‌شد یا شمعها شروع به جرقه زدن در زمانی بخصوصی می‌کردند. حال فرض کنید که عمل مخفی سازی وجود نداشت و شما مجبور بودید که جرقه زدن شمعها را کنترل می‌کردید و به سیستم سوخت رسانی در موقع لزوم دستور می‌دادید که سوخت را بداخل سیلندرها ارسال کند و... یا هیچگاه به مقصد نمی‌رسیدید! یا سالم نمی‌رسیدید!

③ Polymorphism

چند ریختی، کمیتی است که به یک رابط امکان می‌دهد تا از عملیات یکسانی در قالب یک کلاس عمومی استفاده کند. عمل خاص کلاس را ذات حقیقی شیء تعیین می‌کند. مثال ساده‌ای از چند ریختی، فرمان اتومبیل است. عمل فرمان اتومبیل برای تمام اتومبیل‌ها بدون توجه به ساز و کاری که دارند، یکسان است. فرمان برای اتومبیل که به طور مکانیکی کار می‌کند یا با نیروی برق یا هر چیز دیگری، عمل یکسانی را انجام می‌دهد؛ بنابراین، پس از اینکه شما عملکرد فرمان را یاد گرفتید، می‌توانید فرمان هر اتومبیلی را کنترل کنید. همین حرف در برنامه نویسی نیز اعمال می‌شود. به طور کلی، مفهوم چند ریختی، اغلب با عبارت (یک رابط، چندین روش) بیان می‌شود. این بدین معنی است که امکان طراحی رابط عمومی برای گروهی از عملیات مرتبط وجود دارد. چند ریختی یا چندشکلی به این معنا است که اشیاء می‌توانند در موقعیت‌های مختلف، رفتارهای متفاوتی بروز دهند. مثلاً یک تابع در صورتی که بر روی نمونه‌ای از کلاس آ فراخوانی شود، رفتار ب را بروز دهد در حالی که اگر بر روی کلاس ج (که فرزند کلاس آ است) فراخوانی شود، رفتاری متفاوت انجام دهد.

– شی گرائی شامل (۱– تحلیل ۲– طراحی ۳– برنامه نویسی) میشود . تحلیل شی گرائی به دیدگاه شی گرائی بر میگردد.

UML

The Unified Modeling Language

- **UML** یک زبان مدل سازی یکپارچه میباشد. و از خانواده زبان هایی میباشد که با نماد گرافیکی جهت تشریح و توصیف جنبه های مختلف نرم افزاری کار میکنند.
- **UML** بر اساس شی گرایی میباشد و شامل یک **Modeling Language** میگردد.و در زبان هایی که شی گرائی را پشتیبانی نمیکنند نیز قابل استفاده میباشد.
- قبل از **UML** ساختارهای برنامه نویسی دیگری نیز وجود داشته است که یکی از مهمترین ها **SSADM (Structured Systems Analysis & Design Method)** میباشد. که به متد تحلیل و طراحی سیستم بصورت ساختاریافته مشهور میباشد.

جدول مقایسه زبان UML و متدلوژی SSADM

Structured Systems Analysis & Design Methodology		The Unified Modeling Language
SSADM	VS	**UML**
Non-Object oriented		Object oriented
Data •Logical Data Model (LDM)	1	**Data** •The class diagram
Events •Entity Life History (ELH)	2	**Events** •The Behaviour (interaction) diagrams
Process •Data Flow Diagram (DFD)	3	**Process** •The activity diagram
Interfaces •Dialogue Design	4	**Interfaces** •Modelled in class and component diagrams.
Resources •Requirements Catalogue (RC)	5	**Resources** •Modelled by using the stereotype feature.
Quality •Requirements Catalogue (RC)	6	**Quality** •A. In analysis explorative prototypes. •B. In design experimental prototypes
Business issues •A. Data Flow Diagrams (DFD). •B. Entity Life History (ELH).	7	**Business issues** •Activity diagrams describe and model business process.
Identify the problem or problem objectives •The strategic planning defines the problem that needs to be solved.	8	**Identify the problem or problem objectives** •The strategic planning defines the problem that needs to be solved.
User involvement •A. Gathering information about system. •B. Reviewing products of each stage.	9	**User involvement** •A. Gathering information about system in use case models, CRC and tech. dictionary. •B. Review/check prototypes
Organisational structure, goals and policies •Strategic planning looks at organisational structure giving Project Initial Document.	10	**Organisational structure, goals and policies** •Activity diagram models organisational structure and integration.
Employee job satisfaction •User may choose Business System Option (BSO) that defining impact on users and training.	11	**Employee job satisfaction** •Allowing employees to choose suitable way to perform assigned job.
Different point of views •Different views of the system are documented in Requirements Catalogue.	12	**Different point of views** •Analyst considers different views of system and resolves contradictions.
Employee values •Not Supported.	13	**Employee values** •Not Supported.
System acceptability and usability •A. User involvement in developing system. •B. Use prototype. •C. Study of system impact on staff.	14	**System acceptability and usability** •Involvement of users in experimental prototypes to verify usability/acceptability of system.

- **UML** شامل یک **Open standard** میباشد که توسط **Object management group (OMG)** راهبری میگردد و به زبانهای برنامه نویسی نیز وابستگی ندارد و مستقل میباشد و بطور خاص برای نرم افزارهای **Object orinted** بهترین کاربرد را دارد.

- **UML** در واقع **Case tools** نیست و بیشتر ساختار معنایی دارد و به نوعی مرحله قبل از پیاده سازی را انجام میدهد هرچند ابزارهایی شبیه **Rational rose** امکان تولید کد را بعد از طراحی در اختیار کاربر میگذارند.

محاسن UML

1. - چون شامل یک استاندارد باز و گرافیکی میباشد امکان استفاده بهینه از این ابزار برای همگان میسر خواهد بود.
2. - امکان توصیف و درک نرم افزار طراحی شده برای کاربر راحت تر خواهد بود.
3. - از طراحی ابتدایی سیستم تا جزئیات ریز موجود در چرخه حیات نرم افزاری امکان ستفاده از **UML** میباشد.
4. - استفاده از **UML** به علت ویژه گیهای موجود در آن درک متقابل خوبی بین مشتری **Customer** و تولید کننده **Developer** بوجود می آید.
5. - تنوع پشتیبانی از محیط های کاری متنوع بسیار زیاد میباشد و امروزه خیلی از ابزار های نرم افزاری از **UML** پشتیبانی میکنند.
6. - **UML** زبانی است که بر اساس تجربه و نیاز تولید شده است و به همین علت دارای هماهنگی بالایی با طراحی نرم افزار و راحتی استفاده را دارا میباشد.

