مهندسی نرم افزار

Software Engineering

مدرس:

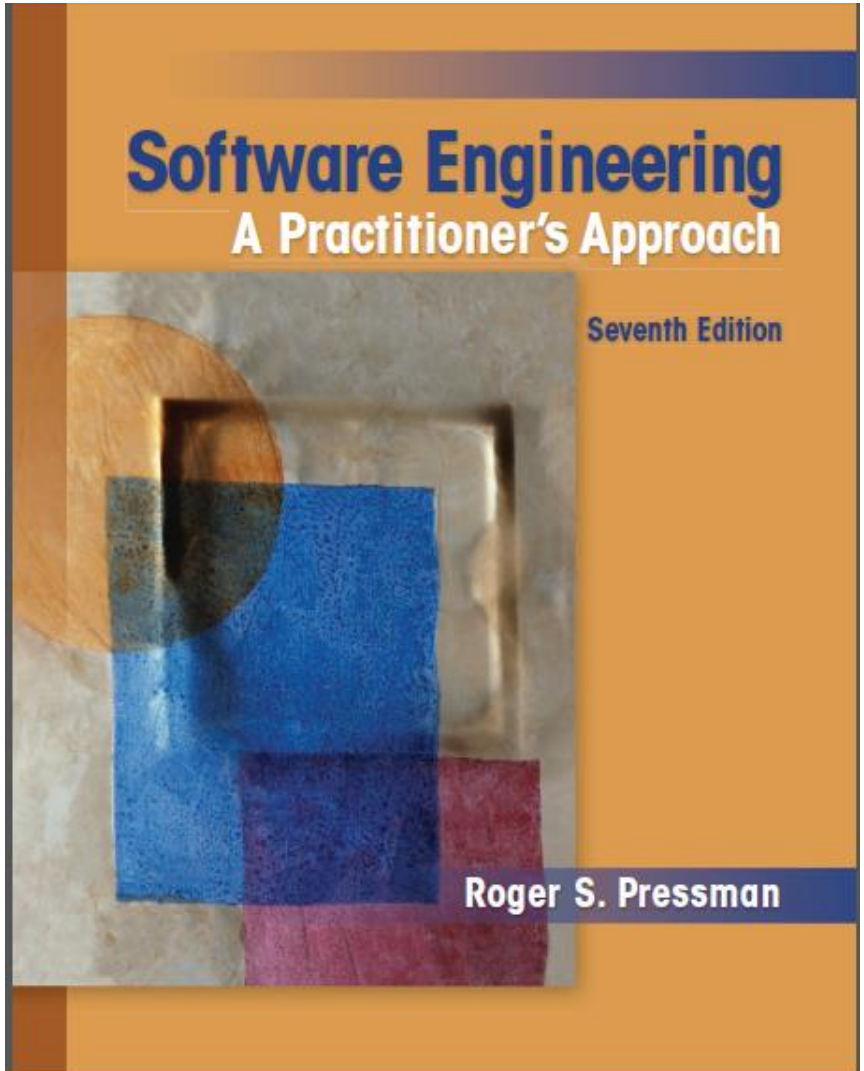
فرشید شیرافکن

دانشجوی دکتری دانشگاه تهران

(کارشناسی و کارشناسی ارشد : کامپیوتر نرم افزار) (دکتری: بیو انفورماتیک)

فهرست

منبع



راجر اس . پرسمن

فصل اول :
نرم افزار و مهندسی نرم افزار

The Nature of Software

- Defining Software
- Software Application Domains
- Legacy Software

The Unique Nature of WebApps

Software Engineering

The Software Process

Software Engineering Practice

- The Essence of Practice
- General Principles

Software Myths

How It All Starts

نرم افزار چیست؟

محصولی است که مهندس نرم افزار طراحی می کند و می سازد.

شامل برنامه هایی است که در کامپیوتری به هر اندازه و با هر معماری قابل اجرا هستند.

مستنداتی دارد که شامل فرم های واقعی و مجازی می شود.

داده هایی دارد که ترکیبی از ارقام و حروف است و می تواند شامل اشکال نمایشی از قبیل اطلاعات تصویری،صوتی و ویدئویی باشد.

محصول کار چیست؟

از دیدگاه مهندس نرم افزار: برنامه ها، مستندات و داده ها که نرم افزار کامپیوتری است.

از دید گاه کاربر: اطلاعاتی است که به نحوی به درد کاربر می خورد.

آیا نرم افزار مرده است؟

اگر چنین بود ، این کتاب را نمی خواندید.

ماهیت نرم افزار

THE NATURE OF SOFTWARE

امروزه نرم افزار نقشی دو گانه دارد:

۱- نوعی محصول است.

توان بالقوه ی یک سخت افزار یا شبکه ای از کامپیوترها را بالفعل می کند.

۲- وسیله ی نقلیه ای برای تحویل یک محصول.

مبنای کنترل کامپیوتر، مخابرات اطلاعات و خلق و کنترل برنامه های دیگر را تشکیل می دهد.

نرم افزار چیست؟

۱- دستورالعمل ها (instructions)

۲- ساختمان داده ها (data structures)

۳- اطلاعات توصیفی (documentation)

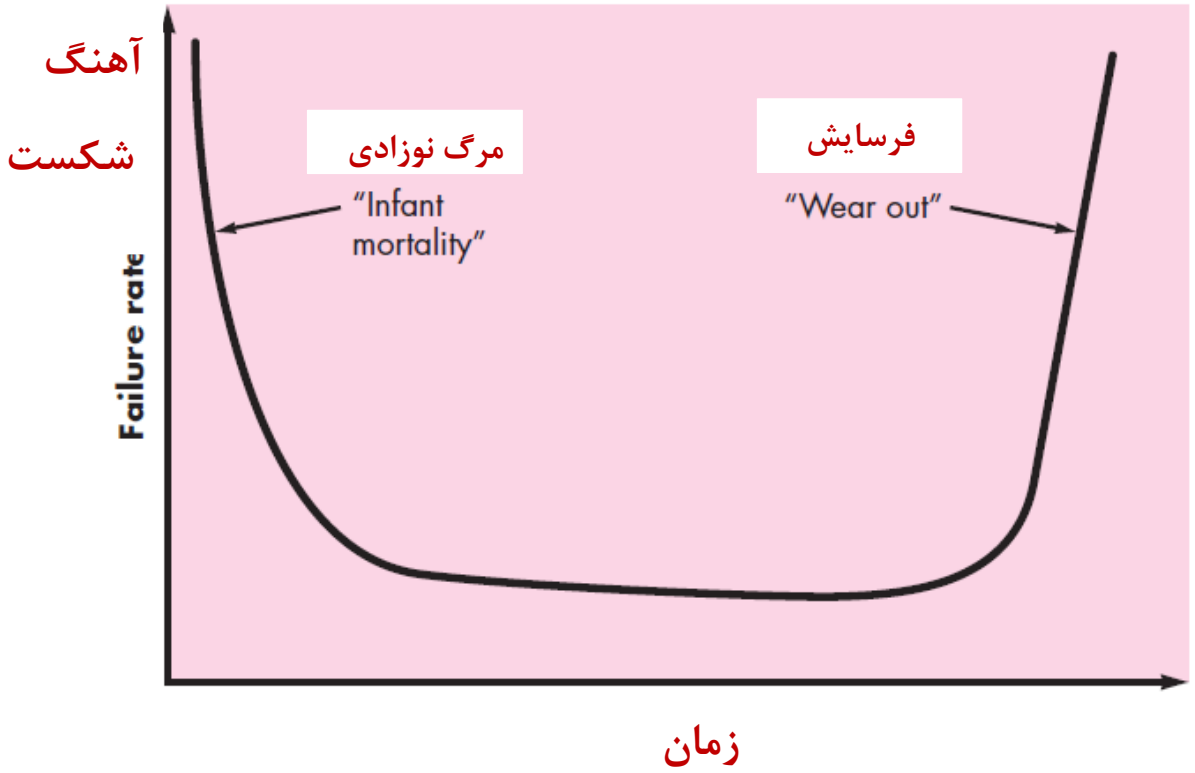
خصوصیات نرم افزار

۱) نرم افزار، مهندسی و بسط داده می شود و چیزی نیست که به معنای کلاسیک کلمه ساخته شود.

۲) نرم افزار فرسوده نمی شود.

۳) گرچه صنعت در حال حرکت به سوی مونتاژ قطعات است ولی اکثر نرم افزار ها همچنان به صورت سفارشی ساخته می شوند.

نمودار آهنگ شکست سخت افزار

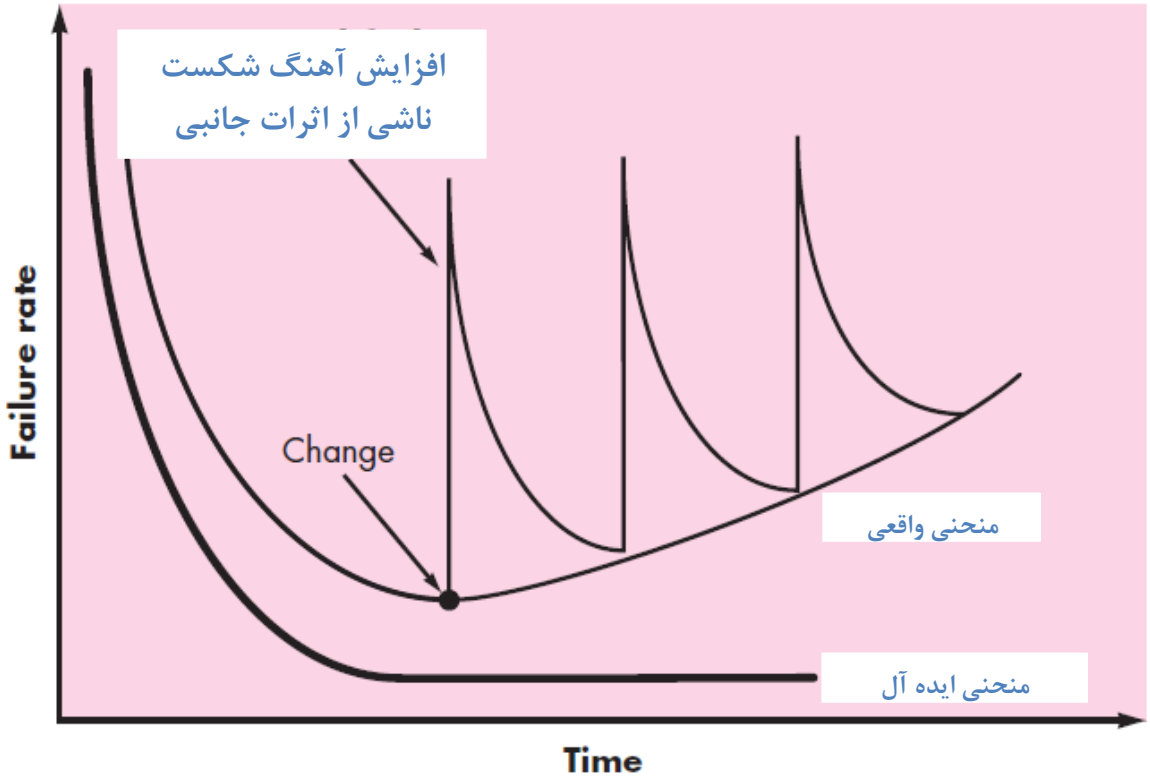


این نمودار نشان می دهد که سخت افزار در ابتدای عمر آهنگ شکست شدیدی دارد.

این عیوب تصحیح می شوند و آهنگ شکست در یک دوره زمانی به مقدار ثابتی نزول می کند.

با گذشت زمان سخت افزار شروع به فرسایش کرده و دوباره آهنگ شکست شدت می گیرد.

منحنی های شکست واقعی و ایده ال برای نرم افزار



نرم افزار در دوران حیات خود دستخوش تغییر می شود با اعمال این تغییرات احتمال دارد برخی عیوب جدید وارد شوند و باعث خیز منحنی آهنگ شکست می شود و پیش از آنکه منحنی بتواند به آهنگ شکست منظم اولیه خود برسد تغییر دیگری درخواست می شود که باعث خیز دوباره منحنی می شود.

دامنه های کاربرد نرم افزار

۱) سیستمی : system software

۲) کاربردی : application software

۳) مهندسی/علمی : engineering/scientific software

۴) تعبیه شده : embedded software

۵) خط تولید : product-line software

۶) برنامه های کاربردی تحت وب: WebApps

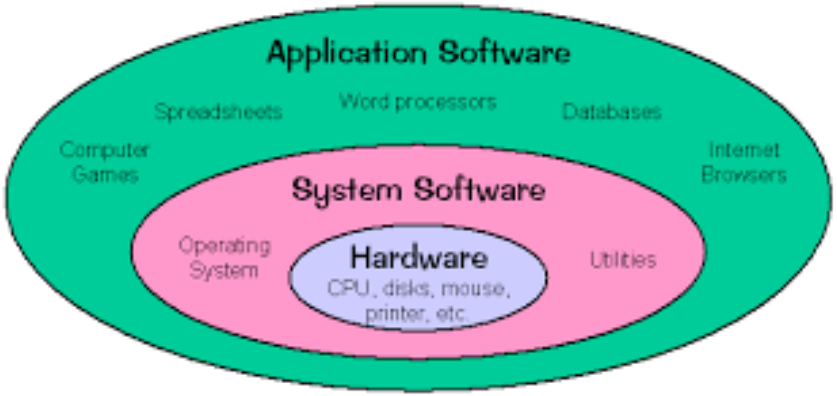
۷) هوش مصنوعی: AI software

دامنه های کاربرد نرم افزار

۱) نرم افزار های سیستمی : system software

مجموعه ای از برنامه ها که برای سرویس دهی به برنامه های دیگر نوشته شده اند.

مشخصه های حیطه ی نرم افزار سیستمی عبارتند از:



الف) تعامل سنگین با سخت افزار

ب) استفاده سنگین توسط چند کاربر

ج)عمل کنونی که مستلزم زمانبندی است

د) مدیریت فرایند پیچیده و اشتراک منابع

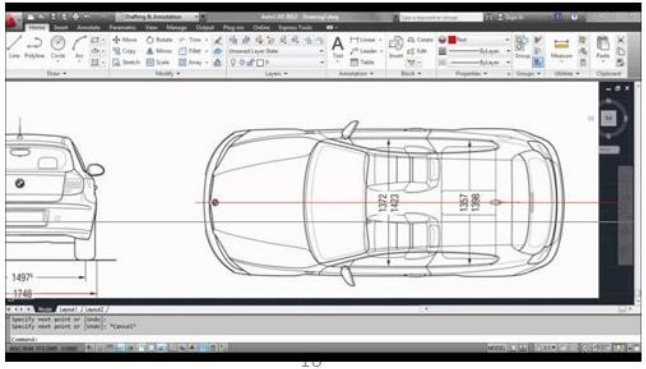
ه)ساختمان داده های پیچیده و واسط های خارجی دیگر

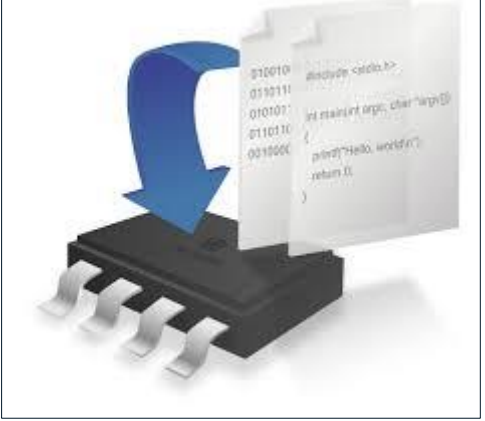
۲) نرم افزارهای کاربردی : application software

برنامه های مستقل که یک نیاز تجاری را برطرف می سازد.

۳) نرم افزارهای مهندسی/علمی : engineering/scientific software

نرم افزارهای علمی توسط الگوریتم هایی مشخص می شوند که ارقام و اعداد را پردازش می کنند.





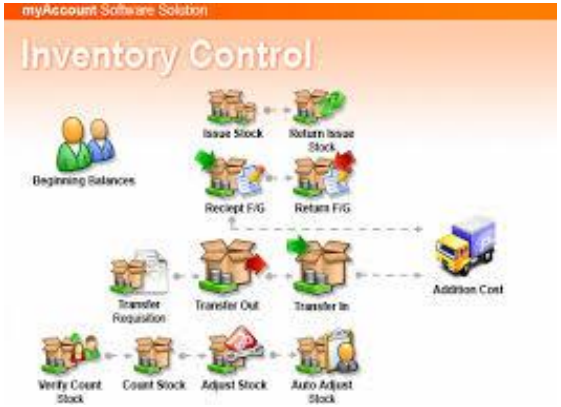
۴) نرم افزارهای تعبیه شده : embedded software

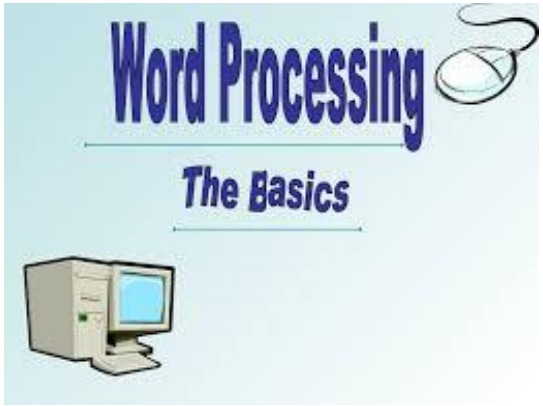
در حافظه فقط خواندنی جای دارند.

برای کنترل محصولات و سیستم های مربوط به بازارهای صنعتی و مصرفی به کار می روند.

۵) نرم افزارهای خط تولید : product-line software

برای فراهم آوردن یک قابلیت خاص جهت استفاده توسط بسیاری از مشتریان مختلف طراحی می شوند.





۶) برنامه های کاربردی تحت وب: WebApps

این گروه از نرم افزارهای شبکه ای شامل مجموعه گسترده ای از برنامه های کاربردی می شوند.

۷) نرم افزار های هوش مصنوعی: AI software

برای حل مسائل پیچیده ای که به روش های عددی قابل حل نیستند از الگوریتم های غیرعددی استفاده می کنند.

مثال هایی از کاربرد :

- سیستم های خبره
- تشخیص الگو های تصویری و صوتی
- شبکه های عصبی مصنوعی
- اثبات قضایا و بازی

چالش ها

- کار با کامپیوتر در جهانی باز (Open world computing)
- تامین منابع از طریق شبکه (Netsourcing)
- کد منبع باز (Open source)

نرم افزارهای قدیمی

نرم افزارهای قدیمی چند دهه قبل ساخته شده اند و پیوسته اصلاح شده اند تا تغییرات به عمل آمده در خواسته های تجاری را پاسخ گو باشند.

دلایل تکامل سیستم های قدیمی:

۱) نرم افزار باید برای برآورده ساختن نیازهای محیط های جدید کامپیوتر یا فن آوری های جدید اصلاح گردد.

۲) نرم افزار باید بهبود یابد تا خواسته های تجاری جدید را پیاده سازی کند.

۳) نرم افزار باید گسترش داده شود تا با سایر سیستم ها یا بانک اطلاعاتی جدیدتر قابلیت همکاری داشته باشد.

۴) نرم افزار باید دوباره معماری شود تا در یک محیط شبکه نیز قادر به حیات خود باشد.

مشخصات برنامه های تحت وب

Characteristics of WebApps

۱- میزان تمرکز شبکه (Network intensiveness)

۲- همروندی (Concurrency)

۳- بار غیر قابل پیش بینی (Unpredictable load)

۴- کارایی (Performance)

۵- قابلیت دسترسی (Availability)

۶- داده – محوری (Data driven)

۷- حساس به محتویات (Content sensitive)

۸- تکامل پیوسته (Continuous evolution)

۹- بی واسطگی (Immediacy)

۱۰- امنیت (Security)

۱۱- زیبایی شناسی (Aesthetics)

چند نکته کلیدی

در مورد ساخت نرم افزار

۱) پیش از آنکه برای مساله راهکاری بیابید آن را درک کنید.

۲) طراحی ، یکی از فعالیت های محوری در مهندسی نرم افزار است.

۳) کیفیت و قابلیت نگهداری هر دو نتیجه طراحی خوب هستند.

تعریف مهندسی نرم افزار

تعریف فریتز باور:

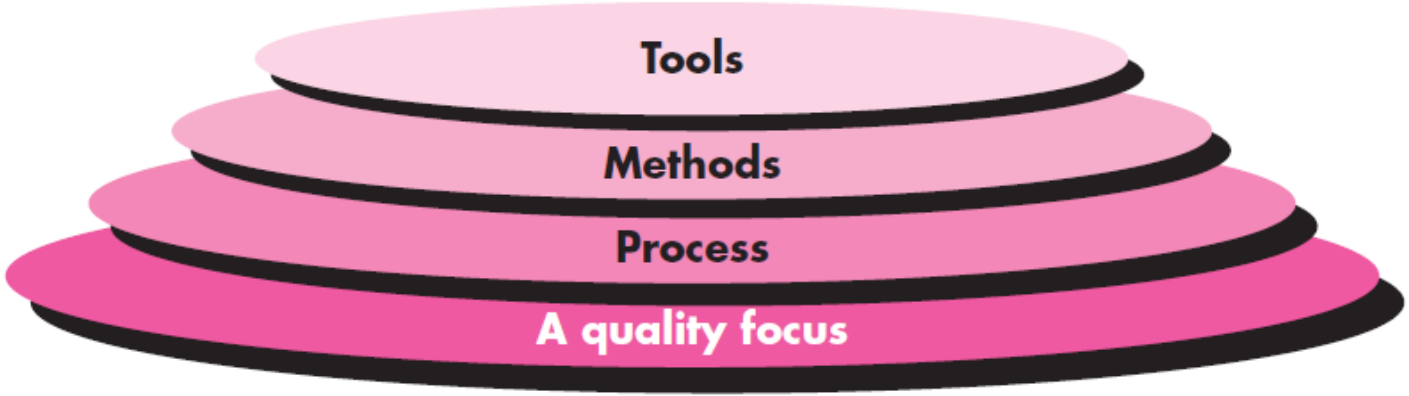
وضع اصول مهندسی بجا و مناسب و استفاده از آنها برای به دست آوردن یک نرم افزار مقرون به صرفه که قابل اطمینان بوده، روی ماشین های واقعی به طرزی کارآمد عمل کند.

تعریف IEEE

۱- کاربرد یک روش سیستماتیک ، علمی و کمیت پذیر در بسط، راه اندازی و نگهداری نرم افزار ، یعنی استفاده از مهندسی در نرم افزار .

۲- مطالعه روش ها به صورت ذکر شده در (۱)

لایه های مهندسی نرم افزار



- توجه به کیفیت، سنگ بنای نگهدارنده ی مهندسی نرم افزار است.
- بنیاد مهندسی نرم افزار لایه ی فرایند است.
- روش های مهندسی نرم افزار، شیوه های فنی برای ساخت نرم افزار را فراهم می کند.
- ابزار مهندسی نرم افزار متضمن پشتیبانی خودکار یا نیمه خودکار برای فرایند و روش ها هستند.

تعریف فرایند

فرایند تعیین می کند که چه کسی چه کاری را در چه زمانی و چگونه انجام دهد تا به هدفی معین برسد.

چار چوب فرایند کلی برای مهندسی نرم افزار شامل ۵ فعالیت می شود

۱- **ارتباطات** (Communication)

هدف ارتباطات درک اهداف طرف های ذینفع برای پروژه و جمع آوری خواسته هایی است که می توانند ویژگی ها و قابلیت های عملیاتی نرم افزار را تعیین کنند.

۲- **برنامه ریزی** (Planning)

با توصیف وظایف فنی که قرار است اجرا شوند، خطرات احتمالی، منابعی که مورد نیاز خواهند بود، محصولات که باید تولید شوند و زمانبندی کاری، مهندسی نرم افزار را مشخص می کند.

۳- **مدل سازی** (Modeling)

مهندس نرم افزار با ایجاد مدل هایی جهت درک بهتر خواسته ها و طراحی که به این خواسته ها برسد مدل سازی می کند.

۴- **ساخت** (Construction)

این فعالیت ها، تولید کدها و آزمون لازم برای آشکار کردن خطاهای موجود در کد ها را با هم تلفیق می کند.

۵- **استقرار** (Deployment)

نرم افزار به مشتری تحویل می شود که محصول تحویل شده را ارزیابی و براساس آن بازخوردی ارائه می دهد.

فعالیت های چتری (Umbrella Activities)

- مدیریت پروژه های نرم افزاری (Software project management)
- مدیریت ریسک(Risk management)
- تضمین کیفیت نرم افزار (Software quality assurance)
- بازبینی های فنی(Formal technical reviews)
- اندازه گیری (Measurement)
- مدیریت پیکر بندی نرم افزار(Software configuration management)
- مدیریت قابلیت استفاده مجدد (Reusability management)
- تهیه و تولید محصول کاری (Work product preparation and production)

تفاوت های مدل فرایند

فرایندی که برای یک پروژه پذیرفته می شود، ممکن است با فرایند پذیرفته شده برای پروژه ای دیگر تفاوتی چشمگیر داشته باشد. از جمله :

۱) جریان کلی فعالیت ها ، کنش ها و وظایف و بستگی آن ها به یکدیگر

۲) درجه تعریف کنش ها و وظایف در هر فعالیت چارچوبی

۳) درجه ی شناسایی محصولات کاری و نیاز به آن ها

۴) شیوه ی اعمال فعالیت های تضمین کیفیت

۵) درجه ی کلی جزئیات به کار رفته در توصیف فرایند

۶) درجه ی دخالت مشتری و طرف های ذینفع در پروژه

۷) سطح استقلال داده شده به تیم نرم افزار

۸) درجه ی توصیف نقش ها و سازمان دهی تیم

مدل فرایند تجویزی

تاکید بر جزئیات تعریف، شناسایی و کاربرد فعالیت ها و وظایف .

هدف :

- بهبود بخشیدن به کیفیت سیستم،
- بالا بردن قابلیت مدیریت پروژه ها،
- قابل پیش بینی کردن تاریخ تحویل و هزینه ها
- راهنمایی تیم مهندسان نرم افزار در اجرای کارهای لازم برای ساخت یک سیستم.

مدل فرایند چابک

بر سرعت تاکید دارد و مجموعه ای از اصول را دنبال می کند که به یک روش غیررسمی تر برای فرایند نرم افزار منجر می شود.

این مدل بر قابلیت مانور و انطباق پذیر تاکید دارد.

برای انواع بسیاری از پروژه ها مناسب بوده، به بویژه هنگام مهندسی برنامه کاربردی تحت وب.

جوهر عمل در مهندسی نرم افزار

۱- شناخت مسئله (برقراری ارتباط و تحلیل)

Understand the problem (communication and analysis)

۲- طرح ریزی برای یک راه حل(مدل سازی و طراحی نرم افزار)

Plan a solution (modeling and software design)

۳- اجرای برنامه ریزی (ایجاد کد)

Carry out the plan (code generation)

۴- بررسی نتیجه برای صحت(آزمایش و تضمین کیفیت)

Examine the result for accuracy (testing and quality assurance)

اصول کلی در مهندسی نرم افزار

Hooker’s General Principles

اصل : یک قانون یا فرض زیر بنایی است که در یک سیستم فکری وجود آن ضروری است.

۱- دلیل وجود سیستم (The Reason It All Exists)

۲- ساده نگه داشتن (Keep It Simple)

۳- حفظ چشم انداز (Maintain the Vision)

۴- آنچه که شما تولید می کنید، دیگران مصرف کنند (What You Produce, Others Will Consume)

۵- آینده نگری (Be Open to the Future)

۶- برنامه ریزی پیشاپیش برای استفاده مجدد (Plan Ahead for Reuse)

۷- تفکر (Think!)

پندارهای باطل نرم افزاری

Software Myths

۱) پندارهای باطل مدیریتی

۲) پندارهای باطل مشتریان

۳) پندارهای باطل سازندگان

پندارهای باطل مدیریتی

اگر از برنامه عقب بیفتیم ، می توانیم بر تعداد برنامه نویسان بیفزایم و عقب افتادگی را جبران کنیم.

اگر تصمیم به برون سپاری (outsource) یک پروژه ی نرم افزاری به شرکت دیگری بگیرم ، می توانم خودم را آسوده سازم و بگذارم تا آن شرکت آن را بسازد.

پندارهای باطل مشتریان

Customer myths

نیازهای پروژه پیوسته در حال تغییر است، ولی این تغییرات را به راحتی می توان در نرم افزار جای داد زیرا نرم افزار انعطاف پذیر است.

پندارهای باطل سازندگان

تا هنگامی که برنامه را اجرا نکرده ایم، راهی برای ارزیابی کیفیت آن ندارم.

تنها چیز قابل تحویل برای یک پروژه ی موفق، برنامه ای ست که کار می کند.

مهندسی نرم افزار ما را وادار می کند که مستندات حجیم و بیهوده تهیه کنیم و از سرعت کار ما می کاهد.

چگونگی آغاز یک پروژه

SafeHome

How a Project Starts

The scene: Meeting room at CPI Corporation, a (fictional) company that makes consumer products for home and commercial use.

The players: Mal Golden, senior manager, product development; Lisa Perez, marketing manager; Lee Warren, engineering manager; Joe Camalleri, executive VP, business development

مدیر ارشد　　مدیر بازرگانی

مدیر مهندسی　　معاون مدیر اجرایی

پایان فصل اول

مهندسی نرم افزار

فصل دوم : مدل های فرایند

مدرس:

فرشید شیرافکن

دانشجوی دکتری دانشگاه تهران

(کارشناسی و کارشناسی ارشد : کامپیوتر نرم افزار) (دکتری: بیو انفورماتیک)

فرآیند چیست؟

وقتی کار می کنید تا یک سیستم یا یک محصول بسازید، حتماً باید یک سری مراحل قابل پیش بینی را چک کنید: یک نقشه راه که در ایجاد نتیجه ای با کیفیت بالا و به موقع شما را یاری می کند.

این نقشه که آن را دنبال می کنید **فرآیند نرم افزار** نام دارد.

چه کسی آن را انجام می دهد؟

مهندسان نرم افزار و مدیران آنها، فرآیند را با نیازهای خود مطابقت داده و سپس آن را دنبال می کنند.

به علاوه کسانی که نرم افزار را درخواست کرده اند در فرآیند نرم افزار نقش دارند.

چرا فرآیند نرم افزار اهمیت دارد؟

زیرا باعث ثبات، کنترل و سازماندهی فعالیتی می شود که اگر به حال خود گذاشته شود ممکن است باعث آشوب شود.

چه مراحلی دارد؟

مراحل فرآیند به نرم افزاری که میخواهید بسازید بستگی دارد .

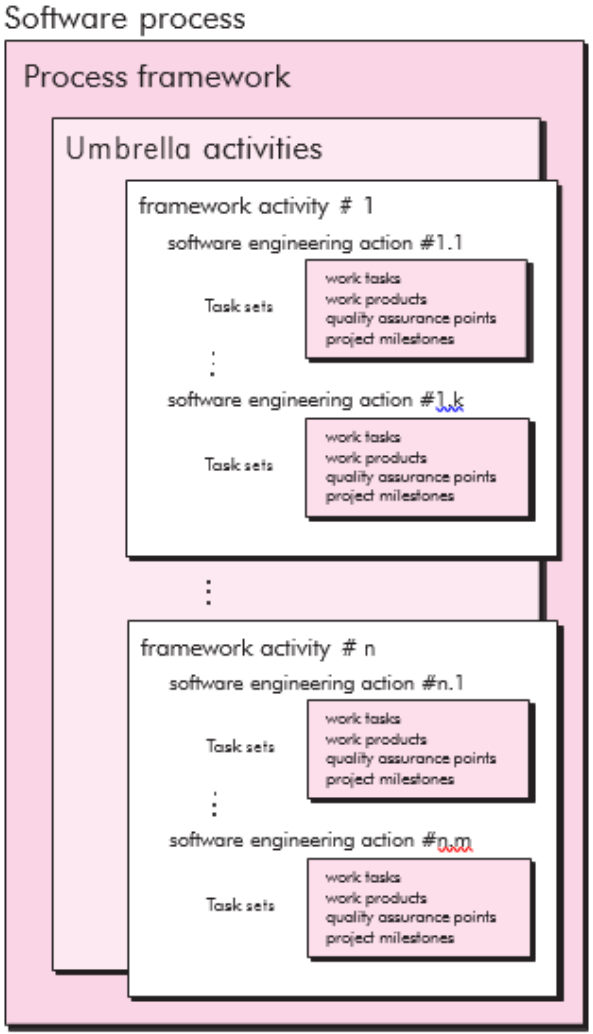
از دیدگاه مهندس نرم افزار حاصل کار چیست؟

برنامه ها، داده ها و مستنداتی است که به عنوان نتیجه ای از فعالیت های مهندسی نرم افزار مشخص شده توسط فرآیند تولید می شوند.

چطور مطمئن شوم که درست از عهده کار برآمده ام؟

چند راهکار برای ارزیابی فرآیند نرم افزار وجود دارد.

ولی کیفیت،به موقع بودن و کارایی درازمدت محصولی که ساخت اید، بهترین ملاک ها برای بازدهی فرآیند مورد استفاده است.



یک مدل فرایند کلی

work tasks
work products
quality assurance points
project milestones

جریان فرآیند

یک جنبه ی مهم از فرآیند نرم افزار است.

شرح می دهد که فعالیتهای چتری و کنش ها و وظایفی که در داخل هر فعالیت چارچوبی رخ

می دهند از نظر ترتیب زمانی چگونه سازماندهی می شود.

انواع جریان فرایند:

۱- خطی

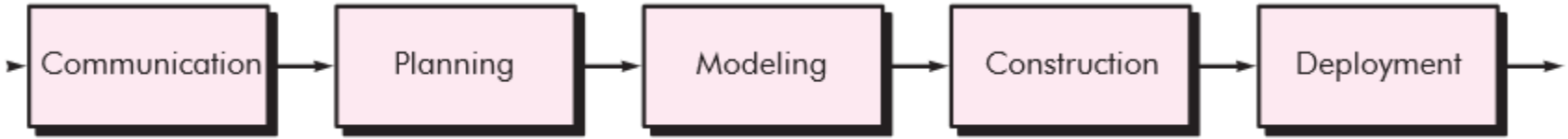
۲- مبتنی بر تکرار

۳- تکاملی

۴- موازی

جریان فرآیند خطی

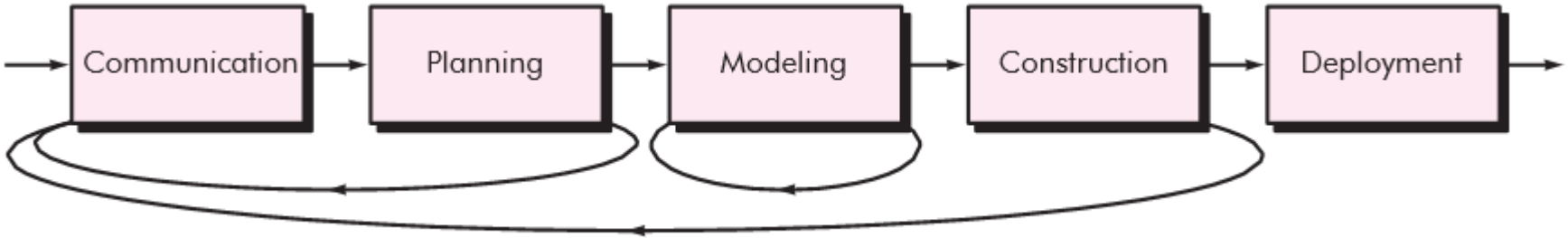
هر کدام از ۵ فعالیت چارچوبی به ترتیب اجرا می شود، به طوری که با ارتباطات آغاز و به استقرار ختم می شود.



Linear process flow

جریان فرآیند مبتنی بر تکرار

در این جریان پیش از رفتن به دور تکرار بعدی یک یا چند فعالیت تکرار میشود.

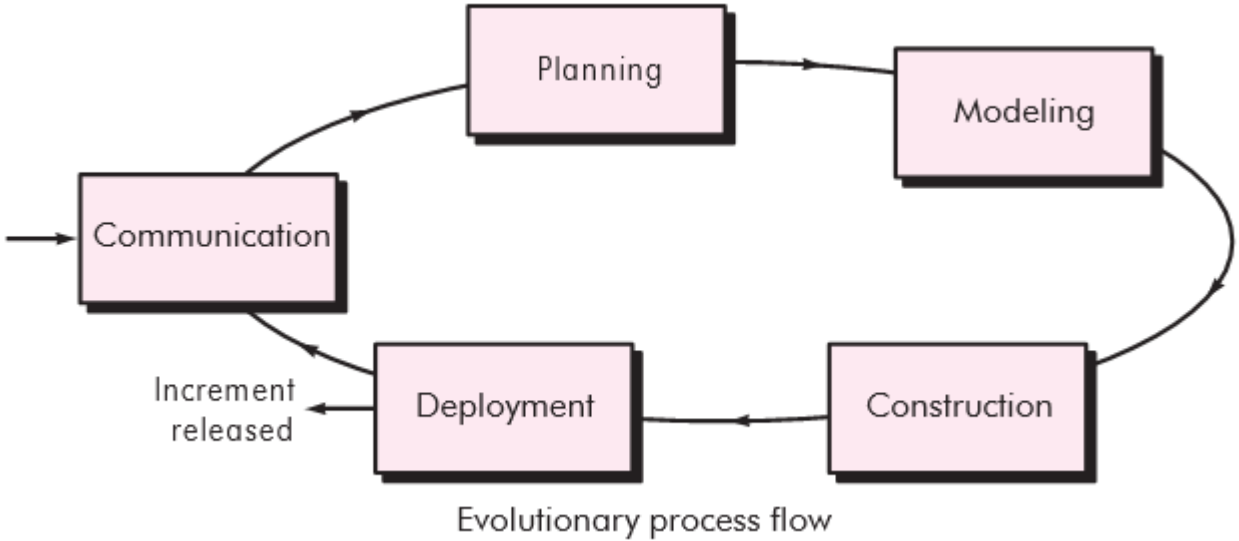


Iterative process flow

جریان فرآیند تکاملی

در این جریان فعالیت ها به شیوه ی حلقوی اجرا می شوند .

هر مدار از ۵ فعالیت عبور می کند که به نسخه کاملتری از نرم افزار می انجامد .

