فصل اول

محصول (نرم‌افزار)

نرم‌افزار تبدیل به عنصر کلیدی در تکامل محصولات و سیستم‌های مبتنی بر رایانه شده است. در ۶۰ سال گذشته، نرم‌افزار از یک ابزار تحلیل اطلاعات و حل مسئله، به صنعتی مستقل تکامل یافته است. عملکرد گذشته در توسعهٔ نرم‌افزارهای کاربردی، یک‌سری مشکلات ایجاد کرده است که هنوز هم آثار آن قابل دیدن است.

نرم‌افزار به عاملی محدودکننده در ادامهٔ تکامل سیستم‌های رایانه‌ای تبدیل شده است. هر نرم‌افزار شامل مجموعه‌ای از برنامه‌ها، داده‌ها و مستندات است. هریک از این اجزا دربردارندهٔ پیکربندی خاص خود هستند که به عنوان بخشی از فرایند مهندسی نرم‌افزار به وجود می‌آیند. هدف مهندسی نرم‌افزار فراهم آوردن چارچوبی برای توسعه نرم‌افزارهایی است که دارای کیفیت مطلوب و بالایی باشند.

۱ـ۱ تعریف مهندسی نرم‌افزار

از دیدگاه Pressman: به مجموعه‌ای از فن‌آوری‌ها، روش‌ها و ابزارهای مبتنی بر اصول مهندسی که در توسعهٔ نرم‌افزار استفاده می‌شود، مهندسی نرم‌افزار گفته می‌شود.

از دیدگاه Sommerville: مهندسی نرم‌افزار در دههٔ ۱۹۶۰ در کنفرانسی با نام **بحران نرم‌افزار** مطرح می‌گردد.

مهندسی نرم‌افزار در ارتباط با توسعهٔ سیستم‌های نرم‌افزاری به شکل فعالیت گروهی است و از قواعد مهندسی استفاده می‌کند و شامل جنبه‌های فنی و غیر فنی است.

از دیدگاه Bauer: مهندسی نرم‌افزار عبارت از توسعه و استقرار نرم‌افزار با استفاده از اصول دقیق مهندسی برای دستیابی به نرم‌افزاری با صرفهٔ اقتصادی است دو شرط **قابل اعتماد**[1] و **کارا**[2] را دارد.

توجه: مهندسی نرم‌افزار هم‌ارز با برنامه‌نویسی نیست، بلکه برنامه‌نویسی فقط جزئی از مهندسی نرم‌افزار است.

1- Reliable
2- Efficient

۱ـ۲ مشکلات کنونی نرم‌افزار

- عدم تطابق نرم‌افزار با نیازهای واقعی مشتری و بازار (تغییرات)
- عدم امکان رقابت و استفاده کارای نرم‌افزار از سخت‌افزار با توجه به پیشرفت سریع سخت‌افزار (فن‌آوری)
- عمومی شدن کاربرد رایانه در جامعه و عدم امکان تغییر و تعویض نرم‌افزار در مدت‌زمان کوتاه (کاربرد لحظه‌ای)
- فشار عمومی و زیاد برای تولید نرم‌افزاری قابل اعتماد و دارای کیفیت مناسب (آزمون نرم‌افزار)
- قدرت محدود در پشتیبانی و نگهداری نرم‌افزارهای موجود (قابلیت توسعه)

۱ـ۳ تکامل نرم‌افزار و طرح سؤال در برابر مشکلات

در حال حاضر برنامه‌نویس گذشته جای خود را به تیمی از کارشناسان و مهندسین نرم‌افزار داده است که هر یک در بخشی از فن‌آوری 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 توسعهٔ نرم‌افزارطولانی است؟**

جواب: عامل مهم در طولانی شدن توسعهٔ نرم‌افزارها، عدم امکان اندازه‌گیری مستقیم و عدم استفاده از مؤلفه‌های آماده و آزمایش‌شده در پیاده‌سازی است. راهکار استفاده از روش توسعهٔ مبتنی بر مؤلفه[1] (CBD) تاحدی می‌تواند در رفع این مشکل کمک کند.

- **چرا هزینهٔ توسعهٔ نرم‌افزار بالا است؟**

جواب: به علت تغییرات محیطی، فنی و طولانی شدن زمان توسعهٔ نرم‌افزار، هزینه‌ها افزایش می‌یابد.

- **چگونه می‌توان خطاهای نرم‌افزار را پیش از تحویل شناسایی کرد؟**

جواب: در مراحل تحلیل و طراحی نرم‌افزار، بازبینی‌های مستمر انجام و کد تولید شده، به حد کفایت مورد آزمایش قرار گیرد.

- **چرا در اندازه‌گیری پیشرفت پروژهٔ نرم‌افزاری مشکل وجود دارد؟**

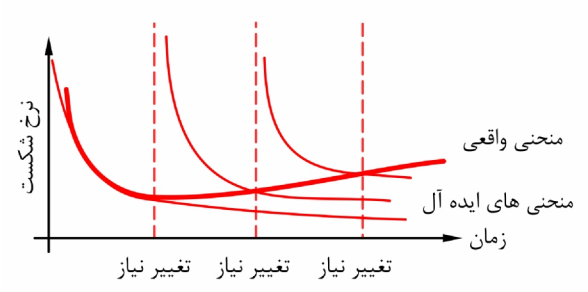
جواب: چون معیارهای دقیق و خاصی برای سنجش نرم‌افزار وجود ندارد و این معیارها به پارامترهای بسیاری از تحلیل، طراحی و پیاده‌سازی وابسته‌اند.

۱ـ۴ ویژگی‌های نرم‌افزار

- نرم‌افزار یک محصول منطقی است و یک محصول فیزیکی نیست.
- نرم‌افزارها بیشتر بر اساس نیاز مشتریان ساخته می‌شوند.
- نرم‌افزار دورانداختنی نیست.

در شکل ۱ به‌وضوح دیده می‌شود که منحنی عمر مفید نرم‌افزار 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.

1- Component Base Development



شکل ۱ـ۱ : منحنی‌های شکست نرم‌افزار

۱ـ۵ کاربردهای نرم‌افزار

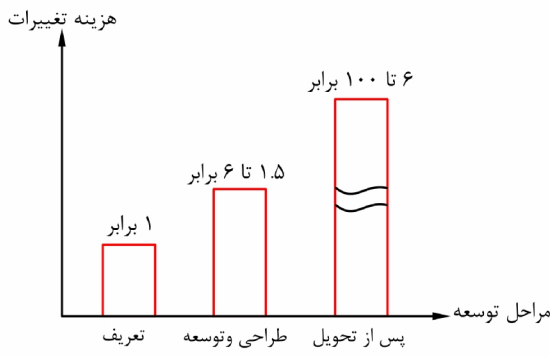
محتوای اطلاعات و قطعیت اطلاعاتی در تعیین ماهیت کاربردی یک نرم‌افزار مطرح هستند. منظور از محتوا معنی و شکل اطلاعات ورودی و خروجی و قطعیت اطلاعاتی به معنای قابلیت پیش‌بینی ترتیب و زمان‌بندی اطلاعات است.

با این وجود تعیین گروه‌های کلی بامعنی برای کاربردهای نرم‌افزاری تاحدی دشوار است. با پیچیده‌تر شدن نرم‌افزار، مرزهای صریح و روشن از بین می‌روند. به‌هرحال می‌توان زمینه‌های زیر را به عنوان گروه‌های کاربردی مشخص کرد:

- نرم‌افزارهای سیستمی
- نرم‌افزارهای زمان حقیقی
- نرم‌افزارهای تجاری
- نرم‌افزارهای مهندسی و علمی
- نرم‌افزارهای توکار و تعبیه شده
- نرم‌افزارهای مبتنی بر رایانه‌های شخصی
- نرم‌افزارهای مبتنی بر وب
- نرم‌افزارهای هوش مصنوعی و شبکه‌های عصبی و ...

۱ـ۶ هزینه‌های ناشی از تغییرات در چرخهٔ توسعهٔ نرم‌افزار

برآوردهای زیر از تعداد زیادی نرم‌افزار در چرخهٔ توسعهٔ نرم‌افزار انجام شده است که نشان‌دهندهٔ شدت افزایش هزینه تغییرات پس از تحویل نرم‌افزار است.



شکل ۱ـ۲ هزینهٔ تغییرات در چرخهٔ توسعهٔ نرم‌افزار

فصل دوم

فرآیند توسعه نرم‌افزار

مهندسی نرم‌افزار یک *فناوری لایه‌ای* است که مبتنی بر لایهٔ کیفیت است. مدیریت کیفیت به ارتقای مداوم فرایند توسعه کمک می‌کند، به طوری که همواره بهبود مستمر در فرایند مهندسی نرم‌افزار انجام شود. اساس مهندسی نرم‌افزار *لایه‌های فرایند* آن است. فرایند مهندسی نرم‌افزار همراه با لایه‌های فن‌آوری (روش‌ها و ابزارها) باعث توسعهٔ تدریجی نرم‌افزارهای رایانه‌ای می‌شوند.

۲ـ۱ فرایند نرم‌افزار چیست؟

چارچوبی برای کارها و فعالیت‌های مورد نیاز برای توسعهٔ نرم‌افزاری با کیفیت بالا را فرایند نامند.

۲ـ۲ ضوابط ارزیابی نرم‌افزار

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

۲ـ۲ـ۱ عوامل خارجی (کاربران)

کسانی که از نرم‌افزار فقط استفاده می‌کنند و در فرایند توسعهٔ آن نقشی ندارند. معیارهای ارزیابی نرم‌افزار از نظر این افراد عبارت‌اند از:

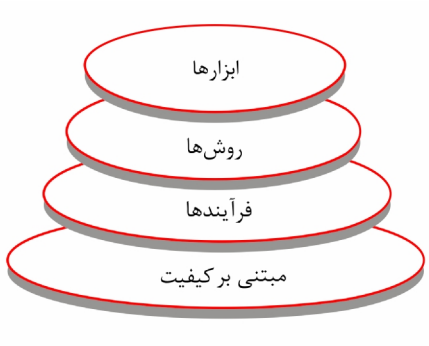
- صحت برنامه[1]
- استحکام[2] از دید کاربر
- قابلیت توسعه[3]
- قابلیت مصرف مجدد[4]
- سازگاری[5]
- قابلیت حمل[6]
- کارایی[1]

1 - Correctness
2 - Robustness
3 - Extendability
4 - Reusability
5 - Compatibility
6 - Portability

۲ـ۲ـ۲ عوامل خارجی (کاربران)

عوامل داخلی به توسعه‌دهندگان نرم‌افزار اطلاق می‌شود. مهندسین نرم‌افزار از دیدگاه فنی به ارزیابی نرم‌افزار می‌پردازند و به دنبال ارائهٔ پاسخ 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 ۲ـ۱ لایه‌های مهندسی نرم‌افزار

۲ـ۳ مراحل فرایند مهندسی نرم‌افزار

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

۲ـ۳ـ۱ تعریف

فعالیت‌های مرحلهٔ تعریف

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

- مهندسی سیستم یا اطلاعات[۶]
- برنامه‌ریزی توسعه پروژهٔ نرم‌افزاری[۷]
- تحلیل و مدل‌سازی نیازها[۸]

۲ـ۳ـ۲ توسعه

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

فعالیت‌های مرحلهٔ 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 - Requirments Analysis and Modeling
9 - Software Design
10 - Code Generation
11 - Testing

۲ـ۳ـ۳ نگهداری

- **نگهداری اصلاحی**[1]**:** اشکالات در چرخهٔ زندگی پروژه، شناسایی، بررسی و رفع می‌شود (رفع خطا).
- **نگهداری تطبیقی**[2]**:** تغییر به مرور زمان با توجه به نیازهای محیطی و کاربر صورت می‌گیرد (تطبیق با نیازهای جاری).
- **نگهداری ارتقایی**[3]**:** ارتقای نرم‌افزار پس از راه‌اندازی که توسط کاربر درخواست می‌شود (افزایش امکانات).
- **نگهداری پیشگیرانه**[4]**:** ایجاد تغییراتی در برنامه‌ها که منجر به ارتقا و بهبود عملکرد نرم‌افزار می‌شود (آینده‌نگری).

۲ـ۳ـ۴ فعالیت‌های پشتیبانی فرایند مهندسی نرم‌افزار

فعالیت‌های چتری یا پشتیبانی[5] در تمامی مراحل زیر انجام می‌گیرد:

۱. کنترل و نظارت بر برنامه‌ریزی پروژه[6]
۲. بازنگری‌های مستمر فنی رسمی[7]
۳. اطمینان مرغوبیت نرم‌افزار[8]
۴. مدیریت پیکربندی نرم‌افزار[9]
۵. تهیه و تدوین مستندات فنی[10]
۶. اندازه‌گیری نرم‌افزار با مقیاس‌های اندازه‌گیری[11]
۷. مدیریت ریسک نرم‌افزار[12]

۲ـ۴ مدل بلوغ توانایی (CMM)[13]

مؤسسهٔ مهندسی نرم‌افزار (SEI)[14] آمریکا مدل جامعی ارائه داده است که مجموعه‌ای از توانایی‌های مهندسی نرم‌افزار برای دسترسی سازمان‌ها به سطوح مختلف بلوغ فرایند در آن پیش‌بینی شده است. این مدل تعیین‌کنندهٔ میزان مؤثر بودن فعالیت‌های مهندسی نرم‌افزار در یک سازمان نرم‌افزاری بوده و دارای پنج سطح تکامل فرایندی است.

سطح ۱: اولیه[15]

فرایند نرم‌افزار به صورت موقتی و حتی بعضی اوقات بسیار درهم و برهم توصیف شده است. چند فرایند ساده تعریف می‌شود و موفقیت به تلاش‌های فردی بستگی دارد.

از نظر تضمین کیفی و مدیریت پروژه، وظیفه خاصی وجود ندارد. در این سطح تیم پروژه می‌تواند برای توسعه و ارائه نرم‌افزار هر راهی انتخاب کند. روش‌ها، استانداردها و رویه‌هایی که از کیفیت خوب تا خیلی ضعیف دارند، می‌توانند 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[1]

فرایندهای اولیه مدیریت پروژه برای مشخص کردن هزینه، زمان‌بندی و کارایی، در این سطح صورت می‌گیرد. این سطح نمایانگر این حقیقت است که ارائه‌دهنده نرم‌افزار، فعالیت‌های خاصی را همچون گزارش تکمیل کار و گزارش زمان‌بندی و فعالیت‌های انجام‌شده را تعریف کرده است.

سطح ۳: تعریف‌شده[2]

فرایند نرم‌افزار برای مدیریت فعالیت‌های مهندسی در طول فرایند توسعهٔ نرم‌افزار در سازمان ثبت‌شده، استاندارد و منسجم است. این سطح شامل مشخصه‌های تعریف‌شده در سطح ۲ نیز است. این سطح نمایانگر این است که ارائه‌دهندهٔ نرم‌افزار، فرایندهای فنی و مدیریتی را تعریف کرده است. این سطح از سطوحی است که اکثر تولیدکنندگان نرم‌افزار از طریق استانداردهایی مثل ایزو ۹۰۰۱ آن را مد نظر دارند.

سطح ۴: مدیریت‌شده[3]

اقدامات دقیق صورت‌گرفته در فرایند نرم‌افزار و کیفیت محصول، همگی کنترل می‌شوند. فرایند نرم‌افزار و محصولات از لحاظ کمّی شناسایی شده و با استفاده از اقداماتی دقیق کنترل می‌شوند. این سطح دربرگیرندهٔ مفهوم اندازه‌گیری و استفاده از معیارهای سنجش برای فرایند توسعهٔ نرم‌افزار و محصول نرم‌افزاری است. این سطح شامل تمام مشخصه‌های تعریف‌شده در سطح ۳ نیز است.

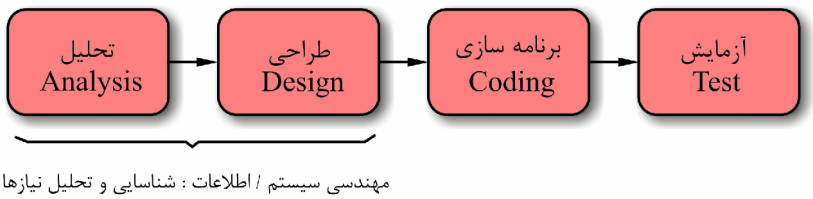
سطح ۵: بهینه‌سازی[4]

بهبود مکرر فرایند با توجه به بازخورد کمّی فرایند و از طریق آزمون ایده‌های نوآور و فن‌آوری‌های نوین و بروز، مقدور است. این سطح شامل تمام مشخصه‌های تعریف‌شده برای سطح ۴ است.

۲ـ۵ مدل‌های پایه در فرایند توسعهٔ نرم‌افزار

۲ـ۵ـ۱ مدل ترتیبی خطی

این مدل گاهی *مدل پایه* نیز نامیده می‌شود. مدل خطی ترتیبی[5] یک‌سری فعالیت متوالی برای توسعهٔ نرم‌افزار پیشنهاد می‌کند که از سطح تحلیل آغاز و به آزمایش نرم‌افزار ختم می‌شود.



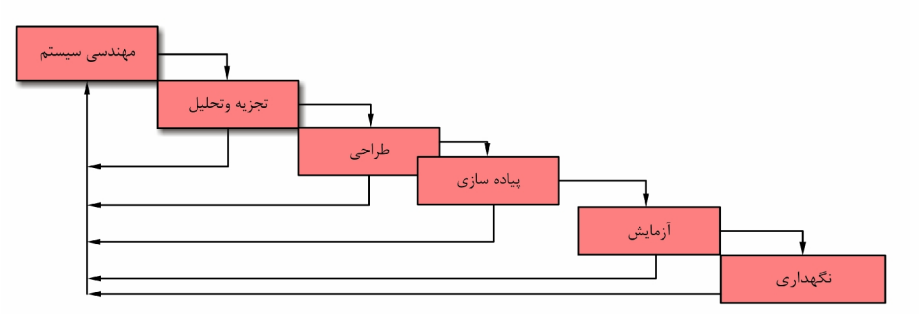
شکل ۲ـ۲ مدل ترتیبی خطی

این مدل برای پروژه‌های کوچک کارا است زیرا در آن تعریف صورت مسئله به راحتی انجام می‌گیرد.