تاریخچه UML

- روش های سنتی برنامه نویسی از سال های ۱۹۷۰ شروع گردیده و در سال ۱۹۸۰ همه گیر شده است. در همین زمان از متدولوژی هایی مانند **SSADM** استفاده میشده است.

- در اوایل دهه ۸۰ که مبحث شی گرائی مطرح میگردد زبان های برنامه نویسی شی گرا بصورت محدود به کار گرفته میشود و طی مرور زمان همه گیر میشود.

- اولین زبان شی گرائی که همه گیر شد **Smart Talk** بود که در سال ۱۹۸۰ عرضه گردید.

- بعد از آن در سال ۱۹۹۰ زبان های شی گرائی توسط مهندسین **(Bosch & Rumble)** در زمینه طراحی شی گرائی و متدلوژی شی گرائی نیز اضافه شده و بسط داده میشود.

- در سال ۱۹۹۴ آقای بوش که در شرکت **Rational** کار میکردند و پیوستن آقای رامبل که در شرکت **IBM** کار میکرد به شرکت رشنال و کار مشترک این دو بر روی بحث شی گرائی منجر به مدل جدیدی که بر اساس تکنیکهای جدید مدل سازی بود میگردد در سال ۱۹۹۵ مجموعه کار گروهی با دیگر اعضاء به تولید یک برنامه مهندسی شی گرا **Object oriented Software Engineering** با مخفف **OOSE** میگردد. سپس شرکت **OMG** خواستار استاندارد شدن **OOSE** میشود و در سال ۱۹۹۷ مجموعه استانداردها تایید میگردد و عملا زبان برنامه نویسی **UML** متولد میگردد و به تدریج نسخه های جدیدتر آن توسط **OMG** به بازار عرضه میشود.

عناصر UML

Model ① - مدل یک مفهوم انتزاعی است از بخشی از سیستم که موجود است.

View ② - مشخص کننده جنبه هایی از مدل میباشد که از زاویه خاصی به آن نگاه و بررسی میکند. مثلا نگاه به سیستم سوخت گیری در شبیه سازی یک ماشین

1. **Logical View** - نگاه از زاویه دید کاربر **End User** به یک سیستم
2. **Implemetation View** - نگاه از زاویه دید یک برنامه نویس **Programmer** به سیستم.
3. **Deployment View** - نگاه از زاویه دید مهندس سیستم به سیستم.
4. **Process View** - نگاه از زاویه دید فرآیند به سیستم.

- همه سیستمها نیاز به همه **View** ها ندارند بطور مثال اگر در سیستم **Single Processor** کار میکنیم **Deployment & Process** را نیاز نداریم یا وقتیکه برنامه یا سیستم ما خیلی کوچک است **Implementation view** مورد نیاز نمیباشد . در حقیقت با توجه به سختی و پیچیدگی یک سیستم از **View** های مختلف جهت واضح شدن یک موضوع استفاده میشود.

- ممکن است با توجه به پیچیدگی مسئله و اهمیت آن به **Security View** ⑥ یا **Data View** ⑤ یا غیره نیاز پیدا کردد.

تقریبا برای ۸۰ درصد بیشتر مسئله ها با دانستن ۲۰ درصد UML امکان پشتیبانی و پوشش میسر خواهد شد

- **UML** یک زبان **Diagram Base** میباشد و همه چیز بر اساس شکل ها توضیح داده میشود. و این دیاگرام ها به سه شکل مختلف میتواند ترسیم گردد.

Structural Diagrams ① نمودارهای ساختاری

- این نمودار ها بر روی ساختار سیستم و چیزی که باید در سیستم وجود داشته باشد تاکید مینماید و بر روی معماری سیستم بیشتر تاکید دارند.که شامل زیر شاخه های متعددی ازقبیل **Deployment Diagram , Package diagram**

Behavior Diagrams ② نمودارهای رفتاری

- این نمودارها بر اساس ترکیب چیزهایی میباشد که در سیستم باید اتفاق بیافتد و کارکردهای سیستم را توصیف میکنند مانند نمودار فعالیت **Activity Diagram** یا **Usecase Diagram**

Intraction Diagrams ③ نمودارهای برهم کنش

- این نمودارها بر روی گردش داده ها در قسمت های مختلف سیستم تاکید دارند مانند **Sequence Diagram,Timing Diagram,Communication Diagram** و مشخص میکند در داخل سیستم چه اتفاقهایی میافتد و اجزاء چگونه باهم ارتباط برقرار میکنند و طول عمر آنها چقدر است یا چه دیتایی چه ورودی و خروجی دارد

نمودارهای رفتاری
Behavior Diagrams

نمودار از دید کاربر 1
Usecase Diagram

- این نمودار از دید کاربر سیستم را توصیف میکند و مشخص مینماید کاربر با سیستم چه کاری انجام میدهد.
- در این نمودار نیازمندیها و فعالیت های سیستم از نگاه کاربر را مشخص مینمائیم. و درواقع سناریو سیستم از طریق **Usecase** مشخص میگردد.
- در این نمودار دو گزینه اصلی وجود دارد.

1 Usecase

- مشخص میکند چه کاری را کاربر با سیستم انجام میدهد. و معمولا با یک بیضی نمایش داده میشود.
- ما در **Usecase** به دنبال نیازمندیهای کاربر هستیم .

2 Actor

- یک ماهیت خارجی در سیستم است که از سیستم خروجی دریافت میکند.
- باید مشخص شود هر کاربر از سیستم چه میخواهد

- بطور مثال در سیستم دانشگاه یکی از **Actor** ها دانشجو میباشد و **Usecase** او مجموعه کارهایی در سیستم است که منجر به انتقاب واحد میگردد و هدف بر این است که تمام نیازهای کاربر و پاسخ های سیستم مشخص گردد
- **Usecase diagram** نقطه ورود به سیستم در نظر گرفته میشود که باید مورد تجزیه و تحلیل قرار گیرد.
- **Usecase diagram** فاز اولیه ورود به سیستم میباشد

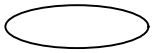
> **Actor** یا عامل کسی است که از سیستم استفاده میکند و **Usecaser** سناریوهایی میباشد که در سیستم اتفاق میافتند تا نیازمندیهای **Actor** را فراهم کند.
>
> درحالت کلی **Actor** میتواند یک فرد و یا یک سیستم دیگر و یا حتی محیط فیزیکی مانند زمان باشد.

