

مقدمه

امروزه نقش کلیدی و حیاتی نرم‌افزار در زندگی روزمره کاملاً محسوس و انکارناپذیر است، به گونه‌ای که متمایز نمودن نرم‌افزارهای رایانه‌ای از زندگی روزمره بشری غیرممکن است. رشد سرسام‌آور فناوری نوین اطلاعات و ارتباطات و تأثیرات آن بر روابط سیاسی، اقتصادی و اجتماعی جوامع انکارناپذیر شده است. نرم‌افزارهایی که بر این بستر فناوری، توسعه می‌یابند دارای حساسیت‌های خاصی هستند به شکلی که شکست و خرابی آن‌ها صدمات اقتصادی جبران‌ناپذیری در جامعه برجا می‌گذارد.

در این اوضاع علم مهندسی نرم‌افزار به برای بهبود فرآیند و توسعه نرم‌افزارهای رایانه‌ای نقش مهمی را ایفا می‌کند. توسعه روش‌ها، ابزارها، فرآیندها و مدل‌های نرم‌افزار گویای این مطلب است که محصول تولید شده کیفیت بالا، قابلیت نگهداری و توسعه را داشته باشد.

خلاصه کتاب حاضر نتیجه تلاش ده سال تدریس دروس مهندسی نرم‌افزار و طراحی سیستم‌های اطلاعاتی در دوره‌های کارشناسی و کارشناسی ارشد مهندسی رایانه در دانشگاه‌های ایران می‌باشد که به صورت فشرده ارائه شده است. این کتاب در بردارنده فرآیندهای اصلی توسعه نرم‌افزار یعنی تعریف، توسعه و نگهداری همراه کلیه فعالیت‌های پشتیبانی نرم‌افزار است.

محتویات کتاب مطابق با سرفصل‌های مصوب ستاد انقلاب فرهنگی در مهندسی نرم‌افزار ۱ و ۲ رشته‌های مهندسی نرم‌افزار، فناوری اطلاعات و علوم رایانه تدوین شده است. لذا این کتاب برای این دسته از دانشجویان در دوره کارشناسی مورد استفاده و مرجعی برای مطالعه دانشجویان کارشناسی و کارشناسی ارشد نرم‌افزار و فناوری اطلاعات می‌باشد. این کتاب راهنمای بسیار جامعی و فشرده برای داوطلبان کنکور کارشناسی ارشد مهندسی فناوری اطلاعات می‌باشد که تمام مباحث به صورت فشرده و خلاصه در آن ارائه شده است.

در اینجا لازم می‌دانم که از دانشجویان مهندسی نرم‌افزار و مهندسی فناوری اطلاعات دانشگاه شهید بهشتی و موسسه آموزش عالی آزاد پارسه که با دقت و باریک بینی مطالب کتاب را مطالعه و نقطه نظرات سازنده در رفع نواقص مطالب ارائه کردند، تشکر و قدردانی نمایم.

اسلام ناظمی

دی ماه ۱۳۸۹

در جدول ذیل دروس به سرفصلهای مهم آن طبقه بندی شده و مشخص شده است که در هر سال از هر مبحث چند تست سوال شده است و دانشجوی محترم می تواند زمان باقیمانده تا کنکور را با توجه به اهمیت مباحث مدیریت نماید.

		رشته: مهندسی فناوری اطلاعات					درس: مهندسی نرم افزار	
نسبت از کل	مجموع ۵ سال	۱۳۸۹	۱۳۸۸	۱۳۸۷	۱۳۸۶	۱۳۸۵	مبحث	ردیف
		تعداد تست	تعداد تست	تعداد تست	تعداد تست	تعداد تست		
3%	1	0	0	1	0	0	ویژگی نرم افزار	1
13%	5	1	1	1	0	2	فرآیند	2
8%	3	1	0	0	1	1	اصول مدیریت پروژه	3
5%	2	1	0	0	0	1	مدیریت و تحلیل ریسک	4
5%	2	0	1	0	1	0	سنجش نرم افزار و متریک های نرم افزار	5
3%	1	0	0	0	0	1	برنامه ریزی پروژه و برآوردها	6
8%	3	0	1	1	1	0	کنترل و نظارت بر پروژه های نرم افزاری	7
8%	3	0	1	0	0	2	مدیریت اطمینان مرغوبیت نرم افزار	8
3%	1	0	1	0	0	0	مدیریت پیکربندی نرم افزار	9
5%	2	1	1	0	0	0	اصول تحلیل ساختیافته	10
8%	3	0	0	1	2	0	مدل سازی تحلیل ساخت یافته	11
5%	2	0	0	1	1	0	اصول طراحی ساخت یافته	12
8%	3	1	0	0	2	0	طراحی معماری	13
0%	0	0	0	0	0	0	طراحی واسط های نرم افزاری	14
0%	0	0	0	0	0	0	طراحی پیمانه ای	15
0%	0	0	0	0	0	0	تحلیل شی گرا	16
5%	2	0	1	1	0	0	طراحی شیئی گرا	17
3%	1	0	0	0	0	1	آزمون نرم افزار	18
13%	5	0	1	2	0	2	مفهوم کلی مربوط به مهندسی نرم افزار	19
100%	39	5	8	8	8	10	جمع	

فصل اول

محصول (نرم افزار)

نرم افزار تبدیل به عنصر کلیدی در تکامل محصولات و سیستم های مبتنی بر رایانه شده است. در ۶۰ سال گذشته، نرم افزار از یک ابزار تحلیل اطلاعات و حل مسئله، به صنعتی مستقل تکامل یافته است. عملکرد گذشته در توسعه نرم افزارهای کاربردی، یک سری مشکلات ایجاد کرده است که هنوز هم آثار آن قابل دیدن است.

نرم افزار به عاملی محدودکننده در ادامه تکامل سیستم های رایانه ای تبدیل شده است. هر نرم افزار شامل مجموعه ای از برنامه ها، داده ها و مستندات است. هر یک از این اجزا در بردارنده پیگیری خاص خود هستند که به عنوان بخشی از فرایند مهندسی نرم افزار به وجود می آیند. هدف مهندسی نرم افزار فراهم آوردن چارچوبی برای توسعه نرم افزارهایی است که دارای کیفیت مطلوب و بالایی باشند.

۱-۱ تعریف مهندسی نرم افزار

از دیدگاه Pressman: به مجموعه ای از فن آوری ها، روش ها و ابزارهای مبتنی بر اصول مهندسی که در توسعه نرم افزار استفاده می شود، مهندسی نرم افزار گفته می شود.

از دیدگاه Sommerville: مهندسی نرم افزار در دهه ۱۹۶۰ در کنفرانسی با نام بحران نرم افزار مطرح می گردد. مهندسی نرم افزار در ارتباط با توسعه سیستم های نرم افزاری به شکل فعالیت گروهی است و از قواعد مهندسی استفاده می کند و شامل جنبه های فنی و غیر فنی است.

از دیدگاه Bauer: مهندسی نرم افزار عبارت از توسعه و استقرار نرم افزار با استفاده از اصول دقیق مهندسی برای دستیابی به نرم افزاری با صرفه اقتصادی است دو شرط قابل اعتماد^۱ و کارا^۲ را دارد.

توجه: مهندسی نرم افزار هم ارز با برنامه نویسی نیست، بلکه برنامه نویسی فقط جزئی از مهندسی نرم افزار است.

۲-۱ مشکلات کنونی نرم افزار

- عدم تطابق نرم افزار با نیازهای واقعی مشتری و بازار (تغییرات)
- عدم امکان رقابت و استفاده کارای نرم افزار از سخت افزار با توجه به پیشرفت سریع سخت افزار (فن آوری)
- عمومی شدن کاربرد رایانه در جامعه و عدم امکان تغییر و تعویض نرم افزار در مدت زمان کوتاه (کاربرد لحظه‌ای)
- فشار عمومی و زیاد برای تولید نرم افزاری قابل اعتماد و دارای کیفیت مناسب (آزمون نرم افزار)
- قدرت محدود در پشتیبانی و نگهداری نرم افزارهای موجود (قابلیت توسعه)

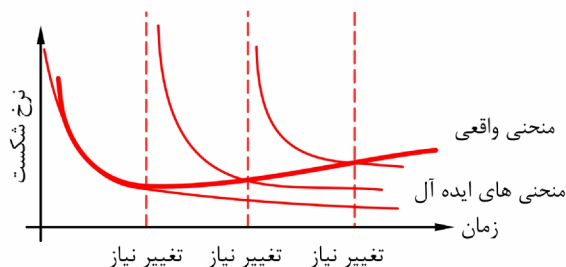
۳-۱ تکامل نرم افزار و طرح سؤال در برابر مشکلات

در حال حاضر برنامه‌نویس گذشته جای خود را به تیمی از کارشناسان و مهندسين نرم افزار داده است که هر یک در بخشی از فن آوری لازم برای تولید یک برنامه کاربردی پیچیده فعالیت می‌کنند. با این وجود هنگام توسعه سیستم‌های رایانه مدرن پرسش‌های قبلی به صورت زیر همچنان مطرح هستند:

- چرا توسعه نرم افزار طولانی است؟
جواب: عامل مهم در طولانی شدن توسعه نرم افزارها، عدم امکان اندازه‌گیری مستقیم و عدم استفاده از مؤلفه‌های آماده و آزمایش شده در پیاده‌سازی است. راهکار استفاده از روش توسعه مبتنی بر مؤلفه^۱ (CBD) تاحدی می‌تواند در رفع این مشکل کمک کند.
- چرا هزینه توسعه نرم افزار بالا است؟
جواب: به علت تغییرات محیطی، فنی و طولانی شدن زمان توسعه نرم افزار، هزینه‌ها افزایش می‌یابد.
- چگونه می‌توان خطاهای نرم افزار را پیش از تحویل شناسایی کرد؟
جواب: در مراحل تحلیل و طراحی نرم افزار، بازبینی‌های مستمر انجام و کد تولید شده، به حد کفایت مورد آزمایش قرار گیرد.
- چرا در اندازه‌گیری پیشرفت پروژه نرم افزاری مشکل وجود دارد؟
جواب: چون معیارهای دقیق و خاصی برای سنجش نرم افزار وجود ندارد و این معیارها به پارامترهای بسیاری از تحلیل، طراحی و پیاده‌سازی وابسته‌اند.

۴-۱ ویژگی‌های نرم افزار

- نرم افزار یک محصول منطقی است و یک محصول فیزیکی نیست.
 - نرم افزارها بیشتر بر اساس نیاز مشتریان ساخته می‌شوند.
 - نرم افزار دوراندختنی نیست.
- در شکل ۱ به‌وضوح دیده می‌شود که منحنی عمر مفید نرم افزار محدب و دارای نرخ شکست صعودی با شیب بسیار کم نسبت به منحنی ایده‌آل در طول زمان است که این رشد نرخ شکست ناشی از تغییر نیازها است.



شکل ۱-۱ : منحنی های شکست نرم افزار

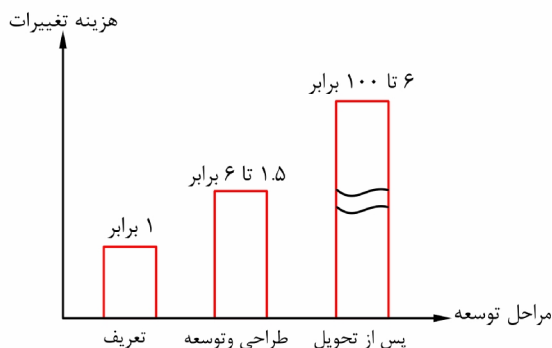
۵-۱ کاربردهای نرم افزار

محتوای اطلاعات و قطعیت اطلاعاتی در تعیین ماهیت کاربردی یک نرم افزار مطرح هستند. منظور از محتوا معنی و شکل اطلاعات ورودی و خروجی و قطعیت اطلاعاتی به معنای قابلیت پیش بینی ترتیب و زمان بندی اطلاعات است. با این وجود تعیین گروه های کلی بامعنی برای کاربردهای نرم افزاری تاحدی دشوار است. با پیچیده تر شدن نرم افزار، مرزهای صریح و روشن از بین می روند. به هر حال می توان زمینه های زیر را به عنوان گروه های کاربردی مشخص کرد:

- نرم افزارهای سیستمی
- نرم افزارهای زمان حقیقی
- نرم افزارهای تجاری
- نرم افزارهای مهندسی و علمی
- نرم افزارهای توکار و تعبیه شده
- نرم افزارهای مبتنی بر رایانه های شخصی
- نرم افزارهای مبتنی بر وب
- نرم افزارهای هوش مصنوعی و شبکه های عصبی و ...

۶-۱ هزینه های ناشی از تغییرات در چرخه توسعه نرم افزار

برآوردهای زیر از تعداد زیادی نرم افزار در چرخه توسعه نرم افزار انجام شده است که نشان دهنده شدت افزایش هزینه تغییرات پس از تحویل نرم افزار است.



شکل ۲-۱ هزینه تغییرات در چرخه توسعه نرم افزار

فصل دوم

فرآیند توسعه نرم افزار

مهندسی نرم افزار یک فناوری لایه‌ای است که مبتنی بر لایه کیفیت است. مدیریت کیفیت به ارتقای مداوم فرآیند توسعه کمک می‌کند، به طوری که همواره بهبود مستمر در فرآیند مهندسی نرم افزار انجام شود. اساس مهندسی نرم افزار لایه‌های فرآیند آن است. فرآیند مهندسی نرم افزار همراه با لایه‌های فن‌آوری (روش‌ها و ابزارها) باعث توسعه تدریجی نرم افزارهای رایانه‌ای می‌شوند.

۱-۲ فرآیند نرم افزار چیست؟

چارچوبی برای کارها و فعالیت‌های مورد نیاز برای توسعه نرم افزاری با کیفیت بالا را فرآیند نامند.

۲-۲ ضوابط ارزیابی نرم افزار

ارزیابی نرم افزار از طریق عوامل داخلی و خارجی، با در نظر داشتن پارامترهای متفاوت صورت می‌گیرد.

۱-۲-۲ عوامل خارجی (کاربران)

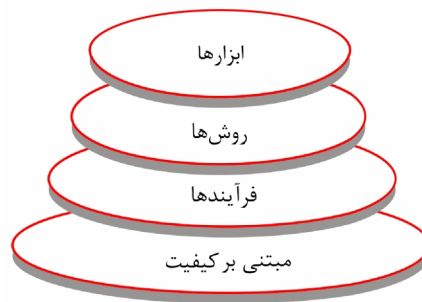
کسانی که از نرم افزار فقط استفاده می‌کنند و در فرآیند توسعه آن نقشی ندارند. معیارهای ارزیابی نرم افزار از نظر این افراد عبارت‌اند از:

- صحت برنامه^۱
- استحکام^۲ از دید کاربر
- قابلیت توسعه^۳
- قابلیت مصرف مجدد^۴
- سازگاری^۵
- قابلیت حمل^۶
- کارایی^۱

۲-۲-۲ عوامل خارجی (کاربران)

عوامل داخلی به توسعه‌دهندگان نرم‌افزار اطلاق می‌شود. مهندسی نرم‌افزار از دیدگاه فنی به ارزیابی نرم‌افزار می‌پردازد و به دنبال ارائه پاسخی مشخص برای سؤالات زیر هستند:

- از چه ابزارهایی^۲ استفاده شده است؟
- چه روش‌هایی^۳ به کار برده شده است؟
- چطور پردازش^۴ انجام شده است؟
- مستندات کیفیت^۵ چیست؟ (تمرکز بر کیفیت)



شکل ۱-۲ لایه‌های مهندسی نرم‌افزار

۳-۲ مراحل فرایند مهندسی نرم‌افزار

فرایند مهندسی نرم‌افزار شامل سه مرحله تعریف، توسعه و نگهداری نرم‌افزار است.

۱-۳-۲ تعریف

فعالیت‌های مرحله تعریف

این مرحله از فرایند توسعه نرم‌افزار با سه فعالیت زیر انجام می‌شود:

- مهندسی سیستم یا اطلاعات^۶
- برنامه‌ریزی توسعه پروژه نرم‌افزاری^۷
- تحلیل و مدل‌سازی نیازها^۸

۲-۳-۲ توسعه

در این مرحله مدل‌های مورد نیاز تولید می‌شوند و روند کلی برنامه‌ها تعیین و ساختار پایگاه داده‌ها نیز طراحی می‌شود.

فعالیت‌های مرحله توسعه

- طراحی نرم‌افزار^۹
- تولید کد^{۱۰}
- آزمایش^{۱۱}

1 - Efficiency
 2 - Tools
 3 - Methods
 4 - Process
 5 - Quality
 6 - System/Information Engineering
 7 - Software Development Project Planning
 8 - Requirements Analysis and Modeling
 9 - Software Design
 10 - Code Generation
 11 - Testing

۳-۳-۲ نگهداری

- نگهداری اصلاحی^۱: اشکالات در چرخه زندگی پروژه، شناسایی، بررسی و رفع می شود (رفع خطا).
- نگهداری تطبیقی^۲: تغییر به مرور زمان با توجه به نیازهای محیطی و کاربر صورت می گیرد (تطبیق با نیازهای جاری).
- نگهداری ارتقایی^۳: ارتقای نرم افزار پس از راه اندازی که توسط کاربر درخواست می شود (افزایش امکانات).
- نگهداری پیشگیرانه^۴: ایجاد تغییراتی در برنامه ها که منجر به ارتقا و بهبود عملکرد نرم افزار می شود (آینده نگری).

۴-۳-۲ فعالیت های پشتیبانی فرایند مهندسی نرم افزار

فعالیت های چتری یا پشتیبانی^۵ در تمامی مراحل زیر انجام می گیرد:

۱. کنترل و نظارت بر برنامه ریزی پروژه^۶
۲. بازنگری های مستمر فنی رسمی^۷
۳. اطمینان مرغوبیت نرم افزار^۸
۴. مدیریت پیکربندی نرم افزار^۹
۵. تهیه و تدوین مستندات فنی^{۱۰}
۶. اندازه گیری نرم افزار با مقیاس های اندازه گیری^{۱۱}
۷. مدیریت ریسک نرم افزار^{۱۲}

۴-۲ مدل بلوغ توانایی (CMM)^{۱۳}

مؤسسه مهندسی نرم افزار (SEI)^{۱۴} آمریکا مدل جامعی ارائه داده است که مجموعه ای از توانایی های مهندسی نرم افزار برای دسترسی سازمان ها به سطوح مختلف بلوغ فرایند در آن پیش بینی شده است. این مدل تعیین کننده میزان مؤثر بودن فعالیت های مهندسی نرم افزار در یک سازمان نرم افزاری بوده و دارای پنج سطح تکامل فرایندی است.

سطح ۱: اولیه^{۱۵}

فرایند نرم افزار به صورت موقتی و حتی بعضی اوقات بسیار درهم و برهم توصیف شده است. چند فرایند ساده تعریف می شود و موفقیت به تلاش های فردی بستگی دارد.

از نظر تضمین کیفی و مدیریت پروژه، وظیفه خاصی وجود ندارد. در این سطح تیم پروژه می تواند برای توسعه و ارائه نرم افزار هر راهی انتخاب کند. روش ها، استانداردها و رویه هایی که از کیفیت خوب تا خیلی ضعیف دارند، می توانند در این انتخاب قرار بگیرند.

-
- 1 - Correction
 - 2 - Adaptation
 - 3 - Enhancement
 - 4 - Prevention
 - 5 - Umbrella Activities
 - 6 - Project Monitoring and Control
 - 7 - Formal Technical Review
 - 8 - Quality Assurance
 - 9 - Configuration/Change Management
 - 10 - Technical Documentation
 - 11 - Software Measurements
 - 12 - Risk Management
 - 13 - Capability Maturity Model
 - 14 - Software Engineering Institute
 - 15 - Initial

سطح ۲: قابل تکرار^۱

فرایندهای اولیه مدیریت پروژه برای مشخص کردن هزینه، زمان بندی و کارایی، در این سطح صورت می گیرد. این سطح نمایانگر این حقیقت است که ارائه دهنده نرم افزار، فعالیت های خاصی را همچون گزارش تکمیل کار و گزارش زمان بندی و فعالیت های انجام شده را تعریف کرده است.

سطح ۳: تعریف شده^۲

فرایند نرم افزار برای مدیریت فعالیت های مهندسی در طول فرایند توسعه نرم افزار در سازمان ثبت شده، استاندارد و منسجم است. این سطح شامل مشخصه های تعریف شده در سطح ۲ نیز است. این سطح نمایانگر این است که ارائه دهنده نرم افزار، فرایندهای فنی و مدیریتی را تعریف کرده است. این سطح از سطوحی است که اکثر تولیدکنندگان نرم افزار از طریق استانداردهایی مثل ایزو ۹۰۰۱ آن را مد نظر دارند.

سطح ۴: مدیریت شده^۳

اقدامات دقیق صورت گرفته در فرایند نرم افزار و کیفیت محصول، همگی کنترل می شوند. فرایند نرم افزار و محصولات از لحاظ کمی شناسایی شده و با استفاده از اقداماتی دقیق کنترل می شوند. این سطح دربرگیرنده مفهوم اندازه گیری و استفاده از معیارهای سنجش برای فرایند توسعه نرم افزار و محصول نرم افزاری است. این سطح شامل تمام مشخصه های تعریف شده در سطح ۳ نیز است.

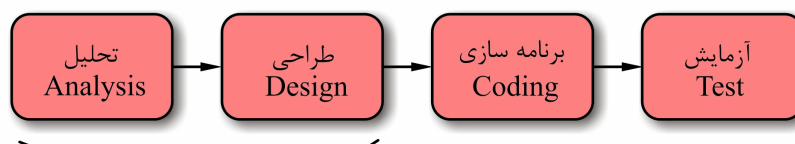
سطح ۵: بهینه سازی^۴

بهبود مکرر فرایند با توجه به بازخورد کمی فرایند و از طریق آزمون ایده های نوآور و فن آوری های نوین و بروز، مقدور است. این سطح شامل تمام مشخصه های تعریف شده برای سطح ۴ است.

۵-۲ مدل های پایه در فرایند توسعه نرم افزار

۱-۵-۲ مدل ترتیبی خطی

این مدل گاهی مدل پایه نیز نامیده می شود. مدل خطی ترتیبی^۵ یک سری فعالیت متوالی برای توسعه نرم افزار پیشنهاد می کند که از سطح تحلیل آغاز و به آزمایش نرم افزار ختم می شود.



مهندسی سیستم / اطلاعات : شناسایی و تحلیل نیازها

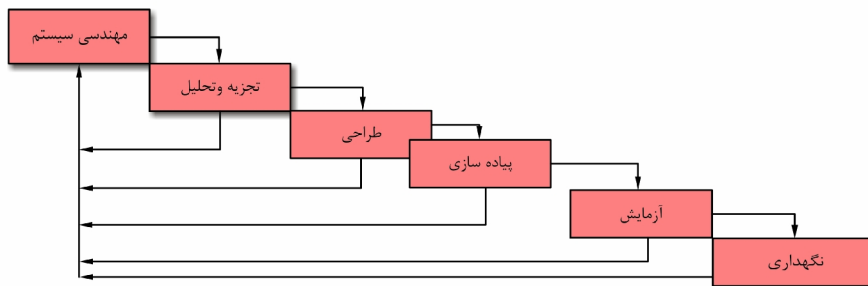
شکل ۲-۲ مدل ترتیبی خطی

این مدل برای پروژه های کوچک کارا است زیرا در آن تعریف صورت مسئله به راحتی انجام می گیرد.

۲-۵-۲ مدل آبشاری

مدل آبشاری^۶ نیز اغلب با نام مدل دوره زندگی کلاسیک معرفی می شود. در این مدل فعالیت توسعه نرم افزار از مهندسی سیستم شروع و به نگهداری نرم افزار ختم می شود.

1 - Repeatable
2 - Defined
3 - Managed
4 - Optimizing
5 - The Linear Sequential Model
6 - WaterFall Model



شکل ۳-۲ مدل آبشاری

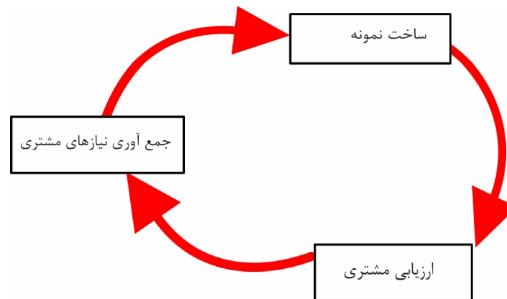
مدل آبشاری ساده است. در پروژه‌های بزرگ، مدت زمان انجام پروژه طولانی است و بنابراین نیازها تغییر می‌یابد؛ از این رو بدیهی است از این مدل در پروژه‌های بزرگ استفاده نشود.

مشکلات مدل

- مشکل برگشت به عقب در نتیجه بروز تغییرات
- عدم بیان نیازها توسط مشتری در ابتدای پروژه
- طولانی منتظر ماندن مشتری برای دستیابی به نرم افزار

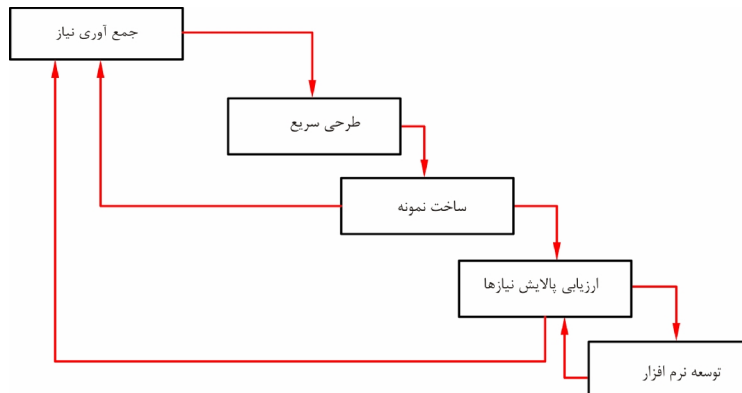
۳-۵-۲ مدل نمونه سازی

بخش‌های مختلف سیستم به صورت مرحله‌ای شناسایی و نمونه سازی می‌شود. مدل نمونه سازی^۱، نوعی تولید مجدد محسوب می‌شود (دوباره کاری). در حالت ایده آل، نمونه سازی روشی برای مشخص کردن نیازهای نرم افزار محسوب می‌شود.



شکل ۴-۲ روش نمونه سازی

در ابتدا برنامه خیلی ساده در اختیار مشتری قرار می‌گیرد و سپس آن را طبق نیازهای مشتری تغییر می‌دهند.



شکل ۵-۲ نمونه سازی در مدل آبشاری

مزیت

- ارتباط با مشتری در تمام طول اجرای پروژه برقرار است.
- تغییر نیازها به راحتی امکان پذیر است.

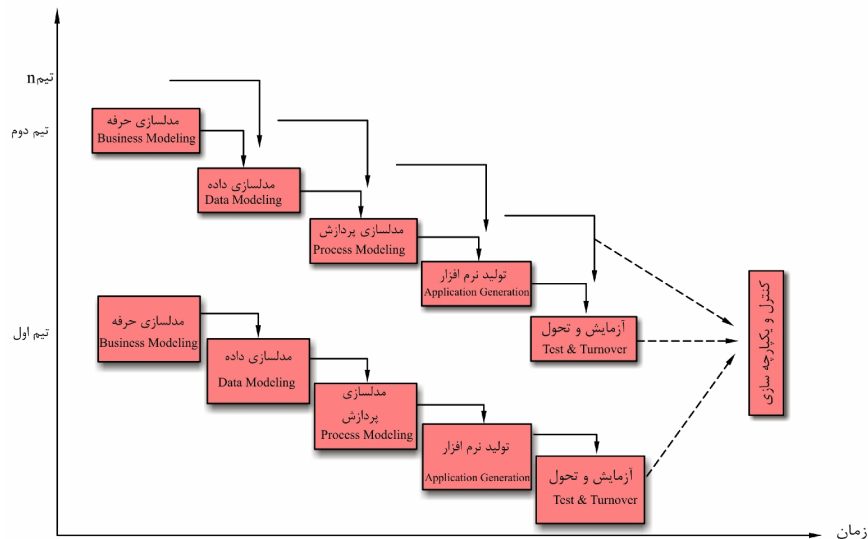
عیب

- مشتری، کل نرم افزار را نمی بیند و ملاحظات کلی از نرم افزار را نمی تواند داشته باشد (مثل قابلیت نگهداری).
- تغییر زیاد ممکن است باب میل مشتری نباشد (مانند نمونه های ساخته شده دوراندختنی).
- سرعت انجام کار باعث بروز اثرات جانبی می شود (مثل تغییر و محدودیت های سخت افزاری یا نرم افزاری).

۴-۵-۲ مدل توسعه سریع کاربرد (RAD)

در این مدل تأکید بر توسعه سریع نرم افزار^۱ در مدت زمان کوتاه، حداکثر در سه ماه است. در هر قسمت، روش خطی و یا هر مدل دیگر قابل استفاده است.

در این روش سیستم در قالب کارکردهای (پیمانه های) مختلف شکسته می شود و تیم های مختلف، همزمان روی هر یک از آنها فعالیت می کنند و در نهایت نتیجه کار با هم ترکیب و یکپارچه می شود.



شکل ۶-۲ مدل توسعه سریع کاربرد

مزیت

- زمان توسعه و تولید نرم افزار کوتاه است.
- تأکید بر قابلیت استفاده مجدد مؤلفه های نرم افزاری است.

عیب

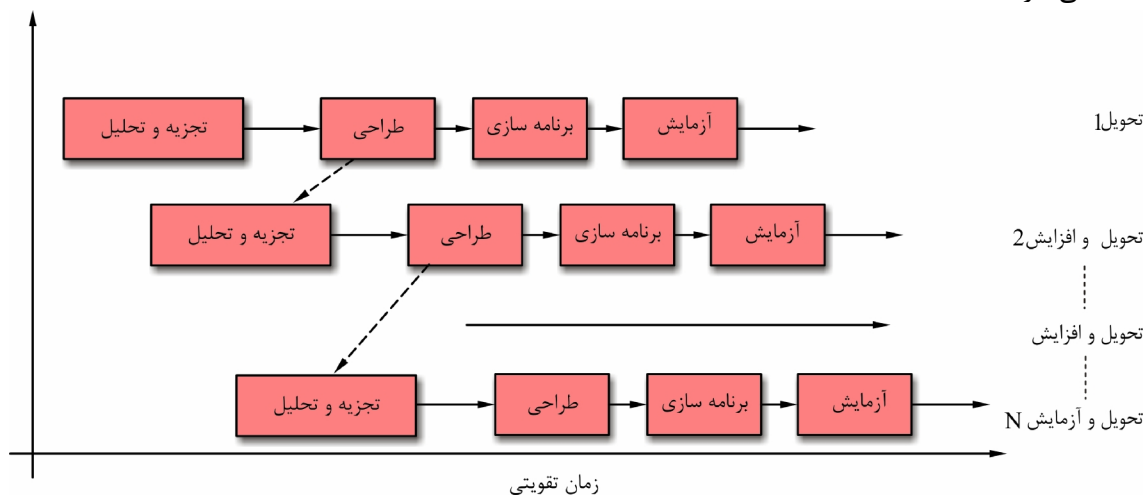
- مدیریت تیمها و هماهنگی آنها خیلی مشکل است.
- در سیستم هایی می توان این کار را کرد که قابلیت پیمانه های^۲ وجود داشته باشد که اغلب مواقع این طور نیست.
- در پروژه های بزرگ نیاز به افراد متخصص زیادی است.
- عدم اطمینان از تعهد کاری و پایبندی افراد در تیم های پروژه به عنوان ریسک، همواره وجود دارد.

۶-۲ مدل های تکاملی فرایند توسعه نرم افزار

ایده گسترش و بلوغ نرم افزار مانند سایر سیستم های پیچیده همواره مطرح است. با رشد ابزارها و فن آوری مربوطه، فرایندهای توسعه نرم افزار نیز تکامل می یابند. در این بخش مدل های تکاملی فرایند توسعه نرم افزار مورد بررسی قرار می گیرند. مدل های تکاملی ماهیتاً تکراری هستند، یعنی مهندسین نرم افزار را قادر می کند تا به صورت افزایشی نسخه های کامل تری از نرم افزار را توسعه و عرضه کنند.

۱-۶-۲ مدل افزایشی

در مدل افزایشی^۱ ابتدا محصول پایه ای تولید می شود و سپس در هر مرحله، نسخه ای از نرم افزار به مشتری ارائه می شود که امکانات بیشتری نسبت به نسخه قبلی دارد.



شکل ۷-۲ مدل افزایشی

مزیت

- برعکس روش نمونه سازی، نسخه های نرم افزار دور ریخته نمی شود.
- افزودن امکانات جدید به نرم افزار، ضمن به هنگام سازی آن، هزینه کمتری دربردارد.

عیب

- نیاز به نیروی انسانی متخصص و یا سخت افزار کافی دارد.

۲-۶-۲ مدل حلزونی

مدل حلزونی^۲ (معروف به مدل Bohem) روش خطی را با فلسفه نمونه سازی ترکیب می کند که در پروژه های بزرگ مورد استفاده است. در این مدل فعالیت های مربوط به مهندسی نرم افزار به شش زمینه کاری تقسیم می شود:

ارتباط با مشتری^۳: قراردادهای و کل توافقات انجام گرفته و اطلاعات کلان و کلی از پروژه جمع آوری می شود.

برنامه ریزی^۴: بر اساس نتایج به دست آمده از شناخت کلان، اعضای تیم شناسایی و تیم های کاری تشکیل و برای هر تیم، تخصیص کار صورت می گیرد و برنامه کنترل و نظارت پروژه تدوین می شود.

تحلیل ریسک^۵: پیش بینی مخاطرات غیر منتظره مثل تغییرات فن آوری سخت افزار و نرم افزار، عدم تعهد کارفرما یا نیروی انسانی، تغییرات مدیریتی، تغییر خواسته مشتری (عوامل محیطی) و ... انجام می شود.

مهندسی و طراحی^۱: فعالیت های مهندسی مورد نیاز برای طراحی و ساخت نمونه هایی از برنامه کاربردی انجام می گیرد.

1 - Incremental Model

2 - Spiral Model

3 - Customer Communication

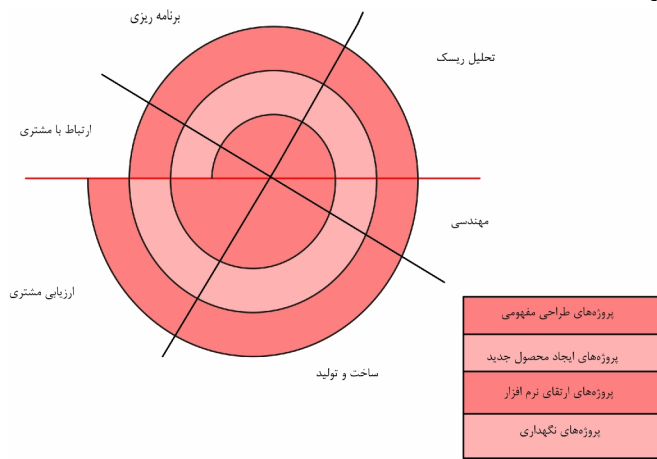
4 - Planning

5 - Risk Analysis

ساخت و تحویل^۲: برنامه نویسی، آزمون، کنترل کیفیت، نصب و آموزش از عمده فعالیت‌های این بخش است. ارزیابی مشتری^۳: اخذ نظرات مشتری و کاربران در مورد محصولات تحویلی به آنان و ارزیابی و اعمال تغییرات در صورت نیاز در این مرحله صورت می‌گیرد.

مزایا

- در هر دور از مدل، تحلیل ریسک برای بهبود کیفیت نرم افزار صورت می‌گیرد.
- تعداد دورهای این روش با توافق مشتری و هزینه‌ای که پرداخت می‌کند تعیین می‌شود.
- مدیریت پروژه همواره بر پروژه اعمال می‌شود.
- کنترل ریسک همواره وجود دارد.

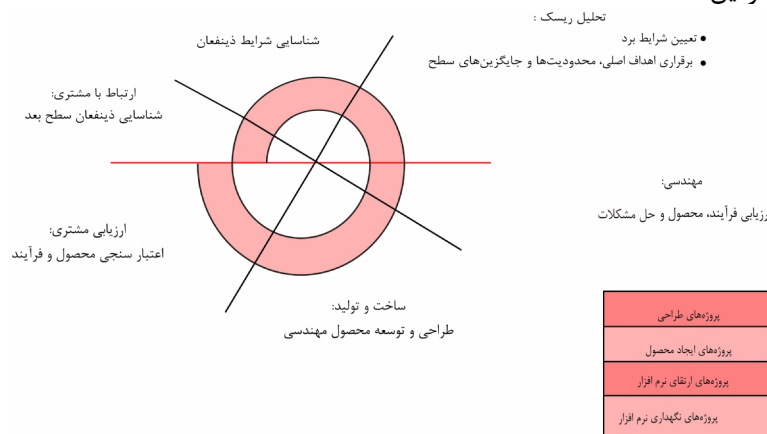


شکل ۲-۸ مدل حلزونی (مارپیچی)

۲-۶-۳ مدل حلزونی WinWin

استخراج نیازهای نرم افزار وابسته به مذاکرات است. مذاکرات موفق یعنی توفیق و موفقیت طرفین. مدل حلزونی بوهم یک سری مذاکرات را در آغاز هر مرحله از مارپیچ بیان می‌کند و تمامی فعالیت‌های زیر تعریف می‌شوند:

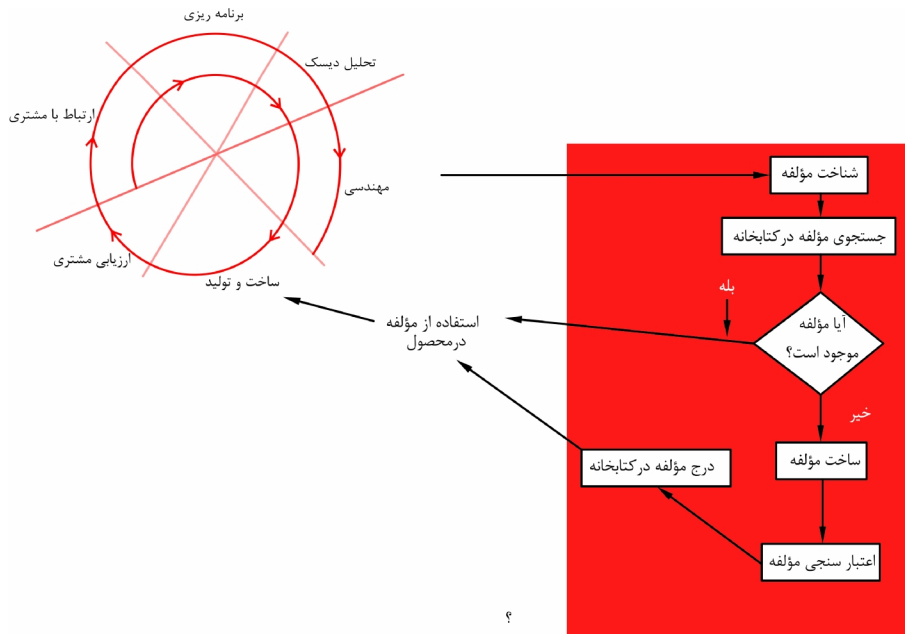
- ۱- شناسایی سیستم یا دارندگان اصلی سیستم‌های فرعی.
- ۲- تعیین شرایط برد طرفین.
- ۳- مذاکره در مورد شرایط برد طرفین.



شکل ۲-۹ مدل حلزونی WinWin

۴-۶-۲ مدل مونتاز مؤلفه‌ها^۱

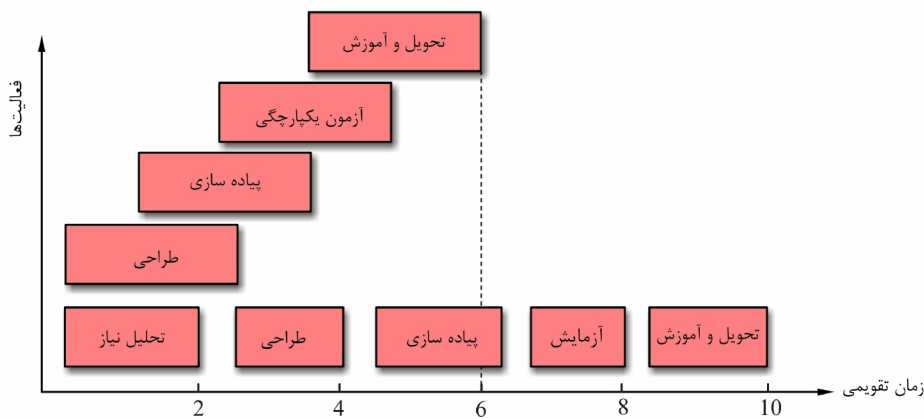
اجزای سیستم به صورت مؤلفه‌هایی در نظر گرفته می‌شوند که از قابلیت ساخت، آزمون و استفاده مناسبی برخوردار هستند. هر مدل یا روشی در توسعه نرم افزار در اینجا استفاده شود، فرقی نمی‌کند. در این مدل هدف و تأکید بر قابلیت استفاده مجدد از مؤلفه‌هاست.



شکل ۱۰-۲ مدل توسعه مبتنی بر مؤلفه

۵-۶-۲ مدل توسعه همروند^۲

در مدل توسعه همروند^۲ فعالیت‌ها به منظور کاهش زمان ایجاد نرم افزار به طور موازی انجام می‌گیرند.



شکل ۱۱-۲ مدل توسعه همروند

در حالت موازی، زمان کاهش می‌یابد ولی ممکن است بعضی پارامترهای دیگر مثل دوباره کاری و ... افزایش یابد. در این حالت مدیریت پروژه نرم افزاری پیچیدگی زیادی دارد.

1 - Component Assembly Model
2 - Concurrent Development Model

۲-۶-۶ مدل روش‌های رسمی^۱

مجموعه‌ای از فعالیت‌ها که نرم‌افزار رایانه‌ای را در قالب روابط ریاضی طراحی و بیان می‌کند.

مزایا	معایب
<ul style="list-style-type: none"> ✓ بدیع ✓ سازگار ✓ بدون ابهام ✓ کاربرد در سیستم‌های حساس ✓ و ... 	<ul style="list-style-type: none"> ✓ وقت‌گیر و پرهزینه ✓ کاربرد محدود ✓ نیاز به آموزش جامع ✓ مشکل ارتباط با مشتری ✓ و ...

۲-۶-۷ ابزارهای نسل چهارم^۲

شامل محدوده وسیعی از ابزارهای نرم‌افزاری است که قادر به تعیین و مشخص کردن بعضی از خصوصیات و ویژگی‌های نرم‌افزار در سطح بالاست و می‌تواند حتی کدی بر مبنای مشخصات تولیدشده به صورت خودکار ایجاد کند. در این ابزارها قابلیت‌های زیر وجود دارد:

- قابلیت استفاده برای مستندسازی
- قابلیت استفاده برای مدل‌سازی سیستم
- قابلیت تولید گزارشات
- قابلیت استفاده آسان از امکانات محیط برنامه‌سازی
- قابلیت مدیریت آسان پایگاه داده‌ها
- قابلیت تولید کد برنامه
- و ...

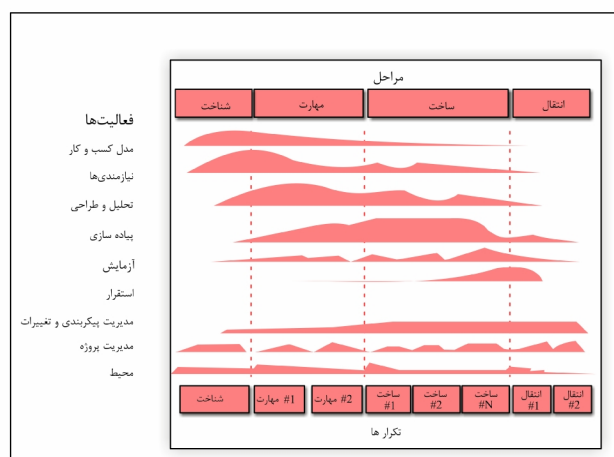
عیب

- تکنیک‌ها و ابزارهای موجود در حال حاضر خیلی عام نیستند و کاربردهای خاص دارند.

۲-۶-۸ فرایند مدل‌سازی یکنواخت^۳

در اواخر دهه ۹۰ میلادی به منظور ایجاد یک بستر یکسان برای انجام فرایندهای مهندسی نرم‌افزار چارچوبی توسط شرکت Rational Rose تحت عنوان RUP^۴ ارائه شد که در آن چهار فاز اصلی و نه نظام کاری تعریف شد.

خروجی‌های هر فاز در چارچوب مستندات مشخص تدوین و جریان کار نظام‌های کاری نیز مشخص شده است. در این فرایند ایده تکراری^۵ بودن فرایند مهندسی نرم‌افزار نیز مد نظر بوده که در هر فاز تکرارهای متفاوتی را می‌توان اجرا کرد. چارچوب RUP به صورت زیر است.



شکل ۲-۱۲ چارچوب RUP

1 - Formal Methods Model
 2 -4GT-Forth Generation Tool
 3 -Unified Modeling Process
 4 -Rational Unified Process
 5 -Iterative

فصل سوم

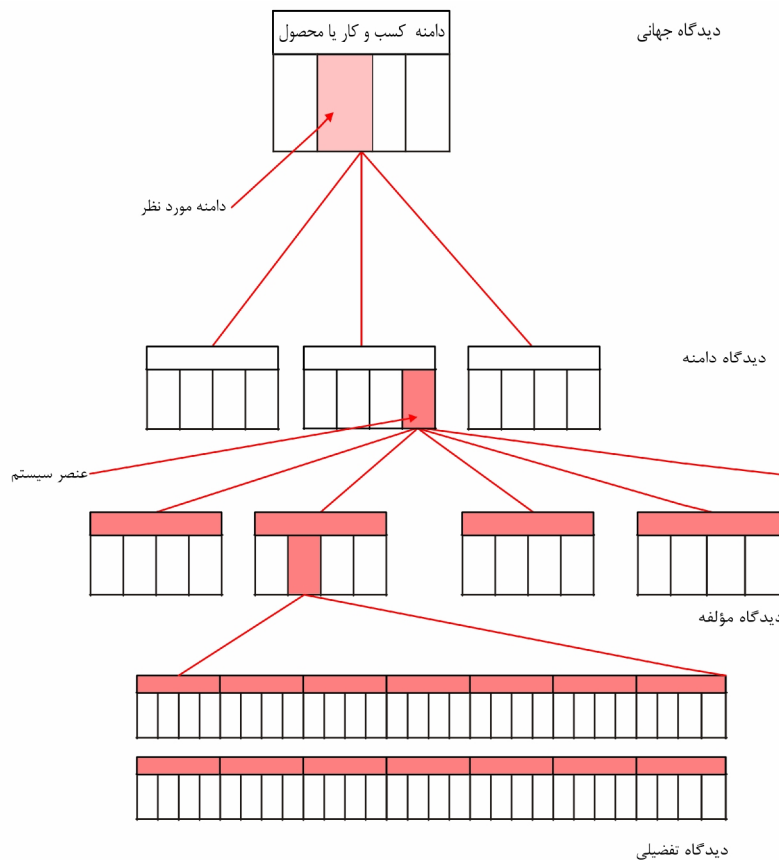
مهندسی سیستم

۱-۳ تعریف مهندسی سیستم

مهندسی سیستم^۱ فرایندی است که تمرکز خود را صرف مجموعه‌ای از عناصر تحلیل، طراحی و سازماندهی آنها در قالب یک سیستم می‌کند که حاصل این کار تولید یک محصول، ارائه خدمت یا سرویس یا یک فن‌آوری مناسب برای تبدیل اطلاعات و کنترل آنهاست. اگر محدوده کار مهندسی سیستم تمرکز بر کسب و کار داشته باشد، مهندسی فرایند حرفه یا مهندسی اطلاعات نامیده می‌شود. هنگامی که توسعه یک محصول مد نظر باشد، این فرایند، مهندسی محصول نامیده می‌شود. مهندسی اطلاعات و مهندسی محصول، هر دو در قلمرو علمی سعی در توسعه سیستم‌های مبتنی بر رایانه دارند، اگرچه هر یک در محدوده کاربرد متفاوتی به کار گرفته می‌شوند.

۲-۳ سلسله‌مراتب مهندسی سیستم

علی‌رغم دامنه تمرکز، مهندسی سیستم شامل گروهی از روش‌های بالا به پایین و پایین به بالا برای حرکت در سلسله‌مراتب نشان داده شده در شکل زیر است. فرایند مهندسی سیستم معمولاً با "دید جهانی" شروع می‌شود. دیدگاه جهانی پالایش می‌شود تا تمرکز کامل‌تری بر دامنه خاصی از خواسته‌ها به دست آید. در یک دامنه خاص، نیاز برای عناصر سیستم هدف (برای مثال، داده‌ها، نرم‌افزار، سخت‌افزار، افراد) بررسی می‌شود. در نهایت، تحلیل، طراحی و ساخت یک عنصر سیستمی مورد نظر آغاز می‌شود. در بالای این سلسله‌مراتب، یک دامنه وسیع وجود دارد و در پایین‌ترین سطح سلسله‌مراتب سیستمی، جزئیات فعالیت‌های فنی که با روش‌های خاص مهندسی مدل‌سازی و پیاده‌سازی می‌شوند، مورد توجه قرار می‌گیرند.



شکل ۱-۳ سلسله مراتب مهندسی سیستم

۳-۳ بخش های مهندسی سیستم

۱-۳-۳ مهندسی فرایند حرف (BPE)^۱

هدف مهندسی فرایند حرفه که به آن **مهندسی اطلاعات (IE)**^۲ نیز گفته می شود، تعریف معماری هایی است که امکان استفاده کارا از اطلاعات در کاربردهای عملی را فراهم می سازند.

۱-۱-۳-۳ سه نوع معماری مهندسی اطلاعات

- معماری داده ها^۳
 قالبی برای نیازهای اطلاعاتی مؤسسه در نظر گرفته و واحد سازنده این قالب Data Object است که پس از شناسایی، رابطه بین آن ها مشخص می شود. یک رابطه، نشان دهنده نحوه ارتباط اشیا با یکدیگر است.
- معماری کاربردها^۴
 معماری کاربرد شامل عناصری از یک سیستم است که صفات اشیا را در معماری داده ای برای اهداف حرفه خاص دستکاری و تبدیل می کند. معماری کاربردها، سیستم یا برنامه های (نرم افزاری) است که این تبدیلات را انجام می دهند. بهر حال، در یک محدوده وسیع، معماری کاربرد، نقش افراد (که مبدل های اطلاعات یا کاربران هستند) و رویه های حرفه را که خودکار نشده اند بر عهده دارد.
- زیرساخت فن آوری^۵

1 - Business Process Engineering
 2 - Information Engineering
 3 - Data Architecture
 4 - Application Architecture
 5 - Technology Infrastructure

مجموعه‌ای از نرم افزارها و سخت افزارها که دو معماری قبلی را پوشش می‌دهند. زیرساخت فن آوری، اساس معماری‌های داده و کاربرد را فراهم می‌کند. این زیرساخت شامل سخت افزار و نرم افزاری است که برای پشتیبانی کاربردها و داده‌ها از آن‌ها استفاده می‌شود.