۲ـ۵ـ۲ مدل آبشاری

مدل آبشاری[6] نیز اغلب با نام مدل دورهٔ زندگی کلاسیک معرفی می‌شود. در این مدل فعالیت توسعه نرم‌افزار از مهندسی سیستم شروع و به نگهداری نرم‌افزار ختم می‌شود.

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



شکل ۲ـ۳ مدل آبشاری

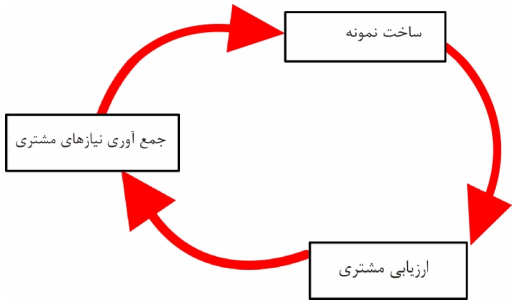
مدل آبشاری ساده است. در پروژه‌های بزرگ، مدت‌زمان انجام پروژه طولانی است و بنابراین نیازها تغییر می‌یابد؛ ازاین‌رو بدیهی است از این مدل در پروژه‌های بزرگ استفاده نشود.

مشکلات مدل

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

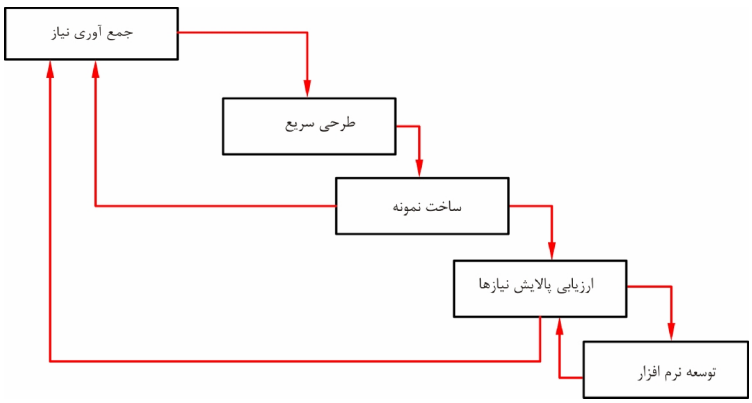
۲ـ۵ـ۳ مدل نمونه‌سازی

بخش‌های مختلف سیستم به صورت مرحله‌ای شناسایی و نمونه‌سازی می‌شود. مدل نمونه‌سازی[1]، نوعی تولید مجدد محسوب می‌شود (دوباره‌کاری). در حالت ایده‌آل، نمونه‌سازی روشی برای مشخص کردن نیازهای نرم‌افزار محسوب می‌شود.



شکل ۲ـ۴ روش نمونه‌سازی

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



شکل ۲ـ۵ نمونه‌سازی در مدل آبشاری

1 -Prototyping

مزیت

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

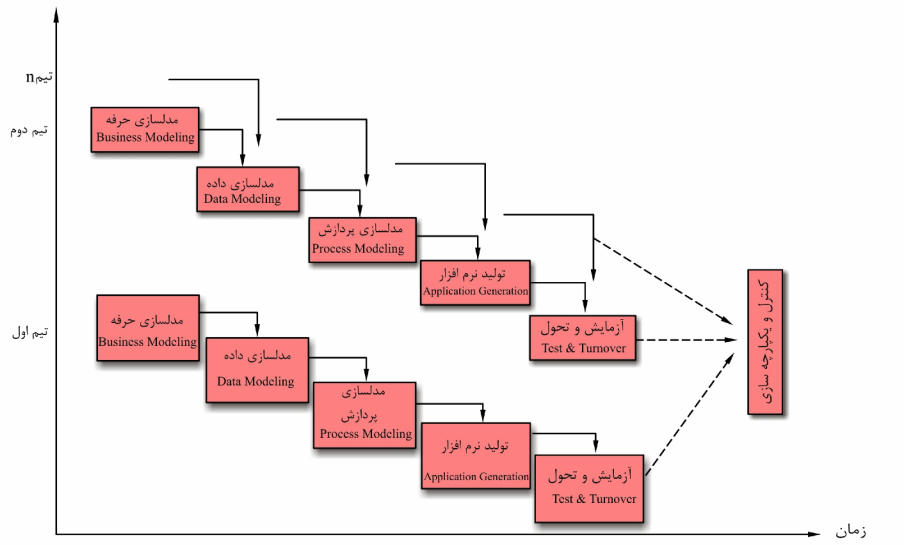
عیب

- مشتری، کل نرم‌افزار را نمی‌بیند و ملاحظات کلی از نرم‌افزار را نمی‌تواند داشته باشد (مثل قابلیت نگهداری).
- تغییر زیاد ممکن است باب میل مشتری نباشد (مانند نمونه‌های ساخته‌شدهٔ دورانداختنی).
- سرعت انجام کار باعث بروز اثرات جانبی می‌شود (مثل تغییر و محدودیت‌های سخت‌افزاری یا نرم‌افزاری).

۲ـ۵ـ۴ مدل توسعه سریع کاربرد (RAD)

در این مدل تأکید بر توسعه سریع نرم‌افزار[1] در مدت‌زمان 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 (پیمانه‌های) مختلف شکسته می‌شود و تیم‌های مختلف، همزمان روی هر یک از آن‌ها فعالیت می‌کنند و درنهایت نتیجهٔ کار با هم ترکیب و یکپارچه می‌شود.



شکل ۲ـ۶ مدل توسعه سریع کاربرد

مزیت

- زمان توسعه و تولید نرم‌افزار کوتاه است.
- تأکید بر قابلیت استفادهٔ مجدد مؤلفه‌های نرم‌افزاری است.

عیب

- مدیریت تیم‌ها و هماهنگی آن‌ها خیلی مشکل است.
- در سیستم‌هایی می‌توان این کار را کرد که قابلیت پیمانه‌ای[2] وجود داشته باشد که اغلب مواقع این طور نیست.
- در پروژه‌های بزرگ نیاز به افراد متخصص زیادی است.
- عدم اطمینان از تعهد کاری و پایبندی افراد در تیم‌های پروژه به عنوان ریسک، همواره وجود دارد.

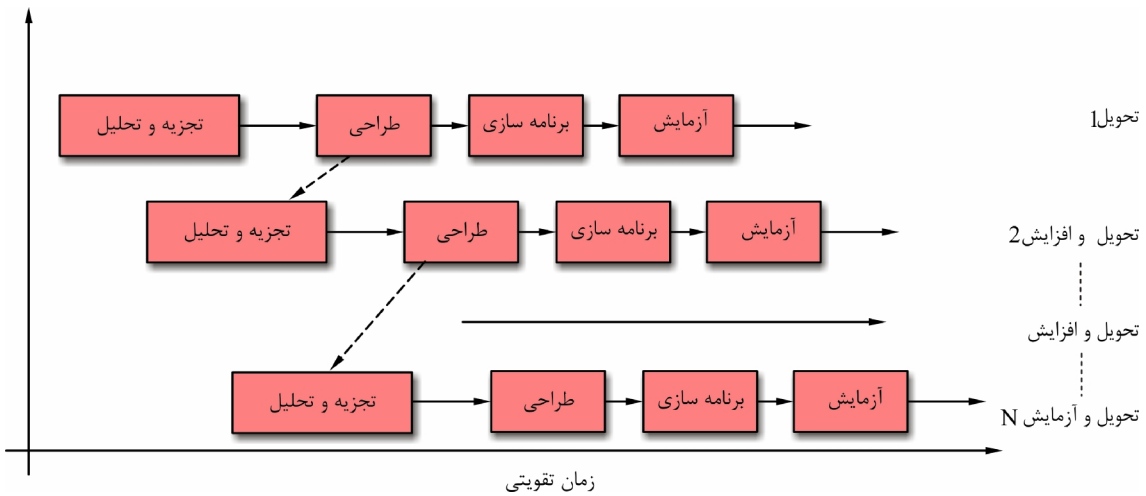
1 - Rapid Application Development
2 - Modularity

۲ـ۶ مدل‌های تکاملی فرایند توسعهٔ نرم‌افزار

ایده گسترش و بلوغ نرم‌افزار مانند سایر سیستم‌های پیچیده همواره مطرح است. با رشد ابزارها و فن‌آوری مربوطه، فرایندهای توسعهٔ نرم‌افزار نیز تکامل می‌یابند. در این بخش مدل‌های تکاملی فرایند توسعه نرم‌افزار مورد بررسی قرار می‌گیرند. مدل‌های تکاملی ماهیتاً تکراری هستند، یعنی مهندسین نرم‌افزار را قادر می‌کند تا به صورت افزایشی نسخه‌های کامل‌تری از نرم‌افزار را توسعه و عرضه کنند.

۲ـ۶ـ۱ مدل افزایشی

در مدل افزایشی[1] ابتدا محصول پایه‌ای تولید می‌شود و سپس در هر مرحله، نسخه‌ای از نرم‌افزار به مشتری ارائه می‌شود که امکانات بیشتری نسبت به نسخه قبلی دارد.



شکل ۲ـ۷ مدل افزایشی

مزیت

- برعکس روش نمونه سازی، نسخه‌های نرم‌افزار دور ریخته نمی‌شود.
- افزودن امکانات جدید به نرم‌افزار، ضمن به‌هنگام‌سازی آن، هزینهٔ کمتری دربردارد.

عیب

- نیاز به نیروی انسانی متخصص و یا سخت‌افزار کافی دارد.

۲ـ۶ـ۲ مدل حلزونی

مدل حلزونی[2] (معروف به مدل Bohem) روش خطی را با فلسفهٔ نمونه‌سازی ترکیب می‌کند که در پروژه‌های بزرگ مورد استفاده است. در این مدل فعالیت‌های مربوط به مهندسی نرم‌افزار به شش زمینه کاری تقسیم می‌شود:

ارتباط با مشتری[3]: قراردادها و کل توافقات انجام گرفته و اطلاعات کلان و کلی از پروژه جمع‌آوری می‌شود.

برنامه‌ریزی[4]: بر اساس نتایج به‌دست‌آمده از شناخت کلان، اعضای تیم شناسایی و تیم‌های کاری تشکیل و برای هر تیم، تخصیص کار صورت می‌گیرد و برنامهٔ کنترل و نظارت پروژه تدوین می‌شود.

تحلیل ریسک[5]: پیش‌بینی مخاطرات غیر منتظره مثل تغییرات فن‌آوری سخت‌افزار و نرم‌افزار، عدم تعهد کارفرما یا نیروی انسانی، تغییرات مدیریتی، تغییر خواستهٔ مشتری (عوامل محیطی) و ... انجام می‌شود.

مهندسی و طراحی[1]: فعالیت‌های مهندسی مورد نیاز برای طراحی و ساخت نمونه‌هایی از برنامهٔ کاربردی انجام می‌گیرد.

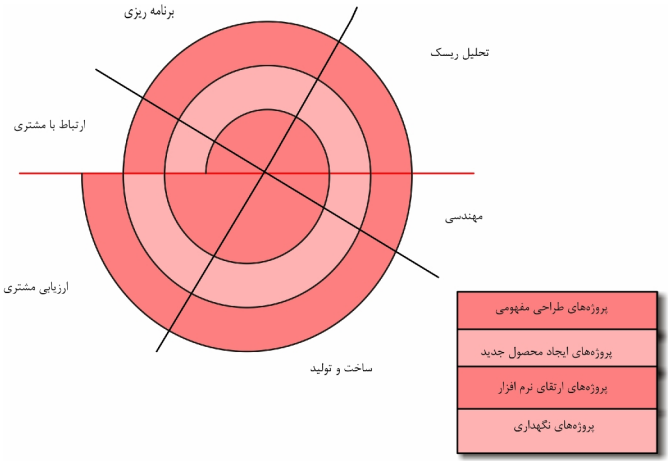
---

1 - Incremental Model
2 - Spiral Model
3 - Customer Communication
4 - Planning
5 - Risk Analysis

ساخت و تحویل[۲]: برنامه‌نویسی، آزمون، کنترل کیفیت، نصب و آموزش از عمدهٔ 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 نرم‌افزار 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.

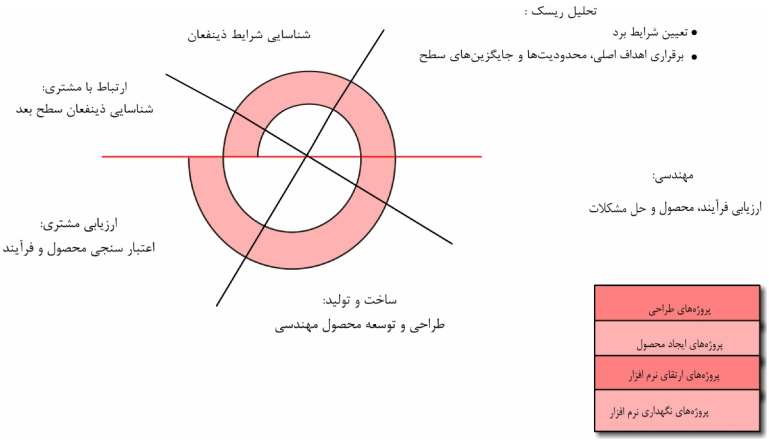


شکل ۲ـ۸ مدل حلزونی (مارپیچی)

۲ـ۶ـ۳ مدل حلزونی WinWin

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

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

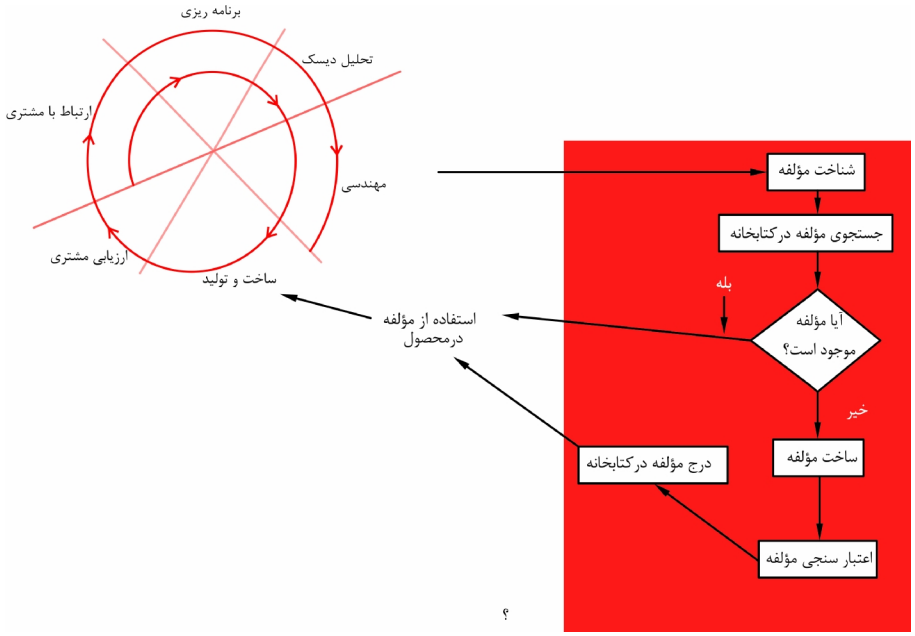


شکل ۲ـ۹ مدل حلزونیWinWin

1 - Engineering and Design
2 - Construction and Release
3 - Customer Evaluation

۲ـ۶ـ۴ مدل مونتاژ مؤلفه‌ها[1]

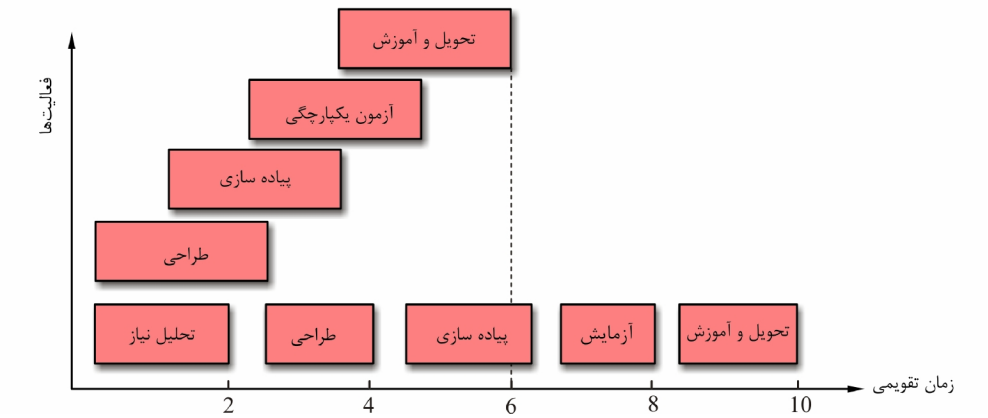
اجزای سیستم به صورت مؤلفه‌هایی در نظر گرفته می‌شوند که از قابلیت ساخت، آزمون و استفادهٔ مناسبی برخوردار هستند. هر مدل یا روشی در توسعهٔ نرم‌افزار در اینجا استفاده شود، فرقی نمی‌کند. در این مدل هدف و تأکید بر قابلیت استفاده مجدد از مؤلفه‌هاست.



شکل ۲ـ۱۰ مدل توسعهٔ مبتنی بر مؤلفه

۲ـ۶ـ۵ مدل توسعهٔ هم‌روند

در مدل توسعه هم‌روند[2] فعالیت‌ها به منظور کاهش زمان ایجاد نرم‌افزار به طور موازی انجام می‌گیرند.



شکل ۲ـ۱۱ مدل توسعهٔ هم‌روند

در حالت موازی، زمان کاهش می‌یابد ولی ممکن است بعضی پارامترهای دیگر مثل دوباره‌کاری و ... افزایش یابد. در این حالت مدیریت پروژهٔ نرم‌افزاری پیچیدگی زیادی دارد.

1 - Component Assembly Model
2 - Concurrent Development Model

۲ـ۶ـ۶ مدل روش‌های رسمی[1]

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

مزایا	معایب
✓ بدیع	✓ وقت‌گیر و پرهزینه
✓ سازگار	✓ کاربرد محدود
✓ بدون ابهام	✓ نیاز به آموزش جامع
✓ کاربرد در سیستم‌های حساس	✓ مشکل ارتباط با مشتری
✓ و ...	✓ و ...