سعی بر این است که سیستم طوری طراحی شود تا تعداد **Usecase** ها خیلی زیاد نباشد تا پیچیدگی کمتر شود(افزایش های کوچک) و این از اصول طراحی سریع نرم افزار میباشد.

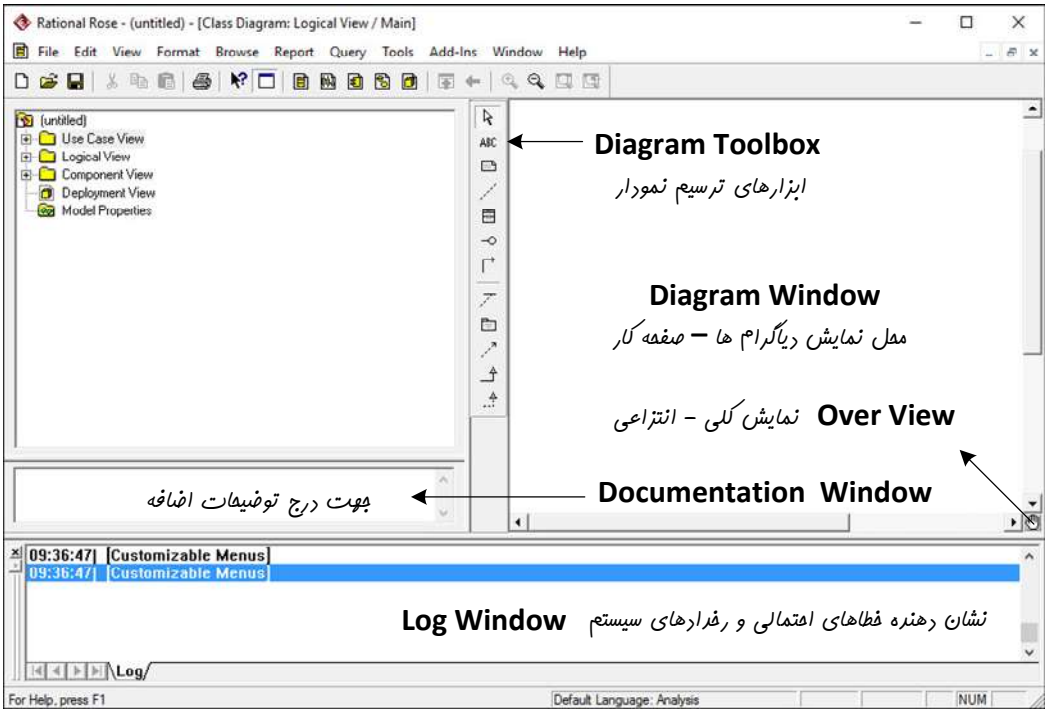
- **Usecase Digram** ها جهت ارتباط با مشتری و تحلیل نیازمندی ها بشکل قابل فهم و درک مورد استفاده قرار میگیرد.

- دو ارتباط مهم بین **Actor** و **Usecase**

Extend Relation ⟷ یک ارتباط انتخابی وجود دارد و میتواند انتخاب نگردد.



Include Relation ⟷ یک وابستگی واجبی بین دو رابطه وجود دارد.اگر یکی نباشد دیگری فعال نمیشود.



1 Usecase View

شامل نمای مورد استفاده سیستم از دید کاربر میباشد. و شامل **Usecae,Actor** و رابطه **Association** بین آنها میباشد.

1-Usecase Diagram
2-Sequence Diagram
3-Collaboration Diagram
4-Activity Diagram

2 Logical View

شامل نمای منطقی سیستم میباشد.و مربوط به نقشه سیستمی میباشد که قرار است پیاده سازی گردد و شامل **Class,Attribute, Method** میگردد

1-Class Diagram
2-Statechart Diagram

3 Component View

شامل یک سری کدهای فیزیکی سیستم و کتابخانه های پیاده سازی میباشد

1-Component Diagram

4 Deployment View

شامل ساختار سخت افزار فیزیکی میباشد در حالتی که در جاهای مختلف قرار گرفته است.

(Development Process)

1 فرایند توسعه (ساخت محصول)

- شناسایی مدل ها (بصورت سطح بالا) **(Use Case Diagram)**
- شناسایی اشیاء مهم و ارتباطات آنها **(Class & Object Diagram)**
- ایجاد سناریو برای موارد کاربرد **(Sequence & Collaboration Diagram)**
- توسعه (تعمیم) سناریوها برای توصیف رفتار **(State & Activity Diagram)**
- پالایش (اضافه) کردن جزئیات پیاده سازی **(Component & Deployment Diagram)**

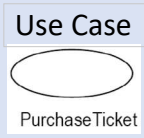
2

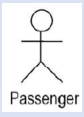
Usecase View – در اینحالت بخشی از مدل مد نظر است.

– یافتن **Usecase** ها : چه سرویسهایی در سیستم ارائه میشود (مورد استفاده)

– یافتن **Actor** ها (عامل ها ، بازیگران)

هر **Case Use** از تعامل اشیاء چندین کلاس با یکدیگر شکل میگیرد.بعنوان مثال سرویس ثبت نام از تعامل اشیاء کلاسهای دانشجو ،درس ،استاد ،فضای آموزشی و...شکل میگیرد.

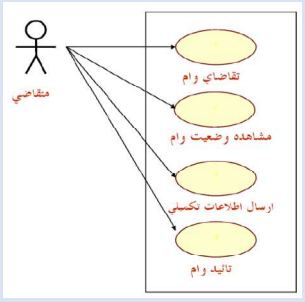




-هرکس یا هرچیزی که با سیستم موجود برهم کنش دارد عامل گفته میشود.بعبارت دیگر به هر استفاده کننده از **Usecase** گویند.

– انسان (کاربران سیستم)
– سیستمهای سخت افزاری یا نرم افزاری دیگری که با سیستم موجود در ارتباطند.