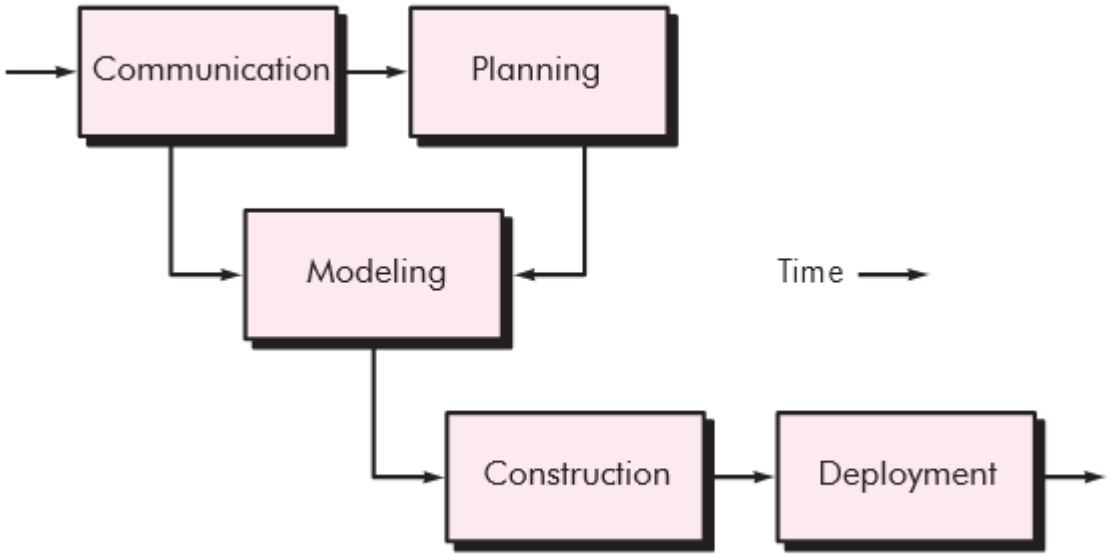


Evolutionary process flow

جریان فرآیند موازی

یک یا چند فعالیت به موازات سایر فعالیت ها انجام می شود .

مثلاً مدل سازی برای یک جنبه از نرم افزار ممکن است به موازات ساختار جنبه ی دیگری از نرم افزار اجرا گردد.



Parallel process flow

مجموعه وظایف (مثال)

برای یک پروژهٔ نرم افزاری کوچک که یک نفر با خواسته های صریح و ساده در مکانی دوردست درخواست می کند ، فعالیت برقراری ارتباط ممکن است شامل چیزی در حد یک تماس با ذینفع مورد نظر باشد.

پس تنها کنش لازم ، مکالمه تلفنی است.

مجموعه وظایف :

۱- بحث درباره خواسته ها و یادداشت برداشتن

۲- سازمان دهی یادداشت ها دریک بیان مختصر از خواسته ها

۳- ارسال ایمیل برای بازبینی و تصویب

الگوی های فرآیند

Process Patterns

الگوی فرآیند، مشکلی مرتبط با آن فرآیند را توصیف می کند که طی کار مهندسی نرم افزار با آن مواجه شده باشند ،

محیطی که در آن مشکل مشاهده شده را مشخص می کند

و

یک یا چند راهکار برای آن پیشنهاد می کند.

قالب برای توصیف الگوی فرآیند

نام الگو: دادن نام با معنی به الگو نامی .

نیروها: محیطی که الگو در آن مشاهده می شود و مواردی که مسئله را پدیدار می کند و ممکن است بر راهکار آن تأثیر بگذارند.

نوع : نوع الگو مشخص می شود. (مرحله ای – وظیفه ای- فازی)

حیطه ی اولیه: توصیف شرایطی که الگو در آن کاربرد دارد.

مسئله : مسئله ی خاصی که قرار است توسط این الگو حل شود.

راهکار: توصیف چگونگی پیاده سازی موفق الگو

حیطه ی حاصل : شرایطی را توصیف می کند که نتیجه ی پیاده سازی موفق الگو هستند.

الگوهای مرتبط : فهرستی از همه ی الگوهای فرآیند تهیه کنید که با این الگو ارتباط مستقیم دارند.

الگوهای پیشنهادی توسط امبلر

۱- **الگوی مرحله ای**: مسئله ای مرتبط با یک مسئله ی چارچوبی را برای یک فرآیند تعریف می کند.

۲- **الگوی وظیفه ای:** مسئله ای مرتبط با یک کنش یا وظیفه ی کاری نرم افزاری را تعریف می کند که کار مهندسی نرم افزار موفق به آن بستگی دارد.

۳- **الگوی فازی**: یک سری فعالیت های چارچوبی را تعریف می کند که درون فرآیند رخ می دهند حتی هنگامی که جریان کلی فعالیت ها ماهیتی تکراری داشته باشند.

روش های ارزیابی فرایند و بهبودی

CMMI برای بهسازی (SCAMPI)

یک مدل ارزیابی فرآیند ۵ مرحله ای فراهم می آورد که شامل فازهای زیر می باشد:

۱) شروع ۲) عیب یابی ۳) ساخت ۴) عملیات ۵) یادگیری

ارزیابی مبتنی بر CMM برای بهبود بخشیدن به فرایندهای داخلی (CBA IPI):

یک تکنیک عیب یابی برای ارزیابی بلوغ نسبی یک سازمان نرم افزاری فراهم می آورد .

SP/CE (ISO/IEC15504) :

استانداردی که مجموعه ای از خواسته ها را برای ارزیابی فرآیند نرم افزار تعریف می کند .

هدف این استاندارد کمک به سازمان ها در توسعه ی یک ارزیابی عینی از بازدهی هرگونه فرآیند نرم افزار تعریف شده است.

ISO 9001:2000 برای نرم افزار:

یک استاندارد عمومی که برای هر سازمانی که مایل به بهسازی کیفیت کلی محصولات، سیستم ها یا سرویس های ارائه شده اش باشد قابل استفاده است.

مدل های فرآیند تجویزی(سنتی)

PRESCRIPTIVE PROCESS MODELS

این مدل در ابتدا برای نظم بخشیدن به آشوب موجود در توسعه ی نرم افزار پیشنهاد شد.

۱- مدل آبشاری The Waterfall Model

۲- مدل های فرایند افزایشی Incremental Process Models

۳- مدل های فرایند تکاملی Evolutionary Process Models

۴- مدل توسعه همروند Concurrent Models

تجویز موارد زیر برای هر پروژه

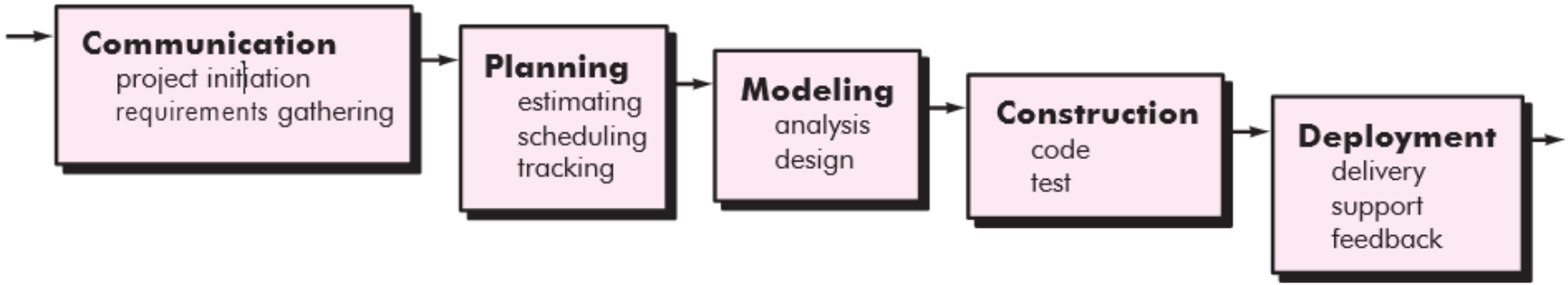
- مجموعه ای از عناصر فرآیند
- فعالیت های چارچوبی
- عملیات مهندسی نرم افزار
- وظایف
- محصولات کاری
- تضمین کیفیت
- ساز وکارهای کنترل تغییرات

مدل آبشاری

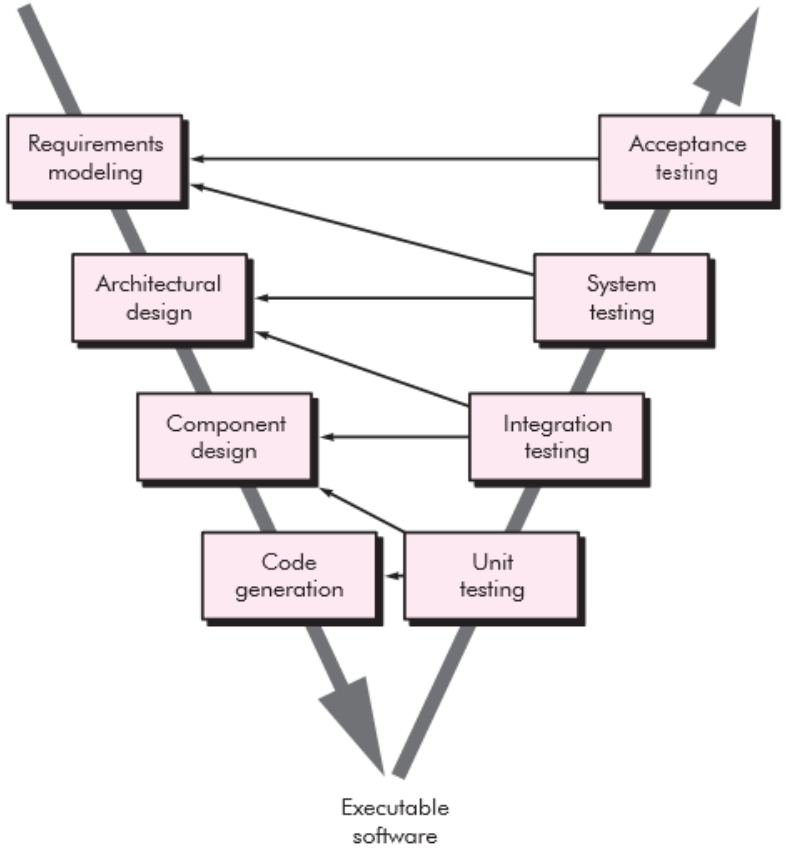
The Waterfall Model

وقتی خواسته های مربوط به یک مسئله به خوبی شناخته شده اند.

هنگامی که کار به طریق خطی از برقراری ارتباط تا استقرار جریان پیدا می کند .



مدل V



شکل دیگر در نمایش مدل آبشاری .

تصویر رابطه تضمین کیفیت با :

- واکنش های مرتبط با ارتباط،
- مدلسازی
- فعالیت های ساخت اولیه.

با حرکت تیم نرم افزاری به طرف پایین و سمت چپ V خواسته های اساسی مسئله رفته رفته پالایش شده و جزئیات بیشتری از آنها تعیین می شود و مسئله و راهکار آن بهتر نشان داده می شود .

هنگامی که کدها نوشته شد، تیم در طرف راست V به طرف بالا حرکت می کند و اساسا یک سری آزمون اجرا می کند تا هرکدام از مدل های ایجادشده در مدت حرکت تیم به طرف پایین را وارسی کند.

مدل V راهی برای تجسم بخشیدن به چگونگی وارسی و اعتبارسنجی در ابتدای کار نرم افزار را فراهم می آورد.

مشکلاتی که به هنگام اجرای مدل ترتیبی خطی پیش می آید:

مدل ترتیبی خطی، قدیمی ترین و پرکاربردترین الگو برای مهندسی نرم افزار است.

مشکلات :

۱- پروژه های واقعی به ندرت جریان ترتیبی پیشنهادشده توسط این مدل را دنبال می کنند.

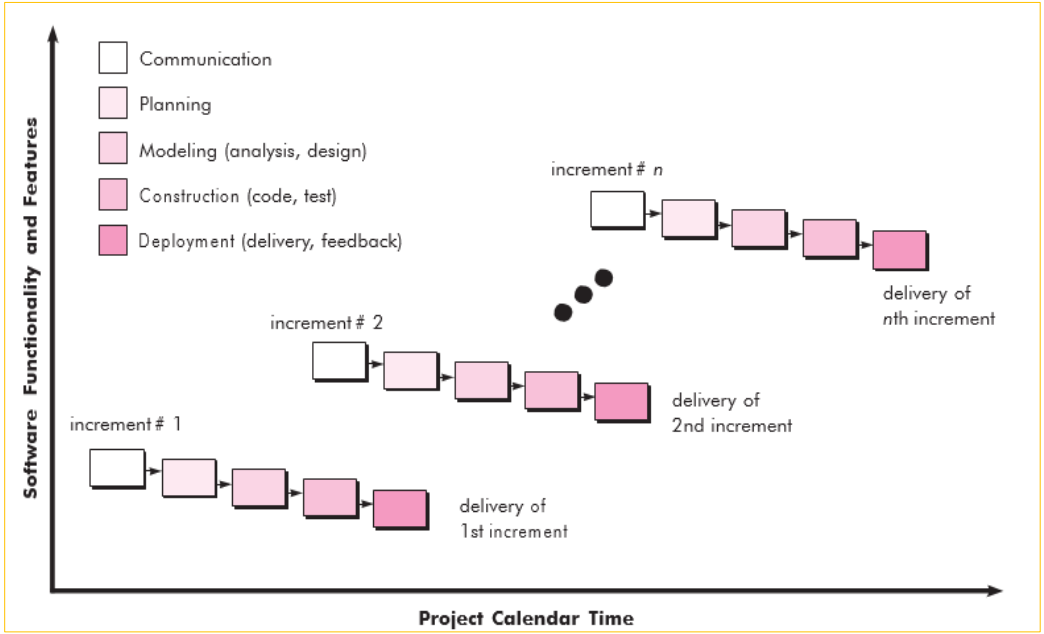
۲- غالباً برای مشتری دشوار است که همه نیاز خود را به وضوح بیان کند.

۳- مشتری باید حوصله داشته باشد.

Incremental Process Models مدل های فرآیند افزایشی

موارد انتخاب یک مدل فرآیند برای تولید نرم افزار به شیوه ی افزایشی :

- وضعیت هایی که در آنها خواسته های اولیه ی نرم افزار به خوبی تعریف شده اند ولی حوضه ی کلی تلاش های به عمل آمده در توسعه ی نرم افزار مانع از یک فرآیند خطی محض می شود.
- ضرورت نیاز به فرآهم کردن سریع مجموعه محدودی از عملکردهای نرم افزار برای کاربران و سپس پالایش و بسط براساس آن عملکردها در نتیجه های بعدی نرم افزار .



نرم افزار واژه پردازی که با استفاده از الگوی افزایشی توسعه یافته است، ممکن است اعمالی از قبیل:

مدیریت فایل، تولید و ویرایش مستندات در نسخه ی اول

قابلیتهای پیچیده تر ویرایشی و تولید مستندات در نسخه دوم

چک کردن املا و دستور در نسخه سوم

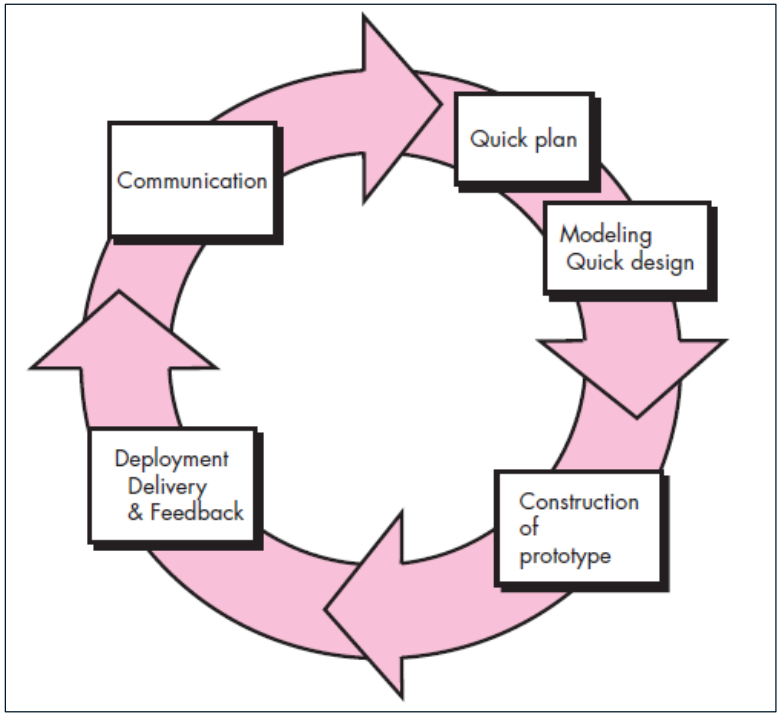
قابلیت های پیشرفته ی صفحه بندی را در نسخه چهارم

تحویل دهد.

Evolutionary
Process Models

مدل های فرآیند تکاملی

ساخت نمونه اولیه



الگوی ساخت نمونه اولیه با جمع آوری خواسته ها آغاز می شود.

مشتری و سازنده با هم ملاقات می کنند و اهداف کلی نرم افزار را تعیین می کنند. همه ی خواسته های معلوم را شناسایی می کند و زمینه هایی را مطرح می کند که تعریف بیشتر در آنها ضروری است. سپس یک طراحی سریع صورت می گیرد.

در طراحی سریع هدف اصلی ارائه آن دسته از ویژگی های نرم افزار است که مورد توجه کاربر می باشد مثل روش های واردکردن اطلاعات و فرمت های خارجی، طراحی سریع منجر به ساخت یک نمونه اولیه می شود.

نمونه اولیه مورد ارزیابی مشتری قرار گرفته و از آن برای پالایش نرم افزار مورد نظر استفاده می شود.

با تنظیم نمونه اولیه برای برآوردن نیازهای مشتری تکرار رخ می دهد و در عین حال سازنده بهتر می فهمد که چه نیازهایی باید برآورده شود.

در اکثر پروژه ها نخستین سیستمی که ساخته می شود چندان قابل استفاده نیست زیرا ممکن است بیش از حد آهسته باشد یا بیش از حد بزرگ باشد و یا استفاده از آن دشوار باشد و یا اینکه هر ۳ ایراد را با هم داشته باشد و چاره ای جز شروع دوباره وجود ندارد و باید نسخه دیگری ساخت که این مشکلات در آن حل شده باشد.

نمونه اولیه می تواند بعنوان نخستین سیستم عمل کند، یعنی دور انداخته شود.

بعضی نمونه های اولیه طبیعتی تکاملی دارند و به تدریج به سیستم واقعی تکامل می یابد.

دلایل مشکل ساز بودن ساخت نمونه اولیه

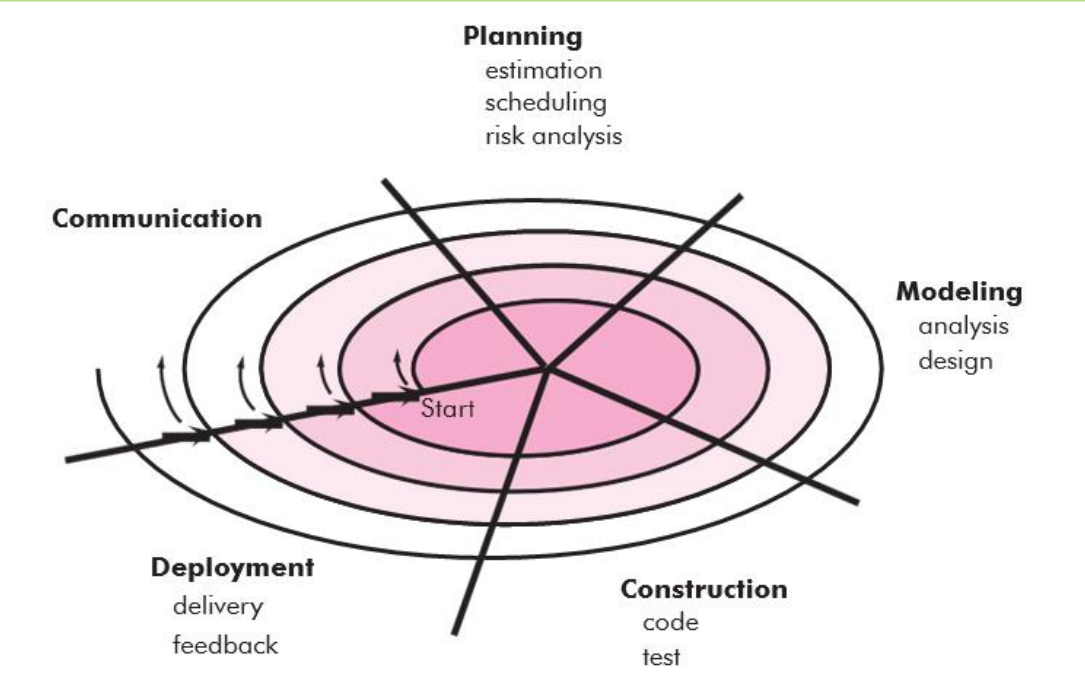
۱- افراد ذی نفع چیزی را می بینند که ظاهراً یک نسخه کاری از نرم افزار است ولی نمی دانند که این نسخه به درستی کار نمی کند و کیفیت کلی نرم افزار و قابلیت نگهداری درازمدت را ندارد.

۲- مهندس نرم افزار غالباً برای بکارگیری هرچه سریعتر نمونه اولیه در پیاده سازی آن کوتاهی می کند.

ممکن است از یک سیستم عامل یا زبان برنامه نویسی نامناسب استفاده شود

و یا حتی ممکن است یک الگوریتم ناکارآمد پیاده سازی شود .

مدل مارپیچی (حلزونی)



یک مدل فرآیند نرم افزاری تکاملی است که ماهیت تکراری مدل ساخت نمونه اولیه را با جنبه های کنترلی و سیستماتیک مدل ترتیبی خطی (آبشاری) تلفیق می کند.

این مدل پتانسیل لازم برای بسط سریع نسخه های تکاملی نرم افزار را داراست.

با استفاده از مدل مارپیچی، نرم افزار به صورت یک سری نگارش های تکاملی توسعه می یابد. طی نخستین دوره های تکرار، نگارش تکاملی، ممکن است یک مدل کاغذی یا یک نمونه اولیه باشد. طی تکرارهای بعدی هر بار نسخه کاملتری از سیستم مهندسی شده تولید می شود .

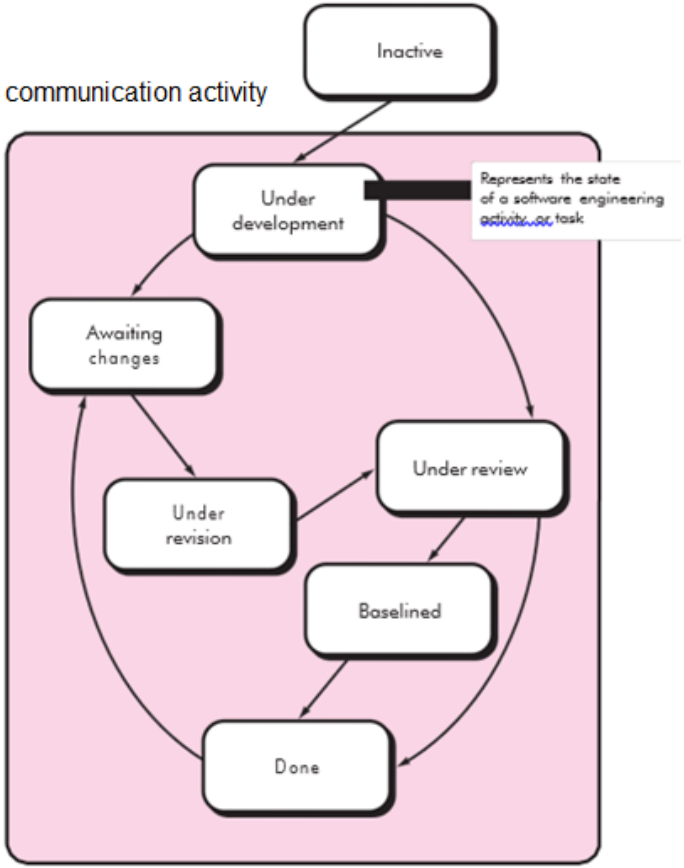
برخلاف سایر مدل های فرآیند کلاسیک که با تحویل نرم افزار پایان می یابند، مدل مارپیچی را می توان طوری تطبیق داد که در سرتاسر عمر نرم افزار کامپیوتری قابل به کارگیری باشد.

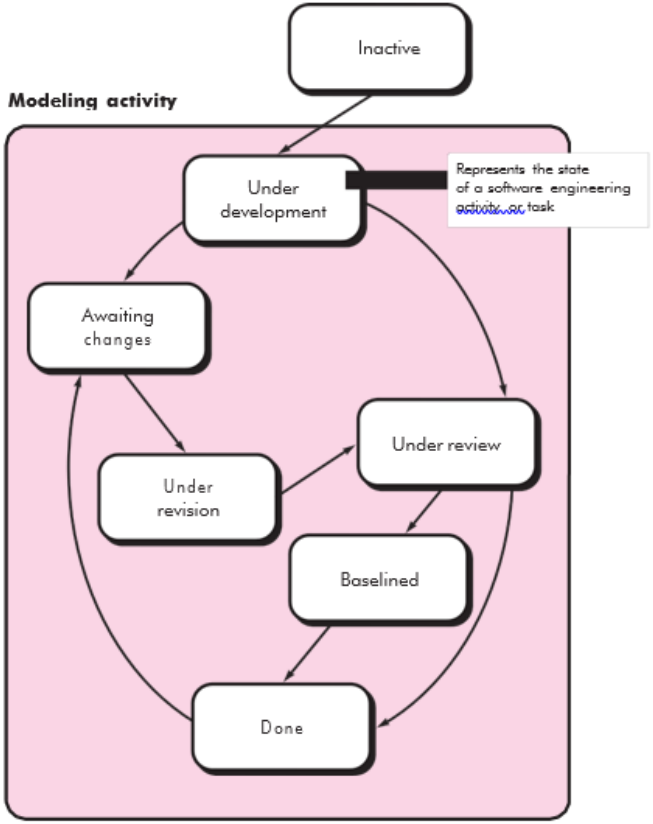
مدل مارپیچی یک روش واقع گرا برای توسعه نرم افزارها و سیستم هایی در مقیاس انبوه است.

اگر یک خطر عمده کشف و اداره نشود بدون شک مشکلاتی به بار خواهد آمد.

مدل توسعه همروند

Concurrent Models





مدل توسعه همروند (مهندسی همروند)به تیم نرم افزار این امکان را می دهد عناصر تکراری و همروند هرکدام از مدل های فرآیند را ارائه نماید.

نقاط ضعف فرایندهای تکاملی

۱- تهیه نمونه اولیه به دلیل قطعی نبودن تعداد چرخه های لازم برای ساخته شدن محصول، برای برنامه ریزی پروژه ایجاد مشکل می کند.

۲- حداکثر سرعت تکامل را تعیین نمی کنند.

۳- در فرایندهای نرم افزاری ، انعطاف پذیری و بسط پذیری باید بیش از کیفیت بالا مورد توجه قرار گیرد.

شکل طرحی از یک فعالیت با مدل توسعه همروند ارائه می دهد . این فعالیت مدل سازی ممکن است در هر زمان و در حالت های مختلف ایجاد شود برای مثال در ابتدای یک پروژه فعالیت برقراری ارتباط نخستین تکرار خود را به پایان رسانده و در حالت انتظار تغییرات قرار دارد در واقع فعالیت مدل سازی که هنگام کامل شدن ارتباط اولیه با مشتری در حالت غیرفعال قرارداشت اکنون دستخوش گذار به حالت تحت توسعه می شود ولی اگر مشتری متذکر شود که تغییراتی در خواسته ها باید صورت گیرد فعالیت تحلیل از حالت تحت توسعه به حالت انتظار تغییرات می رود.

درواقع مدل توسعه همروند یک سری رویداد تعریف می کند که باعث گذار از حالتی به حالت دیگر برای هر یک از فعالیت های مهندسی نرم افزار می شود .

برای مثال طی اولین مراحل طراحی یکی از کنش های اصلی در مهندسی نرم افزار که طی فعالیت مدل سازی انجام می شود یک ناسازگاری در مدل تحلیل کشف می شود این باعث تولید رویداد تصحیح مدل تحلیل می شود که گذار از کنش تحلیل خواسته ها را از حالت انجام شده به حالت انتظار تغییرات سبب می شود.

مدل های فرایند تخصصی

SPECIALIZED PROCESS MODELS

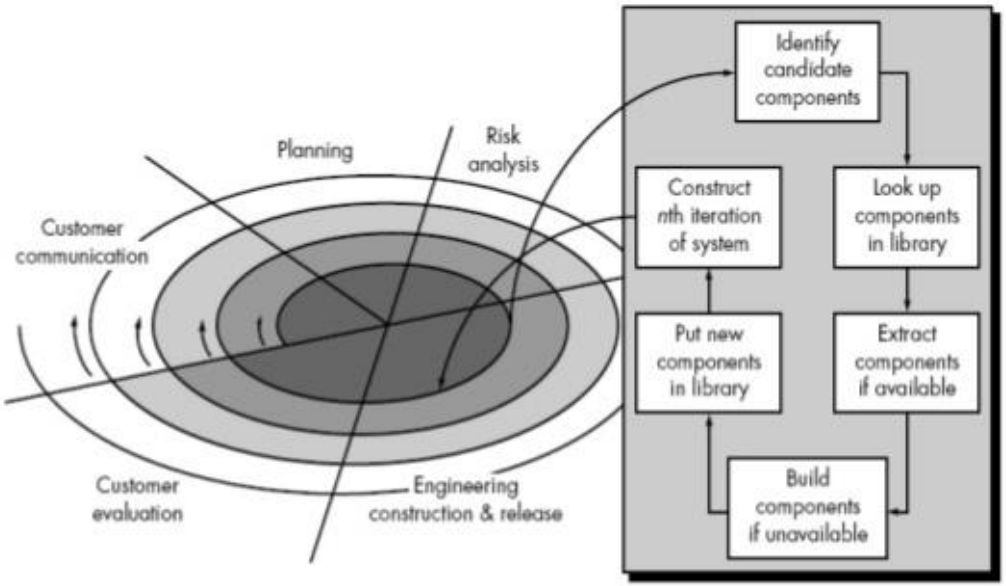
این مدل ها شامل بسیاری از ویژگی های یک یا چند مدل سنتی ارائه شده در بخش های قبلی می شوند. ولی، این مدل ها را معمولا هنگامی به کار می برند که یک روش مهندسی تخصصی یا روشی با مشخصات دقیق انتخاب می شود.

۱- توسعه مبتنی بر مولفه ها

۲- مدل روش های رسمی

۳- توسعه نرم افزار به روش جنبه گرا

توسعه مبتنی بر مؤلفه



مؤلفه های نرم افزاری آماده، توسط عده ای از فروشندگان این مؤلفه ها توسعه داده می شوند، عملکرد مورد نظر را با واسطه هایی مناسب فراهم می آورند، به طوری که مؤلفه را می توان به خوبی در سیستم درحال ساخت الحاق کرد.

توسعه مبتنی بر مؤلفه ها بسیاری از خصوصیات مدل مارپیچی را در برمی گیرد.

توسعه مبتنی بر مؤلفه شامل موارد زیر است:

۱- محصولات مبتنی بر مؤلفه های موجود از نظر دامنه ی کاربرد مورد نظر بررسی و ارزیابی می شوند.

۲- مسائل مربوط به انسجام مؤلفه ها در نظر گرفته می شود.

۳- برای چیدمان مؤلفه ها یک معماری نرم افزار طراحی می شود.

۴- مؤلفه ها در این معماری قرار داده می شوند.

۵- آزمونی جامع برای حصول اطمینان از عملکرد درست بدست می آید.

The Formal Methods Model مدل روش های رسمی

شامل مجموعه ای از فعالیت ها می شود که به مشخص کردن ریاضی و رسمی نرم افزار کامپیوتری منجر می شود.

روش های رسمی، مهندسی نرم افزار را قادر می سازد که با اعمال یک نظم ریاضی شدید، سیستم کامپیوتری را مشخص کند، بست دهد و وارسی کند.

شکل دیگر از این روش که مهندسی نرم افزار اتاق تمیز cleanroom software engineering نامیده می شود درحال حاضر توسط برخی از سازمان های نرم افزاری به کار می رود.

مدل روش های رسمی گرچه چندان عمومیت نخواهد یافت اما نوید بخش نرم افزاری عاری از نقص است.

با این حال، ملاحظات مربوط به قابلیت اجرای آن در محیط های تجارتی چنین اعلام شده است:

۱- توسعه این مدل ها، بسیار وقت گیر و پرهزینه است.

۲- از آنجا که تعداد محدودی از نرم افزارسازان دارای زمینه ی لازم برای اجرای روش های رسمی هستند آموزش موردنیاز است.

۳- استفاده از مدل ها به عنوان راهکار ارتباطی با مشتریانی که دید فنی ندارند دشوار است.

توسعه ی نرم افزار به روش جنبه گرا

Aspect-Oriented Software Development(AOSD)

AOSD یک الگوی مهندسی نسبتاً جدید است که رویکردی فرآیندی و روش شناختی برای تعریف، مشخص سازی، طراحی و ساخت جنبه ها ارائه می دهد.

هر فرآیند نرم افزار که انتخاب شود، سازندگان نرم افزارهای پیچیده، مجموعه ای از ویژگی ها، عملکردها و محتویات اطلاعاتی متمرکز را پیاده سازی می کنند.

این خصوصیات نرم افزاری متمرکز به صورت مؤلفه هایی مثلا کلاس های شی گرا، مدل سازی و سپس در حیطه یک معماری سیستم بنا می شوند.

با پیچیده تر شدن سیستم های کامپیوتری مدرن، دغدغه های خاصی کل معماری را دربر می گیرد. هنگامی که این دغدغه ها در وظایف، ویژگی و اطلاعات سیستمی با یکدیگر تلاقی می کند غالبا از آنها به عنوان دغدغه های متلاقی یاد می شود. خواسته های جنبه گرا، آن دسته از دغدغه های متلاقی را تعریف می کنند که بر معماری نرم افزار تأثیر می گذارند.

فرایند یکپارچه

فرآیند یکپارچه (UP) از جهاتی تلاشی برای گرد هم آوردن بهترین ویژگی ها و خصوصیات مدل های فرآیند سنتی است، ولی آنها را به شیوه ای مشخص می کند که بسیاری از بهترین اصول توسعه نرم افزار چابک را پیاده سازی می کند.

در فرایند یکپارچه اهمیت برقراری ارتباط با مشتریان و روش های ساده برای توصیف دیدگاه مشتریان

(use case) یک سیستم به خوبی درک می شود.

در این فرایند بر اهمیت نقش معماری نرم افزار تاکید می شود.

در این فرایند ، یک جریان فرایند مبتنی بر تکرار و افزایشی پیشنهاد می شود.

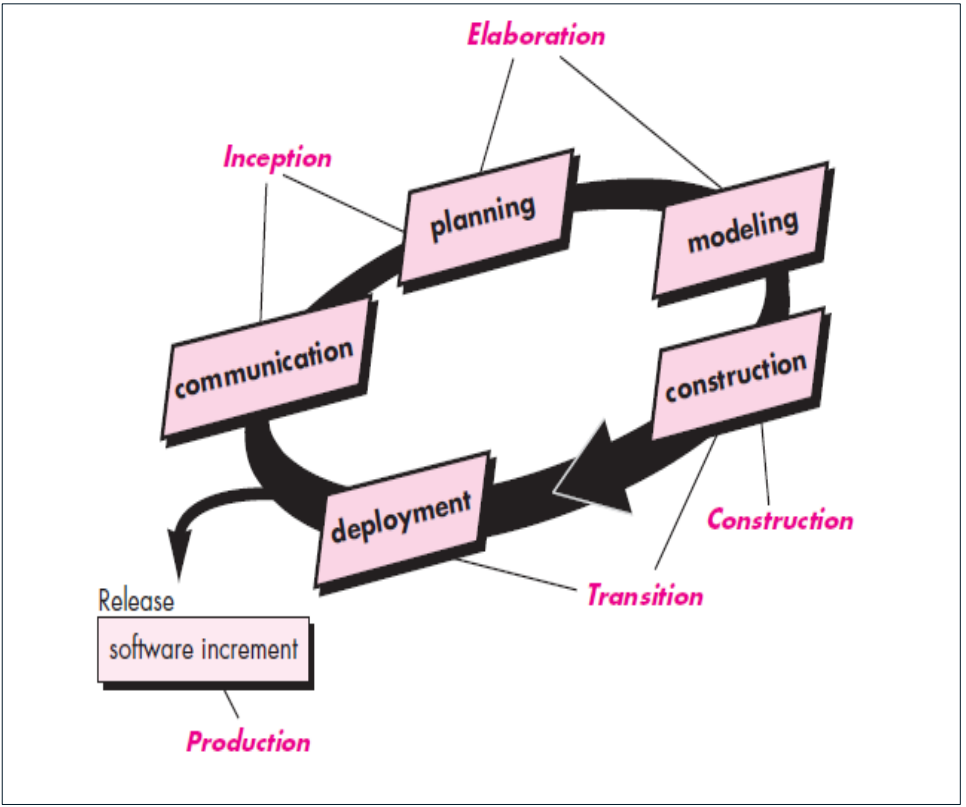
در اوایل دهه ۱۹۹۰ ، کار روی یک روش یکپارچه آغاز شد که بهترین ویژگی های هرکدام از روش های طراحی و

تحلیل شی گرا را تلفیق می کرد و ویژگی هایی از سایر کارشناسان در مدل سازی شی گرا بر آن منطبق می ساخت.

نتیجه، زبان مدل سازی یکپارچه (UML) است که حاوی یک نمادگذاری قدرتمند برای مدلسازی و توسعه سیستم های شی گرا است.

Phases of the Unified Process

مراحل فرآیند یکپارچه



مرحله اول : شامل هر ۲ فعالیت برقراری ارتباط با مشتریان و برنامه ریزی می شود.

مرحله شناخت: شامل فعالیت های برقراری ارتباط و مدل سازی در مدل فرآیند کلی می شود.

در این مرحله یوزکیس های مقدماتی که به عنوان بخشی از مرحله آغازین ایجاد شوند، پالایش یافته و بسط داده می شوند.

مرحله ساخت : هم ارز فعالیت ساخت است که برای فرایند نرم افزار کلی تعریف شد.

مرحله گذار: شامل آخرین مراحل در فعالیت ساخت در مدار کلی و اولین بخش از استقرار در مدل کلی می شود.

مرحله تولید: منطبق بر فعالیت استقرار در فرآیند کلی است.

مدل های فرایند تیمی و شخصی

PERSONAL AND TEAM PROCESS MODELS

بهترین فرآیند نرم افزار ، فرآیندی است که به کسانی که کار می کنند نزدیک باشد.

فرآیند نرم افزار شخصی Personal Software Process (PSP)

فرایند نرم افزار تیمی Team Software Process (TSP)

Personal Software Process (PSP)

فرآیند نرم افزار شخصی

هر سازنده ای برای ساختن نرم افزار کامپیوتری از یک فرآیند استفاده می کند.

این فرآیند ممکن است برحسب اتفاق شکل گرفته باشد یا با هدفی خاص ممکن است روزانه تغییر کند.