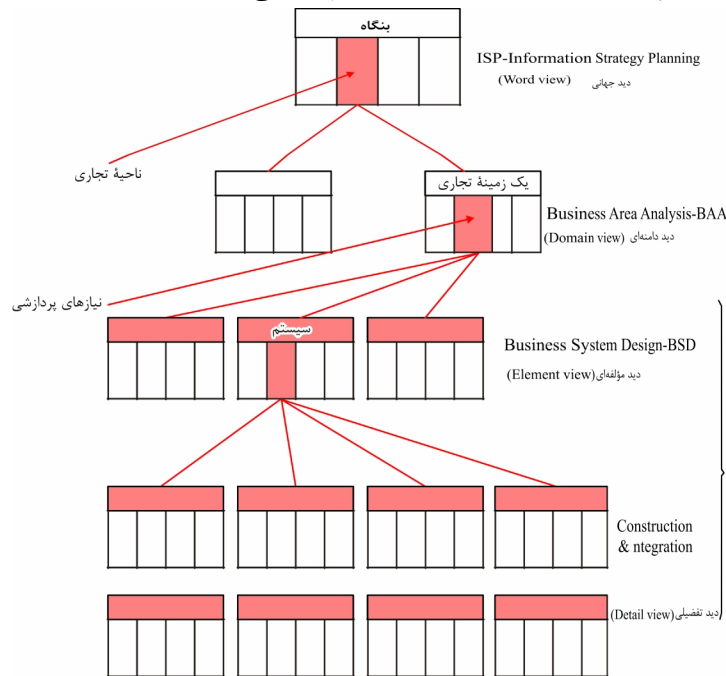
۲-۱-۳-۳ سلسله مراتب مهندسی اطلاعات

سلسله مراتب فعالیت‌های مهندسی اطلاعات با دیدگاه کلی از طریق برنامه ریزی راهبردی اطلاعات (ISP)^۱ به دست می‌آید. ISP به کل مجموعه کسب و کار به عنوان یک موجودیت می‌نگرد و دامنه‌های این حرفه را (برای مثال، مهندسی، ساخت، بازاریابی، تجارت، فروش) که برای کل سازمان مهم هستند شناسایی و تفکیک می‌کند. ISP اشیای داده‌ای را که در سطح سازمان قابل رؤیت هستند، رابطه آن‌ها و نحوه جریان آن‌ها بین دامنه‌های موجود در حرفه را نیز تعریف می‌کند.

دیدگاه دامنه، با فعالیت تحلیل زمینه حرفه (BAA)^۲ مورد توجه قرار می‌گیرد. BAA جزئیات داده‌ها (به شکل انواع شیء داده‌ای و موجودیت‌ها) و نیازهای عملکردی (به شکل فرایندها) مربوط به ناحیه‌های انتخاب شده (دامنه‌ها) در حرفه که در ISP مشخص شده‌اند و تعیین ارتباطات آن‌ها (به شکل ماتریس‌های متقابل ارجاعات) را مشخص می‌کند.

پس از تعیین سیستم اطلاعاتی، در مرحله طراحی سیستم حرفه (BSD)^۳، نیازمندی‌های اصلی سیستم اطلاعاتی خاص مدل سازی می‌شوند و این نیازها به معماری داده‌ها، معماری کاربردها و زیرساخت فن آوری ترجمه می‌شوند.

مرحله نهایی، ساخت و یکپارچه سازی مؤلفه‌هاست که بر جزئیات پیاده سازی تمرکز دارد. فعالیت یکپارچه سازی نیز سیستم اطلاعاتی جدیدی را در محدوده زمینه تجاری قرار می‌دهد و تمام آموزش‌های کاربران و حمایت‌های پشتیبانی را برای به دست آوردن تغییر ملایم، ایجاد می‌کند.



شکل ۲-۳ سلسله مراتب مهندسی اطلاعات

۲-۳-۳ مهندسی محصول

هدف مهندسی محصول، تولید یا توسعه یک محصول (نرم افزار) و یا ارائه خدمت بر اساس نیازهای مشتری است که شامل 4 عنصر مجزاست:

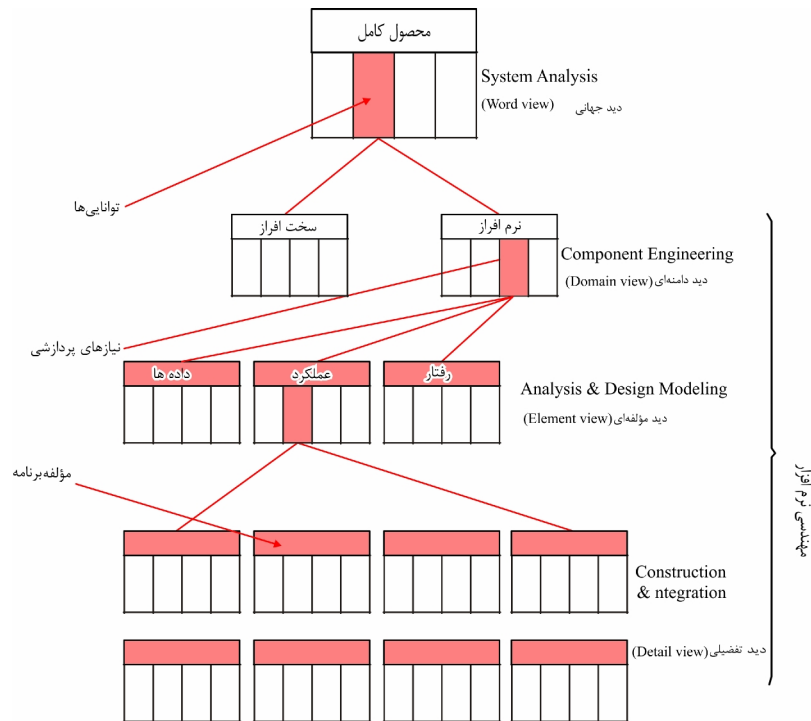
- نرم افزار
- سخت افزار
- افراد
- پایگاه داده‌ها

1 - Information Strategy Planning

2 -Business Area Analysis

3 -Business System Design

زیرساخت پشتیبان برای مهندسی محصول عبارت‌اند از:
 فن‌آوری: (ملزومات مورد نیاز برای ایجاد ارتباط بین چهار عنصر مذکور چیست؟)
 اطلاعات: (چه اطلاعاتی برای ارتباط بین عناصر مذکور برای تولید محصول وجود دارد؟)



شکل ۳-۳ سلسله‌مراتب مهندسی محصول

با مراجعه به شکل ۳-۳، دیدگاه کلی از طریق مهندسی نیازها به دست می‌آید. نیازهای کلی محصول از طریق مشتری جمع‌آوری می‌شود. این نیازها عبارت‌اند از: نیازهای اطلاعاتی و کنترلی، عملکرد و رفتار محصول، کارایی کلی محصول، محدودیت‌های طراحی و ارتباطی و نیازهای خاص.

۴-۳ مهندسی نیازها

مهندسی نیازها مکانیزم مناسبی را برای درک خواسته‌های مشتری، تحلیل نیازها، تعیین امکان‌پذیر بودن اجرای پروژه، مذاکره در مورد راه حل قابل قبول، مشخص کردن راه حل به صورت غیر مبهم، اعتبارسنجی مشخصه محصول و مدیریت نیازها در ضمن تبدیل به یک سیستم عملیاتی فراهم می‌کند. بنابراین فرایند مهندسی نیازها در ۶ مرحله مشخص به صورت زیر انجام می‌گیرد:

۱-۴-۳ بیان نیازها

نیازها از طریق مصاحبه با مشتری به دست می‌آید ولی این کار به دلایل زیادی خیلی ساده نیست. Christel و Kang چند مورد را مشخص کرده‌اند که به فهم پیچیدگی تعیین نیازها کمک می‌کنند:

- مشکلات محدوده
- مشکلات مفهومی
- مشکلات اعتبار

۲-۴-۳ تحلیل نیازها

پس از جمع‌آوری نیازها، مبنایی برای تحلیل نیازها به وجود می‌آید. انجام فعالیت تحلیل، نیازها را دسته‌بندی می‌کند و آن‌ها را به صورت زیرمجموعه‌های مرتبط سازماندهی می‌کند، رابطه هر نیاز با بقیه را مشخص می‌کند و برای مواردی از قبیل تطابق، موارد گمشده و ابهام، نیازها را مجدداً بررسی می‌کند و آن‌ها را بر مبنای ضروریات مشتری و کاربر طبقه‌بندی می‌کند.

۳-۴-۳ تدوین مشخصه نیازها

یک مشخصه می‌تواند یک سند مکتوب، یک مدل گرافیکی، یک مدل ریاضی، مجموعه‌ای از سناریوهای کاربرد یا هر ترکیبی از این‌ها باشد و از قالب استاندارد استفاده شود. البته گاهی لازم است در هنگام تدوین مشخصه، انعطاف مناسبی در استفاده از استانداردها وجود داشته باشد.

۴-۴-۳ مدل‌سازی سیستم

ساخت مدلی از سیستم با توجه به نیازها و مشخصه نیازهای سیستم که تصویر روشنی از عملکرد سیستم را ارائه کند.

۵-۴-۳ اعتبارسنجی نیازها

اعتبارسنجی نیازها با مرورهای رسمی و بازنگری سند مشخصه سیستم انجام می‌شود تا اطمینان حاصل شود که:

تمام نیازهای سیستم به شکل غیر مبهم بیان شده‌اند. عدم تطابق‌ها، موارد حذف‌شده و خطاها آشکار شده و اصلاح شوند، در این فرایند بررسی می‌شود که آیا این محصول کاری با استانداردهای تعریف‌شده برای فرایند، پروژه و محصول تطابق دارد یا خیر؟ عناصر اولیه مکانیزم اعتبارسنجی، مورد بازنگری و مرورهای رسمی فنی قرار می‌گیرند.

۶-۴-۳ مدیریت نیازها

همواره نیازهای سیستم‌های رایانه‌ای در حال تغییر هستند و تمایل برای تغییر نیازها در طول دوران زندگی سیستم باقی است. مدیریت نیازها شامل مجموعه‌ای از فعالیت‌ها است که به تیم پروژه کمک می‌کند تا نیازها و تغییر نیازها را در هر زمان از پیشرفت پروژه، مشخص، کنترل و پیگیری کنند. بسیاری از این فعالیت‌ها مشابه روش‌های مدیریت پیکربندی نرم‌افزار هستند که در ادامه مورد بررسی قرار می‌گیرند.

۵-۳ فعالیت‌های اساسی در مهندسی نیازها

۱-۵-۳ تعیین دامنه اطلاعات

دامنه اطلاعات شامل سه دیدگاه متفاوت از داده‌ها و کنترل در ضمن پردازش توسط برنامه رایانه‌ای است: ۱- محتوا و واسطه اطلاعات ۲- جریان اطلاعات و ۳- ساختار اطلاعات.

۲-۵-۳ مدل‌سازی

مدل‌سازی سیستم شامل تهیه مدل‌های تابعی و رفتاری است.

مدل‌های تابعی: مدل‌هایی هستند که نحوه تبدیل داده‌ها به اطلاعات را در نرم‌افزار نشان می‌دهند.

مدل‌های رفتاری: مدل‌هایی هستند که عکس‌العمل نرم‌افزار را در مواجهه با وقایع دنیای بیرونی نشان می‌دهند.

۳-۵-۳ تجزیه

مسائل اغلب آن‌قدر بزرگ و پیچیده هستند که به صورت یک مجموعه بسته قابل فهم نیستند؛ بنابراین افزاین دامنه‌های اطلاعات، عملکرد و رفتار نرم‌افزار برای درک روابط بین اجزا مهم است.

۴-۵-۳ دیدگاه‌های مورد نیاز پیاده‌سازی

یک دیدگاه ضروری از نیازهای نرم‌افزار، باید به توابع و جزئیات پیاده‌سازی پردازش‌ها نیز بپردازد.

۵-۵-۳ تهیه مشخصه نیازهای نرم‌افزار

در انتهای مرحله نیازسنجی و مدل‌سازی نیازها، گزارشی شامل شناخت نیازهای کاربر، تهیه و به کاربر یا مشتری ارائه می‌شود.

توجه: بعد از شناخت محیط، امکان‌سنجی^۱ صورت می‌گیرد و زمان و هزینه برآورد شده و به مشتری ارائه می‌شود.

فصل چهارم

اصول و قواعد تحلیل نیاز

مهندسی نیازهای نرم افزار، فرایند کشف، پالایش، مدل سازی و مشخص کردن نیازها را شامل می شود. نیازهای سیستم و نقش تخصیص داده شده به نرم افزار، در ابتدا توسط مهندس سیستم شناسایی، تعریف و همراه با جزئیات پالایش می شوند. مدل های داده ای، جریان اطلاعات و کنترل و در نهایت رفتار عملیاتی سیستم شناسایی می شوند. راه حل های جایگزین تحلیل و سپس مدل تحلیل کاملی تهیه می گردد. بنابراین فرایند تحلیل نیازها شامل شناسایی، مدل سازی، پالایش و تعیین مشخصات زمینه مورد مطالعه است.



شکل ۱-۴ ارتباط مراحل مهندسی نرم افزار

تحلیل نیاز، مهندسین نرم افزار را قادر می سازد که:

- ۱- توابع و عملکرد آنها را تعیین کنند.
- ۲- ارتباط بین نرم افزار و سایر اجزای سیستم را مشخص سازند.
- ۳- محدودیتها را تعریف و نشان دهند.
- ۴- نرم افزار مورد نظر را انتخاب کند و مدل سازی از داده ها، توابع و رفتار سیستم را انجام دهند.
- ۵- با استفاده از این مدل ها اقدام به طراحی مدل های داده ای، معماری، واسطها و پردازشها کنند.

۱-۴ فعالیت های عمده تحلیل نیاز

شناسایی مسئله

در ابتدا، تحلیل گر، مشخصه سیستم را (در صورت وجود) همراه با طرح توسعه پروژه نرم افزار (SDP)^۱ مطالعه می کند. درک نرم افزار در رابطه با سیستم و مرور محدوده نرم افزار، دارای اهمیت است. هدف از این کار تشخیص عناصر اصلی مسئله شامل شناسایی اهداف، محدودیتها و تعیین ارتباطات موجود تا حد جزئیات است، همان گونه که توسط مشتری و کاربران بیان شده است.

ارزیابی و ترکیب

تحلیل گر باید تمام اشیای قابل مشاهده را تعریف کند، جریان و محتویات اطلاعات را ارزیابی کند، جزئیات تمام توابع نرم افزار را تعریف کرده، رفتار نرم افزار را در رابطه با وقایعی که بر سیستم اثر می گذارند تشخیص دهد، خصوصیات واسطه های سیستم را شناسایی کند و محدودیت های بیشتری از طراحی را آشکار کند. این کارها برای توصیف روش یا راه حلی برای مسئله صورت می گیرد که قابل ارزیابی و ترکیب باشد. در حین ارزیابی و ترکیب راه حل، تمرکز اولیه تحلیل گر بر "چه چیزی" و نه بر "چگونگی" است.

مدل سازی

ایجاد مدل داده ای، مدل عملکردی و رفتاری در این مرحله انجام می گیرد.

تعیین و شناسایی مشخصات مسئله

پیش بینی امکانات نرم افزار و حدود و ثغور آن در این مرحله انجام می شود.

بازنگری و کنترل

استقرار مکانیزم های کنترل مستمر و بازنگری در تمامی مراحل فوق به منظور دستیابی به کیفیت مناسب نرم افزار صورت می گیرد.

۲-۴ روش های جمع آوری نیازها

۱-۲-۴ روش های ارتباط با مشتری برای تحلیل نیازها

جمع آوری اطلاعات

- - مشاهده
- - مصاحبه
- - پرسشنامه
- - مطالعه و تحقیق
- - مناظره (توفان فکری)
- - برگزاری سمینار و کارگاه آموزشی
- - استفاده از ابزارهای مفید مانند نرم افزار

۲-۲-۴ روش هدایت شده تعیین مشخصه کاربرد (نیازها)

محققین گوناگون شیوه تیمی را برای جمع آوری نیازها و کاربردهای سیستم توسعه داده اند که در مراحل اولیه تحلیل و تعیین مشخصات نیازها به کار گرفته می شود. روش هدایت شده تعیین مشخصه کاربرد (FAST)^۱، ایجاد یک تیم شامل مشتری و توسعه دهنده را پیشنهاد می کند که با یکدیگر کار کنند تا مسئله و نیازها را شناسایی و راه حلها را مشخص کنند. FAST به طور عمده توسط جامعه سیستم های اطلاعاتی به کار گرفته شده است ولی قابلیت بهبود و استفاده در کلیه کاربردها را نیز دارد.

۳-۲-۴ استقرار تابع کیفیت

استقرار تابع کیفیت (QFD)^۲ یک فن مدیریت کیفیت است که نیازهای مشتری را به نیازهای فنی نرم افزار ترجمه می کند. QFD سه نوع نیاز را مشخص می کند:

نیازهای معمولی: اهدافی است که برای محصول یا سیستم در نظر گرفته شده اند. اگر این نیازها وجود داشته باشند، مشتری راضی است. مثال هایی از نیازهای معمولی عبارتند از درخواست نوعی صفحه نمایش رنگی، توابع خاص سیستم و سطوح کارایی تعریف شده و ...

نیازهای مورد انتظار: این نیازها برای محصول یا سیستم ضمنی هستند و ممکن است که مشتری به صراحت آنها را بیان نکند، فقدان آنها باعث نارضایتی فراوانی می شود. مثال هایی از نیازهای مورد انتظار عبارتند از: سهولت ارتباط آسان، صحت کلی و قابلیت اطمینان و سهولت نصب و ...

نیازهای ایده آل: این جنبه‌ها فراتر از انتظارات مشتری هستند و در صورت وجود، رضایت زیادی را به دنبال دارند. برای مثال نیازی که در حال حاضر قابل پیاده‌سازی نیست اما مشتری آن‌ها را اعلام می‌کند مثل تجارت الکترونیکی در ایران و ...

مواردی که در جلسات QFD اتفاق می‌افتد:

- گسترش و شناسایی وظایف^۱: وظایف سازمان چگونه تبدیل به توابع عملیاتی می‌شود؟
- گسترش و شناسایی اطلاعات^۲: اطلاعات موجود در فرم‌ها شامل چیست؟
- گسترش و شناسایی کار^۳: تبدیل فعالیت‌ها به اجرا چگونه صورت می‌گیرد؟
- گسترش و شناسایی مقادیر داده‌ها^۴: حدود مقادیر و ساختار و محتوای داده‌ها چیست؟

۴-۲-۴ تهیه مورد کاربرد

در ضمن جمع‌آوری نیازها به عنوان بخشی از جلسات غیر رسمی، FAST یا QFD، مهندس نرم‌افزار (یا تحلیل‌گر) می‌تواند مجموعه‌ای از سناریوها را که مشخص‌کننده مسیر استفاده از سیستم است ایجاد کند. این سناریوها اغلب مورد کاربرد یا Use-Case نامیده می‌شوند و توصیفی را از چگونگی استفاده سیستم فراهم می‌آورند.

برای ایجاد Use-Case، تحلیل‌گر باید ابتدا انواع مختلف افراد یا دستگاه‌ها را که Actors نامیده می‌شود، مشخص کند که با سیستم یا محصول تعامل داشته و از آن استفاده می‌کنند. این بازیگران اغلب نقش‌هایی (Role) را ایفا می‌کنند که افراد (یا دستگاه‌ها) به عنوان اپراتورهای سیستم عمل می‌کنند. با تعریفی رسمی‌تر، یک بازیگر هر چیزی است که با سیستم یا محصول ارتباط برقرار کرده و خارج از سیستم قرار دارد.

توجه به این نکته مهم است که یک بازیگر و یک کاربر مشابه نیستند. یک کاربر می‌تواند نقش‌های متفاوتی را هنگام استفاده از سیستم بازی کند، درحالی‌که یک بازیگر، یک گروه از موجودیت‌های خارجی (در اکثر موارد افراد) را مشخص می‌کند که فقط یک نقش را در سیستم بازی می‌کنند.

۴-۳ اصول تحلیل نیاز

در دو دهه گذشته، محققین مسایل مختلفی را تحلیل و علت آن‌ها را مشخص کرده و نشانه‌گذاری‌های متعدد مدل‌سازی را همراه با مجموعه ابتکاراتی برای غلبه بر آن‌ها توسعه داده‌اند. هر روش تحلیل دارای دیدگاه منحصربه‌فردی است. به هر حال، تمام روش‌های تحلیل توسط مجموعه‌ای از اصول عملیاتی مشترک به شرح زیر به هم مرتبط شده‌اند:

محدوده اطلاعات مسئله نمایش داده شود و قابل درک باشد.

توابعی که نرم‌افزار را اجرا می‌کنند باید تعریف شوند.

رفتار نرم‌افزار (در نتیجه وقایع خارجی) باید نشان داده شود.

مدل‌های اطلاعاتی، عملکردی و رفتاری باید به صورتی افراز شوند که جزئیات را به روش لایه‌ای (یا سلسله‌مراتبی) آشکار کنند.

فرایند تحلیل باید از اطلاعات پایه و ضروری به سمت جزئیات پیاده‌سازی (اجرایی) حرکت کند.

۴-۳-۱ شش اصل راهنمای DAVIS برای مدل‌سازی نیازها

- قبل از شروع به ساخت مدل تحلیل، درک درست مسئله الزامی است.
- یک نمونه از مسئله تهیه شود تا مشتری از نحوه ارتباط ماشین با انسان آگاه شود. نمونه تهیه‌شده می‌تواند دورانداختنی^۵ و یا تکاملی^۶ باشد.
- منبع و دلیل هر نیاز باید ثبت شود.
- به صورت‌های مختلف، نیازها نمایش داده شوند.
- نیازها باید اولویت‌بندی شوند.
- روی موارد مبهم کار شود تا حذف و یا به طور واضح و روشن بیان شوند.

1 - Function Deployment
2 - Information Deployment
3 - Task Deployment
4 - Value Deployment
5 - Throwaway Prototype
6 - Evolutionary Prototype

فصل پنجم

مدل سازی تحلیل نیازهای نرم افزار

در سطح فنی، مهندسی نرم افزار با یک سری از فعالیت های مدل سازی شروع می شود که به مشخصات کامل نیازها و نمایش طرحی جامع برای نرم افزار تبدیل می شوند. مدل تحلیل، شامل مجموعه ای از مدل هاست و اولین نمایش فنی از سیستم است.

۱-۵ تاریخچه فعالیت های عمده تحلیل نیاز

واژه تحلیل ساخت یافته که ابتدا توسط Douglas Ross استفاده شد، توسط DeMarco مشهور شد و بعدها توسط Page-Jones و سپس Sarson و Gane توسعه یافت. در دهه ۸۰ توسعه این روش برای سیستم های بلادرنگ توسط Ward و Mellor و سپس Hatley و Pirbhai گسترش یافت. این توسعه ها باعث قوی تر شدن انجام تحلیل شدند که می توانست به طور مؤثری در مسایل مهندسی به کار گرفته شود. تلاش هایی برای توسعه یک نشانه گذاری یکپارچه پیشنهاد و روش های مدرنی با استفاده از ابزارهای CASE نیز معرفی شدند. در حال حاضر، در مجموع دو متدولوژی متداول ساخت یافته^۱ و شیء گرا^۲ برای مدل سازی تحلیل سیستم ها وجود دارد. متداول ترین و آخرین روش ساخت یافته تحت عنوان "روش تحلیل و طراحی ساخت یافته سیستم" به نام SSADM^۳ معرفی شده است که کاربردهای زیادی در سازمان ها داشته است. در روش ساخت یافته نگرش سازماندهی شده و منظم سیستمی از بالا به پایین^۴ برای تحلیل و طراحی و نگرش پایین به بالا^۵ برای پیاده سازی نرم افزار مد نظر است. در این روش:

- مستندات قوی باعث بالا بردن قابلیت نگهداری می شود.
- اندازه مسئله با افراز مناسب تعیین می شود.
- از نماد گرافیکی برای نمایش روابط سیستم استفاده می شود.
- تمایز بین نمایش فیزیکی و منطقی وجود دارد.

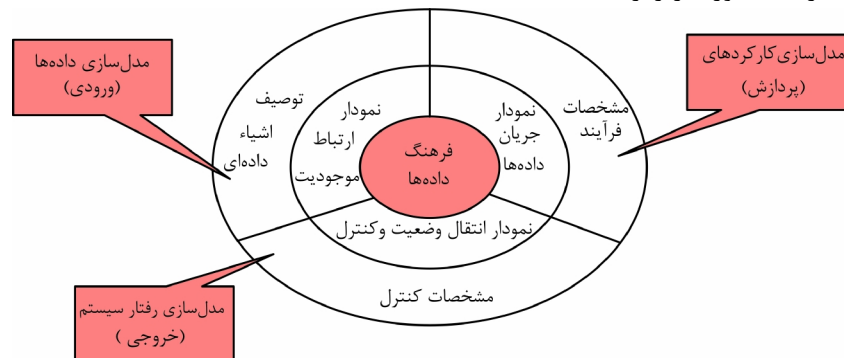
1 - Structured
2 - Object Oriented
3 - Structured System Analysis & Design Method
4 - Top Down
5 - Bottom Up

۲-۵ اهداف مدل تحلیل نیاز

- نیازهای مشتری را تشریح می کند.
- پایه و مبنایی برای مدل طراحی نرم افزار به وجود می آورد.
- مجموعه ای از نیازها را تعریف می کند که می توان درستی و اعتبار نرم افزار را آزمایش کرد.

۳-۵ عناصر مدل تحلیل نیاز

مدل پیشنهادی برای تحلیل نیازها به صورت زیر ارائه شده است:



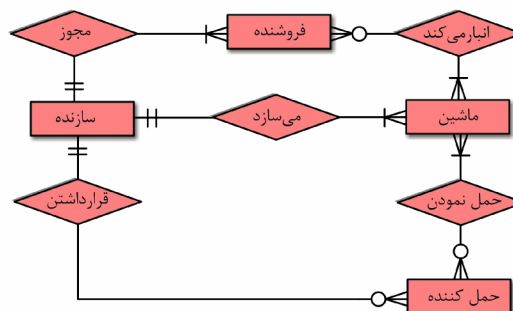
شکل ۱-۵ مدل تحلیل نیاز ساخت یافته

بنابراین مدل های اصلی تحلیل در روش ساخت یافته به شرح زیر هستند:

- نمودار ارتباط موجودیت (ERD)^۱: مدل سازی داده ها،
- نمودار جریان داده ها (DFD)^۲: مدل سازی کارکردها و
- نمودار انتقال حالت (STD)^۳: مدل سازی رفتار سیستم است.

۱-۳-۵ مدل سازی داده ها^۴

مدل داده شامل سه عنصر اطلاعاتی توصیف شده است: شیء داده ای، صفاتی که توصیف کننده شیء داده ای هستند و روابطی که اشیای داده را به یکدیگر مرتبط می کنند.



شکل ۲-۵ نمودار ارتباط موجودیت

1 - Entity Relationship Diagram
 2 - Data Flow Diagram
 3 - State Transition Diagram
 4 - Data Modeling

توجه: حداقل نتیجه حاصل از مدل سازی داده ای به دست آوردن ساختار و شکل جداول پایگاه داده ای است.

۲-۳-۵ مدل سازی کارکردها و جریان اطلاعاتی^۱

داده ها در ضمن جریان در سیستم رایانه ای توسط پردازشها تبدیل و دستکاری می شوند. سیستم، ورودی را به شکل های مختلف می پذیرد، سخت افزار، نرم افزار و عناصر انسانی را برای تبدیل آن به کار می گیرد و خروجی را به شکل های مختلف تولید می کند. توجه: در اینجا تمرکز بر حرکت و پردازش اطلاعات است که با عنوان جریان اطلاعات^۲ مطرح است.

• نمودار جریان داده ها

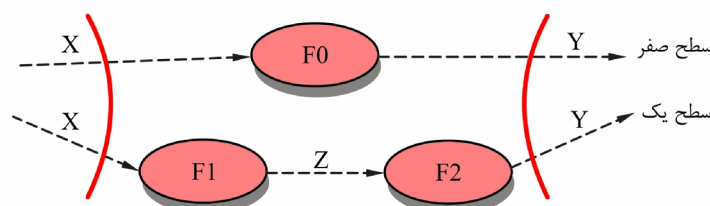
نمودار جریان داده (DFD) نمایشی گرافیکی است که جریان اطلاعات و تبدیلات را در ضمن حرکت داده ها از ورودی به خروجی نشان می دهد. شکل اصلی، نمودار جریان داده، گراف جریان داده، یا چارت حسابی نیز نامیده می شود. DFD سطح صفر سیستم، مدل زمینه نیز نامیده می شود که کل عنصر نرم افزار را به صورت یک حساب، همراه با داده های ورودی و خروجی نشان داده شده توسط پیکان های وارد شده و خارج شده از آن نشان می دهد. فرایندهای اضافی و مسیر جریان اطلاعات در ضمن تجزیه DFD سطح صفر برای نشان دادن جزئیات بیشتر، نمایش داده می شوند.

• نکات مربوط به DFD

- فقط گردش اطلاعات در DFD نشان داده می شود.
 - توالی، تأخر و تقدم اطلاعات در DFD مشخص نیست.
 - نحوه تبدیل داده ها در DFD مشخص نیست و باید توصیفی برای شرح پردازشها نوشت.
 - هیچ عمل فیزیکی در DFD نشان داده نمی شود.
 - در DFD کنترلها دیده نمی شوند و باید شرح کنترلها را جداگانه نوشت.
- توجه: مجموعه ای از پردازشها، فعالیت و مجموعه ای از فعالیتها، زیرسیستم و مجموعه ای از زیرسیستمها، سیستم نامیده می شود.

• توازن جریان در DFD

مهم ترین کار در رسم DFD شکستن سیستم به اجزای کوچکتر با حفظ توازن جریان بین سطوح مختلف تا حد تفصیلی (به طور منطقی و سلسله مراتبی) است. به مثال زیر توجه و آن را تحلیل کنید.



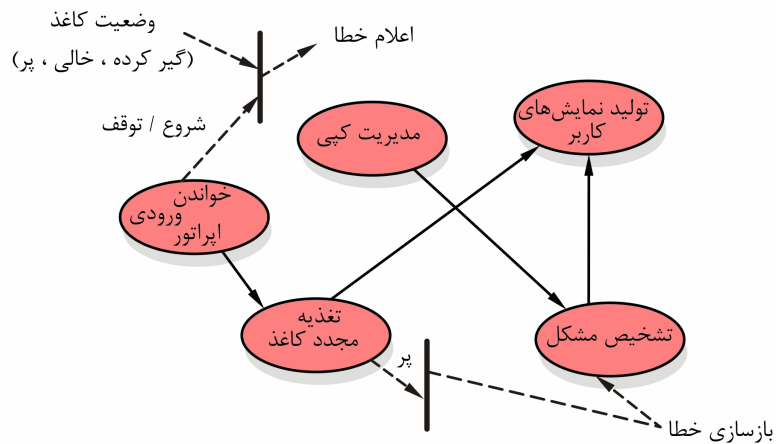
شکل ۳-۵ نحوه شکست نمودار جریان داده با حفظ توازن جریان

۳-۳-۵ مدل سازی رفتار سیستم با نمودار انتقال وضعیت

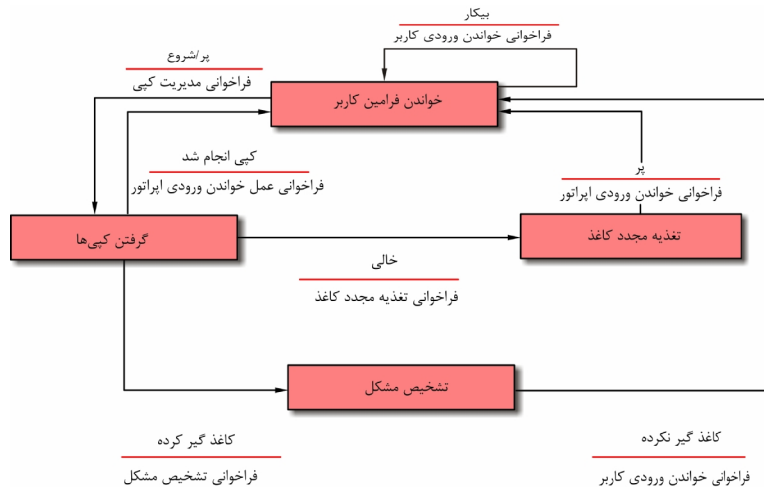
مدل سازی رفتار سیستم مبنای عملیاتی تمام روشهای تحلیل نیازهاست. نمودار انتقال و تغییر حالت (STD)، نشان دهنده رفتار سیستم، با استفاده از مشخص کردن حالتها و وقایعی است که باعث تغییر وضعیت سیستم می شوند. علاوه بر آن STD نشان می دهد که چه

1 -Functional Modeling
2 -Information Flow

عکس‌عمل‌هایی (برای مثال فعال‌سازی فرایند) در نتیجهٔ واقعهٔ خاصی باید انجام شود. در شکل‌های ۵-۸ و ۵-۹، نمودارهای جریان کنترل (CFD) و انتقال حالت (STD) مربوط به یک دستگاه کپی نشان داده شده است.



شکل ۴-۵ نمودار جریان کنترل دستگاه کپی



شکل ۵-۵ نمودار تغییر حالت برای نرم‌افزار دستگاه کپی

توجه: در نمودارهای جریان کنترل و انتقال وضعیت به موارد زیر توجه می‌شود:

- STD رفتار سیستم را در برخورد با رویدادهای مختلف که منجر به تغییر وضعیت سیستم می‌شود به تصویر می‌کشد.
- در STD حرکت سیستم از یک وضعیت به وضعیت دیگر نشان داده می‌شود.
- در STD نقاط کنترلی مشخص می‌شود که در نمودار جریان کنترل (CFD) نشان داده شده است.

رهنمودهای رسم ERD

مهندس نرم‌افزار از طریق نمودار ER، اقدام به شناسایی ورودی و خروجی‌های سیستم می‌کند بنابراین:

- ۱- در حین جمع‌آوری نیازها با مشتری، لیست اشیای مورد نظر را بنویسید.
- ۲- یک شیء را انتخاب و با مشتری ارتباط آن را با سایر اشیای تعیین کنید.
- ۳- هر زمان ارتباطی موجود بود به طور مشترک با مشتری این ارتباط را تعریف و ایجاد کنید.
- ۴- برای هر جفت ارتباط اشیا، چندی و الزام آن را تعیین کنید.
- ۵- مراحل ۲ تا ۴ را مرتب تکرار کنید تا تمام ارتباطات تعریف شوند.
- ۶- ویژگی هر موجودیت را تعیین و تعریف کنید.

- ۷- نمودار ارتباط موجودیت حاصل را مورد بازنگری قرار دهید.
- ۸- مراحل ۱ تا ۷ را تکرار کنید تا ERD کامل شود.

رهنمودهای رسم DFD

- ۱- DFD سطح صفر باید عملکرد کلان نرم افزار یا سیستم را به تصویر بکشد.
- ۲- ورودی و خروجی های اولیه باید به دقت تعریف شوند.
- ۳- با مجزا کردن پردازش ها، موجودیت ها و ذخایر داده ای، اقدام به تجزیه در سطوح بعدی کنید به طوری که امکان پالایش در هر سطح وجود داشته باشد.
- ۴- تمام خطوط اطلاعاتی باید با اسامی با معنی مشخص شوند.
- ۵- پیوستگی جریان اطلاعات در تمام سطوح باید حفظ شود (توازن جریان).
- ۶- در هر لحظه فقط یک بخش را تجزیه و پالایش کنید.

فصل ششم

اصول و قواعد طراحی نرم افزار

در حوزه مهندسی نرم افزار، طراحی بر چهار محور اصلی داده، معماری، واسط و پیمانه متمرکز است.

۱-۶ تعریفی از طراحی

فرایند به کارگیری تکنیک‌ها، اصول و قواعد مورد نیاز برای تعریف یک وسیله، فرایند، پردازش یا یک سیستم تا حد تفصیل است که با واقعیت فیزیکی آن تطبیق داشته باشد. مدل طراحی مبنایی را برای پیاده‌سازی نرم افزار به وجود می‌آورد. هر تغییری در اثنای پیاده‌سازی نرم افزار باید روی مدل طراحی و متعاقب آن روی مدل تحلیل نیاز اعمال شود.

۲-۶ طراحی نرم افزار و مهندسی نرم افزار

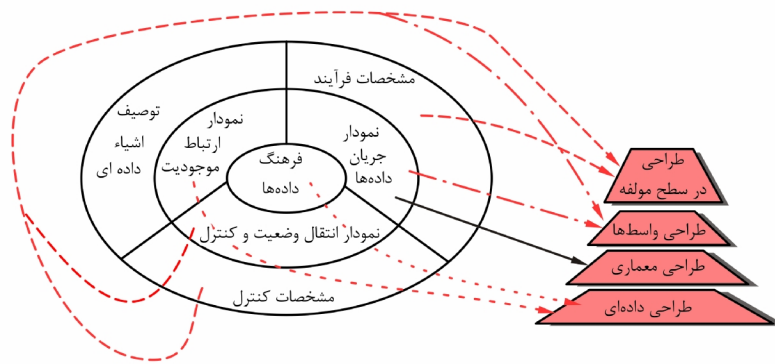
در آغاز و پس از تحلیل و تعیین نیازهای نرم افزار، طراحی نرم افزار با انجام سه فعالیت یعنی طراحی، تولید برنامه و آزمون، انجام می‌شود. هر یک از این فعالیت‌ها، اطلاعات را به گونه‌ای تغییر می‌دهند که در نهایت به نرم افزار معتبر رایانه‌ای منجر می‌شود. هر یک از عناصر مدل تحلیل (فصل ۵)، اطلاعات لازم را در ایجاد چهار مدل طراحی فراهم می‌آورند.

طراحی داده‌ها، مدل داده‌ای تولیدشده در مدل تحلیل را به ساختارهای داده‌ای لازم در اجرای نرم افزار تبدیل می‌کند. اشیا و روابط تعیین شده داده‌ها در نمودار ارتباط موجودیت‌ها و محتوای مشروح داده‌ای در فرهنگ داده‌ها، مبنای فعالیت طراحی داده‌ها را فراهم می‌کنند. بخشی از طراحی داده‌ها ممکن است با طراحی معماری نرم افزار همراه شود. البته طراحی جزئی‌تر داده‌ها همراه با طراحی هر یک از مؤلفه‌های نرم افزار صورت می‌گیرد.

طراحی معماری، رابطه بین عناصر اصلی مؤلفه‌های نرم افزاری را تعیین می‌کند، یعنی رابطه "الگوهای طراحی" به کاررفته در تحقق نیازهای سیستم و محدودیت‌های مؤثر بر شیوه اجرای الگوهای طراحی معماری تدوین می‌شود.

طراحی واسط، توصیف کننده نحوه ارتباط نرم افزار با سیستم‌ها و افرادی است که با آن تعامل داشته و آن را به کار می‌برند. طراحی مؤلفه (پیمانه)، عناصر ساختاری معماری نرم افزار را به توصیف رویه‌ای مؤلفه نرم افزاری تبدیل می‌کند. اطلاعات به دست آمده از STD و CSPEC، پایه و اساس طراحی مؤلفه به شمار می‌روند.

ارتباط عناصر مدل طراحی با مدل تحلیل در شکل ۱ نشان داده شده است.



شکل ۱-۶ برگردان مدل تحلیل به مدل طراحی نرم افزار

اهمیت طراحی نرم افزار را تنها با یک کلمه یعنی "کیفیت" می توان بیان کرد. طراحی روندی است که طی آن کیفیت فرایند مهندسی نرم افزار، بهبود می یابد. طراحی، نمونه هایی از نرم افزار را که از لحاظ کیفی قابل ارزیابی هستند، در اختیار ما قرار می دهد. این تنها راهی است که به واسطه آن می توانیم شکل درست نیازها و خواسته های مشتری را به یک محصول نرم افزاری یا سیستم تبدیل کنیم. طراحی نرم افزار فرایندی تکراری است که به موجب آن نیازها و ضرورت ها برای ساخت نرم افزار، تبدیل به یک "طرح یا نقشه" می شوند. در ابتدا، طراحی یک دید کلی از نرم افزار را در سطح بالایی از انتزاع نشان می دهد. با تکرار و پالایش در طراحی، نمایش طرحی در سطوح بسیار پایین تر از انتزاع حاصل می شود. این سطوح برای دستیابی به نیازهای قابل ردیابی، ملاک کار خواهند بود.

۱-۲-۶ معیارهای ارزیابی کیفیت یک طراحی خوب

- طراحی باید یک ساختار معماری ارائه کند که (۱) با استفاده از الگوهای قابل تشخیص طراحی ایجاد شده باشد. (۲) متشکل از اجزا و عناصری باشد که خصوصیات طراحی خوب را نشان می دهند (بعداً در این فصل مورد بحث قرار می گیرند و (۳) به شیوه ای تکاملی اجرا شده و بدین ترتیب اجرا و آزمون را تسهیل کند.
- طراحی باید پیمانهای^۱ باشد؛ یعنی نرم افزار باید به طور منطقی به اجزایی تقسیم شود که اعمال اصلی و فرعی خاصی را انجام دهند.
- طراحی باید نمایش های مجزایی از داده ها، معماری، واسطها و پیمانها را دربرداشته باشد.
- طراحی باید منجر به ساختارهای داده ای شود که برای پیاده سازی اشیا مناسب بوده و از الگوهای قابل تشخیص داده ها ناشی شوند.
- طراحی باید به اجزایی منتهی شود که خصوصیات مستقل کاربردی را نمایش دهند.
- طراحی باید به واسطه ای ختم شود که از پیچیدگی روابط بین پیمانها و محیط خارجی می کاهند.
- طراحی باید حاصل کاربرد یک شیوه قابل تکرار با استفاده از اطلاعات به دست آمده در اثنای تحلیل نیازهای نرم افزاری باشد.

۳-۶ اصول طراحی

طراحی نرم افزار، هم یک فرایند و هم یک مدل است. Davis مجموعه اصولی را برای طراحی نرم افزار به شرح زیر پیشنهاد می کند:

- باریک بینی نباید در فرایند طراحی وجود داشته باشد.
- طراحی باید قابل ردیابی به مدل تحلیل نیاز باشد.
- طراحی نباید دوباره کاری باشد.
- طراحی باید فاصله منطقی بین نرم افزار و مسئله موجود در جهان واقعی را به حداقل برساند.
- طراحی باید از یکنواختی و یکپارچگی برخوردار باشد.
- ساختار طراحی باید پذیرای تغییر باشد.
- ساختار طراحی حتی در صورت مواجهه با داده های غیر عادی، رویدادها یا شرایط کاری، باید به آرامی از کار بایستد.
- طراحی باید ضمن شکل گیری و نه بعد از اتمام آن، از نظر کیفی مورد ارزیابی قرار گیرد.
- طراحی باید برای به حداقل رساندن خطاهای مفهومی (معنایی) مرور و بررسی شود.
- طراحی به معنای برنامه نویسی نیست و برنامه نویسی نیز معادل طراحی نیست.

۴-۶ قواعد طراحی

قواعد نه گانه طراحی زیر در ارائه پاسخ سوالات زیر کمک می کنند:

- چه معیارهایی برای تقسیم بندی نرم افزار به مؤلفه های مجزا به کار گرفته شده است؟
- چگونه کارکردها یا ساختار داده ها از نمایش مفهومی نرم افزار مجزا می شوند؟
- آیا معیار یکنواختی وجود دارد که کیفیت طراحی نرم افزار را تعریف و ارزیابی کند؟
- هدف از نوشتن برنامه، کارکردن آن یا درست کار کردن آن است؟ توجه داشته باشید که هدف از نوشتن برنامه درست کارکردن آن است و برای این کار:

تحلیل سیستم درست \Leftarrow تحلیل نیاز درست \Leftarrow طراحی خوب

۱-۴-۶ انتزاع

وقتی برای هر مسئله به دنبال راه حل پیمانه ای باشیم، سطوح انتزاع زیادی نیز مطرح می شوند. در بالاترین سطح انتزاع، راه حل با به کارگیری زبان رایج در محیط مسئله و به صورت کلی بیان می شود. در سطوح پایین تر انتزاع، جهت گیری و گرایش بیشتر رویه ای است. حین پیش روی در سطوح مختلف انتزاع، تلاش می کنیم تا انتزاع های رویه ای، داده ای و کنترلی ایجاد شوند.

انتزاع رویه ای توالی مشخصی از دستورالعمل هاست که عملکرد خاص و محدودی دارد.

انتزاع داده ای مجموعه ای از داده های با نام است که یک شیء داده ای را توصیف می کند.

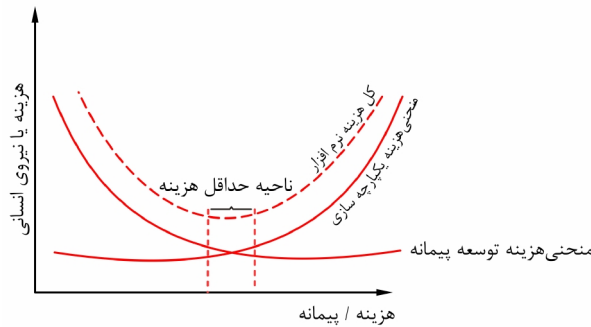
انتزاع کنترلی سومین شکل انتزاعی است که در طراحی نرم افزار به کار می رود. همانند انتزاع رویه ای و داده ای، انتزاع کنترلی بر مکانیزم کنترل برنامه بدون مشخص کردن جزئیات داخلی اشاره دارد.

۲-۴-۶ پالایش

پالایش گام به گام یک راهبرد طراحی بالا به پایین است. برنامه با سطوح پالایشی متوالی جزئیات رویه ای، توسعه می یابد. تا زمان دستیابی به دستورات زبان برنامه نویسی، توسعه سلسله مراتب با تجزیه عملکردها (انتزاع رویه ای) طی شیوه ای گام به گام صورت می گیرد.

۳-۴-۶ پیمانهای

معماری یک نرم افزار، وقتی پیمانهای است که در آن نرم افزار به اجزای نشانی پذیر با اسامی جداگانه به نام "پیمانها" تقسیم شده باشد و سپس برای رفع نیازهای مسئله، یکپارچه و مجتمع شوند. نمودار زیر رابطه بین تعداد پیمانها در یک نرم افزار و هزینه های متناظر را نشان می دهد. با افزایش پیمانها هزینه یکپارچگی و در نهایت کل هزینه نرم افزار افزایش می یابد. بنابراین تعیین تعداد پیمانهای مطلوب در طراحی نرم افزار مهم و عموماً به صورت شهودی و بر اساس تجربه صورت می گیرد.



شکل ۲-۶ پیمان شدن و هزینه نرم افزار

Meyer پنج معیار را در ارزیابی یک شیوه طراحی و بر اساس توانایی آن در تعریف یک سیستم مؤثر پیمانهای معرفی می کند:

- تجزیه پذیری پیمانهای، (کاهش پیچیدگی کل)
- قابلیت ترکیب پیمانهای، (استفاده از پیمانهای با قابلیت استفاده مجدد)
- قابلیت درک پیمانهای، (پیمانها مستقل و قابل تغییر)
- پایداری پیمانهای، (قابلیت توسعه)
- حفاظت پیمانهای، (قابلیت عدم انتشار خطا)

۴-۴-۶ معماری نرم افزار

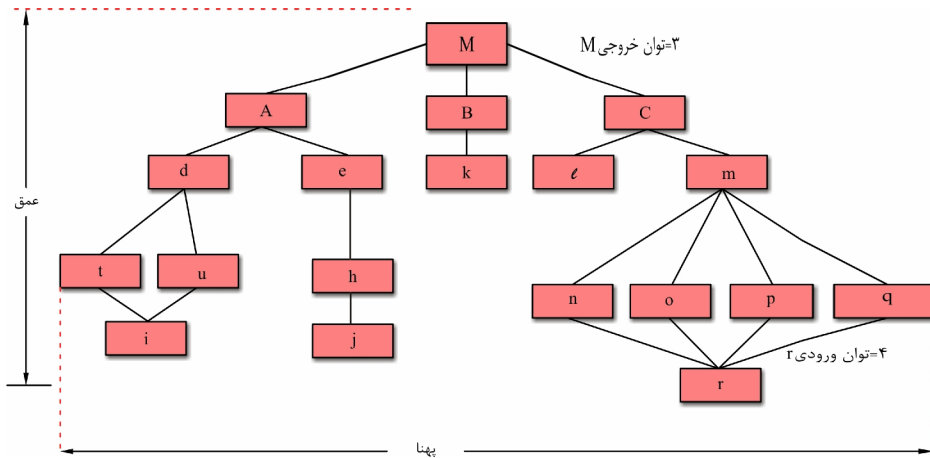
معماری نرم افزار به "ساختار کلی نرم افزار و راههای ایجاد یکپارچگی ذهنی سیستم از طریق این ساختار" اشاره می کند. معماری در ساده ترین شکل خود عبارت است از: ساختار سلسله مراتبی اجزای برنامه (پیمانها)، شیوه ارتباط این اجزا و ساختار داده هایی که توسط این اجزا مورد استفاده قرار می گیرند. معماری نرم افزار باید از مدل نیاز استخراج شود.

۵-۴-۶ سلسله مراتب کنترل در برنامه ها

"سلسله مراتب کنترل" که "ساختار برنامه" نیز نام دارد، بیانگر سازماندهی اجزای برنامه (پیمانها) بوده و بر سلسله مراتب کنترل نیز دلالت دارد. دو مفهوم زیر در این ارتباط تعریف می شوند:

- توان خروجی مقیاس تعداد پیمانهای است که مستقیماً توسط پیمانهای دیگر کنترل می شوند.
 - توان ورودی بیانگر تعداد پیمانهای است که یک پیمان خاص را به طور مستقیم کنترل می کنند.
- رابطه کنترلی میان پیمانها به شیوه زیر بیان می شود:

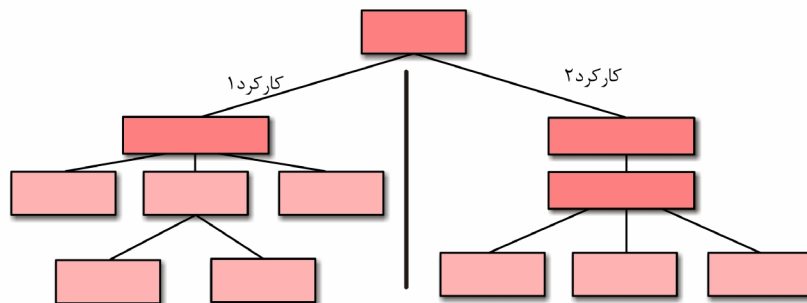
پیمانهای که پیمان دیگری را کنترل می کند، پیمان **حاکم** نام دارد و برعکس پیمان تحت کنترل پیمان دیگر، پیمان **کنترل شده** نامیده می شود.



شکل ۳-۶ ساختار سلسله‌مراتبی پیمانه‌ها در برنامه

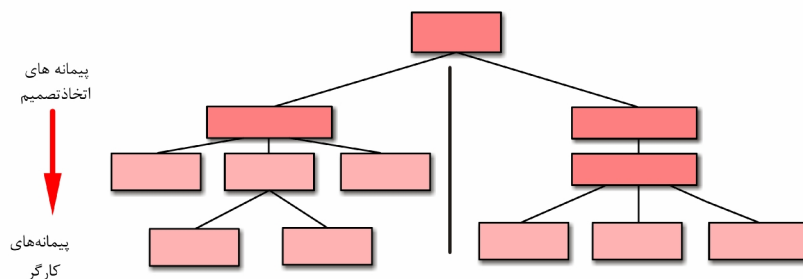
۶-۴-۶ تجزیه ساختاری

اگر سبک معماری یک سیستم، سلسله‌مراتبی باشد، می‌توان ساختار برنامه را، هم به صورت افقی و هم به طور عمودی تجزیه و افراز کرد. با توجه به شکل ۴ (۱)، افراز افقی، شاخه‌های جداگانه سلسله‌مراتب پیمانه‌های را برای هر یک از وظایف اصلی برنامه تعیین می‌کند. پیمانه‌های کنترلی که با سایه تیره‌تر نشان داده شده‌اند، برای هماهنگی ارتباط بین وظایف و اجرای آن‌ها به کار می‌روند. ساده‌ترین شیوه افراز افقی، سه بخش ورودی، تغییر و تبدیل داده‌ها (که اغلب فرایند نام دارد) و خروجی را تعیین می‌کند. از آنجاکه کارکردهای اصلی از یکدیگر جدا می‌شوند، اعمال تغییرات، پیچیدگی کمتری دارد و بسط و توسعه سیستم بدون تأثیرات جانبی، راحت‌تر انجام می‌شود. از جنبه منفی، افراز افقی باعث می‌شود داده‌های بیشتری از واسطه‌های پیمانه عبور کرده و کنترل کلی گردش برنامه دشوار و پیچیده شود.