رسم **Usecase Diagram**



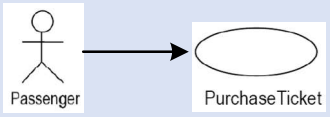
توصیف **Usecase** ها
– شرح هر کدام از سرویس هایی که استفاده شده اند (ارائه سناریو)

– برای بیان تمام سرویسهای استفاده شده در سیستم بکار میرود.
– هر **Usecase** یک **Class diagram** دارد.
– شامل ترکیب کلی **Usecase** و **Actor** میشود.

3

انواع ارتباطات در نمودار مورد استفاده

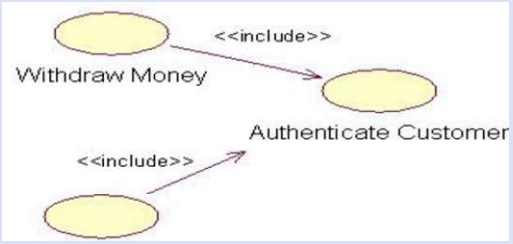
– ارتباط **Communication**

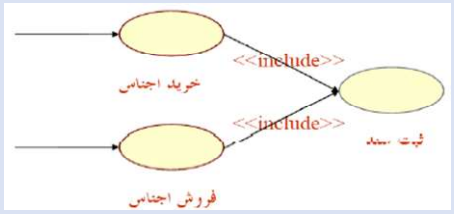


– به ارتباط بین عامل **Actor** و مورد استفاده **Usecase** گویند.

– ارتباط **Include**

– به ارتباط بین **Usecase** ها گویند.
– ارتباط **Include** به یک **Usecase** اجازه استفاده از عملیات میباشده توسط یک **Usecase** دیگر را میدهد.





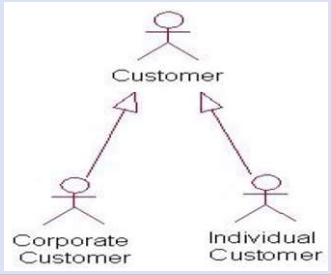
– ارتباط **Extend**

– به ارتباط بین عامل ها گویند.
– یک ارتباط **Extend** به یک **Usecase** اجازه میدهد که بطور دلخواه عملیات مهیا شده توسط دیگر **Usecase** ها را بسط دهد

– یک **Usecase** در حالت خاص میتواند بصورت ویژه ارائه گردد (به یک سرویس ویژه توسعه پیدا کند)

– ارتباط **Generalization**

– ارث بری میان عامل ها برای نشان دادن همانندی چندین عامل بکار میرود.



ستاریو Use Case

تعریف مختصر: یک تعریف مختصر مرتبط با آنچه که **Use Case** انجام میدهد.

پیش شرایط: لیستی از شرایطی هستند که قبل از شروع کار **Use Case** باید بررسی شوند؛بطور مثال یک **Use Case**از قبل اجراء شود یا آیا کاربر حق دستیابی مستقیم برای اجراء**Use Case**ای را دارد یا نه؟

جریان رخدادهای اصلی: آنچه را که در اجرای عملیات در **Use Case** پیش خواهد آمد را مرحله به مرحله توصیف میکند .جریان رخدادها برآنچه سیستم انجام خواهد داد متمرکز میشود ولی به چگونگی انجام آن کاری ندارد.

جریان رخدادهای فرعی: در صورتی اجراء میشوند که در اجرای جریان اصلی مشکلی بوجود آید.مثلا در **Use case** برداشت پول میزان درخواستی از میزان موجودی حساب بیشتر باشد.

شرایط پسین: شرایطی هستند که باید بعد از پایان اجرای**Usecase** دارای مقدار **true** باشند.مثلا ممکن است بعد از کامل شدن یک **Use Case** یک **flag** تنظیم شود.

مثال ستاریوی Use Case

جریان رخدادهای اصلی:

۱ _ دانشجو فرم تکمیل شده حذف و اخذ واحدها را به کارمند آموزش تحویل می دهد .
۲ _ کارمند آموزش پیش نیاز درسها را در پرونده کارنامه های دانشجو کنترل می کند .
۳ _ کارمند آموزش بر مبنای معدل سقف واحدها را چک می کند .
٤ _ کارمند آموزش ظرفیت کلاسها را کنترل می کند .
٥ _ کارمند آموزش تداخل دروس را کنترل می کند .
٦ _ کارمند آموزش برگه حذف و اخذ را مهر تائید می زند .
۷ _ کارمند آموزش برگه تائید شده را به دانشجو تحویل می دهد .

جریان رخدادهای فرعی:

۲ _ در صورتی که پیش نیاز یا هم نیاز رعایت نشده باشد آن درس از لیست دروس حذف می شود .

۳ _ در صورتیکه معدل زیر ۱۲ باشد حداکثر ۱٤ واحد به دانشجو می دهد .
۳ _ در صورتیکه معدل بالای ۱۷ باشد حداکثر ۲٤ واحد به دانشجو می دهد .
۳ _ در صورت تائید مدیر آموزش هر تعداد واحدی به دانشجو می دهد .

برای به واقعیت پیوستن یک **Case Use** تعدادی شی از کلاسهای مختلف با هم تعامل و همکاری میکنند.

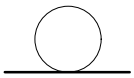
کلاسهای اشیاء یک **Case Use** در نمودار کلاس آن نمایش داده میشوند.برای هر **Case Use** یک نمودار کلاس ارائه میشود.

تعامل و همکاری اشیاء در یک **Case Use** از طریق نمودارهای محاوره ای **(INTRACTION)** نشان داده میشود که به دو نوع نمودارهای همکاری و توالی تقسیم میشود.

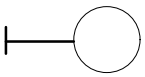
نمودار کلاس Diagram Class

انواع کلاسها (Stereotype)

کلاسهای کنترلی وظیفه کنترل عملیات را بر عهده دارند و معمولا کلاس بی صفت هستند و متد دارند و شی با این کلاس ایجاد نمیشود.

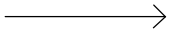


کلاسهای **Entity** مبنایی برای ایجاد **Table** ها و بانک اطلاعاتی سیستم هستند و به جدولی در پایگاه داده ها تبدیل میشوند.