۲ـ۶ـ۷ ابزارهای نسل چهارم[2]

شامل محدودهٔ وسیعی از ابزارهای نرم‌افزاری است که قادر به تعیین و مشخص کردن بعضی از خصوصیات و ویژگی‌های نرم‌افزار در سطح بالاست و می‌تواند حتی کُدی 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 استفادهٔ 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.

۲ـ۶ـ۸ فرایند مدل‌سازی یکنواخت[3]

در اواخر دهه ۹۰ میلادی به منظور ایجاد یک بستر یکسان برای انجام فرایندهای مهندسی نرم‌افزار چارچوبی توسط شرکت Rational Rose تحت عنوان RUP[4] ارائه شد که در آن چهار فاز اصلی و نُه 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[5] 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 ۲ـ۱۲ 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

۳ـ۱ تعریف مهندسی سیستم

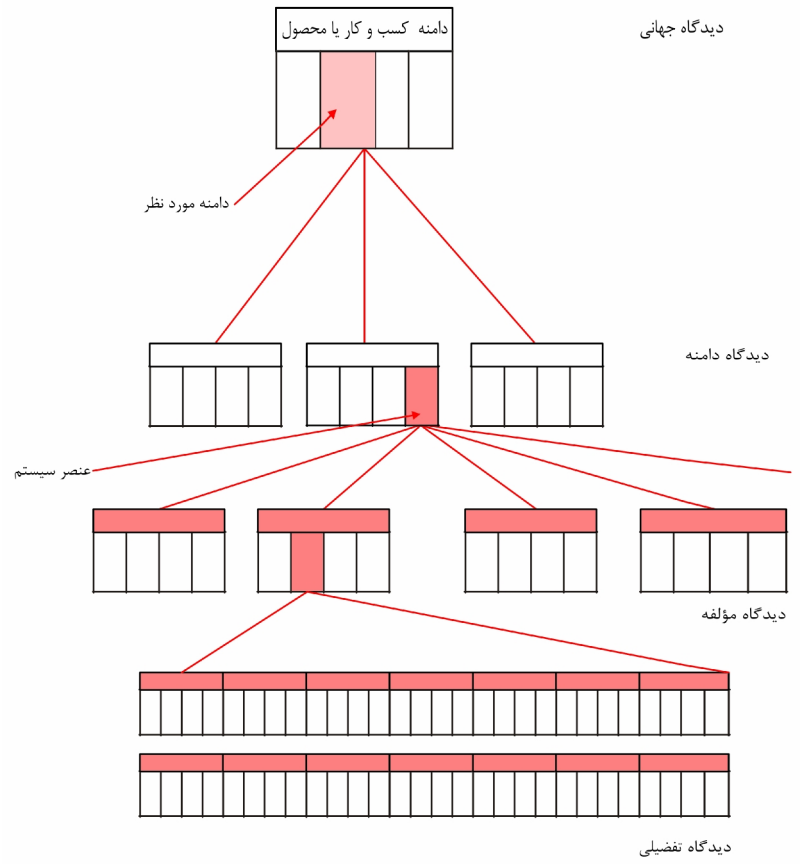
مهندسی سیستم[1] فرایندی است که تمرکز خود را صرف مجموعه‌ای از عناصر تحلیل، طراحی و سازماندهی آن‌ها در قالب یک سیستم می‌کند که ماحصل این کار تولید یک محصول، ارائهٔ خدمت یا سرویس یا یک فن‌آوری مناسب برای تبدیل اطلاعات و کنترل آن‌هاست.

اگر محدودهٔ کار مهندسی سیستم تمرکز بر کسب و کار داشته باشد، **مهندسی فرایند حرفه** یا **مهندسی اطلاعات** نامیده می‌شود. هنگامی که توسعهٔ یک محصول مد نظر باشد، این فرایند، **مهندسی محصول** نامیده می‌شود. **مهندسی اطلاعات** و **مهندسی محصول**، هر دو در قلمرو علمی سعی در توسعهٔ سیستم‌های مبتنی بر رایانه دارند، اگرچه هر یک در محدودهٔ کاربرد متفاوتی به کار گرفته می‌شوند.

۳ـ۲ سلسله‌مراتب مهندسی سیستم

علی‌رغم دامنهٔ 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 دامنهٔ خاصی از خواسته‌ها به دست آید. در یک دامنهٔ خاص، نیاز برای عناصر سیستم هدف (برای مثال، داده‌ها، نرم‌افزار، سخت‌افزار، افراد) بررسی می‌شود. درنهایت، 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.

1 - System Engineering



شکل ۳ـ۱ سلسله‌مراتب مهندسی سیستم

۳ـ۳ بخش‌های مهندسی سیستم

۳ـ۳ـ۱ مهندسی فرایند حرف (BPE)[1]

هدف مهندسی فرایند حرفه که به آن **مهندسی اطلاعات (IE)**[2] نیز گفته می‌شود، تعریف معماری‌هایی است که امکان استفادهٔ کارا از اطلاعات در کاربردهای عملی را فراهم می‌سازند.

۳ـ۳ـ۱ـ۱ سه نوع معماری مهندسی اطلاعات

- معماری داده‌ها[3]

قالبی برای نیازهای اطلاعاتی مؤسسه در نظر گرفته و واحد سازندهٔ این قالب Data Object است که پس از شناسایی، رابطهٔ بین آن‌ها مشخص می‌شود. یک رابطه، نشان‌دهندهٔ نحوهٔ ارتباط اشیا با یکدیگر است.

- معماری کاربردها[4]

معماری کاربرد شامل عناصری از یک سیستم است که صفات اشیا را در معماری داده‌ای برای اهداف حرفهٔ خاص دستکاری و تبدیل می‌کند. معماری کاربردها، سیستم یا برنامه‌های (نرم‌افزاری) است که این تبدیلات را انجام می‌دهند. به‌هرحال، در یک محدودهٔ وسیع، معماری کاربرد، نقش افراد (که مبدل‌های اطلاعات یا کاربران هستند) و رویه‌های حرفه را که خودکار نشده‌اند بر عهده دارد.

- زیرساخت فن‌آوری[5]

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

مجموعه‌ای از نرم‌افزارها و سخت‌افزارها که دو معماری قبلی را پوشش می‌دهند. زیرساخت فن‌آوری، اساس معماری‌های داده و کاربرد را فراهم می‌کند. این زیرساخت شامل سخت‌افزار و نرم‌افزاری است که برای پشتیبانی کاربردها و داده‌ها از آن‌ها استفاده می‌شود.

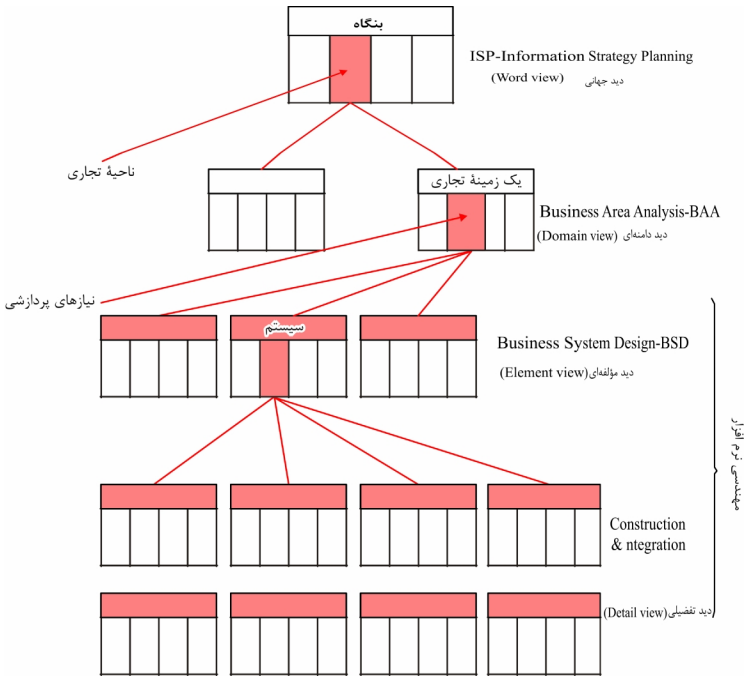
۳ـ۳ـ۱ـ۲ سلسله‌مراتب مهندسی اطلاعات

سلسله‌مراتب فعالیت‌های مهندسی اطلاعات با دیدگاه کلی از طریق **برنامه‌ریزی راهبردی اطلاعات** (ISP)[1] به دست می‌آید. ISP به کل مجموعهٔ 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، رابطهٔ آن‌ها و نحوهٔ جریان آن‌ها بین دامنه‌های موجود در حرفه را نیز تعریف می‌کند.

دیدگاه دامنه، با فعالیت **تحلیل زمینهٔ حرفه** (BAA)[2] مورد توجه قرار می‌گیرد. BAA جزئیات داده‌ها (به شکل انواع شیء داده‌ای و موجودیت‌ها) و نیازهای عملکردی (به شکل فرایندها) مربوط به ناحیه‌های انتخاب‌شده (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 مرحلهٔ **طراحی سیستم حرفه** (BSD)[3]، نیازمندی‌های اصلی سیستم اطلاعاتی خاص مدل‌سازی می‌شوند و این نیازها به معماری داده‌ها، معماری کاربردها و زیرساخت فن‌آوری ترجمه می‌شوند.

مرحلهٔ نهایی، ساخت و یکپارچه‌سازی مؤلفه‌هاست که بر جزئیات پیاده‌سازی تمرکز دارد. فعالیت یکپارچه‌سازی نیز سیستم اطلاعاتی جدیدی را در محدودهٔ زمینهٔ تجاری قرار می‌دهد و تمام آموزش‌های کاربران و حمایت‌های پشتیبانی را برای به دست آوردن تغییر ملایم، ایجاد می‌کند.



شکل ۳ـ۲ سلسله‌مراتب مهندسی اطلاعات

۳ـ۳ـ۲ مهندسی محصول

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

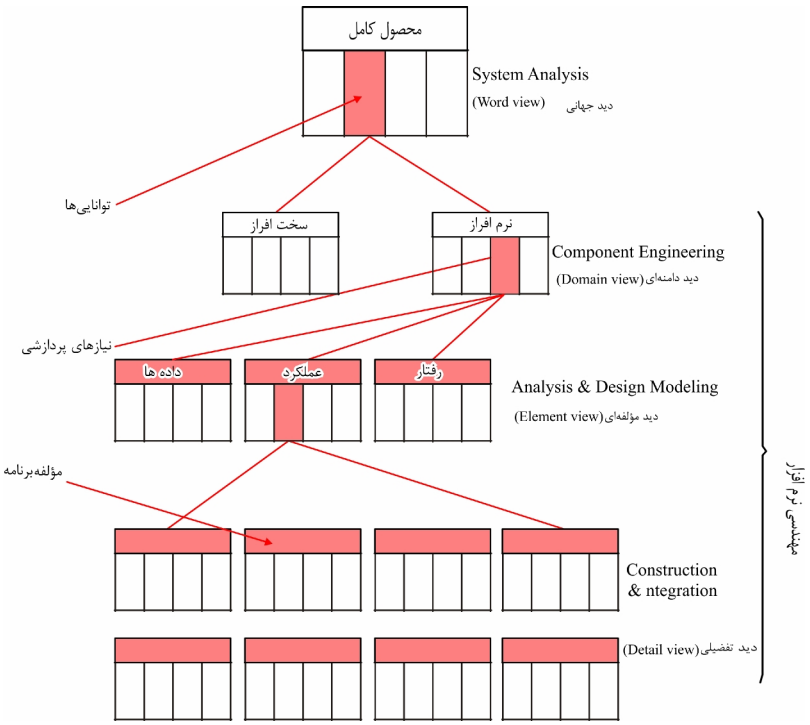
- نرم‌افزار
- سخت‌افزار
- افراد
- پایگاه داده‌ها

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 ۳ـ۳ 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.**

۳ـ۴ مهندسی نیازها[1]

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 مشخصهٔ محصول و مدیریت نیازها در ضمن تبدیل به یک سیستم عملیاتی فراهم می‌کند. بنابراین فرایند مهندسی نیازها در ۶ مرحلهٔ مشخص به صورت زیر انجام می‌گیرد:

۳ـ۴ـ۱ 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**

1 - Requirements Engineering

۳ـ۴ـ۲ 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، رابطهٔ 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.

۳ـ۴ـ۳ تدوین مشخصهٔ 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 استانداردی استفاده شود. البته گاهی لازم است در هنگام تدوین مشخصه، انعطاف مناسبی در استفاده از استانداردها وجود داشته باشد.

۳ـ۴ـ۴ مدل‌سازی سیستم

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

۳ـ۴ـ۵ اعتبارسنجی نیازها

اعتبارسنجی نیازها با مرورهای رسمی و بازنگری سند مشخصهٔ 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 اولیهٔ مکانیزم اعتبارسنجی، مورد بازنگری و مرورهای رسمی فنی قرار می‌گیرند.

۳ـ۴ـ۶ 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.

۳ـ۵ فعالیت‌های اساسی در مهندسی نیازها

۳ـ۵ـ۱ـ تعیین دامنهٔ اطلاعات

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

۳ـ۵ـ۲ـ 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.

۳ـ۵ـ۳ـ تجزیه

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

۳ـ۵ـ۴ـ دیدگاه‌های مورد نیاز پیاده‌سازی

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

۳ـ۵ـ۵ـ تهیهٔ مشخصهٔ 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[1] صورت می‌گیرد و زمان و هزینه برآورد شده و به مشتری ارائه می‌شود.

1 - Feasibility Stady

فصل چهارم

اصول و قواعد تحلیل نیاز

مهندسی نیازهای نرم‌افزار، فرایند کشف، پالایش، مدل‌سازی و مشخص کردن نیازها را شامل می‌شود. نیازهای سیستم و نقش تخصیص داده شده به نرم‌افزار، در ابتدا توسط مهندس سیستم شناسایی، تعریف و همراه با جزئیات پالایش می‌شوند. مدل‌های داده‌ای، جریان اطلاعات و کنترل و درنهایت رفتار عملیاتی سیستم شناسایی می‌شوند. راه حل‌های جایگزین تحلیل و سپس مدل تحلیل کاملی تهیه می‌گردد.

بنابراین فرایند تحلیل نیازها شامل شناسایی، مدل‌سازی، پالایش و تعیین مشخصات زمینهٔ مورد مطالعه است.

شکل ۴ـ۱ ارتباط مراحل مهندسی نرم‌افزار

تحلیل نیاز، مهندسین نرم‌افزار را قادر می‌سازد که:

۱ـ توابع و عملکرد آن‌ها را تعیین کنند.

۲ـ ارتباط بین نرم‌افزار و سایر اجزای سیستم را مشخص سازند.

۳ـ محدودیت‌ها را تعریف و نشان دهند.

۴ـ نرم‌افزار مورد نظر را انتخاب کند و مدل‌سازی از داده‌ها، توابع و رفتار سیستم را انجام دهند.

۵ـ با استفاده از این مدل‌ها اقدام به طراحی مدل‌های داده‌ای، معماری، واسط‌ها و پردازش‌ها کنند.

۴ـ۱ فعالیت‌های عمدهٔ تحلیل نیاز

شناسایی مسئله

در ابتدا، تحلیل‌گر، مشخصهٔ سیستم را (در صورت وجود) همراه با طرح توسعهٔ پروژهٔ نرم‌افزار (SDP)[1] مطالعه می‌کند. درک نرم‌افزار در رابطه با سیستم و مرور محدودهٔ نرم‌افزار، دارای اهمیت است. هدف از این کار تشخیص عناصر اصلی مسئله شامل شناسایی اهداف، محدودیت‌ها و تعیین ارتباطات موجود تا حد جزئیات است، همان‌گونه که توسط مشتری و کاربران بیان شده است.

1 - Software Development Plan

ارزیابی و ترکیب

تحلیل‌گر باید تمام اشیای قابل مشاهده را تعریف کند، جریان و محتویات اطلاعات را ارزیابی کند، جزئیات تمام توابع نرم‌افزاری را تعریف کرده، رفتار نرم‌افزار را در رابطه با وقایعی که بر سیستم اثر می‌گذارند تشخیص دهد، خصوصیات واسط‌های سیستم را شناسایی کند و محدودیت‌های بیشتری از طراحی را آشکار کند. این کارها برای توصیف روش یا راه حلی برای مسئله صورت می‌گیرد که قابل ارزیابی و ترکیب باشد. در حین ارزیابی و ترکیب راه حل، تمرکز اولیهٔ تحلیل‌گر بر **"چه چیزی"** و نه بر **"چگونگی"** است.

مدل‌سازی

ایجاد مدل داده‌ای، مدل عملکردی و رفتاری در این مرحله انجام می‌گیرد.

تعیین و شناسایی مشخصات مسئله

پیش‌بینی امکانات نرم‌افزار و حدود و ثغور آن در این مرحله انجام می‌شود.

بازنگری و کنترل

استقرار مکانیزم‌های کنترل مستمر و بازنگری در تمامی مراحل فوق به منظور دستیابی به کیفیت مناسب نرم‌افزار صورت می‌گیرد.

۴ـ۲ روش‌های جمع‌آوری نیازها

۴ـ۲ـ۱ روش‌های ارتباط با مشتری برای تحلیل نیازها

جمع‌آوری اطلاعات

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

۴ـ۲ـ۲ روش هدایت‌شدهٔ تعیین مشخصهٔ کاربرد (نیازها)

محققین گوناگون شیوهٔ تیمی را برای جمع‌آوری نیازها و کاربردهای سیستم توسعه داده‌اند که در مراحل اولیهٔ تحلیل و تعیین مشخصات نیازها به کار گرفته می‌شود. روش هدایت‌شدهٔ تعیین مشخصهٔ کاربرد (FAST)[1]، ایجاد یک تیم شامل مشتری و توسعه‌دهنده را پیشنهاد می‌کند که با یکدیگر کار کنند تا مسئله و نیازها را شناسایی و راه حل‌ها را مشخص کنند. FAST به طور عمده توسط جامعهٔ سیستم‌های اطلاعاتی به کار گرفته شده است ولی قابلیت بهبود و استفاده در کلیهٔ کاربردها را نیز دارد.

۴ـ۲ـ۳ استقرار تابع کیفیت

استقرار تابع کیفیت (QFD)[2] یک فن مدیریت کیفیت است که نیازهای مشتری را به نیازهای فنی نرم‌افزار ترجمه می‌کند.