شکل ۴-۶ (۱) افراز عمودی

افراز عمودی که فاکتورگیری نیز نامیده می‌شود، پیشنهاد توزیع کنترل (تصمیم‌گیری) و کارها را به صورت از بالا به پایین می‌کند. پیمانه‌های سطح بالا باید توابع کنترل را اجرا کنند و کار پردازش واقعی کمتری را انجام دهند. پیمانه‌های سطح پایین متولی اجرای پردازش‌ها هستند و پیمانه‌های کارگر نامیده می‌شوند.



شکل ۴-۶ (۲) افراز عمودی

۶-۴-۲ ساختمان داده‌ها

ساختار داده‌ها، نمایش رابطه منطقی میان عناصر جداگانه داده‌هاست. از آنجاکه ساختار اطلاعات بر طراحی نهایی رویه‌ای تأثیر خواهد داشت، ساختار داده‌ها به اندازه ساختار برنامه در نمایش معماری نرم افزار اهمیت دارد. ساختار داده‌ها تعیین کننده نحوه سازماندهی، روش‌های دستیابی، میزان اتصال و راه‌های مختلف پردازش اطلاعات را نمایش می‌دهد.

۶-۴-۸ پردازش‌های نرم افزار

ساختار برنامه، سلسله مراتب کنترل را بدون توجه به توالی پردازش‌ها و تصمیمات تعیین می‌کند. رویه نرم افزار بر جزئیات پردازشی هر پیمانانه به طور جداگانه تأکید دارد. رویه باید مشخصه دقیق پردازش از جمله توالی رویدادها، نقاط دقیق تصمیم‌گیری، اعمال تکراری و حتی سازماندهی / ساختار داده‌ای را ارائه دهد.

۶-۴-۹ پنهان کردن اطلاعات

مفهوم پیمانانه‌ای بودن، یک سؤال اساسی را برای هر طراح نرم افزاری مطرح می‌کند. «برای دستیابی به بهترین مجموعه پیمانانه‌ها، چگونه یک راه حل نرم افزاری را تجزیه کنیم؟» اصل پنهان سازی اطلاعات^۱ بیانگر آن است که "وجه مشخصه پیمانانه‌ها، تصمیمات طراحی است که هر پیمانانه را از پیمانانه‌های دیگر مخفی می‌سازد". به عبارتی دیگر، پیمانانه‌ها باید طوری طراحی و مشخص شوند که اطلاعات (رویه و داده‌ها) موجود در هر پیمانانه برای پیمانانه‌های دیگری که به چنین اطلاعاتی نیاز ندارند، قابل دسترسی نباشند.

۶-۵ استقلال عملیاتی

مفهوم "استقلال عملیاتی" پیامد مستقیم طراحی پیمانانه‌ای، مفاهیم انتزاع و پنهان سازی اطلاعات است. استقلال عملیاتی از طریق ایجاد پیمانانه‌هایی با عملکرد یک منظوره و عدم ارتباط بیش از حد با پیمانانه‌های دیگر، تحقق می‌یابد. حال سؤال این است که چرا استقلال عملیاتی با اهمیت است؟ زیرا این کار باعث:

- توسعه آسان تر
- دارای واسطه‌های ساده
- ساده تر شدن در نگهداری و آزمایش پیمانانه‌های مستقل
- کاهش انتشار خطا
- و قابلیت استفاده مجدد می‌شود.

استقلال عملیاتی با دو معیار کیفی انسجام و اتصال ارزیابی می‌شود که در ادامه به آن‌ها خواهیم پرداخت.

۶-۵-۱ انسجام^۲

انسجام یا چسبندگی، بسط طبیعی مفهوم پنهان سازی اطلاعات است. یک پیمانانه یکپارچه، یک وظیفه منفرد را در یک رویه نرم افزاری با برقراری ارتباط محدود با رویه‌های در حال اجرای سایر بخش‌های برنامه، انجام می‌دهد. به بیان ساده‌تر، یک پیمانانه منسجم (به طور ایده‌آل) باید تنها یک کار را انجام دهد. در طراحی باید به انسجام بالا دست یافت.

انواع انسجام

- انسجام عملیاتی^۳: یک پیمانانه فقط یک کار را به طور انحصاری انجام می‌دهد و نتیجه را بازمی‌گرداند.
- انسجام لایه‌ای^۴: در سطح زیر سیستم^۵، مؤلفه و کلاس، طراحی و پیاده‌سازی می‌شود. در این حالت مؤلفه سطح بالاتر به مؤلفه سطح زیرین دسترسی داشته ولی برعکس آن امکان پذیر نیست.

1 - Information Hiding
2 - Cohesion
3 - Functional Cohesive
4 - Layer Cohesive
5 - Pakage

- انسجام ارتباطی^۱: همهٔ عناصر پردازشی به یک ناحیه از ساختمان داده‌ها تمرکز می‌یابند. به تعبیر دیگر تمامی عملیات امکان دسترسی به دادهٔ یکسان را دارند که در یک کلاس داده تعریف شده است.
- انسجام متوالی^۲: عملیات مؤلفه‌ها پشت سر هم اجرا می‌شوند. در واقع هدف پیاده‌سازی یک‌سری عملیات پشت سر هم است. در این حالت خروجی مؤلفهٔ اول، ورودی مؤلفه دوم را تأمین می‌کند.
- انسجام رویه‌ای^۳: در این حالت سطوح میانه‌ای از انسجام از لحاظ استقلال پیمانه‌ها وجود دارند. مؤلفه‌ها و عملیات یکی پس از دیگری حتی بدون رد و بدل شدن داده اجرا می‌شوند. بدین ترتیب عناصر پردازشی یک پیمانه با هم ارتباط داشته و باید به ترتیب مشخصی انجام شوند.
- انسجام موقتی^۴: هنگامی که پیمانه‌ای شامل وظایفی است که باید در یک گسترهٔ زمانی مشترک اجرا شود، گوییم پیمانه دارای انسجام موقتی است. این عملیات فقط منعکس‌کنندهٔ یک وضعیت و یا یک حالت از سیستم مانند اجرای عملیاتی در زمان وقوع خطا در سیستم است.
- انسجام ابزاری^۵: در انتهای پایینی (نامطلوب) طیف، با پیمانه‌هایی سر و کار داریم که مجموعه‌ای از وظایف را انجام می‌دهند که ارتباط محکمی با هم ندارند.

۲-۵-۶ اتصال^۶

اتصال، مقیاس ارتباط بین پیمانه‌ها در ساختار نرم‌افزار است. اتصال به پیچیدگی واسط بین پیمانه‌ها، نقطهٔ ورود و ارجاع به یک پیمانه و نوع داده‌های عبوری از واسط، بستگی دارد. در طراحی نرم‌افزار، تلاش در دستیابی به پایین‌ترین سطح ممکن اتصال است.

• انواع اتصال

- اتصال محتوای^۷: بالاترین درجهٔ اتصال، اتصال محتویات است و هنگامی رخ می‌دهد که یک پیمانه داده‌های داخلی پیمانهٔ دیگر را تغییر دهد. از این نوع اتصال می‌توان و باید حذر کرد.
- اتصال مشترک^۸: زمانی که تعدادی پیمانه از یک مجموعه دادهٔ سرتاسری استفاده و آن‌ها را آدرس‌دهی می‌کنند، اتصال بالایی رخ می‌دهد. در این حالت کنترل تغییرات داده بسیار مشکل خواهد بود.
- اتصال کنترلی^۹: در سطوح میانی، اتصال با تبادل کنترل بین پیمانه‌ها مشخص می‌شود. اتصال کنترلی در اکثر طراحی‌های نرم‌افزاری بسیار متداول است. اگر Flag کنترلی در پیمانهٔ بعدی تغییر یابد، مشکل جدی ایجاد می‌شود.
- اتصال برجستگی^{۱۰}: هنگامی رخ می‌دهد که بخشی از ساختمان داده‌ها (به‌جای آرگومان‌های ساده) از طریق یک واسط پیمانه عبور داده شوند.
- اتصال داده‌ای^{۱۱}: مادامی که لیستی از آرگومان‌های ساده وجود داشته باشند (یعنی داده‌های ساده عبور داده شوند، و بین عناصر موجود، تناظر یک به یک برقرار باشد، در این بخش از برنامه اتصال اندکی موسوم به اتصال ساختاری یا داده‌ای به چشم می‌خورد.
- اتصال نوع داده^{۱۲}: وقتی مؤلفهٔ A قادر به استفاده از Data Type تعریف شده در مؤلفهٔ B باشد این اتصال رخ می‌دهد. اگر تعریف Type تغییر کند باید تمام کلاس‌ها و مؤلفه‌هایی که از این Type استفاده می‌کنند، تغییر یابند.
- اتصال خارجی^{۱۳}: سطوح نسبتاً بالای اتصال هنگامی رخ می‌دهد که پیمانه‌ها با محیط خارجی نرم‌افزار ارتباطی تنگاتنگ داشته باشند. اتصال خارجی ضروری است، ولی باید به تعداد اندکی از پیمانه‌ها با یک ساختار مشخص، محدود شوند.

1 - Communicational Cohesive

2 - Sequential Cohesive

3 - Procedural Cohesive

4 - Temporal Cohesive

5 - Utility Cohesive

6 - Coupling

7 - Content Coupling

8 - Common Coupling

9 - Control Coupling

10 - Stamp Coupling

11 - Structural Coupling

12 - Data Type Coupling

13 - External Coupling

فصل هفتم

طراحی معماری نرم افزار

۱-۷ چرا معماری؟

معماری یک نرم افزار نمودی است که مهندس نرم افزار را قادر می کند:

- میزان تأثیر طرح را در مرتفع کردن نیازهای بیان شده، تحلیل کند.
- معماری های جایگزین را در مرحله ای که تغییر طرح هنوز نسبتاً آسان است، بررسی کند.
- خطرات مربوط به ساخت نرم افزار را کاهش دهد.

طراحی داده باعث نمایش مؤلفه داده ای معماری و طراحی معماری بر تمرکز نمایش ساختار مؤلفه های نرم افزار، خواص آن ها و ارتباط بین آن ها قرار دارد.

۲-۷ طراحی داده ها

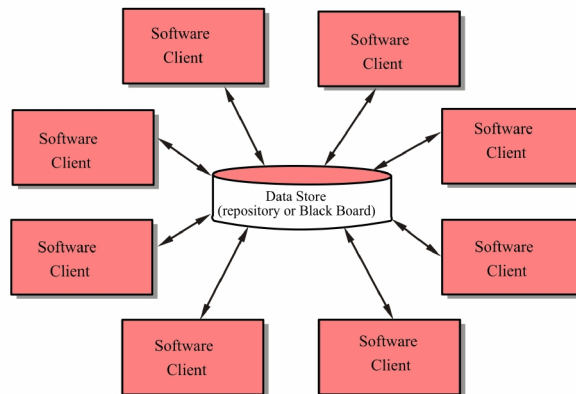
طراحی داده ها، همچون دیگر فعالیت های مهندسی نرم افزار (که گاهی به آن معماری داده ها نیز می گویند) مدلی از داده ها یا اطلاعات را در سطح بالایی از انتزاع، ایجاد می کند. سپس مدل داده ای به صورت بازنمای خاص پیاده سازی درمی آید که می توان آن را با سیستم مبتنی بر رایانه پردازش کرد. در بسیاری از برنامه های کاربردی، معماری داده ها تأثیر شگرفی بر معماری نرم افزار دارد که باید آن را پردازش کند؛ بنابراین مدل داده ای:

- اشیای داده ای را پالایش کرده و مجموعه ای از انتزاعات داده ای را توسعه می دهد.
- صفات اشیای داده ای را به عنوان یک یا چند ساختمان داده ای پیاده سازی می کند.
- ساختمان های داده ای برای اطمینان از ارتباط های مناسب بازنگری می شوند.
- ساختمان داده ها تا حد ممکن ساده سازی می شود.

طراحی مبتنی بر داده ها یک روش طراحی معماری نرم افزار است که به راحتی امکان گذر از مرحله تحلیل نیاز به طراحی توصیف ساختمان برنامه را می دهد.

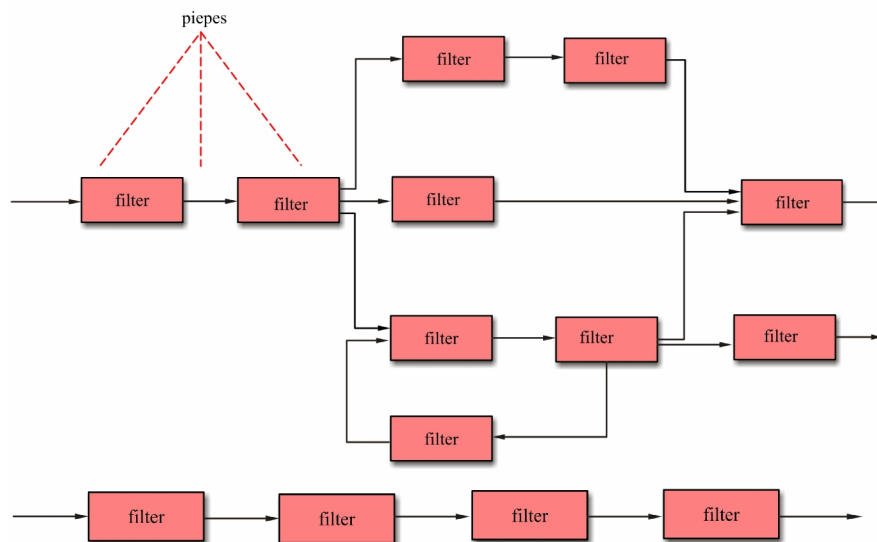
۳-۷ سبک‌های معماری نرم افزار

معماری مبتنی بر مخزن داده‌ها^۱: یک مخزن داده‌ای در مرکز این معماری قرار دارد و اغلب توسط دیگر اجزایی که به‌هنگام‌سازی، افزودن، حذف یا کارهای دیگر اصلاحی را در مورد مخزن انجام می‌دهند، قابل دسترسی است.



شکل ۱-۷ معماری داده‌گرا

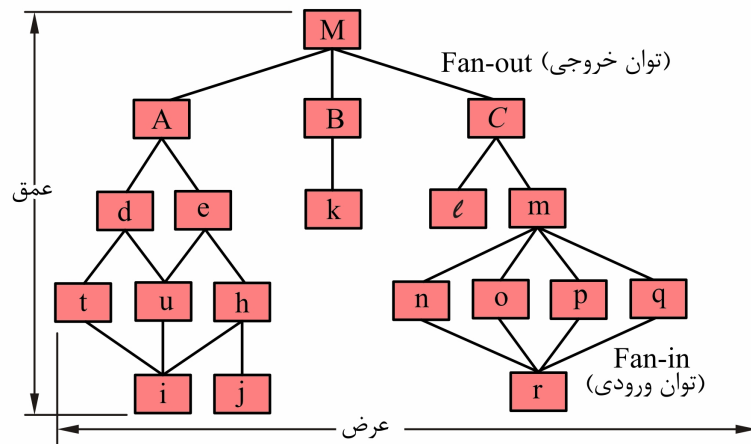
معماری مبتنی بر جریان داده^۲: این معماری وقتی به کار گرفته می‌شود که قرار است داده‌های ورودی از طریق اجرای یک‌سری پردازش‌های محاسباتی و اعمال تغییرات، به داده‌های خروجی تبدیل شوند.



شکل ۲-۷ معماری داده‌گرا

معماری فراخوانی و بازگشت^۳: این سبک از معماری، طراح نرم‌افزار (معمار سیستم) را قادر می‌کند تا به ساختار برنامه‌ای دست یابد که از نظر اصلاح و ارزیابی نسبتاً ساده است.

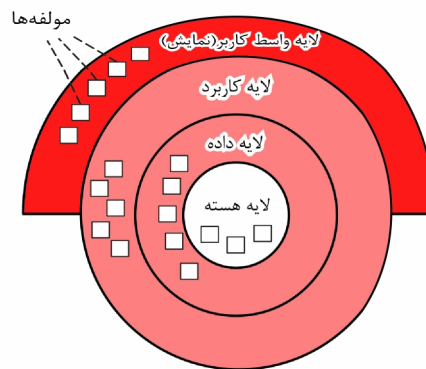
1 - Data Store Architecture
 2 - Data flow Architecture
 3 - Call and Return Architecture



شکل ۳-۷ معماری فراخوانی و برگشت

معماری‌های شیء‌گرا^۱: اجزای یک سیستم دربرگیرنده داده‌ها و عملیاتی هستند که باید برای تغییر داده‌ها مورد استفاده قرار گیرند. ارتباط و هماهنگی بین اجزا از طریق عبور پیام‌ها حاصل می‌شود (این معماری در فصول آتی مورد بحث قرار می‌گیرد).

معماری‌های لایه‌ای^۲: ساختار اصلی این معماری در شکل زیر نشان داده شده است. تعدادی لایه مختلف تعریف شده‌اند که هر کدام به عملیاتی دست می‌یابند که به طور گسترده‌ای به مجموعه دستورات ماشین نزدیک‌تر می‌شوند. در لایه خارجی‌تر، اجزا در خدمت عملیات واسط کاربر هستند. در لایه داخلی‌تر، اجزا وظیفه ارتباط با سیستم عامل را انجام می‌دهند. لایه‌های میانی خدمات استفاده، بهره‌برداری و عملیات کارکردی نرم‌افزار را مهیا می‌کنند.



شکل ۴-۷ معماری لایه‌ای

۱-۳-۷ پیچیدگی معماری

یک روش مفید برای ارزیابی پیچیدگی کلی معماری پیشنهادی، عبارت است از در نظر گرفتن وابستگی میان اجزای درون معماری. این وابستگی‌ها ناشی از جریان کنترل / اطلاعات درون سیستم هستند. ZHAO سه نوع وابستگی ارائه می‌دهد:

- **وابستگی‌های مشترک:** نمایانگر ارتباطات وابسته‌ای در میان مصرف‌کنندگانی هستند که از منبعی یکسان استفاده کرده یا تولیدکنندگانی که برای مصرف‌کنندگانی یکسان تولید می‌کنند. مثلاً، در مورد دو جزء u و v اگر u و v هر دو به یک‌سری داده سرتاسری یکسانی رجوع کنند، یک واسطه وابستگی مشترک بین u و v وجود دارد.

1 - Object-oriented architecture
2 - Layered architecture

- وابستگی‌های جریان: نمایانگر ارتباطات وابستگی میان تولیدکننده و مصرف‌کننده‌های منبع است. در مورد دو جزء u و v اگر باید u قبل از جریان یافتن کنترل در v تکمیل شود یا اگر u به وسیله پارامترهایی با v ارتباط برقرار کند، در این صورت یک جریان وابستگی میان این دو وجود دارد.
- وابستگی‌های محدودشده: نمایانگر محدودیت‌ها و قیودی در جریان نسبی کنترل میان مجموعه‌ای از فعالیت‌ها هستند. مثلاً در مورد دو جزء u و v ، آن‌ها نمی‌توانند در یک زمان اجرا شوند؛ پس یک واسطه وابستگی محدود و مقیدشده بین آن‌ها وجود دارد. وابستگی‌های مشترک و جریان که مورد توجه ZAHO قرار گرفت از بعضی جهات شبیه مفهوم انسجام و اتصال هستند.

۴-۷ نگاشت نیازها به معماری نرم افزار

طراحی ساخت‌یافته اغلب به عنوان یک روش طراحی مبتنی بر جریان داده معرفی می‌شود؛ زیرا به کمک آن می‌توان به راحتی از نمودار جریان داده به معماری نرم‌افزار دست یافت. گذار از جریان داده‌ها به ساختار برنامه با انجام یک فرایند شش مرحله‌ای زیر صورت می‌گیرد:

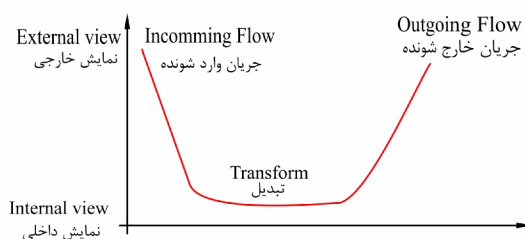
- تعیین نوع جریان اطلاعات
- مشخص کردن مرز جریان
- نگاشت نمودار جریان داده به ساختمان برنامه
- تعیین سلسله‌مراتب کنترل
- پالایش و بازنگری معماری حاصل با استفاده از معیارها، اصول و قواعد طراحی
- توصیف و تدوین معماری نهایی

۱-۴-۷ تعیین نوع جریان اطلاعات

دو نوع جریان تبدیلی و تراکنشی اطلاعات وجود دارد که در ادامه به نحوه نگاشت آن‌ها به معماری نرم‌افزار می‌پردازیم.

۱- جریان تبدیلی^۱

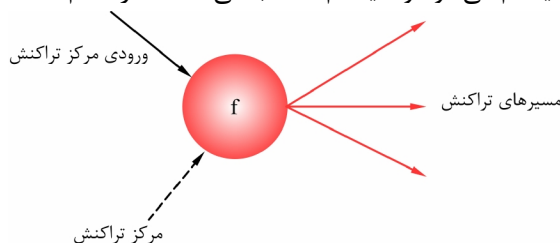
در جریان تبدیلی روال به صورت زیر در شکل ۵-۷ نشان داده شده است:



شکل ۵-۷ جریان تبدیلی

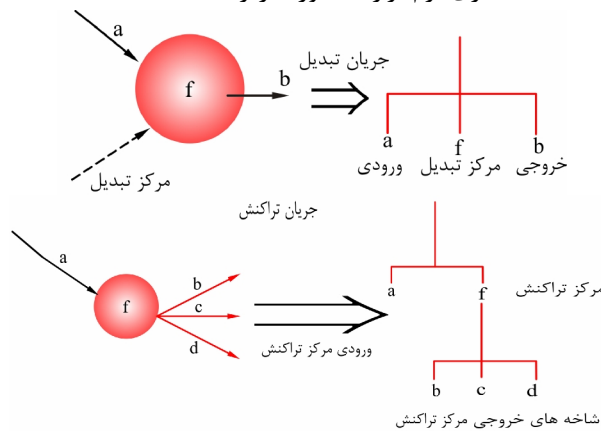
۲- جریان تراکنشی^۲

در جریان تراکنشی، جریانی وارد سیستم می‌شود و سیستم انتخاب می‌کند که از کدام شاخه خارج شود.



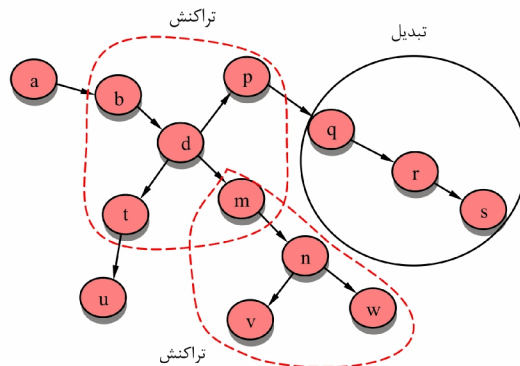
شکل ۶-۷ نمودار جریان تراکنشی

مثال ۱ دو نمونه از نحوه تبدیل جریان داده به معماری نرم افزار به صورت زیر است:



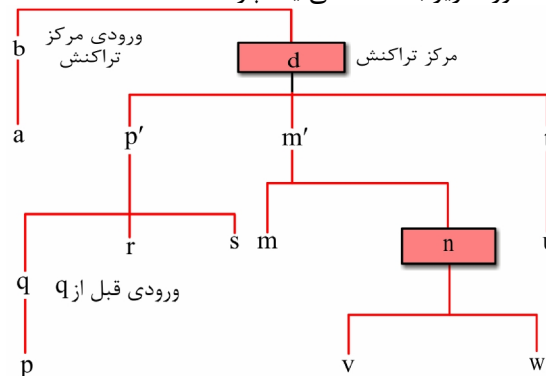
شکل ۷-۷ نحوه نگاشت نمودار جریان داده تبدیلی و تراکنشی به معماری

مثال ۲ معماری (ساختار برنامه) متناظر با DFD داده شده را به دست آورید.



شکل ۸-۷ نموداری شامل جریان های تبدیلی و تراکنشی

جواب: در نمودار جریان داده ای بالا d مرکز تراکنشی است مگر آنکه شرحی مبنی بر رخداد همزمان تمام خروجی ها از d ارائه شده باشد؛ بنابراین معماری متناظر با DFD داده شده به صورت زیر به دست می آید. چرا؟



شکل ۹-۷ نگاشت جریان داده به ساختار برنامه

استفاده موفق از نگاشت تراکنش یا تبدیل با کارهای اضافی دیگری تکمیل می شود که به عنوان بخشی از طراحی معماری ضروری هستند. بعد از طراحی معماری (ساختار) برنامه و اصلاح آن، کارهای زیر نیز باید انجام گیرد:

- گزارشی از پردازش های هر پیمانانه ارائه شود.
- توضیحی در مورد واسطه های هر پیمانانه داده شود.
- ساختارهای داده ای محلی و سراسری تعریف شوند.
- تمام محدودیت های طراحی ذکر شوند.
- مجموعه ای از بازنگری طراحی ها انجام شده باشد.
- و در صورت نیاز امکان پالایش در طراحی در نظر گرفته شود.

فصل هشتم

طراحی واسط کاربر

طراحی واسط بین اجزای نرم افزار
طراحی واسطها بین نرم افزار و سایر تولیدکنندگان و مصرف کنندگان اطلاعات غیر انسانی (یعنی اشیای خارجی)
طراحی واسط بین انسان (یعنی کاربر) و رایانه
در این فصل صرفاً بر سومین مقوله طراحی واسطها یعنی طراحی واسط کاربر توجه خواهیم داشت.

۱-۸ قواعد طلایی Mendal

Mental در کتاب خود با عنوان طراحی واسط، سه "قانون طلایی" زیر را مطرح می کند. این قوانین طلایی، عملاً به عنوان مجموعه‌ای از اصول طراحی واسط کاربر مطرح هستند که فعالیت مهم طراحی واسطهای نرم افزاری را هدایت می کنند.

۱-۱-۸ واگذاری کنترل به کاربر

در این قانون رعایت نکات زیر الزامی است:

- تعیین شیوه‌های تعاملی به نحوی که کاربر را مجبور به اعمال غیر ضروری یا نامطلوب نکند.
- ایجاد تعامل انعطاف‌پذیر در ارتباط با کاربر.
- امکان ایجاد وقفه و خنثی‌سازی (بازگشت) در تعامل با کاربر
- کارآمد ساختن تعامل همراه با پیشرفت سطوح مهارتی و امکان سفارشی کردن
- مخفی کردن موارد فنی داخلی از دید کاربران عادی
- طراحی تعامل مستقیم با اشیایی که روی صفحه نمایش ظاهر می‌شوند.

۲-۱-۸ کاستن از بار حافظه کاربر

در این قانون نیز رعایت نکات زیر الزامی است:

- کاهش بار در حافظه کوتاه مدت
- ایجاد پیش‌گزیده‌های معنی‌دار
- تعیین میان‌برهای شهودی
- آشکارسازی اطلاعات به شیوه‌ای تدریجی
- طرح بصری واسط باید بر اساس استعاره جهان واقعی استوار باشد.

۳-۱-۸ سازگارسازی واسط

در این قانون نیز رعایت نکات زیر الزامی است:

- قرار دادن عمل فعلی در یک بافت معنی‌دار توسط کاربر.
- حفظ ثبات در خانواده برنامه‌های کاربردی.
- اگر مدل‌های تعاملی پیشین انتظاراتی را در کاربر به وجود آورده‌اند، تا زمانی که دلیل قانع‌کننده‌ای ندارید از انجام تغییرات خودداری کنید.

۲-۸ طراحی واسط کاربر

طراحی واسط کاربر، با ایجاد مدل‌های مختلف کارکرد سیستم (آن طور که از بیرون مشاهده می‌شود) آغاز می‌شود. سپس وظایف انسانی و رایانه‌ای لازم برای تحقق کارکرد سیستم، توصیف می‌شوند. موضوعات طراحی که در تمام طراحی‌های واسط کاربرد دارند مد نظر قرار می‌گیرند. برای الگوسازی و پیاده‌سازی نهایی مدل طراحی واسط‌ها، ابزارهایی به کار می‌روند و نتیجه از لحاظ کیفی ارزیابی می‌شود.

۱-۲-۸ مدل‌های طراحی واسط

هنگام طراحی یک واسط کاربر، چهار مدل مختلف مطرح هستند:

- مهندس نرم‌افزار مدل طراحی را ایجاد می‌کند.
- مهندس فاکتورهای انسانی (یا مهندس نرم‌افزار) مدل کاربر را تعیین می‌کند.
- کاربر نهایی یک تصویر ذهنی می‌سازد که غالباً مدل ذهنی کاربر را به وجود می‌آورد.
- پیاده‌کنندگان سیستم نیز یک تصویر سیستم ایجاد می‌کنند.

متأسفانه ممکن است هر یک از این مدل‌ها، تفاوت‌های قابل ملاحظه‌ای با یکدیگر داشته باشند. نقش طراح واسط، رفع اختلافات و به دست آوردن یک نمایش منسجم و سازگار از واسط‌هاست.

۳-۸ فرایند طراحی واسط کاربر

فرایند طراحی واسط‌های کاربر، یک فرایند تکراری است و با استفاده از مدل حلزونی، قابل ارائه است. با مراجعه به شکل ۱، روند طراحی واسط کاربر شامل چهار فعالیت مجزای زیر است.

۱-۳-۸ تحلیل نیازهای کاربر، وظایف، محیط و مدل‌سازی

تحلیل اولیه بر سابقه کاربرانی تکیه دارد که با سیستم تعامل می‌کنند. سطح مهارت افراد، شناخت حرفه و سازمان و سطح آمادگی آنان شناسایی و تحلیل می‌گردد و گروه‌های کاربر متفاوت تعریف می‌شود. در واقع مهندس سیستم سعی می‌کند تا برداشت سیستم را برای هر طبقه از کاربران شناسایی کند. تحلیل محیط کاربر بر محیط کار فیزیکی تکیه دارد. اطلاعات حاصل از این فعالیت به منظور ایجاد یک مدل تحلیل برای واسط کاربر استفاده می‌شود.

۲-۳-۸ طراحی واسط

هدف طراحی واسط، تعریف یک مجموعه از اشیا و عملیات واسط و نمایش آن‌ها در صفحه نمایش است که کاربر را قادر به انجام تمام وظایف تعریف شده می‌سازد، به طوری که همه اهداف قابلیت استفاده را برای یک سیستم برآورده سازد.

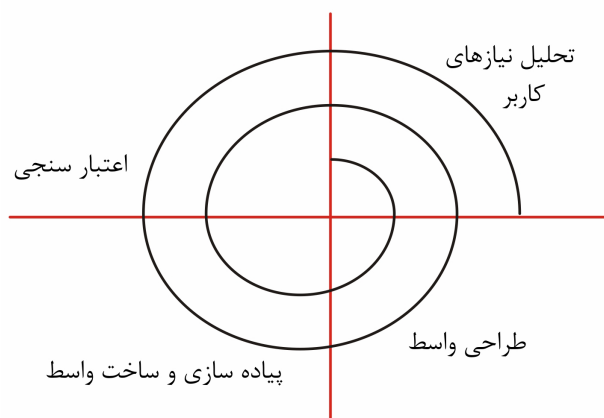
۳-۳-۸ پیاده‌سازی و ساخت واسط

این فعالیت با ایجاد یک نمونه اولیه آغاز می‌شود که ارزیابی سناریوهای کاربر را امکان‌پذیر می‌سازد.

۴-۳-۸ اعتبارسنجی واسط

اعتبارسنجی بر موارد زیر تکیه دارد:

- توانایی واسط در پیاده‌سازی تمامی وظایف، انجام تمام وظایف و دستیابی به کلیه خواسته‌های عمومی کاربر،
- میزان سهولت استفاده و فراگیری واسط، و
- برداشت کاربران از واسط به عنوان یک ابزار مفید در کارهای آنان.



شکل ۱-۸ فرایند طراحی واسط کاربری

۴-۸ مسایل طراحی واسط‌ها

در حین تکمیل طراحی واسط کاربر، چهار مسئله معمول طراحی تقریباً همیشه سطحی تلقی می‌شوند که باید به آن‌ها توجه کرد:

- زمان پاسخگویی سیستم
- تسهیلات کمکی کاربر
- خطاگردانی اطلاعات
- برچسب‌گذاری فرمان

زمان پاسخگویی سیستم: از زمانی که کاربر یک عمل کنترلی را انجام می‌دهد (مثلاً کلید بازگشت را زده یا بر ماوس کلیک می‌کند تا زمان پاسخگویی نرم‌افزار با یک اقدام یا خروجی مطلوب، اندازه‌گیری می‌شود.

زمان پاسخگویی سیستم دو ویژگی مهم دارد: **طول و تغییرپذیری**.

تسهیلات کمکی یکپارچه از آغاز در داخل نرم‌افزار طراحی می‌شود. این نوع تسهیلات کمکی، غالباً به متن حساس هستند و کاربر را قادر کنند تا از میان موضوعات مرتبط با اعمال در حال اجرا، اقدام به انتخاب کند. واضح است که این امر، زمان لازم برای دریافت کمک توسط کاربر را کاهش داده و کاربرپسندی واسط را افزایش می‌دهد.

تسهیلات کمکی افزودنی. پس از ساخت سیستم، این‌گونه تسهیلات کمکی به نرم‌افزار افزوده می‌شود. ممکن است کاربر مجبور شود برای یافتن راهنمایی صحیح و مناسب، فهرستی با صدها موضوع را جستجو کند و اغلب با شروع نادرست، اطلاعات غیر مرتبط را دریافت کند.

شکی نیست که امکانات کمکی یکپارچه بر نوع افزودنی آن برتری دارد. خطاگردانی اطلاعات، هنگام بروز خطا، پیغام‌های خطا و هشدار، "اخبار بدی" است که به کاربران سیستم‌های تعاملی ارائه می‌شود. در بدترین حالت، پیغام‌های خطا و اخطارها، اطلاعات بی‌فایده یا گمراه‌کننده را منتقل کرده و تنها باعث تشدید ناکامی کاربر می‌شوند. تعداد کاربران رایانه که با خطایی از نوع زیر مواجه نشده باشند، بسیار اندک است.

اطلاعات داده‌شده غلط می‌باشد!

به طور کلی هر پیام خطا یا هشدار تولیدشده توسط یک سیستم محاوره‌ای باید دارای ویژگی‌های زیر باشد:

- پیام باید مشکل را به زبانی شرح دهد که کاربر قادر به درک آن باشد.
- پیام باید حاوی یک توصیه سازنده برای رهایی از وضعیت خطا باشد.
- پیام باید هرگونه تبعات منفی خطا (مثلاً فایل‌های داده‌ای مخدوش‌شده) را خاطر نشان کند تا کاربر بتواند آن‌ها را کنترل کند.
- پیام باید با یک نشانه سمعی یا بصری همراه باشد.
- پیام باید قضاوت‌گونه باشد.

برچسب‌گذاری فرمان‌ها، فرمان تایپ‌شده زمانی رایج‌ترین شیوه محاوره بین کاربر و نرم‌افزار بود و در هر نوع برنامه کاربردی به طور معمول به کار می‌رفت. امروزه، استفاده از واسط‌های پنجره‌ای، اشاره‌ای و انتخاب، کاربرد فرامین تایپ‌شده را کاهش داده است، اما بسیاری از کاربران ماهر همچنان شیوه ارتباطی مبتنی بر فرمان را ترجیح می‌دهند. هنگام انتخاب فرامین تایپ‌شده به عنوان نوعی شیوه محاوره، برخی مسایل طراحی باید مورد توجه قرار گیرند:

- آیا هر یک از گزینه‌های موجود در پنجره، یک فرمان متناظر دارند؟
- فرامین چه شکلی خواهند داشت؟ امکانات موجود عبارت‌اند از: توالی کنترل (مثل Alt + P)؛ کلیدهای تابعی، واژه تایپ‌شده.
- یادگیری و به خاطر سپردن فرامین تا چه حد دشوار است؟ در صورت فراموشی یک فرمان، چه می‌توان کرد؟
- آیا کاربر می‌تواند فرمان‌ها را به سلیقه خویش مختصر، کوتاه و به دلخواه خود تنظیم کند؟

۵-۸ ابزارهای پیاده‌سازی واسط‌ها

برای انجام روش تکراری طراحی واسط‌های نرم‌افزاری، انواع ابزارهای طراحی واسط و ساخت نمونه اولیه وجود دارند. این ابزارها با عنوان سیستم توسعه واسط کاربر (UIDS)^۱ نامیده می‌شوند. این سیستم با استفاده از مؤلفه‌های نرم‌افزاری، راهکاری برای موارد زیر فراهم می‌آورد:

- مدیریت دستگاه‌های ورودی (مثل ماوس یا صفحه کلید)
- اعتبارسنجی ورودی کاربر
- کنترل خطا و نمایش پیغام‌های خطا
- فراهم آوردن بازخورد (مثل بازتاب ورودی خودکار)
- فراهم آوردن راهنما و پیغام
- کنترل پنجره‌ها و فیلدها، حرکت در داخل پنجره‌ها
- برقراری ارتباط میان نرم‌افزار کاربردی و واسط
- جدا کردن برنامه کاربردی از عملکردهای مدیریت واسط
- مجاز کردن کاربر به سفارشی کردن واسط

۶-۸ ارزیابی طراحی واسط

پس از ایجاد الگوی عملی واسط کاربر، این مدل باید مورد ارزیابی قرار گیرد تا معلوم شود آیا نیازهای کاربر را برطرف می‌کند یا خیر؟ ارزیابی می‌تواند در یک طیف رسمی صورت گیرد که گستره آن با انجام آزمونی غیر رسمی که در ضمن آن کاربر، بازتابی بدون فکر قبلی دارد شروع شود و به مطالعه رسمی ختم می‌گردد که از روش‌های آماری برای ارزیابی پرسشنامه‌های تکمیل‌شده توسط کاربران نهایی، استفاده می‌کند. برخی از معیارهای ارزیابی را هنگام بررسی‌های اولیه طراحی، می‌توان اعمال کرد:

- طول و پیچیدگی مشخصات مکتوب سیستم و واسط آن، بیانگر میزان یادگیری لازم توسط کاربران سیستم است.
 - تعداد وظایف تعیین‌شده کاربر و میانگین اعمال در هر کار، نشان‌دهنده زمان محاوره و کارایی کلی سیستم است.
 - تعداد اعمال، وظایف و وضعیت‌های سیستم که در مدل طراحی تعیین‌شده، به بار حافظه کاربران سیستم دلالت دارد.
 - روش ارتباط واسط، امکانات کمکی و خطاگردانی، در کل بیانگر پیچیدگی واسط و میزان پذیرش از سوی کاربر است.
- پس از ساخته شدن اولین مدل، طراح می‌تواند مجموعه‌ای از داده‌های کیفی و کمی را که به ارزیابی واسط کمک می‌کنند جمع‌آوری کند.

فصل نهم

طراحی در سطح مؤلفه

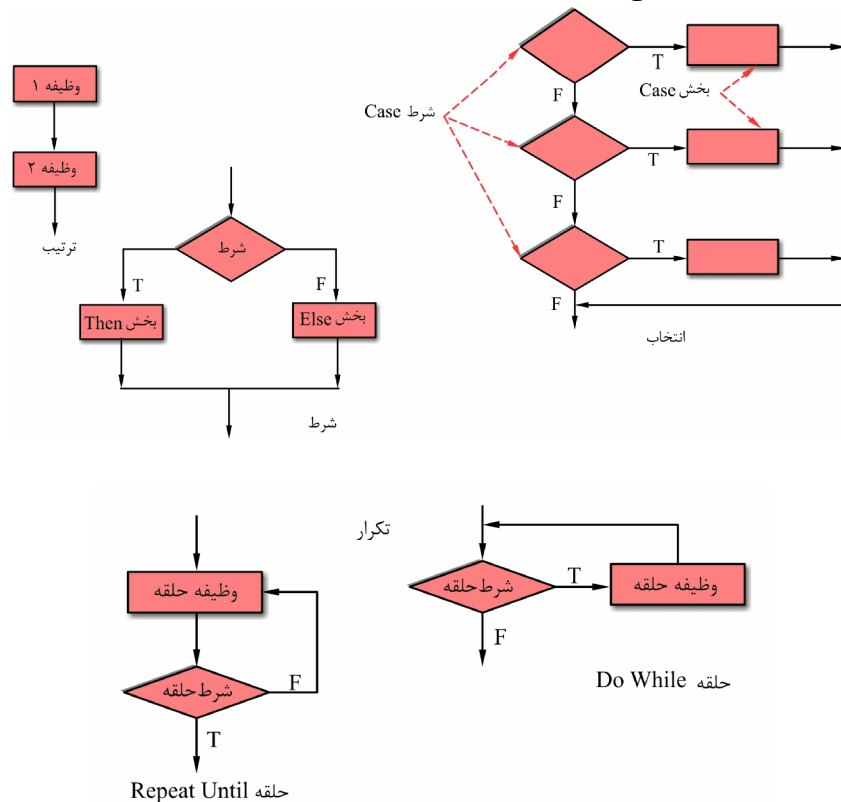
طراحی در سطح مؤلفه که "طراحی رویه‌ای" نیز نام دارد، پس از طراحی داده، معماری و واسط انجام می‌گیرد و هدف آن تبدیل مدل طراحی به نرم‌افزار عملیاتی است. نمایش طراحی در سطح مؤلفه با استفاده از یک زبان برنامه‌سازی امکان‌پذیر است. روش دیگر نشانه‌گذاری و نمایش طراحی رویه‌ای با استفاده از نمایش واسطه (مثلاً گرافیکی، جدولی یا متنی) است که به راحتی قابل تبدیل به برنامه‌مبدأ^۱ است.

۱-۹ برنامه‌سازی ساخت‌یافته

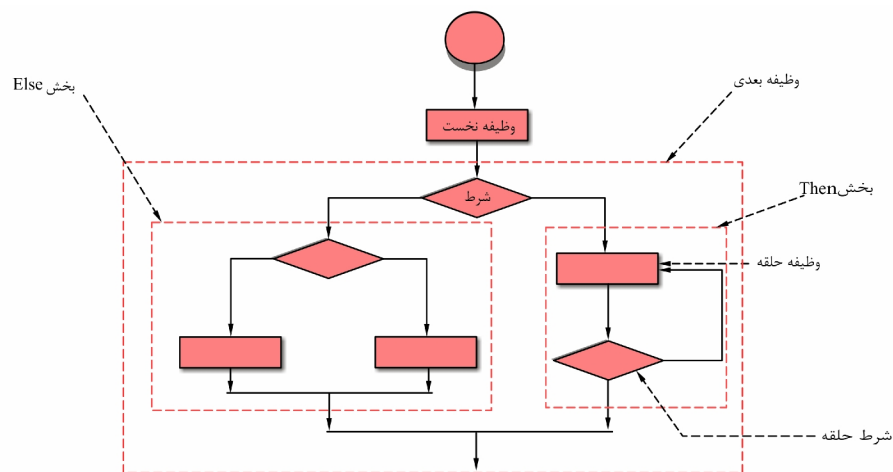
استفاده از ساختارهای منطقی در دهه ۱۹۶۰ توسط محققین بسیاری از جمله Dijkstra پیشنهاد شده است که هر برنامه‌ای را با آن‌ها می‌توان نوشت. این ساختارها عبارت‌اند از توالی، شرط و تکرار.

۲-۹ نشانه گذاری طراحی گرافیکی

هیچ تردیدی نیست که ابزارهای گرافیکی مانند روندنما^۱ یا نمودارهای مستطیلی، الگوهای تصویری مفیدی هستند که به سادگی جزئیات رویه‌ای را مرسوم می‌سازند. در اشکال زیر نمونه‌هایی از ساختارهای مختلف نشان داده شده است.



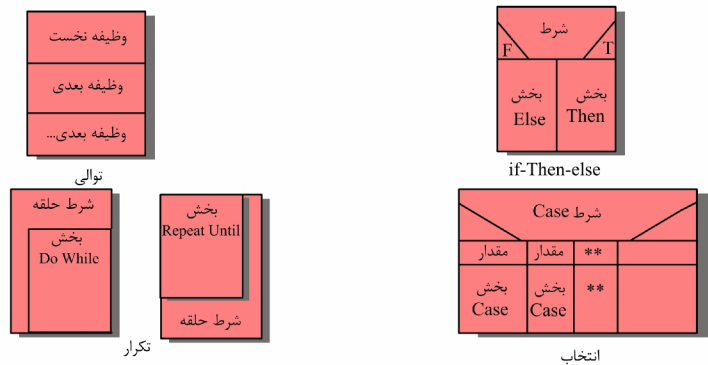
شکل ۹-۱ ساختمان‌های توالی، شرط و تکرار در روندنما



شکل ۹-۲ ساختمان‌های تو در تو در روندنما

۳-۹ نمودار مستطیلی

نمایش گرافیکی سازه‌های ساخت‌یافته با استفاده از **نمودار مستطیلی** (یا جعبه‌ای یا نمودارهای ناسی - اشنايدرمن) در اشکال زیر نشان داده شده است. عنصر اصلی نمودار یک کادر یا جعبه است. برای نمایش توالی، مستطیل‌ها از بالا به پایین به یکدیگر متصل می‌شوند. برای نمایش if-Then-else، مستطیل شامل دو بخش Then- Part و else- Part به دنبال شرط if قرار می‌گیرد و الی آخر.



شکل ۳-۹ ساختمان‌های نمودار مستطیلی

ویژگی‌های نمودارهای مستطیلی عبارت‌اند از:

- دامنه عملیاتی به خوبی تعیین می‌شود و به صورت یک نمایش تصویری قابل مشاهده است.
- انتقال اختیاری کنترل غیر ممکن است.
- دامنه داده‌های محلی و سراسری را به سهولت می‌توان تعیین کرد.
- نمایش دادن روابط برگشتی آسان است.

۴-۹ علایم طراحی جدولی

در بسیاری از کاربردهای نرم‌افزاری، برای ارزیابی ترکیب پیچیده شرایط و انتخاب اعمال مناسب مبتنی بر این حالت‌ها، ممکن است به پیمان‌های نیاز باشد. جداول تصمیم‌گیری، علایمی را به کار می‌برند که اعمال و شرایط (توصیف‌شده در شرح پردازش) را به شکل جدولی تبدیل می‌کند. تعبیر غلط جدول، دشوار است و حتی ممکن است به عنوان ورودی برای یک الگوریتم قابل خواندن ولی برای یک دستگاه قابل خواندن نباشد.

ساختار جدول تصمیم‌گیری در شکل ۴-۹ نمایش داده شده است.

شرایط	۱	۲	۳	۴	...	n
شرط شماره ۱						
شرط شماره ۲						
شرط ...						
اقدام شماره ۱						
اقدام شماره ۲						
اقدام شماره ۳						
اقدام ...						

شکل ۴-۹ ساختار جدول تصمیم‌گیری

در ایجاد جدول تصمیم‌گیری، مراحل زیر طی می‌شود:

- تمامی اعمال مرتبط با یک رویه (یا پیمان‌ه) خاص را فهرست کنید.
- تمامی شرایط (یا تصمیمات اتخاذشده) طی اجرای رویه را لیست کنید.
- مجموعه خاص شرایط را با اعمال و اقدامات به‌خصوصی که ترکیبات غیر ممکن شرایط را برطرف می‌سازند، مرتبط کنید یا اینکه هرگونه جابه‌جایی ممکن را در شرایط ایجاد کنید.

- با بیان اعمال انجام شده در ارتباط با مجموعه شرایط، قوانین را تعریف کنید.

۵-۹ زبان توصیف برنامه (PDL)^۱

زبان توصیف برنامه (PDL) که زبان ساخت یافته یا شبه کد نیز نام دارد، یک زبان آمیخته است طوری که واژگان یک زبان (یعنی انگلیسی) و نحو کلی زبانی دیگر (یعنی زبان برنامه سازی ساخت یافته) را به کار می برد.

یک زبان توصیف باید ویژگی های زیر را داشته باشد:

- یک نحو ثابت از کلمات کلیدی که تمام سازه های ساخت یافته، تعاریف داده ها و ویژگی های پیمانته ای شدن را تهیه کند.
- یک نحو آزاد از زبان طبیعی که ویژگی های پردازشی را تشریح کند.
- تسهیلات تعریف داده ها که باید مشتمل بر ساختارهای داده ای ساده (اسکالر، آرایه) و پیچیده (لیست پیوندی یا درخت) باشد.
- تعریف زیر برنامه و فنون فراخوانی که حالات مختلف توصیف واسط را پشتیبانی کند.
- نمونه ای از کاربرد PDL برای سیستم خانه امن به صورت زیر خواهد بود:

```
PROCEDURE security.monitor;
  INTERFACE RETURNS system.status;
  TYPE signal IS STRUCTURE DEFINED
    Name IS STRING LENGTH VAR;
    Address IS HEX device location;
    bound. Value IS upper bound SCALAR;
    message IS STRING LENGTH VAR;
  END signal TYPE;
  TYPE system.status IS BIT (4);
  TYPE alarm. Type DEFINED
    Smoke. Alarm IS INSTANCE OF signal;
    Fire.alarm IS INSTANCE OF signal;
    Water.alarm IS INSTANCE OF signal;
    temp.alarm IS INSTANCE OF signal;
    burglar.alarm IS INSTANCE OF signal;
  TYPE phone. Number IS area code +7-digit number;
  ...
Case of Contro.Pane.Switches (ops):
  When ops = "test" select
    Call alarm Procedure with "on" for test.time in seconds;
  When ops = "alarm-off" select
  ....
ENDCASE
```

...
END security.monitor

نشان گذاری طراحی باید به نمایش رویه ای منجر شود که درک و بررسی آن آسان باشد. به علاوه، نشان گذاری باید توانایی تبدیل به برنامه را تقویت کند طوری که برنامه یا کد به محصول جانبی و طبیعی از طراحی تبدیل شود. و نهایتاً اینکه نمایش طراحی باید به راحتی قابل نگهداری باشد به شکلی که طراحی همواره به طور صحیح نشان دهنده برنامه باشد.

۶-۹ مقایسه نشانه گذاری های طراحی

خصوصیات زیر در مورد نشانه گذاری طراحی تعیین شده اند:

- قابلیت پیمانته ای
- سادگی همه جانبه
- سهولت ویرایش
- قابلیت خواندن سیستم
- قابلیت نگهداری
- تقویت ساختار
- پردازش خودکار
- بازنمایی داده ها
- تأیید منطق
- توانایی تبدیل به برنامه

فصل دهم

اصول و قواعد تحلیل شیء گرا

ما در جهانی از اشیا زندگی می‌کنیم. این اشیا در طبیعت، در نهادهای ساخت دست بشر، در تجارت و در محصولاتی که استفاده می‌کنیم، وجود دارند. آن‌ها را می‌توان توصیف، بسته‌بندی، سازماندهی، ترکیب، دستکاری و ایجاد کرد؛ بنابراین، تعجیبی ندارد که برای توسعه نرم‌افزارهای رایانه‌ای نیز دیدگاهی شیء‌گرا پیشنهاد شود.

۱-۱۰ تاریخچه

روش شیء‌گرا در توسعه نرم‌افزار اولین بار در اواخر دهه ۱۹۶۰ به کار گرفته شد ولی ۲۰ سال طول کشید تا فن‌آوری شیء‌گرا به طور گسترده مورد استفاده قرار گیرد.