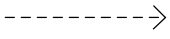


کلاسهای **Boundry** در مز بین **Actor** و سیستم قرار دارند. مثلا فرم ثبت نام شی از کلاس باندری فرم است.(کلیه فرمها یا گزارش عملکرد مالی شی از کلاس باندری گزارش است.) و کلا این کلاسها شاخص فرمهای ورودی – خروجی و رابطها میباشند.

انواع ارتباطات در نمودار کلاس

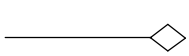


Association ارتباط تناظر



Dependency ارتباط وابستگی

Generalization ارتباط تعمیم



Aggregation ارتباط تجمیع

Composition ارتباط ترکیب

موارد استفاده Class Diagram

- ساختار سیستم را ارائه میکند.
- در ضمن تحلیل نیازمندیها برای مدل کردن
- برای مدل کردن زیر سیستم ها و واسط ها
- طراحی شی برای مدل کردن کلاسها

- روش کار با نرم افزار های طراحی بدین شرح میباشد.

۱- شناخت کامل سیستمی که باید پیاده سازی گردد. (توضیح پروژه و ماهیت انجام کار) - تشخیص صحیح نیازمندیهای کاربر و انتظارات او از سیستم

۲-ترسیم کاری که باید انجام گیرد. (مستند سازی)

مثال از سیستم ثبت نام – انتخاب واحد

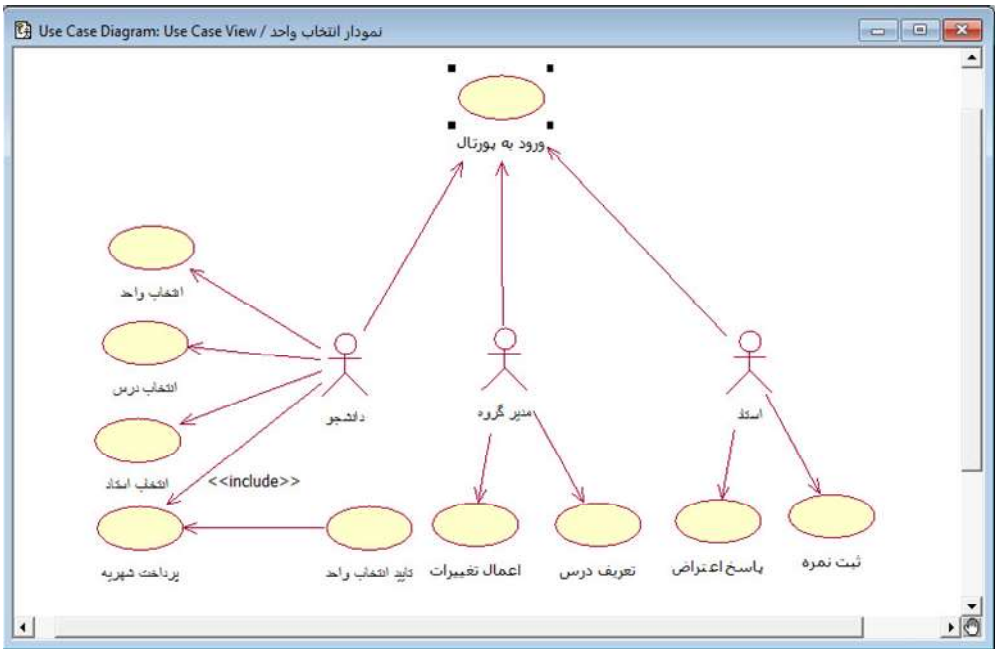
- ابتدا از قسمت **Usecase View** مجموعه **Actor** های مورد نیاز را ترسیم میکنیم.

دانشجو: انتخاب واحد، انتخاب درس، انتخاب استاد، پرداخت شهریه، تایید انتخاب واحد

مدیر گروه: تعریف درس، اعمال تغیرات

استاد: ثبت نمره، پاسخ اعتراض

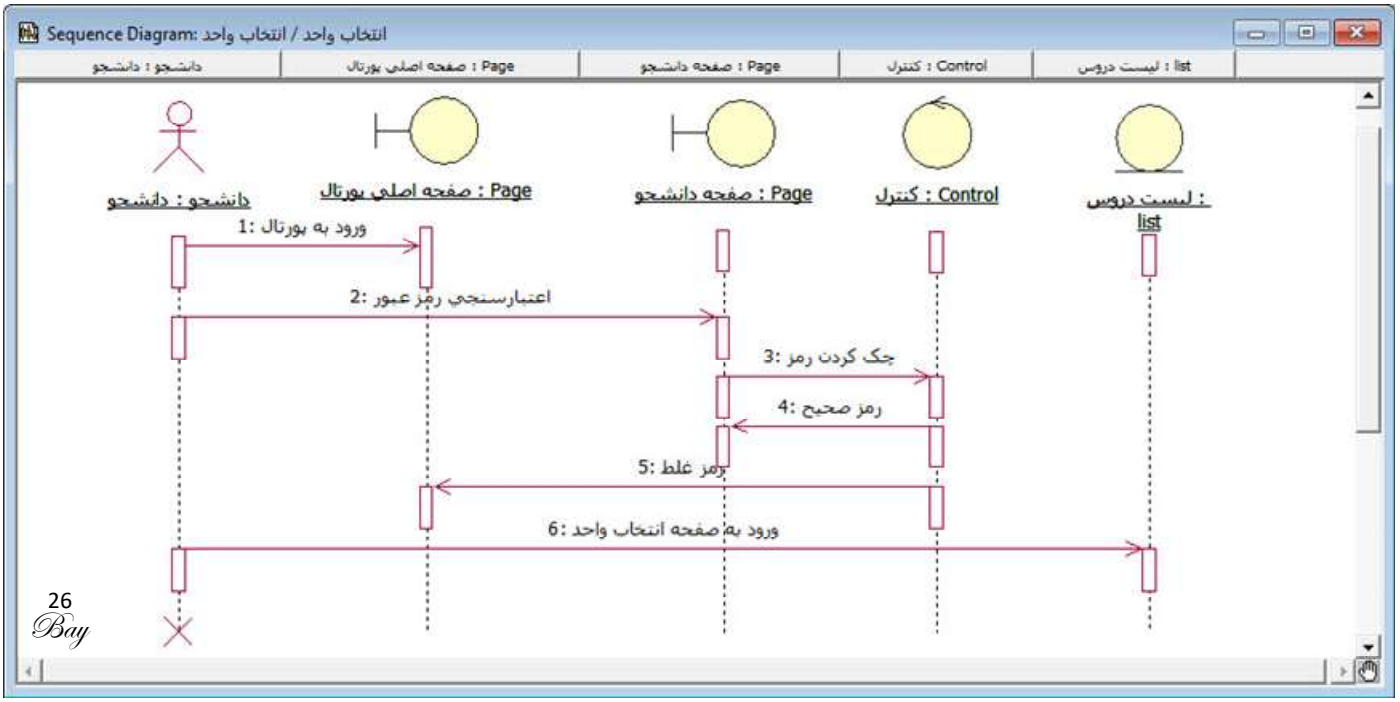
- در یک سیستم معقول بین بیست تا پنجاه **Usecase** بیشتر ترسیم نمیشود.چون باعث پایین آمدن خوانایی و کیفیت کار میگردد.