ممکن است اثربخش نباشد مؤثر نباشد یا حتی موفقیت آمیز هم نباشد ولی فرآیندی وجود دارد.

PSP بر اندازه گیری شخصی محصول کاری تولید شده و کیفیت حاصل از محصول کاری تأکید دارد.

فعالیت های چارچوبی در مدل PSP

۱) برنامه ریزی Planning

۲) طراحی سطح بالا High-level design

۳) مرور طراحی سطح بالا High-level design review

۴) توسعه Development

۵) پایان کار Postmortem

فرایند نرم افزار تیمی

Team Software Process (TSP)

هدف TSP تشکیل یک تیم پروژه خود هدایت گراست که سازماندهی برای تولید نرم افزارهای با کیفیت را خود عهده دار می شود.

فعالیت های زیر برای TSP تعریف می شود:

۱- تشکیل تیم های خود هدایت گری که کار خود را برنامه ریزی و پیگیری می کنند، اهداف را تعیین می کنند و خود به تعیین فرآیندها و طرح ها اقدام می نمایند. این تیم ها می توانند تیم های نرم افزاری محض یا تیم های محصولات انسجام یافته شامل ۳ تا حدود ۲۰ مهندس باشد.

۲- نشان دادن شیوه ی راهبری و ایجاد انگیزه در تیم ها به مدیران و چگونگی کمک به آنها در حفظ حداکثر کارایی

۳- شتاب بخشیدن به بهبود فرآیند نرم افزار با نهادینه ساختن CMM سطح ۵.

۴- فراهم ساختن دستور العمل بهسازی برای سازمان های بالغ

۵- تسهیل آموزش دانشگاهی مهارت های تیمی در سطح صنعتی.

فعالیت های چارچوبی تعریف شده در TSP

۱- آغاز پروژه

۲- طراحی سطح بالا

۳- پیاده سازی

۴- انسجام دهی

۵- آزمون

انتخاب یک مدل فرایند

SafeHome

Selecting a Process Model, Part 1

The scene: Meeting room for the software engineering group at CPI Corporation, a (fictional) company that makes consumer products for home and commercial use.

The players: Lee Warren, engineering manager; Doug Miller, software engineering manager; Jamie Lazar, software team member; Vinod Raman, software team member; and Ed Robbins, software team member.

انتخاب یک مدل فرایند(بخش ۲)

SafeHome

Selecting a Process Model, Part 2

The scene: Meeting room for the software engineering group at CPI Corporation, a company that makes consumer products for home and commercial use.

The players: Lee Warren, engineering manager; Doug Miller, software engineering manager; Vinod and Jamie, members of the software engineering team.

پایان فصل ۲

مهندسی نرم افزار

فصل سوم : توسعه ی چابک

AGILE DEVELOPMENT

مدرس:

فرشید شیرافکن

دانشجوی **دکتری دانشگاه تهران**

(کارشناسی و کارشناسی ارشد : کامپیوتر نرم افزار) (دکتری: بیو انفورماتیک)

توسعه چابک چیست؟

مهندسی نرم‌افزار چابک، تلفیقی از یک فلسفه و مجموعه‌ای از دستورالعمل‌های توسعه است.

این فلسفه مشوق موارد زیر است:

- جلب رضایت مشتری
- تحویل افزایشی نرم‌افزار از همان ابتدای پروژه.
- تیم های پروژه ای کوچک با انگیزه بالا
- روش های غیر رسمی
- حداقل محصولات مهندسی نرم افزار
- سادگی کلی مهندسی نرم افزار

توسعه چابک

چه کسی این کار را انجام می‌دهد؟ مهندسین نرم‌افزار و طرفهای ذی‌نفع در پروژه با یکدیگر کار می کنند و یک تیم چابک را تشکیل می‌دهند.

تیمی که سازمان‌دهی‌اش بر عهده خودش است. سیستم چابک به ارتباطات و همکاری میان همه افراد سیستم رسیدگی می‌کند.

اهمیت؟ مهندسی نرم‌افزار چابک، برای مهندسی سنتی گروه و انواع معینی از پروژه‌های نرم‌افزاری جایگزین منطقی ارائه می‌کند.

مراحل کار: فعالیت‌های چارچوبی پایه(ارتباطات ،برنامه‌ریزی،مدل‌سازی، ساخت و استقرار) به قوت خود باقی خواهند ماند،ولی به یک مجموعه وظایف کمینه تغییر شکل می‌دهند.

محصول کار: یک نرم‌افزار عملیاتی است که در تاریخ مناسب به مشتری تحویل شود.

چگونه اطمینان حاصل کنیم که کار، درست انجام شده‌است؟ تولید نرم‌افزار قابل تحویل و رضایت مشتری

مفاد بیانیه توسعه نرم‌افزار چابک

- افراد و تعامل ها را بر فرآیندها و ابزارها برتری دهیم.
- نرم افزار عملیاتی را بر مستندات جامع برتری دهیم.
- همکاری با مشتری را بر مذاکره و قرارداد برتری دهیم.
- پاسخ به تغییر را بر دنبال کردن یک برنامه برتری دهیم.

روش چابک : فقط برای مهندسی نرم‌افزار نیست بلکه به عنوان فلسفه‌ای جایگزین برای همه کارهای نرم‌افزاری قابل استفاده است.

یکی از بارزترین ویژگی‌های روش چابک : توانایی آن در کاهش دادن هزینه‌های ناشی از تغییر است.

در مدل‌ها معرفی شده در فصل ۲، ضعف و سستی کسانی که نرم‌افزارها را می‌سازند،فراموش می‌شود.

مهندسان نرم افزار، روبات نیستند و در سبک کاری باهم تفاوت‌های بزرگ دارند.از جمله :

در سطح مهارت ، خلاقیت، نظم‌و انضباط ،سرعت عمل.

با اعمال انضباط یا تحمل می‌توان نقاط ضعف افراد را اداره کرد.

چون سازگاری درکنش از نقاط ضعف انسان است،روش‌هایی با انضباط بالا شکننده‌اند.

چابکی در حیطه کار مهندسی نرم افزار

هنگام توصیف یک فرآیند نرم افزار مدرن، چابکی واژه‌ای است که به وفور به گوش می‌رسد.

تیم چابک تیمی فرز و چالاک است که قادر است به تغییرات پاسخ مناسب بدهد.

تغییر چیزی است که در توسعه نرم‌افزار ،بسیار با آن مواجه می‌شویم.

- تغییرات در نرم‌افزارهایی که در حال ساخته‌شدن هستند ،
- تغییرات در اعضای تیم ،
- تغییرات به دلیل فناوری جدید ،

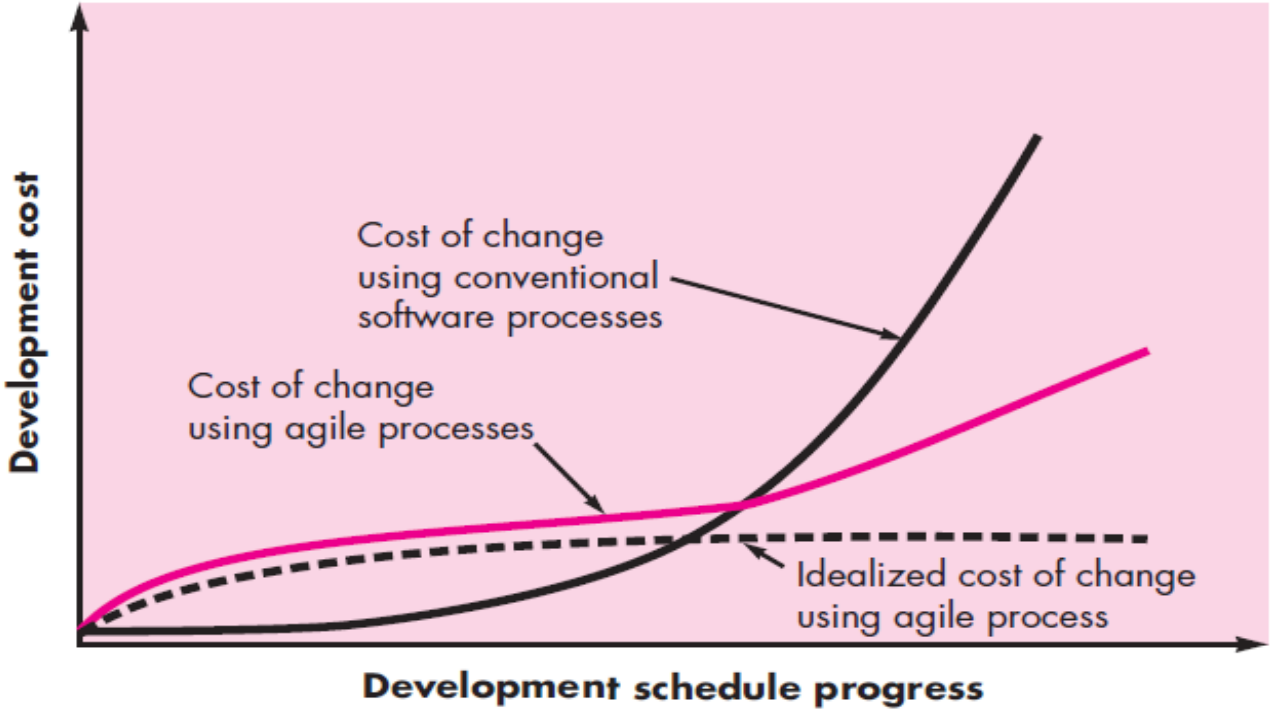
تغییرات از هر نوع ممکن است بر محصول در حال ساخت یا محصول نهایی ، تاثیرگذار باشند.

سیستم چابک می‌داند که نرم‌افزار توسط افرادی توسعه می‌یابد که در قالب تیمی کار می‌کنند و مهارت‌های این افراد و توانایی ایشان در همکاری ، هسته اصلی موفقیت پروژه است.

فراگیر بودن تغییر ، دلیل اصلی برای چابکی است.

مرتکب این اشتباه نشوید که چابکی این مجوز را به شما می‌دهد که بدون برنامه‌ریزی به راهکار برسید. یک فرآیند لازم است و انضباط ضروری است .

چابکی و هزینه‌های تغییر



یک فرآیند چابک با طراحی خوب ، منحنی هزینه‌ی تغییر را تسطیح می‌کند و به این‌ترتیب ، سیستم نرم‌افزاری قادر به پاسخ‌گویی به تغییرات در اواخر پروژه خواهد بود، بدون اینکه ضربه‌ای قابل ملاحظه از نظر زمان و هزینه به پروژه وارد آید.

فرآیند چابک هزینه‌ی تغییرات را کاهش می‌دهد چون نرم‌افزار در قالب چند گام روانه می‌شود و تغییرات را در یک گام بهتر می‌شود کنترل کرد.

فرایند چابک چیست ؟

هر فرآیند نرم‌افزار چابک، به گونه ای مشخص می شود که تعدادی از فرض‌های کلیدی را درباره اکثریت پروژه‌های نرم‌افزاری پاسخ گو باشد:

۱)پیش‌بینی اینکه کدام خواسته‌های نرم‌افزاری باقی است و کدام‌یک تغییر می‌کند ، دشوار است.

۲) برای بسیاری از انواع نرم‌افزارها ، طراحی و ساخت به طور موازی انجام می‌شود که دشوار است.

۳) تحلیل ، طراحی ، ساخت و آزمودن ، ممکن است به آن اندازه که دوست داریم ، قابل پیش‌بینی نباشد.

پرسش :

چگونه فرآیندی ایجاد کنیم که قادر به مدیریت موارد غیر قابل پیش‌بینی باشد؟ فرآیندی با انطباق پذیری بالاتر.

در یک فرآیند چابک باید روند انطباق به طور افزایشی انجام پذیرد. برای دستیابی به آن نیاز به بازخورد از مشتری دارد.

Agility Principles

اصول چابکی

دوازده اصل در پیمان چابک :

۱- جلب رضایت مشتری با تحویل زود هنگام و پیوسته، بیشترین اولویت را نزد ما دارد.

۲- پذیرا بودن تغییرات در خواسته‌ها حتی در اواخر فرآیند توسعه.

۳- تحویل پیوسته نرم‌افزارها از دو هفته تا دو ماه .

۴- دست‌اندرکاران و افراد تجاری باید در سرتاسر پروژه هرروز با هم کار کنند.

۵- سپردن پروژه به افراد با انگیزه، فراهم‌سازی محیط و پشتیبانی مورد نیاز آنها و اطمینان کردن به آنها در انجام کارها.

۶- اثر بخش‌ترین و موثرترین روش انتقال اطلاعات به درون و بیرون سیستم توسعه ،گفتگوی رودررو است.

Agility Principles

اصول چابکی

۷- نرم‌افزاری کاری ،میزان اصلی در سنجش پیشرفت است.

۸- فرآیندهای چابک ، توسعه پایدار را ارتقا می‌بخشد.

۹- توجه پیوسته به اعتلای فنی و طراحی خوب ، باعث بهبود افزایش چابکی می‌شود.

۱۰- سادگی ضروری است.

۱۱- بهترین معماری‌ها ، خواسته‌ها و طراحی‌ها از تیم های خودسازمان‌دهی شده ظهور می‌کنند.

۱۲- تیم در بازه‌های منظم، بازخوردی از میزان بهبود اثربخش خود ارائه می‌دهد و سپس رفتار خود را مطابق این بازخورد تنظیم می‌کند.

سیاست توسعه چابک

The Politics of Agile Development

درباره مزایا و قابلیت چابک بحث‌وجدل فراوان است.

جیم اسمیت می‌گوید: روش‌شناسان ،یک مشت آدم‌های فاقد خلاقیت هستند که ترجیح می‌دهند مستنداتی بدون نقص تهیه کنند تا اینکه سیستمی کاری ارائه دهند که نیازهای تجاری را برآورده کند.

وی می‌گوید: روش‌شناسان چابک یک مشت نفوذگر با استعدادند که وقتی تلاش می‌کنند اسباب‌بازی های خود را بزرگ کنند و به نرم‌افزاری در سطح شرکت‌ها تبدیل کنند، کلی ذوق زده می‌شوند.

نکته مهم اینکه با در نظر گرفتن بهترین ایده‌ها از هر دو مکتب ،بیشترین بهره عاید خواهد شد و چیزی از تخریب دیگری به‌دست نخواهد آمد.

عوامل انسانی

در این توسعه به عوامل انسانی تاکید فراوان دارند .

توسعه چابک بر استعدادها و مهارت‌های افراد و شکل‌دهی به فرآیند براساس افراد و تیم‌ها موجود تاکید دارد.

فرایند باید بر اساس نیازهای افراد و تیم ها شکل پیدا کند نه بر عکس.

خصوصیات زیر باید در میان افراد سیستم و خود سیستم وجود داشته باشد :

۱- رقابت : Competence

شامل استعداد ذاتی، مهارت‌های خاص مرتبط با نرم‌افزار و آگاهی کلی از فرآیندی است که تیم برگزیده است.

۲- کانون توجه مشترک : Common focus

با وجود وظایف هر کدام از اعضای تیم هدف واحد تحویل نسخه جدیدی از نرم‌افزار به مشتری در زمان مقرر.

۳- همکاری : Collaboration

داشتن همکاری در هزینه‌های فرآیند و انتشار اطلاعاتی که برای مشتری ارزش تجاری در بر داشته باشد.

۴- توانایی تصمیم‌گیری : Decision-making ability

آزادی کنترل سرنوشت خود، یعنی اجازه تصمیم‌گیری برای مسائل فنی و پروژه.

۵- توانایی حل مسئله با منطق فازی : Fuzzy problem-solving ability

ممکن مسئله‌ای که امروز در حال حل کردن آن است،فردا مسئله‌ای باشد که دیگر نیازی به حل آن نباشد،ولی درس هایی که از حل هر مسئله گرفته می‌شود ،ممکن است بعدا در پروژه به‌کار آید.

۶- احترام و اطمینان متقابل : Mutual trust and respect

باید تیم قوام یافته باشد و اعضای تیم چنان به هم پیوسته باشند که کلیت حاصل چیزی بیش از مجموع اجزای تشکیل‌دهنده باشد.

۷- خودسازمان‌دهی :Self-organization

- تیم چابک خودش را سازمان‌دهی می‌کند تا کارها به انجام برسد .
- تیم فرآیند را سازمان‌دهی می‌کند تا به بهترین نحو در محیط محلی اسکان یابد.
- تیم ، زمان‌بندی را سازمان‌دهی می‌کند.

EXTREME PROGRAMMING (XP)

برنامه‌نویسی حدی

پرکاربردترین رویکرد در توسعه نرم‌افزار به روش چابک است.

ارزش های Xp

۱- **ارتباطات** : همکاری نزدیک و در عین حال غیر رسمی میان مشتریان و سازندگان.

۲- **سادگی**: ایجاد یک طراحی ساده که به سادگی در قالب کد نویسی قابل پیاده سازی است.

۳- **بازخورد**: از سه منبع قابل حصول است: نرم افزار، مشتری، سایر اعضای تیم.

۴- **جرات**: تیم xp چابک باید انضباط (جرات) لازم را برای طراحی امروز و خواسته های آینده را داشته باشد.

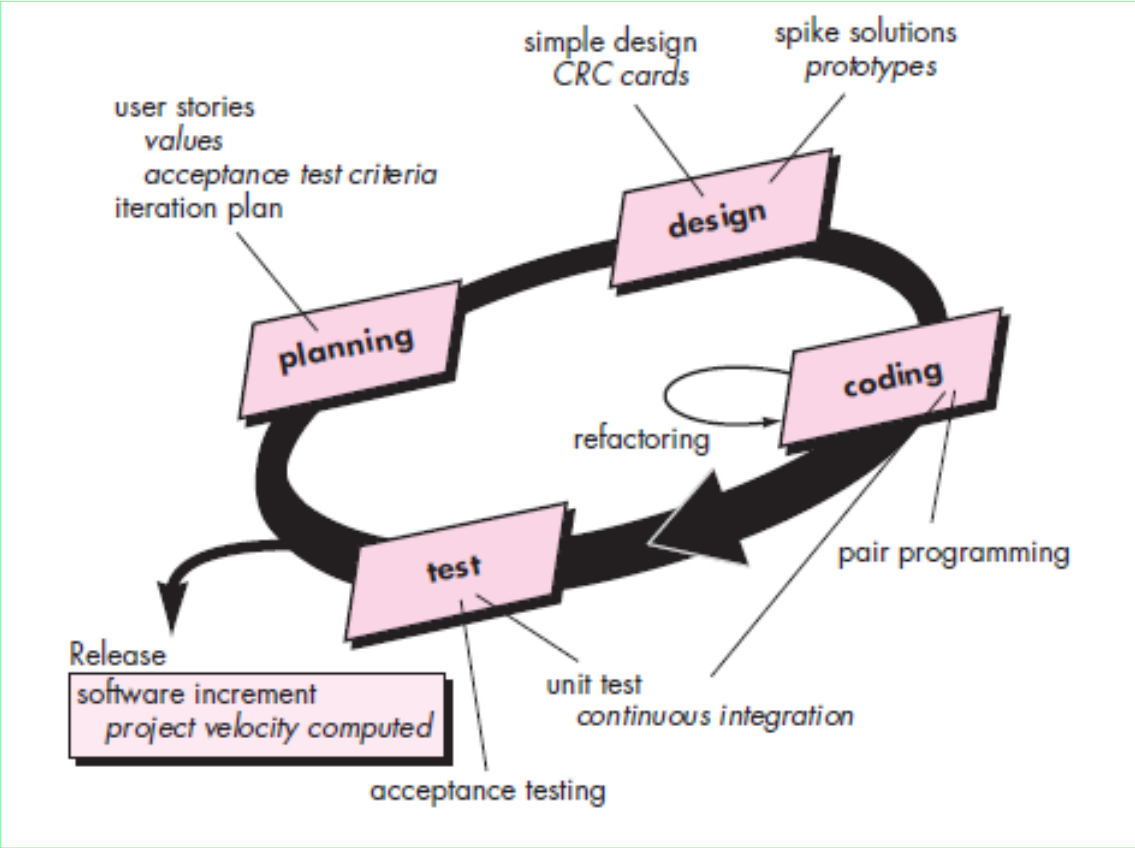
۵- **احترام** : در میان اعضای تیم و سایر طرف های ذی نفع

Xp پاسخ این سئوال است که چقدر می توان کوچک عمل کرد و هنوز نرم افزارهای بزرگ ساخت.

فرایند xp

The XP Process

این فرایند در حیطه چهار فعالیت چارچوبی رخ می دهد:



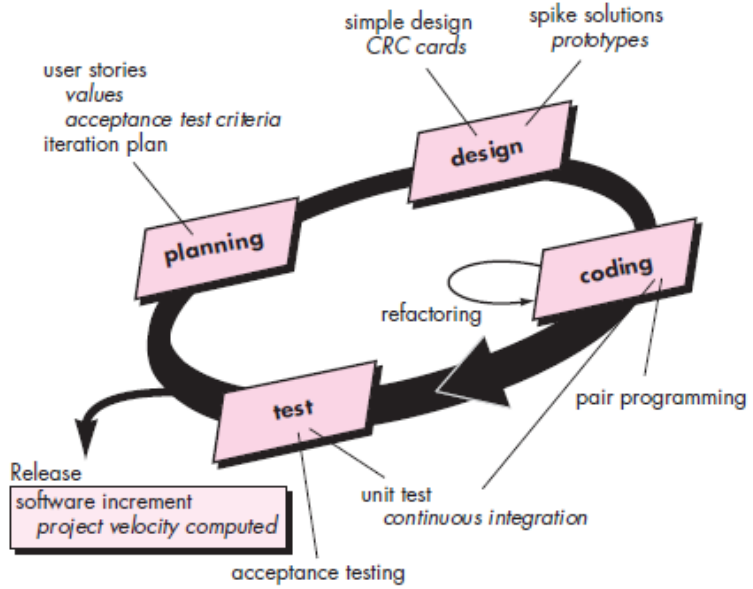
۱- برنامه ریزی

۲- طراحی

۳- کد نویسی

۴- آزمون

فرایند xp (برنامه ریزی)



۱- گوش کردن و جمع آوری خواسته های مشتری و شناخت حیطه تجاری مناسب

۲- تعیین خروجی ها از روی "داستان ها"

۳- اولویت بندی یک خروجی به خروجی دیگر (خروجی ها و برنامه ها) توسط مشتری

۴- گروه بندی اولویت ها با همکاری مشتریان و سازندگان

چگونگی گروه بندی و ترتیب داستان ها:

۱- همه نیازها بلافاصله پیاده سازی شود(در عرض چند هفته)

۲- اولویت های بالاتر زودتر انجام شود

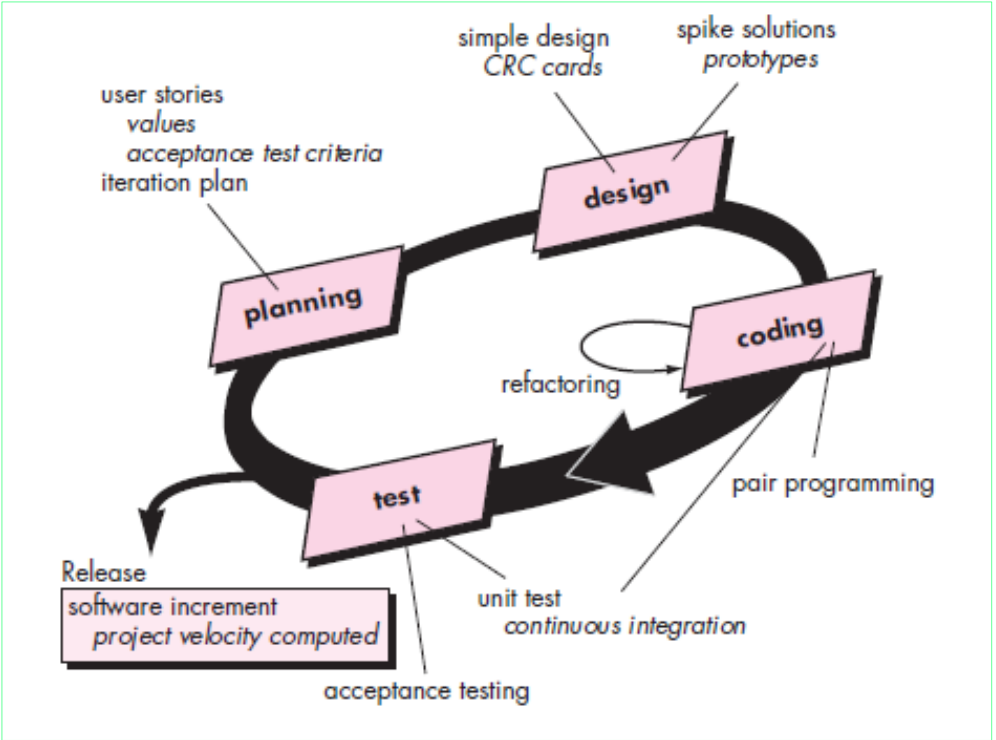
۳- اولویت بندی با توجه به مقدار ریسک

در این مرحله سرعت پروژه نیز با توجه به نیازهای مشتری محاسبه می شود که:

۱- کمک به برآورد تاریخ های تحویل و زمان بندی

۲- جلوگیری از زیاده روی در یک مورد

فرایند xp (طراحی)



طراحی XP از اصل KIS پیروی می کند.

کارت های CRC (کلاس-مسئوولیت-همکار)

راهکار خیزشی (spike solution)

بازآرایی

محصول گذرا

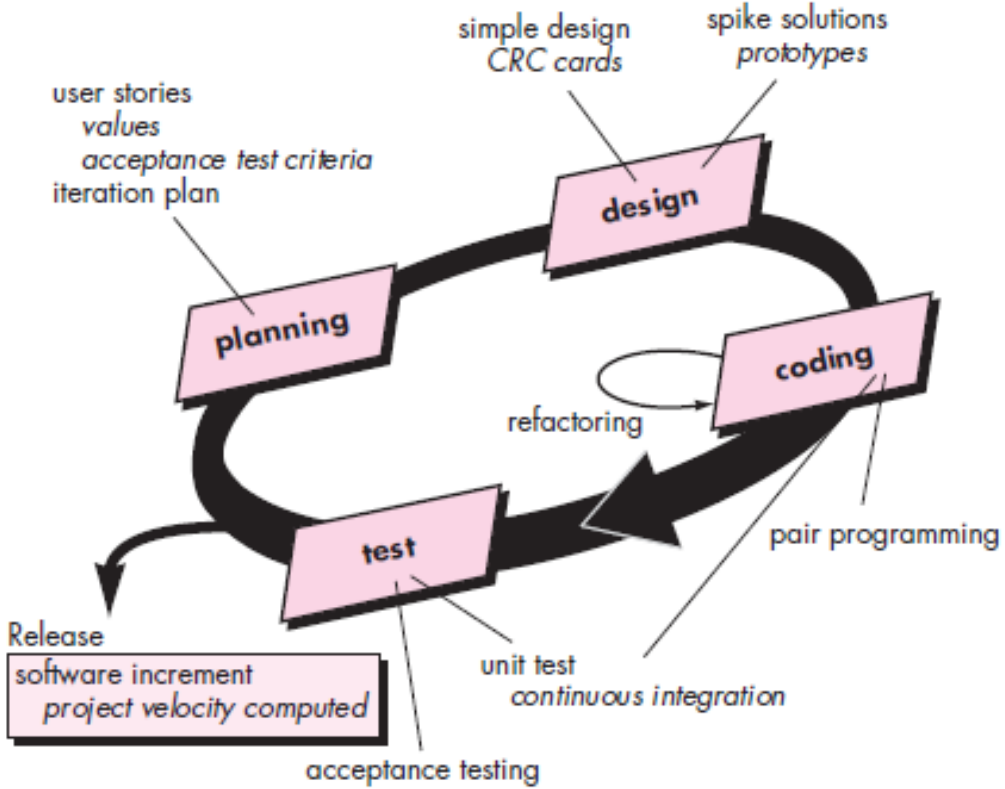
فرایند xp (کدنویسی)

پس از توسعه یافتن داستان ها و انجام شدن کارهای طراحی مقدماتی تیم به کد نویسی نمی پردازد بلکه یک سری آزمون واحد تهیه می کند.

برنامه نویسی جفتی :

دو نفر با هم روی یک ایستگاه کاری کار کنند و کد مربوط به یک داستان را بنویسند.

فرایند xp (آزمون)



- آزمون واحدها قبل از شروع کدنویسی
- آزمون های پذیرش Xp (آزمون مشتری)

برنامه‌نویسی حدی‌صنعتی IXP

تکاملی ارگانیک از Xp است.

الهام گرفته از کمینه‌گرایی، مشتری‌مداری و آزمون‌مداری Xp .

بیشترین تفاوت IXP با XP

۱- اعمال مدیریت بیشتر

۲- گسترش نقش مشتریان

۳- ارتقای روش‌های فنی

شش عمل جدید IXP

۱- ارزیابی آمادگی :

پیش از شروع یک پروژه ی IXP ، سازمان باید ارزیابی آمادگی را انجام دهد. در این عمل اطمینان حاصل می‌شود که:

۱- یک محیط توسعه مناسب وجود دارد که IXPرا پشتیبانی میکند

۲- تیم شامل مجموعه مناسبی از طرف های ذی نفع است.

۳-سازمان دارای یک برنامه کیفیت ممتاز است و بهبود مستمر را پشتیبانی می‌کند.

۴- فرهنگ سازمانی،پشتیبان ارزش‌های جدید یک تیم جدید است.

۵-جامعه ی وسیعتر پروژه از افراد مناسبی تشکیل شده است.

شش عمل جدید IXP

۲- جامعه پروژه:

وقتی که قرار باشد که برای یک پروژه مهم در سازمانی بزرگ از روش Xp استفاده شود مفهوم تیم باید به جامعه تغییر شکل پیدا کند.

این جامعه شامل :

یک متخصص فناوری ومشتریانی که در موفقیت پروژه نقش محوری دارند

و نیز شامل طرف های ذی‌نفع نظیر کارمندان حقوقی ،کارمندان بخش تولید و فروش و... باشد.

در IXP اعضای جامعه و نقش هرکدام از آنها باید به صراحت تعریف می‌شود.

۳- چارتر کردن پروژه :

تیم Ixp ، خود به ارزیابی پروژه می پردازد تا تعیین کند آیا یک توجیه تجاری مناسب برای پروژه وجود دارد و آیا پروژه اهداف و مقاصد کلی سازمان راپیش می برد یا خیر.

با عمل چارتر کردن ،حیطه پروژه برای تعیین چگونگی کامل شدن ،توسعه یافتن یا جایگزین ساختن سیستم‌ها یا فرآیندهای موجود تعیین می‌شود.

۴- مدیریت مبتنی بر آزمون :

یک سری، مقاصد قابل سنجش و سازوکارهایی برای رسیدن به این مقاصد.

۵- بازنگری :

مرور و بازبینی مسائل رویدادها و درس‌های آموخته‌شده با هدف بهبود بخشیدن به فرآیند Ixp

۶- آموزش پیوسته :

چون اموزش بخش حیاتی از بهبود مستمر است، اعضای تیم XP تشویق می شوند تا روش ها و تکنیک‌های جدیدی را بیاموزند که به محصول با کیفیت بالاتر منجر شود.

مشاجره XP

۱- متغیر بودن خواسته ها

۲- نیازهای متناقض مشتریان

۳- خواسته ها به صورت غیر رسمی بیان می‌شود.

۴- فقدان طراحی رسمی

مدل های فرایند چابک

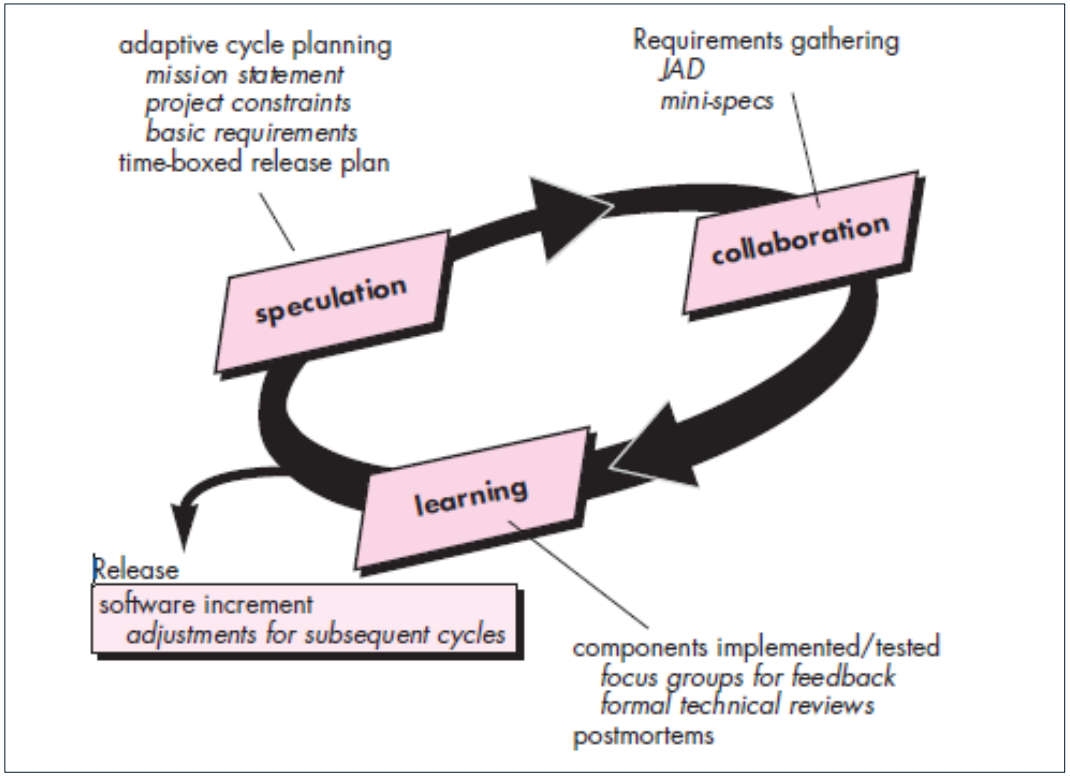
- ✓ برنامه نویسی حدی (XP)
- ✓ توسعه وفقی نرم افزار(ASD)
- ✓ روش توسعه سیستم های پویا (DSDM)
- ✓ کریستال
- ✓ توسعه ویژگی محور (FDD)
- ✓ توسعه نرم افزار ناب (LSD)
- ✓ مدل سازی چابک (AM)
- ✓ فرآیند یکپارچه چابک (AUP)

Adaptive Software Development

توسعه وفقی نرم افزار(ASD)

تکنیکی برای ساخت نرم افزارها و سیستم های پیچیده.

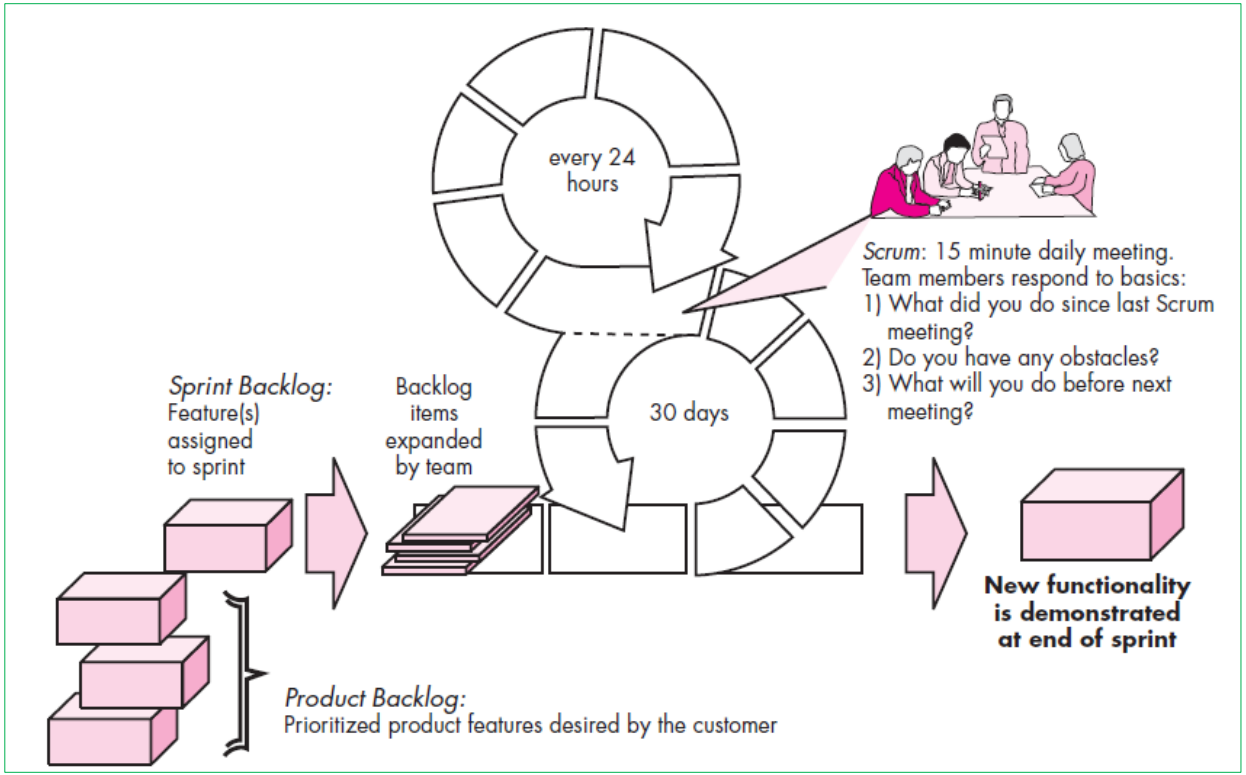
تاکید بر همکاری انسانی و خود سازمان دهی تیمی .



مراحل چرخه حیات ASD

- تعمق(تفکر)
- همکاری
- یادگیری

اسکرام(Scrum)



اصول اسکرام با بیانیه چابکی سازگاری دارد.

شامل فعالیت‌های چارچوبی زیر است :

خواسته‌ها ، تحلیل ، طراحی ، تکامل ، تحویل.

تاکید اسکرام ...

- **فهرست جامانده‌ها** (Backlogs)
- **sprint ها .**
- **نشست‌های اسکرام.**
- **دموها.**

روش توسعه سیستم‌های پویا (DSDM)

Dynamic Systems Development Method (DSDM)

چارچوبی بر ساخت و نگهداری سیستم‌هایی فراهم می آورد که قید و بندهای زمانی فشرده را از طریق به کار گیری نمونه ی اولیه در یک محیط پروژه کنترل شده برآورده می سازند.

چرخه حیات DSDM :

۱- امکان‌سنجی : آیا نرم‌افزار کاندیدای لایقی برای فرآیند DSDM هست یا خیر.

۲- مطالعه تجاری : خواسته‌ها ی عملیاتی و اطلاعاتی را تعیین می‌کند که به برنامه کاربردی ارزش تجاری می‌دهند.