فن‌آوری‌های شیء‌گرا منجر به استفاده مجدد می‌شود و استفاده مجدد (از مؤلفه‌های برنامه) منجر به توسعه سریع‌تر نرم‌افزارها و برنامه‌هایی با کیفیت بالاتر می‌شود. نگهداری نرم‌افزارهای شیء‌گرا آسان‌تر است زیرا ساختار آن ذاتاً فاقد پیوستگی و دارای انسجام مناسب است. این موضوع، هنگام اعمال تغییرات، اثرات جانبی کمتری به وجود می‌آورد و برای مهندس نرم‌افزار و مشتری دردسر کمتری ایجاد می‌کند. به علاوه، تطبیق دادن و تغییر دادن اندازه سیستم‌های شیء‌گرا آسان‌تر است (یعنی سیستم‌های بزرگ را می‌توان با مونتاژ کردن زیرمؤلفه‌های قابل استفاده مجدد ایجاد کرد (مقیاس پذیری)).

۲-۱۰ مفاهیم کلی شیء‌گرایی

- برای درک موضوع از دیدگاه شیء‌گرایی، مثالی از اشیای دنیای واقعی مثلاً یک "صندلی" را در نظر بگیرید.
- صندلی یک عضو (Member) یا یک نمونه (Instance) از یک کلاس بزرگ‌تر از اشیای اثاثیه منزل است.
 - در کلاس اثاثیه مجموعه‌ای از صفات (Attributes) کلی و مشترک به تمام اشیای این کلاس نسبت داده می‌شود. به عنوان مثال تمام اثاثیه دارای قیمت، ابعاد، وزن، محل، رنگ و خیلی صفات احتمالی دیگر هستند.
 - چون صندلی عضوی از اثاثیه خانه است، تمام صفاتی را که برای کلاس اثاثیه تعریف شده است به ارث می‌برد (Inherits).
- در این مثال کوشش کرده‌ایم تا تعریفی حکایت‌وار از کلاس را با توصیف صفات آن ارائه کنیم ولی چیزی کم است. هر کدام از اعضای کلاس اثاثیه را می‌توان به چندین شیوه دستکاری کرد. می‌توان آن را خرید، فروخت، تغییر فیزیکی در آن ایجاد کرد (مثلاً پایه‌ها را اره کرد یا آن را

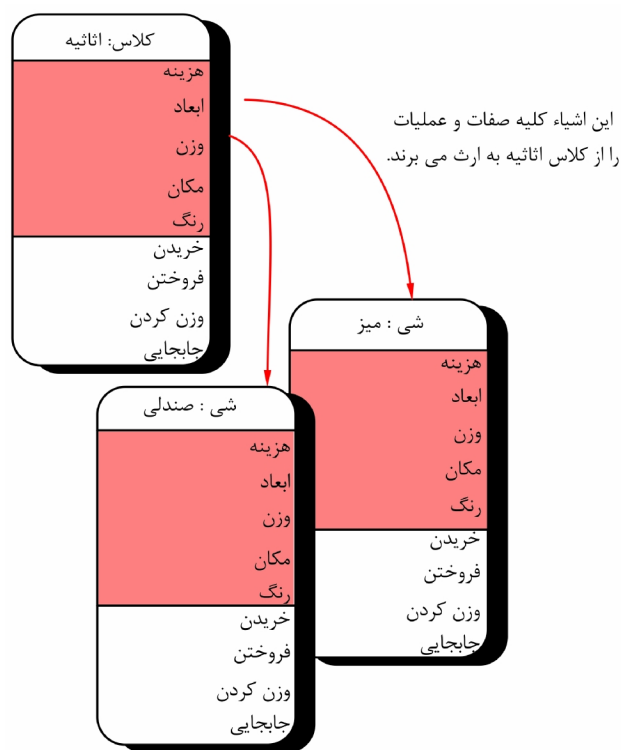
ارغوانی رنگ کرد) یا از مکانی به مکان دیگر منتقل کرد. هر یک از عملیات (یا متدها یا سرویسها) یک یا چند صفت شیء را تغییر می دهند. برای مثال، اگر صفت مکان یک عنصر داده‌ای مرکب به صورت زیر باشد:

اتاق + طبقه + ساختمان = مکان

در این صورت، عملی که جابه‌جایی نام دارد، یک یا چند مورد از این عناصر داده‌ای (ساختمان، طبقه، اتاق) که مکان شیء را تشکیل می دهند، تغییر می دهد. برای انجام این کار، عمل جابه‌جایی باید از این عناصر داده‌ای آگاه باشد. مادامی که صندلی و میز هر دو نمونه‌های کلاس اثاثیه باشند، از عمل جابه‌جایی می توان برای آن‌ها نیز استفاده کرد. همه عملیات معتبر (مثل خریدن، فروختن، توزیع کردن) برای کلاس اثاثیه به تعریف شیء متصل هستند و برای تمام نمونه‌های این کلاس به ارث گذاشته می شوند.

شیء صندلی (و کلاً همه اشیا)، داده‌ها (مقادیر صفاتی که صندلی را تعریف می کنند)، عملیات (کاری که برای تغییر دادن صفات صندلی به کار می روند)، اشیای دیگر (اشیای مرکبی که قابل تعریف هستند)، ثابت‌ها (مقادیر ثابت) و اطلاعات مربوطه دیگر را بسته‌بندی می کنند. بسته‌بندی بدان معناست که تمامی این اطلاعات تحت یک نام بسته‌بندی شوند و به عنوان یک مشخصه یا قطعه برنامه به کار برده شوند. Coad و Yourdon تعریف رسمی از شیء‌گرایی را به صورت زیر ارائه داده‌اند:

ارتباطات + وراثت + طبقه‌بندی + اشیا = شیء‌گرایی



شکل ۱۰-۱ وراثت بین کلاسها

۱۰-۲-۱ کلاسها و اشیا

کلاس یک مفهوم شیء‌گرایی است که داده‌ها و رویه‌هایی را که برای توصیف محتوا^۱ و رفتار^۲ یک موجودیت دنیای واقعی لازم است، بسته‌بندی می کند.

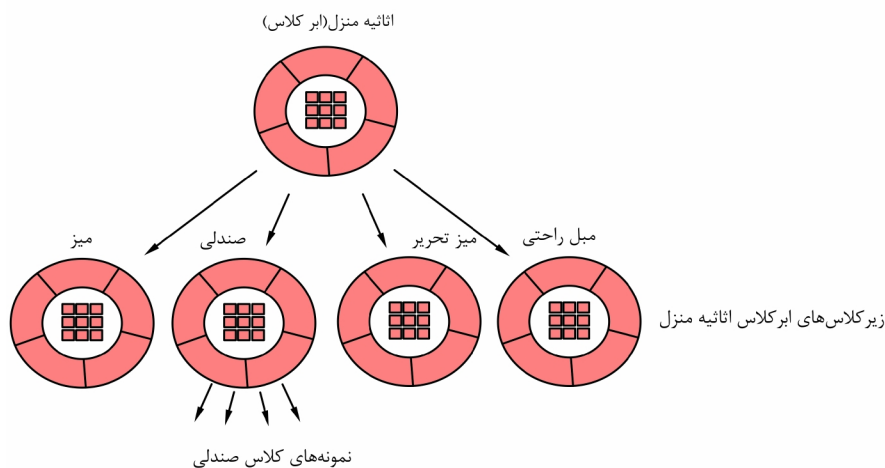
انتزاع داده‌ای (صفات) که کلاس را توصیف می کند در میان یک "دیوار" به نام انتزاع رویه‌ای (که عملیات، خدمات یا متدها نامیده می شود) قرار گرفته‌اند که قادر به تغییر داده‌ها در بعضی موارد است.

تنها راه برای دسترسی به صفات از طریق یکی از متدهاست (کلاس، داده‌ها و متدها را بسته‌بندی می‌کند). این کار باعث پنهان‌سازی اطلاعات^۱ می‌شود و خود منجر به کمتر شدن تأثیر اثرات جانبی مرتبط با تغییرات می‌شود.

از آنجاکه متدها تعداد محدودی از صفات را تغییر می‌دهند، آن‌ها منسجم^۲ هستند و به خاطر اینکه ارتباطات بین کلاس‌ها فقط از طریق متدها رخ می‌دهد، هر کلاس از سایر اجزای سیستم جدا^۳ می‌شود. این خصوصیات منجر به شکل‌گیری یک نرم‌افزار با کیفیت بالا^۴ می‌شود.

• سلسله‌مراتبی کلاس‌ها

ابرجلاس^۵ مجموعه‌ای از کلاس‌هاست و زیرکلاس^۶ یک نمونه خاص از یک کلاس است. این تعاریف دلالت بر وجود سلسله‌مراتب کلاس^۷ می‌کند که در آن صفات و عملیات ابرکلاس به وسیله زیرکلاس‌ها به ارث برده می‌شوند. البته هر کدام از این زیرکلاس‌ها ممکن است داده‌ها و متدهای "خصوصی" نیز داشته باشند.



شکل ۲-۱۰ نمایشی از سلسله‌مراتب کلاس‌ها

۲-۲-۱۰ پیغام‌ها^۸

پیغام، ابزاری برای تعامل اشیا با یکدیگر است. پیغام باعث برانگیختن رفتاری خاص در شیء گیرنده می‌شود. این رفتار زمانی دیده می‌شود که عملی اجرا شود؛ بنابراین:

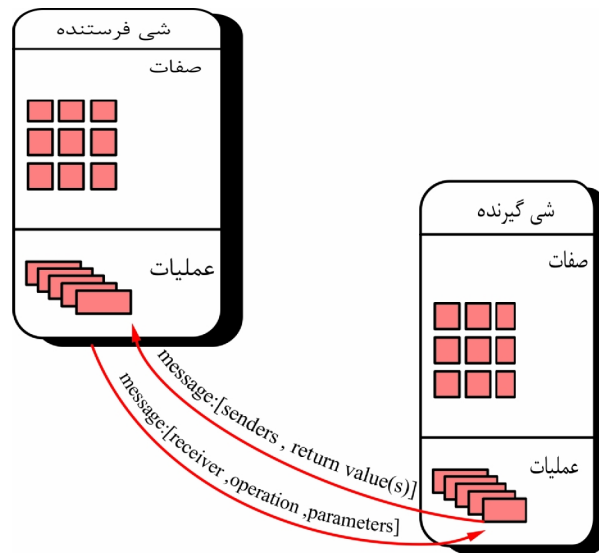
- پیغام‌ها ابزاری هستند که اشیا از طریق آن‌ها با یکدیگر ارتباط برقرار می‌کنند.
- متد موجود در شیء فرستنده پیغامی به صورت زیر تولید می‌کند:

پیغام : [مقصد، متد، پارامترها]

مقصد، شیء گیرنده را - که با دریافت پیغام فعال می‌شود - تعریف می‌کند، متد به تابعی اشاره می‌کند که باید اجرا شود و پارامترها اطلاعاتی را ارائه می‌کنند که برای درست انجام شدن عملیات لازم‌اند.

شیء گیرنده با انتخاب عملیاتی که نام پیغام را پیاده‌سازی می‌کند، اجرای آن و بازگرداندن کنترل به فراخواننده به پیغام پاسخ می‌دهد.

1 - Information Hiding
 2 - Cohesive
 3 - Decoupled
 4 - High-Quality Software
 5 - Superclass
 6 - Subclass
 7 - Class Hierarchy
 8 - Messages



شکل ۳-۱۰ مبادله پیام‌ها بین دو شیء

۳-۱۰ ویژگی‌های سیستم‌های شیء‌گرا^۱

گرچه ساختار و اصطلاحات گفته‌شده، سیستم‌های OO را از هم‌تاهای سنتی موجود متفاوت می‌سازد، اما سه ویژگی سیستم‌های شیء‌گرا آن‌ها را نسبت به شیوه‌های سنتی منحصر به فرد می‌سازد.

۱-۳-۱۰ بسته‌بندی^۱

چنانکه پیش از این نیز گفتیم، کلاس و اشیایی که در آن ایجاد می‌شوند، داده‌ها و عملیاتی را که بر روی آن‌ها عمل می‌کنند، یک جا بسته‌بندی می‌کنند. مزایای این کار عبارت‌اند از:

- جزئیات داخلی پیاده‌سازی داده‌ها و رویه‌ها از دنیای خارج پنهان هستند (پنهان‌سازی اطلاعات). این کار باعث می‌شود که گسترش اثرات جانبی به هنگام تغییرات کم شود.
- ساختارهای داده و عملیاتی که آن‌ها را تغییر می‌دهند در یک موجودیت تکی به نام کلاس با یکدیگر ادغام می‌شوند. این کار نیز باعث می‌شود که استفاده مجدد از مؤلفه^۲ راحت‌تر شود.
- واسط‌های بین اشیای بسته‌بندی شده ساده‌تر می‌شوند. شیء فرستنده یک پیغام، لازم نیست که به جزئیات داخلی ساختارهای داده توجه کند. این کار باعث می‌شود که اتصال سیستم کاهش یابد.

۲-۳-۱۰ وراثت^۳

وراثت یکی از مهم‌ترین تفاوت‌های بین سیستم‌های سنتی و سیستم‌های شیء‌گراست. یک زیرکلاس Y تمام صفات و عملیات را از ابرکلاس خود X به ارث می‌برد. این بدین معنی است که تمام ساختارهای داده و الگوریتم‌هایی که در ابتدا برای X طراحی و پیاده‌سازی شده‌اند، اکنون برای Y نیز موجودند.

هر تغییر در داده‌ها یا عملیات موجود در ابرکلاس سریعاً به وسیله تمام زیرکلاس‌هایی که از آن ارث‌بری دارند، به ارث برده می‌شود؛ بنابراین سلسله‌مراتب کلاس تبدیل به مکانیزمی شده است که به وسیله آن تغییرات (در سطوح بالا) سریعاً می‌تواند در سیستم پخش شود.

1 - Encapsulation
2 - Component Resuability
3 - Inheritance

۳-۳-۱۰ چندریختی^۱

چندریختی باعث می شود که چندین عملیات بتوانند از یک نام استفاده کنند. این کار باعث می شود که تعداد خطوط برنامه برای پیاده سازی کاهش یافته و اعمال تغییرات تسهیل شود.

۴-۱۰ شناسایی عناصر یک مدل شیء گرا

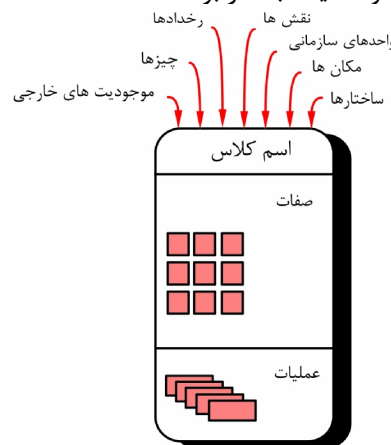
کلاس ها، اشیا، صفات، عملیات و پیغام ها عناصر مدل تحلیل شیء گرا هستند. حال ببینیم چگونه این عناصر را باید شناسایی کرد.

۱-۴-۱۰ شناسایی کلاس ها و اشیا

شناسایی اشیا را با بررسی بیان مسئله یا (با استفاده از اصطلاحی که در فصل ۴ معرفی شد) با اجرای تجزیه گرامری روی توصیف پردازش های سیستمی که قرار است ساخته شود، آغاز می کنیم. برای تعیین اشیا، زیر اسم ها یا عبارات اسمی خط کشیده، آن ها را در جدولی لیست می کنیم. به اسم های مترادف نیز باید توجه داشت. اشیا می توانند به یکی از طرق زیر خود را نشان دهند:

- موجودیت های خارجی (مثلاً سیستم های دیگر، دستگاه ها، افراد) که اطلاعات مورد استفاده سیستم رایانه ای را تولید یا مصرف می کنند.
- چیزهایی (مانند گزارش ها، صفحات نمایش، نامه ها، علائم الکترونیکی) که بخشی از دامنه اطلاعاتی مسئله به شمار می روند.
- رخدادها یا وقایعی (مثلاً کامل شدن حرکات روبات) که در حیطه عملکرد سیستم رخ می دهند.
- نقش هایی (مثل مدیر، مهندس و فروشنده) باشند که افراد در حال تعامل با سیستم بر عهده دارند.
- واحدهای سازمانی (مثل بخش، گروه و تیم) که با یک کاربرد خاص سر و کار دارند.
- مکان هایی (مثل سالن تولید) که حیطه مسئله و وظیفه کلی سیستم را مشخص می کنند.
- ساختارهایی (مثل حسگرها، وسایل نقلیه یا رایانه ای) که کلاسی از اشیا یا کلاس های مرتبطی از اشیا را تعریف می کنند.

توجه به این نکته نیز مهم است که اشیا چه چیزهایی نیستند. به طور کلی یک شیء هیچ گاه نباید یک "نام روبه ای دستوری" داشته باشد. تجزیه گرامری را می توان برای تفکیک اشیا (اسامی) و عملیات به کار برد.



شکل ۴-۱۰ چگونگی شناسایی اشیا

Coad و Yourdon شش خصوصیت گزینشی ارائه داده اند که تحلیل گر باید در مورد هر شیء بالقوه ای که در مدل تحلیل به کار می برد در نظر بگیرد:

- اطلاعات ذخیره شده: شیء بالقوه فقط در صورتی هنگام تحلیل مفید خواهد بود که اطلاعات آن برای عملکرد سیستم لازم باشد.
- خدمات مورد نیاز: شیء بالقوه باید مجموعه ای از عملیات قابل شناسایی داشته باشد که می توانند صفات آن را به طریقی تغییر دهند.
- صفات چندگانه: یک شیء با یک صفت ممکن است در طول طراحی مفید باشد ولی ممکن است هنگام تحلیل به عنوان صفتی از شیء دیگر منظور شود.

- **صفات مشترک:** مجموعه‌ای از صفات را می‌توان برای یک شیء بالقوه تعریف کرد که این صفات در تمام نمونه‌های شیء به کار می‌روند.
- **عملیات مشترک:** مجموعه‌ای از عملیات را می‌توان برای یک شیء بالقوه تعریف کرد که این عملیات در تمام نمونه‌های شیء به کار می‌روند.
- **نیازهای ضروری:** موجودیت‌های خارجی که در فضای مسئله ظاهر می‌شوند و اطلاعات ضروری برای کارکرد هر راهکار را تولید یا مصرف می‌کنند که در مدل نیازها باید به عنوان شیء تعریف شوند.

۲-۴-۱۰ مشخص کردن صفات

- صفات، اشیا را طوری تعریف و توصیف می‌کنند که برای ورود به مدل تحلیل انتخاب شده‌اند.
- برای توسعه یک مجموعه بامعنی از صفات، تحلیل‌گر می‌تواند شرح پردازش مسئله را مطالعه کند و چیزهایی را انتخاب کند که به طور منطقی به شیء تعلق دارند.
- به‌علاوه باید به این سؤال پاسخ داد: "چه عناصر داده‌ای (مرکب و یا ساده) این شیء را در حیطه مسئله به طور کامل تعریف می‌کنند؟"

۳-۴-۱۰ تعریف عملیات

- عملیات، رفتار یک شیء را تعریف کرده و صفات آن را به طریقی تغییر می‌دهند. به طور مشخص، یک عمل مقدار یک یا چند صفت موجود در شیء را تغییر می‌دهد.
- عملیاتی که با داده‌ها به طریقی کار می‌کنند (مانند اضافه کردن، حذف کردن، قالب‌بندی دوباره و گزینش).
 - عملیاتی که محاسبه‌ای را انجام می‌دهند.
 - عملیاتی که برای رخ دادن یک رویداد کنترلی بر شیء نظارت می‌کنند.

۴-۴-۱۰ نهایی‌سازی تعریف اشیا

تعریف عملیات، آخرین مرحله در کامل کردن تشخیص اشیاست. عملیات اضافه‌ای را می‌توان با در نظر گرفتن **تاریخچه حیات** یک شیء و پیغام‌هایی که در میان اشیای تعریف‌شده در سیستم مبادله می‌شوند، تعیین کرد. تاریخچه حیات کلی یک شیء را می‌توان با در نظر گرفتن این نکته تعیین کرد که شیء باید به شیوه‌های دیگری ایجاد، اصلاح، دستکاری یا فراخوانی و احتمالاً حذف شود.

فصل یازدهم

مدل سازی تحلیل شیء گرا

OOA ریشه در مجموعه‌ای از اصول بنیادی دارد که مبنای روش OOA را تشکیل می‌دهند که در فصل ۴ معرفی شدند. برای ساخت یک مدل تحلیل، پنج اصل بنیادی زیر به کار برده می‌شود:

- ۱- دامنه اطلاعاتی مدل سازی می‌شود.
 - ۲- عملکرد توصیف می‌شود.
 - ۳- رفتار نمایش داده می‌شود.
 - ۴- مدل‌های داده‌ای، عملیاتی و رفتاری افزای می‌شوند تا جزئیات بیشتری در معرض دید قرار گیرند.
 - ۵- مدل‌های اولیه، بنیاد و ماهیت مسئله را نشان می‌دهند، حال آنکه مدل‌های نهایی، جزئیات ساده‌ای را نمایش می‌دهند.
- هدف OOA تعریف تمام کلاس‌هایی است که به نوعی با مسئله ارتباط دارند و شامل عملیات، صفات مرتبط با آنها، روابط میان آنها و رفتاری است که از خود نشان می‌دهند. برای این منظور، چند کار باید صورت گیرد:

- خواسته‌های اصلی کاربر باید بین مشتری و مهندس تبادل شود.
- کلاس‌ها باید شناسایی شوند.
- سلسله‌مراتب کلاس‌ها باید مشخص شود.
- روابط شیء با شیء باید نشان داده شود.
- رفتار اشیا نیز باید مدل سازی شود.

گرچه مراحل، اصطلاح‌شناسی و فرایندهای هر یک از روش‌های OO متفاوت است، ولی فرایندهای کلی تحلیل شیء گرا بسیار مشابه‌اند. مهندس نرم‌افزار برای اجرای تحلیل شیء گرا باید مراحل کلی زیر را دنبال کند:

- شناسایی نیازهای مشتری برای سیستم
- شناسایی سناریوها یا موردهای کاربرد
- انتخاب کلاس‌ها و اشیا با استفاده از نیازها (خواسته‌ها) به عنوان یک راهنما
- شناسایی صفات و عملیات مربوط به هر یک از اشیا سیستمی
- تعریف ساختارها و سلسله‌مراتبی که کلاس‌ها را سازماندهی می‌کنند
- ساخت یک مدل شیء-واسط
- ساخت یک مدل شیء-رفتار

- بازبینی مدل تحلیل با توجه به موردهای کاربرد/ سناریوها
بنابراین مدل پیشنهادی تحلیل شیء گرا به صورت زیر خواهد بود:



شکل ۱-۱۱ مدل تحلیل نیازهای شیء گرا

۱-۱۱ زبان مدل سازی یکنواخت

زبان یکنواخت و یکپارچه مدل سازی (UML)^۱ زبان مدل سازی گرافیکی است که با ترکیبی از نمادها به تشریح عناصر اصلی سیستم‌های نرم افزاری (که در UML به آن محصول گفته می‌شود) می‌پردازد. UML شامل سه گروه مدل به شرح زیر است:

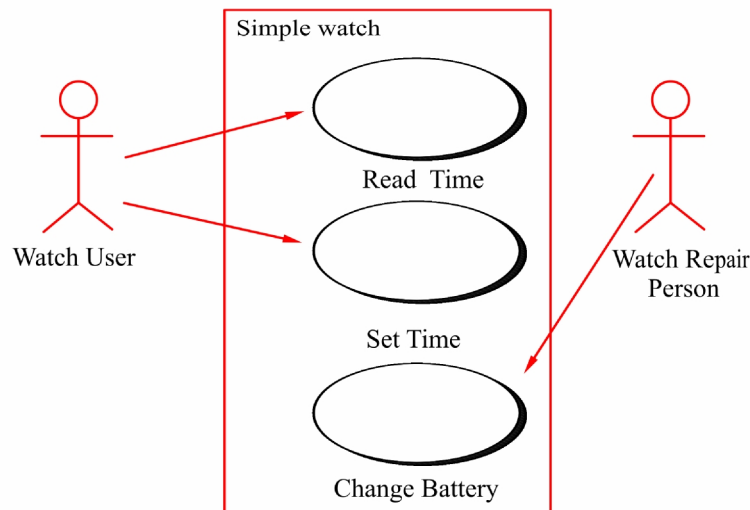
- **مدل تابعی^۲:** عملکرد سیستم را از دید کاربر شرح می‌دهد. در UML برای این مدل سازی از نمودارهای Use-Case استفاده می‌شود.
- **مدل شیء^۳:** ساختار سیستم را به صورت مجموعه‌ای از اشیا، صفات، عملیات و ارتباطات شرح می‌دهد. در UML برای این مدل سازی از نمودار کلاس (Class Diagram) استفاده می‌شود.
- **مدل پویا^۴:** رفتار داخلی سیستم را شرح می‌دهد. در UML برای این مدل سازی از نمودارهای Sequence، Statechart، و Activity استفاده می‌شود.

۱-۱-۱۱ نمونه‌ای از نمودار مورد کاربرد

نمودار مورد کاربرد^۵ در طول استخراج نیازها و تحلیل سیستم برای مشخص کردن عملکرد سیستم ایجاد می‌شود. Use Case ها بر رفتار سیستم از دیدگاه خارج از سیستم تمرکز می‌کنند. یک Use Case در واقع عملیاتی را شرح می‌دهد که توسط سیستم تهیه شده و نتیجه‌ای مشخص برای یک بازیگر (Actor) دارد.

بازیگر یک موجودیت است که با سیستم در حال تعامل است. (مانند کاربر، یک سیستم دیگر و ...). نمونه‌ای از نمودار مورد کاربرد برای عملیات سیستم یک ساعت مچی در شکل زیر نشان داده شده است. در این مورد کاربرد Watch Repair Person و Watch User دو بازیگر هستند که با این سیستم در تعامل هستند.

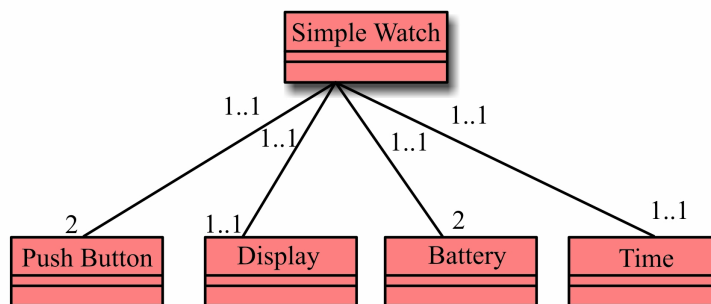
1 - Unified Modelling Language
2 - Functional Model
3 - Object Model
4 - Dynamic Model
5 - Use Case



شکل ۲-۱۱ نمودار مورد کاربرد سیستم ساعت مچی

۲-۱-۱۱ نمونه‌ای از نمودار کلاس

از نمودار کلاس^۱ برای نشان دادن ساختار سیستم استفاده می‌شود. کلاس‌ها انتزاعی از ساختارهای مشترک و رفتار مجموعه‌ای از اشیا هستند. اشیا نمونه‌هایی از کلاس‌ها هستند که در حین اجرای سیستم ساخته می‌شوند، تغییر می‌کنند و غالباً بعد از پایان اجرای سیستم از بین می‌روند. در شکل زیر نمونه‌ای از نمودار کلاس، همراه روابط بین آن‌ها برای مورد کاربرد رسم‌شده در مثال مذکور نشان داده شده است.

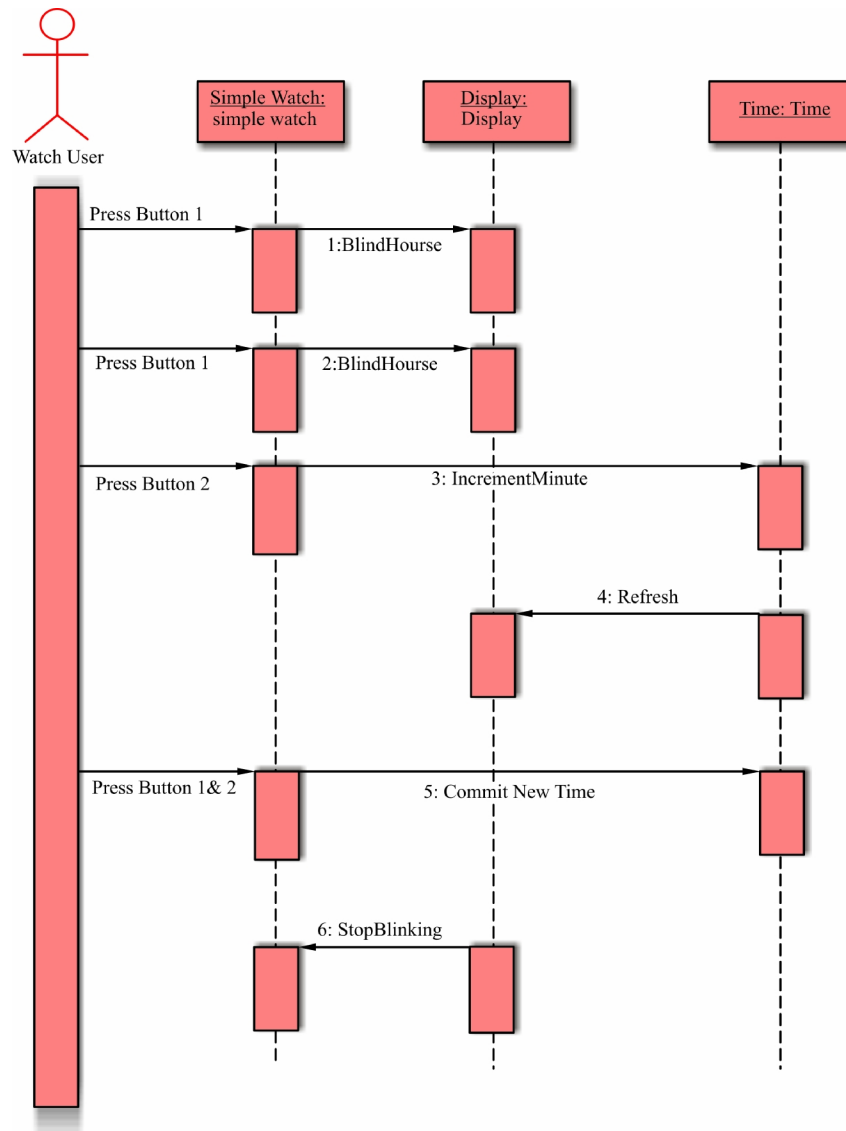


شکل ۳-۱۱ نمودار سلسله‌مراتبی کلاس‌های سیستم ساعت مچی

۳-۱-۱۱ نمونه‌ای از نمودار توالی

نمودار توالی^۲ برای فرموله کردن رفتار سیستم و همچنین رخ دادن ارتباطات بین اشیا به کار می‌رود. نمودار توالی برای نشان دادن اشیا^۳ که در یک Use Case سهیم هستند مفید است. به همین دلیل به اشیا^۳ که در یک نمودار توالی نشان داده می‌شوند، اشیا شرکت‌کننده^۳ می‌گویند.

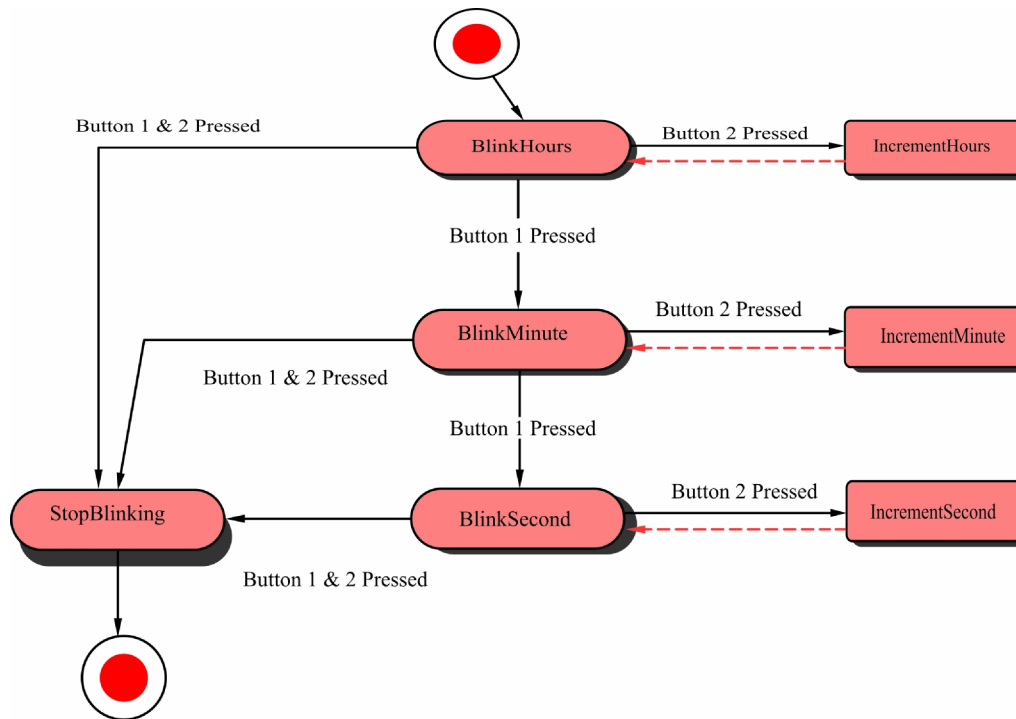
1 - Class Diagram
2 - Sequence Diagram
3 - Participating Object



شکل ۱۱-۴ نمودار توالی سیستم ساعت مچی

۱۱-۴ نمونه‌ای از نمودار حالت

نمودار حالت^۱، رفتار یک شیء مشخص را با تعدادی حالت و حرکت بین این حالات را شرح می‌دهد. یک شیء در یک حالت، مقادیر مشخصی برای ویژگی‌های خود دارد. منظور از حرکت، رفتن شیء از یک حالت به حالت دیگر با رخ دادن یک رویداد خاص است. در نمودار توالی تمرکز روی پیام‌هایی است که تحت تأثیر یک رویداد رخ داده توسط بازیگر، بین اشیای مختلف مبادله می‌شود. اما در نمودار حالت تمرکز بر انتقال بین حالت‌ها است که این انتقال ناشی از وقوع یک رویداد برای آن شیء خاص است. شکل زیر نمودار توالی سیستم ساعت مچی را نشان می‌دهد.



شکل ۵-۱۱ نمودار حالت سیستم ساعت مچی

۲-۱۱ فرایند تحلیل شیء گرا

فرایند تحلیل شیء گرا (OOAP)^۱ در ابتدا کاری با خود اشیا ندارد بلکه با درک شیوه استفاده از سیستم توسط کاربران انسانی (اگر تعامل با انسان وجود داشته باشد)، توسط ماشینها (اگر در کنترل فرایند به کار گرفته شده باشد) یا توسط برنامه‌های دیگر (اگر هدف، کنترل و هماهنگی برنامه‌های کاربردی مد نظر باشد) آغاز می‌شود. پس از جمع‌آوری نیازهای مشتری اقدام به مدل‌سازی تحلیل برای یک سیستم به شرح زیر می‌شود:

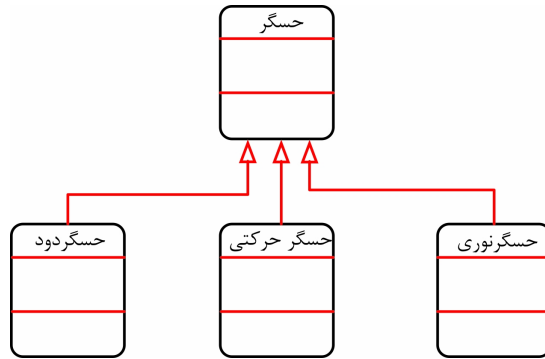
- طراحی موردهای کاربرد^۲
- مدل‌سازی مسئولیت و مشارکت کلاس‌ها^۳
- تعریف ساختارها، سلسله‌مراتب^۴ کلاس‌ها
- مدل‌سازی روابط بین اشیا
- ساخت مدل رفتار اشیا
- تعریف موضوع‌ها و زیرسیستم‌ها^۵

۳-۲-۱۱ تعریف ساختار و سلسله‌مراتب کلاس‌ها

- ساختار کلاس تعمیم - تخصیص^۶

در این ساختار یک کلاس به عنوان زیرکلاس، کلاس دیگر مطرح می‌شود و تمام صفات و عملیات‌های آن را به ارث می‌برد. مثال:

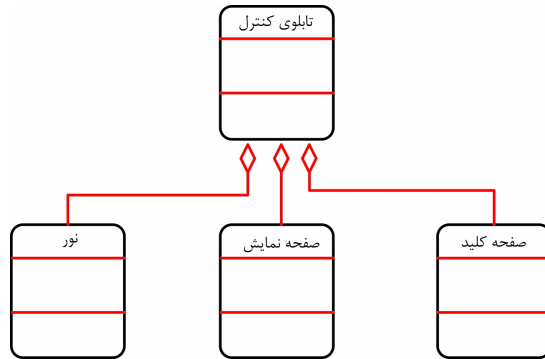
1 - Object Oriented Analysis Process
 2 - Use-Cases Design
 3 - Class-Responsibility-Collaborator Modelling
 4 - Defining Structures and Hierarchies
 5 - Defining Subjects and Subsystems
 6 - Generalization- Specialization



شکل ۱۱-۶ نمودار کلاس‌ها برای تعمیم- تخصیص

• ساختار کلاس تجمع^۱

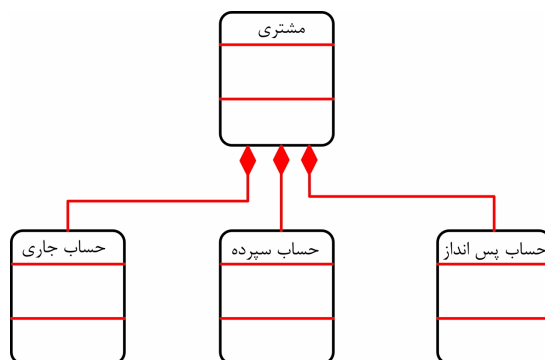
در این ساختار یک شیء از چند مؤلفه تشکیل شده است که این مؤلفه‌ها را می‌توان به عنوان یک شیء تعریف کرد. مثال:



شکل ۱۱-۷ نمودار کلاس‌ها برای کلاس‌های مجتمع مرکب

• ساختار کلاس ترکیب^۲

شکل خاصی از رابطه تجمع است که یک شیء شامل چندین شیء دیگر است و وقتی شیء اصلی حذف شود، همه اشیای موجود در آن نیز از بین می‌روند.

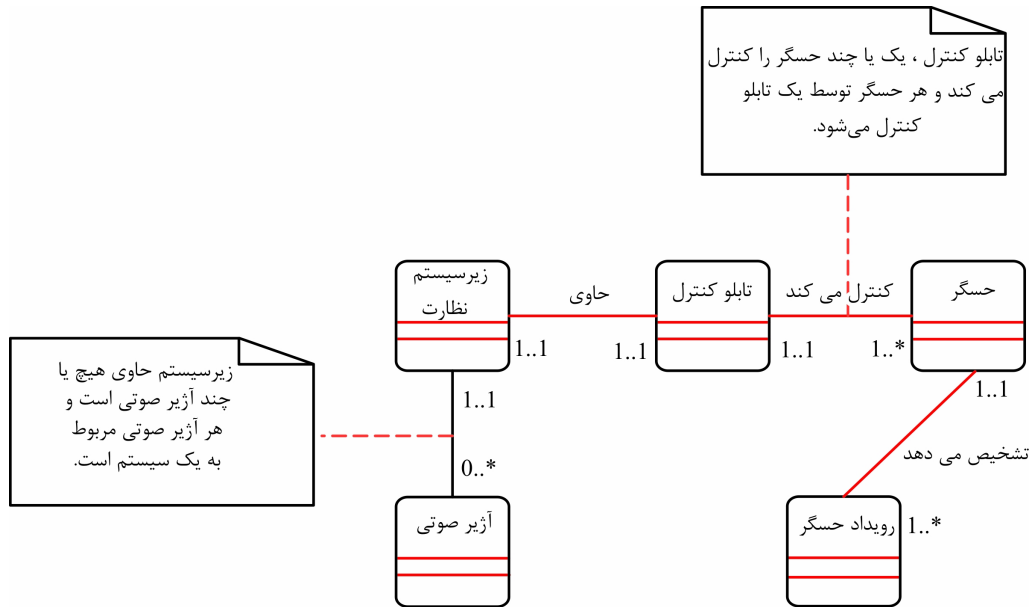


شکل ۱۱-۸ نمودار کلاس‌ها برای کلاس‌های ترکیبی

۴-۲-۱۱ مدل روابط بین اشیا

مدل روابط میان اشیا مانند شکل زیر در سه مرحله به دست می آید:

- ۱- با استفاده از کارت‌های شاخص مدل CRC، مجموعه‌ای از اشیای مشارکت کننده رسم می‌شوند.
- ۲- با بازبینی کارت‌های شاخص مدل CRC، مسئولیت‌ها و مشارکت کننده‌ها مورد ارزیابی قرار گرفته و ارتباطات نامگذاری می‌شوند.
- ۳- هنگامی که روابط با نام مشخص شد، نحوه مشارکت کلاس‌ها در ارتباط، ارزیابی و مدل نهایی به دست می‌آید.



شکل ۹-۱۱ نمودار روابط بین اشیا

۵-۲-۱۱ مدل رفتار اشیا^۱

مدل رفتار اشیا نشان می‌دهد که سیستم شیء‌گرا چگونه به رویدادها و محرک‌های خارجی پاسخ می‌دهد. برای این کار از نمودارهای توالی و حالت استفاده می‌شود.

• مراحل ایجاد مدل رفتار

- ۱- ارزیابی تمام Use-Case ها برای درک کامل ترتیب تعامل‌ها در سیستم
- ۲- شناسایی رویدادهایی که ترتیب تعامل‌ها را هدایت می‌کنند و درک چگونگی ارتباط این رویدادها با اشیای خاص
- ۳- ایجاد یک پیگرد رویداد برای هر Use-Case
- ۴- ساختن نمودار توالی و حالت
- ۵- بازبینی مدل رفتار اشیا برای اعتبارسنجی، صحت و سازگاری

۶-۲-۱۱ تعریف موضوع‌ها و زیرسیستم‌ها

هرگاه گروهی از کلاس‌ها با یکدیگر مشارکت کنند تا مجموعه‌ای از مسئولیت‌های منسجم را به وجود آورند، می‌توان آن‌ها را به عنوان یک زیرسیستم در نظر گرفت. در UML زیرسیستم را بسته^۲ می‌نامند.

فصل دوازدهم

طراحی شیء گرا

برخلاف مدل‌های طراحی نرم‌افزار سنتی، OOD منجر به طراحی می‌شود که شامل سطوح متفاوتی از پیمانها است. مؤلفه‌های سیستم اصلی به صورت زیرسیستم یا پیمانها سطح سیستمی سازماندهی می‌شوند. داده‌ها و عملیاتی که داده‌ها را دستکاری می‌کنند، در اشیا (یک شکل پیمانهای که مؤلفه سازنده سیستم‌های OO است) بسته‌بندی می‌شوند. به علاوه، OOD باید سازمان داده‌های مربوط به صفات و جزئیات رویه‌ای هر یک از عملیات را توصیف کند.

۱-۱۲ لایه‌های طراحی شیء گرا

در فصل ۶ با مفهوم هرم طراحی نرم‌افزارهای سنتی آشنا شدیم. چهار لایه طراحی شامل داده، معماری، واسط و سطح مؤلفه‌ها تعریف و مورد بحث قرار گرفت. برای سیستم‌های شیء گرا نیز می‌توان یک هرم طراحی تعریف کرد، ولی در اینجا لایه‌های متناظر قدری تفاوت دارند. چهار لایه هرم طراحی شیء گرا عبارت‌اند از:

لایه زیرسیستم: شامل نمایشی از هر یک از زیرسیستم‌ها است که نرم‌افزار را قادر می‌کند تا به نیازهای تعیین‌شده توسط مشتری دست پیدا کند و زیرساخت فنی پشتیبان پیاده‌سازی نیازهای مشتری است.

لایه کلاس‌ها و اشیا: شامل سلسله‌مراتبی از کلاس‌هاست که امکان ایجاد سیستم را با استفاده از تعمیم‌ها و تخصیص‌های هدفمند فراهم می‌آورد. این لایه همچنین شامل نمایشی از کلیه اشیا است.

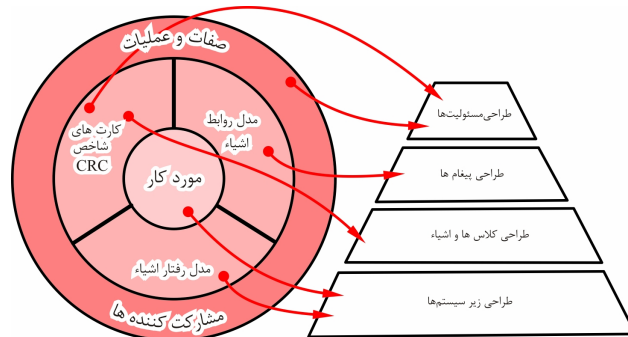
لایه پیغام‌رسانی: شامل جزئیاتی از طراحی است که هر کلاس را قادر به برقراری ارتباط با مشارکت‌کننده‌های خود می‌سازد. این لایه واسط‌های داخلی و خارجی را برای سیستم ایجاد می‌کند.

لایه مسئولیت‌ها: شامل طراحی ساختمان داده‌ها و الگوریتم‌های مورد نیاز برای کلیه صفات و عملیات مربوط به هر شیء می‌شود.

هرم طراحی انحصاراً بر طراحی محصول یا سیستمی مشخص تأکید دارد. ولی لازم به ذکر است که لایه دیگری از طراحی نیز وجود دارد که لایه مبنایی نامیده شده و هرم بر آن استوار است. لایه مبنای طراحی اشیا تأکید دارد. اشیا دامنه نقشی کلیدی در ایجاد زیرساختی برای سیستم مبتنی بر شیء‌گرایی ایفا می‌کنند. زیرا فعالیت‌های واسط انسان-رایانه، مدیریت وظایف و مدیریت داده‌ها را پشتیبانی می‌کنند. اشیا دامنه را می‌توان برای افزودن اطلاعات بیشتر به برنامه کاربردی نیز به کار برد.

۲-۱۲ تبدیل مدل تحلیل شیء گرا به مدل طراحی شیء گرا

اگرچه میان مدل‌های طراحی سنتی و شیء گرا شباهت وجود دارد، ولی تصمیم گرفتیم لایه‌های هرم طراحی را تغییر دهیم تا ماهیت طراحی شیء گرا را به طور صحیحی منعکس کند. شکل زیر رابطه میان مدل تحلیل و مدل طراحی شیء گرای حاصل از آن را نشان می‌دهد.



شکل ۱۲-۱ رابطه بین مدل‌های تحلیل و طراحی شیء گرا

اگرچه اصطلاحات و مراحل فرایند برای هر یک از روش‌های طراحی شیء گرا ارائه شده توسط محققین مختلف متفاوت است، ولی ماهیت کل فرایند طراحی شیء گرا یکسان است. مهندس نرم‌افزار برای انجام طراحی بر مبنای شیء گرای بیاید مراحل زیر را اجرا کند:

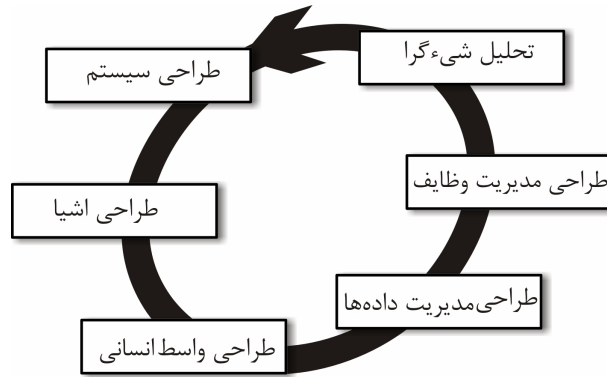
- ۱- توصیف کلیه زیرسیستم‌ها و اختصاص دادن آن به پردازش‌ها و وظایف
- ۲- انتخاب یک راهبرد طراحی برای پیاده‌سازی مدیریت داده‌ها، پشتیبانی واسط‌ها و مدیریت وظایف
- ۳- طراحی یک راهکار کنترلی مناسب برای سیستم
- ۴- طراحی اشیا با ایجاد یک نمایش رویه‌ای برای هر عمل و ساختمان داده‌ها برای صفات کلاس‌ها
- ۵- طراحی پیغام‌ها با استفاده از مشارکت میان اشیا و روابط میان آن‌ها
- ۶- ایجاد مدل نهایی طراحی
- ۷- بازبینی مدل طراحی و تکرار آن تا حد کفایت

۳-۱۲ روش یکنواخت برای طراحی شیء گرا

Jacobson و Rumbaugh، Booch، Grady بهترین ویژگی‌های روش‌های تحلیل و طراحی شیء گرا را کنار هم قرار دادند و روشی یکنواخت حاصل شد. نتیجه کار که زبان مدل‌سازی یکنواخت (UML)^۱، خوانده می‌شود، در سرتاسر صنعت نرم‌افزار، کاربردی گسترده یافته است. در فرایند مدل‌سازی تحلیل (فصل ۱۱)، دیدگاه‌های مدل کاربر و مدل تحلیل ارائه شدند. این دیدگاه‌ها شناختی از سناریوهای استفاده (که راهنمایی برای مدل‌سازی رفتاری هستند ارائه داده و با شناسایی و توصیف عناصر ساختاری ایستای سیستم، دیدگاه‌هایی از مدل پیاده‌سازی و مدل رفتاری ارائه می‌کنند.

UML در دو فعالیت طراحی عمده سازماندهی می‌شود: طراحی سیستم و طراحی اشیا.

هدف اصلی طراحی سیستم به صورت یکنواخت، نمایش دادن معماری نرم‌افزار است. جریان فرایند از تحلیل به طراحی در شکل زیر نیز نشان داده شده است. در سرتاسر فرایند طراحی با UML، دیدگاه مدل کاربر و دیدگاه مدل ساختار، به نمایش طراحی بسط داده می‌شود.



شکل ۱۲-۲ جریان فرایند طراحی شیء گرا

۴-۱۲ فرایند طراحی شیء گرای سیستم

فرایند طراحی شیء گرای سیستم شامل فعالیت‌های زیر می‌شود:

- افراز مدل تحلیل به زیرسیستم‌ها (لایه‌بندی زیرسیستم‌ها)
- شناسایی همزمانی و تخصیص زیرسیستم‌ها
- تخصیص زیرسیستم‌ها به پردازش‌ها و وظایف (مدیریت وظایف)
- توسعه یک طراحی برای واسط کاربر
- انتخاب یک راهبرد پایه برای پیاده‌سازی مدیریت داده‌ها
- شناسایی منابع سراسری و راهکارهای کنترلی مورد نیاز برای دستیابی به آن‌ها (مدیریت منابع)
- طراحی یک راهکار کنترلی مناسب برای سیستم از جمله مدیریت وظایف
- در نظر گرفتن چگونگی پرداختن به شرایط مرزی
- بازبینی و بازنگری مدل طراحی

۵-۱۲ فرایند طراحی اشیا

طراحی شیء گرا را با توجه به استعاره‌ای که در این متن به کار گرفته شد، می‌توان به عنوان نقشه طرح یک ساختمان در نظر گرفت. نقشه طرح، هدف از ساخت هر اتاق و ویژگی‌های معماری، اتصال اتاق‌ها به یکدیگر و به محیط خارج را مشخص می‌سازد. اکنون زمان آن فرا رسیده که جزئیات مورد نیاز برای ساخت هر اتاق تهیه شود. در حیطه طراحی شیء گرا، **طراحی اشیا** بر طراحی اتاق‌ها تأکید دارد. در این مرحله اصول و مفاهیم پایه‌ای مرتبط با طراحی در ساختار مؤلفه‌ها مطرح می‌شوند. ساختمان داده‌های محلی (برای صفات) تعیین و الگوریتم‌ها (برای عملیات) طراحی می‌شوند.