QFD سه نوع نیاز را مشخص می‌کند:

نیازهای معمولی: اهدافی است که برای محصول یا سیستم در نظر گرفته شده‌اند. اگر این نیازها وجود داشته باشند، مشتری راضی است. مثال‌هایی از نیازهای معمولی عبارت‌اند از درخواست نوعی صفحهٔ نمایش رنگی، توابع خاص سیستم و سطوح کارایی تعریف‌شده 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 می‌شود. مثال‌هایی از نیازهای مورد انتظار عبارت‌اند از: سهولت ارتباط آسان، صحت کلی و قابلیت اطمینان و سهولت نصب و ... .

1 - Facilitated Application Specification Technique
2 - Quality Function Deployment

نیازهای ایده‌آل: 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[1] : وظایف سازمان چگونه تبدیل به توابع عملیاتی می‌شود؟
- گسترش و شناسایی اطلاعات[2] : اطلاعات موجود در فرم‌ها شامل چیست؟
- گسترش و شناسایی کار[3] : تبدیل فعالیت‌ها به اجرا چگونه صورت می‌گیرد؟
- گسترش و شناسایی مقادیر داده‌ها[4] : حدود مقادیر و ساختار و محتوای داده‌ها چیست؟

۴ـ۲ـ۴ تهیهٔ 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 مشخص‌کنندهٔ مسیر استفاده از سیستم است ایجاد کند. این سناریوها اغلب مورد کاربرد یاUse-Case نامیده می‌شوند و توصیفی را از چگونگی استفادهٔ 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.

۴ـ۳ اصول تحلیل نیاز

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

محدودهٔ اطلاعات مسئله نمایش داده شود و قابل درک باشد.

توابعی که نرم‌افزار را اجرا می‌کنند باید تعریف شوند.

رفتار نرم‌افزار (در نتیجهٔ 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 نحوهٔ ارتباط ماشین با انسان آگاه شود. نمونهٔ تهیه‌شده می‌تواند دورانداختنی[5] و یا تکاملی[6] 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 - Evalutionary Prototype

فصل پنجم

مدل‌سازی تحلیل نیازهای نرم‌افزار

در سطح فنی، مهندسی نرم‌افزار با یک‌سری از فعالیت‌های مدل‌سازی شروع می‌شود که به مشخصات کامل نیازها و نمایش طرحی جامع برای نرم‌افزار تبدیل می‌شوند. مدل تحلیل، شامل مجموعه‌ای از مدل‌هاست و اولین نمایش فنی از سیستم است.

۵-۱ تاریخچه فعالیت‌های عمدهٔ تحلیل نیاز

واژهٔ تحلیل ساخت‌یافته که ابتدا توسط Douglas Ross استفاده شد، توسط DeMarco مشهور شد و بعدها توسط Page-Jones و سپس Sarson و Gane توسعه یافت. در دههٔ ۸۰ توسعهٔ این روش برای سیستم‌های بلادرنگ توسط Ward و Mellor و سپس Hatley و Pirbhai گسترش یافت. این توسعه‌ها باعث قوی‌تر شدن انجام تحلیل شدند که می‌توانست به طور مؤثری در مسایل مهندسی به کار گرفته شود. تلاش‌هایی برای توسعهٔ یک نشانه‌گذاری یکپارچه پیشنهاد و روش‌های مدرنی با استفاده از ابزارهای CASE نیز معرفی شدند.

در حال حاضر، در مجموع دو متدولوژی متداول **ساخت‌یافته**[1] و **شیءگرا**[2] برای مدل‌سازی تحلیل سیستم‌ها وجود دارد. متداول‌ترین و آخرین روش ساخت‌یافته تحت عنوان " روش تحلیل و طراحی ساخت‌یافته سیستم" به نام SSADM[3] معرفی شده است که کاربردهای زیادی در سازمان‌ها داشته است. در روش ساخت‌یافته نگرش سازماندهی شده و منظم سیستمی از بالا به پایین[4] برای تحلیل و طراحی و نگرش پایین به بالا[5] برای پیاده‌سازی نرم‌افزار مد نظر است. در این روش:

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

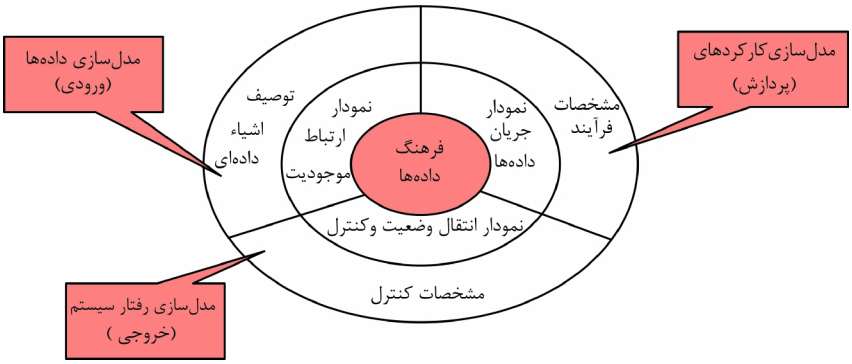
1 - Structured
2 - Object Oriented
3 - Structured System Analysis & Design Method
4 - Top Down
5 - Bottom Up

۵-۲ اهداف مدل تحلیل نیاز

- نیازهای مشتری را تشریح می‌کند.
- پایه و مبنایی برای مدل طراحی نرم‌افزار به وجود می‌آورد.
- مجموعه‌ای از نیازها را تعریف می‌کند که می‌توان درستی و اعتبار نرم‌افزار را آزمایش کرد.

۵-۳ عناصر مدل تحلیل نیاز

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



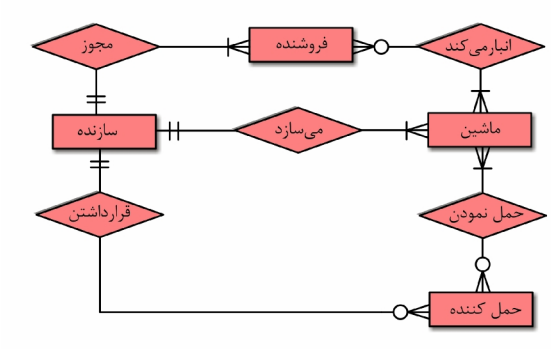
شکل ۵-۱ مدل تحلیل نیاز ساخت‌یافته

بنابراین مدل‌های اصلی تحلیل در روش ساخت‌یافته به شرح زیر هستند:

- نمودار ارتباط موجودیت (ERD)[1]: مدل‌سازی داده‌ها،
- نمودار جریان داده‌ها (DFD)[2]: مدل‌سازی کارکردها و
- نمودار انتقال حالت (STD)[3]: مدل‌سازی رفتار سیستم است.

۵-۳-۱ مدل‌سازی داده‌ها[4]

مدل داده شامل سه عنصر اطلاعاتی توصیف‌شده است: **شیء داده‌ای**، **صفاتی** که توصیف‌کنندهٔ شیء داده‌ای هستند و **روابطی** که اشیای داده را به یکدیگر مرتبط می‌کنند.



شکل ۵-۲ نمودار ارتباط موجودیت

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

توجه: حداقل نتیجهٔ حاصل از مدل‌سازی داده‌ای به دست آوردن ساختار و شکل جداول پایگاه داده‌ای است.

۵ـ۳ـ۲ مدل‌سازی کارکردها و جریان اطلاعاتی[1]

داده‌ها در ضمن جریان در سیستم رایانه‌ای توسط پردازش‌ها تبدیل و دستکاری می‌شوند. سیستم، ورودی را به شکل‌های مختلف می‌پذیرد، سخت‌افزار، نرم‌افزار و عناصر انسانی را برای تبدیل آن به‌کار می‌گیرد و خروجی را به شکل‌های مختلف تولید می‌کند.

توجه: در اینجا تمرکز بر حرکت و پردازش اطلاعات است که با عنوان جریان اطلاعات[2] مطرح است.

- **نمودار جریان داده‌ها**

نمودار جریان داده (DFD) نمایشی گرافیکی است که جریان اطلاعات و تبدیلات را در ضمن حرکت داده‌ها از ورودی به خروجی نشان می‌دهد. شکل اصلی، **نمودار جریان داده، گراف جریان داده، یا چارت حبابی** نیز نامیده می‌شود.

DFD سطح صفر سیستم، **مدل زمینه** نیز نامیده می‌شود که کل عنصر نرم‌افزار را به صورت یک حباب، همراه با داده‌های ورودی و خروجی نشان داده شده توسط پیکان‌های واردشده و خارج‌شده از آن نشان می‌دهد. فرایندهای اضافی و مسیر جریان اطلاعات در ضمن تجزیهٔ DFD سطح صفر برای نشان دادن جزئیات بیشتر، نمایش داده می‌شوند.

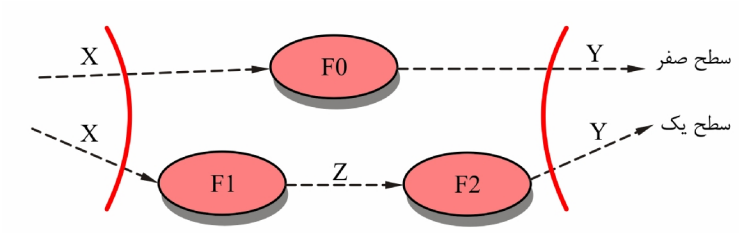
- **نکات مربوط به DFD**

- فقط گردش اطلاعات در DFD نشان داده می‌شود.
- توالی، تأخر و تقدم اطلاعات در DFD مشخص نیست.
- نحوهٔ تبدیل داده‌ها در DFD مشخص نیست و باید توصیفی برای شرح پردازش‌ها نوشت.
- هیچ عمل فیزیکی در DFD نشان داده نمی‌شود.
- در DFD کنترل‌ها دیده نمی‌شوند و باید شرح کنترل‌ها را جداگانه نوشت.

توجه: مجموعه‌ای از پردازش‌ها، فعالیت و مجموعه‌ای از فعالیت‌ها، زیرسیستم و مجموعه‌ای از زیرسیستم‌ها، سیستم نامیده می‌شود.

- **توازن جریان در DFD**

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



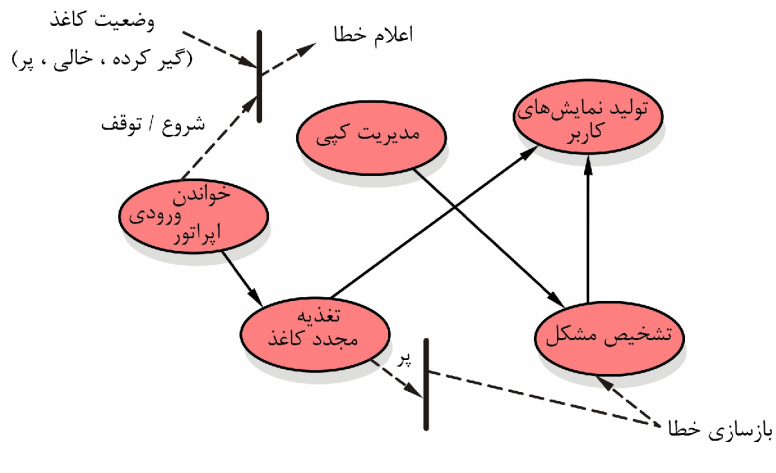
شکل ۵ـ۳ نحوه شکست نمودار جریان داده با حفظ توازن جریان

۵ـ۳ـ۳ مدل‌سازی رفتار سیستم با نمودار انتقال وضعیت

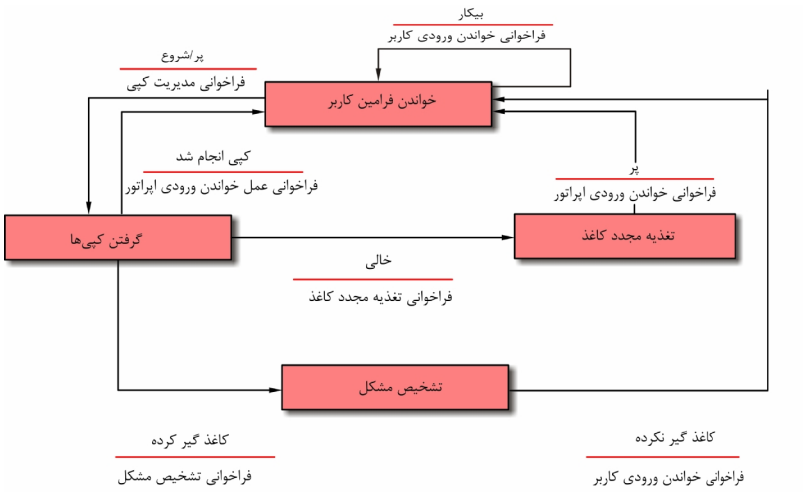
مدل‌سازی رفتار سیستم مبنای عملیاتی تمام روش‌های تحلیل نیازهاست. نمودار انتقال و تغییر حالت (STD)، نشان‌دهندهٔ رفتار سیستم، با استفاده از مشخص کردن حالت‌ها و وقایعی است که باعث تغییر وضعیت سیستم می‌شوند. علاوه بر آن STD نشان می‌دهد که چه

1 -Functional Modeling
2 -Information Flow

عکس‌العمل‌هایی (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 ۵ـ۴ نمودار جریان کنترل دستگاه کپی



شکل ۵ـ۵ 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 اشیا 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 نرم‌افزار یا سیستم را به تصویر بکشد.
۲- ورودی و خروجی‌های اولیه باید به دقت تعریف شوند.
۳- با مجزا کردن پردازش‌ها، موجودیت‌ها و ذخایر داده‌ای، 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 نرم‌افزار

در حوزهٔ مهندسی نرم‌افزار، طراحی بر چهار محور اصلی **داده، معماری، واسط و پیمانه** متمرکز است.

۶ـ۱ تعریفی از طراحی

فرایند به‌کارگیری 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 نرم‌افزار به وجود می‌آورد. هر تغییری در اثنای پیاده‌سازی نرم‌افزار باید روی مدل طراحی و متعاقب آن روی مدل تحلیل نیاز اعمال شود.

۶ـ۲ طراحی نرم‌افزار و مهندسی نرم‌افزار

در آغاز و پس از تحلیل و تعیین نیازهای نرم‌افزار، طراحی نرم‌افزار با انجام سه فعالیت یعنی طراحی، تولید برنامه و آزمون، انجام می‌شود. هر یک از این فعالیت‌ها، اطلاعات را به‌گونه‌ای تغییر می‌دهند که درنهایت به نرم‌افزار معتبر رایانه‌ای منجر می‌شود.

هر یک از عناصر مدل تحلیل (فصل ۵)، اطلاعات لازم را در ایجاد چهار مدل طراحی فراهم می‌آورند.

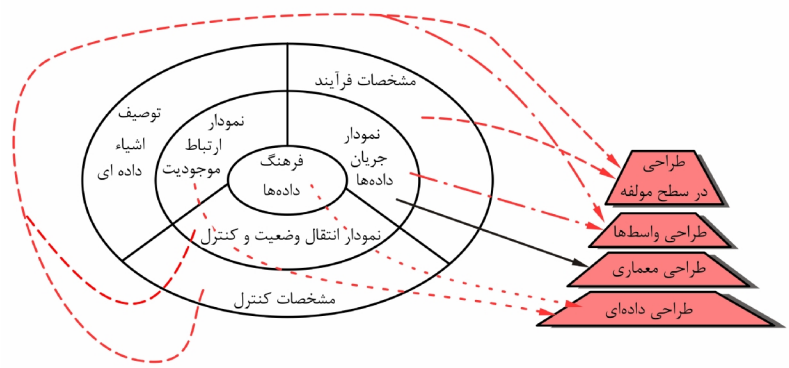
طراحی داده‌ها، مدل داده‌ای تولیدشده در مدل تحلیل را به ساختارهای داده‌ای لازم در اجرای نرم‌افزار تبدیل می‌کند. اشیا و روابط تعیین‌شدهٔ داده‌ها در نمودار ارتباط موجودیت‌ها و محتوای مشروح داده‌ای در فرهنگ داده‌ها، مبنای فعالیت طراحی داده‌ها را فراهم می‌کنند. بخشی از طراحی داده‌ها ممکن است با طراحی معماری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

طراحی معماری، رابطهٔ بین عناصر اصلی مؤلفه‌های نرم‌افزاری را تعیین می‌کند، یعنی رابطهٔ "الگوهای طراحی" به‌کاررفته در تحقق نیازهای سیستم و محدودیت‌های مؤثر بر شیوهٔ اجرای الگوهای طراحی معماری تدوین می‌شود.

طراحی واسط، توصیف‌کنندهٔ نحوهٔ ارتباط نرم‌افزار با سیستم‌ها و افرادی است که با آن تعامل داشته و آن را به کار می‌برند.

طراحی مؤلفه (پیمانه)، عناصر ساختاری معماری نرم‌افزار را به توصیف رویه‌ای مؤلفهٔ نرم‌افزاری تبدیل می‌کند. اطلاعات به‌دست‌آمده از STD CSPEC و PSPEC، پایه و اساس طراحی مؤلفه به شمار می‌روند.

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



شکل ۶ـ۱ برگردان مدل تحلیل به مدل طراحی نرم‌افزار

اهمیت طراحی نرم‌افزار را تنها با یک کلمه یعنی "کیفیت" می‌توان بیان کرد. طراحی روندی است که طی آن کیفیت فرایند مهندسی نرم‌افزار، بهبود می‌یابد. طراحی، نمونه‌هایی از نرم‌افزار را که از لحاظ کیفی قابل ارزیابی هستند، در اختیار ما قرار می‌دهد. این تنها راهی است که به واسطه آن می‌توانیم شکل درست نیازها و خواسته‌های مشتری را به یک محصول نرم‌افزاری یا سیستم تبدیل کنیم.

طراحی نرم‌افزار فرایندی تکراری است که به موجب آن نیازها و ضرورت‌ها برای ساخت نرم‌افزار، تبدیل به یک **"طرح یا نقشه"** می‌شوند. در ابتدا، طراحی یک دید کلی از نرم‌افزار را در سطح بالایی از انتزاع نشان می‌دهد. با تکرار و پالایش در طراحی، نمایش طرحی در سطوح بسیار پایین‌تر از انتزاع حاصل می‌شود.. این سطوح برای دستیابی به نیازهای قابل ردیابی، ملاک کار خواهند بود.