نمودار توالی انتخاب واحد **Sequence Diagram**

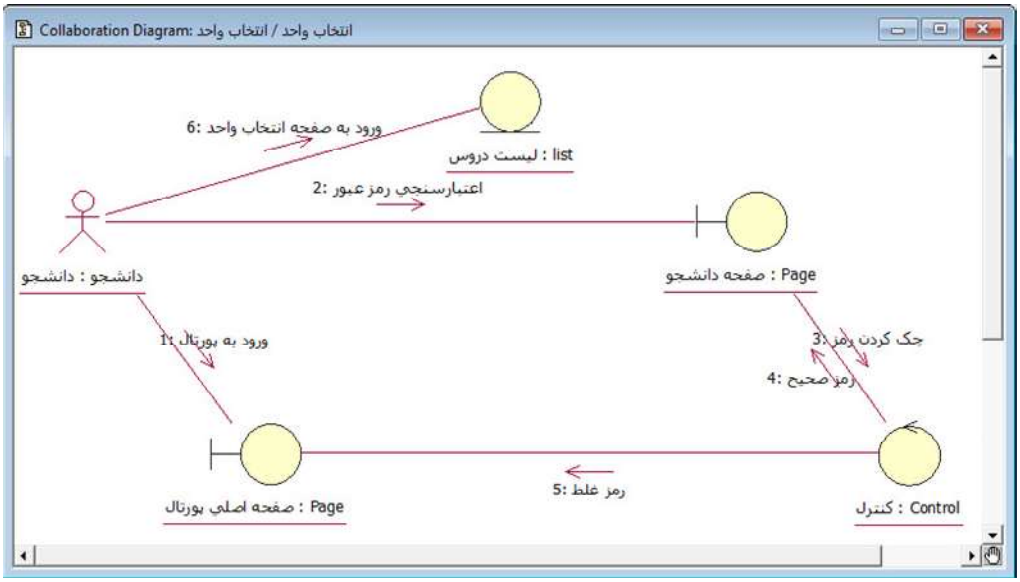
Sequence Diagram=Object+Message

- نمودار توالی برای کل یک سیستم از طریق **Logical View** یا یک **Usecase** خاص از طریق **Usecase Diagram** میتواند ترسیم گردد.
- هدف از **Sequence Diagram** نشان دادن جریان عملیات داخل **Usecase** بر حسب زمان میباشد. و روند منطقی یک سناریو را نشان میدهد.
- طرح کلی یک **Object**را کلاس تعیین میکند. چه اطلاعاتی در یک **Object** ذخیره میشود و چه رفتارهایی میتواند داشته باشد.



نمودار همکاری Collaboration Diagram

- مهمترین تفاوت بین نمودار همکاری و نمودار توالی در شمای ظاهری آن میباشد در نمودار همکاری بیشتر تاکید بر روی رابطه بین **Object** ها میباشد.
- ترتیب زمانی در نمودار های همکاری در نظر گرفته نمیشود.
- جهت ترسیم راحت تر نمودار همکاری ، استفاده از نمودار توالی میباشد که قبلا رسم شده است. **Browse/Create Collaboration Program F5**



نمودار کلاس Class Diagrams

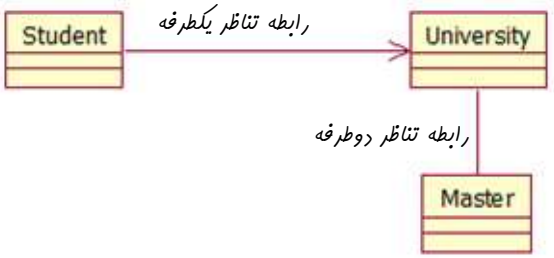
- چهار رابطه بین **Class Diagram** متصور میباشد. **(Association,Generalization,Aggregation,Dependancy)**
- رابطه های **Association** و **Dependancy** از طریق نمودار توالی قابل تشخیص میباشد.
- رابطه های **Generalization** و **Aggregation** از طریق بررسی کلاسها قابل تشخیص میباشد.

ارتباط تناظر Association

- تعامل یک یا چند شی از یک کلاس با یک یا چند شی از کلاس دیگر را نشان میدهد.یک تناظر یک ارتباط معنایی بین کلاس ها است.
- رابطه تناظر میتواند یک رابطه یک طرفه یا دو طرفه باشد.
- هر کلاسی میتواند از طریق یک رابطه توالی یا همکاری به کلاس دیگر یک رابطه پیامی داشته باشد.

جهت ترسیم کلاسها در **Rational rose** از **Logical View** استفاده میشود.

- در اینحالت مشخص میگردد که کلاس **Student** از تمام اتفاقاتی که در کلاس **University** می افتد با خبر است. به عبارت دیگر زمانی که کد تولید میشود داخل کلاس **Student** یک سری از صفات (نمونه ها) کلاس **University** ساخته میشود.چون دانشجو باید از مسائل مرتبط با دانشگاه باخبر باشد.اما دانشگاه نیازی به دانستن اطلاعات مربوط به دانشجو را ندارد.