• توصیف اشیا

- توصیف طراحی یک شیء (نمونه‌ای از یک کلاس یا زیرکلاس) می‌تواند به یکی از دو شکل زیر انجام گیرد:
- توصیف قرارداد که واسط شیء را با تعریف هر پیغامی که شیء می‌تواند دریافت کند و عملی که شیء هنگام دریافت آن پیغام اجرا می‌کند، برقرار می‌سازد، یا
 - توصیف پیاده‌سازی مربوط به عمل درخواست‌شده توسط پیغام را نشان می‌دهد. جزئیات پیاده‌سازی شامل اطلاعاتی درباره بخش خصوصی شیء یعنی جزئیات ساختمان داده‌ای صفات شیء و جزئیات رویه‌ای نشان‌دهنده عملیات شیء است.

• طراحی الگوریتم‌ها و ساختمان داده‌ها

تنوع نمایش‌های موجود در مدل تحلیل و طراحی سیستم، مشخصه‌ای برای کلیه عملیات و صفات فراهم می‌آورند. الگوریتم‌ها و ساختمان‌های داده‌ای با استفاده از روشی طراحی می‌شوند که با روش‌های طراحی داده‌ها و طراحی در سطح مؤلفه، در روش سنتی کمی تفاوت دارد. برای پیاده‌سازی مشخصات مربوط به هر عمل، الگوریتمی ایجاد می‌شود. در بسیاری از موارد، الگوریتم یک دنباله محاسباتی یا رویه‌ای است که می‌توان آن را به صورت یک پیمانه نرم‌افزاری مستقل پیاده کرد. ولی، اگر مشخصات عملی پیچیده باشد، ممکن است نیاز به پیمانه‌ای کردن عمل باشد. برای این منظور می‌توان از تکنیک‌های سنتی طراحی در سطح مؤلفه‌ها استفاده کرد. ساختمان داده‌ها همزمان با الگوریتم‌ها طراحی می‌شوند. از آنجاکه عملیات، صفات یک کلاس را دستکاری می‌کنند، طراحی ساختمان داده‌ها که صفات را به خوبی منعکس کند، طراحی الگوریتم‌های عملیات مربوط را بسیار دشوار می‌سازد.

فصل سیزدهم

آزمون نرم افزار و راهبردها

آزمایش نرم افزار عنصری حیاتی از تضمین کیفیت نرم افزار است و بیانگر مرور و بازنگری نهایی مشخصه سیستم، تحلیل، طراحی و برنامه های مبدأ است.

۱۳-۱ شیوه راهبردی آزمایش نرم افزار

آزمایش، شامل مجموعه ای از فعالیت ها است که می تواند از قبل به صورت سیستماتیک برنامه ریزی و هدایت شود. راهبردهای آزمایش نرم افزار همگی خصوصیات زیر را دارند:

- آزمایش از سطح مؤلفه شروع می شود، به سمت خارج در جهت یکپارچه سازی کل سیستم رایانه ای پیش می رود.
- روش های متفاوت آزمایش، در نقاط زمانی مختلف مناسب هستند.
- آزمایش توسط توسعه دهنده نرم افزار و برای پروژه های بزرگ توسط گروه مستقل آزمایش، هدایت می شود.
- آزمایش و اشکال زدایی فعالیت های متفاوتی هستند، اما اشکال زدایی باید با هر راهبرد آزمایش همراه باشد.

یک راهبرد برای آزمایش نرم افزار باید آزمایش های سطح پایینی را هدایت کند که برای بازبینی صحت پیاده سازی یک قطعه کد کوچک لازم هستند. همچنین این راهبرد باید آزمایش های سطح بالایی را سازماندهی کند که اکثر توابع سیستم را در رابطه با نیازهای مشتری اعتبارسنجی می کنند. یک راهبرد باید رهنمودهایی را برای مجری و مجموعه ای از علائم نشان دهنده را برای مدیر فراهم کند.

۱۳-۲ راهبرد آزمایش نرم افزار

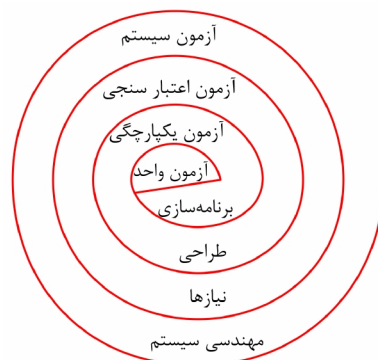
فرایند مهندسی نرم افزار را می توان به صورت یک مارپیچ مانند شکل زیر در نظر گرفت. در ابتدا، مهندس سیستم، نقش نرم افزار را تعریف می کند و به تحلیل نیازهای نرم افزار وارد می شود که دامنه اطلاعات، عملکرد، رفتار، کارایی، محدودیت ها و معیارهای اعتبارسنجی برای نرم افزار ایجاد می شود. با حرکت در مارپیچ به سمت طراحی و در نهایت به برنامه نویسی می رسیم. به منظور توسعه نرم افزار، به سمت درون مارپیچ حرکت می کنیم تا در هر دور، سطح انتزاع کاسته شود.

یک راهبرد برای آزمایش نرم افزار می تواند در این مارپیچ مورد توجه قرار گیرد. **آزمایش واحد**، از مرکز مارپیچ شروع می شود و بر روی هر واحد (یعنی مؤلفه) نرم افزار متمرکز می شود که با برنامه های مبدأ پیاده سازی شده است. با حرکت به سمت خارج در مارپیچ، آزمایش به سمت

آزمایش یکپارچه سازی ادامه می‌یابد. این آزمایش بر طراحی و ساخت معماری نرم افزار تأکید دارد. با حرکت به اندازه یک دور دیگر به سمت خارج ماریج، به آزمایش اعتبارسنجی می‌رسیم.

نیازهایی که به عنوان بخشی از تحلیل نیازهای نرم افزار ایجاد شده‌اند، در مقابل نرم افزاری که ساخته شده اعتبارسنجی می‌شوند. در نهایت باید آزمایش سیستم مورد توجه قرار گیرد که در آن نرم افزار و عناصر دیگر سیستم به صورت یک مجموعه یکپارچه، آزمایش می‌شوند. برای آزمایش نرم افزار، در طول ماریج حرکت کرده و در هر دور، محدوده آزمایش را گسترش می‌دهیم.

بنابراین آزمایش واحد، اقدام به آزمایش هر مؤلفه به صورت مجزا می‌کند تا اطمینان حاصل شود که مؤلفه به صورت یک واحد، درست کار می‌کند. **آزمایش واحد، از تکنیک‌های آزمایش جعبه سفید بهره می‌برد و مسیرهای خاصی را در ساختار کنترلی پیمانه بررسی می‌کند تا از صحت آن اطمینان حاصل شود.**



شکل ۱-۱۳ فرایند راهبردی آزمایش نرم افزار

سپس مؤلفه‌ها باید مونتاژ یا یکپارچه شوند تا بسته نرم افزاری کامل را تشکیل دهند. آزمایش یکپارچه سازی، مسایل مربوط به مشکلات دوگانه بازبینی و ساخت برنامه را مورد توجه قرار می‌دهد. تکنیک‌های طراحی نمونه‌های آزمایش جعبه سیاه، در زمان یکپارچه سازی بیشترین استفاده را دارند.

آزمایش اعتبارسنجی، اطمینان نهایی را ایجاد می‌کند که نرم افزار تمام نیازهای عملکردی، رفتاری و کارایی را برآورده می‌کند. تکنیک‌های آزمایش جعبه سیاه به طور انحصاری در ضمن اعتبارسنجی استفاده می‌شوند.

آخرین مرحله آزمایش مرتبه بالا، خارج از مرز مهندسی نرم افزار است و در چارچوب مهندسی سیستم رایانه‌ای مطرح است. نرم افزار، پس از اعتبارسنجی، باید با عناصر دیگر سیستم ترکیب شود (برای مثال، سخت افزار، افراد، بانک‌های اطلاعاتی). در **آزمایش سیستم** اقدام به بازبینی و کنترل یکپارچگی تمام عناصر به طور منظم می‌شود و عملکرد و کارایی کل سیستم نیز بررسی و تأیید می‌شود.

۳-۱۳ مبانی آزمایش نرم افزار

آزمایش، موارد غیر معمول جالبی را برای مهندس نرم افزار آشکار می‌کند. مهندس نرم افزار در اثنای انجام فعالیت‌های اولیه مهندسی نرم افزار، سعی در توسعه نرم افزار با استفاده از مفهومی مجرد و به دست آوردن محصولی واضح و کامل دارد. اینک آزمایش باید انجام شود.

۱-۳-۱۳ اهداف آزمایش

Myers چند قانون زیر را بیان می‌کند که اهداف مناسبی برای آزمایش هستند:

- آزمایش فرایند اجرای برنامه با هدف یافتن خطاست.
- یک نمونه آزمایش خوب، نمونه‌ای است که با احتمال بالایی خطاها را بیابد.
- آزمایش موفق، آزمایشی است که خطاهای تاکنون پیدانشده را بیابد.

۱۳-۳-۲ اصول آزمایش

قبل از به کارگیری روش‌های طراحی نمونه‌های آزمایش مؤثر، مهندس نرم‌افزار باید درک درستی از اصول اولیه آزمایش نرم‌افزار را داشته باشد. Davis مجموعه‌ای از اصول زیر را پیشنهاد می‌کند:

- تمام آزمایش‌ها باید بر اساس نیازهای مشتری قابل پیگیری باشند.
- آزمایش‌ها باید مدتی طولانی قبل از شروع آزمایش طراحی و برنامه‌ریزی شوند.
- اصل پاریتو^۱ برای آزمایش نرم‌افزار به کار گرفته شود (قانون ۸۰-۲۰).
- آزمایش باید با "توجه به اجزا" شروع شود و به سمت آزمایش "کلی" پیش رود.
- برای داشتن بیشترین تأثیر، آزمایش باید توسط تیم مستقلی هدایت شود.
- آزمایش کامل و جامع امکان‌پذیر نیست.

۱۳-۳-۳ قابلیت آزمایش

در موارد ایده‌آل، مهندس نرم‌افزار اقدام به طراحی و توسعه برنامه رایانه‌ای، یک طرح سیستمی یا محصولی را با در نظر داشتن قابلیت آزمایش می‌کند.

- عملیاتی بودن^۲: "نرم‌افزار هرچه بهتر کار کند، با کارایی بالاتری آزمایش می‌شود."
- قابلیت مشاهده^۳: "آنچه می‌بینید همان است که آزمایش می‌کنید."
- قابلیت کنترل^۴: "هرچه نرم‌افزار بهتر کنترل شود، آزمایش بیشتر به طور خودکار و بهینه‌پذیر انجام‌پذیر است."
- تجزیه‌پذیری^۵: "با کنترل دامنه کاربرد آزمایش، می‌توان مسایل را با سرعت بیشتری تجزیه کرد و آزمایش‌های مجدد را با هوشمندی انجام داد."
- سادگی^۶: "هرچه مورد برای آزمایش کمتر باشد، آزمایش با سرعت بیشتری انجام می‌گیرد."
- پایداری^۷: "هرچه تغییرات کمتر باشد، انحراف از آزمایش کمتر است."
- قابلیت فهم^۸: "هرچه اطلاعات بیشتری در اختیار داشته باشیم، آزمایش هوشمندانه‌تر انجام می‌شود."

۱۳-۳-۴ طراحی نمونه‌های آزمایش

طراحی نمونه آزمایش‌هایی برای نرم‌افزار و محصولات مهندسی دیگر می‌تواند به اندازه طراحی اولیه خود محصول متغیر باشد. با این وجود، مهندسين نرم‌افزار اغلب با آزمایش به عنوان فعالیتی نهایی برخورد می‌کنند و نمونه‌های آزمایش را طوری طراحی می‌کنند که ظاهراً درست هستند اما اطمینان کمی از کامل بودن آن‌ها وجود دارد. اهداف آزمایش را به خاطر آورید که باید بر اساس آن‌ها آزمایش‌هایی طراحی شوند که احتمال بالایی برای یافتن اکثر خطاها، با حداقل مقدار زمان و فعالیت داشته باشند.

هر محصول مهندسی (و اکثر چیزهای دیگر) می‌تواند به یکی از دو روش زیر آزمایش شود:

- ۱- با دانستن عملکرد تابع خاصی که یک محصول برای آن طراحی شده است، آزمایش‌هایی طراحی می‌شوند که مشخص می‌کنند هر یک از توابع کاملاً عملیاتی هستند و درعین حال در هر تابع برای یافتن خطاها جستجو صورت می‌گیرد (آزمایش جعبه سیاه).
- ۲- با دانستن عملکرد داخلی محصول، آزمایش‌ها به گونه‌ای طراحی می‌شوند که تعیین کنند اعمال داخلی مطابق با مشخصه‌های تعریف شده انجام می‌شوند و تمام مؤلفه‌های داخلی به شکل مناسبی آزمایش می‌شوند (آزمایش جعبه سفید).

1 - Pareto
 2 - Operability
 3 - Observability
 4 - Controllability
 5 - Decomposability
 6 - Simplicity
 7 - Stability
 8 - Understandability

۴-۱۳ آزمایش جعبه سفید

آزمایش جعبه سفید گاهی آزمایش جعبه شیشه‌ای هم نامیده می‌شود، یک روش طراحی نمونه‌های آزمایش است که از ساختار کنترل طراحی رویه‌ای برای هدایت نمونه‌های آزمایش استفاده می‌کند. با استفاده از روش‌های آزمایش جعبه سفید، مهندس نرم افزار می‌تواند نمونه‌های آزمایشی را طراحی کند که:

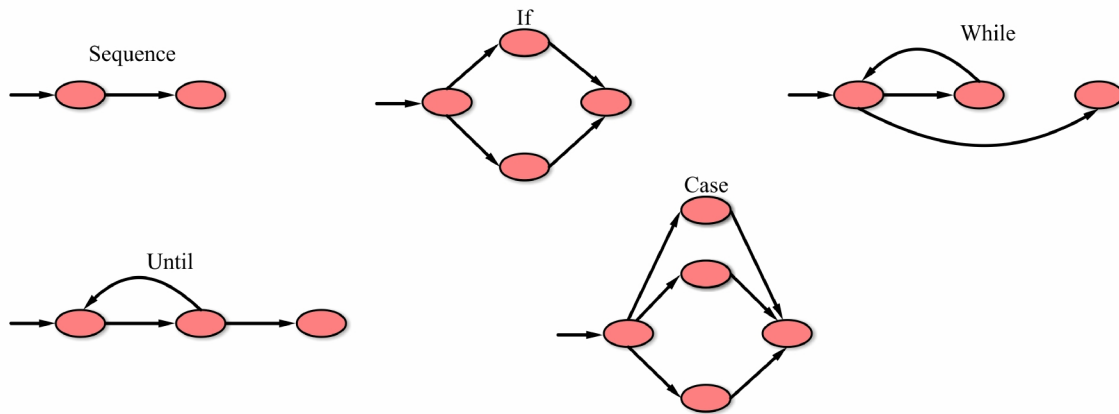
- ۱- تضمین کنند تمام مسیرهای مستقل در پیمانانه حداقل یکبار آزمایش می‌شوند.
 - ۲- تمام تصمیمات شرطی را در دو بخش درست و غلط بررسی کنند.
 - ۳- تمام حلقه‌ها را در شرایط مرزی و در محدوده‌های عملیاتی اجرا کنند.
 - ۴- ساختمان داده‌های داخلی را بررسی کنند تا از اعتبار آن‌ها مطمئن شوند.
- آزمایش جعبه سفید مرتبط با دو بحث مسیر پایه و گراف جریان است.

۱-۴-۱۳ آزمایش مسیر پایه

آزمایش مسیر پایه، یک روش آزمایش جعبه سفید است که ابتدا توسط Mc Cabe پیشنهاد شد. روش مسیر پایه، طراح نمونه‌های آزمایش را وادار می‌کند که اندازه پیچیدگی منطقی طراحی رویه‌ای را به دست آورد و این اندازه را به عنوان راهنمایی برای تعریف مجموعه مسیرهای پایه اجرایی به کار ببرد. مسیر پایه تضمین می‌کند که با اجرای نمونه‌های آزمایش به دست آمده، هر دستور برنامه حداقل یکبار در ضمن آزمایش اجرا می‌شوند.

۲-۴-۱۳ گراف جریان

گراف جریان، جریان کنترل منطقی را با استفاده از نشانه‌گذاری‌های نشان داده شده در شکل‌های زیر به تصویر می‌کشد.



شکل ۲-۱۳ نشانه‌گذاری گراف جریان

۳-۴-۱۳ پیچیدگی دورانی

پیچیدگی دورانی به یکی از سه شکل زیر محاسبه می‌شود:

۱- تعداد نواحی گراف جریان

۲- با استفاده از تعداد یال‌ها و گره‌ها

پیچیدگی دورانی، $V(G)$ ، برای گراف جریان، G ، به این صورت تعریف می‌شود:

$$V(G) = E - N + 2$$

۳- با استفاده از گره‌های گزاره‌ای

پیچیدگی دورانی، $V(G)$ ، برای گراف جریان، G ، با استفاده از گره‌های گزاره‌ای نیز به صورت زیر تعریف می‌شود:

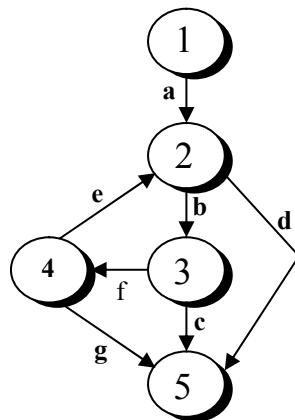
$$V(G) = P + 1$$

۴-۴-۱۳ ماتریس‌های گراف

این روش برای به دست آوردن گراف جریان و حتی مشخص کردن مجموعه‌ای از مسیرهای پایه در حالتی که بخواهیم از مدل مکانیزه استفاده کنیم، مناسب است. به منظور توسعه ابزار نرم‌افزاری که بر پایه آزمایش مسیر پایه عمل کند، ساختمان داده‌ای به نام **ماتریس گراف** معرفی شده است.

ماتریس گراف، ماتریس مربعی است که ابعاد آن (یعنی تعداد سطرها و ستون‌ها) برابر با تعداد گره‌ها در گراف جریان است. در شکل زیر هر گره در گراف جریان با عدد مشخص و هر یال با یک حرف مشخص شده است. یک حرف در ماتریس در محلی متناظر با ارتباط بین دو گره درج می‌شود. به‌رحال با افزودن ارزش اتصال به هر واردهٔ ماتریس، این ماتریس گراف می‌تواند به ابزاری قدرتمند برای ارزیابی ساختار کنترل برنامه در ضمن آزمایش تبدیل شود. در ساده‌ترین شکل، ارزش اتصال، یک (وجود ارتباط) یا صفر (عدم وجود ارتباط) تعریف می‌شود. اما به ارزش‌های ارتباطی، خواص جالب دیگری نیز نسبت داده می‌شود:

- احتمال اینکه یک اتصال (یال) اجرا می‌شود.
- زمان پردازش صرف‌شده در ضمن پیمایش اتصال.
- حافظهٔ لازم در ضمن پیمایش اتصال.
- منابع لازم در ضمن پیمایش آن اتصال.



	۱	۲	۳	۴	۵
۱		a			
۲			b		d
۳				f	c
۴		e			g
۵					

شکل ۳-۱۳ ماتریس گراف متناظر با گراف جریان

۵-۴-۱۳ آزمایش ساختار کنترل

تکنیک آزمایش مسیر پایه، یکی از چند تکنیک آزمایش ساختار کنترلی است. اگرچه آزمایش مسیر پایه ساده و بسیار مفید است، ولی به‌تنهایی کافی نیست. در این بخش، دربارهٔ حالت‌های دیگر آزمایش ساختار کنترلی بحث می‌شود. این حالت‌ها پوشش آزمایش را گسترش داده و کیفیت آزمایش جعبه سفید را افزایش می‌دهند.

۱-۵-۴-۱۳ آزمایش شرط

آزمایش شرط روشی برای طراحی نمونه‌های آزمایش است که شرط‌های منطقی موجود در یک پیمانۀ برنامه را بررسی می‌کند. یک شرط ساده، متغیر بولی یا عبارت رابطه‌ای است، عبارت رابطه‌ای به صورت زیر تعریف می‌شود:

$$E_1 < \text{عملگر رابطه‌ای} < E_2$$

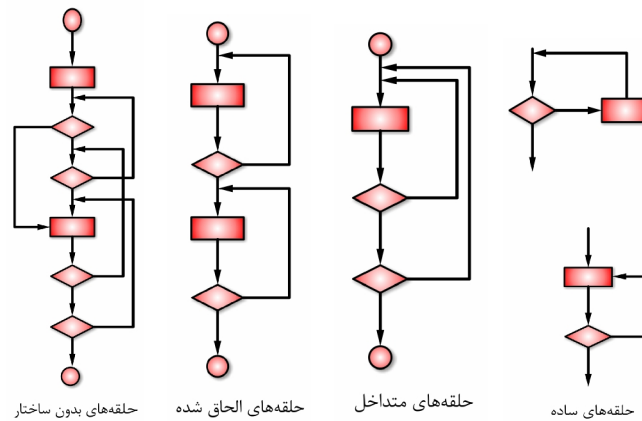
که E_1 و E_2 عبارت‌های محاسباتی هستند و $<$ عملگر رابطه‌ای $>$ شامل یکی از عملگرهای $<$ ، $=$ ، \neq (نامساوی)، $>$ یا \geq است. یک شرط مرکب از دو یا چند شرط ساده، عملگر بولی و پرانتزها تشکیل شده است. فرض می‌کنیم عملگرهای بولی امکان شرط‌های ترکیبی را با اضافه کردن $(|)$ OR، $(\&)$ AND و (\neg) NOT می‌دهند.

اگر یک شرط غلط باشد، حداقل یک مؤلفهٔ آن شرط غلط خواهد بود؛ بنابراین انواع خطاها در شرط به شرح زیر مطرح هستند که باید مورد آزمایش قرار گیرند:

- خطاهای عملگر منطقی (غلط / حذف شده / عملگرهای بولی اضافی)
- خطای متغیر بولی
- خطای پرانتزهای بولی
- خطای عملگر رابطه‌ای
- خطای عبارت محاسباتی

۲-۵-۴-۱۳ آزمایش حلقه

حلقه‌ها برای اکثر الگوریتم‌های پیاده‌سازی شده در نرم‌افزار نقش محوری دارند. با این وجود، اغلب در ضمن هدایت آزمایش‌های نرم‌افزار، توجه کمی به آن‌ها می‌شود.



شکل ۴-۱۳ انواع حلقه‌ها

آزمایش حلقه تکنیکی بر پایه آزمایش جعبه سفید است که منحصراً بر اعتبار ساختارهای حلقه تأکید دارد. چهار رده از حلقه‌ها قابل تعریف هستند: حلقه‌های ساده، حلقه‌های الحاق شده، حلقه‌های متداخل و حلقه‌های بدون ساختار که در شکل بالا نشان داده شده است.

۵-۱۳ آزمایش جعبه سیاه

آزمایش جعبه سیاه که آزمایش رفتاری نیز نامیده می‌شود، بر نیازهای تابعی نرم‌افزار تأکید دارد. آزمایش جعبه سیاه راه حل جایگزینی برای آزمایش جعبه سفید نیست. در عوض، روش مکملی است که احتمالاً رده متفاوتی از خطاها را نسبت به روش‌های جعبه سفید آشکار می‌کند. آزمایش جعبه سیاه سعی در یافتن خطاهایی در دست‌بندی‌های زیر دارد:

- ۱- توابع غلط یا حذف شده
 - ۲- خطاهای واسط‌ها
 - ۳- خطا در ساختمان داده‌ها یا دسترسی به بانک اطلاعاتی خارجی
 - ۴- خطاهای رفتاری یا کارایی
 - ۵- خطاهای آماده‌سازی و اختتامیه
- برخلاف آزمایش جعبه سفید که در اوایل فرایند آزمایش انجام می‌شود، آزمایش جعبه سیاه در مراحل آخر آزمایش به کار گرفته می‌شود؛ چون آزمایش جعبه سیاه عمده به ساختار کنترلی توجهی ندارد و بر دامنه اطلاعات متمرکز است. آزمایش‌های جعبه سیاه به منظور پاسخگویی به سؤالات زیر طراحی می‌شوند:

- چگونه اعتبار عملکردی آزمایش می‌شود؟
- چگونه رفتار و کارایی سیستم آزمایش می‌شود؟
- چه رده‌هایی از ورودی، نمونه‌های آزمایش خوبی می‌سازند؟
- آیا سیستم به مقادیر خاص ورودی حساس است؟
- چگونه مرزهای یک رده از داده‌ها مجزا می‌شود؟

- سیستم چه نواساناتی در مقابل تغییرات سرعت و حجم داده‌ها دارد؟
- ترکیبات خاص داده‌ها چه اثری بر عملکرد سیستم دارند؟

با به‌کارگیری روش‌های آزمایش جعبه سیاه، مجموعه‌ای از نمونه‌های آزمایشی به دست می‌آیند که صحت عملکرد سیستم را تأیید می‌کنند.

۱-۵-۱۳ تحلیل مقادیر مرزی

به دلایلی که کاملاً واضح نیست، تعداد خطاهای نرم‌افزار در مرزهای دامنه ورودی نسبت به مقادیر مرکزی بیشتر است. به همین دلیل روش تحلیل مقادیر مرزی (BVA)^۱ به عنوان یک روش آزمایش توسعه داده شده است. تحلیل مقادیر مرزی باعث انتخاب نمونه‌های آزمایشی می‌شود که مقادیر مرزی را مورد آزمایش قرار می‌دهند.

تحلیل مقادیر مرزی یک روش طراحی نمونه‌های آزمایش است که به‌جای انتخاب هر عنصر از رده مساوی، انتخاب نمونه‌های آزمایش را به لبه‌های این رده هدایت می‌کند. به‌جای تمرکز بر شرایط ورودی، BVA، نمونه‌های آزمایش را از دامنه خروجی به دست می‌آورد. اکثر مهندسی‌ن نرم‌افزار BVA را تا حدی به طور ضمنی اجرا می‌کنند. با به‌کارگیری این رهنمودها، آزمایش مقادیر مرزی کامل‌تر خواهد شد و احتمال بیشتری برای آشکارسازی خطا وجود دارد.

۲-۵-۱۳ آزمایش محیط‌ها، معماری‌ها

نرم‌افزارهای رایانه‌ای پیچیده‌تر شده و نیاز برای شیوه‌های آزمایش خاص نیز رشد کرده است. روش‌های آزمایش جعبه سیاه و جعبه سفید، برای تمام محیط‌ها، معماری‌ها و کاربردها قابل به‌کارگیری هستند، اما رهنمودهای منحصربه‌فرد و شیوه‌هایی برای آزمایش گاهی توصیه می‌شوند. در این بخش، رهنمودهای آزمایش محیط‌ها، معماری‌ها که به طور متداول مهندسی‌ن نرم‌افزار با آن‌ها روبه‌رو می‌شوند، ارائه شده است.

آزمایش واسط‌های گرافیکی کاربران

واسط‌های گرافیکی کاربران (GUI)^۲ زمینه جالبی را برای مهندسی‌ن نرم‌افزار ارائه می‌کنند. به علت اجزای قابل استفاده مجددی که به عنوان بخشی از محیط‌های توسعه GUI فراهم شده‌اند، ایجاد واسط کاربر زمان کمتری نیاز دارد و دقیق‌تر است. اما درعین حال، پیچیدگی GUI‌ها نیز افزایش یافته است و باعث مشکلات بیشتر در طراحی و اجرای نمونه‌های آزمایش می‌شود.

چون بسیاری از GUI‌های مدرن، احساس و جلوه یکسانی دارند، یک‌سری از آزمایش‌های استاندارد قابل انجام است. گراف‌های مدل‌سازی حالت محدود^۳ می‌توانند برای طراحی یک‌سری آزمایش‌ها مورد استفاده قرار گیرند به طوری که اشیا و داده‌های خاص مربوط به GUI برنامه را مورد توجه قرار دهند.

به دلیل ترکیبات زیاد اعمال GUI، آزمایش باید با نمونه‌های خودکار انجام شود. در چند سال اخیر تعداد زیادی از ابزارهای آزمایش GUI به بازار عرضه شده‌اند.

آزمایش معماری مشتری / کارگزار

معماری‌های مشتری / کارگزار (C/S)^۴ موارد مهمی را برای آزمایش‌کننده‌های نرم‌افزار نشان می‌دهند. ماهیت توزیع‌شده محیط‌های C/S، موارد کارایی مربوط به پردازش تراکنش، حضور بالقوه سکوه‌های سخت‌افزاری متفاوت، پیچیدگی‌های ارتباط شبکه، نیاز به ارائه سرویس به چندین مشتری از بانک اطلاعاتی متمرکز (یا توزیع‌شده) و نیازهای هماهنگ‌سازی تحمیل‌شده بر کارگزار، همگی ترکیب شده و باعث می‌شوند آزمایش معماری‌های C/S و نرم‌افزاری که بر روی آن‌ها قرار می‌گیرد، تا حد قابل توجهی مشکل‌تر از کاربردهای مجزا باشد. مطالعات صنعتی اخیر نشان‌دهنده افزایش عمده در زمان و هزینه آزمایش محیط‌های C/S است.

1 - Boundary Value Analysis
2 - Graphical User Interface
3 - Finite State
4 - Client/Server

۳-۵-۱۳ آزمایش مستندات و امکانات راهنما

تعریف نرم افزار را به خاطر بیاورید که در اولین فصل ارائه شد. توجه به این نکته مهم است که آزمایش باید به سومین عنصر پیکربندی نرم افزار، یعنی مستندات نیز توسعه یابد.

خطاها در مستندات می توانند به همان اندازه خطاها در برنامه های مبدأ یا داده ها، باعث اشکالاتی در پذیرش برنامه شوند. هیچ چیز نگران کننده تر از این نیست که راهنمای کاربر یا امکان کمک سیستم دقیقاً دنبال و به نتایج یا رفتاری منتهی شود که منطبق با آنچه در مستندات پیش بینی شده نباشد. به همین دلیل آزمایش مستندات باید بخش معنی داری برای هر طرح آزمایش نرم افزار باشد. آزمایش مستندات در دو مرحله قابل انجام است. اولین مرحله که مرور و بازبینی مستندات نام دارد، مستندات را برای وضوح مطالب و رعایت قواعد نگارشی بررسی می کند. مرحله دوم که آزمایش زنده مستندات نام دارد، مستندات با استفاده از برنامه واقعی، مورد استفاده قرار می دهد. استفاده از برنامه از طریق مستندات پیگیری می شود.

۴-۵-۱۳ آزمایش یکپارچه سازی

کنار هم قرار دادن مؤلفه ها، به معنی ایجاد ارتباط بین آنهاست. داده ممکن است در یک ارتباط ناپدید شود. یک پیمانانه ممکن است اثر نامطلوب و ناخواسته ای را بر دیگری داشته باشد. زیرتوابع، زمانی که با هم ترکیب می شوند، ممکن است تابع بزرگ تر مورد نظر را به وجود نیاورند. مقادیر غیر دقیقی که در هر یک به تنهایی پذیرفته شده اند، ممکن است بزرگ شده و به سطوح غیر قابل قبولی برسند. ساختمان داده های سراسری ممکن است مشکل ساز شود. این لیست همچنان به طور نگران کننده ای ادامه دارد.

آزمایش یکپارچه سازی^۱، روشی سیستماتیک برای ایجاد ساختار برنامه است. هدف، آزمایش مؤلفه ها و ایجاد یکپارچه ساختار برنامه ای است که توسط طراح دیکته شده است.

یکپارچه سازی بالا به پایین

آزمایش یکپارچه سازی بالا به پایین، روشی افزایشی برای ایجاد ساختار برنامه است. پیمانانه ها با حرکت به سمت پایین در سلسله مراتب کنترل، یکپارچه می شوند. کار، با پیمانانه کنترل اصلی^۲ شروع می شود. پیمانانه های پایین تر نسبت به پیمانانه کنترل اصلی در این ساختار به صورت عمقی یا سطحی یکپارچه می شوند.

یکپارچه سازی پایین به بالا

آزمایش یکپارچه سازی پایین به بالا، ساخت و آزمایش نرم افزار را با پیمانانه های ساده (یعنی مؤلفه های پایین ترین سطوح ساختار برنامه) شروع می کند. چون مؤلفه ها از پایین به بالا کنار هم قرار می گیرند، پردازش های لازم برای مؤلفه های سطح بعدی همیشه در دسترس هستند و نیاز به جانگهدار از بین می رود.

۵-۵-۱۳ آزمایش رگرسیون^۳

هر بار که پیمانانه جدیدی به عنوان بخشی از آزمایش یکپارچه سازی به مجموعه نرم افزار افزوده می شود، نرم افزار تغییر می کند. مسیرهای جریان داده جدیدی ایجاد می شوند، I/O جدیدی انجام می گیرد و منطق کنترل جدیدی فراخوانی می شود. این تغییرات ممکن است باعث بروز مشکلات با توابعی شوند که قبلاً بدون خطا کار می کردند. آزمایش رگرسیون، اجرای مجدد زیرمجموعه ای از آزمایش هایی است که قبلاً انجام شده اند تا اطمینان حاصل شود که تغییرات، باعث انتشار اثرات جانبی ناخواسته نمی شوند.

۶-۵-۱۳ آزمایش دود

آزمایش دود^۱ یک روش یکپارچه سازی است و به شکل متداول زمانی استفاده می شود که محصولات نرم افزاری کم اهمیت توسعه داده می شوند. در نتیجه، روش آزمایش دود شامل فعالیت های زیر است:

- 1 - Integration Testing
- 2 - Main Program
- 3 - Regression Testing

- مؤلفه‌های نرم‌افزاری که به کد ترجمه شده‌اند، در قالب یک "بنا" یکپارچه می‌شوند. یک "بنا" شامل تمام فایل‌های داده، کتابخانه‌ها، پیمانه‌های قابل استفاده مجدد و مؤلفه‌های ایجادشده در فرایند مهندسی است که برای پیاده‌سازی یک یا چند تابع محصول مورد نیاز هستند.
- یک‌سری آزمایش طراحی می‌شوند تا خطاهایی را آشکار کنند که باعث می‌شوند یک "بنا" به طور منظم عمل خود را انجام ندهد. هدف از این کار، یافتن خطاهای بازدارنده‌ای است که بالاترین احتمال به تأخیر انداختن پروژه را دارند.
- این "بنا"، با "بناهای" دیگر یکپارچه می‌شود و محصول کامل (به شکل جاری) به صورت مرتب با این روش آزمایش می‌شود. در اینجا روش یکپارچه‌سازی می‌تواند بالا به پایین یا پایین به بالا باشد.

۷-۵-۱۳ صحت و اعتبارسنجی

آزمایش نرم‌افزار یک فعالیت از عنوان گسترده‌تری است که اغلب با **صحت و اعتبارسنجی (V&V)**^۲ شناخته می‌شود. **صحت** اشاره به مجموعه فعالیت‌هایی دارد که از درستی پیاده‌سازی یک تابع خاص از نرم‌افزار اطمینان حاصل شود. **اعتبارسنجی** اشاره به مجموعه‌ای متفاوت دارد که مطمئن می‌کند نرم‌افزار ایجادشده منطبق با نیازهای مشتری است. Boehm این مطلب را این‌گونه بیان می‌کند:

صحت: " آیا محصول را درست ایجاد می‌کنیم؟ "

اعتبارسنجی: " آیا محصول درستی را ایجاد می‌کنیم؟ "

۸-۵-۱۳ آزمایش اعتبارسنجی

در نتیجه آزمایش یکپارچه‌سازی، نرم‌افزار به طور کامل به صورت یک بسته آماده می‌شود، خطاهای واسط‌ها آشکار و برطرف می‌شود و سری نهایی آزمایش‌های نرم‌افزار با عنوان **آزمایش اعتبارسنجی** شروع می‌شود. اعتبارسنجی می‌تواند به چندین روش تعریف شود، اما یک تعریف ساده عبارت است از: " اعتبارسنجی نرم‌افزار وقتی موفق است که عملکرد نرم‌افزار مورد انتظار کاربر باشد." به راستی انتظارات منطقی چیست؟ انتظارات منطقی در مشخصه‌نیازهای نرم‌افزار تعریف شده‌اند. این مشخصه شامل بخشی به نام معیارهای اعتبارسنجی است که شامل اطلاعات مبنایی برای آزمایش اعتبارسنجی است.

• مرور پیکربندی

یک فعالیت مهم فرایند اعتبارسنجی، مرور و بازنگری پیکربندی نرم‌افزار است. ماهیت این مرورها، حصول اطمینان از توسعه مناسب تمام عناصر پیکربندی نرم‌افزار، مستندسازی و ثبت آن‌ها و ارائه جزئیات لازم برای مرحله پشتیبانی در دوره زندگی نرم‌افزار است. مرور پیکربندی گاهی تطبیق نیز نامیده می‌شود.

۹-۵-۱۳ آزمایش‌های پذیرش

هنگامی که نرم‌افزاری برای مشتری ایجاد می‌شود، یک‌سری آزمایش‌های پذیرش طراحی و انجام می‌شوند تا مشتری بتواند صحت تمام نیازهای خود را اعتبارسنجی کند. **آزمایش پذیرش**^۳ به‌جای مهندسی نرم‌افزار، توسط کاربر نهایی انجام می‌شود و می‌تواند شامل هدایت غیر رسمی یا یک‌سری آزمایش‌های برنامه‌ریزی شده و منظم باشد. در واقع، آزمایش پذیرش می‌تواند در بازه زمانی هفته‌ها یا ماه‌ها انجام گیرد و خطاهای موجود که ممکن است کارایی سیستم را در طول زمان کاهش دهند، کشف کند. اگر نرم‌افزار به صورت بسته نرم‌افزاری توسعه داده شود که توسط کاربران متعددی اجرا می‌شود، اجرای آزمایش‌های پذیرش با هر یک، غیر عملی خواهد بود. اکثر سازندگان محصولات نرم‌افزاری از فرایندی به نام **آزمایش آلفا و بتا**^۴ استفاده می‌کنند تا خطاهایی را که به نظر می‌رسد فقط کاربر نهایی می‌تواند بیابد، کشف کنند.

1 - Smoke Test
2 - Validation & Verification
3 - Acceptance Test
4 - Alpha & Beta Testing

آزمایش آلفا در سایت توسعه‌دهنده توسط مشتری انجام می‌شود. نرم‌افزار با تنظیمات معمول مورد استفاده قرار می‌گیرد و توسعه‌دهنده بر آن نظارت دارد و خطاها را ثبت می‌کند؛ بنابراین آزمایش‌های آلفا در محیطی کنترل شده انجام می‌شوند.

آزمایش بتا در یک یا چند سایت مشتری توسط کاربر نهایی نرم‌افزار انجام گیرد. برخلاف آزمایش آلفا، توسعه‌دهنده عموماً در زمان آزمایش حضور ندارد. بنابراین آزمایش بتا، به‌کارگیری زنده نرم‌افزار در محیطی است که توسط توسعه‌دهنده قابل کنترل نیست. مشتری تمام مشکلات را (واقعی یا خیالی) که در طول آزمایش بتا شناسایی می‌شوند ثبت و گزارشات آن را به توسعه‌دهنده در بازه‌های زمانی منظم تحویل می‌دهد.

۱۳-۵-۱۱ آزمایش سیستم

نرم‌افزار فقط یک عنصر از یک سیستم بزرگ رایانه‌ای است و با عناصر دیگر سیستم (مثل، سخت‌افزار، افراد و اطلاعات) یکپارچه می‌شود و یک‌سری آزمایش‌های یکپارچه‌سازی و اعتبارسنجی روی آن انجام می‌گیرد.

۱۳-۵-۱۱ آزمایش احیا

بسیاری از سیستم‌های رایانه‌ای باید بعد از بروز خطا قابل بازیابی باشند و پردازشی را در زمان مشخص شده ادامه دهند. سیستم باید در برابر بروز مشکل مقاوم باشد؛ یعنی خطاهای پردازشی نباید باعث خاتمه کل عملکرد سیستم شود. در موارد دیگر، شکست سیستم باید در بازه زمانی مشخصی اصلاح شود؛ زیرا در غیر این صورت ممکن است ضربه اقتصادی شدیدی به وجود آید. **آزمایش احیا**^۱ نوعی آزمایش سیستم است که باعث شکست نرم‌افزار به روش‌های گوناگون می‌شود و سپس اقدام به بازیابی آن می‌کند به‌گونه‌ای که تعیین شود آیا بازیابی به طور مناسب انجام شده است یا خیر؟

۱۳-۵-۱۲ آزمایش امنیت

هر سیستم رایانه‌ای که اطلاعات حساس را مدیریت می‌کند یا اعمالی انجام می‌دهد که می‌تواند باعث ضرر رساندن (یا فایده رساندن) به افراد شود، هدفی برای نفوذ غیر قانونی یا نامناسب است. نفوذ شامل محدوده وسیعی از فعالیت‌ها است:

- افراد مهاجمی که سعی در نفوذ به سیستم‌ها برای تفریح دارند.
- کارمندان ناراضی که سعی در نفوذ برای انتقام دارند.
- افراد متقلبی که سعی در نفوذ برای رسیدن به اهداف شخصی دارند.

آزمایش امنیت^۲ سعی در بازبینی مکانیزم‌های امنیتی تعبیه‌شده در سیستم را دارد تا از نفوذ غیر قانونی به سیستم محافظت کند.

۱۳-۵-۱۳ آزمایش فشار

در اثنای مراحل اولیه آزمایش نرم‌افزار، روش‌های آزمایش جعبه سفید و جعبه سیاه، عملکردهای معمول و کارایی متداول سیستم را ارزیابی می‌کنند. **آزمایش‌های فشار**^۳ طوری طراحی می‌شوند تا برنامه‌ها را با موقعیت‌های غیر معمول مواجه کنند. در نتیجه آزمایش‌کننده‌ای که آزمایش فشار را انجام می‌دهد این سؤال را می‌پرسد که: **"قبل از شکست تا چه مدت می‌توان نرم‌افزار را در حال کار نگهداشت؟"** آزمایش فشار سیستم را به روشی اجرا می‌کند که منابع با کمیت، تکرار یا حجم غیر معمول درخواست شوند.

۱۳-۵-۱۴ آزمایش کارایی

برای سیستم‌های بلادرنگ و توکار، نرم‌افزاری که عملکرد مورد نظر را اجرا می‌کند، اما منطبق بر معیارهای کارایی نیست، قابل قبول نیست. **آزمایش کارایی**^۴ برای آزمون کارایی زمان اجرای نرم‌افزار صورت می‌گیرد. آزمایش کارایی در طول تمام مراحل فرایند آزمایش انجام می‌شود.

1 - Recovery Testing

2 - Security Testing

3 - Stress Testing

4 - Performance Testing

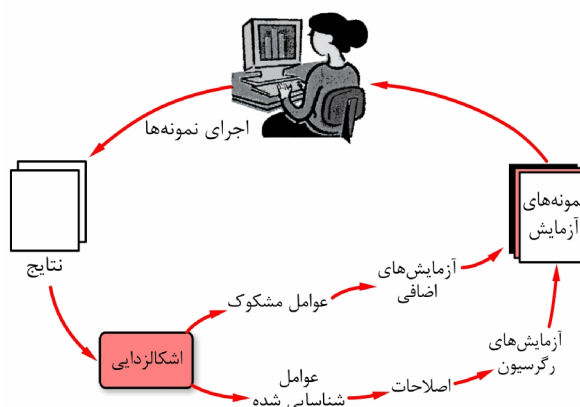
حتی در سطح واحد، کارایی هر پیمانۀ ممکن است با آزمایش‌های جعبه سفید نیز به دست آید. به‌رحال تا زمانی که تمام عناصر سیستم به طور کامل یکپارچه نشده باشند، اندازه‌گیری کارایی کامل سیستم قابل دستیابی نیست.

۱۳-۶ هنر اشکال‌زدایی

اشکال‌زدایی^۱ در نتیجهٔ آزمایش موفق انجام می‌شود؛ یعنی هنگامی که نمونه‌های آزمایش خطایی را کشف می‌کند، اشکال‌زدایی آغاز می‌شود. اشکال‌زدایی فرایندی است که باعث حذف خطا می‌شود. اگرچه اشکال‌زدایی فرایندی پوششی است، ولی تا حد زیادی هنری است. مهندس نرم‌افزاری که نتایج آزمایش را ارزیابی می‌کند، اغلب با علایم یک مشکل نرم‌افزاری مواجه می‌شود. بدین معنی که وقوع خارجی خطا و علت داخلی خطا ممکن است رابطهٔ روشنی با یکدیگر نداشته باشند؛ بنابراین اشکال‌زدایی بیشتر به شانس و هنر آزمایش‌کننده بستگی دارد.

۱۳-۶-۱ فرایند اشکال‌زدایی

چرا اشکال‌زدایی چنین مشکل است؟ با در نظر گرفتن تمام احتمالات، پاسخ به این سؤال، بیشتر به روان‌شناسی انسان مربوط می‌شود تا فن‌آوری نرم‌افزار. فرایند اشکال‌زدایی در شکل زیر نمایش داده شده است.



شکل ۱۳-۵ فرایند اشکال‌زدایی

۱۳-۶-۲ شیوه‌های اشکال‌زدایی

علی‌رغم شیوه‌ای که به کار گرفته می‌شود، اشکال‌زدایی یک هدف پوشش‌دهنده یعنی یافتن و تصحیح علت خطای نرم‌افزار را دارد. این هدف با ترکیب ارزیابی سیستماتیک، ادراک و شانس به دست می‌آید. در حالت کلی، سه دسته‌بندی برای روش‌های اشکال‌زدایی توسط Meyer به شرح زیر پیشنهاد شده است:

• نیروی مطلق

روش نیروی مطلق^۲ اشکال‌زدایی: احتمالاً متداول‌ترین و کم‌بازده‌ترین روش برای شناسایی علت خطای نرم‌افزار است. روش اشکال‌زدایی نیروی مطلق زمانی به کار گرفته می‌شوند که همه روش‌های دیگر با شکست روبه‌رو شده باشند. این کار با فلسفه "یافتن خطا توسط خود رایانه"، صورت می‌گیرد که محتویات حافظه روی صفحه نمایش نشان داده می‌شود. پیگیری‌های زمان اجرا انجام می‌شوند و احکام write در برنامه قرار می‌گیرند. انتظار می‌رود در جایی از اطلاعات تولیدشده، کلیدی پیدا شود که بتواند باعث هدایت به سمت خطا شود. علی‌رغم حجم زیاد اطلاعات تولیدشده ممکن است این روش تا حدی به موفقیت منتهی شود ولی در اکثر موارد باعث اتلاف تلاش و زمان می‌شود.

1 - Debugging
2 - Brute force

- عقب‌گرد

روش عقب‌گرد^۱ اشکال‌زدایی: روشی نسبتاً متداول است که می‌تواند با موفقیت در برنامه‌های کوچک به کار گرفته شود. با شروع از محلی که علامت در آنجا ظاهر شده، برنامه مبدأ به سمت عقب (به صورت دستی) دنبال می‌شود تا زمانی که محل علت بروز خطا پیدا شود. بدبختانه با افزایش تعداد خطوط برنامه، تعداد مسیرهای برگشت بسیار زیاد خواهد بود و کارایی روش پایین می‌آید.

- حذف علت

حذف علت^۲: روش سوم اشکال‌زدایی است که با بسط یا حذف انجام می‌شود و مفهوم تقسیم‌بندی دودویی را همراه دارد. داده‌های مرتبط با خطا سازماندهی می‌شوند تا علل بالقوه را جدا کنند. در یکی از حالات، لیستی از تمام علت‌های ممکن توسعه داده می‌شود و آزمایش‌های مربوط انجام می‌شوند تا هر یک را حذف کند. اگر آزمایش‌های اولیه نشان دهند که یک علت خاص، مربوط به علامت ظاهر شده است، داده‌ها باید پالایش شوند تا خطا از بین برود.

1 - Backtracking

2 - Cause Elimination

فصل چهاردهم

مفاهیم مدیریت پروژه‌های نرم‌افزاری

مدیریت پروژه شامل برنامه‌ریزی، نظارت و کنترل افراد، فرایندها و رویه‌هایی می‌شود که به موازات توسعه و تکامل نرم‌افزار از مفهوم مقدماتی تا پیاده‌سازی عملی رخ می‌دهند.

در مورد روش‌های مدیریت پروژه دیدگاه‌های مختلفی تاکنون عرضه شده است، ولی معتبرترین دیدگاه توسط مؤسسه مدیریت پروژه آمریکا (PMI)^۱ در سال ۱۹۹۹ ارائه شده است. در سال‌های اخیر نسخه‌های جدیدتری از استاندارد مدیریت پروژه توسط این مؤسسه انتشار و توزیع شده است. این مؤسسه کتابی تحت عنوان **پیکره دانش مدیریت پروژه (PMBok)**^۲ برای مدیریت پروژه ارائه داده است. این کتاب استاندارد برای تمام کسانی است که مسئولیت مدیریت پروژه را بر عهده دارند. در این کتاب، مدیریت پروژه شامل نه وظیفه یا حوزه است که در ادامه شرح مختصری ارائه شده است.

۱-۱۴ حوزه‌های پیکره دانش مدیریت پروژه

۱-۱-۱۴ مدیریت حوزه عملکرد^۳

فرایندهای لازم برای حصول اطمینان از اینکه پروژه شامل تمامی فعالیت‌های مورد نیاز بوده و کار اضافی در آن انجام نمی‌شود. فعالیت‌های اصلی این حوزه عبارت‌اند از:

۲-۱-۱۴ مدیریت یکپارچگی^۴

توصیف‌کننده فرایندهای مورد نیاز برای حصول اطمینان از هماهنگی مناسب عناصر مختلف پروژه است. برای این کار حفظ فعالیت‌ها و یکپارچگی افراد و تیم‌های کاری به صورت همسو صورت می‌گیرد تا حاصل کار محصولی یک‌دست و یکپارچه باشد. این حوزه شامل موارد زیر است:

۳-۱-۱۴ مدیریت زمان^۵

فرایندهای مورد نیاز برای حصول اطمینان از اتمام به‌موقع پروژه که شامل فعالیت‌های زیر است:

- 1 - Project Management Institute
- 2 - Project Management Body of Knowledge
- 3 - Scope Management
- 4 - Integration Management
- 5 - Time Management

۴-۱-۱۴ مدیریت هزینه^۱

توصیف کننده فرایندهای مورد نیاز برای حصول اطمینان از اینکه پروژه در چارچوب بودجه مصوب به اتمام می رسد. شامل:

۵-۱-۱۴ مدیریت منابع انسانی^۲

شامل فرایندهایی است که بهترین شکل به کارگیری افراد در پروژه مانند نحوه مدیریت منابع انسانی، سازماندهی، آموزش، ارتباط، تشویق، جذب و ... را تضمین می کند و شامل فعالیت های زیر است:

۶-۱-۱۴ مدیریت ریسک^۳

فرایندهای مورد نیاز برای تعیین، تجزیه و تحلیل ریسک و واکنش مناسب در مقابل ریسک پروژه است. مدیریت ریسک با انجام فعالیت های زیر صورت می گیرد:

۷-۱-۱۴ مدیریت کیفیت^۴

شامل فرایندهای مورد نیاز برای حصول اطمینان از برآورده شدن ضرورت های کیفی تعریف شده توسط پروژه است و شامل موارد زیر است:

۸-۱-۱۴ مدیریت ارتباطات^۵

فرایندهای مورد نیاز برای حصول اطمینان از تولید، جمع آوری، ذخیره، انتشار و توزیع مناسب و به موقع اطلاعات پروژه است. شامل:

۹-۱-۱۴ مدیریت تدارکات^۶

فرایندهای مورد نیاز برای فراهم کردن کالا و خدمات از خارج سازمان های مجری پروژه در راستای تأمین نیازها و پشتیبانی پروژه است و شامل موارد زیر است:

۲-۱۴ طیف مدیریتی

در مدیریت مؤثر پروژه های نرم افزاری بر چهار محور اساسی تأکید شده است: افراد، محصول، فرایند و پروژه.