۳- تکرار مدل‌های عملیاتی : مجموعه‌ای از نمونه‌های اولیه افزایشی که عملکرد را برای مشتری به نمایش می‌گذارد. هدف این چرخه تکرار، جمع‌آوری خواسته‌های اضافی با توجه به بازخورد از کاربران در حین تمرین روی نمونه اولیه است.

۴- تکرار طراحی و ساخت : بازبینی نمونه‌های اولیه ساخته‌شده طی مرحله‌ی تکرار مدل‌های عملیاتی برای حصول اطمینان از اینکه هر کدام به شیوه‌ای مهندسی شده‌است که بتواند برای کاربران نهایی ارزش تجاری داشته باشد.

۵- پیاده‌سازی : قرار دادن آخرین نسخه نرم‌افزاری در محیط کار.

کریستال Crystal

کریستال به خانواده ای از مدل های فرایند چابک گفته می شود که بر خصوصیات ویژه ی یک پروژه قابل انطباق است.

توسعه ویژگی – محور

Feature Driven Development (FDD)

قدری رسمی تر از سایر روش های چابک است، ولی همچنان با جلب توجه تیم پروژه به توسعه ی یک سری ویژگی ها چابکی را حفظ می کند.

فلسفه :

(1) emphasizes col- laboration among people on an FDD team;

(2) manages problem and project complexity using feature-based decomposition followed by the integration of software increments

(3) communication of technical detail using verbal, graphical, and text-based means.

ویژگی *feature*

ویژگی : یک عملکرد است که نزد متقاضی دارای ارزش بوده در کمتر از دو هفته قابل پیاده سازی است.

قالب تعریف ویژگی

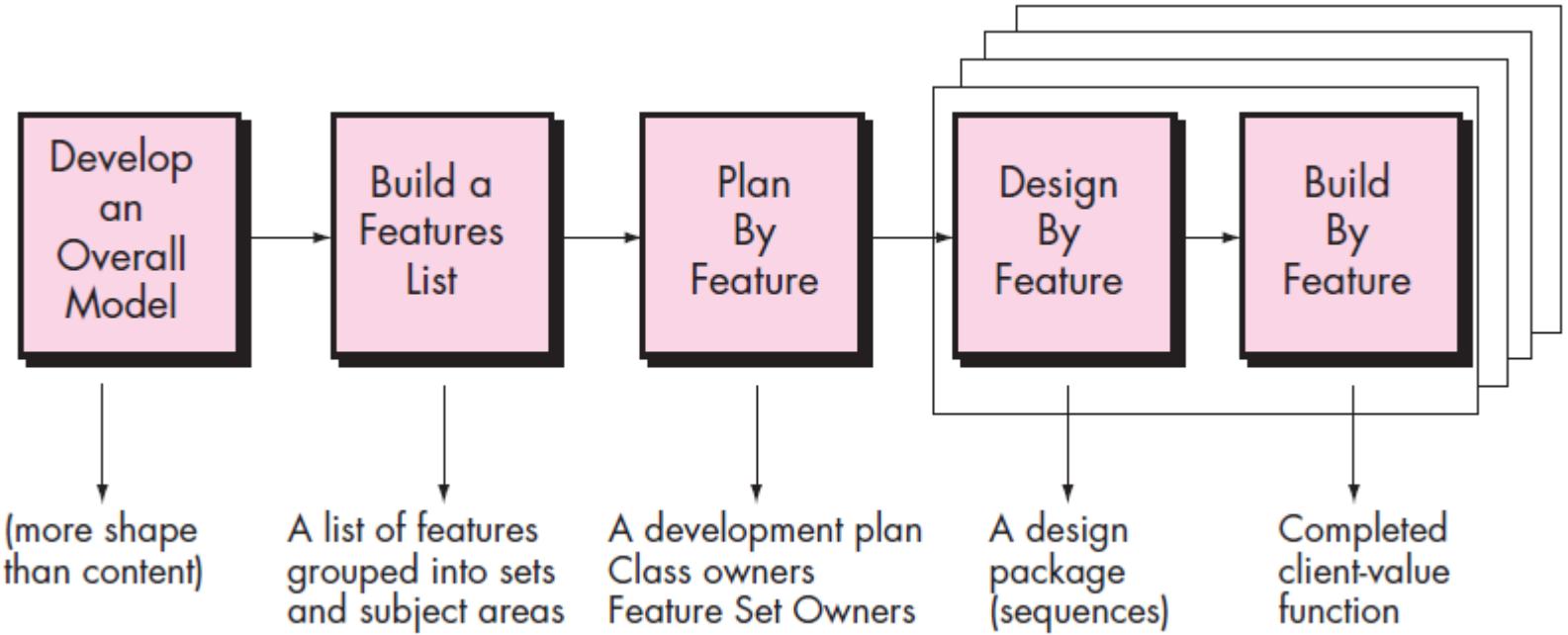
<**action**> the <**result**> <**by** | **for** | **of** | **to**> a(n) <**object**>

- *Add the product to shopping cart*
- *Display the technical-specifications of the product*
- *Store the shipping-information for the customer*

<**action**><**-ing**> a(n) <**object**>

Making a product sale

فعالیت های چارچوبی مبتنی بر همکاری در رویکرد FDD



توسعه نرم‌افزار ناب

Lean Software Development (LSD)

اصول تولید ناب را وارد دنیای مهندسی نرم افزار کرده است.

- *eliminate waste*
- *build quality in*
- *create knowledge*
- *defer commitment*
- *deliver fast*
- *respect people*
- *optimize the whole.*

تفسیر حذف ضایعات در حیطه یک پروژه نرم افزاری چابک

1- adding no extraneous features or functions

2- assessing the cost and schedule impact of any newly requested requirement

3- removing any superfluous process steps

4- establishing mechanisms to improve the way team members find information,

5- ensuring the testing finds as many errors as possible

6- reducing the time required to request and get a decision that affects the software or the process that is applied to create it

7- streamlining the manner in which information is transmitted to all stakeholders involved in the process.

مدل‌سازی چابک

Agile Modeling (AM)

پیشنهاد می کند که مدل سازی برای همه سیستم ها ضروروی است، ولی پیچیدگی ، نوع و اندازه مدل باید متناسب با

نرم افزاری که قرار است ساخته شود، تنظیم گردد.

(1) all constituencies can better understand what needs to be accomplished

(2) the problem can be partitioned effectively among the people who must solve it

(3) quality can be assessed as the system is being engineered and built.

اصولی که AM را از سایر روش ها متمایز می سازد

Model with a purpose.

Use multiple models.

Travel light.

Content is more important than representation.

Know the models and the tools you use to create them.

Adapt locally.

فرایند چابک یکپارچه

Agile Unified Process (AUP)

فلسفه ترتیب در مقیاس انبوه و تکرار در مقیاس کوچک را بر ساخت نرم افزار مطرح می سازد.

1- Modeling.

2- Implementation.

3- Testing.

4- Deployment

5- Configuration and project management.

6- Environment management.

در نظر گرفتن فرایند چابک

SafeHome

Considering Agile Software Development

The scene: Doug Miller's office.

The Players: Doug Miller, software engineering manager; Jamie Lazar, software team member; Vinod Raman, software team member.

پایان فصل ۳

مهندسی نرم افزار

فصل چهارم : اصول راهنما در مهندسی نرم افزار

مدرس:

فرشید شیرافکن

دانشجوی دکتری دانشگاه تهران

(کارشناسی و کارشناسی ارشد : کامپیوتر نرم افزار) (دکتری: بیو انفورماتیک)

نگاهی گذار

اصول راهنما چیستند؟

مهندسی نرم افزار،آرایه وسیعی از اصول، مفاهیم،روش ها و ابزارهاست که باید در برنامه ریزی برای توسعه یک نرم افزار در نظر گرفت.

چرا اهمیت دارد؟

فرایند نرم افزار برای رسیدن به هدفی موفق، یک نقشه راه در اختیار همه ی افراد دخیل در ایجاد یک سیستم یا محصول کامپیوتری قرار می دهد. کار مهندسی، جزئیات لازم برا ی طی این مسیر را برای شما فراهم می سازد. در حیطه ی مهندسی نرم افزار، کار مهندسی چیزی است که در طول روز انجام می دهید تا نرم افزار را از یک ایده به واقعیت برسانید.

کار مهندسی نرم افزار چیست؟

از یک دیدگاه کلی، کار مهندسی به مجموعه ای مفاهیم، اصول، روش ها و ابزارها گفته می شود که یک مهندس نرم افزار در طول روز با آنها سرو کار دارد.

کار مهندسی به مدیران این امکان را می دهد که پروژه ها ی نرم افزاری را مدیریت کنند وبه مهندسان نرم افزار این امکان را می دهد که برنامه های کامپیوتری بسازند.

دانش مهندسی نرم افزار

نظر مک کانل:

توده ی دانش مهندسی نرم افزار به یک هسته ی پایدار متکامل شده است که نشان گر ۷۵٪ از دانش مورد نیاز برای توسعه یک سیستم پیچیده است.

اصول هسته ای اکنون بستری فراهم می سازند که مدل های نرم افزاری، روش ها و ابزار ها را در آن بستر می توان به کاربرد و ارزیابی کرد.

اصول هسته ای

مجموعه ای از اصول هسته ای وجود دارد که راهنمای مهندسی نرم افزار است و به استفاده از یک فرایند نرم افزار با معنی و اجرای اثربخش روش های مهندسی نرم افزار کمک می کند.

در سطح فرایند، اصول فرایند، یک بنیاد فلسفی ایجاد می کنند که :

- تیم نرم افزاری را هدایت می کند،
- جریان فرایند را مورد کاوش قرار می دهد و
- مجموعه ای از محصولات کاری مهندسی نرم افزار تولید می کند.

در سطح کاری، اصول مهندسی نرم افزار، مجموعه ای از ارزش ها و قواعد را تعیین می کند که شما را در تحلیل یک مساله، طراحی راهکار، پیاده سازی و آزمون آن راهکار و سرانجام استقرار نرم افزار در جامعه ی کاربری راهنمایی می کند.

اصول راهنمای فرایند مهندسی

مجموعه اصول هسته ای زیر را می توان برای چار چوب و با بسط دادن آن برای هر فرایند نرم افزار به کار گرفت:

اصل ۱. چابک باشید.

اصل ۲. در هر مرحله کیفیت را در کانون توجه قرار دهید.

اصل ۳. آمادگی انطباق را داشته باشید.

اصل ۴. تیمی اثر بخش تشکیل دهید.

اصل ۵. سازو کارهایی برای برقراری ارتباط و هماهنگی ایجاد کنید.

اصل ۶. مدیریت تغییرات.

اصل ۷. ارزیابی ریسک.

اصل ۸. ایجاد محصولات کاری که برای دیگران ارزش فراهم می کنند.

اصول راهنمای کار مهندسی نرم افزار

اصل ۱. تقسیم و حل

اصل ۲. درک به کارگیری انتزاع

اصل ۳. تلاش برای سازگاری

اصل ۴. توجه ویژه به انتقال اطلاعات

اصل ۵. توسعه نرم افزاری که ساختار پیمانه ای اثر بخش داشته باشد.

اصل ۶. جستجو به دنبال الگوها

اصل ۷. هرگاه که امکان دارد، مساله و راهکار آن را از چند دیدگاه متفاوت به نمایش بگذارید.

اصل ۸. به خاطر داشته باشید که نرم افزار را نگهداری خواهید کرد.

اصول راهنمای فعالیت های چارچوبی

۱- اصول ارتباطی

۲- اصول برنامه ریزی

۳- اصول مدل سازی

۴- اصول ساخت

۵- اصول استقرار

اصول ارتباطی

اصول ارتباطی : بسیاری از این اصول در سایر شکلهای ارتباطی که در یک پروژه ی نرم افزاری رخ می دهند کاربرد دارند:

اصل ۱. گوش سپردن.

اصل ۲. خود را قبل از برقراری ارتباط آماده کنید.

اصل ۳. یک نفر باید این فعالیت را تسهیل کند.

اصل ۴. بهترین راه، ارتباط رودرروی است.

اصل ۵. یادداشت بردارید و تصمیم گیری را مستند کنید.

اصل ۶. تلاش برای همکاری.

اصل ۷. توجه خود را معطوف کنید، بحث خود را پیمانه ای کنید.

اصل ۸. اگر چیزی واضح نبود، یک تصویر بکشید.

اصل ۹. الف) هنگامی که بر سر بحثی به توافق رسیدید، به مبحث دیگر بپردازید.

ب) اگر به توافق نرسیدید، به مبحث دیگر بپردازید.

پ) اگر ویژگی یا قابلیتی واضح نیست و نمی توان در حال حاضر آن را واضح کرد ،باز هم به مبحث دیگر بپردازید.

اصل ۱۰. مذاکره، یک مسابقه یا بازی نیست. وقتی بهترین نتیجه را می دهد که هر دو طرف برنده باشند.

اصول برنامه ریزی

فعالیت برنامه ریزی شامل مجموعه ای از امور مدیریتی و فنی می شود که تیم نرم افزاری را قادر به تعریف نقشه راه در سفر به سوی اهداف راهبردی و مقاصد تاکتیکی اش می سازد.

فلسفه های متفاوت برای برنامه ریزی:

- کمینه گرا : تغییر، غالباً نیاز به برنامه ریزی مفصل را منتفی می سازد.
- سنت گرا : برنامه ریزی، یک نقشه راه اثربخش فراهم می آورد و هرچه جزئیات آن بیشتر باشد، احتمال گم شدن تیم کمتر می شود.
- چابک گرایان : یک "بازی برنامه ریزی" سریع ممکن است ضروری باشد، ولی نقشه راه چیزی است که با "کار واقعی" روی نرم افزار شروع می شود.

برنامه ریزی با هر میزان سخت گیری که اجرا شود، اصولی که به دنبال خواهد آمد، همواره کاربرد خواهد داشت:

اصل ۱. شناخت حوزه ی پروژه.

اصل ۲. طرف های ذی نفع را در فعالیت برنامه ریزی دخالت دهید.

اصل ۳. این را بدانید که برنامه ریزی ماهیتی مبتنی بر تکرار دارد.

اصل ۴. برآوردهای خود را براساس آنچه که می دانید، انجام دهید.

اصل ۵. همزمان با برنامه ریزی، ریسک را هم در نظر بگیرید.

اصل ۶. واقع بین باشید. مردم هر روز صددرصد کار نمی کنند.

اصل ۷. هنگام تعریف برنامه ریزی، گرانولیته(granulity) را تعیین کنید.

اصل ۸. تعیین کنید که چگونه می خواهید از کیفیت اطمینان یابید.

اصل ۹. چگونگی انجام دادن تغییرات را شرح دهید.

اصل ۱۰. برنامه ریزی را به وفور پیگیری کنید و در صورت نیاز، تنظیماتی به عمل آورید.

اصول مدل سازی

ما مدل ها را برای درک بهتر یک موجودیت واقعی که قرار است ساخته شود، ایجاد می کنیم.

مدل ها با توجه به نوع موجودیت متفاوت اند.

در کار مهندسی نرم افزار، دو نوع مدل ایجاد می شود:

۱- مدل های خواسته ها (مدل تحلیل)
۲- مدل های طراحی

مدل خواسته ها، خواسته های مشتری را با تصویر کردن نرم افزار در سه دامنه متفاوت به نمایش می گذارند.

۱- دامنه اطلاعاتی
۲ – دامنه عملیاتی
۳- دامنه رفتاری

مدل های طراحی: نشان گر خصوصیاتی از نرم افزارند که به نرم افزار نویس کمک می کنند تا آن را بهتر بسازد:

۱- معماری
۲- واسط کاربر
۳ - جزئیات در سطح مولفه ها

اصول مدل سازی

اصل۱ – هدف اصلی تیم نرم افزاری ساخت نرم افزار است نه ایجاد مدل .

اصل۲– سبک بار سفر کنید. مدل های بیش از نیاز خود ایجاد نکنید.

اصل۳– بکوشید ساده ترین مدلی را بسازید که مساله یا نرم افزار را توصیف کند.

اصل۴– مدل ها را طوری بسازید که قابل تغییر باشد.

اصل۵– توانایی بیان صریح هدف هر مدل ایجاد شده را داشته باشید.

اصل۶- مدل هایی را که توسعه می دهید بر سیستم مورد نظر مطابقت دهید.

اصل۷ - سعی کنید مدل های مفید بسازید، ولی ساخت مدل های کامل را فراموش کنید .

اصل۸ - در مورد قالب و نحو مدل، تعصب به خرج ندهید، اگر در انتقال مفاهیم موفق است ،نمایش در مرحله ی دوم اهمیت قرار دارد.

اصل۹- اگر غریزه شما می گوید مدلی درست نیست، هر چند که روی کاغذ درست به نظر می رسد، احتمالا دلیلی برای این نگرانی دارید.

اصل۱۰- به محض اینکه توانستید ، بازخورد بگیرید.

در مدل سازی خواسته ها(تحلیل) ، ویژگی های نرم افزاری زیر مورد توجه قرار می گیرند :

۱- اطلاعاتی که باید پردازش شوند.

۲- قابلیت هایی که باید تحویل شوند.

۳- رفتاری که باید به نمایش گذارده شود.

اصول مدل سازی خواسته ها

اصل۱ – دامنه ی اطلاعاتی یک مساله باید نمایش داده و درک شود.

اصل۲– عملکردهای نرم افزار باید تعریف شوند.

اصل۳– رفتار نرم افزار(بعنوان نتیجه ای از رویداد های خارجی) باید نمایش داده شوند.

اصل۴– مدل هایی که اطلاعات، قابلیت ها و رفتارها را تصویر می کنند باید به شیوه ای تقسیم بندی شوند که جزئیات را به گونه ای لایه ای(یا سلسله مراتبی) نمایش دهند.

اصل۵– وظیفه ی تحلیل باید از اطلاعات ضروری به سمت جزئیات پیاده سازی حرکت کند.

اصول مدل سازی طراحی

اصل۱ – طراحی باید تا مدل خواسته ها قابل ردگیری باشد.

اصل۲– همواره معماری سیستمی را که قرار است ساخته شود، در نظر داشته باشید.

اصل۳– طراحی داده ها به اندازه طراحی عملکردها اهمیت دارد .

اصل۴– واسط ها(چه درونی و چه بیرونی) باید با احتیاط طراحی شوند.

اصل۵– طراحی واسط کاربر باید مطابق با نیاز های کاربر نهایی تنظیم گرد، ولی در هر مورد باید بر سهولت کاربرد نیز تاکید ورزیده شود.

اصل۶- طراحی در سطح مولفه ها باید مستقل از عملکرد باشد.

اصل۷ - مولفه ها باید با یکدیگر و با محیط خارجی ارتباطی سست داشته باشند .

اصل۸ - مدل های طراحی باید به آسانی قابل درک باشند.

اصل۹- طراحی باید به صورت تکراری توسعه یابد. در هر دور تکرار، طراحی باید بکوشد تا سادگی بیشتر شود.

اصول ساخت

فعالیت ساخت شامل مجموعه ای از وظایف کدنویسی و آزمایش است که نتیجه ی آن نرم افزاری عملیاتی و آماده ی تحویل به کاربر نهایی است.

کدنویسی می تواند یکی از موارد زیر باشد

۱- ایجاد مستقیم کد منبع در زبان برنامه نویسی.

۲- تولید خودکار کد منبع با استفاده از نمایش شبه طراحی از مؤلفه ای باشد که قرار است ساخته باشد

۳- تولید خودکار کد قابل اجرا با استفاده از یک زبان برنامه نویسی نسل چهارم

در مرحله ی آزمون،توجه به سیستم ابتدا در سطح مؤلفه ها رخ می دهد،این رویکرد را آزمون واحدها می نامند.

سایر سطوح آزمون عبارتند از:

۱- آزمون انسجام

۲- آزمون اعتبارسنجی

۳- آزمون پذیرش

اصول کد نویسی

اصولی راهگشا در وظیفه کدنویسی ، بستگی تنگاتنگی با موارد زیر دارند:

- سبک برنامه نویسی
- زبان برنامه نویسی
- شیوه برنامه نویسی موردنظر

اصول آماده سازی

قبل از نوشتن حتی یک خط از برنامه اطمینان حاصل کنید که :

- می دانید چه مسأله ای را قرار است حل کنید.
- اصول و مفاهیم طراحی پایه را می دانید.
- زبانی برای برنامه نویسی انتخاب کنید که نیازهای نرم افزاری که قرار است در آن کارکند، برآورده سازد.
- محیطی برای برنامه نویسی انتخاب کنید که ابزارهای لازم برای آسان تر کردن کار را در اختیارقراردهد.
- مجموعه ای از آزمون های پایه را ایجاد کنید که با کامل شدن کدنویسی مؤلفه، بتوانید آنها را به کار ببرید.

اصول برنامه نویسی

با شروع کدنوسی اطمینان حاصل کنید که:

۱- الگوریتم هایتان را با دنباله روی از برنامه سازی ساخت یافته، مقید کنید.

۲- استفاده از برنامه نویسی جفتی را در نظرداشته باشید.

۳- انتخاب ساختمان داده هایی که نیازهای طراحی را برآورده کند.

۴- شناختن معماری نرم افزار و ساختن واسط هایی سازگار با آن

۵- ساده نگه داشتن منطق شرطی تا حد امکان

۶- نوشتن حلقه های تودرتو به شیوه ای که به آسانی قابل آزمون باشند.

۷- انتخاب نام های با معنی برای متغیرها و پیروی از سایر استانداردهای کدنویسی محلی

۸- نوشتن کدهایی که خود مستندسازی شده باشند.

۹- ایجاد یک چیدمان بصری (مثلا با تورفتگی و خطوط خالی) که به فهم کدها کمک کند.

اصول اعتبارسنجی

پس از به پایان رساندن اولین دور کدنویسی حتما:

- در صورت امکان گشتی در میان کدها بزنید.
- آزمون واحدها را اجرا کنید و خطاهایی را که کشف می شوند را تصحیح کنید.
- کدها را بازآرایی کنید.

اصول آزمون

چند قاعده که می توان آن ها را به عنوان اهداف آزمون در نظرگرفت:

- آزمون، فرآیند اجرای برنامه به قصد یافتن خطاهاست.
- یک مورد آزمون خوب باید خطاهای کشف نشده را با احتمال زیادی کشف کند.
- آزمون موفق،آزمونی است که خطای کشف نشده تاکنون را کشف کند .

آزمون نمی تواند نبودن خطاها و نقایص را نشان بدهد، فقط می تواند نشان دهد که خطاها و نقایص وجود دارند.

اصول آزمون

اصل۱: همه آزمون ها تا خواسته های مشتری قابل ردگیری باشند.

اصل۲ : آزمون ها را باید مدت ها قبل از شروع آزمون برنامه ریزی کرد.

اصل۳: اصل پارتو در آزمون نرم افزار کاربرد دارد.

اصل پارتو : اثر ۸۰٪ از همه خطاهای کشف شده طی آزمون را احتمالادر ۲۰٪ ازکل مؤلفه های نرم افزار می توان پیدا کرد.

اصل۴: آزمون باید در مقیاس کوچک آغاز شود و به سمت مقیاس بزرگ پیش رود.

اصل۵: آزمون کامل امکان پذیر نیست.

اصول استقرار

فعالیت استقرار شامل سه کنش است: تحویل، پشتیبانی و بازخورد.

چون مدل های فرآیند نرم افزار مدرن، ماهیتی تکاملی یا افزایشی دارند، به یکباره استقرار رخ نمی دهد بلکه با حرکت نرم افزار به سوی تکامل، چند با تکرار می شود.

چرخه تحویل: یک نسخه عملیاتی از نرم افزار در اختیار کاربر نهایی قرار می دهد که قابلیت ها و ویژگی های جدیدی فراهم می سازد.

چرخه پشتیبانی: کمک انسانی و مستند سازی برای کلیه قابلیت ها و ویژگی های ارائه شده طی همه چرخه های استقرار تا آن زمان را فراهم می سازد.

چرخه بازخورد: راهنمایی های مهمی را در اختیار تیم نرم افزاری قرار می دهد که به اصلاح قابلیت ها،ویژگی ها و رویکرد در نظرگرفته شده برای نسخه بعدی نرم افزار می انجامند.

در حالی که تیم نرم افزاری آماده تحویل یک نسخه جدید می شود چند اصل کلیدی را باید رعایت کند:

اصل۱: انتظارات مشتری برای نرم افزار باید مدیریت شود .

اصل۲: پکیج تحویل کامل باید مونتاژ و آزمایش شود.

اصل۳: قبل از تحویل نرم افزار، یک روال پشتیبانی باید مشخص کرد.

اصل۴: مواد آموزشی مناسب باید برای کاربران نهایی تهیه شود.

اصل۵: نرم افزار مشکل دار ابتدا باید اصلاح و بعدا تحویل داده شود.

اشتباه در برقراری ارتباط

SafeHome

Communication Mistakes

The scene: Software engineering team workspace

The players: Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member.

پایان فصل ۴

مهندسی نرم افزار

فصل پنجم : شناخت خواسته ها

مدرس:

فرشید شیرافکن

دانشجوی دکتری دانشگاه تهران

(کارشناسی و کارشناسی ارشد : کامپیوتر نرم افزار) (دکتری: بیو انفورماتیک)

خواسته ها
REQUIREMENTS

خواسته ها چیستند؟

چرا اهمیت دارد؟

محصول کار چیست؟

چگونه اطمینان حاصل کنم که درست از عهده کار بر آمده ام؟

مراحل کار در مهندسی خواسته ها

۱- شروع Inception

۲- استخراجElicitation

چند نمونه از مشکلات در مرحله ی استخراج :

۱- مشکلات مربوط به حوزه ی پروژه

۲- مشکلات مربوط به درک پروژه

۳- مشکلات مربوط به تغییر پذیری

۳- شناخت Elaboration

۴- مذاکره .Negotiation

۵- تعیین مشخصات Specification

۶- اعتبار سنجی Validation

۷- مدیریت خواسته ها Requirements management

مراحل مورد نیاز برای تدارک مقدمات شناخت خواسته های نرم افزار

۱- شناسایی طرف های ذی نفع Identifying Stakeholders

ذی نفع: هر کسی که به طور مستقیم یا غیر مستقیم از سیستم در حال توسعه بهره مند خواهد شد .

مانند مدیران تولید، بازاریاب ها ، مشتریان، کاربران نهایی، مشاوران ، مهندسان تولید، مهندسان نرم افزار و...

۲- شناخت دیدگاه های چند گانه Recognizing Multiple Viewpoints

از آنجایی که طرف های ذی نفع فراوانی وجود دارد ، خواسته های سیستم از دیدگاه های متفاوت بسیاری بررسی می شود.

باید همه ی اطلاعات بدست آمده از طرف های ذی نفع (حتی ناسازگار و متناقض) را به شیوه ای گروه بندی کرد که به تصمیم گیران این امکان را بدهد که مجموعه از خواسته ها را انتخاب کند که از سازگاری درونی برخوردار باشد.

۳- تلاش برای همکاری Working toward Collaboration

وظیفه ی مهندسی خواسته ها، شناسایی وجوه اشتراک و موارد متضاد یا ناسازگار است.

بدیهی است که گروه دوم خواسته ها است که ایجاد چالش می کند.

۴- پرسیدن نخستین سوالات Asking the First Questions

استخراج خواسته ها
ELICITING REQUIREMENTS

۱- همکاری در جمع آوری خواسته ها

۲- استقرار عملکرد کیفیت

۳- سناریوهای کاربرد

۴- محصولات کاری استخراج

همکاری در جمع آوری خواسته ها

Collaborative Requirements Gathering

- Meetings are conducted and attended by both software engineers and other stakeholders.
- Rules for preparation and participation are established.
- A “facilitator” controls the meeting.
- A “definition mechanism” is used.

استقرار عملکرد کیفیت

Quality Function Deployment(QFD)

تکنیک تضمین کیفیتی است که نیاز های مشتری را به خواسته های فنی برای نرم افزار ترجمه می کند.

QFD سه نوع خواسته را مشخص می کند :

۱- خواسته های عادی (Normal)

۲- خواسته های مورد انتظار (Expected)

۳- خواسته های مهیج (Exciting)

سناریوهای کاربرد

Usage Scenarios

به موازات جمع آوری خواسته ها چشم انداز کلی از عملکرد ها و ویژگی های سیستم شکل می گیرد ولی تا زمانی که چگونگی به کارگیری این قابلیت ها و ویژگی ها توسط دسته های متفاوت کاربران نهایی را درک نکرده باشید، حرکت بسوی فعالیت فنی تر مهندسی نرم افزار دشوار خواهد بود.

به این منظور سازندگان می توانند از یک مجموعه سناریو(use case) استفاده کنند که توصیفی از چگونگی بکارگیری سیستم است.

محصولات کاری استخراج

Elicitation Work Products

محصولات کاری تولید شده به عنوان نتیجه ای از استخراج خواسته ها بسته به اندازه ی سیستم یا محصولی که قرار است ساخته شود، متغیر است.

برای اکثر سیستم ها ، محصولات کاری عبارتند از:

- بیان نیاز ها و امکان سنجی.
- بیان محدود حوزه ی مربوط به سیستم یا محصول.
- فهرستی از مشتریان ، کاربران و سایر طرف های ذی نفع که در استخراج خواسته ها مشارکت داشته اند.
- توصیفی از محیط فنی سیستم.
- فهرستی از خواسته ها و قیدوبندهای دامنه که در مورد هر کدام کاربرد دارند.
- مجموعه ای از سناریو های کاربرد که دیدی از کاربرد سیستم یا محصول، تحت شرایط عملیاتی متفاوت به دست می دهند.
- هر نمونه اولیه ای که برای تعریف بهتر خواسته ها توسعه یافته باشد.

توسعه ی use case

DEVELOPING USE CASES

use case :

داستانی است با سبک و سیاق

درباره چگونگی تعامل کاربر نهایی (که یکی از چند نقش ممکن را بر عهده دارد) باسیستم

تحت مجموعه شرایط معین.

گام اول در نوشتن یک use case

تعریف مجموعه ای از کنش گران (actors) است که در داستان مشارکت دارند.

کنش گر : هرچیزی که با سیستم یا محصول ارتباط برقرار می کند و خارج از خود سیستم قرار دارد.

هر کنش گر هنگام استفاده از سیستم یک یا چند هدف دارد.

کنش گر و کاربر نهایی الزاما یکسان نیستند.

شناسایی کنش گرها

از آنجایی که استخراج خواسته ها یک فعالیت تکاملی است، همه کنش گر ها در اولین دور تکرار شناسایی نمی شوند.

کنش گرهای نوع اول :

شناسایی کنش گرهای نوع اول طی نخستین دور تکرار امکان پذیر است، کنش گرهای نوع اول با هم تعامل می کنند تا عملکرد لازم برای سیستم حاصل شود و مزایای مورد نظر از آن بدست آید.

کنش گرهای نوع دوم :

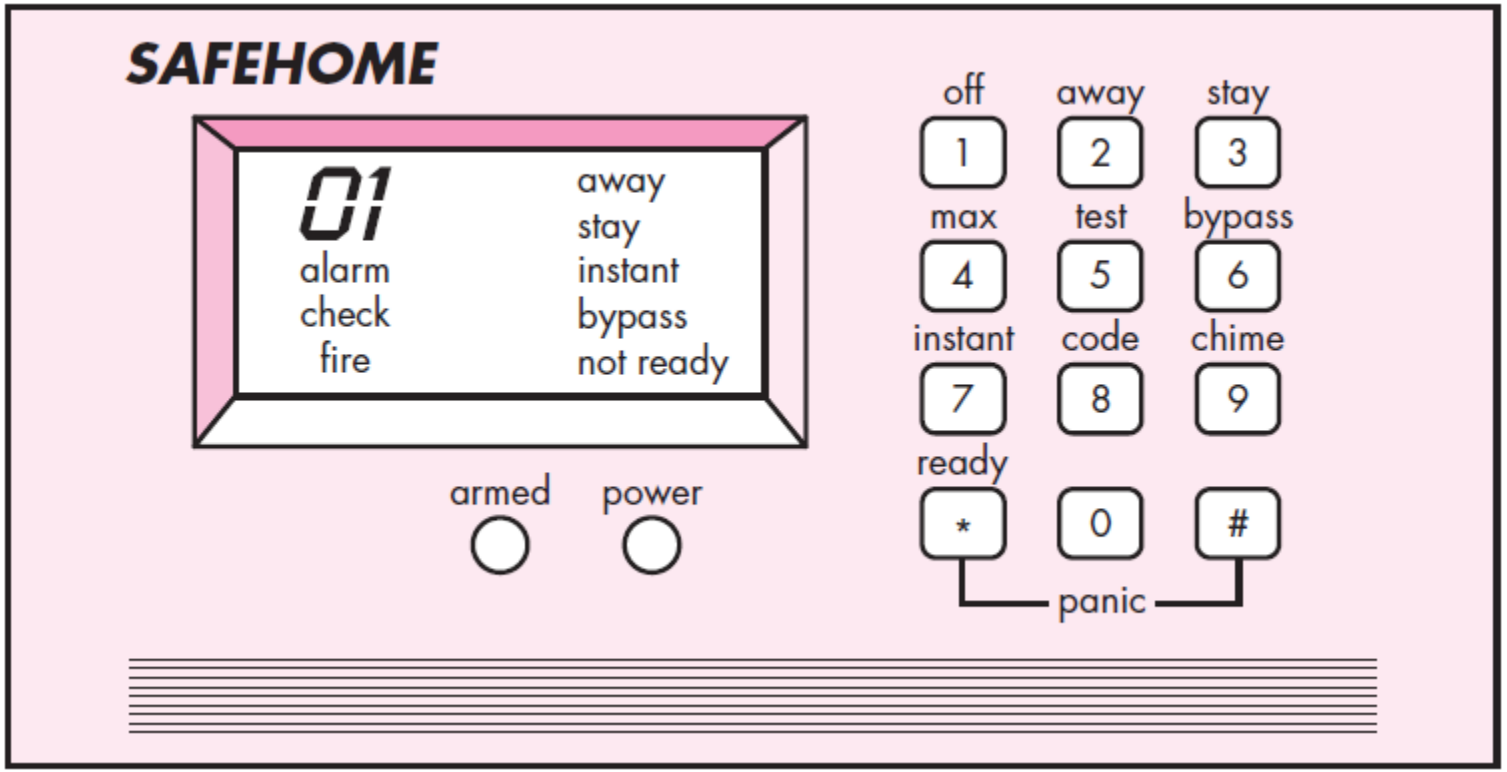
با کسب اطلاعات بیشتر درباره سیستم می توان آنها را شناسایی کرد.

کنش گرهای نوع دوم سیستم را شناسایی می کنند به طوری که کنش گرهای نوع اول بتوانند به کار خود ادامه دهند.

پرسش هایی که در یک use case به آنها پاسخ گفته می شود.

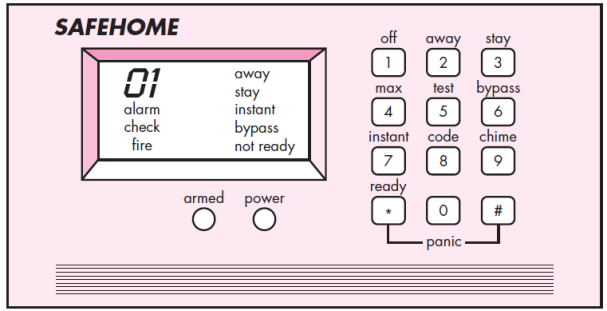
- Who is the primary actor, the secondary actor(s)?
- What are the actor's goals?
- What preconditions should exist before the story begins?
- What main tasks or functions are performed by the actor?
- What exceptions might be considered as the story is described?
- What variations in the actor's interaction are possible?
- What system information will the actor acquire, produce, or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

پانل کنترل SafeHome



Use case: *InitiateMonitoring*

Primary actor: Homeowner.



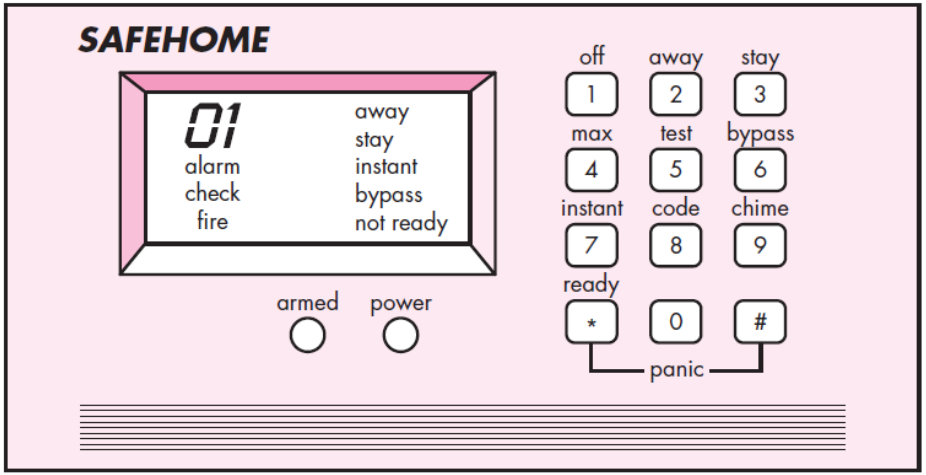
Goal in context: To set the system to monitor sensors when the homeowner leaves the house or remains inside.

Preconditions: System has been programmed for a password and to recognize various sensors.

Trigger : The homeowner decides to “set” the system, i.e., to turn on the alarm functions.

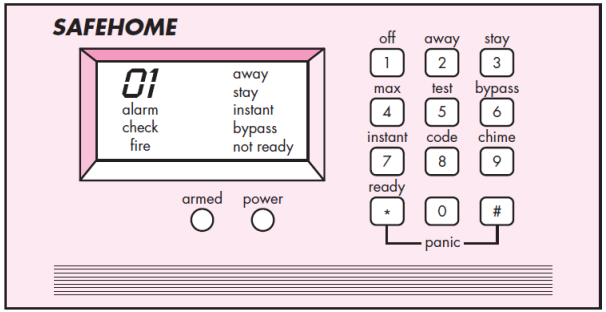
سناریو

1. Homeowner: observes control panel
2. Homeowner: enters password
3. Homeowner: selects “stay” or “away”
4. Homeowner: observes read alarm light to indicate that *SafeHome* has been armed



Exceptions: استثناها

1. Control panel is *not ready:* homeowner checks all sensors to determine which are open; closes them.
2. Password is incorrect (control panel beeps once): homeowner reenters correct password.
3. Password not recognized: monitoring and response subsystem must be contacted to reprogram password.
4. *Stay* is selected: control panel beeps twice and a *stay* light is lit; perimeter sensors are activated.
5. *Away* is selected: control panel beeps three times and an *away* light is lit; all sensors are activated.