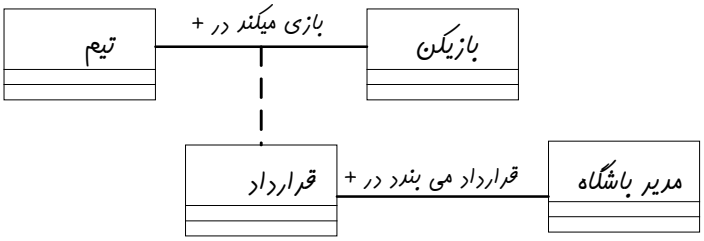


- اگر کلاس دیگری به نام استاد را تعریف کنیم ، پر واضح است که استاد باید اطلاعات مربوط به دانشگاه را داشته باشد ضمن اینکه دانشگاه نیز باید اطلاعات مربوط به استاد را بعنوان کارمند خود باید داشته باشد. جهت تعریف این موضوع باید تیک **Navigable** در **Role A Detail** و **Role B Detail** فعال باشد.

ارتباط وابستگی Dependency

- این رابطه شبیه **Association** است با این تفاوت که رابطه وابستگی همیشه یکطرفه میباشد. و بصورت خط چین نمایش داده میشود و بیشتر برای تعریف مفهوم یک کلاس به کار برده میشود تا بتوان از نمودارها یک کلاس را بهتر درک نمود.
- این رابطه بین دو کلاس یا دو پکیج میتواند برقرار گردد.
- در این رابطه وقتی در یک کلاسی تغییری ایجاد شود مطمئنا در کلاس دیگر هم تغییرات اعمال میگردد.
- در زمان تولید کد در رشنال رز برای یک رابطه وابستگی صفت ها برای کلاس ها اتفاق نمی افتد بدین معنی که کدی برای این رابطه تولید نمیشود و صرفا مبنای مفهومی و درک بهتر از موضوعات را در بردارد.

در یک سیستم گرمایشی وقتی دما بالا یا پایین میرود ، پکیج خاموش و روشن میشود و در واقع پکیج از دماسنج بعنوان یک ورودی استفاده مینماید.

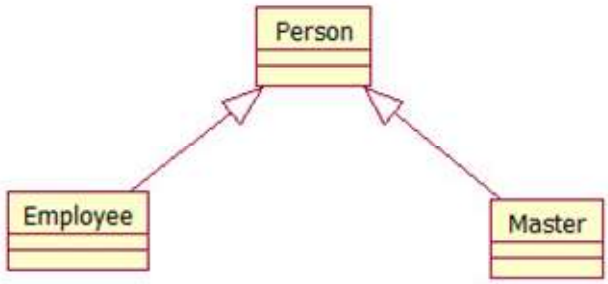


– قرارداد یک شی از کلاس تیم با یک شی از کلاس بازیکن بعنوان یک شی جدید از کلاس تناظر قرارداد مد نظر قرار میگیرد.

ارتباط تعمیم Generalization

– نشان دهنده رابطه وراثتی بین کلاسها میباشد و شامل ۱– کاهش حجم کدنویسی ۲–سازماندهی یک گروه کلاس مرتبط به هم میشود

– در این رابطه که حالت پدر – فرزندی دارد ، در صورتیکه کلاس پدر پاک شود ، قطعا کلاس فرزند نیز از بین خواهد رفت.



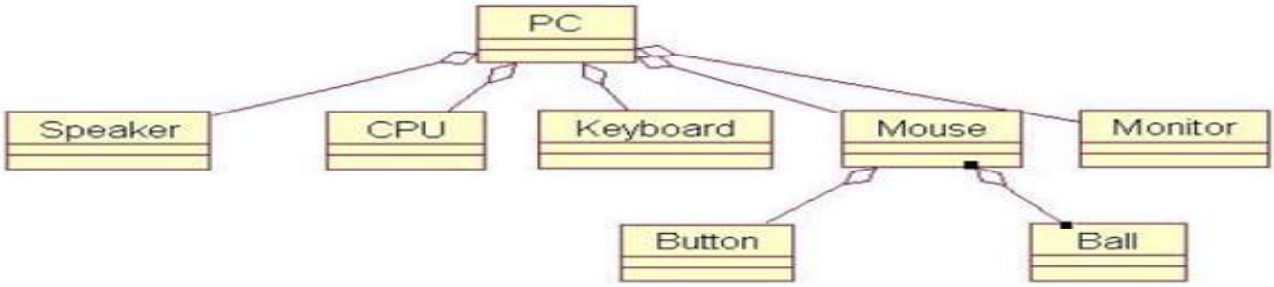
– بعنوان مثال کلاس استاد بغیر از اینکه صفات کلاس شخص را به ارث میبرد خود نیز دارای صفات متمایزی نیز میباشد.

ارتباط تجمیع Aggregation

– این رابطه شبیه رابطه **Association** میباشد ولی با یک فرم قوی تر بدین معنی که کلاس های تجمیع شده از نظر ادراکی در سطح کلاسهای دیگر نمیباشند .

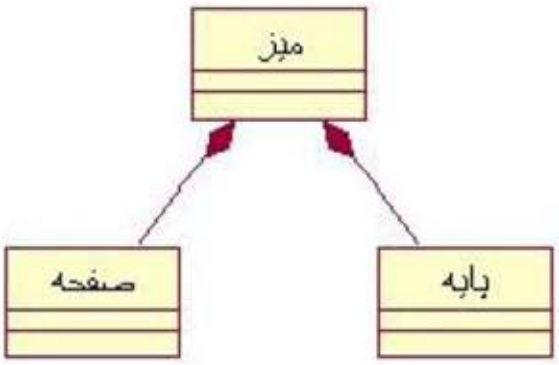
– چند شی از چند کلاس مختلف با هم جمع شده و یک شی از کلاس جدید میسازند.در تجمیع ،هر کدام از اشیاء خارج از مجموعه تجمیع نیز قابل استفاده است.

– در رابطه تجمیع یک رابطه کل به جزء نشان داده میشود.