۱-۲-۱۴ افراد^۷

عامل انسانی چنان اهمیتی دارد که مؤسسه مهندسی نرم افزار، مدل بلوغ قابلیت های مدیریت افراد (PM-CMM)^۸ را توسعه داده است تا آمادگی سازمان های نرم افزاری برای به عهده گرفتن کاربردهای پیچیده را بهبود بخشد. مدل بلوغ قابلیت های مدیریت افراد، زمینه های کلیدی عملی زیر را برای نرم افزارنویسان تعیین می کند: یافتن افراد جدید، انتخاب، مدیریت اجرا، آموزش، جبران خدمت، توسعه حرفه ای، سازماندهی و طراحی کار و توسعه تیمی / فرهنگی.

Weinberger مدل MOI را علاوه بر ویژگی های بالا برای رهبران گروه به شرح زیر پیشنهاد کرده است:

- M = انگیزش^۹
- O = سازماندهی^{۱۰}
- I = نوآوری و خلاقیت^{۱۱}

1 - Cost Management
 2 - Human Resource Management
 3 - Risk Management
 4 - Quality Management
 5 - Communication Management
 6 - Procurement Management
 7 - People
 8 - People Management- Capability Maturity Model
 9 - Motivation
 10 - Organization
 11 - Ideas or Innovation

به هر حال بحث در اصول مدیریت و نحوه سازماندهی و مدیریت تیم‌های نرم‌افزاری گستره وسیعی دارد که در این کتاب به آن‌ها پرداخته نشده است.

• سازماندهی تیم‌های کاری

ساختار بهترین تیم کاری به شیوه مدیریت سازمان، تعداد افراد تشکیل‌دهنده تیم و سطوح مهارتی آنان و همچنین میزان کلی دشواری مسئله بستگی دارد. Manty سه نوع سازماندهی کلی برای یک تیم پیشنهاد می‌کند:

- کنترل نشده غیر متمرکز (DD)^۱
- کنترل شده غیر متمرکز (CD)^۲
- کنترل شده متمرکز (CC)^۳

• عوامل تأثیرگذار بر ساختار و سازماندهی تیم

تیم ویژگی	نوع	ساختار DD	ساختار DC	ساختار CC
۱	دشواری زیاد	✓		
	کم		✓	✓
۲	اندازه برنامه بزرگ		✓	✓
	کوچک	✓		
۳	عمر ماندگاری تیم کوتاه		✓	✓
	درازمدت	✓		
۴	میزان تقسیم پذیری مسئله زیاد		✓	✓
	کم	✓		
۵	کیفیت و قابلیت اطمینان زیاد	✓	✓	
	کم			✓
۶	قطعیت تاریخ تحویل غیر قابل انعطاف		✓	✓
	انعطاف پذیر	✓		
۷	میزان ارتباطات مورد نیاز در پروژه زیاد	✓	✓	
	کم			✓

در انتخاب ساختار تیم از روش‌های زیر نیز می‌توان استفاده کرد:

- تصادفی: تشکیل تیم به صورت تصادفی و آزادانه انجام می‌شود. در این ساختار موفقیت تیم بستگی زیاد به ابتکار فردی اعضای تیم دارد. وقتی تحول و نوآوری مطرح باشد، تیم تصادفی خوب عمل می‌کند. این ساختار عملکرد منظم و مرتبی ندارد.

1 - Democratic Decentralized
 2 - Controled Decentralized
 3 - Controled Centralized

- **بسته:** در این ساختار سلسله مراتب سنتی و قدرتی مانند CC وجود دارد. خلاقیت محدود است و کارهای مشابه فقط انجام می شود.
- **باز:** در این روش سازماندهی تیم پروژه، بعضی از خصوصیات تیم های بسته وجود دارد. بیشتر ابداعات از نمونه تیم های تصادفی تبعیت می کنند. کارهای پروژه به صورت جمعی صورت می گیرد و توافق آرا بین اعضا وجود داشته و تصمیمات به صورت جمعی اتخاذ می شود. این ساختار برای یافتن مشکلات پیچیده بسیار مناسب است.
- **همزمان:** در این ساختار وقتی مشکلی پیش آید تمرکز افراد روی مشکل اعمال می شود. ارتباطات بین اعضا بسیار ضعیف است.

۲-۲-۱۴ محصول^۱

دامنه کاربرد محصول و تجزیه مسئله و همچنین روش های مربوط از طریق ارتباط با مشتری تعیین می شود. انجام برآوردهای منطقی از پروژه، مستلزم یک تحلیل نیازها همراه با جزئیات است که متأسفانه این کار زمان زیادی می برد. بنابراین محصول، دامنه و هدف آن باید بررسی و مبنایی برای برآوردهای پروژه ایجاد شود.

• دامنه کاربرد محصول

نخستین فعالیت مدیریت پروژه نرم افزار، تعیین دامنه کاربرد محصول است. دامنه کاربرد با مشخص کردن مؤلفه های زیر تعیین می شود:

- **محیط:** چگونه نرم افزار ایجاد شده با سیستم های موجود یا محیط حرفه منطبق می شود و محدودیت ها در این محیط چیست؟
- **اهداف اطلاعاتی:** موجودیت ها و اقلام داده ای برای خروجی مطلوب و مورد نظر کدام هستند.
- **عملکرد و کارایی:** نحوه تبدیل داده های ورودی به اطلاعات خروجی و کارایی مورد نظر چیست؟

• تجزیه مسئله

تجزیه مسئله در دو زمینه اصلی صورت می پذیرد:

- قابلیت عملیاتی قابل تحویل
 - فرایندهای مورد نیاز برای تحویل
- تجزیه مسئله کمک بزرگی برای یافتن دامنه و حوزه عملکرد مسئله است. به همین منظور باید از راه های اصولی ارتباط با مشتری استفاده کرد.

۳-۲-۱۴ فرایند^۲

مدیر پروژه باید در مورد انواع مدل های فرایندی توسعه نرم افزار (فصل ۲) تصمیم گیری کند به طوری که مدل فرایندی برای حالت های زیر مناسب باشد:

- مشتری و افرادی که کار را انجام می دهند.
 - ویژگی های محصول
 - محیط پروژه که در آن تیم نرم افزاری کار می کند.
- پس از انتخاب مدل فرایندی توسعه نرم افزار، مدیر پروژه باید برنامه توسعه نرم افزار را تهیه و ارائه کند. به همین منظور وی باید اقدام به "تلفیق فرایند و محصول" و "تجزیه فرایند" اجرای پروژه کند.

۴-۲-۱۴ پروژه^۳

برای مدیریت یک پروژه نرم افزاری موفق، باید بدانیم چه چیزهایی ممکن است اشتباه شود و چگونه می توان آن ها را به راحتی رفع کرد. John Reel علایم در خطر بودن یک پروژه را به شرح زیر اعلام کرده است:

- عدم درک نیازهای مشتری

- عدم تعریف خوب از حوزه محصول
- عدم مدیریت مطلوب تغییرات
- تغییر فن آوری انتخاب شده
- تغییر نیازهای تجاری
- مهلت‌های غیر واقع‌بینانه
- مقاومت کاربران
- از بین رفتن حمایت مالی
- عدم مهارت افراد تیم پروژه
- عدم تلاش کافی برای استفاده از تجارب گذشته

فصل پانزدهم

معیارهای اندازه‌گیری و سنجش نرم‌افزار

اندازه‌گیری پایه نظام مهندسی است و مهندسی نرم‌افزار نیز از این قاعده مستثنی نیست. معیارهای سنجش نرم‌افزار به گستره وسیعی از اندازه‌گیری‌های مربوط به نرم‌افزارهای رایانه‌ای بازمی‌گردد. اندازه‌گیری را در مورد فرایند نرم‌افزار می‌توان

- به قصد بهبود مستمر به کار برد.
- برای کمک به انجام برآورد، کنترل کیفیت، ارزیابی بهره‌وری و کنترل پروژه به کار گرفت.
- و نهایتاً می‌توان برای کمک به ارزیابی کیفیت محصولات فنی و کمک به تصمیم‌گیری‌های تاکتیکی در حین پیشرفت پروژه به کار بست.

در حوزه مدیریت پروژه‌های نرم‌افزاری، ابتدا با معیارهای بهره‌وری و سنجش کیفیت سر و کار داریم که شامل اندازه‌های خروجی فرایند توسعه نرم‌افزار به صورت تابعی از تلاش و زمان به کار رفته است و سپس با اندازه‌گیری‌های مربوط به تطابق با کاربرد محصولات تولیدشده، سر و کار داریم. علاقه ما به برنامه‌ریزی و انجام برآورد، سابقه تاریخی دارد. به عنوان مثال:

- - بهره‌وری پروژه‌های توسعه نرم‌افزاری در گذشته تا چه میزان بوده است؟
- - کیفیت نرم‌افزار تولیدشده چگونه بوده است؟
- - چگونه می‌توان داده‌های مربوط به بهره‌وری و کیفیت پیشین را به زمان حال تغییر داد؟
- - چگونه اطلاعات گذشته می‌تواند به برنامه‌ریزی و انجام برآورد دقیق‌تر به ما کمک کند؟

۱-۱۵ اندازه، معیار سنجش و شاخص

فرهنگ واژه‌های مؤسسه مهندسی نرم افزار IEEE، معیار سنجش را به عنوان **اندازه** مقداری از درجه‌ای که یک سیستم، مؤلفه یا یک فرایند در مورد یک صفت ویژه اعمال می‌کند، تعریف کرده است. مثل تعداد خطاهای حاصل از بازبینی یک پیمانۀ^۱ مجزا. معیار سنجش نرم افزار با تک تک اندازه‌ها در ارتباط است مانند تعداد میانگین خطاها در اثنای بازبینی‌های رسمی. شاخص یک معیار سنجش یا تلفیقی از آن‌هاست که امکان نگرش به درون فرایند نرم افزار، پروژه نرم افزاری و یا خود محصول را فراهم می‌کند. ابزارهای سنجش، بینشی در اختیار مدیر قرار می‌دهد که منجر به تصمیم‌گیری‌های آگاهانه می‌شود. بنابراین:

- **اندازه**^۲: نمایش مقداری از میزان، ابعاد، ظرفیت و ویژگی‌های نرم افزار است.
- **معیار**^۳: اندازه مقداری درجه یک سیستم، مؤلفه یا فرایند است.
- **شاخص**^۴: مجموعه‌ای از معیارهای سنجش برای تصحیح عملکرد پروژه است.

۲-۱۵ معیارهای سنجش فرایند و پروژه

معیارهای سنجش فرایند در طول تمام پروژه‌ها و در طی یک دوره زمانی طولانی گردآوری می‌شوند. هدف از تهیه معیارهای سنجش فرایند فراهم آوردن شاخص‌هایی برای بهبود فرایند نرم افزار در بلندمدت است. شاخص‌های پروژه، یک مدیر پروژه نرم افزاری را قادر می‌کند که:

- ۱- وضعیت یک پروژه در حال جریان را ارزیابی کند.
- ۲- ریسک‌های بالقوه را پیگیری کند.
- ۳- زمینه‌های مشکل را پیش از آنکه «بحرانی» شوند، معلوم کند.
- ۴- جریان کار یا فعالیت‌ها را تعدیل کند.
- ۵- قابلیت گروه پروژه را در مورد کنترل کیفیت محصولات کاری مهندسی نرم افزار مورد ارزیابی قرار دهد.

بسیاری از معیارهای سنجش هم در حوزه پروژه و هم در فرایند نرم افزار به کار می‌روند.

Grady استفاده‌های عمومی و خصوصی برای انواع متفاوت داده‌های فرایند را مطرح می‌کند.

• معیارهای خصوصی

- نرخ خرابی (توسط فرد)
 - نرخ خرابی (توسط پیمانانه)
 - خطاهای پیداشده در طول توسعه نرم افزار
- هدف: یافتن شاخص‌های بهبود و افزایش کارایی فرد است

• معیارهای عمومی

- تعداد نقص‌های گزارش شده در مورد توابع و عملکردهای عمده نرم افزار
 - تعداد خطاهای مشاهده شده در حین بازبینی‌های فنی رسمی
 - تعداد خطوط برنامه (LOC)^۵ یا ارزش تابعی (FP)^۶
- هدف: یافتن شاخص‌های بهبود و افزایش کارایی فرایند سازمانی است.

1 - Module
2 - Measure
3 - Metric
4 - Indicator
5 - Line Of Code
6 - Function Point

۲-۲-۱۵ معیارهای سنجش پروژه

معیارهای سنجش فرایند نرم افزاری برای مقاصد راهبردی به کار می‌روند. اندازه‌های مقداری پروژه نرم افزاری بُعد فنی دارند. یعنی معیارهای سنجش و شاخص‌های پروژه از آن‌ها حاصل شده و از سوی مدیر پروژه و گروه نرم افزاری برای تطبیق جریان کار پروژه و فعالیت‌های فنی مورد استفاده قرار می‌گیرند.

مدل اندازه‌گیری پروژه

مدل دیگری برای اندازه‌گیری پروژه موارد زیر را پیشنهاد می‌کند:

- **ورودی‌ها:** شاخص‌های مقداری منابع (مثلاً افراد و محیط) مورد نیاز برای انجام کار
 - **خروجی‌ها:** شاخص‌های مقداری اهداف واسط یا محصولات کاری تولیدشده در طول فرایند مهندسی نرم افزار
 - **نتایج:** شاخص‌های مقداری که بر ثمربخشی اهداف و محصولات واسط دلالت دارد
- درحقیقت، این مدل را می‌توان هم برای فرایند و هم پروژه به کار برد.

۳-۱۵ اندازه‌گیری نرم افزاری

در دنیای واقعی اندازه‌گیری را می‌توان به دو دسته عمده تقسیم کرد: اندازه‌گیری مستقیم (نظیر طول یک پیچ) و اندازه‌گیری غیر مستقیم (نظیر سنجش «کیفیت» پیچ‌های تولیدشده (از طریق شمارش تعداد پیچ‌های ردشده).

اندازه‌گیری مستقیم فرایندهای مهندسی نرم افزار دربرگیرنده سنجش هزینه و تلاش است. اندازه‌گیری مستقیم نرم افزار شامل تعداد خطوط برنامه (LOC) تولیدشده، سرعت اجرا، میزان حافظه و خرابی‌های گزارش شده در مدت یک دوره زمانی است. سنجش‌های غیر مستقیم نرم افزار شامل قابلیت تابعی بودن^۱، کیفیت، پیچیدگی، کارایی، قابلیت اعتماد، قابلیت نگهداری و «قابلیت»‌های بسیار دیگر است.

۱-۳-۱۵ معیارهای سنجش مبتنی بر اندازه

معیارهای سنجش مبتنی بر اندازه نرم افزار از طریق نرمال‌سازی اندازه‌های مقداری کیفیت یا بهره‌وری با استفاده از «اندازه» نرم افزار تولیدشده حاصل می‌شوند. مانند:

- تعداد افراد مشارکت‌کننده در توسعه نرم افزار (People)
- تعداد صفحات مستندات (pp.Doc)
- میزان هزینه (Cost)
- تعداد خطوط برنامه (LOC)
- تعداد خطاهای پیداشده (Error)
- تعداد خرابی یا شکست‌ها (Defects)

برای به دست آوردن معیارهای سنجش که قابل مقایسه با معیارهای سنجش پروژه دیگر باشد، شاخص تعداد خطوط برنامه (LOC) را به عنوان ارزش هنجارسازی در نظر گرفته‌ایم.

• معیارهای نرمال شده

- تعداد خطاها در هر KLOC (هزار خط برنامه)
- تعداد خرابی یا شکست‌ها در هر KLOC
- هزینه هر LOC
- تعداد صفحات مستندات در هر KLOC

به‌علاوه، شاخص‌های جالب دیگری را نیز می‌توان به صورت زیر محاسبه کرد:

- نسبت خطاها به نفر - ماه

• نسبت LOC به نفر- ماه

• نسبت هزینه به حجم مستندات

البته این روش دارای مجادلات خاص خود است. مخالفین مدعی اند که معیارهای LOC، معیارهایی وابسته به زبان برنامه نویسی هستند، چون زبان‌هایی وجود دارد که به خوبی طراحی شده‌اند و همین امر موجب کوتاه شدن برنامه نویسی شده است و بنابراین نمی‌توان آن‌ها را به سادگی با زبان‌های غیر پردازشی مطابقت داد. درضمن در این محیط‌ها ممکن است سطح جزئیات مورد نیاز برای برآوردها غیر قابل دستیابی باشد (مثلاً، برنامه‌ریز باید LOC را خیلی قبل از تکمیل تحلیل و طراحی برآورد کند).

۱۵-۲-۳ معیارهای سنجش مبتنی بر تابع (عملکرد)

معیارهای سنجش نرم‌افزار مبتنی بر عملکرد از یک معیار اندازه‌گیری تابعی برنامه به عنوان ارزش نرمال شده استفاده می‌کنند. چون تابعی بودن یک نرم‌افزار را نمی‌توان مستقیماً اندازه‌گیری کرد پس باید با استفاده از سایر معیارهای مستقیم، به طور غیر مستقیم به دست آورد. معیارهای سنجش مبتنی بر عملکرد توسط Albrecht با نام **ارزش تابعی** معرفی شده است. ارزش تابعی با استفاده از یک رابطه تجربی مبتنی بر معیارهای قابل اندازه‌گیری (مستقیم) **دامنه اطلاعات نرم‌افزار** و ارزیابی‌های مربوط به پیچیدگی نرم‌افزار حاصل می‌شود. برای این کار پنج ویژگی از دامنه اطلاعاتی به شرح زیر تعیین و شمارش می‌شوند:

تعداد ورودی‌های کاربر: شامل هر ورودی کاربر است که داده‌های مبتنی بر کاربردهای مجزا برای نرم‌افزار فراهم می‌آورند.

تعداد خروجی‌های کاربر: هر خروجی مورد نظر کاربر است و شامل اطلاعات حاصل از برنامه محسوب می‌شود. در این زمینه خروجی‌ها به گزارش‌ها، صفحات نمایش، پیغام‌های خطا و ... اطلاق می‌شود. اقلام داده‌ای مجزا در یک گزارش به طور جداگانه محاسبه نمی‌شوند.

تعداد پرس‌وجو (درخواست)‌های کاربر: یک درخواست به عنوان یک ورودی لحظه‌ای تعریف می‌شود که حاصل آن تولید پاسخ توسط نرم‌افزار به شکل خروجی‌های لحظه‌ای است.

تعداد پرونده‌ها: هر پرونده اصلی منطقی (نظیر گروه‌بندی منطقی داده‌ها به عنوان بخشی از یک پایگاه داده بزرگ یا یک پرونده جداگانه) شمارش می‌شوند.

تعداد واسط‌های خارجی: شمارش تمام واسط‌های قابل خواندن توسط دستگاه (نظیر پرونده‌های داده‌ای روی نوار یا دیسکت) که برای ارسال اطلاعات به یک سیستم دیگر مورد استفاده هستند.

سپس جدول زیر تکمیل و ارزش تابعی از رابطه تجربی ارائه‌شده زیر به دست می‌آید:

ضرایب وزنی پیچیدگی

پارامترهای اندازه‌گیری	تعداد	پیچیده	متوسط ساده	ضرایب	ضرایب
تعداد ورودی‌های کاربر	۳	۴	۶	۳	۱۲
تعداد فر و بی‌های کاربر	۴	۵	۷	۴	۲۸
تعداد درخواست‌های کاربر	۳	۴	۶	۳	۱۲
تعداد فایل‌ها	۴	۱۰	۱۵	۴	۶۰
تعداد واسط‌های خارجی	۵	۷	۱۰	۵	۵۰

شمارش کل

وقتی داده‌های بالا گردآوری شد ارزش پیچیدگی هر یک از آن‌ها تعیین می‌شود. سازمان‌هایی که از روش ارزش تابعی استفاده می‌کنند معیارهایی را برای تعیین ساده، متوسط یا پیچیده بودن داده‌های ورودی، توسعه داده‌اند. برای محاسبه ارزش تابعی از رابطه زیر استفاده می‌شود:

$$FP = \text{شمارش کل} * \left[\frac{0.65}{10} + \frac{0.1}{10} \sum F_i \right]$$

که در آن F_i میزان تعدیل پیچیدگی است و پاسخ ۱۴ سؤال است که به صورت زیر برای سؤالات تعیین می‌شوند:

۰	۱	۲	۳	۴	۵
بی تأثیر	ضعیف	معقول	متوسط	با اهمیت	اساسی

• نرمال سازی ارزش تابعی

وقتی ارزش تابعی محاسبه شد، به طریقی شبیه به روش LOC مقادیر حاصله نرمال سازی می شوند تا برای سنجش بهره وری، کیفیت و سایر مشخصه های نرم افزار به کار روند.

۳-۳-۱۵ معیارهای سنجش ارزش تابعی توسعه یافته

معیارهای سنجش مبتنی بر ارزش تابعی در اصل برای استفاده در برنامه های کاربردی سیستم های اطلاعاتی تجاری طراحی شده اند. به همین دلیل معیارهای سنجش ارزش تابعی برای بسیاری از سیستم ها مهندسی و توکار (که بر عملکرد و کنترل تأکید دارند) نامناسب است. برای بهبود آن، از معیار سنجش ارزش تابعی توسعه یافته با ارزش تابعی سه بعدی استفاده می شود.

ویژگی های هر سه بُعد نرم افزار محاسبه شده و مقداری می شود و سپس تبدیل به مقیاسی می شود که بر تابع گرایی نرم افزار دلالت دارد. بُعد داده ای اکثراً با معیارهای سنجش ارزش تابعی مورد ارزیابی قرار می گیرد. شمارش داده های شناسایی شده (ساختار داده های برنامه نظیر پرونده ها) و داده های خارجی (ورودی ها، خروجی ها، پرس و جوها و ارجعات خارجی) به همراه مقیاس های پیچیدگی برای استخراج شاخص بُعد داده ای مورد استفاده قرار می گیرند.

بُعد تابعی با در نظر گرفتن «تعداد عملیات داخلی مورد نیاز برای تبدیل داده های ورودی به خروجی» اندازه گیری می شود. در محاسبات ارزش تابعی سه بعدی، یک تبدیل به عنوان رشته ای از گام های پردازشی محدود شده توسط دستورات مبنایی در نظر گرفته می شود. بُعد کنترلی با شمارش تعداد تبدیلات بین وضعیت های مختلف اندازه گیری می شود. یک وضعیت نشان دهنده برخی از حالات رفتاری قابل رؤیت خارجی است. یک تبدیل به عنوان نتیجه وقایعی است که منجر به تغییر حالت رفتاری نرم افزار می شود. در حین محاسبه ارزش تابعی سه بعدی (3D)، درجه پیچیدگی به انتقال تخصیص داده نمی شود.

• محاسبه ارزش تابعی سه بعدی

برای محاسبه ارزش تابعی سه بعدی از رابطه زیر استفاده می شود:

$$\text{Index} = I + O + Q + F + E + T + R$$

که در آن I ، O ، Q ، F ، E ، T و R به ترتیب نشان دهنده مقادیر وزنی پیچیدگی یعنی ورودی ها، خروجی ها، درخواست ها، ساختار داده ای داخلی، پرونده های خارجی، تبدیلات و انتقالات است. ارزش وزنی پیچیدگی هر کدام نیز با استفاده از رابطه زیر محاسبه می شود:

$$\text{ارزش وزنی پیچیدگی} = N_{il} W_{il} + N_{ia} W_{ia} + N_{ih} W_{ih}$$

که در آن N_{ij} نشان دهنده تعداد فراوانی رخداد مؤلفه i (نظیر خروجی ها) در هر سطح از پیچیدگی (کم ، متوسط و بالا) و W_{ij} وزن های متناظر مربوط به هر یک از آنهاست.

ارزش تابعی (و بسط آن) نظیر معیارهای LOC همچنان مورد بحث و مجادله است.

۴-۱۵ تلفیق راهکارهای متفاوت سنجش

جدول زیر شامل برآوردهای خام مربوط به متوسط تعداد خطوط برنامه مورد نیاز برای ساخت یک واحد ارزش تابعی توسط زبان‌های برنامه‌نویسی مختلف را نشان می‌دهد.

جدول ۱-۱۵ متوسط ارزش تابعی در محیط‌های مختلف

متوسط LOC/FP	زبان برنامه‌نویسی
۳۲۰	زبان اسمبلی
۱۲۸	زبان C
۱۰۵	زبان کوبول
۱۰۵	زبان فرترن
۹۰	زبان پاسکال
۶۴	زبان C++
۵۳	Ada 95
۳۰	زبان‌های شیء‌گرا مانند Visual Basic
۲۰	زبان‌های نسل چهارم مانند Smaltalk
۱۶	مولد کد
۱۲	SQL
۶	صفحه‌گسترها
۴	زبان‌های گرافیکی

مروری بر این داده‌ها نشان می‌دهد که قابلیت تابعی حاصل از یک LOC زبان C++ به طور تقریبی ۱/۶ برابر LOC زبان فرترن است. همچنین، یک LOC زبان‌های نسل چهارم سه تا پنج برابر زبان‌های برنامه‌نویسی قراردادی، قابلیت تابعی ایجاد می‌کنند.

۱-۴-۱۵ اندازه‌گیری کیفیت

اگرچه معیارهای بسیاری در مورد کیفیت، مانند صحت (درستی عملکرد)، قابلیت نگهداری، یکپارچگی و قابلیت استفاده مجدد نرم‌افزار وجود دارند که شاخص‌های مفیدی را برای گروه پروژه فراهم می‌آورند، با این وجود Gilb تعاریف و معیارهایی را برای هر یک از آنها پیشنهاد می‌کند.

صحت (درستی عملکرد): یک برنامه باید به طور درستی عمل کند وگرنه برای کاربر ارزشی نخواهد داشت. **درستی عملکرد^۱** درجه‌ای است که نشان‌دهنده صحت عملکرد نرم‌افزار است. از معمول‌ترین معیارهای درستی عملکرد می‌توان به **تعداد خرابی در هر KLOC** اشاره کرد که در آن خرابی (شکست) به عنوان عدم تطابق مشاهده‌شده با نیازها تعریف می‌شود.

قابلیت نگهداری: قابلیت نگهداری امکانی است که از طریق آن در صورت بروز **خطا** می‌توان یک برنامه را اصلاح کرد و یا در صورت تغییر محیط آن را با محیط تطبیق داد، یا در صورتی که مشتری خواهان **تغییری در نیازهای نرم‌افزار** باشد امکاناتی به آن اضافه کرد. هیچ راهی برای اندازه‌گیری مستقیم قابلیت نگهداری وجود ندارد، بنابراین باید از معیارهای غیر مستقیم استفاده کرد. معیار ساده مبتنی بر زمان **میانگین زمان مورد نیاز برای تغییر (MTTC)**^۲ است که مدت‌زمان لازم برای تحلیل درخواست تغییر، آزمایش و توزیع نسخه تغییر یافته بین تمام کاربران را نشان می‌دهد. به طور متوسط برنامه‌هایی قابل نگهداری هستند که دارای **MTTC پایین‌تری** (برای انواع تغییرات یکسان) هستند.

1 - Correctness

2 - Mean Time To Change

یکپارچگی: یکپارچگی نرم افزار در عصر سرقتها و ویروس های رایانه ای به طور فزاینده ای اهمیت یافته است. این مشخصه، قابلیت یک سیستم را در مقابله با دستکاری های تصادفی یا عمدی در راستای ایمنی آن اندازه گیری می کند. این دستکاری ها می توانند در مورد سه جزء نرم افزار (برنامه، داده و مستندات) انجام گیرند.

برای سنجش یکپارچگی نرم افزار دو مشخصه تهدید و ایمنی به صورت زیر تعریف می شود:

- تهدید احتمال رخداد یک دستکاری از نوع خاص در یک زمان مشخص است (این مشخصه را می توان از شواهد تجربی برآورد کرد).
- ایمنی احتمال دفع یک دستکاری خاص است (این مشخصه را نیز می توان از شواهد تجربی برآورد کرد).

در این صورت یکپارچگی سیستم به صورت زیر تعریف می شود:

$$[\text{ایمنی} - (1 - \text{تهدید})] = \text{یکپارچگی}$$

که در آن تهدید و ایمنی برای هر نوع دستکاری محاسبه می شوند.

قابلیت استفاده: بحث سازگاری کاربر با محصولات نرم افزاری و به عبارت دیگر کاربرپسندی^۱ نرم افزار جایگاه خاص خود را پیدا کرده است. اگر برنامه ای کاربرپسند نباشد علی رغم دارا بودن توابع و عملکردهای با ارزش، اغلب با شکست مواجه می شود. قابلیت استفاده تلاشی برای مقداری کردن کاربرپسندی نرم افزار است که از طریق چهار ویژگی زیر اندازه گیری می شود:

- ۱- مهارت فیزیکی یا ذهنی مورد نیاز برای یادگیری نرم افزار
 - ۲- مدت زمان مورد نیاز برای کسب کارایی بالا در استفاده از سیستم
 - ۳- خالص افزایش بهره وری در مقایسه با سیستم قبلی
 - ۴- ارزیابی شهودی کاربر از نرم افزار (گاهی این معیار از طریق پرسشنامه و نظرسنجی از کاربران نسبت به سیستم به دست می آید).
- چهار معیاری که در بالا مطرح شدند تنها نمونه هایی از عواملی هستند که به عنوان معیارهای کیفیت نرم افزار پیشنهاد شده اند.

۵-۱۵ بازدهی رفع نقص (شکست)

یکی دیگر از معیارهای کیفی که در هر دو سطح پروژه و فرایند سودمند است، کارایی یا بازدهی رفع نقص (DRE) است و به صورت زیر تعریف می شود:

$$DRE = E / (E + D)$$

که در آن:

E: تعداد خطاهای پیداشده قبل از تحویل نرم افزار به کاربر نهایی

D: تعداد شکست هایی که پس از تحویل نرم افزار به مشتری مشخص می شوند.

مقدار ایده آل DRE برابر با ۱ است که نشان دهنده عدم وجود خرابی در نرم افزار است.

اگر شاخص DRE به عنوان معیاری برای قابلیت فیلترکردن فعالیت های تضمین و کنترل کیفیت به کار رود، می تواند یک گروه پروژه نرم افزاری را به برقراری روش هایی برای یافتن خطاهای هرچه بیشتر قبل از تحویل نرم افزار به مشتری تشویق کند. همچنین DRE را می توان در یک پروژه برای ارزیابی قابلیت خطایابی فعالیت های پشتیبانی یا سایر فعالیت های مهندسی نرم افزار نیز به کار برد. در این حالت داریم:

$$DRE_i = E_i / (E_i + E_{i+1})$$

که در آن:

E_i : تعداد خطاهایی است که در حین فعالیت i ام مهندسی نرم افزار پیدا شده اند.

E_{i+1} : تعداد خطاهایی است که در طی فعالیت $i+1$ ام مهندسی نرم افزار پیدا شده اند و در فعالیت i مشخص نشده اند.

هدف کیفی برای یک گروه نرم افزاری (یا یک مهندس نرم افزار) دستیابی به DRE نزدیک به ۱ است.

۱۵-۶ تعیین معیارها و برآورد پروژه‌های مبتنی بر شیء‌گرایی

روش‌های سنتی برآورد پروژه نرم‌افزاری، نیازمند برآورد کردن تعداد خطوط کد (LOC) یا ارزش تابعی (FP) به عنوان مبنای برآورد هستند. از آنجاکه یکی از اهداف پروژه‌های مبتنی بر شیء‌گرایی استفاده مجدد است در اینجا برآوردهای مبتنی بر LOC چندان معنی پیدا نمی‌کند. برآوردهای FP را می‌توان به طور کارآمد استفاده کرد، زیرا شمارش دامنه اطلاعاتی مورد نیاز به راحتی از روی بیان مسئله به دست می‌آید.

۱۵-۶-۱ مجموعه پیشنهادی از معیارهای پروژه

تعداد متون سناریو: یک متن سناریو، مرحله‌ای است که تعامل میان کاربر و برنامه کاربردی را توصیف می‌کند. هر متن به شکل سه‌تایی زیر سازماندهی می‌شود:

{آغازگر، عمل، شرکت کننده}

که در آن **آغازگر** یک شیء است که سرویسی درخواست می‌کند، **عمل** نتیجه درخواست و **شرکت کننده**، شیء سرویس‌دهنده‌ای است که درخواست را برآورده می‌کند.

تعداد متون سناریو مستقیماً با اندازه برنامه کاربردی و با تعداد نمونه‌های آزمونی که باید پس از ساخته شدن سیستم روی آن آزمایش شوند، ارتباط دارد.

تعداد کلاس‌های کلیدی: کلاس‌های کلیدی «مؤلفه‌های بسیار مستقل» هستند که در مرحله تحلیل شیء‌گرا تعریف می‌شوند. از آنجاکه کلاس‌های کلیدی، در دامنه مسئله اهمیت اساسی دارند، تعداد این‌گونه کلاس‌ها شاخصی از مقدار کار لازم برای توسعه نرم‌افزار و نیز شاخصی از مقدار بالقوه استفاده مجدد در اثنای توسعه سیستم است.

تعداد کلاس‌های پشتیبان: کلاس‌های پشتیبان برای پیاده‌سازی سیستم لازم‌اند، ولی ارتباط مستقیم با دامنه مسئله ندارند. به عنوان مثال کلاس‌های واسط گرافیکی کاربر (GUI)، کلاس‌های دستیابی به بانک‌های اطلاعاتی و کلاس‌های محاسباتی از این نوع هستند.

تعداد میانگین کلاس‌های پشتیبان به ازای هر کلاس کلیدی: اگر تعداد میانگین کلاس‌های پشتیبان به ازای هر کلاس برای یک مسئله مشخص باشد، برآورد بسیار ساده می‌شود. پیشنهاد می‌شود که در برنامه کاربردی با GUI، تعداد کلاس‌های پشتیبان دو تا سه برابر کلاس‌های کلیدی و در برنامه‌های فاقد GUI یک تا دو برابر کلاس‌های کلیدی باشد.

تعداد زیرسیستم‌ها: یک زیرسیستم، مجموعه‌ای از کلاس‌های پشتیبان عملکردی سیستم است که برای کاربر نهایی سیستم قابل مشاهده است. هنگامی که زیرسیستم‌ها شناسایی شدند، تنظیم یک زمان‌بندی منطقی، آسان‌تر می‌شود.

فصل شانزدهم

برنامه ریزی پروژه‌های نرم‌افزاری

فرایند مدیریت پروژه‌های نرم‌افزاری شامل مجموعه‌ای از فعالیت‌هاست که اولین فعالیت آن **برنامه‌ریزی پروژه** نامیده می‌شود. قبل از آغاز پروژه، مدیر و تیم نرم‌افزاری باید ضمن تعریف فعالیت‌های پروژه منابع مورد نیاز، زمان و هزینه پروژه را برآورد کنند. هدف از برنامه‌ریزی پروژه، تهیه و تدوین چارچوبی برای مدیریت پروژه در جهت برآورد معقول و منطقی منابع، هزینه و زمان است که در مدت اجرای پروژه به‌هنگام می‌شوند.

۱-۱۶ برآوردها

برآورد منابع، هزینه و زمان‌بندی برای کارهای مهندسی نرم‌افزار نیاز به داشتن تجربه، دستیابی به اطلاعات تاریخی خوب و جرأت انجام پیش‌بینی‌های کمی و مقداری به هنگام وجود اطلاعات صرفاً کیفی است. برآورد، به طور ذاتی با ریسک همراه است و همین ریسک منجر به عدم قطعیت در فرایند مهندسی نرم‌افزار می‌شود.

بنابراین برآورد منبع، هزینه و زمان‌بندی توسعه نرم‌افزار نیاز به دانستن عوامل تأثیرگذار بر آن‌ها به شرح زیر دارد:

- تجربه
- دستیابی به اطلاعات سابقه‌ای و مشابه
- داشتن معیارهای اندازه‌گیری و
- عدم قطعیت‌ها

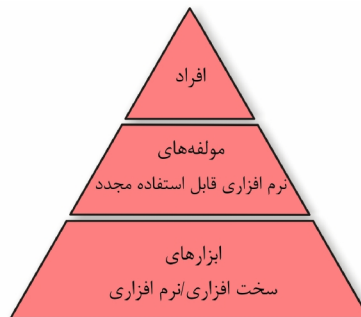
۱-۱۶ عوامل تأثیرگذار بر برآوردها

- پیچیدگی پروژه
- اندازه پروژه
- درجه یا میزان عدم قطعیت ساختاری

۲-۱۶ منابع

دومین کار برنامه‌ریزی پروژه نرم‌افزاری تخمین و برآورد منابع لازم برای انجام فعالیت توسعه نرم‌افزار است. شکل زیر منابع مورد نیاز برای توسعه نرم‌افزار را به صورت هرم نشان می‌دهد. محیط توسعه، شامل ابزارهای نرم‌افزاری و سخت‌افزاری، در بخش بنیادی هرم منابع قرار

می‌گیرد و فراهم‌کننده زیربنایی برای حمایت از تلاش‌های توسعه نرم‌افزار است. در سطح بالاتر، اجزای نرم‌افزاری قابل استفاده مجدد در نظر گرفته می‌شوند که عبارت‌اند از اجزای تشکیل‌دهنده نرم‌افزار که می‌توانند هزینه‌های توسعه را کاهش داده و تحویل نرم‌افزار را شتاب بخشند. در بالای این هرم، منبع افراد قرار دارد. هر منبع با چهار ویژگی مشخص می‌شود: توصیف منبع، قابلیت دسترسی به منبع، زمان مورد نیاز و مدت‌زمان به‌کارگیری منبع.



شکل ۱۶-۰. منابع پروژه

۱۶-۲-۱ منابع انسانی

تعداد افراد لازم برای پروژه نرم‌افزاری می‌تواند فقط بعد از برآورد فعالیت توسعه (برای مثال، نفر-ماه) و تعیین مواردی مانند تعداد افراد و مهارت‌های مورد نیاز شامل شرح مهارت منبع، وضعیت دسترسی، توالی زمانی مورد استفاده از منبع و مدت‌زمان استفاده از آن انجام شود.

۱۶-۲-۲ منابع نرم‌افزاری قابل استفاده مجدد

مهندسی نرم‌افزار مبتنی بر مؤلفه‌ها (CBSE)^۱ بر قابلیت اطمینان تأکید دارد، چنین اجزای تشکیل‌دهنده‌ای، اغلب مؤلفه نامیده می‌شود و باید برای ارجاع آسان به آن‌ها طبقه‌بندی شوند، برای به‌کارگیری آسان باید استاندارد شوند و برای ترکیب ساده باید اعتبارسنجی شوند بنابراین:

- مؤلفه‌های آماده اعتبارسنجی شده
- مؤلفه‌های کاملاً آزمایش‌شده
- مؤلفه‌های آزمایش‌نشده
- مؤلفه‌های جدید

۱۶-۲-۳ منابع محیطی

محیطی که پروژه نرم‌افزار را حمایت می‌کند، اغلب محیط مهندسی نرم‌افزار (SEE)^۲ نامیده می‌شود که نرم‌افزار و سخت‌افزار را به کار می‌گیرد. سخت‌افزار، مبنایی را ایجاد می‌کند که ابزارهای لازم برای تولید محصولات کاری مناسب مهندسی نرم‌افزار را پشتیبانی می‌کند. چون اکثر سازمان‌های نرم‌افزاری دارای چندین گروه تخصصی هستند که نیازمند دسترسی به محیط مهندسی نرم‌افزار (SEE) هستند، برنامه‌ریز پروژه باید دوره زمانی لازم برای استفاده از سخت‌افزار و نرم‌افزار را مشخص کرده و تعیین کند که آیا این منابع در آن دوره‌ها در دسترس هستند یا خیر؟ بنابراین منابع محیطی شامل نرم‌افزار، سخت‌افزار و ارتباطات داده‌ای است.

۱۶-۳ برآورد پروژه نرم‌افزاری

برآورد پروژه نرم‌افزاری می‌تواند از یک هنر ناشناخته به یک سری مراحل سیستماتیک تبدیل شود که تخمین‌هایی را همراه با ریسک قابل قبول فراهم کند. روش‌های برآورد عبارت‌اند از:

- ۱- تأخیر در برآورد حتی تا اواخر پروژه
- ۲- استفاده از بهترین برآوردهای انجام‌شده روی پروژه‌هایی که قبلاً کامل شده‌اند (مشابه‌سازی)

1 - Component Base Software Engineering
2 - Software Engineering Environment

۳- استفاده از روش‌های تجزیه نسبتاً ساده برای برآوردهای هزینه‌ای، زمانی و تلاش پروژه

۴- استفاده از مدل‌های تجربی برآورد هزینه و تلاش مورد نیاز

مدل‌های تجربی را می‌توان برای تکمیل روش‌های تجزیه پروژه به کار برد و یک روش ارزشمند بالقوه ارائه داد. ابزارهای برآورد خودکار نیز یک یا چند روش تجزیه و مدل تجربی را پیاده‌سازی می‌کنند.

روش اول عملی نیست، روش دوم در صورت وجود پروژه مشابه قابل قبول است. حالت‌های باقی‌مانده، روش‌های عملی برای برآورد پروژه نرم‌افزار هستند.

روش‌های تجزیه از شیوه تقسیم و تجزیه به اجزای کوچک برای برآورد پروژه‌های نرم‌افزاری استفاده می‌کنند. با تجزیه پروژه به توابع و فعالیت‌های عمده مهندسی نرم‌افزار، برآورد هزینه و تلاش می‌تواند به روش مرحله‌ای انجام شود.

۱۶-۴ روش‌های تجزیه برآورد پروژه نرم‌افزار

روش تجزیه از دو دیدگاه متفاوت تجزیه مسئله و تجزیه فرایند مد نظر هستند. در برآورد پروژه‌های نرم‌افزاری یکی یا هر دو، مورد استفاده قرار می‌گیرند.

۱۶-۴-۱ تعیین اندازه نرم‌افزار

دقت برآورد پروژه نرم‌افزار به چند عامل بستگی دارد: (۱) درجه‌ای که برنامه‌ریز اندازه محصول را تخمین زده است، (۲) توانایی ترجمه برآورد اندازه نرم‌افزار به فعالیت انسانی، تقویم زمانی و هزینه (تابعی از در دسترس بودن یا قابل اطمینان بودن معیارهای نرم‌افزار از پروژه‌های قبلی)، (۳) درجه‌ای که طرح پروژه توانایی تیم نرم‌افزار را نشان می‌دهد، (۴) پایداری نیازهای پروژه و محیطی که فعالیت مهندسی نرم‌افزار را حمایت می‌کند.

چون برآورد پروژه منوط به برآورد اندازه نرم‌افزار است بنابراین اولین کوشش عمده برنامه‌ریز پروژه است. در برنامه‌ریزی پروژه، منظور از اندازه به نتیجه کمی‌پذیر از پروژه نرم‌افزاری اشاره دارد. اگر روشی مستقیم به کار گرفته شود، اندازه می‌تواند برحسب LOC اندازه‌گیری شود. اگر روش غیر مستقیم به کار گرفته شود، اندازه برحسب FP نشان داده می‌شود.

۱۶-۴-۲ برآورد مبتنی بر مسئله

داده‌های LOC و FP به دو روش در ضمن برآورد پروژه نرم‌افزار استفاده می‌شوند: (۱) به عنوان متغیر تخمین برای تعیین اندازه هر عنصر نرم‌افزاری و (۲) به عنوان معیارهای بنیادی جمع‌آوری شده از پروژه‌های قبلی و استفاده به همراه متغیرهای تخمینی از هزینه و کار.

برآورد LOC و FP روش‌های برآورد متمایزی هستند. با این وجود هر دو ویژگی‌های مشترکی دارند. برنامه‌ریز پروژه اقدام به تجزیه نرم‌افزار می‌کند که هر یک می‌توانند به صورت مجزا برآورد شوند. سپس LOC یا FP (متغیر برآوردی) برای هر تابع تخمین زده می‌شود.

برنامه‌ریز پروژه با برآورد محدوده مقادیری برای هر تابع یا مقداری در محدوده اطلاعات، کار را شروع می‌کند. با استفاده از داده‌های سابقه‌ای یا (در صورت شکست موارد دیگر) با استفاده از ادراک، برنامه‌ریز مقدار اندازه خوشبینانه، محتمل و بدبینانه را برای هر تابع یا هر مقدار موجود در محدوده اطلاعات تخمین می‌زند. سپس یک مقدار مورد انتظار محاسبه می‌شود. مقدار مورد انتظار برای متغیر برآورد (اندازه)، S ، به صورت متوسط وزنی مقدار خوشبینانه (S_{opt})^۱ و محتمل (S_m)^۲ و بدبینانه (S_{pess})^۳ برآوردها به صورت زیر محاسبه می‌شود:

$$S = (S_{opt} + 4S_m + S_{pess}) / 6$$

۱۶-۴-۳ برآورد مبتنی بر فرایند

متداول‌ترین روش برآورد پروژه، استفاده از مبنای فرایندهاست؛ یعنی هر فرایند به مجموعه‌ای نسبتاً کوچک از کارها تقسیم می‌شود و تلاش لازم برای انجام هر کار برآورد می‌شود.

۵-۱۶ مدل های تجربی برآورد هزینه و تلاش نرم افزار

یک مدل برآورد تلاش نرم افزار، از مشاهدات به دست آمده از فرمول های تجربی برای پیش بینی کار و تلاش به صورت تابعی از LOC یا FP تعریف شده است.

۱-۵-۱۶ ساختار مدل های برآورد

$$E = A + B \cdot (ev)^c$$

که در آن A، B و C ثابت های تجربی، E میزان تلاش برحسب نفر-ماه و ev متغیر برآوردی از LOC یا FP یا EFP است.

• مدل های پیشنهادی بر مبنای LOC

$$E = 5/2 * (KLOC)^{0.91} \quad , \quad \text{Walston- Felix}$$

$$E = 3/2 * (KLOC)^{1.05} \quad , \quad \text{Boehm Simple}$$

$$E = 5/5 + 0.73 * (KLOC)^{1.16} \quad , \quad \text{Baily- Basili}$$

• مدل های پیشنهادی بر مبنای FP

$$E = -13/29 + 0.545 * FP \quad , \quad \text{Albrecht-Gaffney}$$

$$E = 585/7 + 15/12 * FP \quad , \quad \text{Matson-Barnett-Mellichamp}$$

۲-۵-۱۶ مدل های COCOMO¹

Boehm سه رده از مدل های COCOMO را تعریف می کند:

- مدل های مقدماتی
- مدل های میانی
- مدل های پیشرفته

هر یک از مدل ها در حالت های اولیه، نیمه جدا و توکار برای پارامترهای A، B و C برآورد شده است.

ساده ترین مدل Boehm در حالت اولیه به صورت زیر ارائه شده است:

$$PM = 2/4 (KIOC)^{1.05} \quad (\text{نفر - ماه})$$

$$TD_{ev} = 2/5 (PM)^{0.28} \quad (\text{مدت زمان برحسب ماه})$$

۳-۵-۱۶ مدل های COCOMOII

این مدل شامل سلسله مراتبی از مدل هاست که زمینه های زیر را مورد توجه قرار می دهد:

مدل ترکیب کاربرد در مراحل اولیه مهندسی نرم افزار استفاده می شود، هنگامی که نمونه سازی واسط های کاربر، توجه به ارتباط نرم افزار و سیستم، تشخیص کارایی و ارزیابی تکامل تکنولوژی، اهمیت دارند.

مدل مرحله اولیه طراحی زمانی استفاده می شود که نیازها تثبیت شده و ساختار پایه معماری ایجاد شده است.

مدل مرحله بعد از معماری در ایجاد نرم افزار استفاده می شود.

۴-۵-۱۶ معادله نرم افزار

معادله نرم افزار مدلی پویا و چندمتغیره است که فرض می کند توزیع خاصی از فعالیت ها در دوره زندگی یک پروژه توسعه نرم افزار وجود دارد. این مدل از داده های قابلیت تولیدی که برای ۴۰۰۰ پروژه نرم افزاری جمع آوری شده اند به دست آمده است. بر مبنای این داده ها، مدل برآورد به صورت زیر ارائه شده است:

یا

$$E = \left[LOC * B^{0.333} / P \right]^3 * \left(\frac{1}{t^4} \right)$$

$$E = L^3 / (P^3 t^4)$$

که در آن

E: تلاش لازم برای توسعه نرم افزار برحسب نفر- ماه یا سال

t: مدت زمان پروژه برحسب ماه

B: ضریب مهارتی ویژه (بین ۰ و ۱)

P: پارامتر قابلیت تولید (بهره‌وری) که مقادیری بیش از ۲۰۰۰ دارد و شامل:

- تکامل کلی پروژه و تجارب مدیریتی
- نهایت به کارگیری تجارب مهندسی نرم افزار خوب
- سطح زبان برنامه‌نویسی استفاده شده
- حالت محیط نرم افزار
- مهارت‌ها و تجربیات تیم نرم افزار
- پیچیدگی کاربرد

مقادیر معمول می‌تواند $P = 2000$ برای توسعه نرم افزار بلادرنگ توکار، $P = 10000$ برای نرم افزار سیستم‌های مخابراتی، $P = 28000$ برای کاربردهای سیستم‌های تجاری باشد. پارامتر قابلیت تولید می‌تواند برای شرایط فعلی با استفاده از داده‌های سابقه‌ای جمع‌آوری شده نیز به دست آید.

توجه به این نکته مهم است که معادله نرم افزار دو پارامتر مستقل دارد:

۱ برآوردی از اندازه (LOC)

۲ شاخصی از مدت پروژه برحسب ماه یا سال

Putnam و Meyers مدل‌های ساده‌تری را برای برآورد به صورت زیر ارائه کرده‌اند:

$$t_{min} = 8/14 * (LOC/p)^{0.43} \quad , \quad \text{if } t \geq 6$$

$$E = 18 * B t^3 \quad , \quad \text{if } E \geq 20$$

که در آن t مدت زمان برحسب ماه و E میزان تلاش برحسب نفر - ماه تعریف شده است.

۶-۱۶ رهیافت پیشنهادی برآورد و زمان بندی برای پروژه‌های شیء گرا

- ۱ ایجاد برآوردهایی با استفاده از تجزیه کار، تحلیل FP و هر روش دیگری که برای کاربردهای سنتی قابل اجرا باشد.
- ۲ با استفاده از مدل‌های تحلیل شیء گرا، متون سناریو را تهیه کرده و یک شمارش از تعداد آن‌ها به دست آورید. تعداد متون سناریو ممکن است با پیشرفت پروژه تغییر کند.
- ۳ با استفاده از مدل‌های تحلیل شیء گرا تعداد کلاس‌های کلیدی را تعیین کنید.
- ۴ نوع واسط را برای برنامه کاربردی دسته بندی کرده و برای کلاس‌های پشتیبان ضریبی ایجاد کنید.

ضریب	نوع واسط
۲/۰	بدون GUI
۲/۲۵	واسط کاربر متنی
۲/۵	GUI
۳/۰	GUI پیچیده

- ۵ تعداد کل کلاس‌ها (کلیدی + پشتیبان) را در تعداد میانگین واحدهای کاری مورد نیاز ضرب کنید. Kidd و Lorenz به ازای هر کلاس ۱۵ تا ۲۰ نفر - روز کار پیشنهاد می‌کنند.
- ۶ برآورد مبتنی بر کلاس‌ها را با ضرب تعداد میانگین واحدهای کار به ازای هر متن سناریو، چک کنید.