Priority: Essential, must be implemented

When available: First increment

Frequency of use: Many times per day

Channel to actor: Via control panel interface

Secondary actors: Support technician, sensors

Channels to secondary actors:

Support technician: phone line

Sensors: hardwired and radio frequency interfaces

Open issues: مشکلات باز

1. Should there be a way to activate the system without the use of a password or with an abbreviated password?

2. Should the control panel display additional text messages?

3. How much time does the homeowner have to enter the password from the time the first key is pressed?

4. Is there a way to deactivate the system before it actually activates?

ساخت مدل های خواسته ها

BUILDING THE REQUIREMENTS MODEL

هدف از این مدل تحلیل ، فراهم کردن توصیفی از دامنه های اطلاعاتی ، عملیاتی و رفتاری مورد نیاز برای سیستم کامپیوتری است.

در واقع مدل ها مانند عکس های فوری از خواسته ها هستند که در جریان تغییرات خواسته ها ، مدل ها نیز تغییر می کنند.

عناصر مدل خواسته ها

Elements of the Requirements Model

راه های توجه به یک سیستم کامپیوتری:

استفاده از یک حالت نمایشی مانند use case و بکارگیری آن در طرد همه مدل های دیگر است.

استفاده از چند حالت نمایشی متفاوت برای به تصویر کشیدن مدل خواست ها ، ارزشمند است.

حالتهای متفاوتی از نمایش شما را وادار می دارد که مدل خواسته ها را ، از دیدگاه های متفاوت در نظر بگیرد.

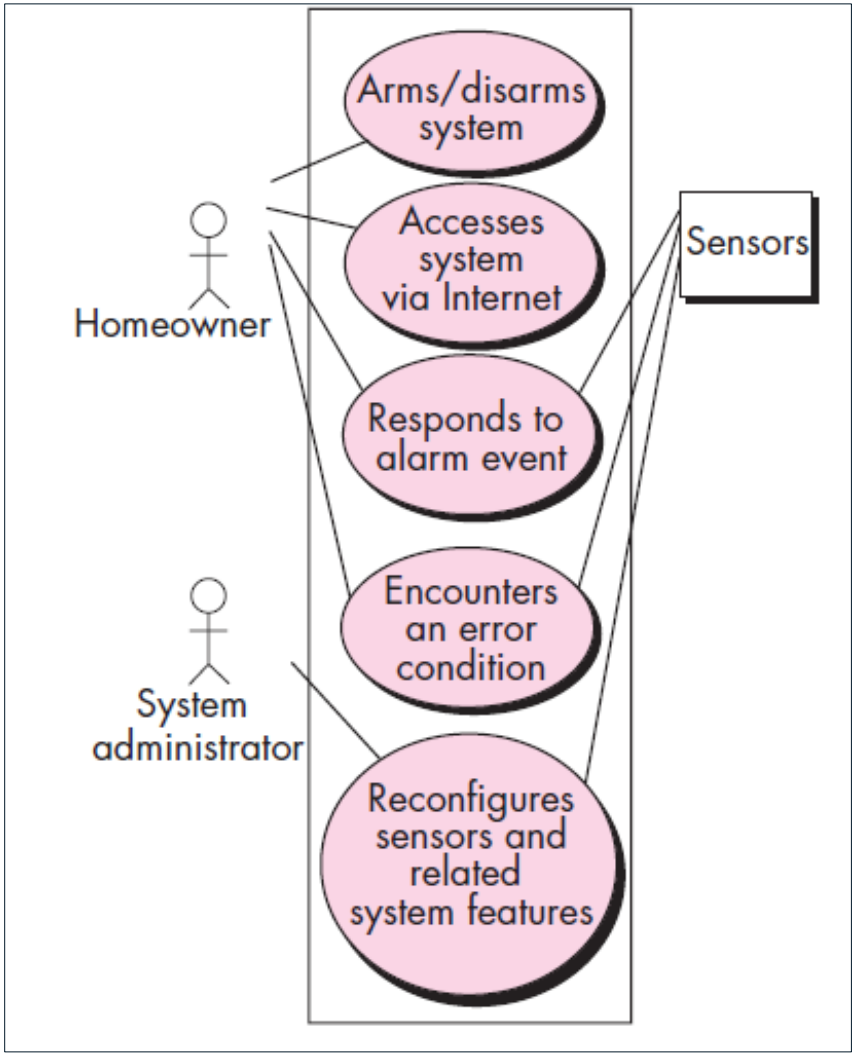
در این رویکرد ، احتمال آشکار شدن موارد جا افتاده ، ناسازگاری ها و ابهامات بیشتر می شود.

۱- عناصر مبتنی بر سناریو **Scenario-based elements**

۲- عناصر مبتنی بر کلاس **Class-based elements**

۳- عناصر رفتاری **Behavioral elements**

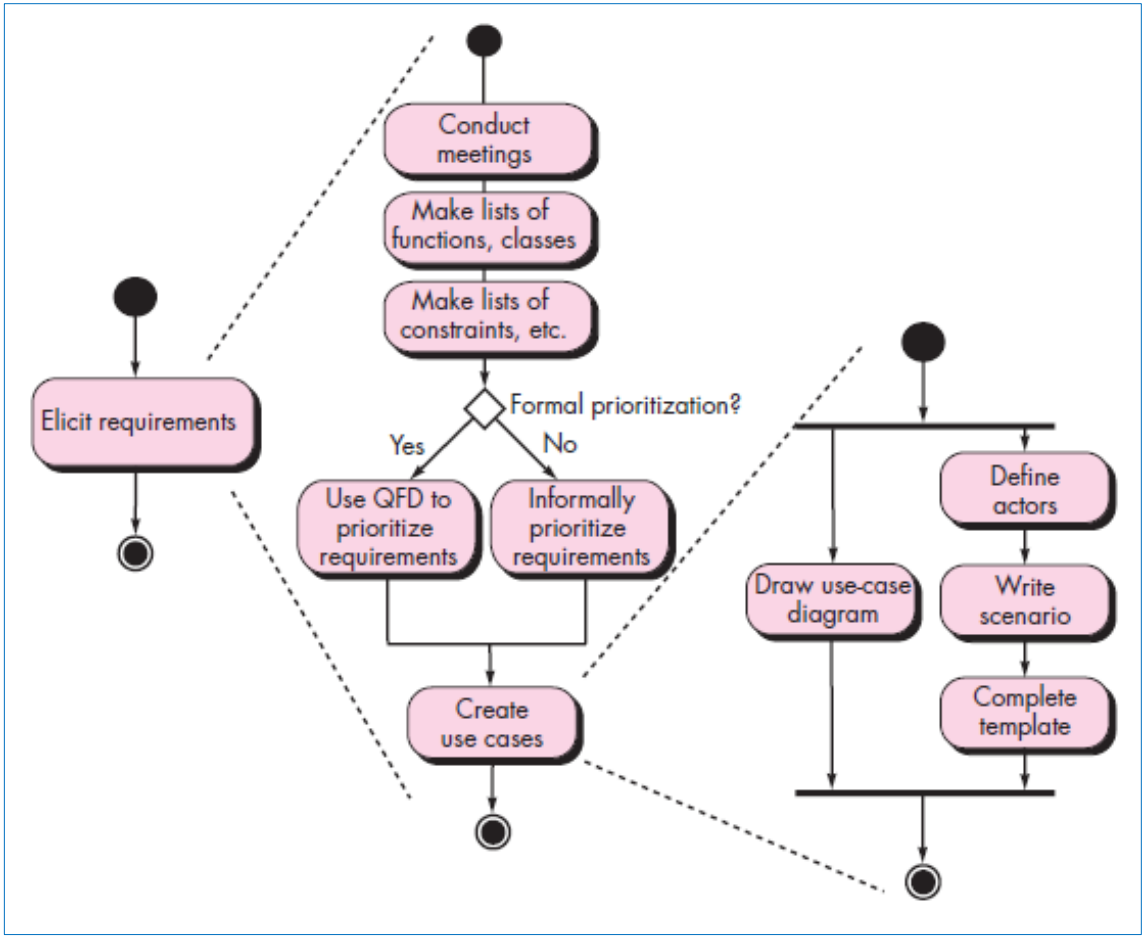
۴- عناصر جریان گرا **Flow-oriented elements**



عناصر مبتنی بر سناریو

عناصر مبتنی بر سناریو در مدل خواسته ها غالبا نخستین بخش از مدلی هستند که توسعه می یابد.

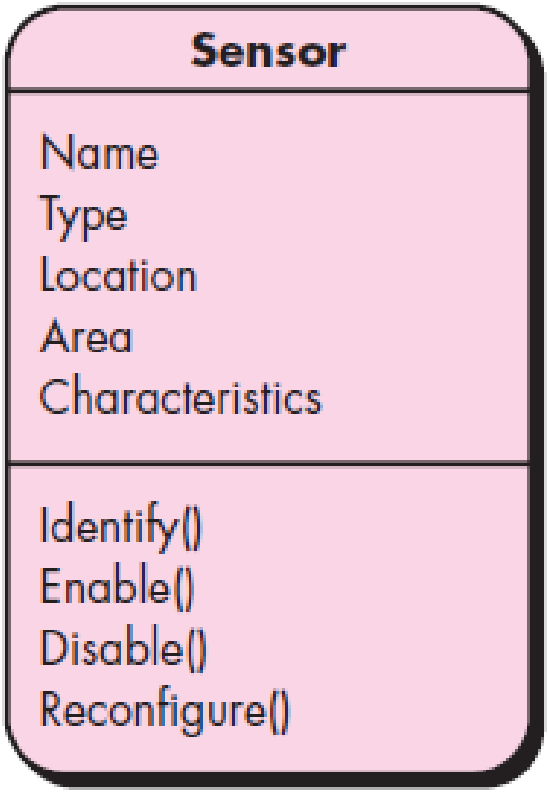
نمودار های فعالیت UML برای استخراج خواسته ها



در شکل مقابل یک نمودار UML از فعالیت ها برای استخراج خواسته ها و نمایش آنها با استفاده use case از نشان داده شده است.

سه سطح از شناخت مشاهده می شود که در یک نمایش مبتنی بر سناریو به اوج خود می رسد.

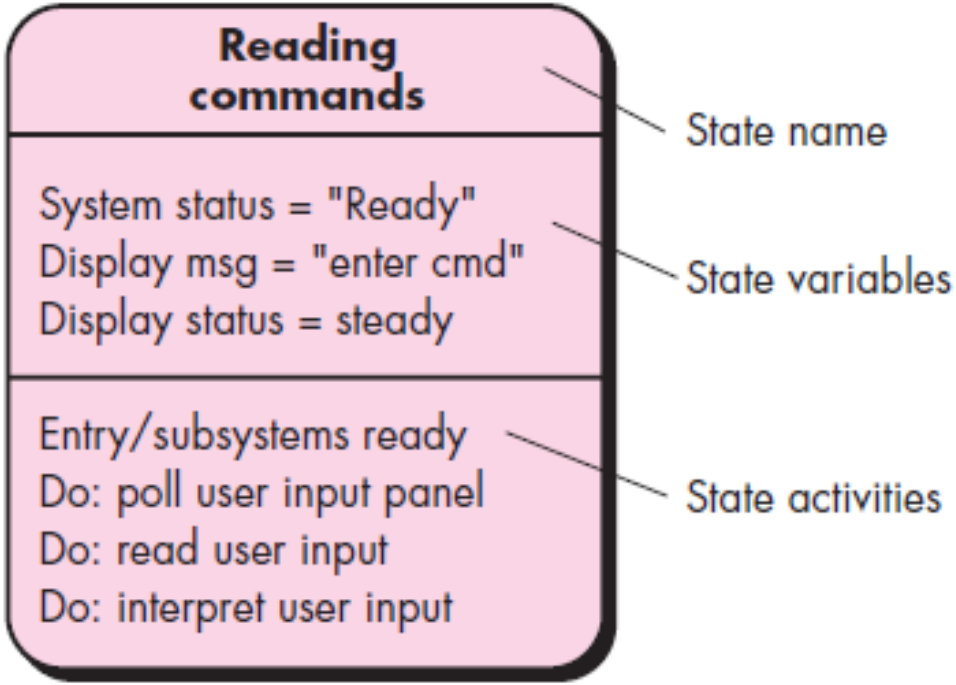
عناصر مبتنی بر کلاس



هر سناریوی کاربرد ، نشانگر مجموعه ای از اشیاء است که با تعامل یک کنش گر با سیستم ، دستکاری می شوند.

این اشیاء در قالب چند کلاس طبقه بندی می شوند.

عناصر رفتاری



رفتار یک سیستم کامپیوتری اثری عمیق بر انتخاب طراحی و و رویکرد مورد استفاده در پیاده سازی داشته باشد.

بنابراین مدل خواسته ها باید عناصر مدل سازی لازم را برای تصویر کردن این رفتار فراهم سازد.

عناصر جریان گرا

انتقال اطلاعات با جریان یافتن در یک سیستم کامپیوتری صورت می گیرد.

الگو های تحلیل

Analysis Patterns

الگوهای تحلیل با ارجاع به نام الگو در مدل تحلیل قرار داده می شوند.

این الگو ها در یک مخزن نگه داری می شوند تا مهندسان خواسته ها بتوانند با استفاده از تسهیلات جستجو گر آنها را پیدا کرده و مورد استفاده قرار دهند.

مذاکره بر سر خواسته ها

NEGOTIATING REQUIREMENTS

به جای یک فعالیت منفرد برقراری ارتباط با مشتری ، فعالیت های زیر تعیین می شود:

۱- شناسایی طرف های ذی نفع مهم سیستم یا زیر سیستم.
۲- تعیین شرایط برد برای طرف های ذی نفع
۳- مذاکره بر سر شرایط برد طرف های ذی نفع برای رساندن آنها به شرایط برد ــ برد ،برای تمامی طرف ها (از جمله تیم نرم افزار.)

اعتبار سنجی خواسته ها

VALIDATING REQUIREMENTS

با ایجاد هر کدام از عناصر مدل خواسته ها باید آن را از نظر ناسازگاری ، ابهام و موارد جا افتاده بررسی کرد.

خواسته هایی که مدل به نمایش می گذارد به موازات رشد و توسعه نرم افزار ، توسط طرف های ذی نفع اولویت بندی و در داخل بسته هایی گروه بندی می شود.

برگزاری جلسه جمع آوری خواسته ها

SafeHome

Conducting a Requirements Gathering Meeting

The scene: A meeting room. The first requirements gathering meeting is in progress.

The players: Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

برگزاری جلسه بعد از نخستین جلسه جمع آوری خواسته ها

SafeHome

The Start of a Negotiation

The scene: Lisa Perez's office, after the first requirements gathering meeting.

The players: Doug Miller, software engineering manager and Lisa Perez, marketing manager.

مدیر مهندسی نرم افزار

مدیر بازاریابی

پایان فصل ۵

مهندسی نرم افزار

فصل ۶: مدل سازی خواسته ها:

سناریوها، اطلاعات و کلاس های تحلیل

مدرس:

فرشید شیرافکن

دانشجوی **دکتری دانشگاه تهران**

(کارشناسی و کارشناسی ارشد : کامپیوتر نرم افزار) (دکتری: بیو انفورماتیک)

QUICK LOOK

نگاهی گذرا

مدل سازی خواسته ها چیست؟

سخنان مکتوب، وسیله ای عالی برای برقراری ارتباط به شمار می روند، ولی ضرورتا بهترین راه برای نمایش خواسته های مربوط به یک نرم افزار کامپیوتری نیستند.

در مدل سازی خواسته ها تلفیقی از شکل های متنی و نموداری برای به تصویر کشیدن خواسته ها استفاده می شود، به شیوه ای که درک آن آسان تر باشد و مهمتر از آن، به سهولت بتوان آن را برای تصحیح ، تکمیل و سازگاری مورد بازبینی قرار داد.

چه کسی آن را انجام می دهد؟

این مدل را یک مهندس نرم افزار (تحلیل گر) با به کار گیری خواسته های استخراج شده از مشتری می سازد.

چرا اهمیت دارد؟

برای اعتبار سنجی خواسته های نرم افزار، باید آنها را از چند دیدگاه متفاوت بررسی کنیم.

- مدل های مبتنی بر سناریو
- مدل های داده ای
- مدل های مبتنی بر کلاس

در هر کدام از این مدل ها خواسته ها از بعدی متفاوت به نمایش درمی آید و از این رو ، احتمال برملا شدن خطاها ، رو شدن ناسازگاری ها و کشف جا افتادگی ها افزایش می یابد.

مراحل کار کدام است؟

مدل های مبتنی بر سناریو، سیستم را از دیدگاه کاربر به نمایش می گذارند.

مدل سازی داده ها فضای اطلاعاتی را به نمایش درمی آورد و اشیای داده را که نرم افزار دستکاری می کند و همچنین روابط میان آنها را به تصویر می کشد.

مدل سازی مبتنی بر کلاس ها به تعریف اشیا، صفات و روابط می پردازد.

پس از اینکه مدل های مقدماتی تهیه شدند مورد پالایش و تحلیل قرار می گیرند تا به وضوح، کمال و سازگاری لازم برسند.

محصول کاری چیست؟

آرایه گسترده ای از فرم های متنی و نموداری را می توان برای مدل خواسته ها برگزید.

هرکدام از این نمایش ها، دیدگاهی از یک یا چند عنصر مدل فراهم می سازند.

چطور اطمینان حاصل کنیم که درست از عهده کار برآمده ایم؟

محصولات کاری از مدل سازی خواسته ها را باید از نظر صحت، کمال و سازگاری بازبینی کرد.
این محصولات باید منعکس کننده نیازهای همه ی طرف های ذی نفع باشند و بستری برای انجام طراحی فراهم سازند.

تحلیل خواسته ها

Requirements Analysis

تحلیل خواسته ها:

به تعیین مشخصات خصوصیات عملیاتی نرم افزار منجر می شود،

واسط نرم افزار با سایر عناصر سیستم را مشخص می کند و

قید و بندهایی را که نرم افزار باید رعایت کند، تعیین می نماید.

تحلیل خواسته ها به شما این امکان را می دهد که طی وظایف دریافت، استخراج و چانه زنی جزئیات خواسته های پایه را تعیین کنید.

کنش مدل سازی خواسته به یک یا چند نوع از مدل های زیر می انجامد:

- مدل های مبتنی بر سناریو از دیدگاه کنش گران گوناگون سیستم.
- مدل های داده ای که دامنه اطلاعاتی مسئله را تصویر می کنند.
- مدل های مبتنی بر کلاس ها که کلاس های شیء گرا و شیوه همکاری این کلاس ها برای دستیابی به خواسته های سیستم را به نمایش می گذارند.
- مدل های جریان گرا که عناصر عملیاتی سیستم و چگونگی تبدیل داده ها توسط این عناصر را به هنگام حرکت در سیستم نمایش می دهند.
- مدل های رفتاری که چگونگی رفتار نرم افزار را به عنوان نتیجه ای از رویدادهای بیرونی به تصویر می کشند.

مدل ها

۱- مدل سازی مبتنی بر سناریو *Scenario-based models*

۲- مدل سازی داده ها *Data models*

۳- مدل سازی کلاس ها *Class-oriented models*

۴- مدل های جریان گرا *Flow-oriented models*

۵- مدل های رفتاری *Behavioral models*

۶- مدل سازی مبتنی بر الگوها

۷- مدل های مربوط به برنامه های تحت وب

فلسفه و اهداف کلی

Overall Objectives and Philosophy

Throughout requirements modeling, your primary focus is on *what,* not *how.*

مدل خواسته ها باید به سه هدف دست پیدا کند :

۱- توصیف آنچه که مشتری نیاز دارد.

۲- ایجاد مبنایی برای تهیه طراحی نرم افزار

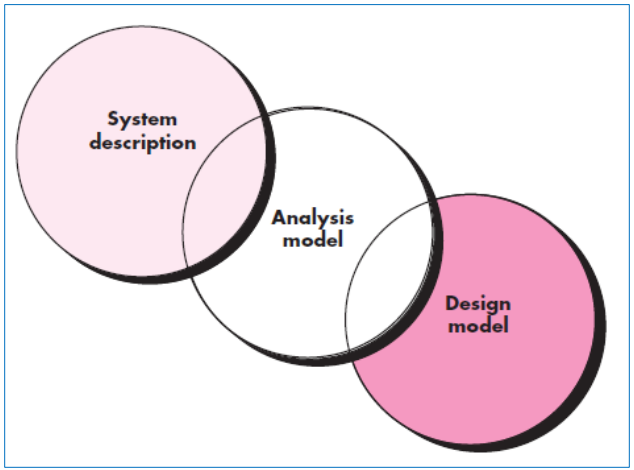
۳- تعریف مجموعه ای از خواسته ها که پس از ساخته شدن نرم افزار بتوان آنها را اعتبار سنجی کرد.

مدل خواسته ها به عنوان پلی میان توصیف سیستم و مدل طراحی

The requirements model as a bridge between the system description and the design model

مدل آنالیز، **پلی** است میان توصیف در سطح سیستم و یک طراحی نرم افزار.

طراحی نرم افزار ، معماری کاربرد نرم افزار، واسط های کاربری و ساختار را در سطح مولفه ها توصیف می کند.



همه عناصر در مدل خواسته ها به طور مستقیم تا بخش هایی از مدل طراحی قابل ردیابی خواهند بود.

مقداری از طراحی به عنوان بخشی از تحلیل و مقداری از تحلیل در طول طراحی انجام می شود.

قواعد ساده تحلیل

Analysis Rules of Thumb

بهتر است هنگام ایجاد مدل تحلیل ، قواعد زیر رعایت شوند:

- مدل باید خواسته هایی را کانون توجه قرار دهد که در دامنه تجاری یا مسئله، قابل مشاهده باشند. سطح انتزاع باید نسبتا بالا باشد.
- هرعنصر از مدل خواسته ها باید در کل چیزی به درک ما از خواسته های نرم افزار بیفزاید و دیدی از دامنه اطلاعاتی عملکرد و رفتار سیستم فراهم سازد.
- ملاحظات زیر ساختی و سایر مدل های غیر عملیاتی را تا طراحی به تاخیر اندازید.
- ارتباط ها را در سرتاسر سیستم به حداقل برسانید.
- مدل خواسته ها باید حتما رضایت همه طرف های ذی نفع را جلب کند.
- سادگی مدل را تا حد امکان حفظ کنیم.

تحلیل دامنه
Domain Analysis

تحلیل دامنه نرم افزار عبارت است از :

شناسایی، تحلیل و تعیین مشخصات خواسته های رایج از یک دامنه کاربردی خاص، معمولا برای استفاده مجدد در چندین پروژه که در همان دامنه ی کاربردی قرار دارند..

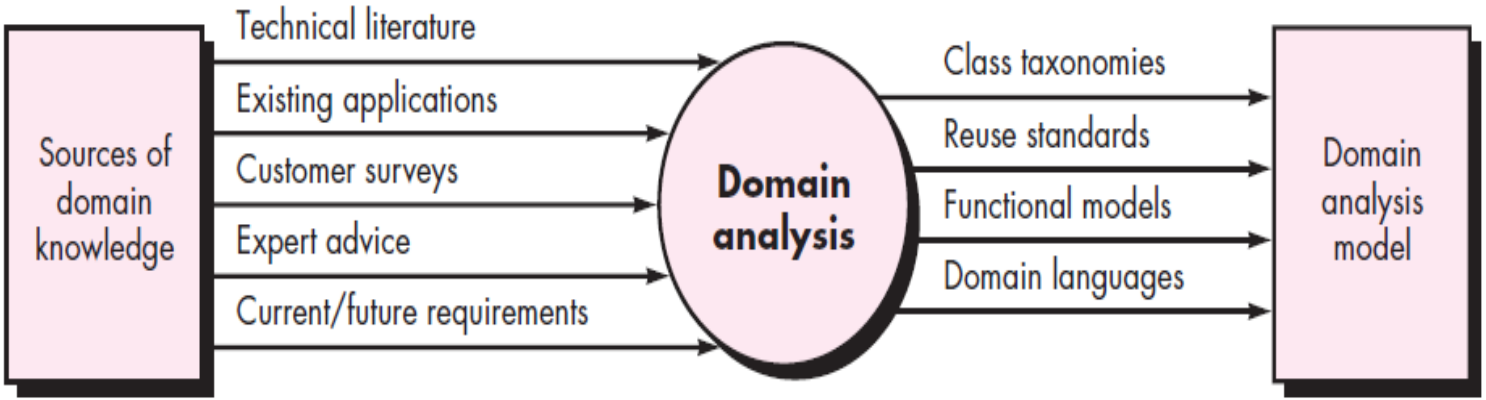
دامنه کاربرد خاص می تواند از هوانوردی تا بانکداری، از بازی های چند رسانه ای تا نرم افزارهای تعبیه شده در دستگاه های پزشکی را در برگیرد.

هدف تحلیل دامنه :

یافتن یا ایجاد کلاس های تحلیل که دارای کاربردی گسترده اند و می توان دوباره از آنها استفاده کرد.

ورودی و خروجی برای تحلیل دامنه

منابع اطلاعاتی دامنه، مورد نظر خواهی قرار می گیرد تا اشیای قابل استفاده مجدد در آن دامنه شناسایی گردد.



تحلیل دامنه

SafeHome

Domain Analysis

The scene: Doug Miller's office, after a meeting with marketing.

The players: Doug Miller, software engineering manager, and Vinod Raman, a member of the software engineering team.

روش های مدل سازی خواسته ها

Requirements Modeling Approaches

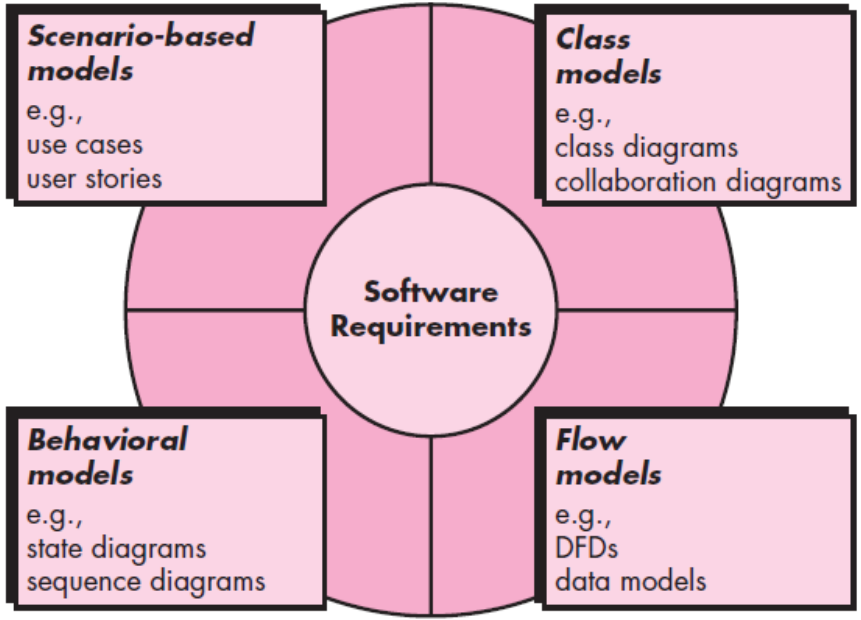
در یک نما از مدل سازی خواسته ها، که به تحلیل ساخت یافته مرسوم است،داده ها و فرایندهایی که این داده ها را تبدیل می کنند، به عنوان موجودیت هایی مجزا درنظر گرفته می شوند.

رویکرد دوم برای مدل سازی تحلیل، که تحلیل شیء گرا نامیده می شود، بر تعریف کلاس ها و شیوه همکاری آنها با یکدیگر برای برآورده ساختن خواسته های مشتری تاکید دارد.

UML و فرایند یکپارچه عمدتا شی ءگرا هستند.

عناصر مدل تحلیل

هر عنصر از مدل خواسته ها مساله را از دیدگاهی متفاوت نشان می دهد.



مدل سازی مبتنی بر سناریو

SCENARIO-BASED MODELING

ایجاد یک use case مقدماتی

(1) what to write about,

(2) how much to write about it,

(3) how detailed to make your description,

(4) how to organize the description?

توسعه یک سناریوی کاربری مقدماتی

برای قابلیت پایشی SafeHome surveillance function

SAFEHOME

The scene: A meeting room, during the second requirements gathering meeting.

The players: Jamie Lazar, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

SafeHome surveillance function

Facilitator

thumbnail views نمای شستی:

قابلیت پایش در محصول safe home قابلیت های زیر را مشخص می کند که کنش گر homeowner آنها را انجام می دهد:

- انتخاب دوربین برای مشاهده
- درخواست تصاویر کوچکی از همه دوربین ها
- به نمایش دراوردن نمای دوربین ها در یک پنجره pc
- کنترل زاویه و زوم یک دوربین مشخص
- ضبط انتخابی خروجی دوربین ها
- پخش خروجی دوربین ها
- دستیابی به پایش دوربین ها از طریق اینترنت

Use case مربوط به ACS-DCV

دستیابی به پایش دوربینی از طریق اینترنت- نمایش خروجی دوربین ها

کنش گر: homeowner

۱- صاحبخانه وارد وب سایت SafeHome Products می شود.

۲- صاحبخانه نام کاربری (user ID) خودش را وارد می کند .

۳- صاحبخانه دو کلمه عبور وارد می کند.

۴- سیستم همه دکمه های عملیاتی اصلی را به نمایش در می اورد.

۵- صاحبخانه پایش (surveillance) را از دکمه های اصلی انتخاب می کند.

۶- صاحبخانه "pick a camera" را برمی گزیند. (انتخاب دوربین)

۷- سیستم، نقشه ساختمان را نمایش می دهد.

۸- صاحبخانه ایکون یکی از دوربین ها را از روی نقشه انتخاب می کند.

۹- صاحبخانه دکمه view را انتخاب می کند.

۱۰- سیستم ، یک پنجره نمایش ظاهر می کند که با شماره شناسایی دوربین مشخص می شود.

۱۱- سیستم، خروجی دوربین را در پنجره نمایش با سرعت یک فریم در ثانیه نشان می دهد.

access camera surveillance via the Internet— display camera views **(ACS-DCV)**

پالایش یک use case مقدماتی

Refining a Preliminary Use Case

- *Can the actor take some other action at this point?*
- *Is it possible that the actor will encounter some error condition at this point?*
- *Is it possible that the actor will encounter some other behavior at this point ?*

۱- صاحبخانه وارد وب سایت SafeHome Products می شود.
۲- صاحبخانه نام کاربری (user ID) خودش را وارد می کند.
۳- صاحبخانه دو کلمه عبور وارد می کند.
۴- سیستم همه دکمه های عملیاتی اصلی را به نمایش در می اورد.
۵- صاحبخانه پایش (surveillance) را از دکمه های اصلی انتخاب می کند.

۶- صاحبخانه "pick a camera" را برمی گزیند.
۷- سیستم، نقشه ساختمان را نمایش می دهد.

۸- صاحبخانه ایکون یکی از دوربین ها را از روی نقشه انتخاب می کند.
۹- صاحبخانه دکمه view را انتخاب می کند.
۱۰- سیستم ، یک پنجره نمایش ظاهر می کند که با شماره شناسایی دوربین مشخص می شود.
۱۱- سیستم، خروجی دوربین را در پنجره نمایش با سرعت یک فریم در ثانیه نشان می دهد.

نوشتن یک use case رسمی

Use case های غیر رسمی گاهی برای مدل سازی خواسته ها کفایت می کنند.

ولی هنگامی که یک use case شامل فعالیتی مهم می شود یا مجموعه پیچیده ای از مراحل

را با تعداد چشمگیری از استثناها توصیف می کند، روش رسمی تر ممکن است مطلوب باشد.

SAFEHOME

Use Case Template for Surveillance

Use case: Access camera surveillance via the Internet—display camera views (ACS-DCV)

Iteration: 2, last modification: January 14 by V. Raman.

Primary actor: Homeowner.

Goal in context: To view output of camera placed throughout the house from any remote location via the Internet.

Preconditions: System must be fully configured; appropriate user ID and passwords must be obtained.

Trigger: The homeowner decides to take a look inside the house while away.

Scenario:

1. The homeowner logs onto the *SafeHome Products* website.
2. The homeowner enters his or her user ID.
3. The homeowner enters two passwords (each at least eight characters in length).
4. The system displays all major function buttons.
5. The homeowner selects the "surveillance" from the major function buttons.
6. The homeowner selects "pick a camera."
7. The system displays the floor plan of the house.
8. The homeowner selects a camera icon from the floor plan.
9. The homeowner selects the "view" button.
10. The system displays a viewing window that is identified by the camera ID.
11. The system displays video output within the viewing window at one frame per second.

Exceptions:

1. ID or passwords are incorrect or not recognized—see use case **Validate ID and passwords.**
2. Surveillance function not configured for this system—system displays appropriate error message; see use case **Configure surveillance function.**
3. Homeowner selects "View thumbnail snapshots for all camera"—see use case **View thumbnail snapshots for all cameras.**
4. A floor plan is not available or has not been configured—display appropriate error message and see use case **Configure floor plan.**
5. An alarm condition is encountered—see use case **Alarm condition encountered.**

Priority: Moderate priority, to be implemented after basic functions.

When available: Third increment.

Frequency of use: Moderate frequency.

Channel to actor: Via PC-based browser and Internet connection.

Secondary actors: System administrator, cameras.

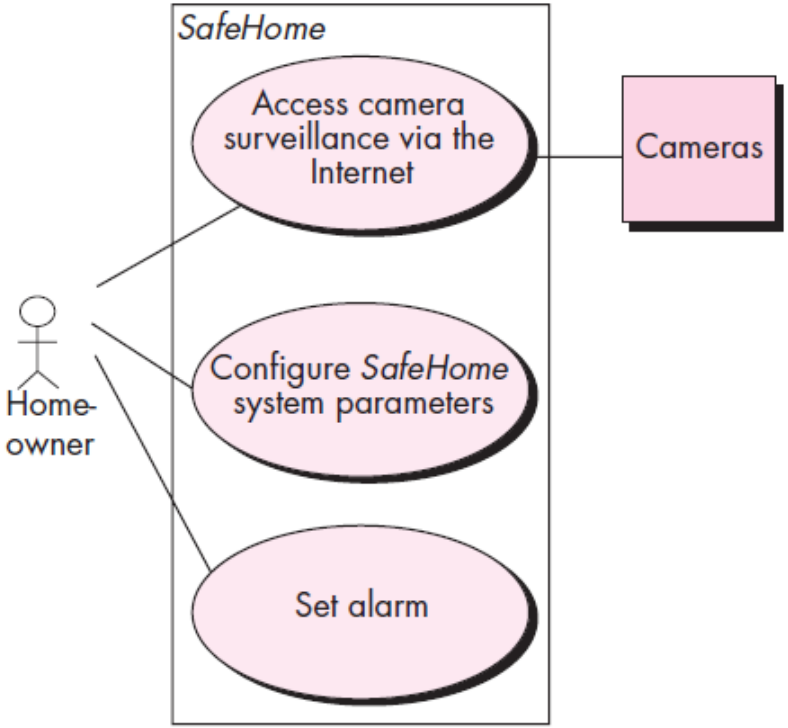
Channels to secondary actors:

1. System administrator: PC-based system.
2. Cameras: wireless connectivity.

Open issues:

1. What mechanisms protect unauthorized use of this capability by employees of *SafeHome Products*?
2. Is security sufficient? Hacking into this feature would represent a major invasion of privacy.
3. Will system response via the Internet be acceptable given the bandwidth required for camera views?
4. Will we develop a capability to provide video at a higher frames-per-second rate when high-bandwidth connections are available?

نمودار یک use case مقدماتی برای سیستم SafeHome



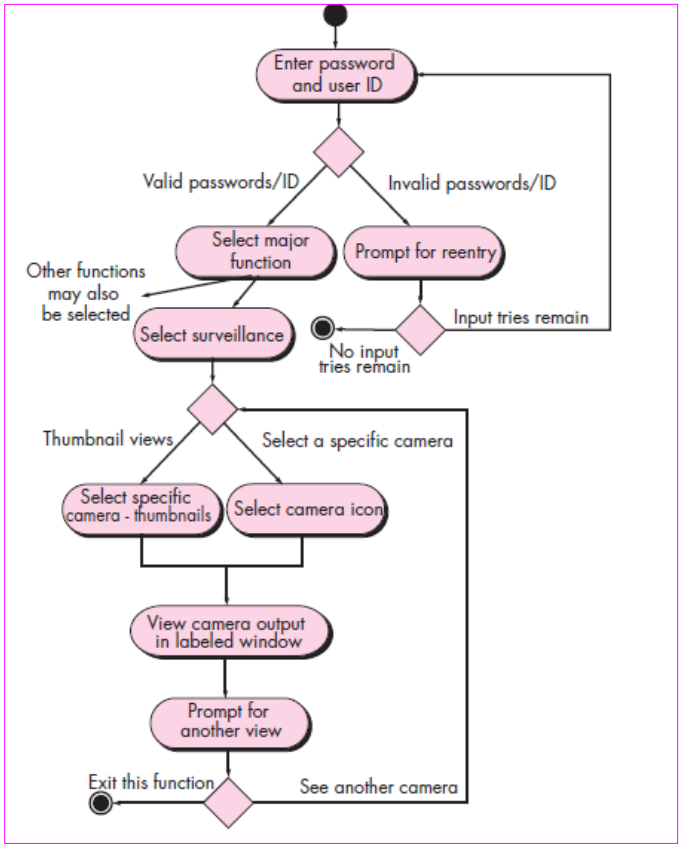
مدل های UML که use case را تکمیل می کنند

وضعیت های زیادی وجود دارد که مدل های مبتنی بر متن ممکن است اطلاعات را واضح ارائه ندهد.