۶ـ۲ـ۱ 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 می‌دهند (بعداً در این فصل مورد بحث قرار می‌گیرند و (۳) به شیوه‌ای تکاملی اجرا شده و بدین ترتیب اجرا و آزمون را تسهیل کند.
- طراحی باید پیمانه‌ای[1] باشد؛ یعنی نرم‌افزار باید به طور منطقی به اجزایی تقسیم شود که اعمال اصلی و فرعی خاصی را انجام دهند.
- طراحی باید نمایش‌های مجزایی از داده‌ها، معماری، واسطها و پیمانه‌ها را دربرداشته باشد.
- طراحی باید منجر به ساختارهای داده‌ای شود که برای پیاده‌سازی اشیا مناسب بوده و از الگوهای قابل تشخیص داده‌ها ناشی شوند.
- طراحی باید به اجزایی منتهی شود که خصوصیات مستقل کاربردی را نمایش دهند.
- طراحی باید به واسطهایی ختم شود که از پیچیدگی روابط بین پیمانه‌ها و محیط خارجی می‌کاهند.
- طراحی باید حاصل کاربرد یک شیوهٔ قابل تکرار با استفاده از اطلاعات به‌دست‌آمده در اثنای تحلیل نیازهای نرم‌افزاری باشد.

1 - Modular

۶ـ۳ اصول طراحی

طراحی نرم‌افزار، هم یک فرایند و هم یک مدل است. Davis مجموعهٔ اصولی را برای طراحی نرم‌افزار به شرح زیر پیشنهاد می‌کند:

- **باریک‌بینی نباید در فرایند طراحی وجود داشته باشد.**
- **طراحی باید قابل ردیابی به مدل تحلیل نیاز باشد.**
- **طراحی نباید دوباره‌کاری باشد.**
- **طراحی باید فاصلهٔ منطقی بین نرم‌افزار 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 داده‌های غیر عادی، رویدادها یا شرایط کاری، باید به آرامی از کار بایستد.**
- **طراحی باید ضمن شکل‌گیری 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 برنامه‌نویسی نیست و برنامه‌نویسی نیز معادل طراحی نیست.**

۶ـ۴ قواعد طراحی

قواعد نه‌گانه طراحی زیر در ارائه پاسخ سؤالات زیر کمک می‌کنند:

- چه معیارهایی برای تقسیم‌بندی نرم‌افزار به مؤلفه‌های مجزا به کار گرفته شده است؟
- چگونه کارکردها یا ساختار داده‌ها از نمایش مفهومی نرم‌افزار مجزا می‌شوند؟
- آیا معیار یکنواختی وجود دارد که کیفیت طراحی نرم‌افزار 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 ⇐ تحلیل نیاز درست ⇐ طراحی خوب

۶ـ۴ـ۱ انتزاع

وقتی برای هر مسئله به دنبال راه حل پیمانه‌ای باشیم، سطوح انتزاع زیادی نیز مطرح می‌شوند. در بالاترین سطح انتزاع، راه حل با به‌کارگیری زبان رایج در محیط مسئله و به صورت کلی بیان می‌شود. در سطوح پایین‌تر انتزاع، جهت‌گیری و گرایش بیشتر رویه‌ای است.

حین پیش‌روی در سطوح مختلف انتزاع، تلاش می‌کنیم تا انتزاع‌های **رویه‌ای، داده‌ای** و کنترلی ایجاد شوند.

انتزاع رویه‌ای توالی مشخصی از دستورالعمل‌هاست که عملکرد خاص و محدودی دارد.

انتزاع داده‌ای مجموعه‌ای از داده‌های با نام است که یک شیء داده‌ای را توصیف می‌کند.

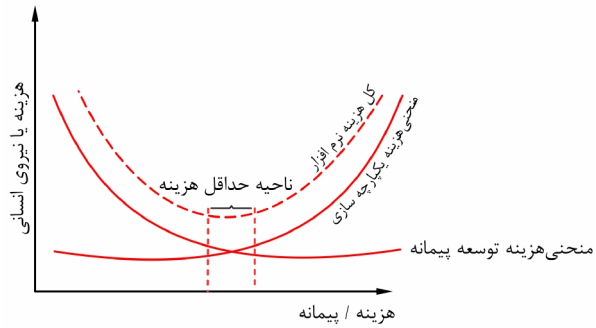
انتزاع کنترلی سومین شکل انتزاعی است که در طراحی نرم‌افزار به کار می‌رود. همانند انتزاع رویه‌ای و داده‌ای، انتزاع کنترلی بر مکانیزم کنترل برنامه بدون مشخص کردن جزئیات داخلی اشاره دارد.

۶ـ۴ـ۲ پالایش

پالایش گام به گام یک راهبرد طراحی بالا به پایین است. برنامه با سطوح پالایشی متوالی جزئیات رویه‌ای، توسعه می‌یابد. تا زمان دستیابی به دستورات زبان برنامه‌نویسی، توسعهٔ سلسله‌مراتب با تجزیهٔ عملکردها (انتزاع رویه‌ای) طی شیوه‌ای گام به گام صورت می‌گیرد.

۶ـ۴ـ۳ پیمانه‌ای

معماری یک نرم‌افزار، وقتی پیمانه‌ای است که در آن نرم‌افزار به اجزای نشانی‌پذیر با اسامی جداگانه به نام **"پیمانه‌ها"** تقسیم شده باشد و سپس برای رفع نیازهای مسئله، یکپارچه و مجتمع شوند. نمودار زیر رابطهٔ بین تعداد پیمانه‌ها در یک نرم‌افزار و هزینه‌های متناظر را نشان می‌دهد. با افزایش پیمانه‌ها هزینه یکپارچگی و درنهایت کل هزینه نرم‌افزار افزایش می‌یابد. بنابراین تعیین تعداد پیمانه‌های مطلوب در طراحی نرم‌افزار مهم و عموماً به صورت شهودی و بر اساس تجربه صورت می‌گیرد.



شکل ۶ـ۲ پیمانه شدن و هزینهٔ نرم‌افزار

Meyer پنج معیار را در ارزیابی یک شیوهٔ طراحی و بر اساس توانایی آن در تعریف یک سیستم مؤثر پیمانه‌ای معرفی می‌کند:

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

۶ـ۴ـ۴ معماری نرم‌افزار

معماری نرم‌افزار به "ساختار کلی نرم‌افزار و راه‌های ایجاد یکپارچگی ذهنی سیستم از طریق این ساختار" اشاره می‌کند. معماری در ساده‌ترین شکل خود عبارت است از: ساختار سلسله‌مراتبی اجزای برنامه (پیمانه‌ها)، شیوهٔ ارتباط این اجزا و ساختار داده‌هایی که توسط این اجزا مورد استفاده قرار می‌گیرند. معماری نرم‌افزار باید از مدل نیاز استخراج شود.

۶ـ۴ـ۵ سلسله‌مراتب کنترل در برنامه‌ها

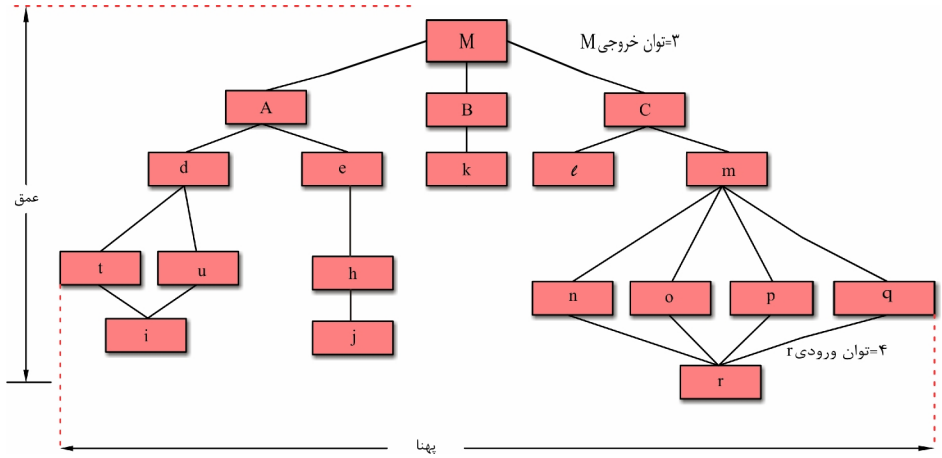
"سلسله‌مراتب کنترل" که **"ساختار برنامه"** نیز نام دارد، بیانگر سازماندهی اجزای برنامه (پیمانه‌ها) بوده و بر سلسله‌مراتب کنترل نیز دلالت دارد. دو مفهوم زیر در این ارتباط تعریف می‌شوند:

توان خروجی مقیاس تعداد پیمانه‌هایی است که مستقیماً توسط پیمانه‌ای دیگر کنترل می‌شوند.

توان ورودی بیانگر تعداد پیمانه‌هایی است که یک پیمانهٔ خاص را به طور مستقیم کنترل می‌کنند.

رابطهٔ کنترلی میان پیمانه‌ها به شیوهٔ زیر بیان می‌شود:

پیمانه‌ای که پیمانهٔ دیگری را کنترل می‌کند، **پیمانهٔ حاکم** نام دارد و برعکس پیمانهٔ تحت کنترل پیمانهٔ دیگر، **پیمانهٔ کنترل‌شده** نامیده می‌شود.

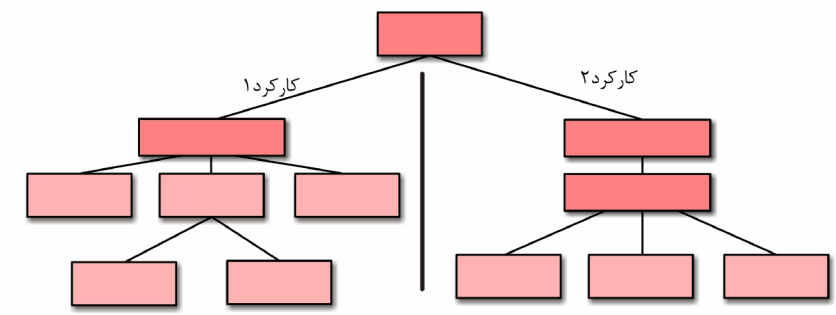


شکل ۶ـ۳ ساختار سلسله‌مراتبی پیمانه‌ها در برنامه

۶ـ۴ـ۶ تجزیهٔ ساختاری

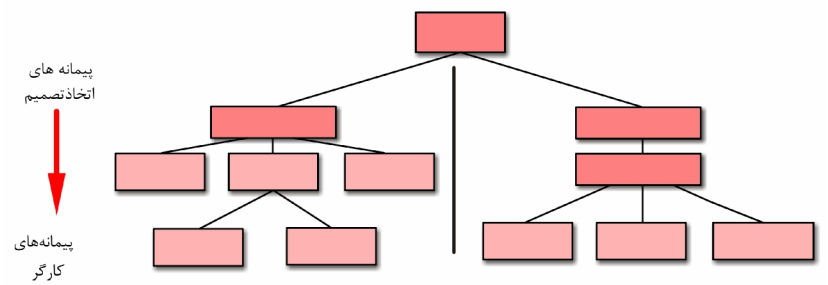
اگر سبک معماری یک سیستم، سلسله‌مراتبی باشد، می‌توان ساختار برنامه را، هم به صورت افقی و هم به طور عمودی تجزیه و افراز کرد. با توجه به شکل۴ (۱)، **افراز افقی**، شاخه‌های جداگانهٔ سلسله‌مراتب پیمانه‌ای را برای هر یک از وظایف اصلی برنامه تعیین می‌کند. پیمانه‌های کنترلی که با سایهٔ تیره‌تر نشان داده شده‌اند، برای هماهنگی ارتباط بین وظایف و اجرای آن‌ها به کار می‌روند. ساده‌ترین شیوهٔ افراز افقی، سه بخش ورودی، تغییر و تبدیل داده‌ها (که اغلب فرایند نام دارد) و خروجی را تعیین می‌کند.

ازآنجاکه کارکردهای اصلی از یکدیگر جدا می‌شوند، اعمال تغییرات، پیچیدگی کمتری دارد و بسط و توسعهٔ سیستم بدون تأثیرات جانبی، راحت‌تر انجام می‌شود. از جنبهٔ منفی، افراز افقی باعث می‌شود داده‌های بیشتری از واسط‌های پیمانه عبور کرده و کنترل کلی گردش برنامه دشوار و پیچیده شود.



شکل ۶ـ۴ (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 ۶-۴: (۲) افراز عمودی

۶ـ۴ـ۷ ساختمان داده‌ها

ساختار داده‌ها، نمایش رابطهٔ منطقی میان عناصر جداگانهٔ داده‌هاست. ازآنجاکه ساختار اطلاعات بر طراحی نهایی رویه‌ای تأثیر خواهد داشت، ساختار داده‌ها به اندازهٔ ساختار برنامه در نمایش معماری نرم‌افزار اهمیت دارد. ساختار داده‌ها تعیین‌کنندهٔ نحوه سازماندهی، روش‌های دستیابی، میزان اتصال و راه‌های مختلف پردازش اطلاعات را نمایش می‌دهد.

۶ـ۴ـ۸ پردازش‌های نرم‌افزار

ساختار برنامه، سلسله‌مراتب کنترل را بدون توجه به توالی پردازش‌ها و تصمیمات تعیین می‌کند. رویهٔ نرم‌افزار بر جزئیات پردازشی هر پیمانه 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.

۶ـ۴ـ۹ پنهان کردن اطلاعات

مفهوم پیمانه‌ای بودن، یک سؤال اساسی را برای هر طراح نرم‌افزاری مطرح می‌کند. «برای دستیابی به بهترین مجموعهٔ پیمانه‌ها، چگونه یک راه‌حل نرم‌افزاری را تجزیه کنیم؟» اصل پنهان‌سازی اطلاعات[1] بیانگر آن است که "وجه مشخصهٔ پیمانه‌ها، تصمیمات طراحی است که هر پیمانه را از پیمانه‌های دیگر مخفی می‌سازد". به عبارتی دیگر، پیمانه‌ها باید طوری طراحی و مشخص شوند که اطلاعات (رویه و داده‌ها) موجود در هر پیمانه برای پیمانه‌های دیگری که به چنین اطلاعاتی نیاز ندارند، قابل دسترسی نباشند.

۶ـ۵ استقلال عملیاتی

مفهوم "استقلال عملیاتی" پیامد مستقیم طراحی پیمانه‌ای، مفاهیم انتزاع و پنهان‌سازی اطلاعات است. استقلال عملیاتی از طریق ایجاد پیمانه‌هایی با عملکرد یک‌منظوره و عدم ارتباط بیش از حد با پیمانه‌های دیگر، تحقق می‌یابد. حال سؤال این است که چرا استقلال عملیاتی با اهمیت است؟ زیرا این کار باعث:

- توسعهٔ آسان‌تر
- دارای واسط‌های ساده
- ساده‌تر شدن در نگهداری و آزمایش پیمانه‌های مستقل
- کاهش انتشار خطا
- و قابلیت استفادهٔ مجدد می‌شود.

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

۶ـ۵ـ۱ انسجام[2]

انسجام یا چسبندگی، بسط طبیعی مفهوم پنهان‌سازی اطلاعات است. یک پیمانهٔ یکپارچه، یک وظیفهٔ منفرد را در یک رویهٔ نرم‌افزاری با برقراری ارتباط محدود با رویه‌های در حال اجرای سایر بخش‌های برنامه، انجام می‌دهد. به بیان ساده‌تر، یک پیمانهٔ منسجم (به طور ایده‌آل) باید تنها یک کار را انجام دهد. در طراحی باید به انسجام بالا دست یافت.

انواع انسجام

- **انسجام عملیاتی**[3]: یک پیمانه فقط یک کار را به طور انحصاری انجام می‌دهد و نتیجه را بازمی‌گرداند.
- **انسجام لایه‌ای**[4]: در سطح زیر سیستم[5]، مؤلفه و کلاس، طراحی و پیاده‌سازی می‌شود. در این حالت مؤلفهٔ سطح بالاتر به مؤلفهٔ سطح زیرین دسترسی داشته ولی برعکس آن امکان‌پذیر نیست.

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

- **انسجام ارتباطی**[1]: 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**[2]: عملیات مؤلفه‌ها پشت سر هم اجرا می‌شوند. درواقع هدف پیاده‌سازی یک‌سری عملیات پشت سر هم است. در این حالت خروجی مؤلفهٔ اول، ورودی مؤلفه دوم را تأمین می‌کند.
- **انسجام رویه‌ای**[3]: 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**[4]: 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**[5]:در انتهای پایینی (نامطلوب) طیف، با پیمانه‌هایی سر و کار داریم که مجموعه‌ای از وظایف را انجام می‌دهند که ارتباط محکمی با هم ندارند.

۶ـ۵ـ۲ اتصال[6]

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 محتوا**[7]: 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**[8]: 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**[9]: 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 برچسبی**[10]: هنگامی رخ می‌دهد که بخشی از ساختمان داده‌ها (به‌جای آرگومان‌های ساده) از طریق یک واسط پیمانه عبور داده شوند.
- **اتصال داده‌ای**[11]: 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**[12]: 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**[13]: 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 نرم‌افزار

۷ـ۱ چرا معماری؟

معماری یک نرم‌افزار نمودی است که مهندس نرم‌افزار را قادر می‌کند:

- میزان تأثیر طرح را در مرتفع کردن نیازهای بیان‌شده، تحلیل کند.
- معماری‌های جایگزین را در مرحله‌ای که تغییر طرح هنوز نسبتاً آسان است، بررسی کند.
- خطرات مربوط به ساخت نرم‌افزار را کاهش دهد.

طراحی داده باعث نمایش مؤلفه داده‌ای معماری و طراحی معماری بر تمرکز نمایش ساختار مؤلفه‌های نرم‌افزار، خواص آن‌ها و ارتباط بین آن‌ها قرار دارد.