۱۶-۲ تصمیم‌گیری ساخت یا خرید

مدیران در پروژه‌های توسعه نرم‌افزار همواره با تعدادی گزینه به شرح زیر مواجه هستند:

- ۱ نرم‌افزار می‌تواند به صورت آماده خریداری شود.
 - ۲ مؤلفه‌های نرم‌افزاری اعتبارسنجی شده یا تجربه‌شده را خرید و سپس اصلاح و یکپارچه کرد تا نیازهای خاص مرتفع شود.
 - ۳ نرم‌افزار به روش مرسوم توسط پیمانکار خارجی توسعه یابد تا مشخصه‌های سفارش را تأمین کند.
 - ۴ نرم‌افزار مورد نظر را خریداری و تغییرات مورد نیاز را در آن اعمال کرد.
- مراحل خرید نرم‌افزار با توجه به اهمیت نرم‌افزار و هزینه نهایی تعیین می‌شود. در برخی موارد، خرید و آزمایش نرم‌افزار، هزینه بسیار کمتری نسبت به هدایت طولانی مراحل تولید بسته‌های نرم‌افزاری دارد.
- برای محصولات نرم‌افزاری پرهزینه، در تحلیل نهایی، تصمیم برای ایجاد یا خرید بر مبنای شرایط زیر صورت می‌گیرد: (۱) آیا تاریخ خرید و تحویل محصول نرم‌افزاری زودتر از توسعه نرم‌افزار به صورت داخلی خواهد بود؟ (۲) آیا هزینه خرید و تطبیق با نیازها کمتر از هزینه توسعه نرم‌افزار به صورت داخلی است؟ (۳) آیا هزینه پشتیبانی خارجی (برای مثال، قرارداد پشتیبانی) کمتر از هزینه پشتیبانی داخلی است؟ این شرایط برای هر یک از حالت‌های توسعه باید در نظر گرفته شوند.

۱۶-۲-۱ ایجاد درخت تصمیم‌گیری

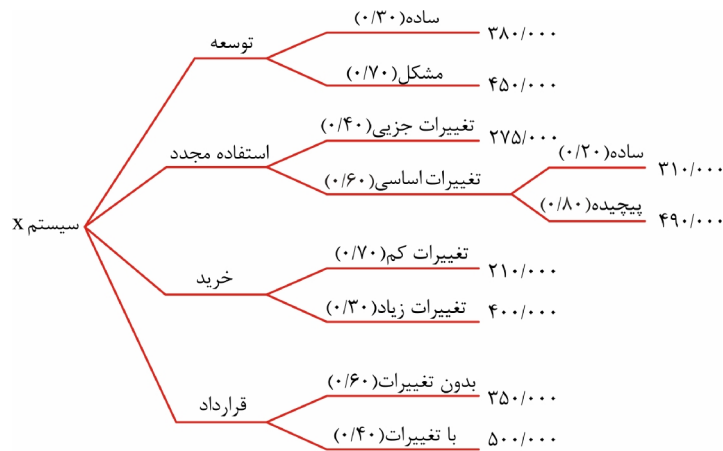
مراحل مطرح‌شده در بالا را می‌توان با استفاده از روش‌های آماری مانند تحلیل درخت تصمیم گسترش داد. برای مثال یک درخت تصمیم را برای تولید یک سیستم نرم‌افزاری به صورت زیر در نظر بگیرید.

(۱) سیستم x را ایجاد کند، (۲) مؤلفه‌هایی با آزمایشات کمتر را استفاده کند تا سیستم را ایجاد کند، (۳) محصول نرم‌افزاری قابل دسترسی را خریداری و اصلاح کرده تا نیازهای مورد نظر را مرتفع کند یا (۴) با یک توسعه‌دهنده نرم‌افزار (پیمانکار)، قراردادی برای توسعه نرم‌افزار منعقد کند.

اگر سیستم از ابتدا ایجاد شود، ۷۰ درصد احتمال دارد که این کار مشکل باشد. با استفاده از روش‌های برآورد بحث‌شده در این فصل، برنامه‌ریز پروژه برآورد می‌کند که فعالیت توسعه به صورت دشوار، هزینه‌ای برابر ۴۵۰۰۰۰ واحد پولی خواهد داشت. برای یک فعالیت توسعه ساده ۳۸۰۰۰۰ واحد پولی هزینه برآورد شده است؛ بنابراین هزینه مورد انتظار هر شاخه از درخت تصمیم عبارت است از:

$$i \text{ (هزینه مسیر برآورد)} \times i \text{ (احتمال مسیر)} = \sum_i \text{ هزینه مورد انتظار}$$

که i مسیری در درخت تصمیم است.



شکل ۱۶-۲ درخت تصمیم برای توسعه نرم افزار

بنابراین داریم:

$$\begin{aligned}
 (1) \quad & \text{هزینه مورد انتظار برای توسعه} = 0/30(380) + 0/70(450) = 429 \\
 (2) \quad & \text{هزینه مورد انتظار برای استفاده مجدد} = 0/40(275) + 0/60[0/20(310) + 0/80(490)] = 320 \\
 (3) \quad & \text{هزینه مورد انتظار برای خرید} = 0/70(210) + 0/30(400) = 267 \\
 (4) \quad & \text{هزینه مورد انتظار برای قرارداد} = 0/60(350) + 0/40(500) = 410
 \end{aligned}$$

بنابراین کمترین هزینه مورد انتظار، حالت "خرید" است.

۸-۱۶ پیمانکاری

استفاده از منابع خارجی از نظر مفهومی، بسیار ساده است. برای انجام فعالیت‌های مهندسی نرم افزار با مجموعه دیگری که این کار را با هزینه و زمان کمتر و کیفیت بالاتر انجام می دهد قرارداد بسته می شود. در این حالت مدیریت توسعه نرم افزاری در یک شرکت، به یک فعالیت مدیریت عقد قرارداد کاهش می یابد.

تصمیم برای استفاده از منابع خارجی می تواند راهبردی یا فنی باشد. در سطح راهبردی، مدیران تجاری در نظر می گیرند که آیا بخش مهمی از تمام کار نرم افزار می تواند به دیگران واگذار شود. در سطح فنی، مدیر پروژه مشخص می کند که آیا بخشی یا تمام یک پروژه می تواند به بهترین شکل با بستن قرارداد پیمانکاری^۱ انجام شود.

به هر حال در دوران کنونی گرایش به استفاده از منابع خارجی (پیمانکاری) بسیار رونق یافته و بدون شک این روند با شتاب بیشتری ادامه می یابد.

۹-۱۶ ابزارهای برآورد خودکار

اگرچه ابزارهای برآورد خودکار متفاوت و بسیاری وجود دارند ولی همگی دارای خصوصیت‌های مشابهی هستند و همگی شش تابع عمومی زیر را انجام می دهند:

- ۱- تعیین اندازه عناصر قابل تحویل پروژه
- ۲- انتخاب فعالیت‌های پروژه
- ۳- پیش‌بینی میزان نیاز به منابع انسانی
- ۴- پیش‌بینی تلاش مورد نیاز نرم افزار
- ۵- پیش‌بینی هزینه نرم افزار

۶- پیش‌بینی زمان مورد نیاز برای انجام فعالیت‌های نرم افزاری

هنگامی که ابزارهای متفاوت برآورد برای داده‌های یک پروژه به کار گرفته می شوند، تفاوت‌های نسبتاً زیادی در نتایج برآورد مشاهده می شود. مهم تر اینکه، مقادیر پیش‌بینی شده گاه تا حد زیادی متفاوت با مقادیر واقعی هستند؛ بنابراین توصیه شود که خروجی ابزارهای برآورد به عنوان یک "نقطه داده‌ای" به کار روند که با استفاده از آن‌ها برآوردها تصحیح شوند؛ بنابراین عیب این ابزارها، تولید برآوردهایی است که با واقعیت فاصله زیادی دارند.

فصل هفدهم

تحلیل و مدیریت ریسک

هنگامی که بحث ریسک در مهندسی نرم افزار مطرح می شود، هر سه فرض مفهومی Pobert Charette همواره قابل مشاهده اند. (۱) ما با آینده سر و کار داریم - چه ریسک‌هایی ممکن است باعث ناکامی در پروژه نرم‌افزاری شوند؟ (۲) ما با تغییر سر و کار داریم، تغییرات در نیازها و خواسته‌های مشتری، فن‌آوری‌های توسعه، رایانه‌های مقصد و تمامی نهادهای دیگری که وابسته به پروژه هستند، چگونه بر زمان‌بندی و موفقیت پروژه تأثیر می‌گذارند؟ و (۳) سرانجام اینکه با انتخاب‌ها سر و کار داریم - از کدام روش‌ها و ابزارها باید استفاده کنیم؟ چند نفر نیرو لازم داریم؟ چقدر بر کیفیت تأکید شود؟

بنابراین:

۱-۱۷ راهبردهای ریسک واکنشی در برابر پیش‌واکنشی

۱-۱-۱۷ راهبرد واکنشی

در بهترین حالت پروژه را برای ریسک‌های احتمالی تحت نظر قرار می‌دهد و در بدترین حالت مدیریت بحران برای مقابله با ریسک تشکیل می‌گردد.

۲-۱-۱۷ راهبرد پیش‌واکنشی

یک راهبرد بسیار هوشمندانه‌تر برای مدیریت ریسک، راهبرد پیش‌بینی ریسک و داشتن برنامه برای مقابله با ریسک است. راهبرد پیش‌واکنشی مدت‌ها قبل از شروع کارهای فنی آغاز می‌شود. ریسک‌های بالقوه شناسایی می‌شود، احتمال وقوع و میزان تأثیر آن‌ها سنجیده و از لحاظ اهمیت، اولویت‌بندی می‌شوند. سپس، تیم نرم‌افزاری برنامه‌ای برای مدیریت ریسک تدارک می‌بیند. هدف اولیه، پرهیز از ریسک است اما از آنجا که همه ریسک‌ها قابل پیش‌بینی و جلوگیری نیستند، تیم کوشش می‌کند تا یک برنامه احتیاطی توسعه دهد و یا به خوبی در برابر آن‌ها عکس‌العمل نشان دهد.

۲-۱۷ ریسک‌های نرم‌افزاری

با توجه به فراوانی تعاریف مربوط به ریسک نرم‌افزار، این تعاریف در دو ویژگی مشترک هستند:

- عدم قطعیت: یعنی ریسک ممکن است اتفاق بیفتد یا نیافتد.
- ضایعات: یعنی اگر ریسک به‌وقوع بپیوندد حتماً ضایعاتی در بر خواهد داشت.

۱۷-۲-۱ گروه‌های متفاوت ریسک‌های نرم‌افزاری

- ریسک‌های پروژه‌ای زمان‌بندی پروژه را مد نظر دارد (مانند زمان‌بندی، هزینه، منابع، پرسنل، مشتری و نیازهای وی، اندازه و پیچیدگی پروژه).
- ریسک‌های فنی کیفیت و سر وقت آماده شدن نرم‌افزار را مد نظر دارد (مثل طراحی، پیاده‌سازی، واسط‌های نرم‌افزار، اعتبارسنجی، نگهداری، ابهام در مشخصات، عدم قطعیت فنی، بی‌استفاده شدن فنی، فن‌آوری پیشرو).
- ریسک‌های تجاری عملی بودن ساخت نرم‌افزار را تهدید می‌کنند که شامل ریسک بازار (محصول بدون مشتری)، ریسک راهبردی (در حیطه راهبردهای شرکت نیست، و بخش فروش نمی‌داند چگونه آن را بفروشد)، ریسک مدیریتی (از دست دادن پشتیبانی مدیریت ارشد)، ریسک بودجه‌ای (از دست دادن وعده‌های بودجه‌ای و پرسنلی) است.

۱۷-۳ شناسایی ریسک

برای هر یک از گروه‌های ارائه‌شده دو نوع متمایز ریسک وجود دارد: **ریسک‌های کلی و ریسک‌های خاص محصول**. ریسک‌های کلی برای همه پروژه‌های نرم‌افزاری تهدید بالقوه به شمار می‌روند. ریسک‌های خاص پروژه را فقط کسانی می‌توانند شناسایی کنند که درک درستی از فن‌آوری، افراد و محیط خاص پروژه مورد نظر دارند. برای شناسایی ریسک‌های خاص پروژه، برنامه‌ریزی پروژه و بیانیه حوزه عملکرد نرم‌افزار مورد بررسی قرار می‌گیرند و پاسخی برای این سؤال به دست می‌آید: "کدام یک از ویژگی‌های این محصول ممکن است برنامه‌ریزی ما را مورد تهدید قرار دهد؟"

- روشی برای شناسایی ریسک‌ها، ایجاد لیست کنترلی موارد ریسک است که در زیرگروه‌های کلی زیر جای می‌گیرند:
 - **اندازه محصول:** ریسک‌های مرتبط با اندازه نرم‌افزاری که قرار است ساخته یا اصلاح شود.
 - **تأثیر تجاری:** ریسک‌های مرتبط با شرایط حدی تحمیل‌شده از سوی مدیریت یا بازار کار.
 - **ویژگی‌های مشتری:** ریسک‌های مرتبط با پیچیدگی مشتری و توانایی توسعه‌دهنده در برقراری ارتباط با مشتری و جلب نظر وی.
 - **تعریف فرایند:** ریسک‌هایی که با درجه تعریف و بلوغ سازمان مهندسی نرم‌افزار توسط توسعه‌دهنده نرم‌افزار مرتبط است.
 - **محیط توسعه:** ریسک‌های مربوط به قابلیت و کیفیت ابزارهایی که برای ساخت محصول استفاده می‌شوند.
 - **فن‌آوری:** ریسک‌های مرتبط با پیچیدگی سیستمی که قرار است توسعه یابد، به همراه ریسک‌های مربوط به "تازگی" فن‌آوری به‌کاررفته در توسعه سیستم.
 - **تجربه و تعداد پرسنل:** ریسک‌های مرتبط با تجربه کاری و فنی مهندسان نرم‌افزار که مسئولیت توسعه محصول را دارند.
- لیست کنترلی موارد ریسک را به شیوه‌های متفاوت می‌توان سازماندهی کرد. سؤالات مرتبط با هر یک از موارد ذکر شده در بالا را می‌توان برای هر پروژه نرم‌افزاری پاسخ گفت. پاسخ این پرسش‌ها به برنامه‌ریز کمک می‌کند تا تأثیر ریسک‌ها را برآورد کند. به همین منظور یک مجموعه از "مؤلفه‌ها و محرک‌های ریسک" همراه با احتمال وقوع آن‌ها باید شناسایی و فهرست شوند.

۱۷-۳-۱ مؤلفه‌های ریسک

- **کارایی** (میزان عدم قطعیت برآورده شدن خواسته‌ها توسط نرم‌افزار)
- **هزینه** (عدم قطعیت در حفظ بودجه پروژه)
- **پشتیبانی** (عدم قطعیت در سهولت تصحیح، تطبیق و ارتقای نرم‌افزار)
- **زمان‌بندی** (میزان عدم قطعیت در رعایت زمان‌بندی و تحویل پروژه)

۱۷-۳-۲ محرک‌های ریسک

- تأثیر مؤلفه‌های ریسک را نشان می‌دهند که به چهار گروه زیر تقسیم می‌شوند:
- قابل چشم‌پوشی
 - کم‌اهمیت

- بحرانی
- فاجعه بار

پس از تعیین مؤلفه‌ها و محرک‌های مناسب، جدولی از محرک‌ها و مؤلفه‌های ریسک شکل می‌گیرد و پیش‌بینی لازم در مورد هر ریسک در جدولی مشابه جدول زیر انجام می‌گیرد.

جدول ۱۷-۱ جدول ارزیابی محرک‌ها و مؤلفه‌های ریسک

زمان بندی	هزینه	پشتیبانی	کارایی	گروه	
	شکست منجر به افزایش هزینه‌ها و تأخیر زمانی با مقدار احتمالی ... ریال می‌شود.	شکست دربرآوردن نیازها به شکست مأموریت پروژه می‌انجامد.	۱	فاجعه بار	
تعداد خطوط کد (LOC) غیر قابل دستیابی	کمبود مالی چشمگیر، احتمال عدم کفایت بودجه	نرم‌افزار نامناسب یا غیر قابل پیش‌بینی	۲		تنزل چشمگیر تا حد عدم دستیابی به کارایی فنی
...	۱	بحرانی
...	۲	
				۱	اهمیت کم
				۲	
				۱	چشم‌پوشی قابل
				۲	
۲- پیامد بالقوه به دست نیامدن نتیجه مطلوب		۱- پیامد بالقوه خطاها یا معایب کشف‌نشده نرم‌افزار			

۴-۱۷ پیش‌بینی ریسک (برآورد ریسک)

در پیش‌بینی ریسک که برآورد ریسک نیز نامیده می‌شود، کوشش می‌شود ریسک از دو لحاظ مورد سنجش قرار گیرد:

- ۱- احتمال وقوع،
- ۲- پیامدهای مرتبط،

۱-۴-۱۷ فعالیت‌های پیش‌بینی ریسک

- تعیین مقیاسی که احتمال ریسک را منعکس می‌کند.
- تعیین پیامدهای ریسک
- برآورد تأثیر ریسک بر محصول
- توجه به صحت کلی پیش‌بینی ریسک

۲-۴-۱۷ تهیه جدول ریسک

پس از تعیین موارد ریسک جدولی برای ارزیابی ریسک به صورت زیر ایجاد می‌گردد.

جدول ۲-۱۷ جدول ارزیابی ریسک

ریسک ها	گروه	احتمال	تأثیر	RMMM
برآورد اندازه‌ها ممکن است بسیار کم باشد	ps	٪۶۰	۲	
تعداد کاربران بیشتر از حد برنامه‌ریزی شده	ps	٪۳۰	۳	
استفاده مجدد کمتر از حد برنامه‌ریزی شده	Ps	٪۷۰	۲	
مقاومت کاربران نهایی در برابر سیستم	bu	٪۴۰	۳	
کم بودن مهلت تحویل	bu	٪۵۰	۲	
از دست رفتن سرمایه	cu	٪۴۰	۱	
تغییر خواسته‌های مشتری	ps	٪۸۰	۲	
عدم برآوردن انتظارات توسط فن‌آوری	te	٪۳۰	۱	
عدم آموزش ابزارها به افراد	de	٪۸۰	۳	
بی‌تجربگی پرسنل	st	٪۳۰	۲	
بالا بودن بازدهی پرسنل	st	٪۶۰	۲	

در این جدول:
 PS: ریسک اندازه پروژه BU: ریسک تجاری CU: ریسک مشتری
 TE: ریسک فن‌آوری DE: ریسک محیط توسعه ST: ریسک پرسنلی
 ارزش تأثیرات: ۱ - فاجعه‌بار ۲- بحرانی ۳- کم‌اهمیت ۴- قابل چشم‌پوشی

هنگامی که چهار ستون اول جدول ریسک تکمیل شد، این جدول بر اساس احتمال و تأثیر آن مرتب می‌شود. ریسک‌های با احتمال بالا و تأثیر زیاد در بالای جدول و ریسک‌های با احتمال پایین در انتهای جدول قرار می‌گیرند. به این ترتیب اولویت‌بندی مرتبه اول ریسک‌ها انجام می‌شود.

مدیر پروژه، جدول مرتب‌شده حاصل را مطالعه کرده یک خط مرزی تعیین می‌کند. این خط مرزی (که به طور افقی در نقطه‌ای از جدول کشیده می‌شود نشان می‌دهد که فقط ریسک‌های واقع در بالای این خط، شایسته توجه بیشتر و برنامه‌ریزی هستند. ریسک‌هایی که زیر این خط قرار می‌گیرند، دوباره ارزیابی می‌شوند تا اولویت‌بندی مرتبه دوم صورت پذیرد.
 همه ریسک‌هایی که در بالای خط مرزی قرار می‌گیرند، باید مدیریت ریسک در مورد آن‌ها اعمال شود. ستونی که با RMMM مشخص شده است، به طرح مدیریت، نظارت و تقلیل ریسک^۱ یا به عبارت دیگر، به مجموعه‌ای از برگه‌های اطلاعات ریسک اشاره دارد که باید برای کلیه ریسک‌های بالای خط مرزی تهیه گردد. شکل زیر ارتباط بین میزان تأثیر، احتمال و مدیریت ریسک را نشان می‌دهد.

۵-۱۷ در معرض ریسک

میزان کلی قرار گرفتن در معرض ریسک^۲ با RE نشان داده می‌شود و به صورت زیر تعریف می‌گردد:

$$RE=P*C$$

که در آن P احتمال وقوع ریسک و C هزینه اضافی ریسک است.

۶-۱۷ سنجش ریسک

فرایند مدیریت ریسک مجموعه‌ای از نقاط سه‌بعدی به صورت زیر را شناسایی می‌کند:

$$[r_j, l_j, x_j]$$

r_j ریسک، l_j احتمال ریسک و x_j تأثیر ریسک است. لذا در فرایند سنجش ریسک باید فعالیت‌های زیر انجام شوند:

- شناسایی ریسک
 - تعیین احتمال ریسک
 - تعیین تأثیر ریسک
 - اولویت‌بندی ریسک
 - تهیه راه‌های کنترل ریسک
- بنابراین در اثنای سنجش ریسک مراحل زیر را اجرا می‌کنیم:
- تعریف سطوح مرجع ریسک برای پروژه
 - کوشش برای ایجاد رابطه میان هر $[r_j, l_j, x_j]$ و هر یک از سطوح مرجع
 - پیش‌بینی مجموعه‌ای از نقاط مرجعی که ناحیه‌ای را معین می‌کنند.
 - کوشش برای پیش‌بینی چگونگی تأثیر‌گذاری ترکیبی از ریسک‌ها بر یک سطح مرجع

۷-۱۷ سنجش تأثیر ریسک

اگر ریسکی به وقوع بپیوندد، احتمالاً یک سری پیامد دربر خواهد داشت که این پیامدها از سه عامل تأثیر می‌پذیرند. این عوامل عبارت‌اند از: **ماهیت، دامنه و زمان‌بندی**. ماهیت ریسک، مشکلاتی را نشان می‌دهد که در صورت وقوع ریسک، مورد انتظار هستند. برای مثال، یک واسط خارجی برای سخت‌افزار که ضعیف تعریف شده باشد (ریسک فنی)، از طراحی اولیه و آزمایشات حذف شده و احتمالاً منجر به بروز مشکلاتی در یکپارچگی سیستم در اواخر پروژه می‌شود. دامنه ریسک، ترکیبی از شدت ریسک (میزان جدی بودن آن) با توزیع کلی آن (چه میزان از پروژه تأثیر می‌پذیرد یا چند مشتری دچار زیان می‌شوند) است. زمان‌بندی ریسک به زمان احساس تأثیر ریسک و مدت به طول انجامیدن آن مربوط می‌شود.

به طور کلی مراحل زیر برای تعیین پیامدهای یک ریسک پیشنهاد می‌شود:

- تعیین احتمال وقوع هر مؤلفه ریسک
- تعیین تأثیر هر مؤلفه بر اساس ملاک‌های یادشده
- کامل کردن جدول ریسک و تحلیل نتایج

۸-۱۷ برنامه تقلیل، نظارت و مدیریت ریسک (برنامه RMMM)

همه فعالیت‌های تحلیل ریسک، یک هدف منفرد دارند، یعنی کمک به تیم پروژه برای توسعه راهبردی در مقابله با ریسک. در یک راهبرد مؤثر، سه مسئله زیر باید در نظر گرفته شود:

- اجتناب از ریسک
- نظارت بر ریسک
- مدیریت ریسک و برنامه‌ریزی احتیاطی

لازم به ذکر است که مراحل RMMM باعث افزایش هزینه پروژه می شود. برای مثال، صرف وقت جهت تأمین نیروی پشتیبان برای هر یک از افراد فنی مهم، هزینه دربردارد. بنابراین، بخشی از مدیریت ریسک، ارزیابی زمانی است که مزایای حاصل از اجرای RMMM، بر هزینه‌های پیاده‌سازی آن‌ها غلبه می کند

برای یک پروژه بزرگ، ۳۰ یا ۴۰ ریسک ممکن است شناسایی شود. اگر برای هر ریسک بین ۳ تا ۷ مرحله مدیریت ریسک در نظر بگیریم، مدیریت ریسک، خود به تنهایی یک پروژه می شود! از همین رو قاعده ۲۰-۸۰ پارتو را در مورد ریسک نرم‌افزاری اعمال می کنیم. به همین دلیل، برخی از ریسک‌های شناسایی شده، سنجش و پیش‌بینی شده ممکن است به طرح RMMM راه پیدا نکنند. در نتیجه:

- در پروژه‌های بزرگ عقل سلیم بر تحلیل، کنترل و نظارت ریسک حکم می کند.
- مدیریت ریسک ممکن است زمان زیادی را صرف کند و هزینه نیز داشته باشد ولی ارزش این کار را دارد.

فصل هجدهم

زمان بندی، نظارت و کنترل پروژه

برای انجام هر پروژه نرم‌افزاری نیاز به داشتن برنامه اجرایی، نظارت و کنترل بر اجرای آن است. برای این کار ابتدا مدل فرایندی مناسبی را انتخاب کنید، کارهای مربوط به مهندسی نرم‌افزار را که باید انجام شوند مشخص کنید. سپس میزان کار و تعداد افراد و نیروی تخصصی مورد نیاز با منابع مربوطه را برآورد کرده و به کارهای پروژه تخصیص دهید. در حین اجرای کار اگر ریسکی رخ دهد، اثرات آن را در شبکه فعالیت‌ها منعکس کرده و شبکه را به دنیای واقعیت نزدیک‌تر کنید. این اعمال را مجموعاً **زمان بندی، نظارت و کنترل پروژه** گویند.

۱-۱۸ دلایل تأخیر و شکست پروژهها

دلایل زیادی برای تأخیر پروژه‌های نرم‌افزاری وجود دارد که در زیر برخی از علل ریشه‌ای آن ارائه شده است:

- مهلت‌های غیر واقع‌بینانه
- تغییر خواسته‌ها و نیازهای مشتری
- برآورد اشتباه منابع و کار
- در نظر نگرفتن ریسک‌ها
- مشکلات فنی غیر قابل پیش‌بینی
- مشکلات منابع انسانی
- ارتباطات نادرست بین مدیران، رهبران و اعضای تیم‌ها
- عدم کنترل مناسب پروژه و زمان بندی توسط مدیر پروژه
- مدیر پروژه بی تجربه و نالایق
- عدم تعهد کافی اعضای تیم
- تأثیرات ناشی از رقبا
- و ...

۱۸-۲ اصول و مفاهیم پایه‌ای زمان بندی

اهداف مدیر پروژه عبارت‌اند از: تعریف تمام فعالیت‌های پروژه، ایجاد شبکه‌ای که وابستگی‌های داخلی آن‌ها را مشخص می‌کند، مشخص کردن فعالیت‌هایی که در این شبکه بحرانی هستند و سپس پیگیری پیشرفت آن‌ها و شناسایی تأخیرات. برای دستیابی به این اهداف، مدیر پروژه باید نوعی زمان بندی ارائه دهد که دارای درجه‌ای از دقت تعریف شده باشد به طوری که قابلیت نظارت و کنترل بر پیشرفت پروژه در آن نیز وجود داشته باشد.

زمان بندی پروژه‌های نرم افزاری فعالیتی است که کارهای برآورد شده را به وظایف مهندسی نرم افزار اختصاص می‌دهد. به هر حال توجه به این نکته مهم است که این زمان بندی با گذشت زمان، تکامل می‌یابد. در **مراحل اولیه برنامه ریزی پروژه**، یک **زمان بندی ماکروسکوپی** توسعه داده می‌شود. این نوع زمان بندی مشخص کننده تمام فعالیت‌های عمده مهندسی نرم افزار و توابعی از محصول است. با قرار گرفتن پروژه در مسیر خود، هر ورودی زمان بندی ماکروسکوپی به صورت زمان بندی همراه با جزئیات تعریف می‌شود. در اینجا، کارهای خاص نرم افزاری (مورد نیاز برای دستیابی به یک فعالیت) مشخص و زمان بندی می‌شوند.

مانند تمام زمینه‌های دیگر مهندسی نرم افزار، اصول بنیادی زیر، رهنمودهایی را در رابطه با زمان بندی پروژه نرم افزاری ارائه می‌دهند:

- **تقسیم:** پروژه باید به چند فعالیت و کار قابل مدیریت تقسیم شود. برای دستیابی به این قطعه بندی، محصول و فرایند باید تجزیه شوند.
- **وابستگی:** وابستگی هر یک از فعالیت‌ها یا کارها باید مشخص شود. برخی کارها باید به ترتیب انجام گیرند در حالی که بقیه می‌توانند به شکل موازی انجام شوند.
- **تخصیص زمان:** به هر کاری که باید زمان بندی شود، تعدادی واحد کاری (برای مثال، نفر-روز) اختصاص می‌یابد. علاوه بر آن، به هر کار باید تاریخ شروع و تاریخ تکمیلی داده شود که تابعی از رابطه بین کارها و نوع تخصیص کار به صورت تمام وقت یا پاره وقت است.
- **اعتبار سنجی کارها:** هر پروژه چند سرپرست تعریف شده دارد. با تخصیص زمان، مدیر پروژه باید مطمئن شود که در هر مرحله بیشتر از تعداد افراد اختصاص یافته برای کار مورد نظر تخصیص داده نشده‌اند.
- **تعیین وظایف:** هر کاری که زمان بندی شده است باید به عضو مشخصی از تیم اختصاص یابد.
- **تعیین پیامدها:** هر کاری که زمان بندی شده است باید نتیجه تعریف شده‌ای نیز داشته باشد. برای پروژه‌های نرم افزاری، این نتیجه به طور معمول یک محصول کاری (برای مثال، طراحی یک پیمانانه)، یا بخشی از یک محصول کاری است. محصولات کاری اغلب به صورت بخش‌های قابل تحویل با هم ترکیب می‌شوند.
- **تعیین نقاط شاخص زمانی:** هر کار یا گروهی از کارها باید با یک معیار پروژه همراه باشد. یک معیار، زمانی به دست می‌آید که یک یا چند محصول کاری برای تعیین کیفیت باید مورد بازنگری و بازبینی قرار گیرند و توسط مدیریت پروژه تأیید شوند.

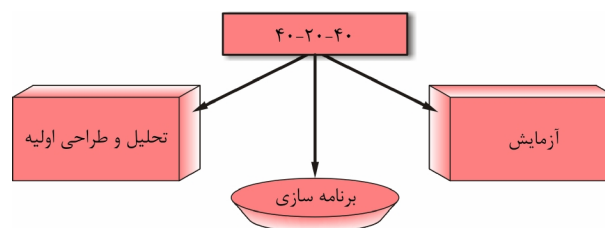
۱۸-۳ رابطه بین افراد و کار

یک نکته کوچک مشترک وجود دارد که هنوز اکثر مدیرانی که مسئول فعالیت توسعه نرم افزار هستند به آن باور دارند: **"اگر از زمان بندی عقب باشیم، همیشه می‌توانیم برنامه نویس‌های بیشتری را در مدت پروژه به کار گیریم."** بدبختانه، افزودن افراد بیشتر به پروژه در طول اجرای آن، اغلب تأثیر بازدارنده در پروژه داشته و باعث می‌شود زمان بندی حتی بیشتر به درازا بکشد. افرادی که به پروژه افزوده می‌شوند، باید آموزش داده شوند و افرادی که به آن‌ها می‌آموزند همان کسانی هستند که پروژه را انجام می‌دهند. در حال آموزش، کاری انجام نمی‌شود و پروژه بیشتر به تأخیر می‌افتد.

علاوه بر زمان لازم برای یادگیری سیستم، افزایش تعداد اعضای تیم باعث افزایش مسیرهای ارتباطی بین اعضای تیم شده و پیچیدگی ارتباطات را در پروژه به دنبال دارد. اگرچه ارتباطات کاملاً برای توسعه نرم افزاری موفق ضروری هستند ولی هر مسیر ارتباطی جدید، نیازمند فعالیت و در نتیجه زمان بیشتری است.

۴-۱۸ توزیع کار

توزیع توصیه شده‌ای برای فعالیت‌های مهندسی نرم افزار در مراحل توسعه نرم افزار، اغلب قانون ۴۰-۲۰-۴۰ نامیده می‌شود. چهل درصد تمام فعالیت‌ها به تحلیل و طراحی اولیه تخصیص یافته است. یک درصد دیگر به آزمایش نهایی اختصاص می‌یابد. می‌توان نتیجه گرفت که تأکید کمتری بر برنامه‌نویسی است (۲۰ درصد کل فعالیت). این توزیع فعالیت‌ها فقط به عنوان راهنما قابل استفاده است. خصوصیات هر پروژه، توزیع فعالیت‌ها را مشخص کند.



شکل ۱-۱۸ توزیع پیشنهادی فعالیت‌ها

۵-۱۸ مجموعه کاری و انواع پروژه

مجموعه‌ای از کارهای مورد نیاز برای سازماندهی انواع مختلف پروژه‌ها و درجات مختلف توانایی طراحی شده‌اند. اگرچه توسعه یک رده‌بندی کامل از انواع پروژه‌های نرم افزاری مشکل است، ولی اکثر سازمان‌های نرم افزاری پروژه‌های زیر را در نظر می‌گیرند:

- ۱- **پروژه‌های توسعه مفهومی**، برای توصیف و طراحی یک مفهوم جدید تجاری یا کاربردی از یک فن‌آوری جدید تعریف می‌شوند.
- ۲- **پروژه‌های توسعه کاربرد جدید**، بر اساس درخواست مشتری منجر به تولید نرم افزار می‌شوند و نتیجه آن تولید یک محصول نرم افزاری کاربردی است.
- ۳- **پروژه‌های ارتقای نرم افزار کاربردی**، زمانی انجام می‌شوند که نرم افزار نیاز به اصلاحات عمده‌ای برای عملکرد، کارایی یا واسطه‌های کاربر نهایی دارد.
- ۴- **پروژه‌های پشتیبانی نرم افزارهای کاربردی** شامل فعالیت‌های نگهداری نرم افزار جاری و در حال کار است.
- ۵- **پروژه‌های مهندسی مجدد**، با هدف باز مهندسی سیستم موجود به صورت کلی یا جزئی اجرا می‌شوند.

۱-۵-۱۸ درجه دشواری

حتی برای پروژه‌ای خاص غیر ممکن است که درجه دشواری فرایند نرم افزار تا حد زیادی تغییر کند. درجه دشواری تابعی از خصوصیات پروژه است که به صورت‌های زیر تعریف شده است.

- **معمولی**: تمام فعالیت‌های پشتیبانی (فصل ۲) به کار گرفته می‌شوند، اما فقط مجموعه کاری حداقل مورد نیاز است. در حالت عمومی، فعالیت‌های مورد نظر به حداقل رسیده و نیازهای مستندسازی کاهش می‌یابند. تمام اصول بنیادی مهندسی نرم افزار هنوز قابل دستیابی هستند.
- **ساخت یافته**: فرایند مهندسی نرم افزار برای پروژه به کار گرفته می‌شود. فعالیت‌های فرایندی، پشتیبانی و کارهای مرتبط مناسب برای این نوع پروژه انجام می‌شوند و فعالیت‌های لازم برای اطمینان از کیفیت بالا صورت می‌گیرند. تضمین کیفیت (SQA)^۱ و مدیریت پیکربندی (SCM)^۲ مستندسازی شده و کارهای اندازه‌گیری به روش تدریجی به کار گرفته می‌شوند.
- **دقیق**: فرایند کامل برای پروژه به کار گرفته می‌شود، البته با درجه‌ای از نظم که دسترسی به کیفیت بالا را فراهم می‌کند. تمام فعالیت‌های فرایند مهندسی نرم افزار به کار گرفته می‌شوند و محصولات کاری قدرتمندی تولید می‌شود.

1 - Software Quality Assurance

2 - Software Configuration Management

- **عکس‌العمل‌های سریع:** زمینه کاری فرایند برای این پروژه به کار گرفته می‌شود، اما به علت سریع بودن، فقط کارهای ضروری برای دستیابی به کیفیت مناسب انجام می‌شود. توسعه مجموعه کاملی از مستندات، هدایت کردن مرورهای اضافی که Backfilling نامیده می‌شوند، بعد از تحویل محصول به مشتری صورت می‌گیرند.

۲-۵-۱۸ معیارهای تطبیق

معیارهای تطبیق برای تعیین درجه دشواری پروژه به منظور تعیین میزان فعالیت‌های مورد نیاز استفاده می‌شود. یازده معیار برای پروژه‌های نرم‌افزاری در جدول ۱-۱۸ تعریف شده است.

به هر یک از معیارهای تطبیق درجه‌ای بین ۱ تا ۵ نسبت داده می‌شود که ۱ نشان‌دهنده پروژه‌ای است که در آن مجموعه‌ای از کارهای فرایند مورد نیاز است و تمام نیازهای فرایندی و مستندسازی حداقل هستند و ۵ نشان‌دهنده پروژه‌ای است که در آن مجموعه‌ای کامل از کارهای فرایند باید به کار گرفته شوند و انجام تمام فعالیت‌های فرایندی و نیازهای مستندسازی ضروری هستند.

۳-۵-۱۸ انتخاب وظایف و محاسبه مقدار TSS

برای تعیین مجموعه منتخب وظایف مناسب برای پروژه، مراحل زیر باید انجام گیرند:

- ۱- هر یک از معیارهای تطبیق را مرور و درجه مناسب (بین ۱ تا ۵) را بر مبنای خصوصیات پروژه به آن‌ها نسبت دهید.
- ۲- **ارزش‌های** نسبت داده‌شده به هر معیار را مرور کنید. مقدار یک فاکتور ارزشی را بین ۰/۸ تا ۱/۲ تعیین کنید که نمایشی از اهمیت نسبی معیار تطبیق خاص برای انواع پروژه‌ها در محیط محلی فراهم کند. اگر اصلاحاتی مورد نیاز باشد باید انجام شود.
- ۳- ضریب نقطه ورودی مقدار صفر یا ۱ برای هر نوع پروژه دارد و رابطه معیار تطبیق را با نوع پروژه مشخص می‌کند.
- ۴- نتیجه حاصل ضرب زیر در ستون نتیجه درج می‌شود:

$$\text{ضریب نقطه ورودی} \times \text{ضریب ارزش} \times \text{درجه}$$

۵- متوسط تمام مقادیر ستون نتیجه محاسبه می‌شود که انتخاب‌کننده مجموعه وظایف (TSS)^۱ نامیده می‌شود. این مقدار برای کمک به

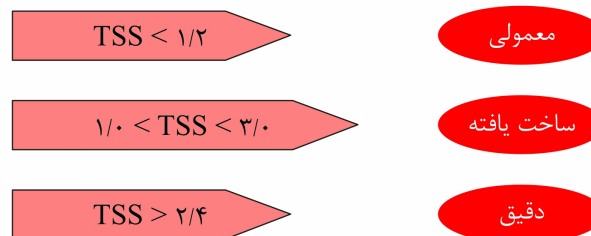
انتخاب مجموعه وظایف مورد نیاز برای یک پروژه مناسب‌ترین معیار است.

در جدول ۱-۱۸ برای توسعه یک نرم‌افزار کاربردی جدید مقادیر درجه و ارزش تعیین شده و با توجه به ضریب نقطه ورودی مربوط به ستون کاربرد جدید، مقادیر مربوط به ستون نتیجه محاسبه شده است.

جدول ۱۸-۱ انتخاب مجموعه وظایف پروژه

نتیجه	ضرب نقطه ورودی بر حسب نوع پروژه					ارزش	درجه	معیارهای انطباق
	مهندسی مجدد	کاربرد جدید	ارتقا	پشتیبانی	مفهومی			
۲/۴	۱	۱	۱	۱	۰	۱/۲	۲	اندازه پروژه
۳/۳	۱	۱	۱	۱	۰	۱/۱	۳	تعداد کاربران بالقوه
۴/۴	۱	۱	۱	۱	۰	۱/۱	۴	اهمیت مأموریت
۲/۷	۰	۱	۱	۰	۰	۰/۹	۳	طول عمر برنامه کاربردی
۲/۴	۱	۱	۱	۱	۰	۱/۲	۲	پایداری نیازها
۱/۸	۱	۱	۱	۱	۱	۰/۹	۲	سهولت ارتباط با مشتری
۱/۸	۱	۱	۰	۰	۱	۰/۹	۲	بلوغ فن آوری قابل اجرا
۲/۴	۱	۱	۱	۰	۰	۰/۸	۳	شرایط اجرایی و کارایی
۳/۶	۱	۱	۱	۰	۱	۱/۲	۳	ویژگی های توکار و غیرتوکار
۲/۰	۱	۱	۱	۱	۱	۱/۰	۲	مدیریت پروژه
۴/۴	۱	۱	۱	۱	۰	۱/۱	۴	عملکرد داخلی
۰/۰	۱	۰	۰	۰	۰	۱/۲	۰	عوامل مهندسی مجدد
۲/۸	TSS							

۶- با توجه به مقدار TSS اقدام به انتخاب مجموعه فعالیت‌های مهندسی نرم‌افزار برای پروژه‌ها به صورت زیر می‌کنیم:



شکل ۱۸-۲ نحوه استفاده از TSS در تعیین درجه دشواری پروژه و انتخاب وظایف

همپوشانی در مقادیر TSS، هدفمند بوده و حاکی از آن است که تعریف مرزهای مشخص به هنگام انتخاب مجموعه وظایف غیر ممکن و بستگی به تجربه و مهارت مدیر پروژه و ویژگی‌های محصول دارد.

۱۸-۶ مدیریت پروژه‌های نرم‌افزاری شی‌گرا

۱۷-۶-۱ فعالیت‌های مدیریت مدرن پروژه نرم‌افزاری

- ۱- ساختن یک چارچوب فرایند مشترک برای پروژه
- ۲- استفاده از چارچوب و معیارهای تاریخی برای برآورد کار و زمان لازم
- ۳- استقرار نقاط شاخص‌های زمانی که اندازه‌گیری پیشرفت کار را میسر می‌کنند.
- ۴- تعریف یک سری نقاط کنترلی برای مدیریت ریسک، تضمین کیفیت و کنترل پروژه
- ۵- مدیریت تغییرات که به طور ذاتی در پیشرفت پروژه رخ می‌دهند.
- ۶- پیگیری، نظارت و کنترل پیشرفت

بنابراین به طور کلی فعالیت‌های زیر باید در فرایند مدیریت پروژه‌های شی‌گرا صورت گیرد:

- تعریف چارچوب فرایند مشترک
- تعیین معیارهای برآورد پروژه شی‌گرا
- تعریف یک رهیافت برآورد و زمان‌بندی
- استقرار پیگیری پیشرفت پروژه شی‌گرا

۱۸-۶-۲ تعریف چارچوب فرایند مشترک

چارچوب فرایند مشترک (CPF) روشی را تعیین می‌کند که یک سازمان برای مهندسی نرم‌افزار تعیین می‌کند.

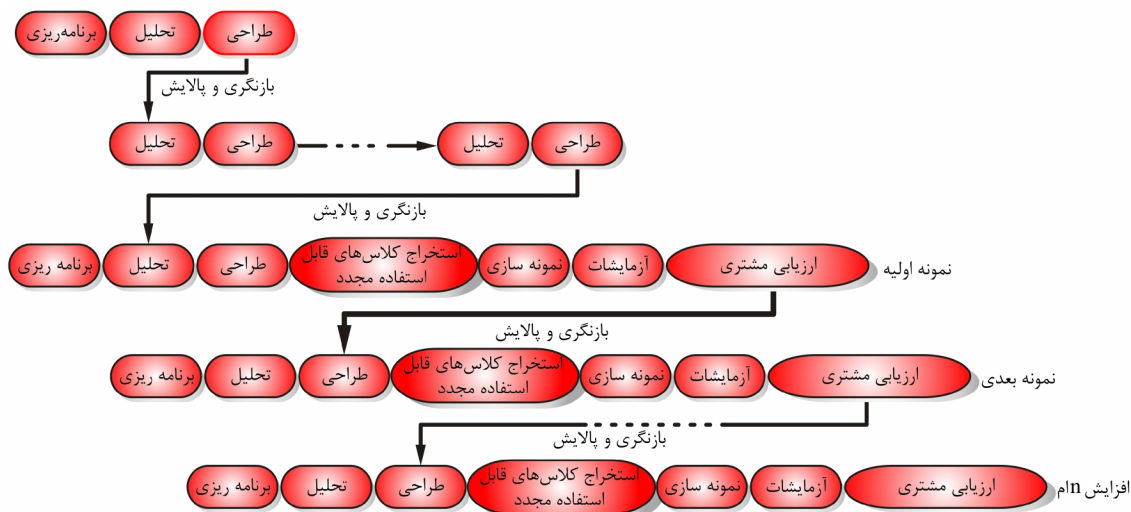
- CPF تعیین می‌کند که چه الگویی برای توسعه و نگهداری نرم‌افزار باید استفاده شود و چه وظایف و نقاط شاخص زمانی مورد نیاز است.
- CPF درجه سختی پروژه‌های مختلف را تعیین می‌کند.
- CPF همواره قابل انطباق است و می‌تواند نیازهای فردی تیم پروژه را برآورده کند.

Grady و Booch مدل بازگشتی/ موازی را برای توسعه نرم‌افزارهای شی‌گرا پیشنهاد می‌کنند. استفاده از مدل بازگشتی/ موازی در مهندسی نرم‌افزار شی‌گرا مستلزم اجرای مراحل زیر است:

- ۱- انجام تحلیل کافی برای جدا کردن کلاس‌ها و ارتباطات اصلی مسئله
- ۲- انجام اندکی طراحی برای تعیین اینکه آیا کلاس‌ها و ارتباطات را می‌توان عملاً پیاده‌سازی کرد یا خیر.
- ۳- استخراج اشیای قابل استفاده مجدد از یک کتابخانه، برای ساختن یک نمونه اولیه تقریبی
- ۴- اجرای چند آزمون جهت کشف خطاها در نمونه اولیه
- ۵- دریافت نظرات مشتری در خصوص نمونه اولیه
- ۶- اصلاح مدل تحلیل بر اساس بازخوردی که از نمونه اولیه، از انجام طراحی و از نظرات مشتری به دست آمده است.
- ۷- پالایش طراحی به منظور انجام تغییرات
- ۸- برنامه‌نویسی اشیای خاص (که در کتابخانه موجود نیست).
- ۹- یکپارچه‌سازی یک نمونه اولیه جدید با استفاده از اشیای کتابخانه و اشیای جدیدی که ایجاد کرده‌اید.
- ۱۰- اجرای چند آزمون برای کشف خطاها در نمونه اولیه
- ۱۱- دریافت نظریات مشتری در خصوص نمونه اولیه

این روش آن‌قدر ادامه می‌یابد تا نمونه اولیه به یک برنامه کاربردی تبدیل شود. مدل بازگشتی/ موازی شبیه به الگوی تکاملی یا حلزونی است زیرا:

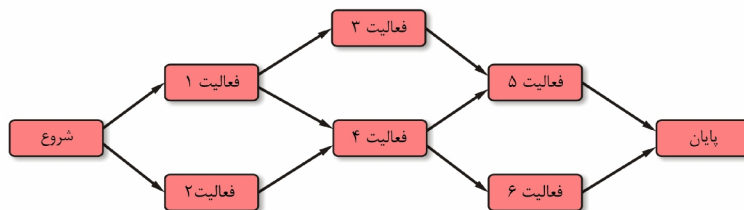
- هر دور تکرار نیاز به برنامه‌ریزی، مهندسی (تحلیل، طراحی، استخراج کلاس‌ها، ایجاد نمونه اولیه و آزمون) و فعالیت‌های ارزیابی دارد.
- در اثنای برنامه‌ریزی، فعالیت‌های مرتبط با هر یک از مؤلفه‌های مستقل، برنامه‌ریزی و زمان‌بندی می‌شود.
- مراحل اولیه مهندسی، تحلیل و طراحی به طور تکراری انجام می‌شود. هدف آن است که تمامی عناصر مهم مدل‌های تحلیل و طراحی شیء‌گرا مشخص شوند.
- به موازات پیشرفت کارهای مهندسی، نسخه‌های تدریجی از نرم‌افزار ساخته می‌شود.
- در اثنای این تکامل، برای هر گام یک‌سری بازبینی، ارزیابی مشتری و آزمون اجرا می‌شود که نتیجه آن بر فعالیت برنامه‌ریزی بعدی تأثیر می‌گذارد، (شکل ۱۸-۳ را ببینید).



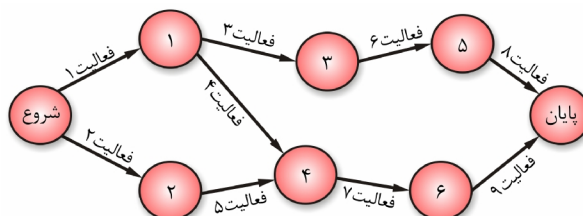
شکل ۱۸-۳ فرایندهای متداول برای یک پروژه شیء‌گرا

۱۸-۲ تعریف شبکه وظایف

یک شبکه کاری که شبکه فعالیت نیز نامیده می‌شود، نمایش گرافیکی از جریان کاری پروژه است. گاهی از آن به عنوان مکانیزمی استفاده می‌شود که از طریق تعریف دنباله کاری و وابستگی‌های آن‌ها به عنوان ورودی به یک ابزار خودکار، زمان‌بندی پروژه انجام می‌شود. دو نوع شبکه AON^۱ و AOA^۲ وجود دارد. شکل‌های زیر یک نمونه از هر نوع شبکه را نشان می‌دهد.



شکل ۱۸-۴ شبکه فعالیت‌ها در AON



شکل ۱۸-۵ شبکه فعالیت‌ها در AOA

1 - Activity On Node
2 - Activity On Arc

۸-۱۸ زمان بندی پروژه

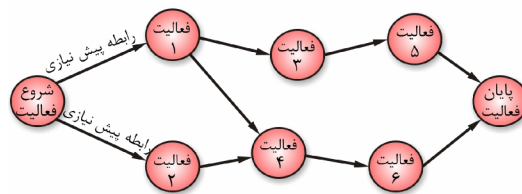
فن ارزیابی و بازنگری پروژه / برنامه (PERT)^۱ و روش مسیر بحرانی (CPM)^۲ دو روش زمان بندی پروژه هستند که می توانند در توسعه نرم افزار به کار گرفته شوند. هر دو روش زمان بندی شبکه توسط اطلاعات حاصل از فعالیت های برنامه ریزی پروژه به دست می آیند. وابستگی های بین کارها می توانند با استفاده از یک شبکه کاری تعریف شوند. کارها که گاهی ساختار شکست کار (WBS)^۳ پروژه نیز نامیده می شوند برای کل محصول و برای توابع جزئی تعریف می شوند.

PERT و CPM ابزارهایی کمی را فراهم می کنند که به برنامه ریز پروژه امکان می دهد تا:

- ۱- مسیر بحرانی را مشخص کند که شامل زنجیره ای از فعالیت ها برای مشخص کردن مدت پروژه است.
- ۲- تخمین های زمانی با احتمال بالا را برای هر یک از فعالیت ها با به کارگیری مدل های آماری ایجاد کند.
- ۳- زمان های بحرانی، شناوری و مرزی را محاسبه کرده که برای فعالیت ها تعریف شده اند.