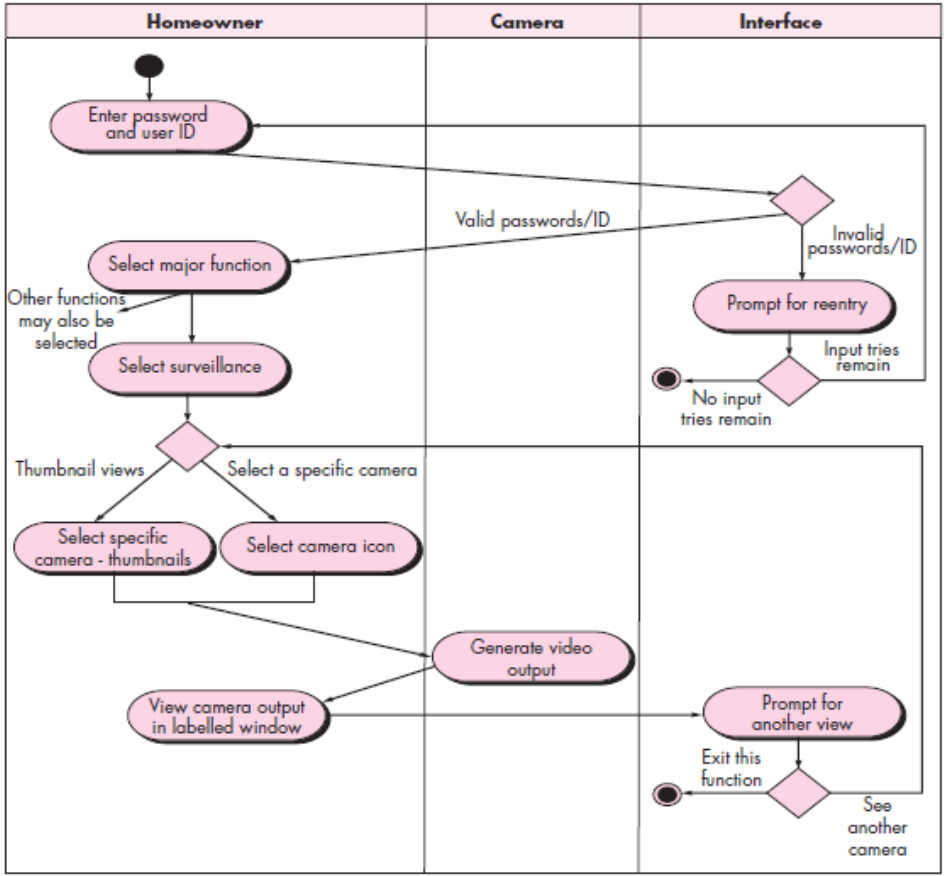
توسعه نمودار فعالیت ها

نمودارهای بخش بندی

نمودار فعالیت ها برای دستیابی به پایش دوربین ها از طریق اینترنت- عملکرد مشاهده دوربین



نمودار بخش بندی برای دستیابی به پایش دوربین ها از طریق اینترنت- عملکرد مشاهده دوربین



مفاهیم مدل سازی داده ها

Data Modeling Concepts

مهندس نرم افزار یا تحلیل گر، همه اشیای داده را که در سیستم پردازش می شوند، روابط میان اشیای داده و سایر اطلاعات مرتبط با این روابط را تعیین می کند.

نمودار موجودیت–ارتباط (ERD) به این مسائل می پردازد و همه اشیای داده را که در یک برنامه کاربردی، وارد، ذخیره، تبدیل و تولید می شوند، به نمایش می گذارند.

entity-relationship diagram (ERD)

اشیای داده

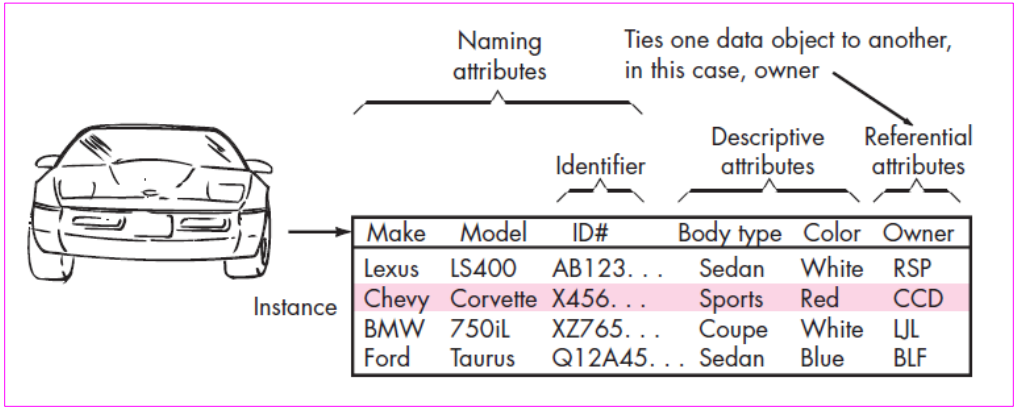
Data Objects

شیء داده نمایشی از اطلاعات مرکب است که باید نرم افزار آنها را بفهمد.

منظور از اطلاعات مرکب چیزی است که دارای چند صفت یا خاصیت متفاوت باشد.

شیء داده تنها داده ها را پنهان سازی می کند.

در داخل یک شیء داده هیچ آدرسی برای عملیات قابل انجام روی داده ها وجود ندارد.
بنابراین، شیء داده را می توان به صورت جدول شکل زیر به نمایش گذاشت.



صفات داده ها

Data Attributes

صفات داده ها، خواص یک شیء داده را تعریف می کنند.

از آنها می توان برای موارد زیر استفاده کرد :

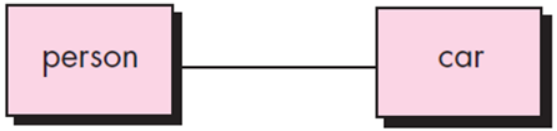
۱- نامگذاری نمونه ای از شیء داده ای

۲- توصیف نمونه

۳- ارجاع به نمونه ای دیگر در جدولی دیگر

ارتباطات

Relationships



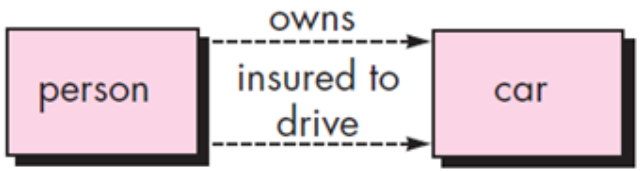
اشیای داده به طرق گوناگون بهم متصل می شوند.

دو شیء داده person و car را در نظر بگیرید.

می توانید مجموعه ای از روابط جفتی میان اشیا تعیین کنید که این ارتباط ها را مشخص کند.

برای مثال

- شخصی صاحب یک خودرو است.
- شخصی مجوز رانندگی خودرو را دارد.



مدل سازی مبتنی بر کلاس ها

Class-Based Modeling

عناصر مدل :

- classes and objects,
- attributes,
- operations,
- class- responsibility-collaborator (CRC) models,
- collaboration diagrams,
- packages.

شناسایی کلاس های تحلیل

Identifying Analysis Classe

می توانیم شناسایی کلاس ها را با بررسی سناریوهای کاربری آغاز کنیم که به عنوان بخشی از مدل خواسته ها توسعه می یابند و use case های تهیه شده برای سیستم را تجزیه گرامری کنیم.

کلاس ها با خط کشیدن زیر اسم ها یا عبارت های اسمی و وارد کردن آنها در یک جدول ساده تعیین می شوند.

کلاس های تحلیل، خود را به یکی از طرق زیر نشان می دهند:

- موجودیت های خارجی که اطلاعات مورد استفاده یک سیستم کامپیوتری را تولید یا مصرف می کنند.
- چیزهایی که بخشیء از دامنه اطلاعاتی مسئله اند.
- رخدادها یا رویدادهایی که در حیطه ی عملیاتی سیستم به وقوع می پیوندند.
- نقش هایی که توسط افراد درحال تعامل با سیستم ایفا می شوند
- واحدهای سازمانی که به کاربردی خاص مربوط می شوند.
- مکان هایی که حیطه مسئله و عملکرد کلی سیستم را تعیین می کند.
- ساختارهایی که کلاسی از اشیا یا کلاس های مرتبطی از اشیا را تعریف می کند.

روایت پردازش

processing narrative

برای اینکه نشان دهیم کلاس های تحلیل را چگونه می توان طی مراحل اولیه مدل سازی تعریف کرد،

تجزیه گرامری روایت پردازش قابلیت امنیتی در محصول SafeHome را در نظر بگیرید.

زیر اسم ها خط کشیده شده است.

فعل ها به صورت ایتالیک نشان داده می شوند.

The SafeHome security function *enables* the homeowner to *configure* the security system when it is *installed, monitors* all sensors *connected* to the security system, and *interacts* with the homeowner through the Internet, a PC, or a control panel.

During installation, the SafeHome PC is used to *program* and *configure* the system. Each sensor is assigned a number and type, a master password is programmed for *arming* and *disarming* the system, and telephone number(s) are *input* for *dialing* when a sensor event occurs.

When a sensor event is *recognized*, the software *invokes* an audible alarm attached to the system. After a delay time that is *specified* by the homeowner during system configuration activities, the software dials a telephone number of a monitoring service, *provides* information about the location, *reporting* the nature of the event that has been detected. The telephone number will be *redialed* every 20 seconds until telephone connection is *obtained*.

The homeowner *receives* security information via a control panel, the PC, or a browser, collectively called an interface. The interface *displays* prompting messages and system status information on the control panel, the PC ,or the browser window. Homeowner interaction takes the following form . . .

با استخراج اسم ها، چند کلاس بالقوه می توان پیشنهاد داد:

Potential Class	General Classification
homeowner	role or external entity
sensor	external entity
control panel	external entity
installation	occurrence
system (alias security system)	thing
number, type	not objects, attributes of sensor
master password	thing
telephone number	thing
sensor event	occurrence
audible alarm	external entity
monitoring service	organizational unit or external entity

انتخاب کلاس های قانونی

Potential Class	Characteristic Number That Applies
homeowner	rejected: 1, 2 fail even though 6 applies
sensor	accepted: all apply
control panel	accepted: all apply
installation	rejected
system (alias security function)	accepted: all apply
number, type	rejected: 3 fails, attributes of sensor
master password	rejected: 3 fails
telephone number	rejected: 3 fails
sensor event	accepted: all apply
audible alarm	accepted: 2, 3, 4, 5, 6 apply
monitoring service	rejected: 1, 2 fail even though 6 applies

۱- اطلاعات نگه داری شده

۲- سرویس های لازم

۳- صفات چندگانه

۴- صفات مشترک

۵- عملیات مشترک

۶- خواسته های اساسی

نحوه تعیین اینکه یک کلاس بالقوه یک کلاس تحلیل است؟

۱- اطلاعات نگه داری شده

کلاس بالقوه، تنها درصورتی در تحلیل مفید خواهد شد که به خاطر سپردن اطلاعات مربوط به آن، برای عملکرد سیستم ضروری باشد.

۲- سرویس های لازم

کلاس بالقوه باید دارای مجموعه ای از عملیات قابل شناسایی باشد که بتوانند مقدار صفات آن را به طریقی تغییر دهند.

۳- صفات چندگانه

۴- صفات مشترک

مجموعه ای از صفات که برای کلاس های بالقوه، قابل تعریف است و این عملیات ها در همه نمونه های کلاس مصداق دارند.

۵- عملیات مشترک

مجموعه ای از عملیات ها که برای کلاس های بالقوه، قابل تعریف است و این صفات در همه نمونه های کلاس تعریف می شوند.

۶- خواسته های اساسی

موجودیت های خارجی که در فضای مسئله ظاهر می شوند و اطلاعات ضروری جهت عملکرد هر راهکار برای سیستم را تولید یا مصرف می کنند،

همچنین در مدل خواسته ها همواره به عنوان کلاس تعریف می شوند.

مشخص کردن صفات

Specifying Attributes

صفات، مجموعه ای از اشیای داده هستند که یک کلاس را به طور کامل در حیطه ی مساله تعریف می کنند.

name,
position,
batting average,
fielding percentage,
years played,
games played

average salary
mailing address
credit toward full vesting
pension plan options chosen

برای توسعه مجموعه ای با معنی از صفات برای یک کلاس تحلیل باید هرکدام از use case ها را مطالعه کنید و آن چیزهایی را انتخاب کنید که به طور منطقی به کلاس **تعلق** دارند.

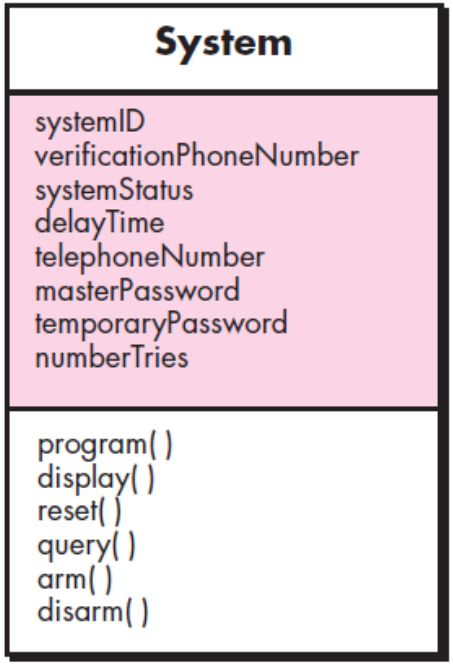
کلاس system برای SafeHome

identification information = system ID + verification phone number + system status

alarm response information = delay time + telephone number

activation/deactivation information = master password + temporary password + number tries

نمایش اقلام داده ای مرکب:



تعریف عملیات ها

Defining Operations

عملیات ها، رفتار شیء را تعریف می کنند.

عملیات ها را در چهار گروه دسته بندی می کنند:

۱- عملیاتی که داده ها را به طریقی دستکاری می کنند.

۲- عملیات هایی که محاسبه انجام می دهند.

۳- عملیات هایی که درباره حالت یک شیء تحقیق می کنند.

۴- عملیات هایی که یک شیء را برای وقوع یک رویداد کنترل کننده پایش می کنند.

عملیات باید از ماهیت همبستگی ها و صفات کلاس اگاهی داشته باشد

جدا سازی افعال

- That an *assign()* operation is relevant for the **Sensor** class.
- That a *program()* operation will be applied to the **System** class.
- That *arm()* and *disarm()* are operations that apply to **System** class.

The SafeHome security function *enables* the homeowner to *configure* the security system when it is *installed, monitors* all sensors *connected* to the security system, and *interacts* with the homeowner through the Internet, a PC, or a control panel.

During installation, the SafeHome PC is used to *program* and *configure* the system. Each sensor is assigned a number and type, a master password is programmed for *arming* and *disarming* the system, and telephone number(s) are *input* for *dialing* when a sensor event occurs.

When a sensor event is *recognized*, the software *invokes* an audible alarm attached to the system. After a delay time that is *specified* by the homeowner during system configuration activities, the software dials a telephone number of a monitoring service, *provides* information about the location, *reporting* the nature of the event that has been detected. The telephone number will be *redialed* every 20 seconds until telephone connection is *obtained*.

The homeowner *receives* security information via a control panel, the PC, or a browser, collectively called an interface. The interface *displays* prompting messages and system status information on the control panel, the PC ,or the browser window. Homeowner interaction takes the following form . . .

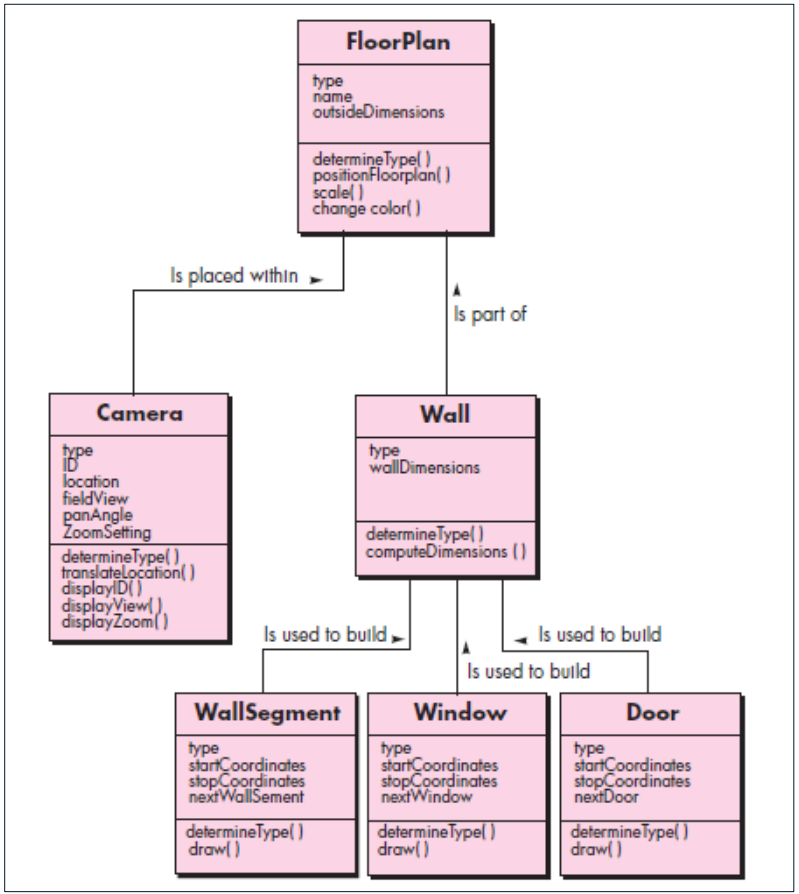
مدل کلاس ها

SafeHome

Class Models

The scene: Ed's cubicle, as requirements modeling begins.

The players: Jamie, Vinod, and Ed—all members of the *SafeHome* software engineering team.



مدل سازی همکار مسئوولیت کلاس ها

Class-Responsibility-Collaborator (CRC) Modeling

مدل CRC ، مجموعه ای از کارت های شاخص استاندارد است که کلاس ها را به نمایش می گذارند.

Class: FloorPlan	
Description	
Responsibility:	**Collaborator:**
Defines floor plan name/type	
Manages floor plan positioning	
Scales floor plan for display	
Scales floor plan for display	
Incorporates walls, doors, and windows	**Wall**
Shows position of video cameras	**Camera**

این کارت ها به سه بخش تقسیم می شوند:

- بالای کارت : نام کلاس
- سمت راست بدنه کارت : فهرست مسئوولیت های کلاس
- سمت چپ بدنه کارت : همکاران

برای مدل CRC از کارت های واقعی یا مجازی استفاده می شود.

کلاس ها

طبقه بندی انواع کلاس ها ارائه شده در قبل را می توان با در نظر گرفتن گروه های زیر بسط داد :

- کلاس های موجودیت (مدل یا تجاری) : مستقیما از بیان مسئله استخراج می شوند.
- کلاس های مرزی : در ایجاد واسط به کار می روند که کاربر به هنگام استفاده از نرم افزار و تعامل مشاهده می کند.
- کلاسهای کنترل گر : یک واحد کار را از ابتدا تا انتها مدیریت می کند.

کلاس های کنترلر گر را می توان طوری طراحی کرد که موارد زیر را مدیریت کنند:

۱- ایجاد یا به هنگام سازی اشیای موجودیت

۲- معرفی اشیای مرزی به هنگام کسب اطلاعات از اشیای موجودیت

۳- ارتباطات پیچیده میان مجموعه های اشیا

۴- اعتبار سنجی داده های ارتباطی میان اشیا یا میان کاربر و برنامه

مسئوولیت ها

پنج دستورالعمل برای تخصیص مسئوولیت ها به کلاس ها :

۱- هوشمندی سیستم باید طوری میان کلاس ها توزیع شود که به بهترین وجه پاسخ گوی نیازهای مسئله باشد.

۲- هر مسئولیتی باید تا حد امکان به صورت کلی بیان شود.

۳- اطلاعات و رفتار مرتبط با آن باید در یک کلاس قرار داده شوند.

۴- اطلاعات مربوط به یک چیز باید تنها در یک کلاس قرار داده شوند و نباید در میان چند کلاس توزیع شوند.

۵- مسئوولیت ها را در صورت امکان باید در میان کلاس های مرتبط به اشتراک گذاشت.

همکاری ها

کلاس ها به یکی از دو شیوه، مسئوولیت های خود را به انجام می رسانند:

۱- کلاس می تواند از عملیات های خودش برای دستکاری صفات استفاده کند.

۲- کلاس می تواند با سایر کلاس ها همکاری کند.

تعریف:

همکاری ها نشان گر درخواست های یک کلاینت از سرور برای به انجام رساندن مسئوولیت یک کلاینت هستند.
همکاری ، تجسم هم پیمانی کلاینت و سرور است.

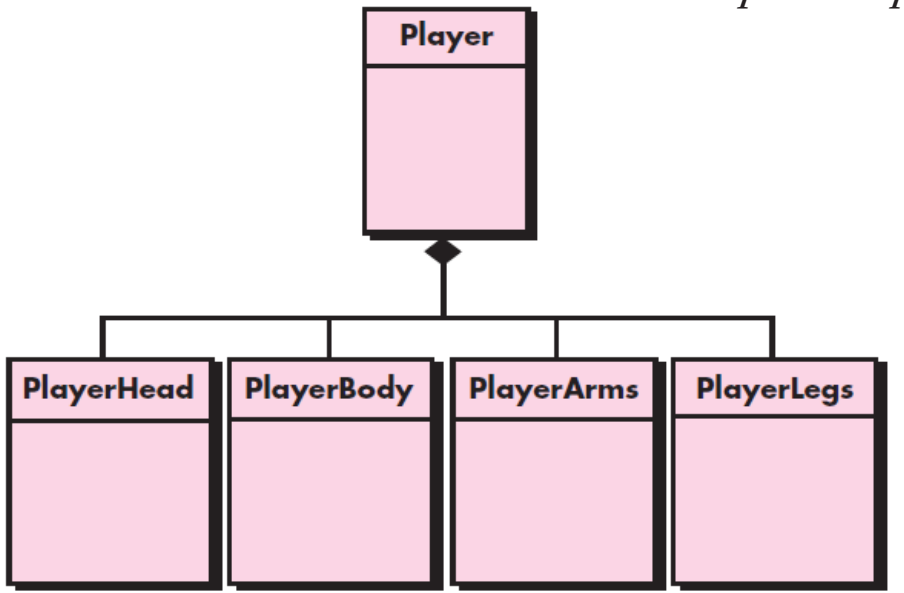
روابط میان کلاس ها

برای کمک به شناسایی همکارها سه رابطه کلی میان کلاس ها رابررسی می کنیم :

۱- رابطه شمول: *is-part-of*

۲- رابطه اگاهی داشتن از : *has-knowledge-of*

۳- بستگی داشتن به : *depends-upon*



مدل های CRC

SafeHome

CRC Models

The scene: Ed's cubicle, as requirements modeling begins.

The players: Vinod and Ed—members of the *SafeHome* software engineering team.

Use case: *SafeHome* home management function.

Narrative:

We want to use the home management interface on a PC or an Internet connection to control electronic devices that have wireless interface controllers.

The system should allow me to turn specific lights on and off, to control appliances that are connected to a wireless interface, to set my heating and air conditioning system to temperatures that I define. To do this, I want to select the devices from a floor plan of the house. Each device must be identified on the floor plan. As an optional feature, I want to control all audiovisual devices—audio, television, DVD, digital recorders, and so forth.

With a single selection, I want to be able to set the entire house for various situations. One is *home,* another is *away,* a third is *overnight travel,* and a fourth is *extended travel.* All of these situations will have settings that will be applied to all devices. In the *overnight travel* and *extended travel* states, the system should turn lights on and off at random intervals (to make it look like someone is home) and control the heating and air conditioning system. I should be able to override these setting via the Internet with appropriate password protection . . .

Attributes:
optionsPanel
situationPanel
Floorplan
deviceIcons
devicePanels

Operations:
displayControl(),
selectControl(),
displaySituation(),
select situation(),
accessFloorplan(),
selectDeviceIcon(),
displayDevicePanel(),
accessDevicePanel(), . . .

Class: HomeManagementInterface

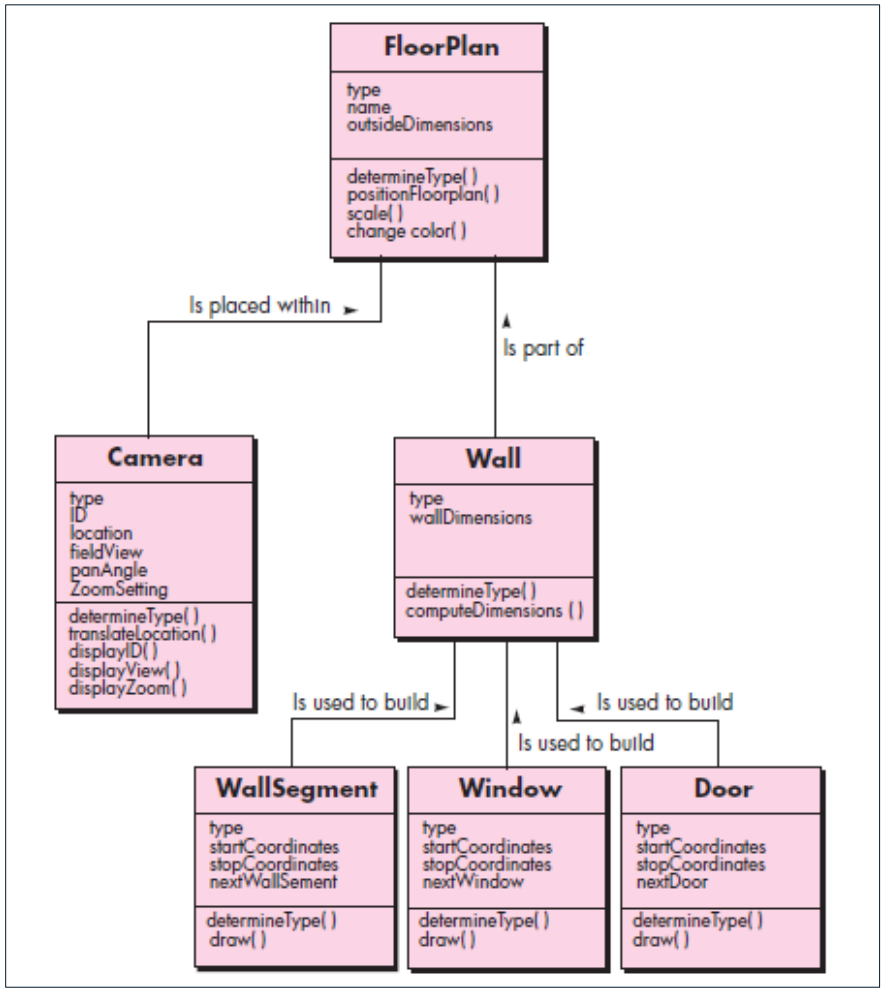
Responsibility	**Collaborator**
displayControl()	**OptionsPanel** (class)
selectControl()	**OptionsPanel** (class)
displaySituation()	**SituationPanel** (class)
selectSituation()	**SituationPanel** (class)
accessFloorplan()	**FloorPlan** (class) . . .

اجتماع

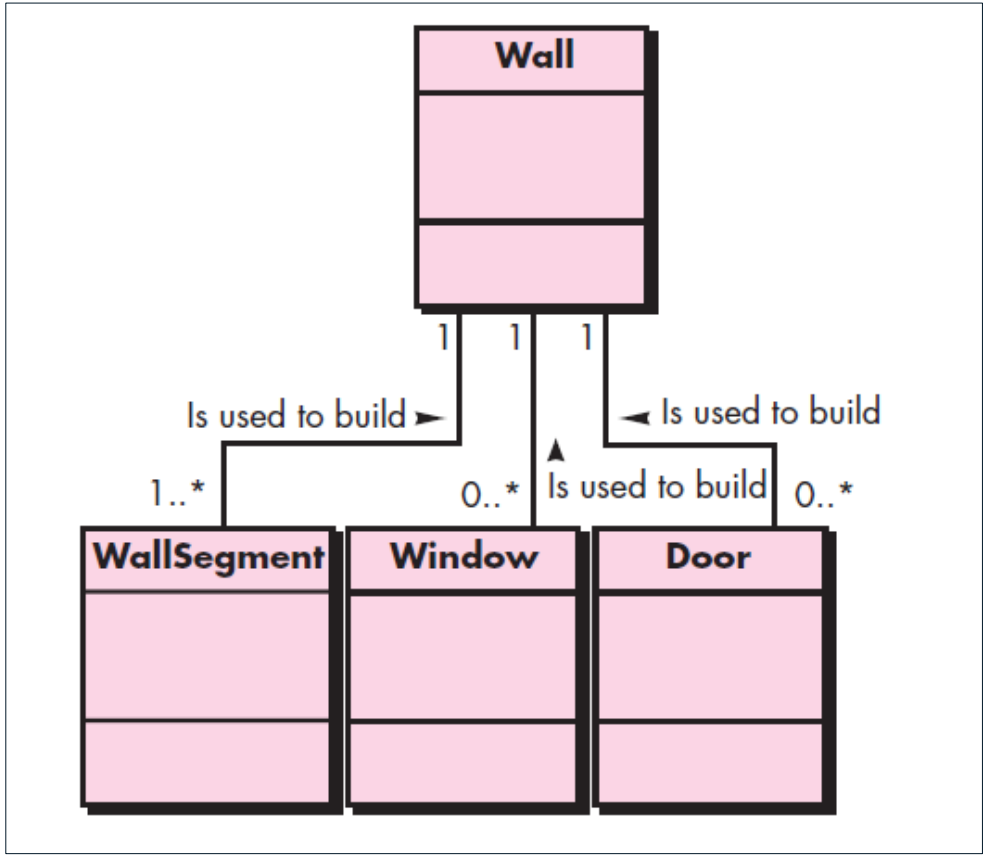
Associations

در بسیاری از موارد، دو کلاس تحلیل، به شیوه ای بسیار مشابه با ارتباط دوشیء داده ای، با هم ارتباط دارند.

در UML این روابط را اجتماع می نامند.



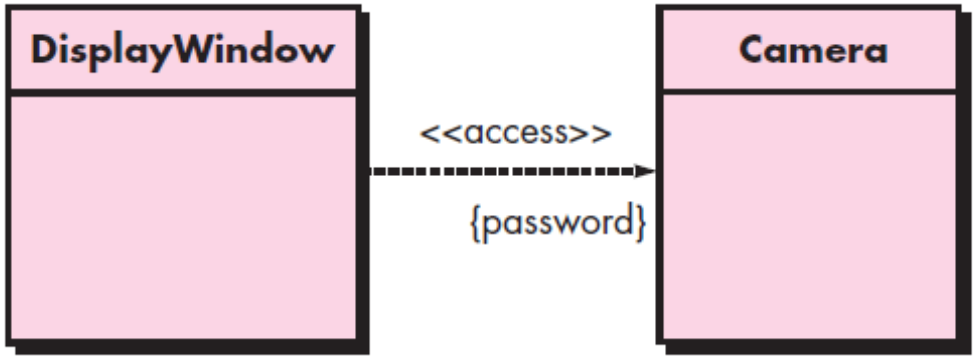
چند گانگی
Multiplicity



وابستگی

Dependencies

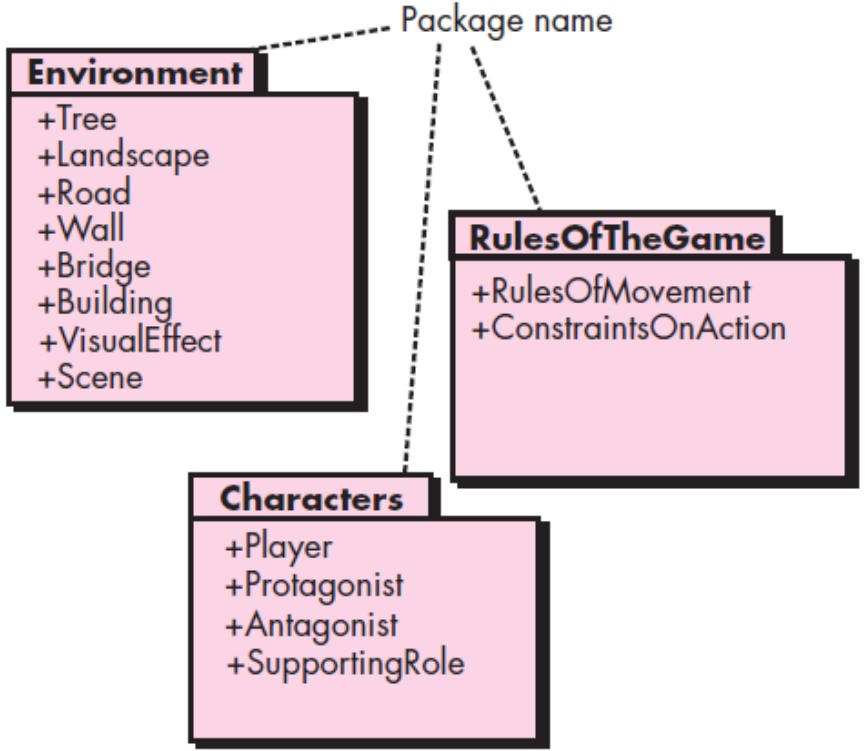
وابستگی ها توسط یک کلیشه (stereotype) تعریف می شوند.



<< access >> به این معنی است که استفاده از خروجی دوربین توسط یک کلمه عبور خاص کنترل می شود.

پکیج های تحلیل

Analysis Packages



بخش مهمی از مدل سازی تحلیل گروه بندی است.

یعنی عناصر گوناگون مدل تحلیل طوری گروه بندی می شوند که یک پکیج از آنها تشکیل شود و به آنها پکیج تحلیل گفته می شود. به هر پکیج یک نام داده می شود.

پکیج برای بسته بندی مجموعه ای از کلاس های مرتبط به کار می رود.

علامت مثبت، قبل از نام کلاس ها تحلیل در هر پکیج نشانگر ان است که این کلاس ها در معرض دید عموم هستند و بنابراین از طریق سایر پکیج ها قابل دستیابی اند.

علامت منفی، نشان می دهد که عنصر از تمام پکیج های دیگر پنهان است.

پایان فصل ۶

مهندسی نرم افزار
فصل ۷: مدل سازی خواسته ها :
جریان ، رفتار، الگوها و برنامه های تحت وب

مدرس:

فرشید شیرافکن

دانشجوی **دکتری دانشگاه تهران**

(کارشناسی و کارشناسی ارشد : کامپیوتر نرم افزار) (دکتری: بیو انفورماتیک)

مدل سازی خواسته ها چیست؟

چه کسی آن را انجام می دهد؟

چرا اهمیت دارد؟

مراحل کار کدام است؟

محصول کار چیست؟

چگونه مطمئن شوم که درست از عهده کار بر آمده ام؟

انواع رویکرد برای مدل تحلیل

۱- تحلیل ساخت یافته:

داده ها و فرآیندهایی که این داده ها را تبدیل می کنند، موجودیت های مجزا در نظر گرفته شوند.

اشیاءی داده ای به شیءوه ای مدل سازی می شوند که صفات و روابط میان آنها را تعریف کند.

فرآیندهای که اشیای داده ای را دستکاری می کنند، به شیوه ای مدل سازی می شوند که چگونگی تبدیل اشیاءی داده ای را به هنگام جریان یافتن آن ها در سیستم نشان دهند.

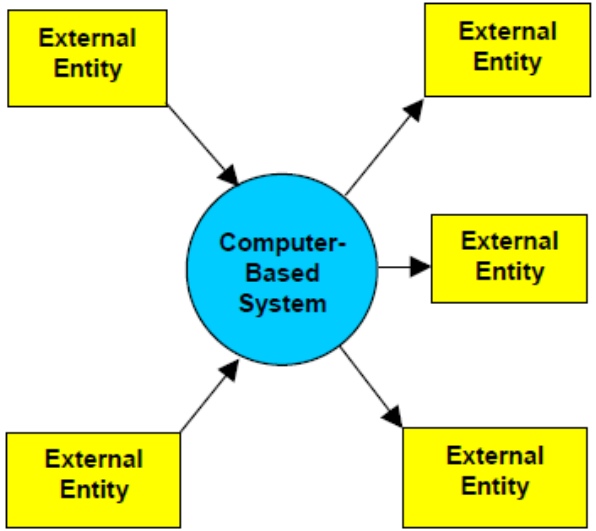
۲- تحلیل شیء گرا:

بر تعریف کلاس ها و شیءو ه ی همکاری آنها با یکدیگر برای برآورده ساختن خواسته های مشتری تأکید دارد.

مدل سازی جریان گرا

data flow diagram

DFD نمایی از سیستم بر اساس ورودی- فرآیند- خروجی به دست می دهد.



General Form of a Level 0 DFD.

یعنی اشیاءی داده ای به درون نرم افزار جریان پیدا می کنند،

توسط عناصر پردازشیء تبدیل می شوند و

اشیاءی داده ای حاصل به بیرون نرم افزار جریان پیدا می کنند.

اشیاءی داده ای با پیکان های بر چسب دار نمایش داده می شوند.

تبدیلات با دایره یا حباب نمایش داده می شوند.

DFD به شیءوه ی سلسله مراتبی نمایش داده می شود.

یعنی اولین مدل در جریان داده ها (DFD سطح صفر یا نمودار حیطه ای) کل سیستم را نمایش می دهد.

نمودارهای بعدی جریان داده ها، نمودار حیطه ای را پالایش می کنند و در هر سطح بعدی، جزئیات بیشتری اضافه می شود.

ایجاد مدل جریان داده ها

با نمودار جریان داده ها می توان مدل هایی از دامنه ی اطلاعاتی و دامنه عملیاتی را توسعه داد.

چند دستورالعمل ساده می تواند در به دست آوردن نمودار جریان داده ها کمک کند

۱- نمودار جریان داده ها در سطح صفر باید نرم افزار / سیستم را به عنوان یک حباب منفرد تصویر کند.

۲- ورودی و خرجی اولیه باید به دقت ذکر شود.

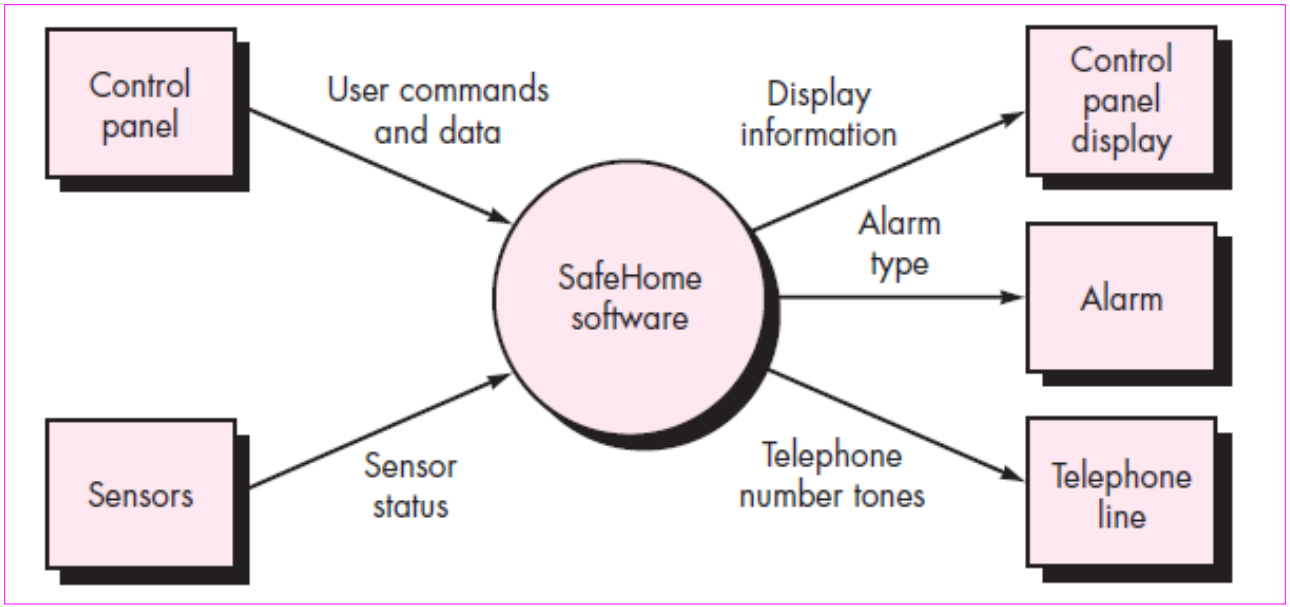
۳- پالایش باید با جداسازی فرآیندهای کاندیدا، اشیاءی داده ای و مخزن های داده ای که قرار است در سطح بعد به نمایش در آیند، آغاز گردد.