۷ـ۲ طراحی داده‌ها

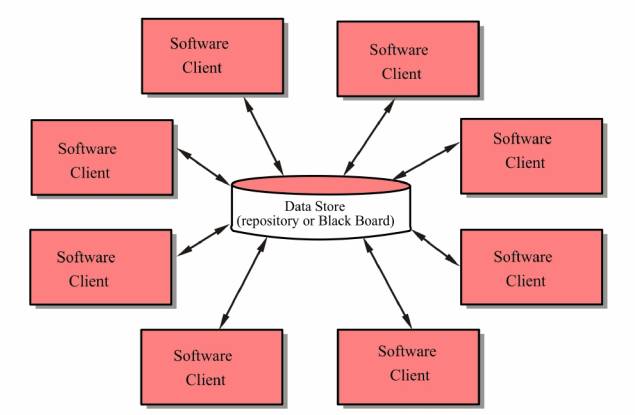
طراحی داده‌ها، همچون دیگر فعالیت‌های مهندسی نرم‌افزار (که گاهی به آن معماری داده‌ها نیز می‌گویند) مدلی از داده‌ها یا اطلاعات را در سطح بالایی از انتزاع، ایجاد می‌کند. سپس مدل داده‌ای به صورت بازنمای خاص پیاده‌سازی درمی‌آید که می‌توان آن را با سیستم مبتنی بر رایانه پردازش کرد. در بسیاری از برنامه‌های کاربردی، معماری داده‌ها تأثیر شگرفی بر معماری نرم‌افزار دارد که باید آن را پردازش کند؛ بنابراین مدل داده‌ای:

- اشیای داده‌ای را پالایش کرده و مجموعه‌ای از انتزاعات داده‌ای را توسعه می‌دهد.
- صفات اشیای داده‌ای را به عنوان یک یا چند ساختمان داده‌ای پیاده‌سازی می‌کند.
- ساختمان‌های داده‌ای برای اطمینان از ارتباط‌های مناسب بازنگری می‌شوند.
- ساختمان داده‌ها تا حد ممکن ساده‌سازی می‌شود.

طراحی مبتنی بر داده‌ها یک روش طراحی معماری نرم‌افزار است که به راحتی امکان گذر از مرحله تحلیل نیاز به طراحی توصیف ساختمان برنامه را می‌دهد.

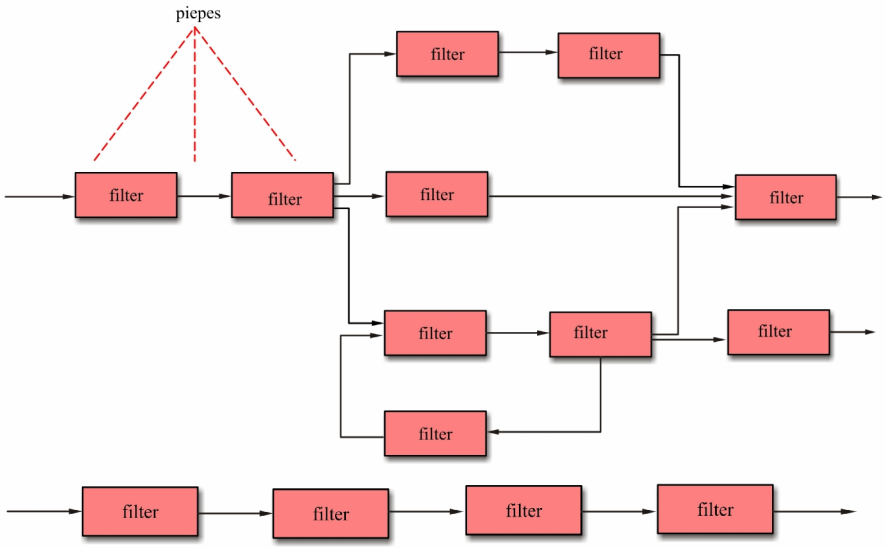
۳ـ۷ سبک‌های معماری نرم‌افزار

معماری مبتنی بر مخزن داده‌ها[1]: 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 ۷ـ۱ معماری داده‌گرا

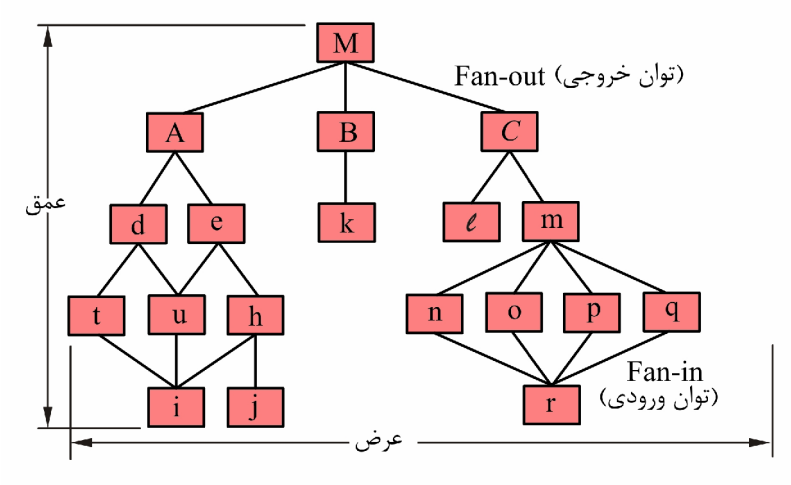
معماری مبتنی بر جریان داده[2]: این معماری وقتی به کار گرفته می‌شود که قرار است داده‌های ورودی از طریق اجرای یک‌سری پردازش‌های محاسباتی و اعمال تغییرات، به داده‌های خروجی تبدیل شوند.



شکل ۷ـ۲ معماری داده‌گرا

معماری فراخوانی و بازگشت[3]: 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.

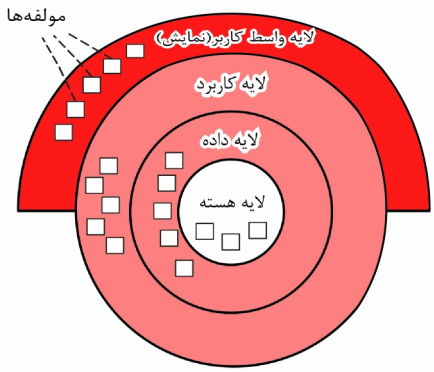
1 - Data Store Architecture
2 - Data flow Architecture
3 - Call and Return Architecture



شکل ۷ـ۳ معماری فراخوانی و برگشت

معماری‌های شیءگرا[1]: اجزای یک سیستم دربرگیرندهٔ 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[2]: ساختار اصلی این معماری در شکل زیر نشان داده شده است. تعدادی لایهٔ 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 ۷ـ۴ معماری لایه‌ای

۷ـ۳ـ۱ 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 وv 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 واسطهٔ 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.

۴ـ۷ نگاشت نیازها به معماری نرم‌افزار

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 شش‌مرحله‌ای زیر صورت می‌گیرد:

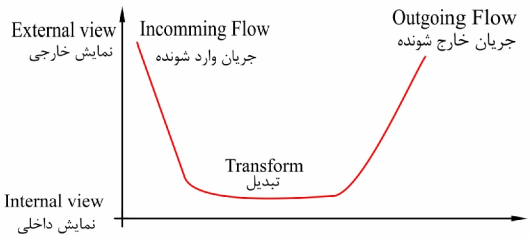
- تعیین نوع جریان اطلاعات
- مشخص کردن مرز جریان
- نگاشت نمودار جریان داده به ساختمان برنامه
- تعیین سلسله‌مراتب کنترل
- پالایش و بازنگری معماری حاصل با استفاده از معیارها، اصول و قواعد طراحی
- توصیف و تدوین معماری نهایی

۱ـ۴ـ۷ تعیین نوع جریان اطلاعات

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

۱- جریان تبدیلی[1]

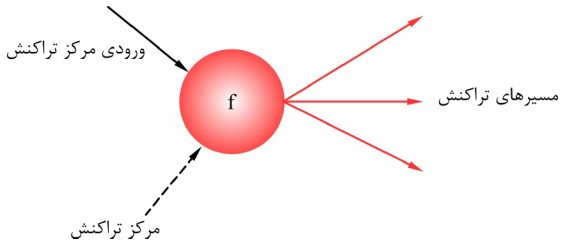
در جریان تبدیلی روال به صورت زیر در شکل ۷-۵ نشان داده شده است:



شکل ۷ـ۵ جریان تبدیلی

۲- جریان تراکنشی[2]

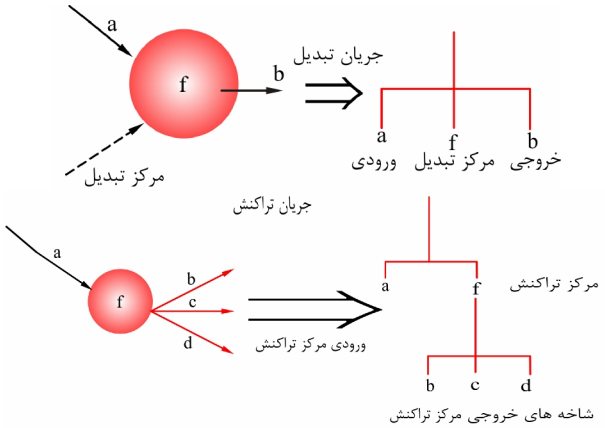
در جریان تراکنشی، جریانی وارد سیستم می‌شود و سیستم انتخاب می‌کند که از کدام شاخه خارج شود.



شکل ۷ـ۶ نمودار جریان تراکنشی

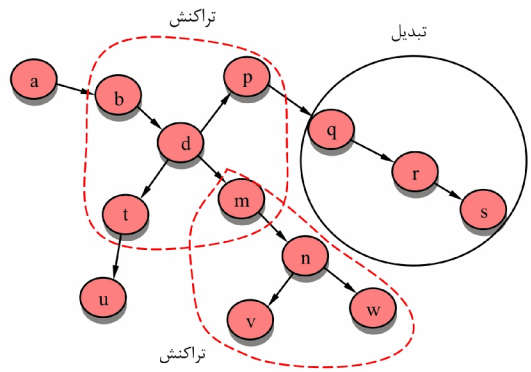
1 - Transform Flow
2 - Transaction Flow

مثال ۱ دو نمونه از نحوهٔ تبدیل جریان داده به معماری نرم‌افزار به صورت زیر است:



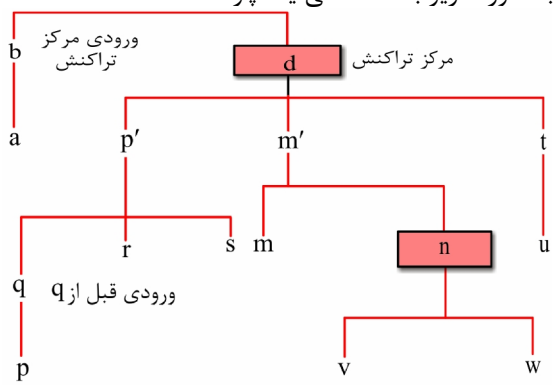
شکل ۷ـ۷ نحوهٔ نگاشت نمودار جریان داده تبدیلی و تراکنشی به معماری

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



شکل ۷ـ۸ نموداری شامل جریان‌های تبدیلی و تراکنشی

جواب: در نمودار جریان داده‌ای بالا d مرکز تراکنشی است مگر آنکه شرحی مبنی بر رخداد همزمان تمام خروجی‌ها از d ارائه شده باشد؛ بنابراین معماری متناظر با DFD داده‌شده به صورت زیر به دست می‌آید. چرا؟



شکل ۷ـ۹ نگاشت جریان داده به ساختار برنامه

استفاده موفق از نگاشت تراکنش یا تبدیل با کارهای اضافی دیگری تکمیل می‌شود 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 پردازش‌های هر پیمانه ارائه شود.
- توضیحی در مورد واسط‌های هر پیمانه داده شود.
- ساختارهای داده‌ای محلی و سراسری تعریف شوند.
- تمام محدودیت‌های طراحی ذکر شوند.
- مجموعه‌ای از بازنگری طراحی‌ها انجام شده باشد.
- و در صورت نیاز امکان پالایش در طراحی در نظر گرفته شود.

فصل هشتم

طراحی واسط کاربر

طراحی واسط بین اجزای نرم‌افزار
طراحی واسط‌ها بین نرم‌افزار و سایر تولیدکنندگان و مصرف‌کنندگان اطلاعات غیر انسانی (یعنی اشیای خارجی)
طراحی واسط بین انسان (یعنی کاربر) و رایانه
در این فصل صرفاً بر سومین مقولهٔ طراحی واسط‌ها یعنی طراحی واسط کاربر توجه خواهیم داشت.

۸ـ۱ قواعد طلایی Mendal

Mendal در کتاب خود با عنوان طراحی واسط، سه **"قانون طلایی"** زیر را مطرح می‌کند. این قوانین طلایی، عملاً به عنوان مجموعه‌ای از اصول طراحی واسط کاربر مطرح هستند که فعالیت مهم طراحی واسط‌های نرم‌افزاری را هدایت می‌کنند.

۸ـ۱ـ۱ واگذاری کنترل به‌کاربر

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 صفحهٔ نمایش ظاهر می‌شوند.

۸ـ۱ـ۲ کاستن از بار حافظهٔ کاربر

در این قانون نیز رعایت نکات زیر الزامی است:

- کاهش بار در حافظهٔ کوتاه‌مدت
- ایجاد پیش‌گزیده‌های معنی‌دار
- تعیین میان‌برهای شهودی
- آشکارسازی اطلاعات به شیوه‌ای تدریجی
- طرح بصری واسط باید بر اساس استعارهٔ جهان واقعی استوار باشد.

۸ـ۱ـ۳ سازگارسازی واسط

در این قانون نیز رعایت نکات زیر الزامی است:

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

۸ـ۲ 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.

۸ـ۲ـ۱ 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.

۸ـ۳ 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.

۸ـ۳ـ۱ تحلیل نیازهای کاربر، وظایف، محیط و مدل‌سازی

تحلیل اولیه بر سابقهٔ 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. درواقع 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.

۸ـ۳ـ۲ 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 تعریف‌شده می‌سازد، به طوری که همه اهداف قابلیت استفاده را برای یک سیستم برآورده سازد.

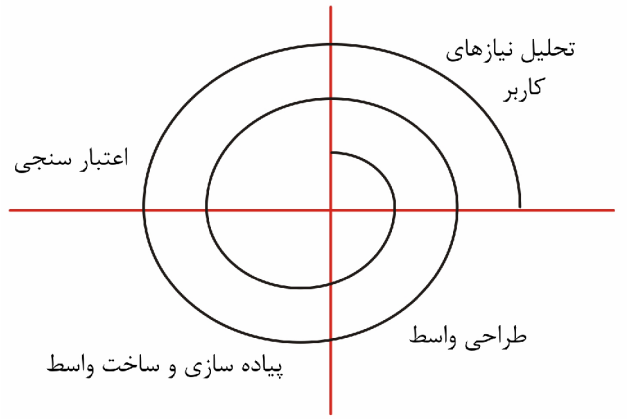
۸ـ۳ـ۳ پیاده‌سازی و ساخت واسط

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

۸ـ۳ـ۴ 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 ۸ـ۱ فرایند طراحی واسط کاربری

۸ـ۴ 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 توصیهٔ 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 نشانهٔ سمعی یا بصری همراه باشد.
- پیام باید قضاوت‌گونه باشد.

برچسب‌گذاری فرمان‌ها، فرمان تایپ‌شده زمانی رایج‌ترین شیوهٔ محاوره بین کاربر و نرم‌افزار بود و در هر نوع برنامهٔ 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 شیوهٔ ارتباطی مبتنی بر فرمان را ترجیح می‌دهند. هنگام انتخاب فرامین تایپ‌شده به عنوان نوعی شیوهٔ 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، واژهٔ 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 سلیقهٔ خویش مختصر، کوتاه و به دلخواه خود تنظیم کند؟

۸ـ۵ ابزارهای پیاده‌سازی واسط‌ها

برای انجام روش تکراری طراحی واسط‌های نرم‌افزاری، انواع ابزارهای طراحی واسط و ساخت نمونهٔ اولیه وجود دارند. این ابزارها با عنوان **سیستم توسعه واسط کاربر** (UIDS)[1] نامیده می‌شوند. این سیستم با استفاده از مؤلفه‌های نرم‌افزاری، راهکاری برای موارد زیر فراهم می‌آورد:

- مدیریت دستگاه‌های ورودی (مثل ماوس یا صفحه کلید)
- اعتبار سنجی ورودی کاربر
- کنترل خطا و نمایش پیام‌های 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

1 - User Interface Design System

۸ـ۶ 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 گسترهٔ آن با انجام آزمونی غیر رسمی که در ضمن آن کاربر، بازتابی بدون فکر قبلی دارد شروع شود و به مطالعهٔ رسمی ختم می‌گردد که از روش‌های آماری برای ارزیابی پرسشنامه‌های تکمیل‌شده توسط کاربران نهایی، استفاده می‌کند. برخی از معیارهای ارزیابی را هنگام بررسی‌های اولیهٔ طراحی، می‌توان اعمال کرد:

- **طول و پیچیدگی مشخصات مکتوب سیستم و واسط آن**، بیانگر میزان یادگیری لازم توسط کاربران سیستم است.
- **تعداد وظایف تعیین‌شدهٔ کاربر و میانگین اعمال در هر کار**، نشان‌دهندهٔ زمان محاوره و کارایی کلی سیستم است.
- **تعداد اعمال، وظایف و وضعیت‌های سیستم که در مدل طراحی تعیین‌شده**، به بار حافظهٔ 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 کمّی را که به ارزیابی واسط کمک می‌کنند جمع‌آوری کند.

فصل نهم

طراحی در سطح مولفه

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 برنامهٔ مبدأ[1] است.