۱۸-۱۸ روش مسیر بحرانی

شبکه AON را در نظر بگیرید:



شکل ۱۸-۶ مثال شبکه AOA فعالیت ها

برای هر گره در شبکه AOA زودترین زمان شروع فعالیت های بعد از گره i را به صورت زیر و به روش پیشرو محاسبه می کنیم:

$$\text{Set } ES_{\text{start}} = 0 \text{ or Date,}$$

$$ES_i = \text{Max} \{ ES_j + d_{ij} \}, \quad \forall i \in \text{شبکه}$$

در انتهای شبکه با قرار دادن مقدار محاسبه شده به عنوان آخرین مهلت تکمیل فعالیت های پروژه در آخرین گره، آخرین مهلت تکمیل فعالیت های قبل از گره j را به روش عقبگرد به شرح زیر محاسبه می کنیم:

$$\text{Set } LF_{\text{end}} = ES_{\text{end}}$$

$$LF_i = \text{Min} \{ LF_j - d_{ij} \}, \quad \forall j \in \text{شبکه}$$

در این صورت، مسیر بحرانی به صورت زیر تعیین می شود:

$$ES_i = LF_i \quad \forall i$$

$$ES_j = LF_j \quad \forall j$$

$$ES_j - ES_i = LF_j - LF_i = d_{ij} \quad \forall i, j \in \text{شبکه}$$

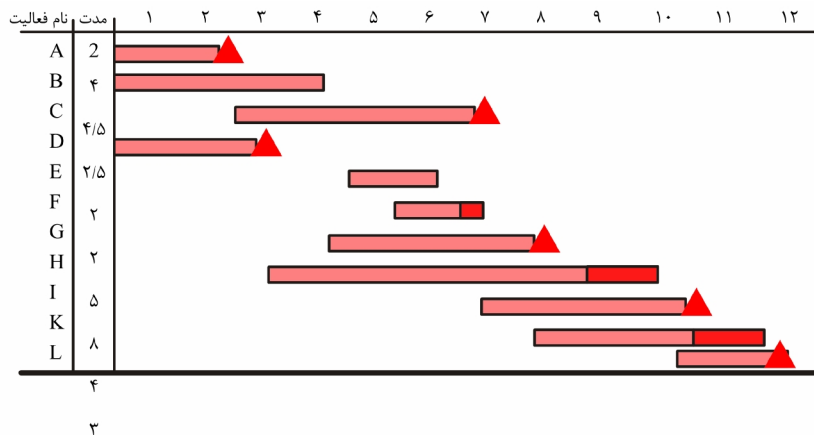
۱۸-۲ نمودار گانت

نمونه ای از نمودار گانت که نمودار خط زمانی نیز نامیده می شود. در شکل زیر نشان داده شده است.

1 - Project/Program Evaluation and Review Technique

2 - Critical Path Method

3 - Work Breakdown Structure



شکل ۱۸-۷ نمونه‌ای از نمودار گانت

۱۸-۳ پیگیری زمان بندی

زمان بندی پروژه امکان نظارت بر پیشرفت پروژه را برای مدیر پروژه نرم افزاری فراهم می کند. اگر زمان بندی پروژه ای امکان کنترل به شکل مناسب را داشته باشد، کنترل پروژه وجود داشته و موفقیت پروژه تا حد زیادی تضمین می شود؛ بنابراین:

- پیگیری زمان بندی، نقشه راهنمای مدیر پروژه است.
 - پیگیری زمان بندی، وظایف و نقاط شاخص های زمانی^۱ پروژه را مشخص می کند.
- نظارت، کنترل و پیگیری زمان بندی پروژه را در صورت مشخص بودن معیارهای پیگیری و کنترل پیشرفت پروژه می توان به چند روش انجام داد:

- برپایی نشست های ادواری
- ارزیابی نتایج بازبینی
- کنترل کردن شاخص های زمانی
- مقایسه تاریخ ها (شروع و پایان)
- استفاده از تحلیل ارزش حاصله

۱۸-۹ استقرار معیارهای پیگیری پیشرفت پروژه شیء گرا

با اجرای همزمان فعالیت های یک پروژه شیء گرا، شاخص های زمانی تعیین شده باید توسط مدیر پروژه کنترل شود. این شاخص ها و موارد مربوطه در ادامه ارائه شده است.

۱- شاخص های تکمیل شدن تحلیل شیء گرا

- همه کلاس ها و سلسله مراتب کلاس ها تعریف و بازبینی شده اند.
- صفات و عملیات کلاس های مرتبط با یک کلاس تعریف و بازبینی شده اند.
- روابط کلاس ها تعیین و بازبینی شده اند.
- یک مدل رفتاری ایجاد و بازبینی شده است.
- کلاس های قابل استفاده مجدد مشخص شده اند.

۲- شاخص‌های تکمیل شدن طراحی شیء گرا

- مجموعه‌ای از زیرسیستم‌ها تعریف و بازبینی شده است.
- کلاس‌ها به زیرسیستم‌ها تخصیص یافته و بازبینی شده است.
- تخصیص وظایف انجام و بازبینی شده است.
- مسئولیت‌ها و مشارکت‌ها مشخص شده‌اند.
- صفات و عملیات، طراحی و بازبینی شده‌اند.
- مدل تبادل پیغام ایجاد و بازبینی شده است.

۳- شاخص‌های تکمیل شدن برنامه‌نویسی شیء گرا

- هر یک از کلاس‌های جدید از مدل طراحی به کد تبدیل شده است.
- کلاس‌های استخراج‌شده (از کتابخانه) پیاده‌سازی شده‌اند.
- نمونه ساخته شده است.

۴- شاخص‌های انجام آزمون شیء گرا

- درستی و کامل بودن مدل تحلیل و طراحی شیء گرا مورد بازبینی قرار گرفته است.
- یک شبکه کلاس - مسئولیت - مشارکت توسعه یافته و مورد بازبینی قرار گرفته است.
- نمونه‌های آزمون طراحی شده‌اند و آزمون‌هایی در سطح هر کلاس اجرا شده‌اند.
- آزمون‌های خوشه‌ای کامل شده و کلاس‌ها بر اساس نمونه‌های آزمون، یکپارچه شده‌اند.
- آزمون‌های سطح سیستم کامل شده‌اند.

۱۸-۱۰ تحلیل ارزش حاصله

آیا روش کمی برای دستیابی به پیشرفت پروژه نرم‌افزاری که دارای زمان‌بندی است، وجود دارد؟ در واقع بله، این روش، تحلیل ارزش حاصله^۱ نام دارد.

برای تعیین مقدار ارزش حاصله، شاخص‌های زیر در طول اجرای پروژه محاسبه و برای کنترل پیشرفت پروژه مورد استفاده قرار می‌گیرد.

۱- **هزینه بودجه‌ای کار زمان‌بندی شده (BCWS)**^۲: این هزینه برای هر کار مشخص‌شده در زمان‌بندی تعیین می‌شود. در مدت برآورد، میزان کار (برحسب نفر- ساعت یا نفر- روز) برای هر فعالیت مهندسی نرم‌افزار برنامه‌ریزی می‌شود؛ بنابراین $BCWS_i$ هزینه فعالیت برنامه‌ریزی شده برای کار i است. به منظور تعیین میزان پیشرفت در نقطه‌ای از زمان‌بندی پروژه، مقدار $BCWS$ برابر با مجموع $BCWS_i$ ها برای تمام کارها در زمان t است:

$$BCWS = \sum_i BCWS_i$$

۲- مقادیر $BCWS_i$ برای تمام کارهای پروژه با هم جمع شده تا **بودجه تکمیل پروژه (BAC)**^۳ را مشخص کند.

$$BAC = \sum_k (BCWS_k) \quad \forall k$$

۳- **هزینه بودجه‌ای کار انجام‌شده (BCWP)**^۴ محاسبه می‌شود که برابر با مجموع مقادیر $BCWP_i$ برای تمام کارهایی است که تا آن نقطه از زمان (t) بر مبنای زمان‌بندی پروژه واقعاً کامل شده‌اند.

1 - Earn Value Analysis
2 - Budgeted Cost of Work Scheduled
3 - Budget At Completion
4 - Budgeted Cost of Work Performed

$$BCWP = \sum_i BCWP_i$$

Wilkins اشاره می‌کند که تفاوت بین BCWS و BCWP این است که اولی نشان‌دهنده بودجه فعالیت‌هایی است که برای تکمیل پروژه برنامه‌ریزی شده‌اند و دومی نشان‌دهنده بودجه فعالیت‌هایی است که به شکل واقعی کامل شده‌اند. با داشتن مقادیر BCWS، BAC و BCWP، شاخص‌های مهم پیشرفت پروژه به صورت زیر قابل محاسبه هستند:

- شاخص کارایی زمان‌بندی^۱

$$SPI = BCWP/BCWS$$

- شاخص واریانس زمان‌بندی^۲

$$SV = BCWP - BCWS$$

SPI نشان‌دهنده کارایی استفاده پروژه از منابع زمان‌بندی شده در پروژه است. مقدار SPI نزدیک به ۱ نشان‌دهنده اجرای کارآمد زمان‌بندی پروژه است. SV فقط نشان‌دهنده واریانس مطلق از زمان‌بندی برنامه‌ریزی شده است.

- شاخص درصد زمان‌بندی شده برای تکمیل پروژه

$$BCWS/BAC$$

نمایشی از درصد کاری است که باید تا زمان t کامل می‌شد.

- شاخص درصد تکمیل شدن پروژه

$$BCWP/BAC$$

نمایش کمی درصد تکمیل پروژه در نقطه‌ای از پروژه در زمان t است.

این امکان نیز وجود دارد که هزینه واقعی کار انجام‌شده (ACWP)^۳، نیز محاسبه شود. مقدار ACWP مجموع هزینه واقعی فعالیت‌های انجام‌شده برای کارهایی است که تا زمان t در زمان‌بندی پروژه کامل شده‌اند؛ بنابراین:

- شاخص کارایی هزینه^۴

$$CPI = BCWP/ACWP$$

- شاخص واریانس هزینه پروژه

$$CV = BCWP - ACWP$$

اگر مقدار CPI نزدیک به یک باشد، نشان‌دهنده قرار داشتن پروژه مطابق بودجه تعریف‌شده است. CV نمایشی مطلق از صرفه‌جویی‌های هزینه (در برابر هزینه‌های برنامه‌ریزی شده) در مرحله خاصی از پروژه است.

1 - Schedule performance index

2 - Schedule Variance

5- Actual Cost of Work Performed

4 - Cost Performance Index

فصل نوزدهم

اطمینان مرغوبیت نرم افزار

برخی نرم افزارنویسان همچنان بر این باورند که کیفیت نرم افزار چیزی است که پس از تولید برنامه باید نگران آن بود. در صورتی که اطمینان مرغوبیت نرم افزار (SQA)^۱ یک فعالیت پشتیبانی است که در سرتاسر فرایند نرم افزار اجرا می شود. SQA شامل موارد زیر است:

- روشی برای مدیریت کیفیت
- اطمینان از به کارگیری فن آوری مهندسی نرم افزار مؤثر (روش ها و ابزارها)
- بازبینی های فنی و رسمی در سرتاسر فرایند توسعه نرم افزار
- یک راهکار آزمون چند لایه ای
- کنترل مستندسازی نرم افزار و تغییرات اعمال شده در آن
- رویه ای برای حصول اطمینان از مطابقت محصول با استانداردهای توسعه نرم افزار (در صورت امکان)
- استفاده از راهکارهای اندازه گیری و گزارش دهی به مدیریت

۱-۱-۱۹ کیفیت چیست؟

در فرهنگ لغات کیفیت به عنوان "ویژگی یا صفتی از یک عنصر" تعریف می شود. به عنوان صفت یک شیء، کیفیت عبارت است از ویژگی های قابل اندازه گیری آن شیء. بر این اساس با دو نوع کیفیت مواجه هستیم:

کیفیت طراحی به ویژگی هایی اطلاق می شود که طراحی برای یک شیء مشخص می کند. پایه مواد اولیه، انحرافات و مشخصات کارایی در کیفیت طراحی مطرح هستند. اگر محصول طبق مشخصات ساخته شود، به موازات استفاده از مواد اولیه با پایه بالاتر، انحرافات محدودتر و با مشخص شدن سطوح بالاتری از کارایی، کیفیت طراحی محصول افزایش می یابد.

کیفیت تطبیق، حد پیروی از مشخصات طراحی در مراحل ساخت است. هرچه میزان مطابقت محصول با مشخصات طراحی بیشتر باشد، سطح کیفیت تطبیق بالاتر است.

در توسعه نرم افزار، کیفیت طراحی شامل خواسته ها، مشخصه ها و طراحی سیستم می شود. کیفیت تطبیق مسئله ای است که اساساً بر پیاده سازی تأکید دارد. اگر پیاده سازی از طراحی پیروی کرده و سیستم حاصل، خواسته ها و اهداف کارایی را برآورده سازد، کیفیت تطبیق بالا است.

۲-۱-۱۹ کنترل کیفیت

کنترل گوناگونی ممکن است با کنترل کیفیت یکسان باشد. ولی چگونه به کنترل کیفیت دست پیدا کنیم؟ کنترل کیفیت عبارت از یک سری بازرسی‌ها، بازمی‌بینی‌ها و آزمون‌هاست که در سرتاسر فرایند نرم‌افزاری به کار می‌روند، تا از برآورده شدن خواسته‌ها توسط محصول (نرم‌افزار) تولیدشده اطمینان حاصل شود. کنترل کیفیت شامل یک حلقه بازخورد به فرایندی است که محصول را ایجاد کرده است.

۳-۱-۱۹ تضمین کیفیت

تضمین کیفیت شامل بازرسی و گزارش وظایف مدیریت است. هدف تضمین کیفیت فراهم آوردن داده‌های مربوط به کیفیت محصول برای مدیریت و در نتیجه حصول اطمینان از این نکته است که کیفیت محصول، اهداف آن را برآورده می‌سازد. البته، اگر داده‌های فراهم‌شده از طریق تضمین کیفیت، مشکلاتی را معین کند، مدیریت باید با مشکلات مقابله و منابع لازم را برای حل مسایل کیفیتی به کار بندد.

۴-۱-۱۹ هزینه‌های کیفیت

هزینه‌های کیفیت، شامل کلیه هزینه‌های صرف‌شده برای دستیابی به کیفیت یا اجرای فعالیت‌های مرتبط با کیفیت است. هزینه‌های کیفیت را می‌توان به هزینه مربوط به پیشگیری، ارزیابی و شکست تقسیم کرد.

۱-۱۹ کیفیت نرم‌افزار

دلایل تفاوت کیفیت نرم‌افزار با کیفیت‌های سایر محصولات به شرح زیر است:

- عدم موجودیت فیزیکی نرم‌افزار
- عدم آگاهی از نیاز کاربر در ابتدای کار
- تغییر نیازهای کاربر در تمام مدت فرایند توسعه نرم‌افزار
- نرخ سریع تغییرات سخت‌افزار و نرم‌افزار
- توقعات زیاد کاربران به خصوص از نظر سازگاری

مجدداً یادآور می‌شویم که در توسعه نرم‌افزار، کیفیت طراحی شامل خواسته‌ها، مشخصه‌ها و طراحی سیستم می‌شود و کیفیت تطبیق بر پیاده‌سازی تأکید دارد. اگر پیاده‌سازی از طراحی پیروی کند و سیستم حاصل خواسته‌ها و اهداف کارایی را برآورده سازد، کیفیت تطبیق بالاست.

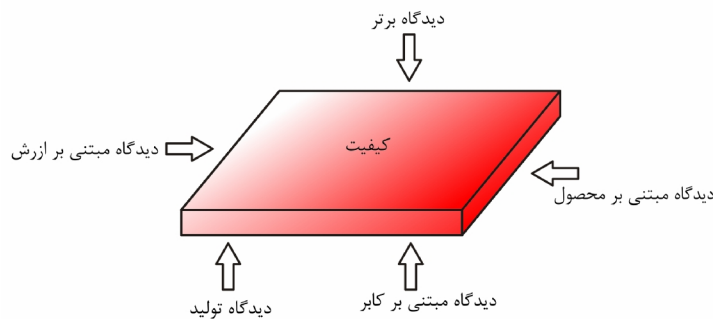
۲-۱۹ مدل‌های سلسله‌مراتبی کیفیت

مدل‌های مختلفی برای مقایسه کیفیت در موقعیت‌های مختلف چه کمی و چه کیفی وجود دارد. بیشتر این مدل‌ها طبیعتی سلسله‌مراتبی دارند. مدل سلسله‌مراتبی کیفیت نرم‌افزار بر پایه یک مجموعه از معیارهای کیفی استوار است که هر کدام از معیارها مجموعه‌ای از درجات و استانداردهای مرتبط به کیفیت را دارا هستند.

۱-۲-۱۹ دیدگاه‌های کیفیتی Garovin

کیفیت دارای ساختار چندبندی است و البته مطابق با تعدادی از دیدگاه‌ها یا افکار خاص طبقه‌بندی می‌شود. این دیدگاه‌ها اغلب متفاوت بوده و ممکن است با یکدیگر نیز اختلاف داشته باشند. هر یک از این دیدگاه‌ها تصویری ناقص، ناشی از یک مفهوم ویژه را القا می‌کنند (اختلافاتی که معمولاً در کیفیت نرم‌افزار بین کاربر و مشتری با طراح یا برنامه‌نویس پیش می‌آید).

با توجه به تقسیمات متفاوت و تداخل دیدگاه‌ها در کیفیت، Garovin (۱۹۸۴) پنج دیدگاه مختلف از کیفیت را مطابق شکل زیر پیشنهاد کرده است:



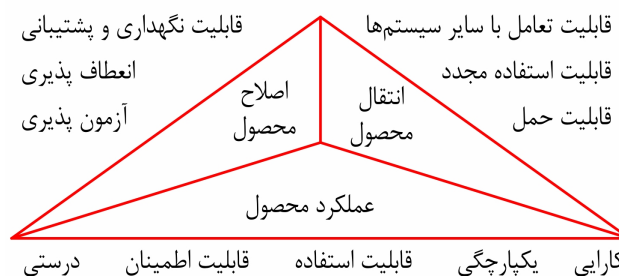
شکل ۱-۱۹ دیدگاه‌های کیفیت مطرح شده توسط Garovin

۱۹-۲-۲ مدل MacCall (۱۹۷۷-۱۹۸۰)

این مدل شکاف بین کاربران و توسعه‌دهندگان را کاهش داده و معیارهایی را انتخاب می‌کند تا دیدگاه‌های کاربران را به خوبی اولویت‌های توسعه‌دهندگان مشخص کند. مدل MacCall سه ناحیه کاری را در نرم‌افزار مد نظر دارد:

- عملکرد محصول،
- تجدید نظر و اصلاح محصول،
- انتقال محصول،

معیارهای کیفیت در مدل MacCall در شکل زیر نشان داده شده است.



شکل ۲-۱۹ معیارهای کیفیت مدل MacCall

۱۹-۲-۳ مدل FURPS

شرکت Hewlett-Packard هم مجموعه‌ای از فاکتورها را معرفی کرده است که آن را FURPS نامیده‌اند، ۵ معیار مدل FURPS، فاکتورهای اصلی کیفیت است.

- قابلیت عملکردی^۱

از طریق ارزیابی مجموعه خصوصیات و توانایی‌های برنامه، کلی بودن توابع ارائه شده و امنیت سراسر سیستم ارزیابی می‌شود.

- قابلیت استفاده^۲

از طریق بررسی فاکتورهای انسانی، زیباشناختی کلی، سازگاری و مستندسازی ارزیابی می‌شود.

- قابلیت اعتماد^۳

از طریق اندازه‌گیری تکرار (توالی) و شدت شکست، درستی و دقت نتایج خروجی، متوسط زمان شکست (MTTF)^۴، توانایی ترمیم و ارتقای شکست و قابلیت پیش‌بینی توسط نرم‌افزار ارزیابی می‌شود.

1 - Functionality
 2 - Usability
 3 - Reliability
 4 - Mean Time To Failure

• قابلیت اجرایی یا کارایی^۱

از طریق سرعت پردازش، زمان پاسخگویی، مصرف منابع، توان عملیاتی سیستم و کارایی ارزیابی می‌شود.

• قابلیت پشتیبانی^۲

ترکیبی است از توانایی گسترش برنامه^۳، تطبیق پذیری^۴، سرویس پذیری^۵، به علاوه قابلیت آزمون^۶، سازگاری^۷، قابلیت پیکربندی^۸، سهولت نصب سیستم و سهولت مکان‌یابی شکست‌ها.

۴-۲-۱۹ مدل Bohem (۱۹۷۸)

مدل Bohem برای تأمین مجموعه‌ای از "تعاریف مناسب و تفاوت‌های مشخصه‌های کیفیت نرم‌افزار" تعریف شده است. مدل از نوع سلسله‌مراتبی توسعه یافته است و معیارهای کیفیت آن به قسمت‌های جزئی تقسیم شده است.

۵-۲-۱۹ مدل Perry (۱۹۸۷)

مدل Perry ارتباط نهفته بین معیارها را به صورت مستقیم، معکوس و خنثی خلاصه می‌کند. در این مدل کنترل‌های مختلف مد نظر قرار می‌گیرد و از طریق ایجاد ماتریس ارتباطات بین آن‌ها به ارزیابی کیفیت می‌پردازد.

۳-۱۹ اطمینان مرغوبیت نرم‌افزار (SQA)

طبق تعریف، کیفیت نرم‌افزار عبارت است از: مطابقت با خواسته‌های عملیاتی، استانداردهای توسعه‌ای و ویژگی‌های ناگفته‌ای که از همه نرم‌افزارهای حرفه‌ای انتظار می‌رود. این تعریف بر سه نکته تأکید دارد:

- ۱- خواسته‌های نرم‌افزار مبنای سنجش کیفیت هستند.
- ۲- استانداردهای مشخص، مجموعه‌ای از معیارهای توسعه را تعیین می‌کنند که شیوه مهندسی نرم‌افزار را هدایت می‌کند.
- ۳- مجموعه‌ای از خواسته‌های ناگفته مانند سهولت به‌کارگیری، قابلیت نگهداری خوب و ... است.

۱-۳-۱۹ فعالیت‌های اطمینان مرغوبیت نرم‌افزار

اهداف

- فراهم آوردن داده‌های مربوط به کیفیت محصول برای، مدیریت
 - حصول اطمینان از این نکته که کیفیت محصول اهداف آن را برآورده می‌سازد.
- تضمین کیفیت نرم‌افزار از چند وظیفه مرتبط با دو گروه متفاوت تشکیل شده است.

۱- گروه مهندسان نرم‌افزار

گروه مهندسان با اعمال روش‌های فنی، اجرای بازبینی‌های فنی رسمی و اجرای آزمون‌های نرم‌افزاری برنامه‌ریزی شده، کیفیت نرم‌افزار را کنترل می‌کنند.

1 - Performance
2 - Supportability
3 - Extendibility
4 - Adaptability
5 - Serviceability
6 - Testability
7 - Compatibility
8 - Configurability

۲- گروه SQA

وظیفه گروه SQA برنامه‌ریزی تضمین کیفیت، نظارت، ثبت وقایع و گزارش‌دهی به مدیریت است. فعالیت‌های این گروه به صورت زیر انجام می‌گیرد:

- تهیه طرح SQA برای پروژه
- شرکت در جلسات توصیف فرایند پروژه نرم‌افزاری
- بازبینی فعالیت‌های مهندسی نرم‌افزار
- بازرسی محصولات کاری تولیدشده
- حصول اطمینان از مستندسازی انحرافات در فرایند مهندسی نرم‌افزار و مقابله با آن
- ثبت هرگونه عدم مطابقت و گزارش‌دهی به مدیریت ارشد

۱۹-۴-۲ طرح SQA

طرح SQA راهی را برای ایجاد تضمین کیفیت نرم‌افزار فراهم می‌کند. این طرح که توسط گروه SQA توسعه داده می‌شود، الگویی برای فعالیت‌های SQA در پروژه‌های نرم‌افزاری ارائه می‌دهد. گروه SQA یک طرح تضمین کیفیت نرم‌افزار برای پروژه که در آن موارد زیر مشخص می‌شود، تهیه می‌کند:

- ارزیابی‌هایی که باید انجام شود.
- بازرسی‌ها و بازبینی‌هایی که باید اجرا شوند.
- استانداردهایی که در پروژه لازم‌الاجرا هستند.
- روال‌هایی که برای گزارش و پیگیری خطاها باید تدوین شوند.
- مستنداتی که باید توسط گروه SQA تولید شود.
- نحوه و میزان ارائه بازخورد به تیم پروژه نرم‌افزاری و مدیریت پروژه باید تهیه شود.

۱۹-۴-۳ بازبینی نرم‌افزار

بازبینی‌های نرم‌افزار به مثابه فیلترهایی برای فرایند مهندسی نرم‌افزار عمل می‌کنند. یعنی در نقاط گوناگونی از توسعه نرم‌افزار اعمال می‌شوند و به کشف خطاها و نقایصی که قابل رفع باشند، کمک می‌کنند. بازبینی‌های نرم‌افزار به "خالص‌سازی" فعالیت‌های مهندسی نرم‌افزار یعنی تحلیل، طراحی، برنامه‌نویسی و آزمایش نرم‌افزار کمک می‌کنند.

۱۹-۵-۱ بازبینی فنی رسمی

بازبینی فنی رسمی (FTR)^۱ از دیدگاه تضمین کیفیت مؤثرترین فیلتر و ابزاری قوی برای بهبود بخشیدن به کیفیت نرم‌افزار است که شامل بررسی مقدماتی، بازرسی‌ها، بازبینی‌های نوبتی و چرخشی و ارزیابی فنی نرم‌افزار است.

۱۹-۵-۱ گزارشات بازبینی و ثبت فعالیت‌ها

گزارش خلاصه بازبینی باید پاسخگوی سه پرسش زیر باشد:

- چه چیزی مورد بازبینی قرار گرفته است؟
- چه کسانی آن را بازبینی کرده‌اند؟
- یافته‌ها و نتایج کدام‌اند؟

۶-۱۹ اندازه گیری کیفیت

۱-۶-۱۹ قابلیت اعتماد نرم افزار

قابلیت اعتماد نرم افزار^۱ به صورت احتمال عملکرد بدون شکست یک برنامه رایانه‌ای در محیطی مشخص برای یک زمان معین تعریف می‌شود. یعنی:

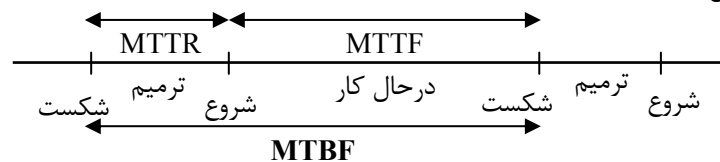
$$SR = 1 - P \text{ (شکست یک برنامه رایانه‌ای در محیط مشخص در زمان معین)}$$

اندازه گیری قابلیت اعتماد

در یک سیستم رایانه‌ای، یک معیار ساده قابلیت اعتماد، متوسط زمان بین شکست‌ها (MTBF)^۲ است که از رابطه زیر به دست می‌آید:

$$MTBF = MTTF + MTTR$$

MTTF و MTTR به ترتیب متوسط زمان تا وقوع خرابی^۳ و متوسط زمان ترمیم یا رفع خرابی^۴ است. بسیاری از محققین معتقدند که MTBF معیاری بسیار مفیدتر از کسر LOC/Errors یا FP/Errors است. به بیان ساده‌تر، کاربر نهایی متوجه اشکال است نه نرخ کلی خطا. چون خطاهای برنامه نرخ وقوع مشابهی ندارد، تعداد کلی خطاها نمایش اندکی از قابلیت اعتماد یک سیستم نرم‌افزاری است. رابطه بین MTTF، MTTR و MTBF در شکل زیر نشان داده شده است.



شکل ۳-۱۹ رابطه بین MTBF با MTTR و MTTF

۲-۶-۱۹ اندازه گیری قابلیت دسترسی

در دسترس بودن نرم افزار^۵ میزان احتمال عملیاتی بودن برنامه بر طبق نیازها در نقطه‌ای از زمان را نشان می‌دهد و به صورت زیر تعریف می‌شود:

$$SA = [MTTF / (MTTF + MTTR)] \times 100\%$$

معیار قابلیت اعتماد (MTBF) به یک اندازه به MTTF و MTTR حساس است. در صورتی که معیار در دسترس بودن بیشتر به MTTR حساس است، که نشان‌دهنده قابلیت نگهداری نرم افزار است.

۳-۶-۱۹ امنیت نرم افزار

امنیت نرم افزار یکی از فعالیت‌های تضمین کیفیت نرم افزار است که بر تشخیص خطرات بالقوه تأکید دارد. این خطرات ممکن است بر نرم افزار تأثیر منفی داشته و باعث شکست کل سیستم شوند. اگر خطرات در فرایند مهندسی نرم افزار زود تشخیص داده شوند، می‌توان اشکالات طراحی نرم افزار را تشخیص و نسبت به حذف یا کنترل خطرات بالقوه اقدام کرد.

اگرچه قابلیت اعتماد و امنیت نرم افزار تا حد زیادی به یکدیگر نزدیک هستند، درک این نکته که بین آن‌ها تفاوت‌های پنهان بسیاری وجود دارد مهم است. قابلیت اعتماد نرم افزار از تحلیل‌های آماری برای تعیین احتمال شکست نرم افزار استفاده می‌کند. به هر حال، وقوع شکست لزوماً خطرناک نیست. امنیت نرم افزار راه‌هایی را بررسی می‌کند که در آن‌ها شکست‌ها شرایطی را به وجود می‌آورند که باعث نگرانی می‌شوند. یعنی، شکست‌ها مستقل در نظر گرفته نمی‌شوند و در ارتباط با کل سیستم رایانه‌ای بررسی می‌شوند.

1 - Software Reliability
2 - Mean Time Between Failure
3 - Mean Time To Failure
4 - Mean Time To Repair
5 - Software Availability

فصل بیستم

مدیریت پیکربندی نرم افزار

تعریف Babich: هنر هماهنگ‌سازی توسعه نرم افزار برای به حداقل رساندن سردرگمی را مدیریت پیکربندی نرم افزار (SCM) می‌گویند. مدیریت پیکربندی هنر شناسایی، سازماندهی و کنترل تغییرات و اصلاحاتی است که باید در یک نرم افزار در حال توسعه توسط تیم مهندسی نرم افزار اعمال شوند. هدف به حداکثر رساندن بهره‌وری در کنار به حداقل رساندن اشتباهات است. مدیریت پیکربندی نرم افزار یک فعالیت پشتیبانی است که در سرتاسر فرایند نرم افزار قابل اجراست. چون امکان تغییر در هر زمانی وجود دارد بنابراین فعالیت‌های SCM باید به صورت زیر اجرا شوند:

- ۱- شناسایی تغییر
- ۲- کنترل تغییر
- ۳- حصول اطمینان از پیاده‌سازی مناسب تغییر
- ۴- تهیه و اعلام گزارش تغییر به ذینفعان پروژه

۱-۲۰ مدیریت پیکربندی نرم افزار

خروجی هر فرایند نرم افزاری، اطلاعاتی است که به سه گروه عمده قابل تقسیم است:

- ۱- برنامه‌های رایانه‌ای (چه در سطح گداهای منبع و چه اجرایی)
- ۲- مستنداتی که برنامه‌های رایانه‌ای را توصیف می‌کنند (چه در بُعد فنی و چه در بُعد کاربر)
- ۳- داده‌ها (که یا در داخل برنامه‌ها جا دارند و یا در خارج آن‌ها قرار گرفته‌اند)

تمام موارد بالا را ارقام پیکربندی نرم افزار گویند. قانون اول مهندسی سیستم‌ها می‌گوید: در هر کجای چرخه حیات سیستم باشید، سیستم تغییر کرده و تمایل به تغییر در سرتاسر چرخه حیات سیستم باقی است. بنابراین باید منشأ تغییرات را شناخت.

۱-۱-۲۰ منشأ تغییرات

- شرایط بازار یا شرایط تجاری جدید که تغییراتی در خواسته‌های محصول یا قواعد تجاری دیکته می‌کنند.

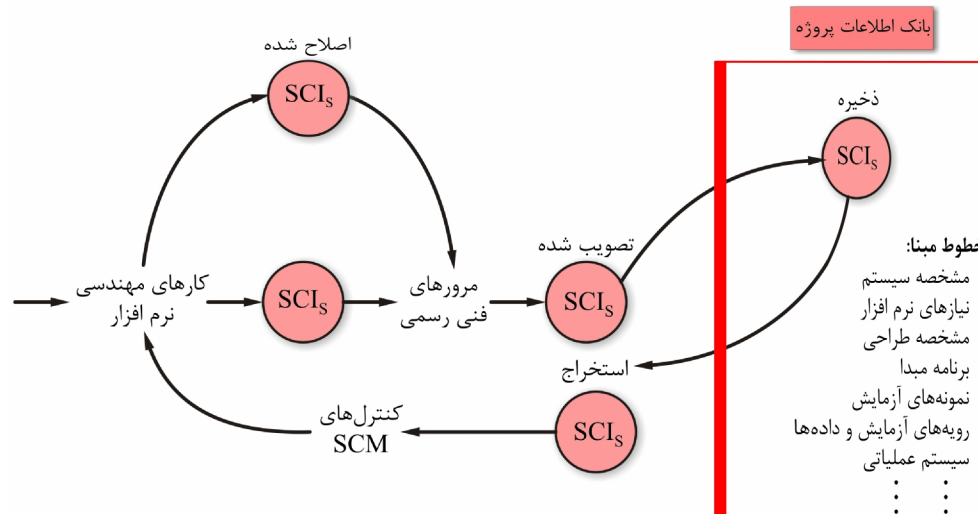
- نیازهای جدید مشتری که اصلاحاتی را در داده‌های تولیدشده توسط سیستم‌های اطلاعاتی، عملکرد تحویل‌شده توسط محصولات یا خدمات تحویل‌شده را توسط یک سیستم رایانه‌ای طلب می‌کنند.
- سازماندهی مجدد یا رشد / زوال تجاری که باعث تعریف مجدد محصول یا سیستم می‌شود.
- شرایط حدی بودجه یا زمان‌بندی که باعث تعریف مجدد محصول یا سیستم می‌شود.

۲-۲۰ خط مبنا

تعریف مبنا (استاندارد IEEE 61012-1940)^۱: مشخصه یا محصولی است که رسماً مورد بازبینی و توافق قرار گرفته و پس از آن به عنوان مبنایی برای توسعه بیشتر استفاده می‌شود و تنها از طریق روال‌های رسمی کنترل تغییرات، قابل تغییر است.

در رابطه با مهندسی نرم‌افزار، یک خط مبنا، علامت نشان‌دهنده‌ای در توسعه نرم‌افزار است که با تحویل یک یا چند قلم پیکربندی نرم‌افزار (SCIs)^۲ و تصویب آن‌ها از طریق بازبینی فنی رسمی شناسایی می‌شود. برای مثال، عنصر مشخصه طراحی عبارت است از تمام اسناد فنی که مستندسازی و مرور شده‌اند، خط‌هایی که پیدا و آشکار شده‌اند و... پس از اینکه تمام بخش‌های این مشخصه مرور، تصحیح و مورد تأیید قرار گرفت، آن‌گاه مشخصه طراحی به یک مبنا تبدیل می‌شود. تغییرات بیشتر در معماری برنامه (که در مشخصه طراحی مستند شده است)، می‌تواند فقط پس از ارزیابی و تأیید هر یک انجام شود.

اگرچه مبنا می‌تواند با هر سطحی از جزئیات تعریف شود ولی متداول‌ترین مبناهای نرم‌افزاری در شکل زیر نشان داده شده‌اند.



شکل ۲-۱۰ SCIs های مبنا و بانک اطلاعاتی پروژه

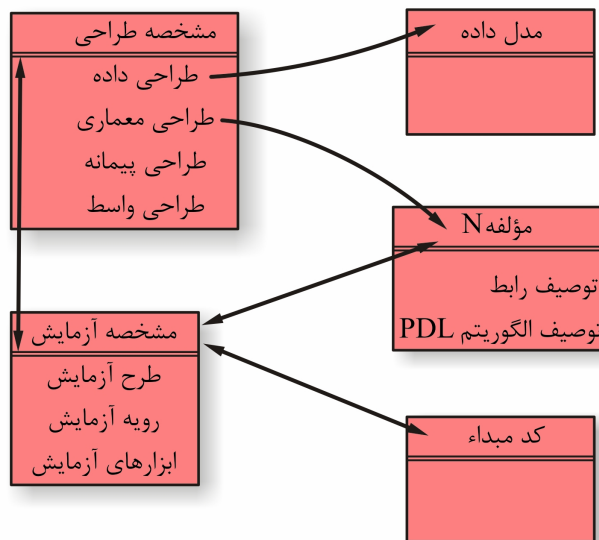
۳-۲۰ اقلام پیکربندی

یک قلم پیکربندی نرم‌افزار به صورت اطلاعاتی تعریف می‌شود که توسط بخشی از فرایند مهندسی نرم‌افزار ایجاد شده است. در نهایت، یک SCI می‌تواند به صورت یک بخش از یک مشخصه بزرگ یا یک جزء آزمایش در مجموعه بزرگی از اطلاعات آزمایش در نظر گرفته شود. به عبارت ساده‌تر، یک SCI می‌تواند یک سند، مجموعه کامل جزء آزمایش یا یک مؤلفه نام‌گذاری شده از برنامه (برای مثال، تابعی در ++C یا Package در Ada) باشد.

در عمل، SCI ها به‌گونه‌ای سازماندهی می‌شوند که اشیای پیکربندی را تشکیل می‌دهند و می‌توانند در بانک اطلاعاتی پروژه با یک نام ذخیره شوند. یک شیء پیکربندی دارای یک نام و صفت مربوطه است و به اشیای دیگر توسط روابط تعریف‌شده متصل می‌شود.

1 - BaseLine

2 - Software Configuration Items(SCIs)



شکل ۲۰-۲ اشیا پیکربندی نرم افزار

۴-۲۰ فرایند SCM

مدیریت پیکربندی نرم افزار عنصر مهمی از تضمین کیفیت نرم افزار است. مسئولیت اولیه آن کنترل تغییر است. به هر حال فرایند SCM مسئول شناسایی اقلام پیکربندی نرم افزار، کنترل نسخه، کنترل تغییرات، بارزسی پیکربندی و تهیه گزارش وضعیت است. هرگونه بحث در مورد SCM پرسش‌هایی را مطرح می‌سازد:

- سازمان چگونه نسخه‌های متعدد یک برنامه را قبل و بعد از تحویل نرم افزار به مشتری کنترل می‌کند؟
- سازمان چگونه تغییرات را قبل و بعد از ارائه نرم افزار به مشتری کنترل می‌کند؟
- چه کسی مسئول به تصویب رساندن تغییرات و اولویت‌بندی آنهاست؟
- چگونه می‌توان اطمینان پیدا کرد تغییرات به طور مناسب اعمال شده است؟
- چه راهکارهایی برای آگاه ساختن دیگران از اعمال تغییرات به کار می‌رود؟

این پرسش‌ها ما را به تعیین پنج وظیفه SCM رهنمون می‌سازند که در ادامه به تشریح آنها می‌پردازیم.

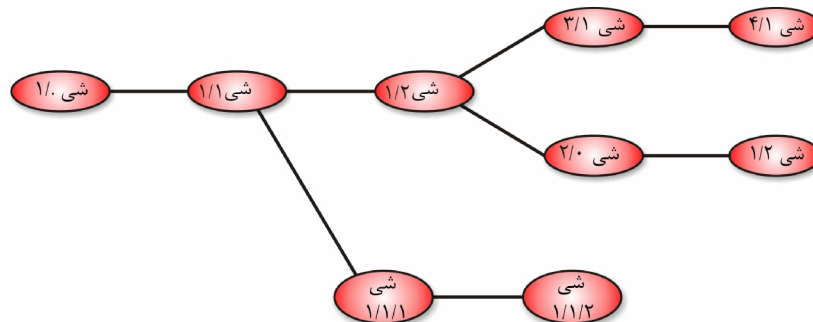
۱-۴-۲۰ شناسایی اشیا پیکربندی نرم افزار

برای کنترل و مدیریت اقلام پیکربندی نرم افزار، هر یک از اقلام باید به طور جداگانه نام‌گذاری و سپس با روش شیء‌گرا سازماندهی شوند. دو نوع از اشیا می‌توانند شناسایی شوند: **اشیای پایه و اشیای ترکیبی**. یک شیء پایه یک واحد متنی است که توسط مهندس نرم افزار در تحلیل، طراحی، کد نویسی، یا آزمایش ایجاد شده است. برای مثال، یک شیء پایه می‌تواند یک ویرایش از مشخصه نیازها، یک لیست از برنامه مبدأ یک مؤلفه، یا یک گروه از نمونه‌های آزمایش باشد. یک شیء ترکیبی، گروهی از اشیا پایه و اشیا مرکب دیگر است. با مراجعه به شکل صفحه قبل، مشخصه طراحی، یک شیء ترکیبی است. از نظر مفهومی، می‌تواند به صورت لیستی دارای نام از اشاره‌گرهایی باشد که اشیا پایه‌ای مانند مدل داده و مؤلفه N را مشخص می‌کنند.

۲-۴-۲۰ کنترل نسخه

مدیریت پیکربندی به کاربر امکان می‌دهد تا پیکربندی‌های جایگزین سیستم نرم افزاری را از طریق انتخاب نسخه مناسب اقلام مشخص کند. این عمل با همراه کردن صفات با هر نسخه از نرم افزار پشتیبانی شده و سپس یک پیکربندی با توصیف مجموعه مناسب صفات، مشخص و ساخته می‌شود.

یک نمایش از نسخ متعدد یک سیستم، مانند گراف تکاملی ارائه شده در شکل زیر است. هر گره در این گراف یک شیء مرکب (ترکیبی) است، یعنی نسخه کاملی از نرم افزار. هر نسخه از این نرم افزار مجموعه‌ای از SCIها است (کد مبدأ، اسناد و داده‌ها) که ترکیبی از تغییرات مختلف است.



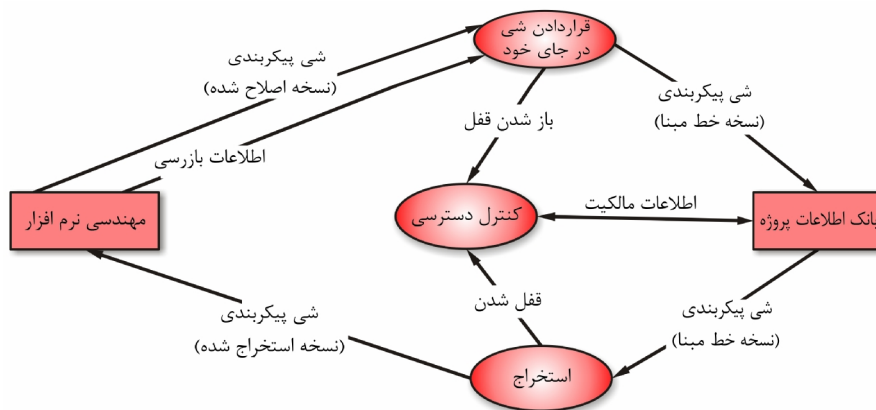
شکل ۲۰-۳ گراف تکامل ارقام پیکربندی نرم افزار

۲۰-۴ کنترل تغییر

مدیریت پیکربندی این امکان را به کاربر می‌دهد که پیکربندی‌های متفاوتی از سیستم‌های نرم افزاری را از طریق گزینش نسخه‌های مناسب، مشخص کند. این امر با نسبت دادن صفاتی به هر نسخه نرم افزار و سپس مشخص کردن پیکربندی [و بنا نهادن آن] با توصیف مجموعه‌ای از صفات مطلوب صورت می‌پذیرد.

فرایند استخراج و قرار دادن ارقام پیکربندی در بانک اطلاعاتی پروژه، دو عنصر مهم از کنترل تغییرات محسوب می‌شود.

- کنترل دسترسی مشخص می‌کند کدام یک از مهندسان نرم افزار اجازه دسترسی و اصلاح یک شیء پیکربندی خاص را دارند.
 - کنترل همزمانی مطمئن می‌سازد تغییرات موازی که توسط دو فرد مجزا انجام می‌شوند، نتایج یکدیگر را از بین نبرند.
- جریان کنترل‌های دسترسی و همزمانی به صورت شماتیک نیز در شکل زیر نشان داده شده است.

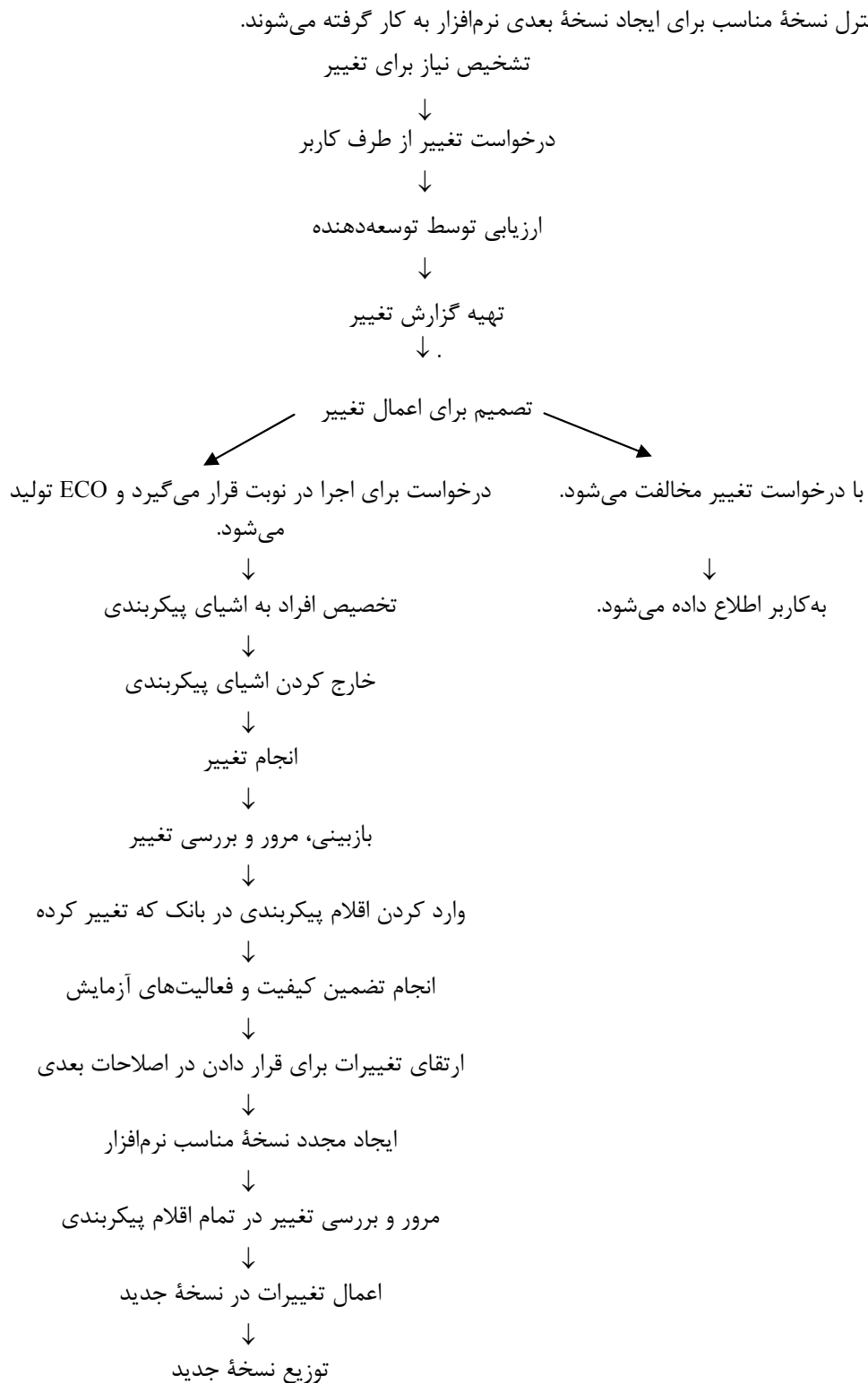


شکل ۲۰-۴ کنترل دسترسی و همزمانی

بر مبنای درخواست تغییر موافقت شده و تهیه دستور تغییر مهندسی (ECO)^۱، مهندس نرم افزار یک شیء پیکربندی را خارج می‌کند. یک تابع کنترل دسترسی مطمئن می‌سازد که این مهندس نرم افزار اجازه برداشتن این شیء را دارد و کنترل همزمانی، این شیء را در بانک اطلاعاتی پروژه قفل می‌کند به گونه‌ای که دیگر هیچ تغییری بر روی آن قابل انجام نباشد، تا زمانی که نسخه جاری خارج شده، مجدد در بانک پروژه جایگزین شود. یک کپی از شیء مبنا، در اختیار مهندس نرم افزار، قرار می‌گیرد و اصلاح می‌شود. بعد از انجام SQA مناسب و آزمایش، نسخه اصلاح شده این شیء وارد بانک می‌شود و شیء مبنای جدید از حالت قفل خارج می‌شود.

بنابراین:

- درخواست تغییر مورد ارزیابی قرار می‌گیرد تا شایستگی فنی، اثرات جانبی بالقوه، تأثیر کلی آن بر دیگر اشیای پیکربندی و عملکردهای سیستم سنجیده شود.
- نتایج این ارزیابی به صورت یک گزارش تغییر تهیه می‌شود که مورد استفادهٔ مسئول کنترل تغییر (CCA)^۱ قرار می‌گیرد، یعنی شخص یا گروهی که در مورد وضعیت و اولویت تغییر، تصمیم نهایی را اتخاذ می‌کنند.
- برای هر تغییر مصوب، یک سفارش تغییر مهندسی (ECO) تهیه می‌شود. ECO تغییری را که باید اعمال شود؛ شرایط حدی که باید رعایت شوند و ملاک‌ها و معیارهای بازبینی و بازرسی را توصیف می‌کند.
- شیء مبنا از بانک اطلاعاتی پروژه خارج می‌شود، تغییر روی آن اعمال می‌شود و فعالیت‌های SQA لازم به اجرا در می‌آیند.
- سپس شیء دوباره در جای خود در بانک اطلاعاتی قرار داده می‌شود.



شکل ۲۰-۵ فرایند کنترل تغییر

۴-۴-۲۰ بازرسی پیکربندی

شناسایی، کنترل نسخه و کنترل تغییرات به نرم افزارنویس کمک می کند تا نظم امور را حفظ کند. ولی، حتی مؤثرترین راهکارهای کنترل نیز یک تغییر را تا جایی پیکربندی می کنند که یک ECO تولید شود. چگونه می توان اطمینان یافت که تغییر به طور مناسب پیاده سازی شده است؟ این کار با انجام دو فعالیت زیر صورت می گیرد:

۱- بازبینی های فنی رسمی

۲- بازرسی پیکربندی نرم افزار

در حین بازبینی و بازرسی سؤالاتی مطرح و پاسخ آن ها داده می شود:

۱- آیا تغییر مشخص شده در ECO اعمال شده است؟ آیا اصلاحی صورت گرفته است؟

۲- آیا بازبینی فنی رسمی برای سنجش درستی کار اجرا شده است؟

۳- آیا فرایند نرم افزار دنبال و استانداردهای مهندسی نرم افزار به طور مناسب به اجرا درآمده اند؟

۴- آیا تغییر در SCI نمایان و مشهود است؟ آیا داده های تغییر و صاحب تغییر مشخص شده اند؟ آیا صفات شیء پیکربندی این تغییر را منعکس می کنند؟

۵- آیا روال های SCM برای شناسایی تغییر، ثبت آن و گزارش آن دنبال شده اند؟

۶- آیا همه SCI های مربوطه به طور مناسب به هنگام سازی شده اند؟

۵-۵-۲۰ تهیه گزارش وضعیت پیکربندی

گزارش پیکربندی (که گاه صورت وضعیت تغییرات نیز خوانده می شود) یک فعالیت SCM است که به سؤالات زیر پاسخ می دهد:

• چه اتفاقی رخ داده است؟

• چه کسی مسبب آن بوده است؟

• چه زمانی رخ داده است؟

• چه چیزهای دیگری از آن تأثیر خواهند پذیرفت؟

گزارش وضعیت پیکربندی، نقشی حیاتی در موفقیت پروژه های نرم افزاری بزرگ ایفا می کند که مهندسان نرم افزار باید به آن توجه کافی داشته باشند.