۴- همه پیکان ها و حباب ها باید با نام های مناسب نشان گذاری شوند.

۵- پیوستگی جریان اطلاعات باید از سطحی به سطح دیگر حفظ شود.

۶- هر بار تنها یک حباب را باید پالایش کرد.

DFD سطح صفر



چهار گوش ها: اطلاعات مورد نیاز سیستم را تولید و اطلاعات تولید شده توسط سیستم را مصرف می کنند.

پیکان های بر چسب دار : اشیاءی داده ای یا سلسله مراتب هایی از اشیاءی داده ای

تجزیه گرامری

The SafeHome security function *enables* the homeowner to *configure* the security system when it is *installed, monitors* all sensors *connected* to the security system, and *interacts* with the homeowner through the Internet, a PC, or a control panel.

During installation, the *SafeHome* PC is used to *program* and *configure* the system. Each sensor is assigned a number and type, a master password is programmed for *arming* and *disarming* the system, and telephone number(s) are *input* for *dialing* when a sensor event occurs.

When a sensor event is *recognized*, the software *invokes* an audible alarm attached to the system. After a delay time that is specified by the homeowner during system configuration activities, the software dials a telephone number of a monitoring service, *provides* information about the location, *reporting* the nature of the event that has been detected. The telephone number will be *redialed* every 20 seconds until telephone connection is *obtained*.

The homeowner *receives* security information via a control panel, the PC, or a browser, collectively called an interface. The interface *displays* prompting messages and system status information on the control panel, the PC, or the browser window. Homeowner interaction takes the following form . . .

در متن روایی :

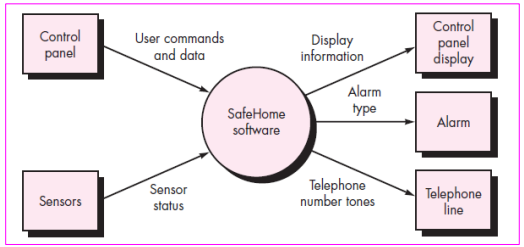
- فعل ها (حباب)
- اسم ها یا موجودیت های خارجی (چهار گوش ها)
- اشیاءی داده ای و کنترل (پیکان ها)
- مخزن داده ها (خطوط موازی)

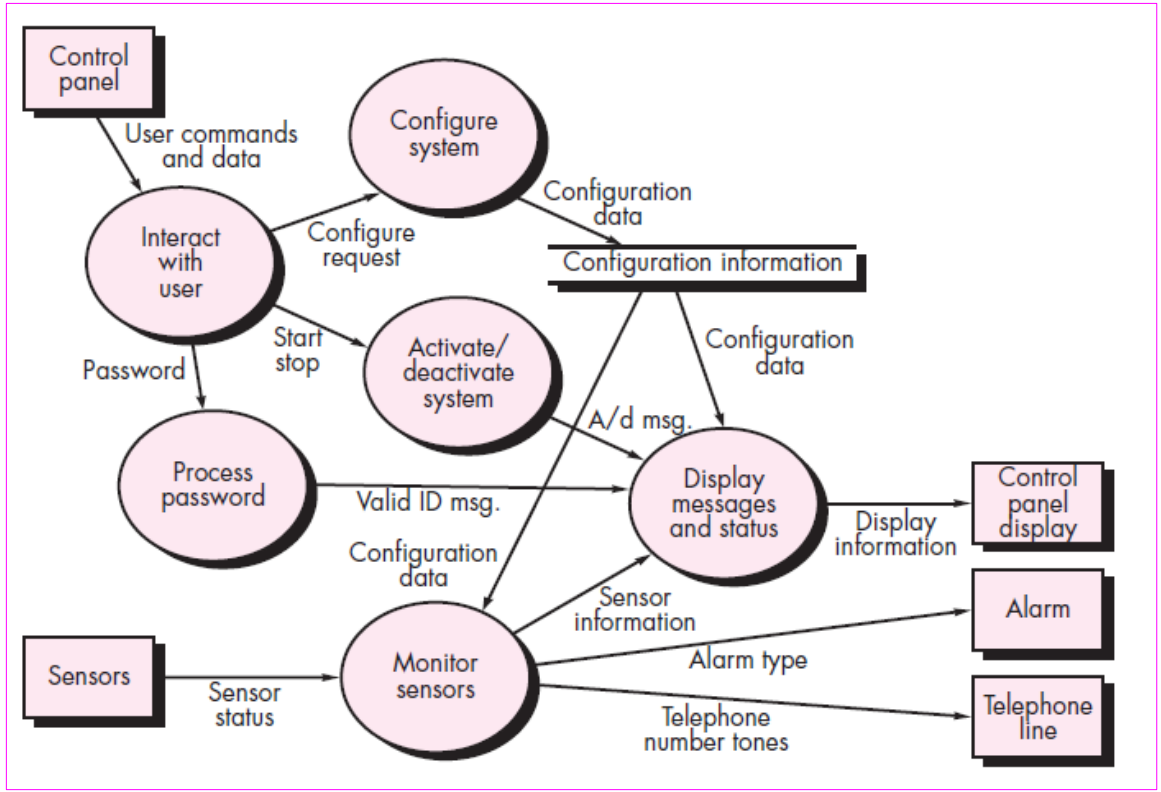
فعل ها و اسم ها را نیز می توان به هم مرتبط کرد.

DFD سطح ۱

فرآیند به ۶ فرآیند بسط داده شده است که از بررسی تجزیه گرامری به دست آمده اند.

پیوستگی جریان اطلاعات بین سطوح صفر و یک حفظ شده است.

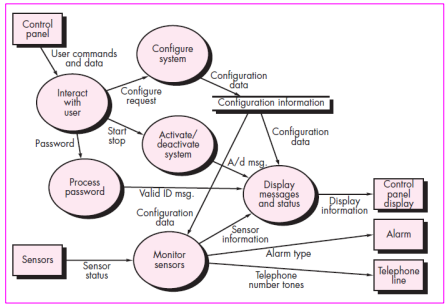


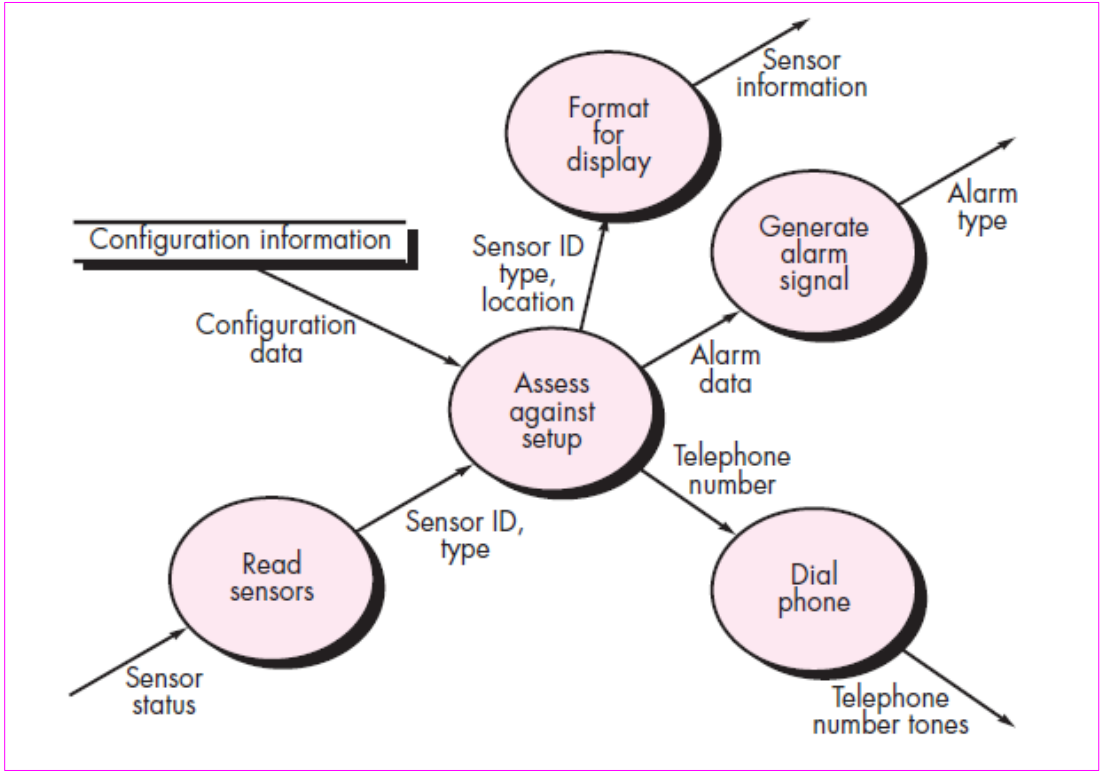


DFD سطح ۲

فرآیندهای DFD سطح یک را باز هم میتوان به سطوح پایین تر پالایش کرد.

برای مثال فرایند monitor sensors را می توان به یک DFD سطح دو پالایش کرد.





پالایش DFD تا کجا ادامه می یابد؟

پالایش DFD ها چندان ادامه می یابد که هر حباب تنها یک عملکرد را نشان دهد.

یعنی تا هنگامی که فرایند نشان داده شده توسط حباب، عملی را انجام دهد که به راحتی به عنوان یک مؤلفه ی برنامه قابل پیاده سازی باشد.

ایجاد مدل جریان کنترل

گروه بزرگی از برنامه های کاربردی بیشتر توسط رویدادها اداره می شوند تا توسط داده ها،

بیشتر اطلاعات کنترلی تولید می کنند تا گزارش و چیزهایی برای نمایش.

و اطلاعات را با توجه جدی به زمان و کارایی پردازش می کنند.

این گونه برنامه های کاربردی علاوه بر مدل سازی جریان داده ها به مدل سازی جریان کنترل هم نیاز دارند.

یک آیتم کنترلی یا رویدادی به صورت مقدار بولی یا فهرستی مجزا از شرط ها پیاده سازی می شود.

برای انتخاب رویدادهای بالقوه ی کاندیدا، دستور العمل های زیر پیشنهاد می شود:

- همه حس گرهایی را که نرم افزار می خواند، فهرست کنید.
- همه شرایط وقفه را فهرست کنید.
- همه کلیدهایی را که توسط اپراتور فعال می شوند، فهرست کنید.
- همه شرایط داده ها را فهرست کنید.
- با به کار بردن تجزیه اسمی/فعلی که در متن روایی پردازش به کار برده شد، همه ی آیتم های کنترلی را به عنوان ورودی ها/ خروجی های ممکن برای تعیین مشخصات کنترلی مرور کنید.
- رفتار سیستم را با شناسایی حالت ها، شناسایی چگونگی رسیدن به هر حالت و تعیین گذارهای میان حالت ها توصیف کنید.
- جا افتادگی های ممکن را کانون توجه قرار دهید.

تعدادی از ایتم های کنترلی و رویدادی نرم افزار SafeHome
events and control items

- **sensor event** (a sensor has been tripped)
- **blink flag** (a signal to blink the display)
- **start/stop switch** (a signal to turn the system on or off)

مشخصات کنترل

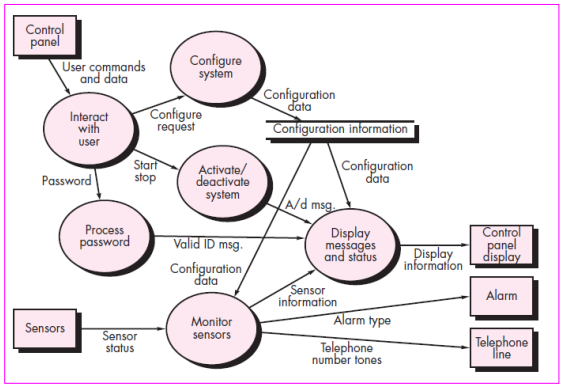
The Control Specification(CSPEC)

مشخصات کنترل (CSPEC) رفتار سیستم را به دو شیءوه به نمایش می گذارد.

CSPEC حاوی یک نمودار حالت است که رفتار را به صورت ترتیبی مشخص می کند.

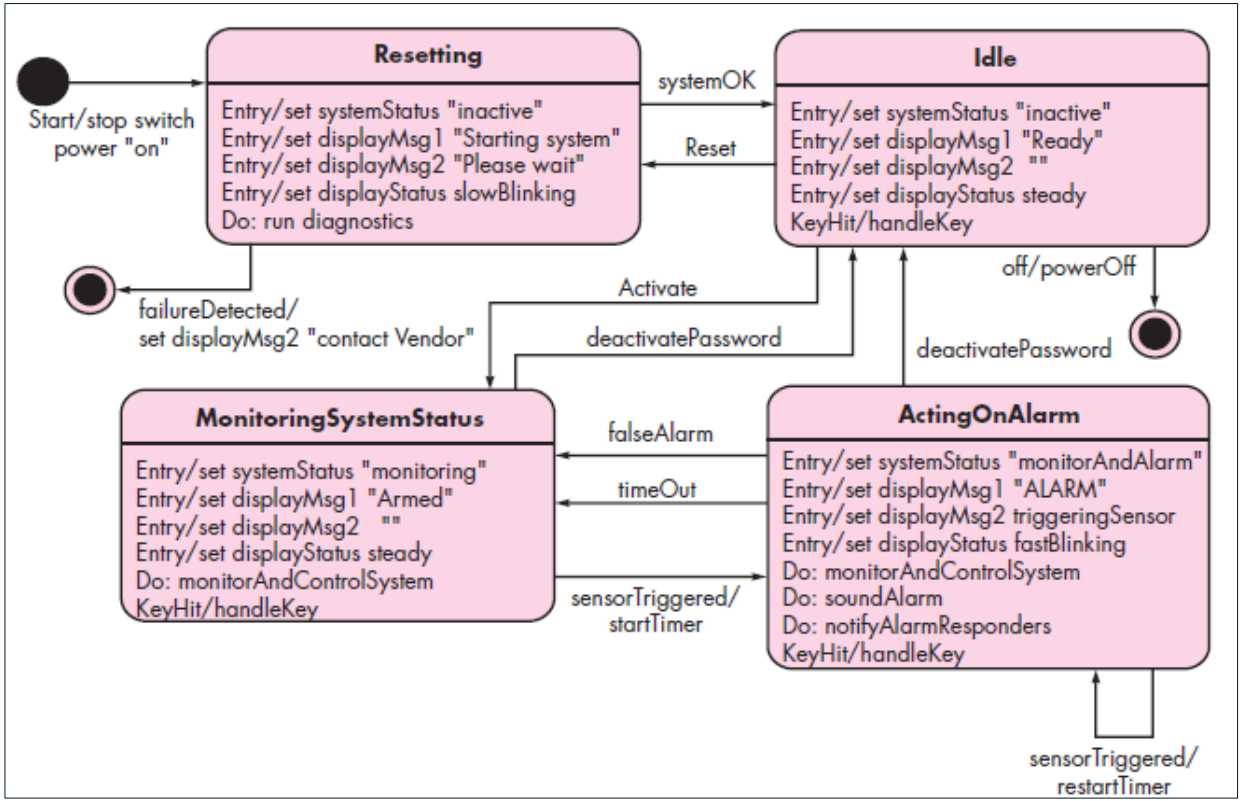
این مشخصات همچنین حاوی یک جدول فعال سازی برنامه ها است.

CSPEC رفتار سیستم را توصیف می کند ولی درباره کارکرد داخلی فرایندهایی که در نتیجه این رفتار فعال می شوند، هیچ اطلاعاتی نمی دهد.



نمودار حالت مقدماتی

preliminary state diagram



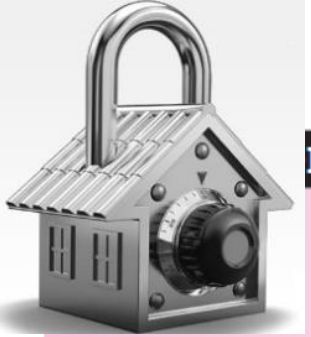
Process activation table

جدول فعال سازی فرایندها(PAT)

یک شیوه نسبتا متفاوت برای نمایش رفتار است.PAT اطلاعات موجود در نمودار حالت را در حیطه ی فرایندها و نه حالت ها، نشان می دهد. این جدول نشان می دهد که کدام فرایندها (حباب ها) در مدل جریان، هنگامی فراخوانی می شود که رویدادی رخ دهد.

input events						
sensor event	0	0	0	0	1	0
blink flag	0	0	1	1	0	0
start stop switch	0	1	0	0	0	0
display action status complete	0	0	0	1	0	0
in-progress	0	0	1	0	0	0
time out	0	0	0	0	0	1
output						
alarm signal	0	0	0	0	1	0
process activation						
monitor and control system	0	1	0	0	1	1
activate/deactivate system	0	1	0	0	0	0
display messages and status	1	0	1	1	1	1
interact with user	1	0	0	1	0	1

مدل سازی جریان داده ها

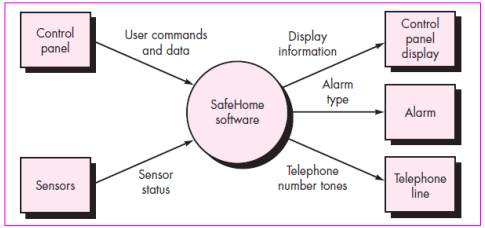


ME

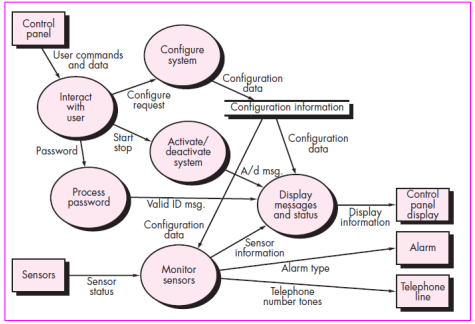
Data Flow Modeling

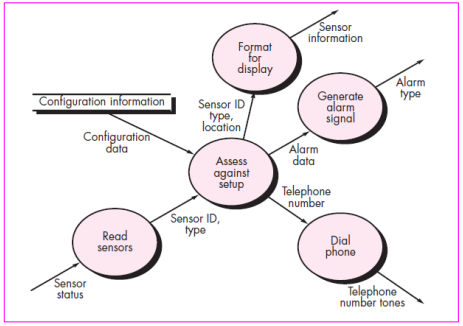
The scene: Jamie's cubicle, after the last requirements gathering meeting has concluded.

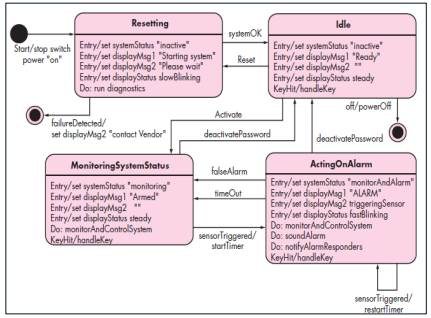
The players: Jamie, Vinod, and Ed—all members of the *SafeHome* software engineering team.



Input-process-output(IOP)







input events						
sensor event	0	0	0	0	1	0
blink flag	0	0	1	1	0	0
start stop switch	0	1	0	0	0	0
display action status complete	0	0	0	1	0	0
in-progress	0	0	1	0	0	0
time out	0	0	0	0	0	1
output						
alarm signal	0	0	0	0	1	0
process activation						
monitor and control system	0	1	0	0	1	1
activate/deactivate system	0	1	0	0	0	0
display messages and status	1	0	1	1	1	1
interact with user	1	0	0	1	0	1

مشخصات فرایندها

The Process Specification

مشخصات فرایندها (PSPEC) در توصیف همه ی فرایندهای مدل جریان که در سطح نهایی پالایش ظاهر می شوند، کاربرد دارد.

محتوای تعیین مشخصات فرایندها می تواند شامل متن روایی، توصیفی از زبان طراحی برنامه(PDL) برای الگوریتم فرایند، معادلات ریاضی،جداول، یا نمودارهای فعالیت های UML باشد.

program design language(PDL)

PSPEC ریز مشخصاتی برای هر تبدیل در پایین ترین سطح پالایش در DFD است.

مثالی از کاربرد PSPEC

PSPEC: process password (at control panel).

The *process password* transform performs password validation at the control panel for the *SafeHome* security function.

Process password receives a four-digit password from the *interact with user* function.

The password is first compared to the master password stored within the system.

If the master password matches, <valid id message = true> is passed to the *message and status display* function.

If the master password does not match, the four digits are compared to a table of secondary passwords (these may be assigned to house guests and/or workers who require entry to the home when the owner is not present).

If the password matches an entry within the table,<valid id message = true> is passed to the *message and status display function*.

If there is no match, <valid id message = false> is passed to the message and status display function.

ایجاد مدل رفتاری

Creating a Behavioral Model

این مدل سازی به رفتار پویای سیستم می پردازد.

مدل های رفتاری نشان می دهند که نرم افزار چگونه به رویدادها یا محرک های خارجی پاسخ می دهند.

برای ایجاد این مدل باید مراحل زیر را اجرا کنید:

۱- ارزیابی همه موارد برای درک کامل تعامل های داخل سیستم

۲- شناسایی رویدادهایی که این تعامل ها را اداره می کنند و درک چگونگی ارتباط این رویدادها با اشیاءی مشخص

۳- ایجاد یک دنباله یا توالی برای هر use case

۴- ساخت یک نمودار حالت برای سیستم

۵- مرور مدل رفتاری برای نشان دادن درستی و سازگاری

شناسایی رویدادها به کمک use case

Identifying Events with the Use Case

Use caseدنباله ای از فعالیت ها را نشان می دهد که کنش گر و سیستم را شامل می شوند.

هر گاه که سیستم و کنش گری به تبادل اطلاعات بپردازند یک رویداد رخ می دهد.

The homeowner uses the keypad to key in a four-digit password.

The password is compared with the valid password stored in the system.

If the password is incorrect, the control panel will beep once and reset itself for additional input.

If the password is correct, the control panel awaits further action.

بخش هایی از use case که زیر آن ها خط کشیءده شده است رویداد ها را نشان می دهند.

پس از شناسایی رویدادها باید آنها را به اشیاءی موجود تخصیص دهید.

اشیاء می توانند مسئول ایجاد رویدادها باشند یا رویدادهایی را که در جای دیگر رخ می دهند شناسایی کنند.

The homeowner uses the keypad to key in a four-digit password.

The password is compared with the valid password stored in the system.

If the password is incorrect, the control panel will beep once and reset itself for additional input.

If the password is correct, the control panel awaits further action.

State Representations

نمایش حالت ها

در حیطه مدل سازی رفتاری، حالت ها را باید از دو نظر مشخص کرد:

۱- حالت هر کلاس در زمانی که به وظیفه خود عمل می کند.
۲- حالت سیستم از دید ناظر خارجی در زمانی که به وظیفه خود عمل می کند.

حالت یک کلاس هر دو خصوصیت انفعالی passive و فعال active را به خود می گیرد.

حالت انفعالی صرفاً حالت فعلی همه صفت های یک شیء است.

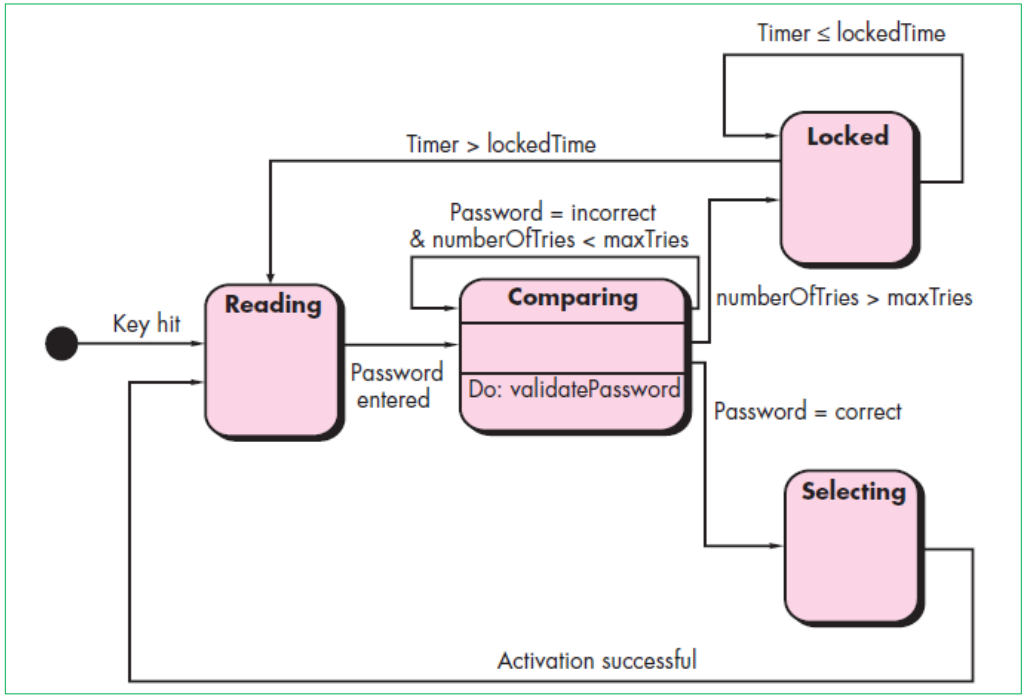
حالت فعال شیء وضعیت فعلی شیء را به هنگام قرار گرفتن در معرض تبدیل یا پردازش مستمر نشان می دهد.

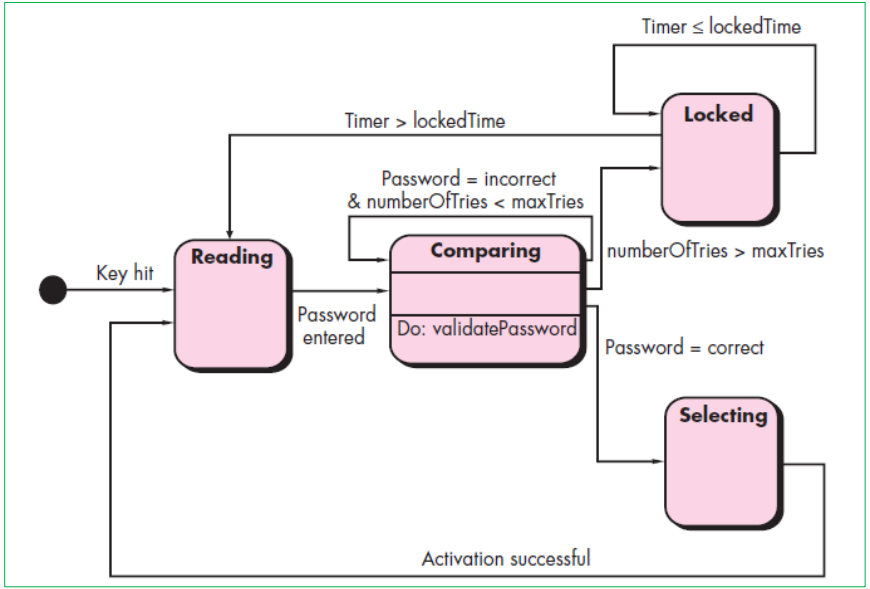
سیستم حالت هایی دارد که رفتار خاص قابل مشاهده از بیرون را به نمایش می گذارد؛ کلاس دارای حالت هایی است که رفتار آن را به هنگام اجرای وظایف به نمایش می گذارد.

نمودار حالت برای کلاس های تحلیل

یک مؤلفه از مدل رفتاری، نمودار حالت UML است که حالت های فعال هر کلاس و رویدادهایی را نشان می دهد که باعث تغییر در این حالت های فعال می شوند.

نمودار حالت برای شیء control panel در عملکرد امنیت SafeHome :





هر کدام از پیکان ها گذاری از یک حالت فعال شیء به حالت فعال دیگر را نشان می دهد.

برچسب های روی هر پیکان رویدادی را نشان می دهند که گذار (transition) را آغاز می کنند.

می توانید یک نگهبان و کنش مشخص کنید.

نگهبان یک شرط بولی است که باید برآورده شود تا گذار رخ دهد.

برای مثال نگهبان گذار از حالت reading به حالت comparing را می توان با بررسی use case تعیین کرد:

If (password input = 4 digits) then compare to stored password

نگهبان مربوط به یک گذار معمولاً به مقدار یک یا چند صفت از شیء بستگی دارد، یعنی نگهبان به حالت انفعالی شیء بستگی دارد.

نمودارهای ترتیب

Sequence diagrams

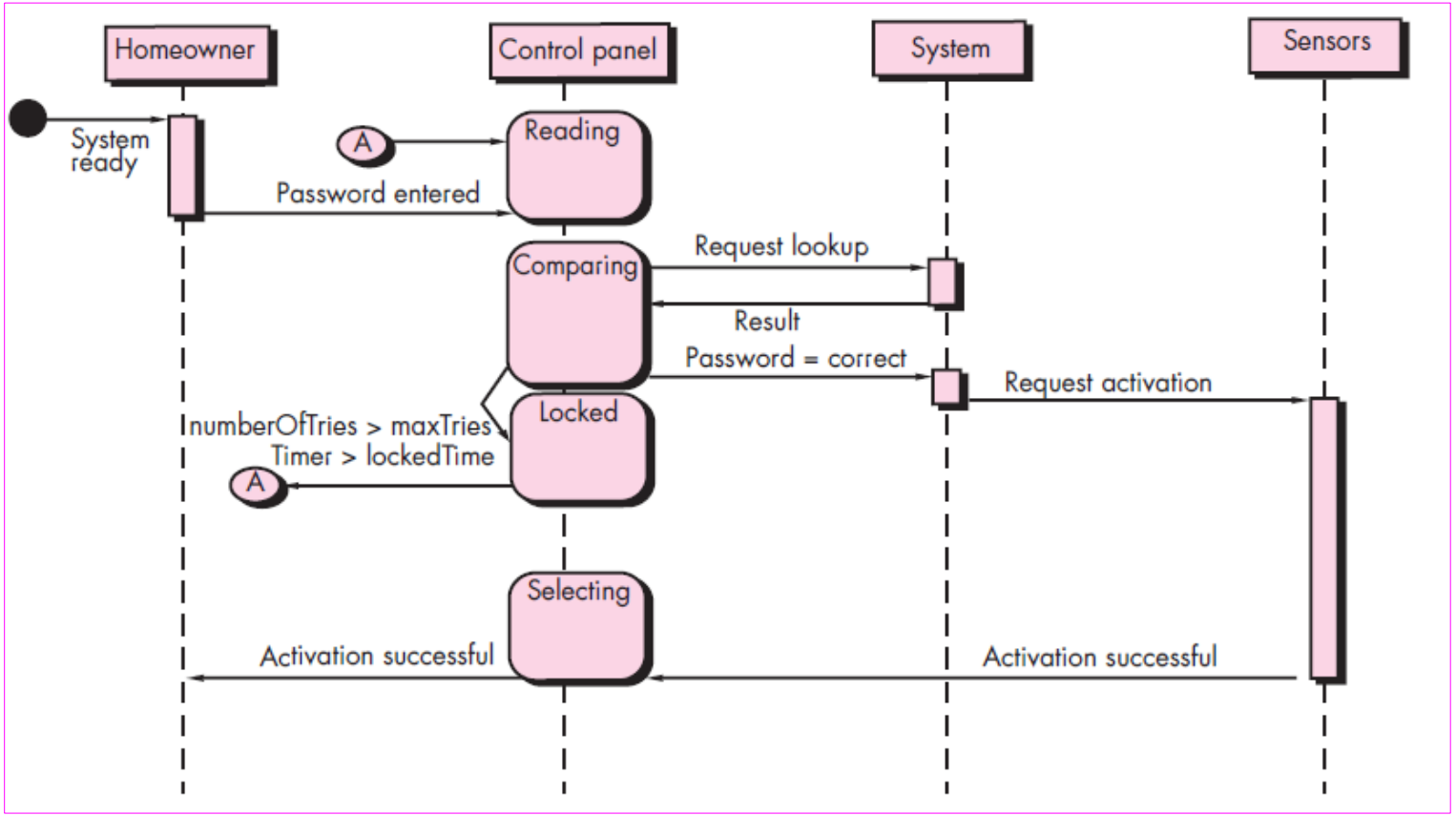
یک روش نمایش رفتار که در UML نمودار ترتیب نامیده می شود، نشان می دهد که رویدادها چگونه باعث گذار یک شیء به شیء دیگر به عنوان تابعی از زمان می شوند.

نسخه خلاصه شده از use case است.

این نمودار کلاس های کلیدی و رویدادهایی را نشان می دهد که باعث جریان یافتن رفتار از کلاسی به کلاس دیگر می شوند.

برخلاف نمودار حالت که رفتار را بدون توجه به کلاس های موجود نشان می دهد، نمودار ترتیب رفتار را با توصیف چگونگی حرکت کلاس ها از حالتی به حالت دیگر به نمایش می گذارد.

بخشی از نمودار ترتیب برای عملکرد امنیت



هر کدام از پیکان ها نشان گر یک رویداد به دست آمده از use case بوده چگونگی کانال زدن رویداد بین اشیاء برای جریان یافتن رفتار را نشان می دهد.
زمان به صورت عمودی (بالا به پایین) سنجیده می شود و مستطیل های باریک عمودی زمان صرف شده در پردازش یک فعالیت را نشان می دهد.

پس از اتمام نمودار ترتیبی، همه رویدادهایی را که باعث گذار میان اشیاءی سیستم می شوند، می توان در مجموعه ای از رویدادهای ورودی و رویدادهای خروجی جمع آوری کرد.

الگوهایی برای مدل سازی خواسته ها

Patterns for Requirements Modeling

الگوهای نرم افزاری، ساز و کارهایی هستند برای آگاهی از دامنه به شیوه ای که بتوان هنگام مواجهه با مسأله ای جدید، آن ها را دوباره به کار برد.

نویسنده ی اصلی یک الگوی تحلیل ، الگو را ایجاد نمی کند، بلکه آن را به موازات اجرای کار مهندسی خواسته ها کشف می کند.

الگوهای تحلیل راهکاری را نشان می دهد که غالبا شامل یک کلاس، یک عملکرد یا رفتار در داخل دامنه کاربردی است.

الگوهای تحلیل در یک مخزن ذخیره می شوند.

کشف الگوهای تحلیل

Discovering Analysis Patterns

مدل خواسته ها از گستره وسیعی از عناصر تشکیل می شود:

مبتنی بر سناریو(Use case)، داده محور (مدل داده ها)، مبتنی بر کلاس، جریان گرا و رفتاری.

Use case اصلی ترین عنصر در توصیف مدل خواسته هاست.

مجموعه ای یکپارچه ازUse case می تواند به عنوان مبنا و اساسی برای کشف یک یا چند الگوی تحلیل عمل کند.

الگوی تحلیل معنا شناختی (Semantic Analysis Pattern)

الگویی است که مجموعه ی کوچکی از Use case یکپارچه را توصیف می کند که به همراه یکدیگر، یک کاربرد کلی و پایه ای را توصیف می کنند.

پایش و کنترل حسگر و محرک در یک سیستم فیزیکی به کمک نرم افزار:

Use case: ***Monitor reverse motion***

Description: When the vehicle is placed in *reverse* gear, the control software enables a video feed from a rear-placed video camera to the dashboard display.

The control software superimposes a variety of distance and orientation lines on the dashboard display so that the vehicle operator can maintain orientation as the vehicle moves in reverse.

The control software also monitors a proximity sensor to determine whether an object is inside 10 feet of the rear of the vehicle.

It will automatically break the vehicle if the proximity sensor indicates an object within *x* feet of the rear of the vehicle, where *x* is determined based on the speed of the vehicle.

نرم افزار در بسیاری از دامنه های کاربرد متفاوت برای پایش حس گرها و کنترل محرک های فیزیکی ضروری است.

بنابراین یک الگوی تحلیل که خواسته های کلی را برای این توانایی توصیف کند، به طور گسترده قابل استفاده خواهد بود.