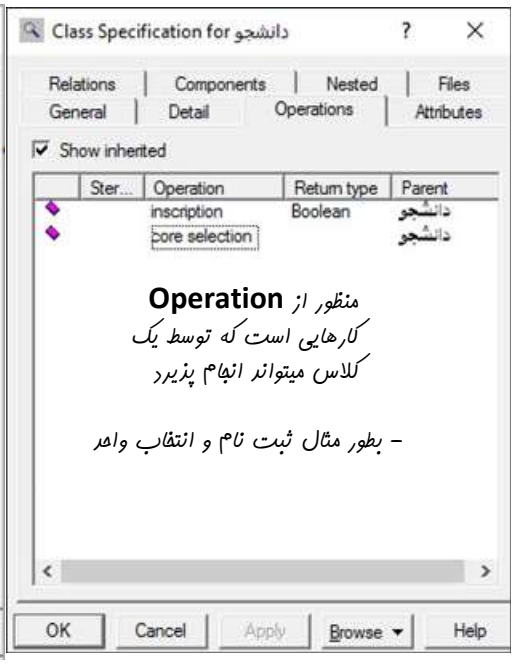
ارتباط ترکیب Composition

– همان مفهوم تجمیع را بصورت قوی تر دارد.در ترکیب هر کدام از اشیاء خارج از مجموعه ترکیب قابل استفاده نیست.



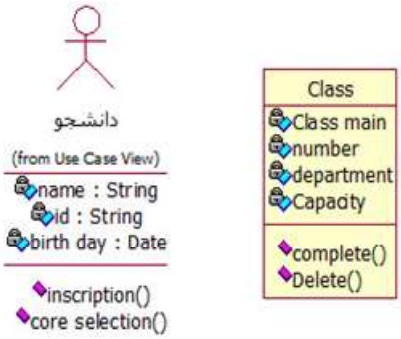
- پس از ایجاد یک کلاس از طریق **Logical view/new class diagram** ابتدا **New class** را زده و سپس با دوبار کلیک وارد قسمت **Specification** میشویم . یکی از قسمتهای مهم آن **Attribute** یا صفات آن میباشد.

- با توجه به کلاس دانشجو صفات آن را تعریف میکنیم.

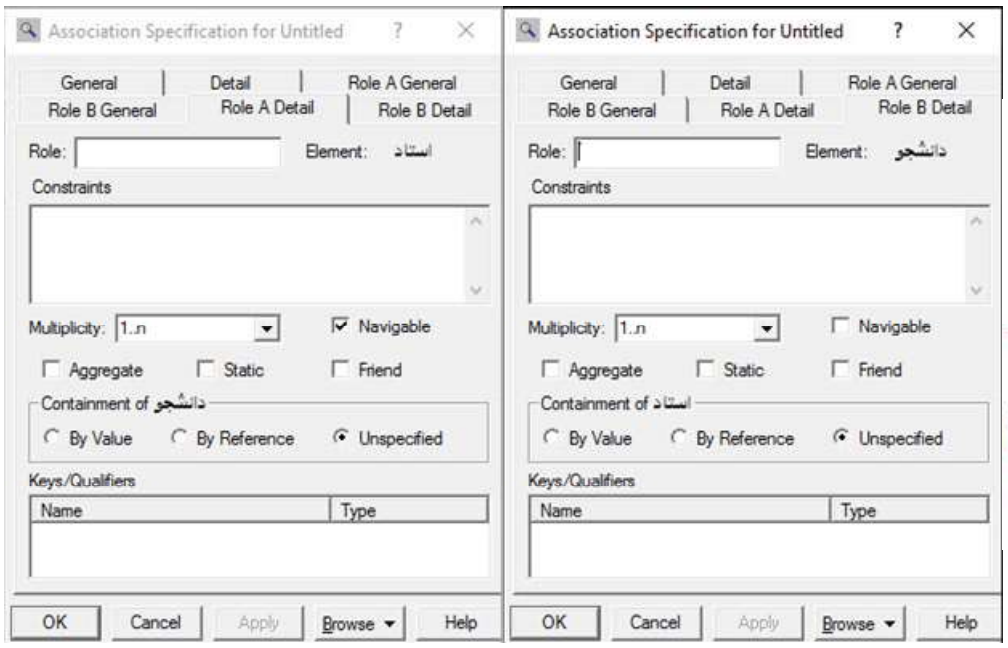


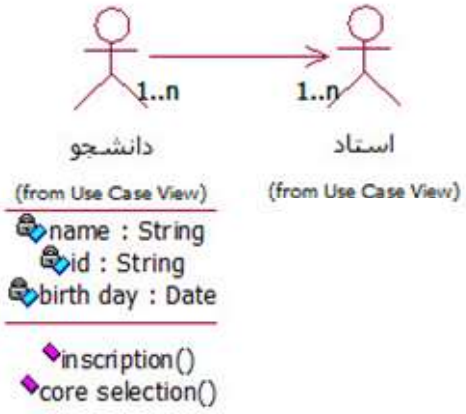
Operation منظور از
کارهایی است که توسط یک کلاس میتواند انجام پذیرد

- بطور مثال ثبت نام و انتخاب واحد



- با دابل کلیک کردن بر روی خط تناظر و مراجعه به **Role A Detail,Role B Detail** از قسمت **Multiplicity** میتوانیم رابطه یک به چند برقرار نماییم.
- حال میتوانیم بین کلاسها رابطه برقرار نماییم . بطور مثال یک دانشجو میتواند چند استاد داشته باشد و استاد نیز میتواند چند دانشجو داشته باشد.





نمودار حالت State Diagram

- این دیاگرام شبیه فلوچارت است و نسبت به دیاگرام های توالی و همکاری کمتر مورد استفاده قرار میگیرد.و حالتهای مختلفی را که یک شی در آن قرار میگیرد را مدل میکند.و به نوعی چرخه حیات شی محسوب میگردد. و به درد اشیائی میخورد که حالتهای زیادی دارند و یا عملکرد قبل و بعد آن شی دارای وابستگی به هم و مهم میباشد.

- بعنوان مثال **Statechart** انتخاب واحد از طریق **Usecase view** ترسیم گردیده است.

- بعد از نماد **Start** نماد **State**را انتخاب میکنیم و سپس در قسمت **Action** چهار حالت متصور میباشد.

1-On Entry
- مجموعه فعالیت هایی که در زمان ورود یک شی به یک حالت باید انجام پذیرد.

2-On Exit
- مجموعه فعالیت هایی که در زمان خروج یک شی از یک حالت باید انجام پذیرد.

3-Do
- شامل فعالیت هایی است که یک شی در هنگام یک حالت انجام میدهد.

4- On Event
- شامل فعالیت هایی است که ملزم به پیش شرط یک اتفاق یا رخداد باشد.

- بخشی از نمودار حالت انتخاب واحد.