مثالی از الگوی خواسته ها

نام الگو : Actuator-Sensor

هدف

انگیزه

قید وبندها

قابلیت استفاده

ساختار

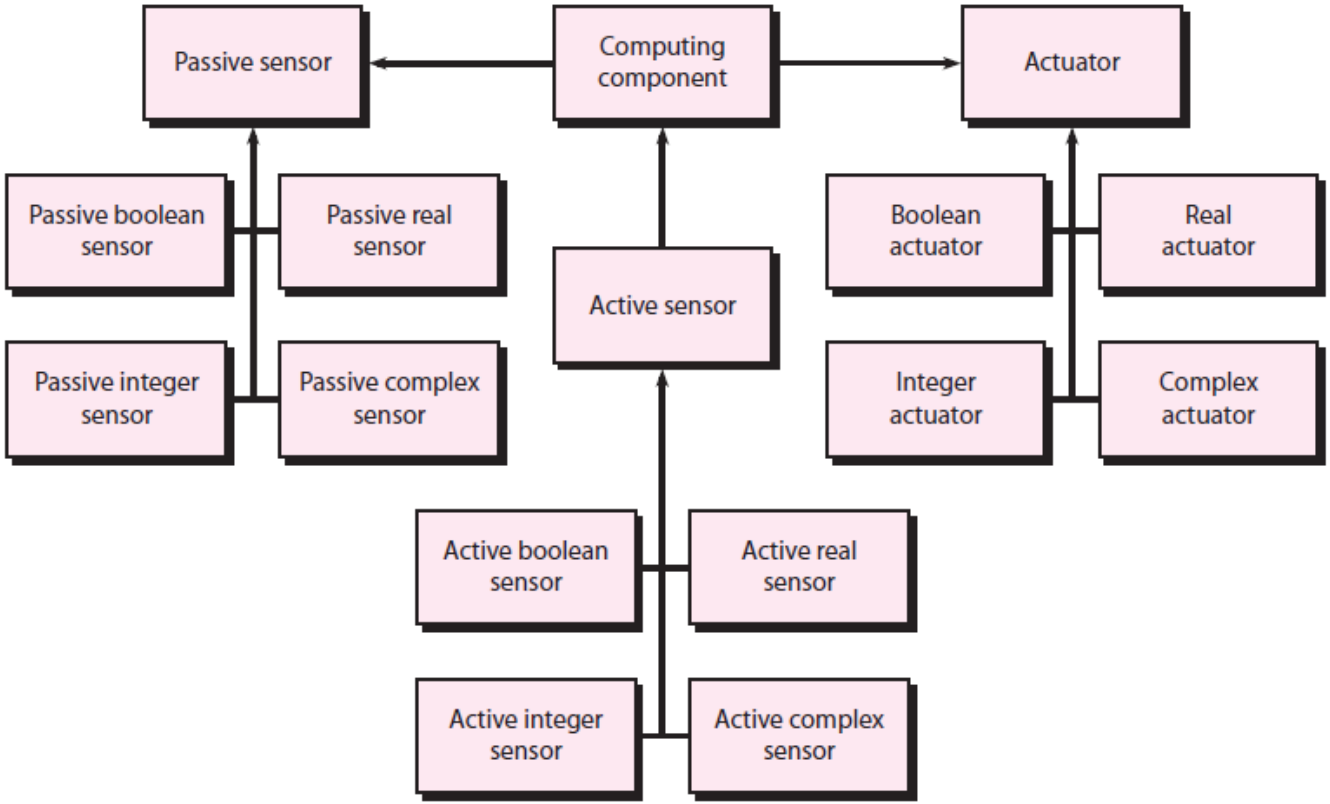
رفتار

مشارکت کنندگان

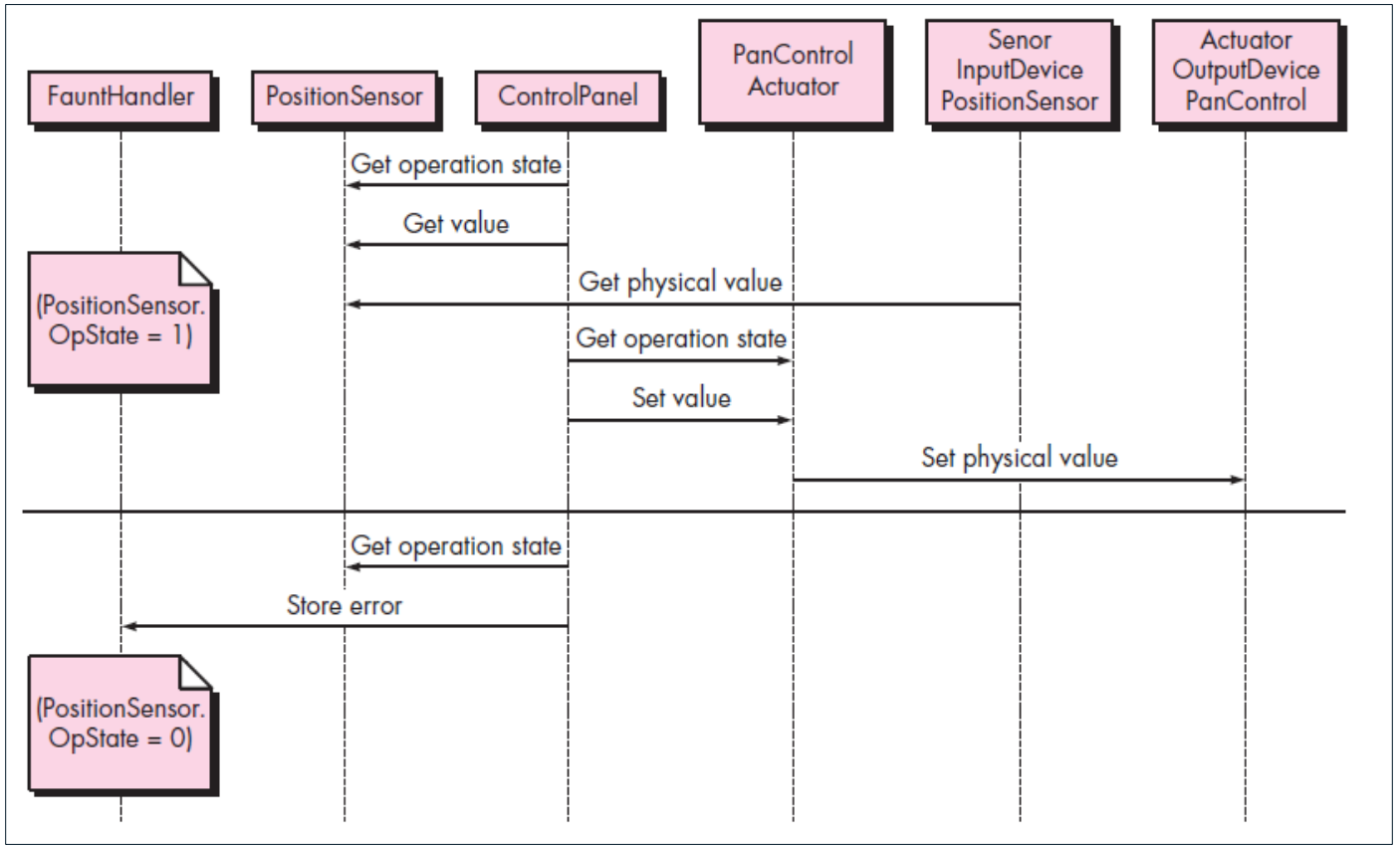
همکاری ها

پیامدها

نمودار ترتیب برای الگوی Actuator-Sensor



نمودار کلاس های UML برای الگوی Actuator-Sensor



پایان فصل ۷

مهندسی نرم افزار

فصل هشتم : مفاهیم طراحی

DESIGN CONCEPTS

مدرس:

فرشید شیرافکن

دانشجوی **دکتری دانشگاه تهران**

(کارشناسی و کارشناسی ارشد : کامپیوتر نرم افزار) (دکتری: بیو انفورماتیک)

نگاهی گذرا

طراحی چیست؟

طراحی، نمایش یا مدلی از نرم افزار ایجاد می کند.

مدل طراحی جزئیات مربوط به معماری نرم افزار ، ساختمان داده ها ، واسط ها و مولفه های لازم برای پیاده سازی سیستم را فراهم می کند.

موارد کانون توجه مدل خواسته ها : توصیف داده ها ، قابلیت ها و رفتار مورد نیاز

چه کسی آن را انجام می دهد؟

مهندس نرم افزار

چرا اهمیت دارد؟

طراحی به شما این امکان را می دهد تا سیستم یا محصولی را که قرار است ساخته شود، مدل سازی کنید.

طراحی جایی است که در آن **کیفیت** نرم افزار تثبیت می شود.

مراحل کار کدام است؟

- معماری سیستم یا محصول باید نمایش داده شود.
- واسط هایی که نرم افزار را به کاربران نهایی و همچنین به مولفه های سازنده خودش مرتبط می سازند ، مدل سازی می شوند.
- مولفه های نرم افزار که در ساخت سیستم به کار می روند ، مدل سازی می شوند.

محصول کار چیست ؟

یک مدل طراحی که شامل نمایش هایی از معماری و واسط ، نمایش در سطح مولفه ها و نمایش های استقرار می شود .

چگونه اطمینان حاصل کنم که درست از عهده کار بر آمده ام؟

طراحی نرم افزار

طراحی برزخ میان دو دنیا می باشد.

هدف طراحی ، ایجاد مدل یا نمایشی است که استحکام ، تناسب و لذت از خود نشان دهد.

برای رسیدن به این مقصود ، باید تنوع و سپس همگرایی را عملی سازید.

طراحی در حیطه مهندسی نرم افزار

Design within the Context of Software Engineering

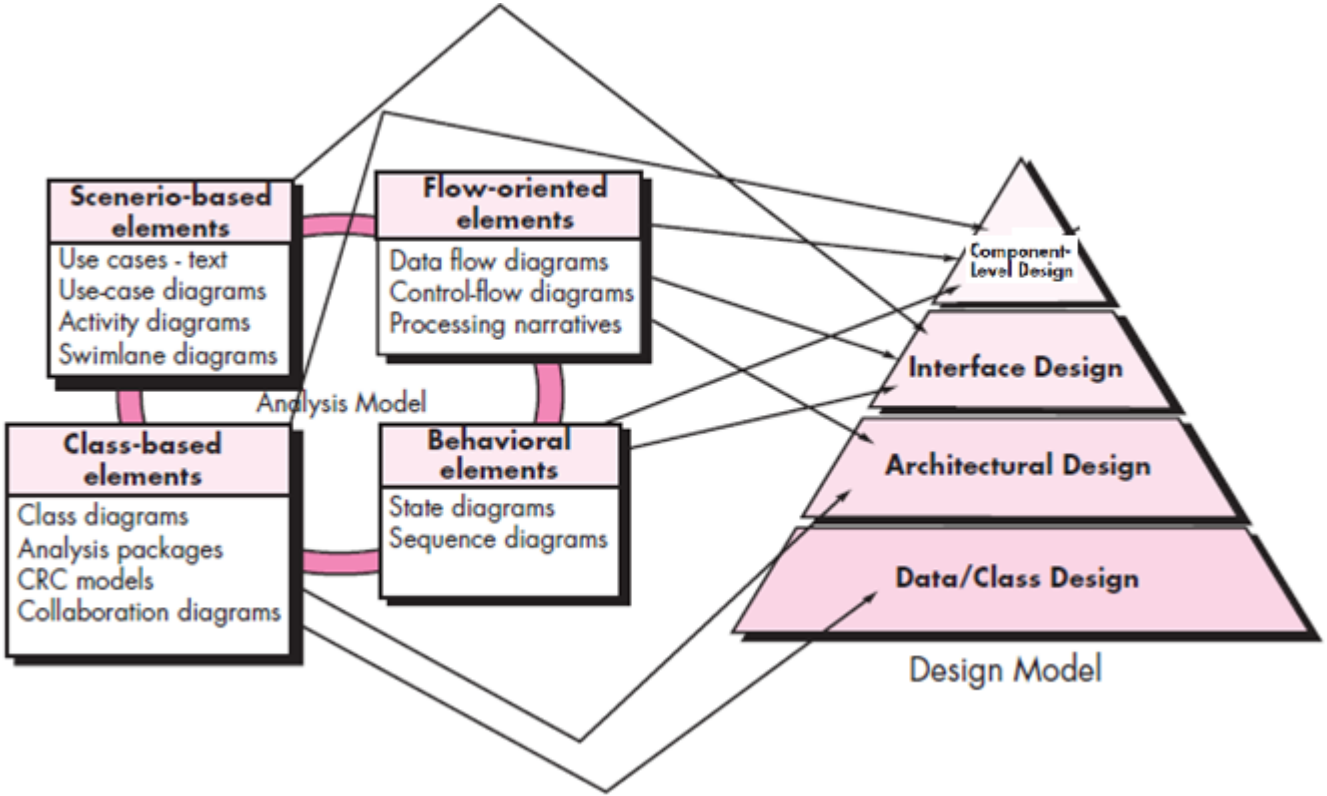
طراحی نرم افزار هسته اصلی نرم افزار را تشکیل می دهد و به کار گیری آن مستقل از نوع مدل فرآیند نرم افزار مورد استفاده است.

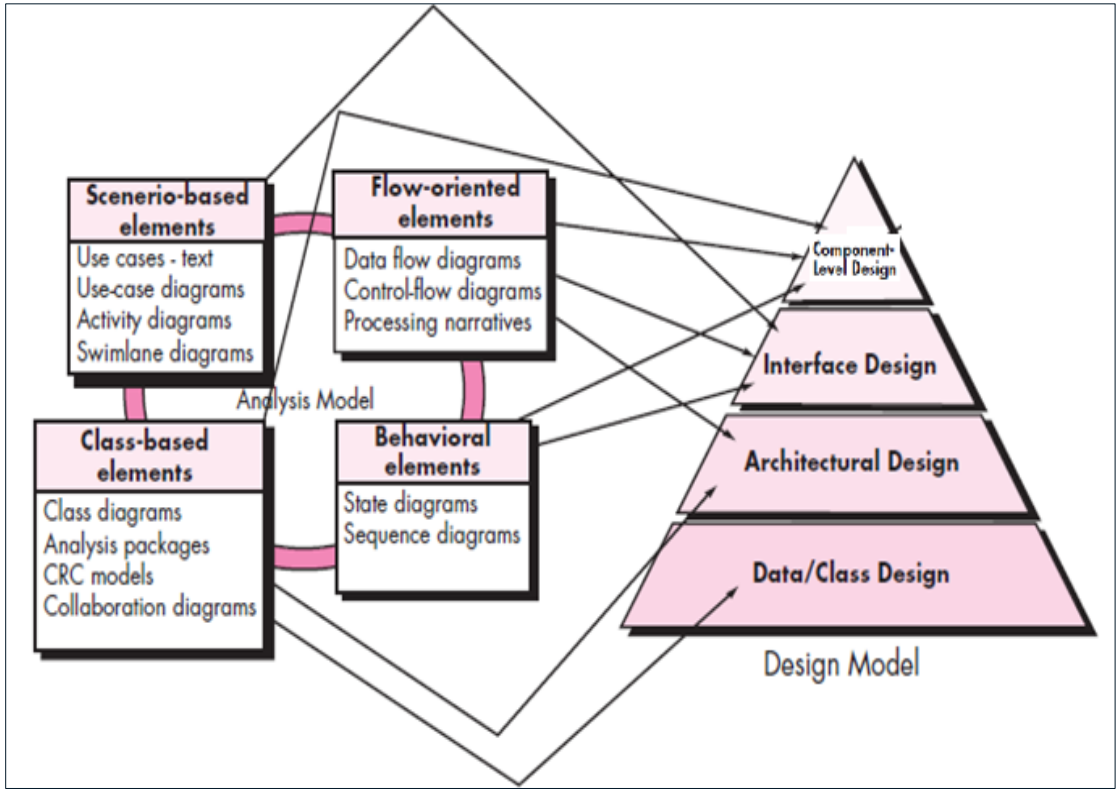
طراحی نرم افزار :

- پس از تحلیل و مدل سازی خواسته ها آغاز می شود.
- آخرین کنش مهندسی نرم افزار در فعالیت مدل سازی است.
- صحنه را برای ساخت (تولید و آزمایش کد) آماده می کند.

برگردان مدل خواسته ها به مدل طراحی

جریان اطلاعات طی طراحی نرم افزار





طراحی داده ها / کلاس ها:

مدل های کلاس ها به کلاس های طراحی و ساختمان داده های لازم برای پیاده سازی نرم افزار تبدیل می شوند.

طراحی معماری:

رابطه ی میان عناصر ساختار اصلی نرم افزار ، سبک های معماری و الگوهای معماری تعریف می شود.

طراحی واسط ها:

چگونگی برقراری ارتباط نرم افزار با سیستم هایی که با آن همکاری متقابل دارد و با افرادی که از آنها استفاده می کنند ، توصیف می شود.

طراحی در سطح مولفه ها:

عناصر ساختاری معماری نرم افزار را به توصیف روالی از مولفه های نرم افزار تبدیل می کند.

اطلاعات بدست آمده از مدل های مبتنی بر کلاس ها، مدل های جریان و مدل های رفتاری به عنوان مبنایی برای طراحی مولفه ها عمل می کنند.

طراحی در مقایسه با کد نویسی

SAFEHOME

Design versus Coding

The scene: Jamie's cubicle, as the team prepares to translate requirements into design.

The players: Jamie, Vinod, and Ed—all members of the *SafeHome* software engineering team.

فرآیند طراحی

The Design Process

طراحی نرم افزار، فرآیندی مبتنی بر تکرار است که از طریق آن خواسته ها به نقشه ای برای ساخت نرم افزار ترجمه می شوند.

صفات و دستور العمل های کیفیت نرم افزار

Software Quality Guidelines and Attributes

سه خصوصیت که به عنوان راهنمایی برای تکامل طراحی خوب به شمار می روند:

۱- طراحی باید همه ی خواسته های صریح موجود در مدل خواسته ها را پیاده سازی کند و باید همه ی خواسته های ضمنی مطلوب طرف های ذینفع را پاسخ گو باشد.

۲- طراحی باید یک راهنمای خوانا و قابل فهم برای کسانی باشد که کدها را تولید می کنند و برای کسانی که نرم افزار را آزمایش و بعداً پشتیبانی می کنند.

۳- طراحی باید تصویر کاملی از نرم افزار باشد که دامنه های داده ای ، عملیاتی و رفتاری را از دیدگاه پیاده سازی به نمایش بگذارد.

دستور العمل های کیفیتی

Quality Guidelines

۱ - طراحی باید معماری را نشان دهد که

(۱) با استفاده از سبک ها یا الگوهای معماری شناخته شده ایجاد شده باشند.

(۲) از مولفه هایی تشکیل شده باشد که خصوصیات طراحی خوبی از خود به نمایش بگذارند.

(۳) به شیوه ای تکاملی قابل پیاده سازی باشند تا به این ترتیب ، پیاده سازی و آزمایش تسهیل گردد.

۲- طراحی باید پیمانه بندی شده باشد ؛ یعنی نرم افزار باید به طرزی منطقی به عناصر یا زیر سیستم هایش افراز شده باشد.

۳- طراحی باید حاوی نمایش های متمایزی از داده ها ، معماری ، واسط ها و مولفه ها باشد.

۴- طراحی باید به ساختمان داده ای منجر گردد که برای کلاس هایی که قرار است پیاده سازی شوندو از الگوهای داده ای قابل تشخیص بیرون کشیده می شوند ، مناسب باشد.

۵- نتیجه طراحی باید مولفه هایی باشد که از خود ، خصوصیات عملیاتی مستقل به نمایش بگذارند.

۶- طراحی باید به واسط هایی بیانجامد که پیچیدگی ارتباطات میان مولفه ها و ارتباط آنها با محیط خارجی را کاهش دهند.

۷- طراحی باید با استفاده از روشی تکرارپذیر به دست آید که خود با اطلاعات حاصل از تحلیل خواسته های نرم افزار به دست می آید.

۸- طراحی باید با به کارگیری نمادهایی ارائه شود که معنا و مفهوم را به خوبی برساند .

Quality Attributes

صفات کیفیتی

صفات کیفیتی FURPS :

- **قابلیت عملیاتی (Functionality)**

با تعیین مجموعه ویژگی ها و قابلیت های برنامه ، عملیات کلی که تحویل می شوند و امنیت کل سیستم ارزیابی می شود.

- **قابلیت کاربرد (Usability)**

با اندازه گیری عوامل انسانی ، زیبایی شناسی کلی ، سازگاری و مستندسازی سنجیده می شود .

- **قابلیت اطمینان (Reliability)**

با اندازه گیری فراوانی و شدت شکست ها ، صحت نتایج خروجی ، میانگین زمان شکست(MTTF) ، توانایی خلاصی یافتن از شکست و قابلیت پیش بینی برنامه تعیین می شود.

- **کارایی (Performance)**

- **قابلیت پشتیبانی (Supportability)**

ترکیبی است از قابلیت نگهداری (بسط پذیری ، قابلیت انطباق و قابلیت سرویس)، قابلیت آزمایش ، سازگاری ، قابلیت پیکربندی ، سهولت نصب سیستم و سهولت پیدا کردن مسائل.

همه صفات کیفیتی نرم افزار به یک میزان اهمیت ندارند و در کاربردهای مختلف اهمیت متغیری دارند.

بنابراین ، صفات کیفیتی را باید همان زمان که طراحی شروع می شود ، مورد توجه قرار داد نه پس از کامل شدن طراحی و شروع ساخت.

تکامل طراحی نرم افزار

The Evolution of Software Design

تکامل طراحی نرم افزار، فرآیندی پیوسته است که اکنون نزدیک به شش دهه را پشت سر گذاشته است.

Early design work concentrated on criteria for the development of modular programs and methods for refining software structures in a top- down manner.
Procedural aspects of design definition evolved into a philos- ophy called *structured programming*.
Later work proposed methods for the translation of data flow or data structure into a design definition.
Newer design approaches proposed an object-oriented approach to design derivation.
More recent emphasis in software design has been on software architecture and the design patterns that can be used to implement software architectures and lower levels of design abstractions.
Growing emphasis on aspect-oriented methods ,model-driven development , and test-driven development emphasize techniques for achieving more effective modularity and architectural structure in the designs that are created.

خصوصیاتی مشترک در همه ی روش های طراحی

۱- سازوکاری برای ترجمه مدل خواسته ها به نمایش طراحی

۲- یک نمادگذاری برای نمایش مولفه های عملیاتی و واسط های آنها

۳- ابتکاراتی برای پالایش و افراز

۴- دستوالعمل هایی برای ارزیابی کیفیت

مفاهیم طراحی

مفاهیم بنیادی طراحی نرم افزار ، چارچوب لازم برای درست انجام دادن را فراهم می آورند.

مفاهیم مهم طراحی نرم افزار که هر دو شیوه سنتی و شی ء گرا را برای توسعه نرم افزار شامل می شوند:

- انتزاع(Abstraction)
- معماری(Architecture)
- الگوها (Patterns)
- جداسازی دغدغه ها(Separation of Concerns)
- پیمانه بندی(Modularity)
- پنهان سازی اطلاعات(Information Hiding)
- استقلال عملیاتی(Functional Independence)
- پالایش(Refinement)
- جنبه ها (Aspects)
- بازآرایی (Refactoring)
- مفاهیم طراحی شیء گرا (Object-Oriented Design Concepts)
- کلاس های طراحی(Design Classes)

انتزاع (Abstraction)

هنگامی که راهکاری پیمانه ای را برای مسأله در نظر می گیرید ، سطوح انتزاع متعددی ممکن است پیش آید.

در بالاترین سطح انتزاع ، راهکار در قالب عبارت هایی کلی و با استفاده از زبان محیط مسأله ، بیان می شود.

در سطوح پایین انتزاع، توصیف مشروحی تری از راهکار ارائه می شود.

در پایین ترین سطح از انتزاع ، راهکار به شیوه ای بیان می شود که به طور مستقیم قابل اجرا و پیاده سازی باشد.

انتزاع فرایندی: دستورالعمل هایی است که وظیفه ای مشخص و محدود دارند.

انتزاع داده ای : مجموعه ای از داده ها با نام مشخص است که شی ء داده ای را توصیف می کند.

معماری(Architecture)

معماری نرم افزار به ساختار کلی نرم افزار و شیوه هایی مربوط می شود که این ساختار باعث یکپارچگی مفهومی در سیستم می گردد.

معماری در ساده ترین شکل خود، ساختار یا سازماندهی مولفه های برنامه، شیوه ی تعامل این مولفه ها و ساختمان داده های قابل استفاده توسط این مولفه هاست.

ولی از دیدگاه گسترده تر، مولفه ها را می توان طوری تعمیم بخشید که عناصر اصلی سیستم و تعامل های آنها را نشان دهد.

یکی از اهداف مهندسی نرم افزار، به دست آوردن یک نمای معماری از سیستم است.

مجموعه ای از خواص که خوب است به عنوان بخشی از معماری در نظر گرفته شوند:

- **خواص ساختاری**

در این جنبه از نمایش طراحی معماری ، مولفه های سیستم و شیوه ی بسته بندی و تعامل آنها با یکدیگر تعیین می شود.

- **خواص عملیاتی اضافی**

می بایست چگونگی برآورده شدن خواسته های کارایی ، ظرفیتی ، قابلیت اطمینان ، امنیت ، انطباق پذیری و سایر خصوصیات سیستم در معماری طراحی در نظر گرفته شود.

- **خانواده های سیستم های مرتبط**

در طراحی معماری باید الگوهایی تکرارپذیر بدست آورده شود که به طور متداول در طراحی خانواده هایی از سیستم های مرتبط با آن مواجه می شویم.

با توجه به خواص مذکور ، طراحی معماری را می توان با به کارگیری یک یا چند مدل متفاوت به نمایش درآورد:

در **مدل های ساختاری** ، معماری به صورت مجموعه ای سازمان یافته از مولفه های برنامه نمایش داده می شود.

مدل های چارچوبی با تلاش برای شناسایی چارچوب های طراحی معماری تکرارپذیری که در انواع مشابه کاربردها مشاهده می شوند ، سطح انتزاع را در طراحی بالا می برند.

مدل های پویا به جنبه های رفتاری معماری برنامه می پردازند و چگونگی تغییر پیکربندی سیستم یا ساختار را به عنوان تابعی از رویدادهای خارجی مشخص می کنند.

در **مدل های فرایندی** طراحی فرایند تجاری یا فنی ای که سیستم باید در نظر بگیرد ، کانون توجه قرار می گیرد.

مدل های عملیاتی را می توان برای به نمایش درآوردن سلسله مراتب عملیات ها در یک سیستم به کار برد.

الگوها (Patterns)

الگوی طراحی ، توصیفی است از :

یک ساختار طراحی که یک مسأله طراحی را در حیطه ای خاص حل می کند و

نیروهایی بینابینی که ممکن است بر شیوه به کار گیری و استفاده از این الگو تأثیرگذار باشند.

هدف هر الگوی طراحی، فراهم ساختن توصیفی است که طراح به کمک آن بتواند تعیین کند که:

۱- آیا این الگو برای کار فعلی قابل استفاده است.

۲- آیا این الگو قابل استفاده مجدد است.

۳- آیا این الگو می تواند به عنوان راهنمایی برای توسعه ی یک الگوی مشابه ولی با ساختار و عملکرد متفاوت عمل کند.

جداسازی دغدغه ها

(Separation of Concerns)

دغدغه: ویژگی یا رفتاری که به عنوان بخشی از مدل خواسته ها برای نرم افزار مشخص می شود.

جداسازی دغدغه ها ، یک مفهوم طراحی است که پیشنهاد می کند هر مسأله پیچیده ای را می توان بهتر حل کرد اگر به قطعاتی تقسیم گردد که هر یک را بتوان به طور مستقل ، حل و یا بهینه سازی کرد.

با جداسازی دغدغه ها به قطعات کوچکتر ، زمان و تلاش کمتری صرف حل مسأله می شود.

پیمانه بندی

(Modularity)

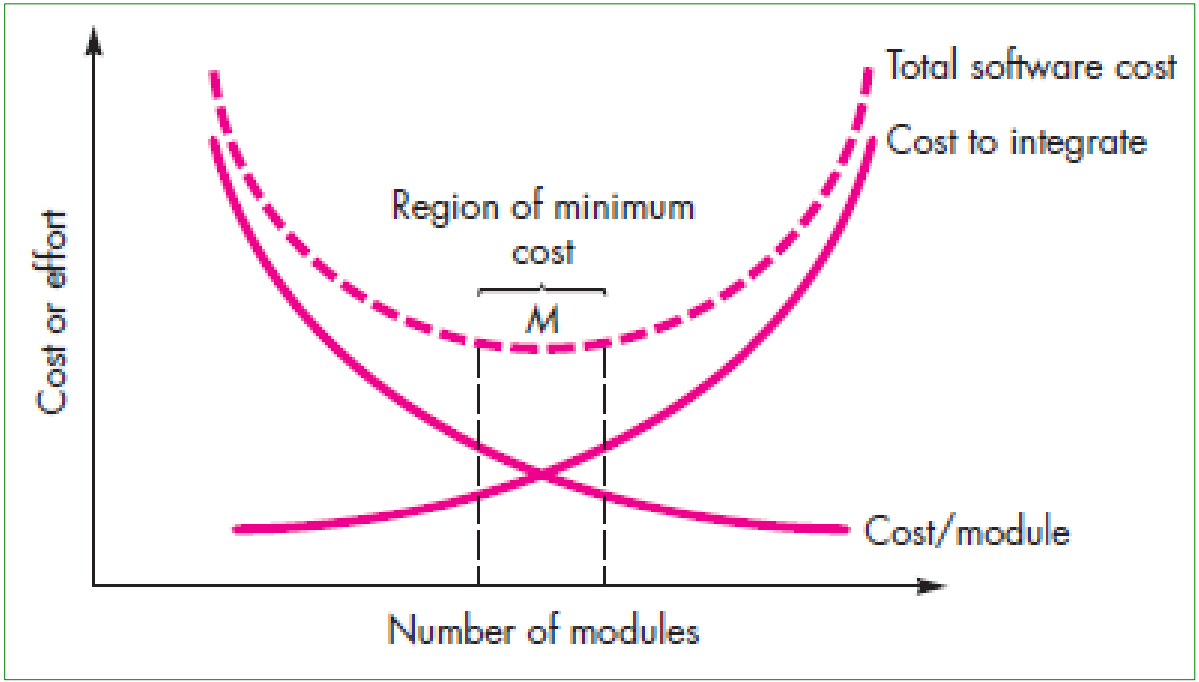
پیمانه بندی، متداول ترین نمود جداسازی دغدغه هاست.

نرم افزار به مولفه های جداگانه و دارای نام تقسیم می شود که گاه از آنها با عنوان پیمانه یاد می شود، از انسجام این پیمانه ها ، خواسته های مسأله برآورده می شود.

پیمانه بندی ، تنها صفت نرم افزار است که اداره ی هوشمندانه ی برنامه را میسر می سازد.

تقریباً در همه ی نمونه ها ، باید طراحی را به چندین پیمانه تقسیم کنید تا درک و فهم مسأله آسانتر شود و در نتیجه هزینه لازم برای ساخت نرم افزار کاهش یابد.

پیمانه بندی و هزینه نرم افزار



شکل نشان می دهد که تلاش (هزینه) لازم برای توسعه یک پیمانه نرم افزار، با افزایش تعداد پیمانه ها کاهش می یابد. اگر مجموعه یکسانی از خواسته ها را در نظر بگیریم، بیشتر بودن تعداد پیمانه ها به معنای کوچک تر شدن اندازه ی هر پیمانه خواهد بود.

ولی با رشد تعداد پیمانه ها، تلاش (هزینه) مرتبط با انسجام بخشیدن به این پیمانه ها نیز رشد می کند.

پنهان سازی اطلاعات (Information Hiding)

پیمانه ها باید طوری مشخص و طراحی شوند که اطلاعات موجود در یک پیمانه نتواند در دسترس پیمانه های دیگری قرار گیرد که به این اطلاعات نیاز ندارند.

این مجموعه پیمانه تنها با پیمانه دیگری ارتباط برقرار می کند که حاوی اطلاعات لازم برای دستیابی به عملکرد نرم افزار است.

استفاده از پنهان کردن اطلاعات به عنوان ملاک طراحی برای سیستم های پیمانه ای ، هنگامی بیشترین مزایا را به همراه خواهد داشت که اصلاحاتی طی آزمایش و سپس طی نگهداری نرم افزار لازم باشد.

چرا که احتمال انتشار خطا های سهوی طی انجام اصلاحات در سایر نقاط نرم افزار ، کمتر می شود.

استقلال عملیاتی

(Functional Independence)

مفهوم استقلال عملیاتی ، نتیجه مستقیم جداسازی دغدغه ها ، پیمانه بندی و مفاهیم پنهان سازی اطلاعات و انتزاع است.

استقلال عملیاتی با توسعه پیمانه هایی با عملکرد یگانه و دوری جستن از تعامل بیش از حد با سایر پیمانه ها بدست می آید.

چرا باید در ایجاد پیمانه های مستقل بکوشید؟

توسعه نرم افزارهایی با پیمانه های مستقل، راحت تر است، چون قابلیت های عملیاتی را می توان به واحدهای کوچک تر تقسیم کرد و واسط ها ساده می شوند.

به طور خلاصه ، استقلال عملیاتی کلید طراحی خوب و طراحی کلید کیفیت نرم افزار است.

استقلال با استفاده از دو ملاک کیفیتی قابل ارزیابی است:

یکپارچگی ، نشان از قدرت عملیاتی نسبی یک پیمانه دارد. اتصال ، نشان از استقلال در میان پیمانه ها دارد.

- **یکپارچگی (cohesion)**

  یکپارچگی بسط طبیعی مفهوم پنهان سازی اطلاعات است.

  یکپارچگی، شاخصی کیفی از میزان تمرکز یک پیمانه بر تنها یک چیز است.

  پیمانه یکپارچه ، به تعامل اندک با سایر مولفه های موجود در بخش های دیگر سیستم نیاز دارد.

- **اتصال (coupling)**

  اتصال شاخصی کیفی از میزان ارتباط یک پیمانه با سایر پیمانه ها و جهان خارج است.

  اتصال به پیچیدگی واسط های میان پیمانه ها، نقطه ای که در آن ورود یا ارجاع به یک پیمانه

  انجام می شود وداده هایی که از واسط عبور می کنند ، بستگی دارد.

پالایش (Refinement)

پالایش مرحله ای، یک راهبرد طراحی از بالا به پایین است.

پالایش در واقع همان فرایند تعیین جزئیات است.

با توصیف عملکرد که در سطح بالایی از انتزاع تعریف می شود، کار خود را شروع می کنید.

سپس جزئیات مربوط به بیان اولیه را تعیین می کنید و با هر بار پالایش پیاپی ، جزئیات بیشتر و بیشتری به آن اضافه می کنید.

پالایش و انتزاع، مفاهیمی مکمل یکدیگرند.

به کمک انتزاع می توانید روال ها و داده ها را از نظر داخلی مشخص کنید، ولی نیاز افراد متفرقه به داشتن آگاهی از جزئیات سطح پایین را برطرف می سازد.

پالایش به شما کمک می کند تا با پیشرفت طراحی، جزئیات طراحی را آشکار کنید.

(Aspects)

جنبه ها

در همان حال که تحلیل خواسته ها رخ می دهد، مجموعه ای از دغدغه ها آشکار می شود.

دغدغه پیش نیاز، خصوصیتی از سیستم است که در میان خواسته های متفاوت کاربرد دارد.

جنبه ، نمایشی از یک دغدغه پیش نیاز است.

شناسایی جنبه ها به طوری که طراحی بتواند به صورت مناسب آنها را ضمن انجام پالایش و پیمانه بندی، در برگیرد، اهمیت دارد.

در حالت ایده آل ، هر جنبه به صورت پیمانه ای جداگانه (مولفه) پیاده سازی می شود نه به صورت تکه هایی از نرم افزار که در سرتاسر چندین مولفه پراکنده و در هم و بر هم شده باشند.

برای دستیابی به این مقصود، معماری طراحی باید از ساز و کاری برای تعریف یک جنبه پشتیبانی کند.

consider two requirements for the SafeHomeAssured.com WebApp.

Requirement *A* is described via the **ACS-DCV** use case.

A design refinement would focus on those modules that would enable a registered user to access video from cameras placed throughout a space.

Requirement *B* is a generic security requirement that states that *a registered user must be validated prior to using* **SafeHomeAssured.com.**

This requirement is applicable for all functions that are available to registered *SafeHome* users.

As design refinement occurs, *A** is a design representation for requirement *A* and *B** is a design representation for requirement *B*.

Therefore, *A** and *B** are representations of concerns, and *B* crosscuts A**.

the design representation, *B**, of the requirement ***a registered user must be validated prior to using*** **SafeHomeAssured.com,** is an **aspect** of the *SafeHome* WebApp.

بازآرایی

(Refactoring)

یک فعالیت مهم طراحی که برای بسیاری از روش های چابک پیشنهاد شده است، بازآرایی است که تکنیکی برای سازمان دهی مجدد به شمار می رود و طراحی (یا کد) یک مولفه را ساده می کند، بدون اینکه قابلیت عملیاتی رفتار آن را تغییر دهد.

بازآرایی : فرایند تغییر دادن سیستم نرم افزاری به گونه ای که رفتار خارجی کد(طراحی) تغییر نکند و در عین حال، ساختار درونی آن بهبود یابد.

هنگامی که نرم افزار بازآرایی می شود، طراحی موجود برای زوائد، عناصر استفاده نشده ی طراحی ، الگوریتم های ناکارآمد یا غیر ضروری ، ساختمان داده ای ضعیف یا نامناسب، یا هر گونه شکست طراحی دیگر که قابل اصلاح باشد، بررسی می شود تا طراحی بهتری به دست آید.

نتیجه، نرم افزاری خواهد بود که راحت تر می توان به آن انسجام بخشید و آزمودن و نگهداری آن آسان تر است.

کلاس های طراحی

(Design Classes)

به موازاتی که مدل طراحی تکامل پیدا می کند، مجموعه ای از کلاس های طراحی را تعریف می کند که کلاس های تحلیل را با فراهم آوردن جزئیات طراحی پالایش می کنند و یک زیر ساخت نرم افزاری را پیاده سازی می کنند که راهکار تجاری را پشتیبانی می کند.

پنج نوع متفاوت از کلاس های طراحی را می توان توسعه داد که هر کدام لایه متفاوتی از معماری طراحی را نشان می دهند:

- **کلاس های واسط کاربری:** تمام انتزاع های لازم برای تعامل میان انسان و کامپیوتر(HCI) را تعریف می کنند.
- **کلاس های دامنه تجاری :** صفات و سرویس ها را مشخص می کنند که برای پیاده سازی عنصری از دامنه تجاری مورد نیازند.
- **کلاس های پردازش:** انتزاع های تجاری سطح پایین لازم برای مدیریت کامل کلاس های دامنه ی تجاری را پیاده سازی می کنند.
- **کلاس های ماندگار:** انباره های داده ها را نشان می دهند که پس از اجرای نرم افزار ، ماندگار می شوند.
- **کلاس های سیستمی:** عملیات های مدیریتی و کنترلی را پیاده سازی می کنند که کارکرد سیستم را میسر می سازند و ارتباط میان درون و بیرون محیط کامپیوتری را برقرار می سازند.

چهار مشخصه برای کلاس طراحی خوش فرم

۱- کامل و کافی (Complete and Sufficient)

طراحی باید پنهان سازی کاملی از همه ی صفات و سندهایی باشد که به طور منطقی برای آن کلاس انتظار می رود.
کافی بودن به آن معناست که کلاس تنها حاوی متدهایی باشد که برای دستیابی به هدف کلاس کفایت می کنند ، نه کمتر و نه بیشتر.

۲- سادگی (Primitiveness)

هنگامی که یک سرویس با یک متد پیاده سازی شد ، کلاس نباید راه دیگری برای دستیابی به همان هدف فراهم سازد.

۳- یکپارچگی بالا (High Cohesion)

یک کلاس طراحی یکپارچه ، دارای مجموعه ای کوچک و متمرکز از مسئولیت هاست که صفات و متدها را برای پیاده سازی همان مسئولیت ها به کار می برد.

۴- اتصال پایین (Low Coupling)

اگر در یک مدل طراحی میزان اتصال بالا باشد (همه ی کلاس های طراحی با همه ی کلاس های طراحی دیگر همکاری کنند) ، پیاده سازی سیستم، آزمایش آن و نگهداری آن در گذر زمان دشوار می شود. به طور کلی ، کلاس های طراحی در داخل یک زیر سیستم فقط باید آگاهی محدودی از سایر کلاس ها داشته باشد. این محدودیت ، که قانون دمتر نامیده می شود ، پیشنهاد می کند که یک متد فقط باید به متدهای موجود در کلاس های همسایه پیام ارسال کند.

مفاهیم طراحی

SAFEHOME

Design Concepts

The scene: Vinod's cubicle, as design modeling begins.

The players: Vinod, Jamie, and Ed—members of the *SafeHome* software engineering team. Also, Shakira, a new member of the team.

عضو ۲

عضو ۱

عضو ۳

عضو جدید

مدل طراحی

(Design Model)

عناصر طراحی داده ها

عناصر طراحی معماری

عناصر طراحی واسط ها

عناصر طراحی در سطح مولفه ها

عناصر طراحی در سطح استقرار

پایان

مشاوره با مدرس شیرافکن : ۰۹۱۲۱۹۷۲۰۲۸