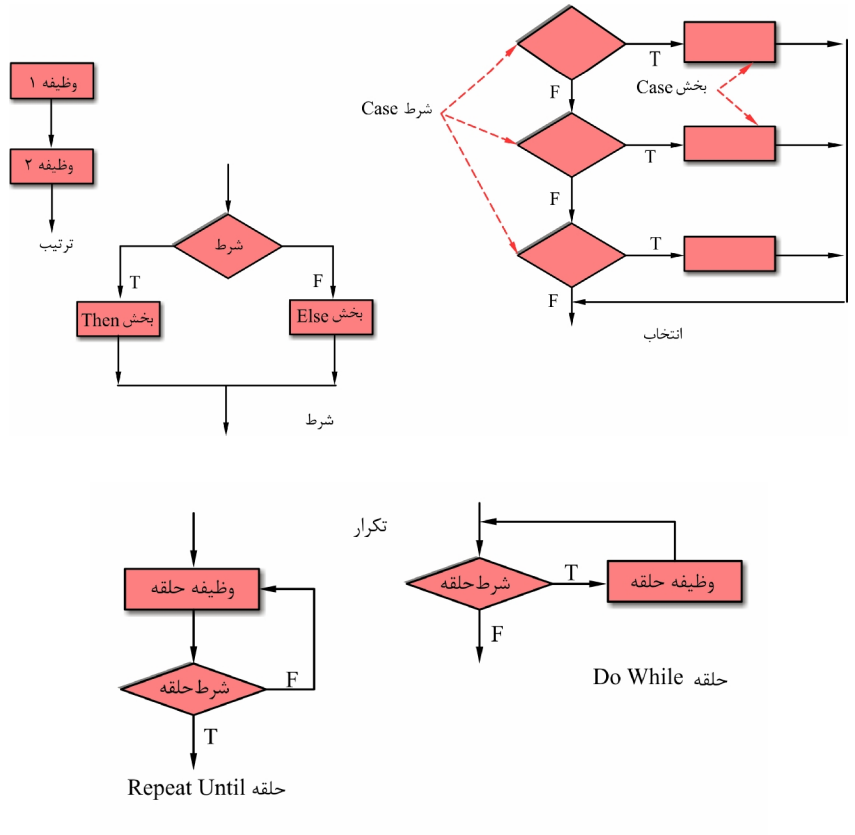
۹ـ۱ برنامه‌سازی ساخت‌یافته

استفاده از ساختارهای منطقی در دههٔ 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**.

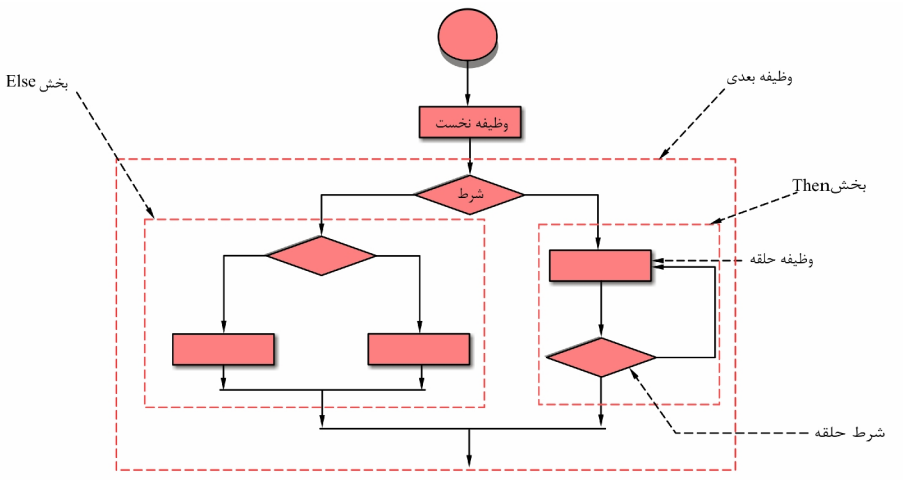
1 - Source Code

۹ـ۲ نشانه‌گذاری طراحی گرافیکی

هیچ تردیدی نیست که ابزارهای گرافیکی مانند **روندنما**[1] یا نمودارهای مستطیلی، الگوهای تصویری مفیدی هستند که به سادگی جزئیات رویه‌ای را مصوّر می‌سازند. در اشکال زیر نمونه‌هایی از ساختارهای مختلف نشان داده شده است.



شکل ۹ـ۱ ساختمان‌های توالی، شرط و تکرار در روندنما

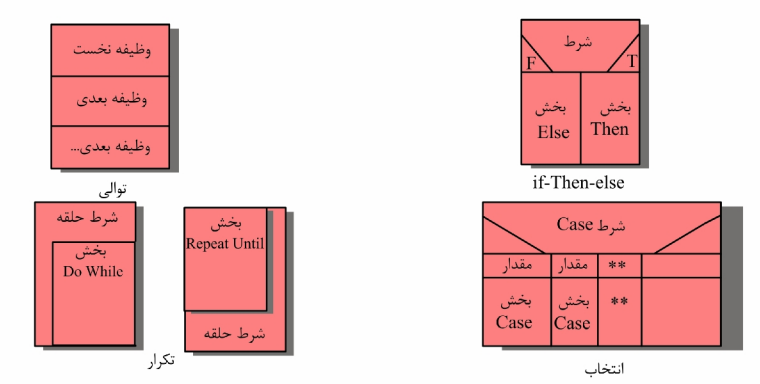


شکل ۹ـ۲ ساختمان‌های تو در تو در روندنما

1 - Flowchart

۹ـ۳ نمودار مستطیلی

نمایش گرافیکی سازه‌های ساخت‌یافته با استفاده از **نمودار مستطیلی** (یا جعبه‌ای یا نمودارهای ناسی- اشنایدرمن) 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 ۹ـ۳ ساختمان‌های نمودار مستطیلی

ویژگی‌های نمودارهای مستطیلی عبارت‌اند از:

- دامنهٔ 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.

۹ـ۴ علایم طراحی جدولی

در بسیاری از کاربردهای نرم‌افزاری، برای ارزیابی ترکیب پیچیدهٔ 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 ۹-۴ نمایش داده شده است.

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

شکل ۹ـ۴ 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 انجام‌شده در ارتباط با مجموعه شرایط، قوانین را تعریف کنید.

۹ـ۵ زبان توصیف برنامه (PDL)[1]

زبان توصیف برنامه (PDL) که زبان ساخت‌یافته یا شبه‌کد نیز نام دارد، یک زبان آمیخته است طوری که واژگان یک زبان (یعنی انگلیسی) و نحو کلی زبانی دیگر (یعنی زبان برنامه‌سازی ساخت‌یافته) را به کار می‌برد.

یک زبان توصیف باید ویژگی‌های زیر را داشته باشد:

- یک نحو ثابت از کلمات کلیدی که تمام سازه‌های ساخت‌یافته، تعاریف داده‌ها و ویژگی‌های پیمانه‌ای شدن را تهیه کند.
- یک نحو آزاد از زبان طبیعی که ویژگی‌های پردازشی را تشریح کند.
- تسهیلات تعریف داده‌ها که باید مشتمل بر ساختارهای داده‌ای ساده (اسکالر، آرایه) و پیچیده (لیست پیوندی یا درخت) باشد.
- تعریف زیربرنامه و فنون فراخوانی که حالات مختلف توصیف واسط را پشتیبانی کند.
- نمونه‌ای از کاربرد PDL برای سیستم خانهٔ 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
```

نشان‌گذاری طراحی باید به نمایش رویه‌ای منجر شود که درک و بررسی آن آسان باشد. به‌علاوه، نشان‌گذاری 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 نشان‌دهندهٔ برنامه باشد.

۹ـ۶ مقایسهٔ نشانه‌گذاری‌های طراحی

خصوصیات زیر در مورد نشانه‌گذاری طراحی تعیین شده‌اند:

- **قابلیت پیمانه‌ای**
- **سادگی همه‌جانبه**
- **سهولت ویرایش**
- **قابلیت خواندن سیستم**
- **قابلیت نگهداری**
- **تقویت ساختار**
- **پردازش خودکار**
- **بازنمایی داده‌ها**
- **تأیید منطق**
- **توانایی تبدیل به برنامه**

1 - Program Description Language

فصل دهم

اصول و قواعد تحلیل شیءگرا

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، تعجبی ندارد که برای توسعهٔ نرم‌افزارهای رایانه‌ای نیز دیدگاهی شیءگرا پیشنهاد شود.

۱۰ـ۱ تاریخچه

روش شیءگرا در توسعهٔ نرم‌افزار اولین‌بار در اواخر دههٔ ۱۹۶۰ به کار گرفته شد ولی ۲۰ سال طول کشید تا فن‌آوری شیءگرا به طور گسترده مورد استفاده قرار گیرد.

فن‌آوری‌های شیءگرا منجر به استفادهٔ مجدد می‌شود و استفادهٔ مجدد (از مؤلفه‌های برنامه) منجر به توسعهٔ 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. به‌علاوه، تطبیق دادن و تغییر دادن اندازهٔ سیستم‌های شیءگرا آسان‌تر است (یعنی سیستم‌های بزرگ را می‌توان با مونتاژ کردن زیرمؤلفه‌های قابل استفادهٔ مجدد ایجاد کرد (مقیاس‌پذیری).

۱۰ـ۲ 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 می‌دهند. برای مثال، اگر صفت مکان یک عنصر داده‌ای مرکب به صورت زیر باشد:

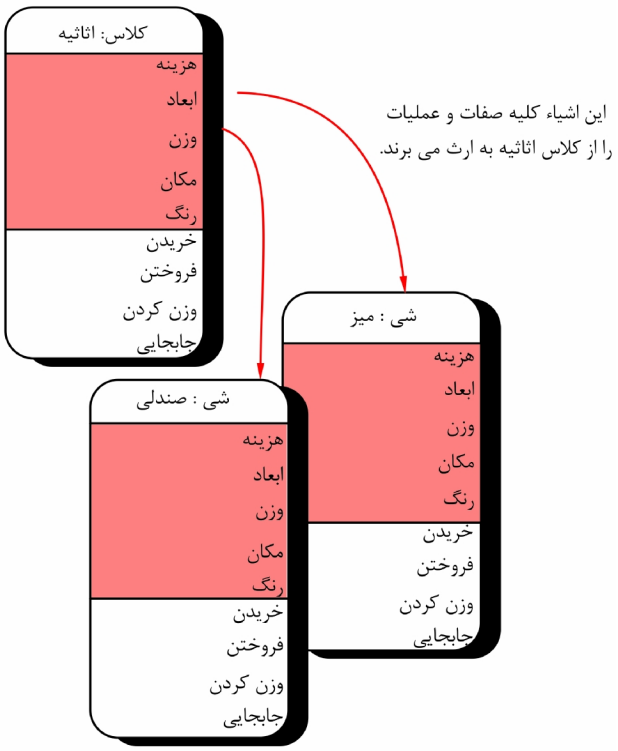
اتاق + طبقه + ساختمان= مکان

در این صورت، عملی که جابه‌جایی نام دارد، یک یا چند مورد از این عناصر داده‌ای (ساختمان، طبقه، اتاق) که مکان شیء را تشکیل می‌دهند، تغییر می‌دهد. 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 می‌توان برای آن‌ها نیز استفاده کرد. همهٔ عملیات معتبر (مثل خریدن، فروختن، توزیع کردن) برای کلاس اثاثیه به تعریف شیء **متصل** هستند و برای تمام نمونه‌های این کلاس به ارث گذاشته می‌شوند.

شیء صندلی (و کلاً همهٔ اشیا)، **داده‌ها** (مقادیر صفاتی که صندلی را تعریف می‌کنند)، **عملیات** (کاری که برای تغییر دادن صفات صندلی به کار می‌روند)، **اشیای دیگر** (اشیای مرکبی که قابل تعریف هستند)، **ثابت‌ها** (مقادیر ثابت) و **اطلاعات** مربوطه دیگر را **بسته‌بندی** می‌کنند. بسته‌بندی بدان معناست که تمامی این اطلاعات تحت یک نام بسته‌بندی شوند و به عنوان یک مشخصه یا قطعهٔ برنامه به کار برده شوند.

Coad و Yourdon تعریف رسمی از شیءگرایی را به صورت زیر ارائه داده‌اند:

ارتباطات + وراثت + طبقه‌بندی+ اشیا= شیءگرایی



شکل ۱۰ـ۱ وراثت بین کلاس‌ها

۱۰ـ۲ـ۱ کلاس‌ها و اشیا

کلاس یک مفهوم شیءگرایی است که داده‌ها و رویه‌هایی را که برای توصیف محتوا[1] و رفتار[2] یک موجودیت دنیای واقعی لازم است، بسته‌بندی می‌کند.

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

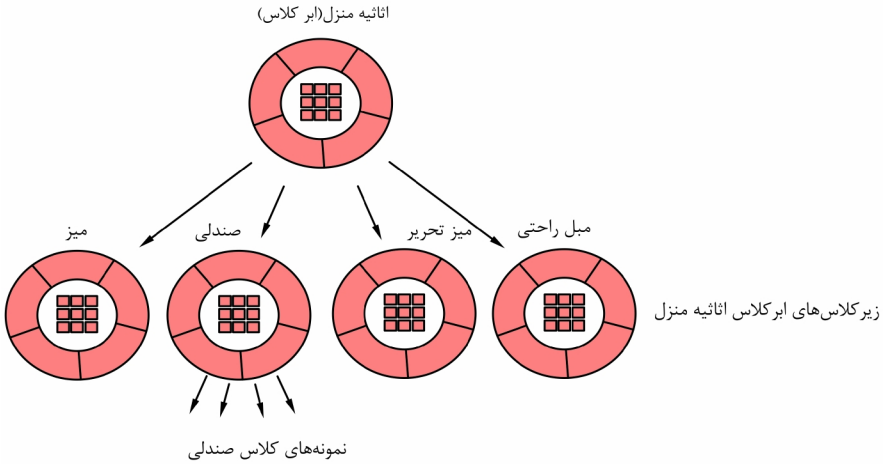
1 - Content
2 - Behaviour

تنها راه برای دسترسی به صفات از طریق یکی از متدهاست (کلاس، داده‌ها و متدها را **بسته‌بندی** می‌کند). این کار باعث **پنهان‌سازی اطلاعات**[1] می‌شود و خود منجر به کمتر شدن تأثیر اثرات جانبی مرتبط با تغییرات می‌شود.

ازآنجاکه متدها تعداد محدودی از صفات را تغییر می‌دهند، آن‌ها **منسجم**[2] هستند و به خاطر اینکه ارتباطات بین کلاس‌ها فقط از طریق متدها رخ می‌دهد، هر کلاس از سایر اجزای سیستم **جدا**[3] می‌شود. این خصوصیات منجر به شکل‌گیری یک **نرم‌افزار با کیفیت بالا**[4] می‌شود.

- **سلسله‌مراتبی کلاس‌ها**

ابرکلاس[5] مجموعه‌ای از کلاس‌هاست و **زیرکلاس**[6] یک نمونه خاص از یک کلاس است. این تعاریف دلالت بر وجود **سلسله‌مراتب کلاس**[7] 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 ۱۰ـ۲ نمایشی از سلسه‌مراتب کلاس‌ها

۱۰ـ۲ـ۲ پیغام‌ها[8]

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:

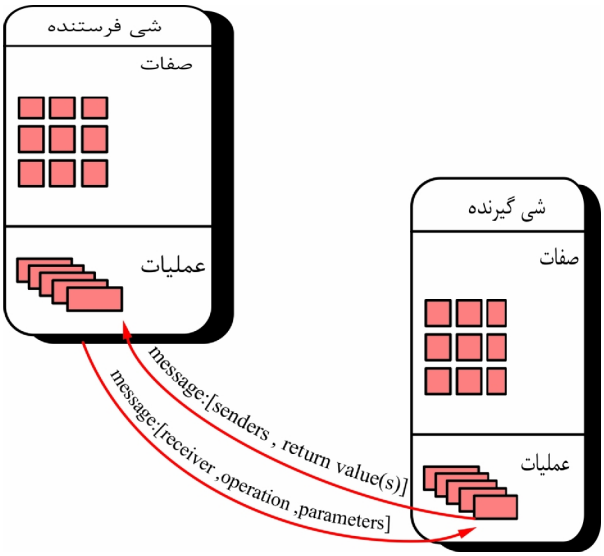
پیغام : [مقصد، متد، پارامترها]

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 ۱۰ـ۳ مبادله پیام‌ها بین دو شیء

۱۰ـ۳ ویژگی‌های سیستم‌های شیءگرا

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 منحصربه‌فرد می‌سازد.

۱۰ـ۳ـ۱ بسته‌بندی[1]

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 **استفادهٔ مجدد از مؤلفه**[2] راحت‌تر شود.
- واسطهای بین اشیای بسته‌بندی شده ساده‌تر می‌شوند. شیء فرستندهٔ یک پیغام، لازم نیست که به جزئیات داخلی ساختارهای داده توجه کند. این کار باعث می‌شود که **اتصال** سیستم کاهش یابد.

۱۰ـ۳ـ۲ وراثت[3]

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

۱۰ـ۳ـ۳ چندریختی[1]

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

۱۰ـ۴ شناسایی عناصر یک مدل شیءگرا

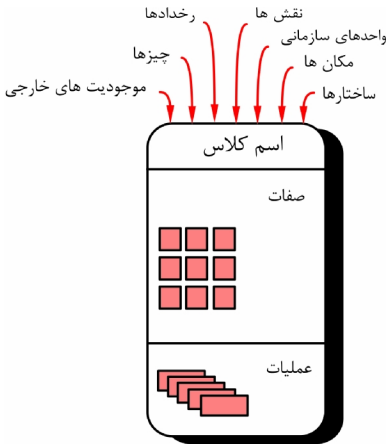
کلاس‌ها، اشیا، صفات، عملیات و پیغام‌ها عناصر مدل تحلیل شیءگرا هستند. حال ببینیم چگونه این عناصر را باید شناسایی کرد.

۱۰ـ۴ـ۱ شناسایی کلاس‌ها 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 پردازش‌های سیستمی که قرار است ساخته شود، آغاز می‌کنیم. برای تعیین اشیا، زیر اسم‌ها یا عبارات اسمی خط کشیده، آن‌ها را در جدولی لیست می‌کنیم. به اسم‌های مترادف نیز باید توجه داشت. اشیا می‌توانند به یکی از طرق زیر خود را نشان دهند:

- **موجودیت‌های خارجی** (مثلاً سیستم‌های دیگر، دستگاه‌ها، افراد) که اطلاعات مورد استفاده سیستم رایانه‌ای را تولید یا مصرف می‌کنند.
- **چیزهایی** (مانند گزارش‌ها، صفحات نمایش، نامه‌ها، علایم الکترونیکی) که بخشی از دامنهٔ اطلاعاتی مسئله به شمار می‌روند.
- **رخدادها یا وقایعی** (مثلاً کامل شدن حرکات روبات) که در حیطهٔ عملکرد سیستم رخ می‌دهند.
- **نقش‌هایی** (مثل مدیر، مهندس و فروشنده) باشند که افرادِ در حال تعامل با سیستم بر عهده دارند.
- **واحدهای سازمانی** (مثل بخش، گروه و تیم) که با یک کاربرد خاص سر و کار دارند.
- **مکان‌هایی** (مثل سالن تولید) که حیطه مسئله و وظیفهٔ کلی سیستم را مشخص می‌کنند.
- **ساختارهایی** (مثل حسگرها، وسایل نقلیه یا رایانه‌ای) که کلاسی از اشیا یا کلاس‌های مرتبطی از اشیا را تعریف می‌کنند.

توجه به این نکته نیز مهم است که اشیا چه چیزهایی نیستند. به طور کلی یک شیء هیچ‌گاه نباید یک **"نام رویه‌ای دستوری"** داشته باشد. تجزیه گرامری را می‌توان برای تفکیک اشیا (اسامی) و عملیات به کار برد.



شکل ۱۰ـ۴ چگونگی شناسایی اشیا

Coad و Yourdon شش خصوصیت گزینشی ارائه داده‌اند که تحلیل‌گر باید در مورد هر شیء بالقوه‌ای که در مدل تحلیل به کار می‌برد در نظر بگیرد:

- **اطلاعات ذخیره‌شده:** شیء بالقوه فقط در صورتی هنگام تحلیل مفید خواهد بود که اطلاعات آن برای عملکرد سیستم لازم باشد.
- **خدمات مورد نیاز:** شیء بالقوه باید مجموعه‌ای از عملیات قابل شناسایی داشته باشد که می‌توانند 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.

1 - Polymorphism

- **صفات مشترک:** مجموعه‌ای از صفات را می‌توان برای یک شیء بالقوه تعریف کرد که این صفت‌ها 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.

۱۰ـ۴ـ۲ مشخص کردن صفات

- صفات، اشیا را طوری تعریف و توصیف می‌کنند که برای ورود به مدل تحلیل انتخاب شده‌اند.
- برای توسعهٔ 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؟"

۱۰ـ۴ـ۳ 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 می دهند.
- عملیاتی که برای رخ دادن یک رویداد کنترلی بر شیء نظارت می‌کنند.

۱۰ـ۴ـ۴ نهایی‌سازی تعریف اشیا

تعریف عملیات، آخرین مرحله در کامل کردن تشخیص اشیاست. عملیات اضافه‌ای را می‌توان با در نظر گرفتن **تاریخچهٔ حیات** یک شیء و پیغام‌هایی که در میان اشیای تعریف‌شده در سیستم مبادله می‌شوند، تعیین کرد. تاریخچهٔ 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

مدل‌سازی تحلیل شیءگرا

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:

۱ـ دامنه اطلاعاتی مدل‌سازی می‌شود.

۲ـ عملکرد توصیف می‌شود.

۳ـ رفتار نمایش داده می‌شود.

۴ـ مدل‌های داده‌ای، عملیاتی و رفتاری افراز می‌شوند تا جزئیات بیشتری در معرض دید قرار گیرند.

۵ـ 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:

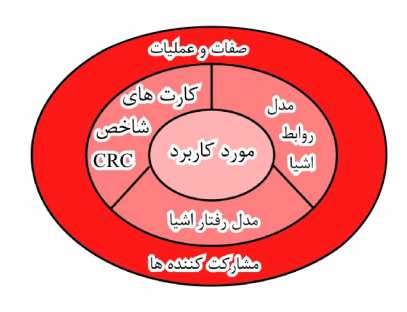
- خواسته‌های اصلی کاربر باید بین مشتری و مهندس تبادل شود.
- کلاس‌ها باید شناسایی شوند.
- سلسله‌مراتب کلاس‌ها باید مشخص شود.
- روابط شیء با شیء باید نشان داده شود.
- رفتار اشیا نیز باید مدل‌سازی شود.

گرچه مراحل، اصطلاح‌شناسی و فرایندهای هر یک از روش‌های OO متفاوت است، ولی فرایندهای کلی تحلیل شیءگرا بسیار مشابه‌اند. مهندس نرم‌افزار برای اجرای تحلیل شیءگرا 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 اشیای سیستمی
- تعریف ساختارها و سلسله مراتبی 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 شیءگرا به صورت زیر خواهد بود:



شکل ۱۱ـ۱ مدل تحلیل نیازهای شیءگرا

۱۱ـ۱ زبان مدل‌سازی یکنواخت

زبان یکنواخت و یکپارچه مدل‌سازی (UML)[1] زبان مدل‌سازی گرافیکی است که با ترکیبی از نمادها به تشریح عناصر اصلی سیستم‌های نرم‌افزاری (که در UML به آن محصول گفته می‌شود) می‌پردازد. UML شامل سه گروه مدل به شرح زیر است:

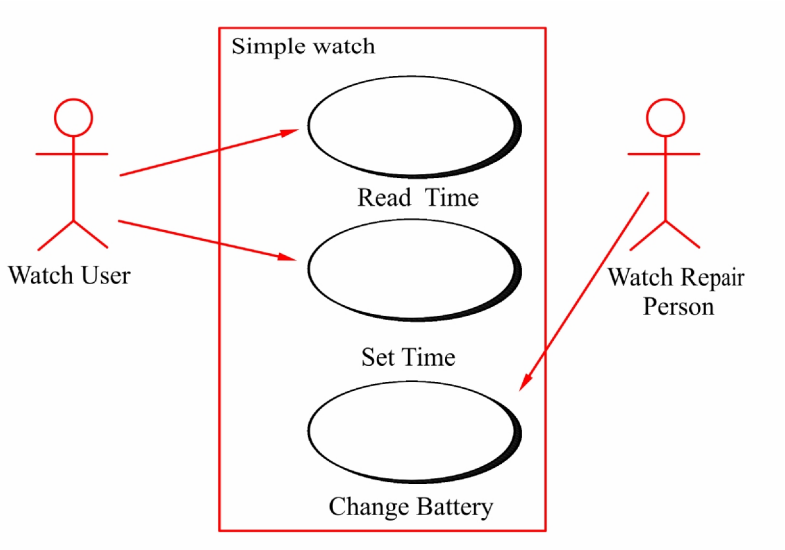
- **مدل تابعی**[2]: عملکرد سیستم را از دید کاربر شرح می‌دهد. در UML برای این مدل‌سازی از نمودارهای Use-Case استفاده می‌شود.
- **مدل شیء**[3]: ساختار سیستم را به صورت مجموعه‌ای از اشیا، صفات، عملیات و ارتباطات شرح می‌دهد. در UML برای این مدل‌سازی از نمودار کلاس (Class Diagram) استفاده می‌شود.
- **مدل پویا**[4]: رفتار داخلی سیستم را شرح می‌دهد. در UML برای این مدل‌سازی از نمودارهای Statechart،Sequence و Activity استفاده می‌شود.

۱۱ـ۱ـ۱ نمونه‌ای از نمودار مورد کاربرد

نمودار مورد کاربرد[5] در طول استخراج نیازها و تحلیل سیستم برای مشخص کردن عملکرد سیستم ایجاد می‌شود. Use Caseها بر رفتار سیستم از دیدگاه خارج از سیستم تمرکز می‌کنند. یک Use Case درواقع 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 User و Watch Repair Person دو بازیگر هستند که با این سیستم در تعامل هستند.

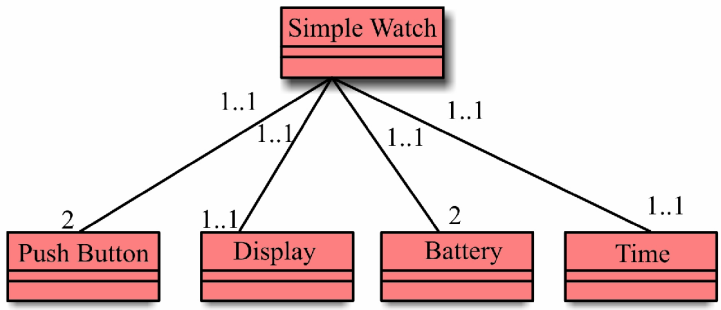
1 - Unified Modelling Language
2 - Functional Model
3 - Object Model
4 - Dynamic Model
5 - Use Case



شکل ۱۱ـ۲ نمودار مورد کاربرد سیستم ساعت مچی

۱۱ـ۱ـ۲ نمونه‌ای از نمودار کلاس

از نمودار کلاس[1] 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.

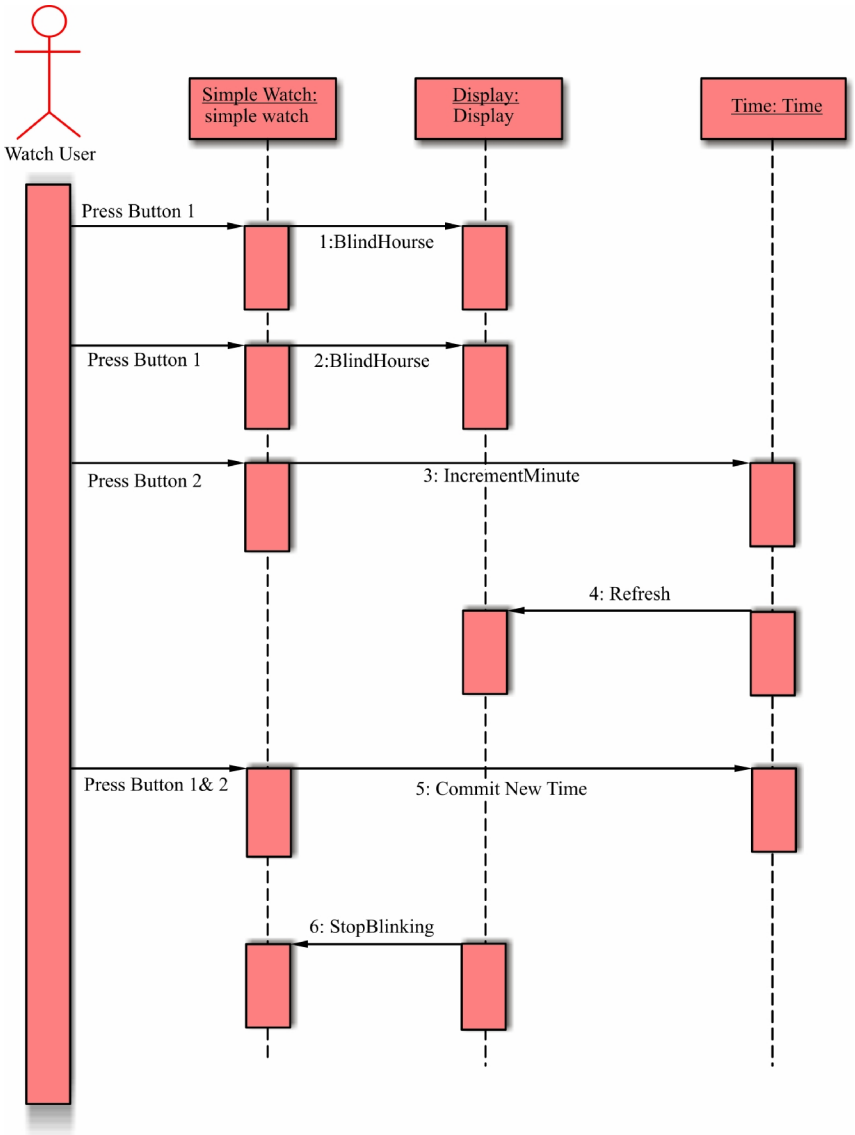


شکل ۱۱ـ۳ نمودار سلسله‌مراتبی کلاس‌های سیستم ساعت مچی

۱۱ـ۱ـ۳ نمونه‌ای از نمودار توالی

نمودار توالی[2] برای فرموله کردن رفتار سیستم و همچنین رخ دادن ارتباطات بین اشیا به کار می‌رود. نمودار توالی برای نشان دادن اشیایی که در یک Use Case سهیم هستند مفید است. به همین دلیل به اشیایی که در یک نمودار توالی نشان داده می‌شوند، اشیای شرکت‌کننده[3] می‌گویند.

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



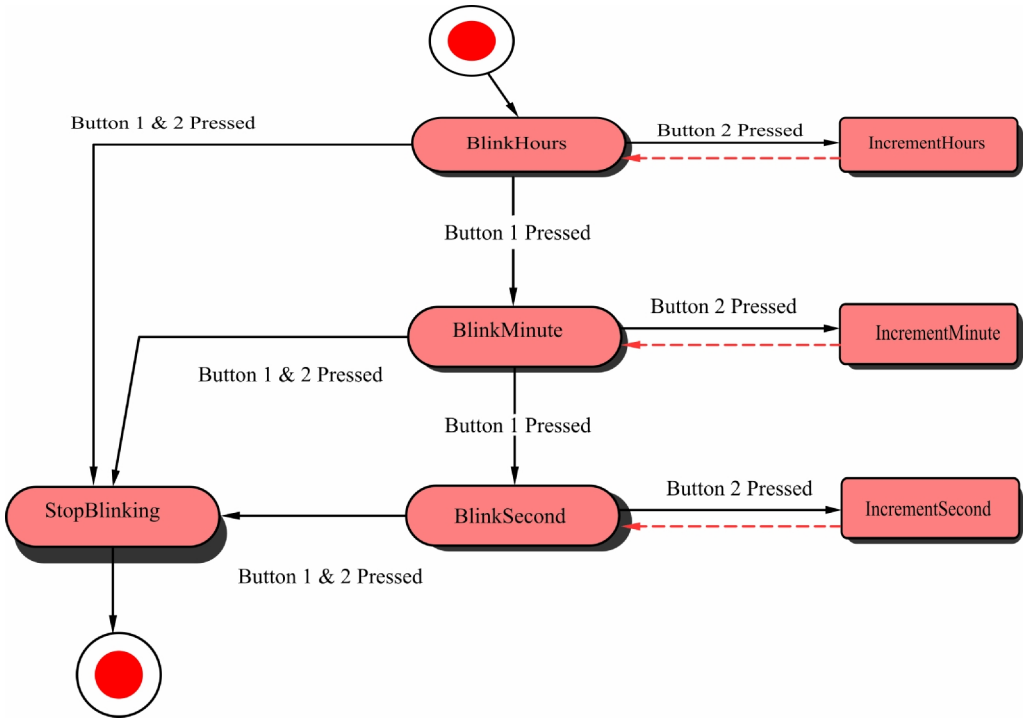
شکل ۱۱ـ۴ نمودار توالی سیستم ساعت مچی

۱۱ـ۱ـ۴ نمونه‌ای از نمودار حالت

نمودار حالت[1]، 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 پیغام‌هایی است که تحت تأثیر یک رویداد رخداده 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.

1 - StateChart Diagram



شکل ۱۱ـ۵ نمودار حالت سیستم ساعت مچی

۱۱ـ۲ فرایند تحلیل شیءگرا

فرایند تحلیل شیءگرا (OOAP)[1] 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:

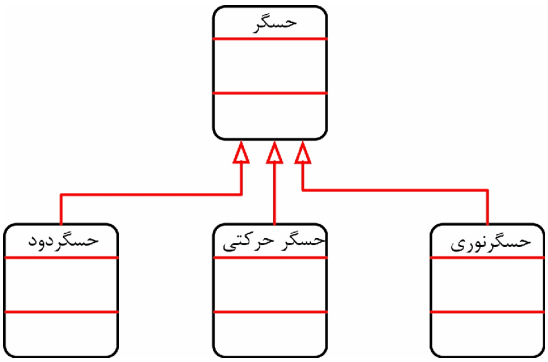
- طراحی موردهای کاربرد[2]
- مدل‌سازی مسئولیت و مشارکت کلاس‌ها[3]
- تعریف ساختارها، سلسله‌مراتب[4] کلاس‌ها
- مدل‌سازی روابط بین اشیا
- ساخت مدل رفتار اشیا
- تعریف موضوع‌ها و زیرسیستم‌ها[5]

۱۱ـ۲ـ۳ تعریف ساختار و سلسله‌مراتب کلاس‌ها

- **ساختار کلاس تعمیم- تخصیص**[6]

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

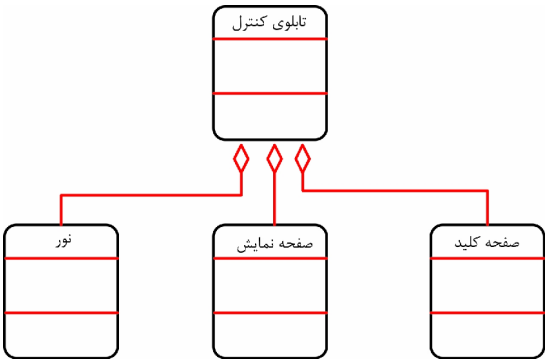
1 - Object Oriented Analysis Process
2 - Use-Cases Design
3 - Class-Responsibility-Collaborator Modelling
4 - Defining Structurers and Hierachies
5 - Defining Subjects and Subsystems
6 - Generalization- Specialization



شکل ۱۱ـ۶ نمودار کلاس‌ها برای تعمیم- تخصیص

- **ساختار کلاس تجمع**[1]

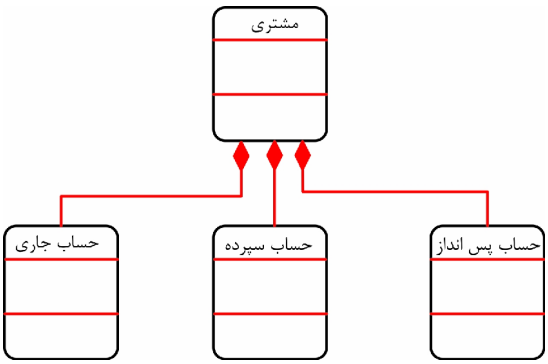
در این ساختار یک شیء از چند مؤلفه تشکیل شده است که این مؤلفه‌ها را می‌توان به عنوان یک شیء تعریف کرد. مثال:



شکل ۱۱ـ۷ نمودار کلاس‌ها برای کلاس‌های مجتمع مرکب

- **ساختار کلاس ترکیب**[2]

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 ۱۱ـ۸ نمودار کلاس‌ها برای کلاس‌های ترکیبی

1 - Is A Part Of
2 - Composition

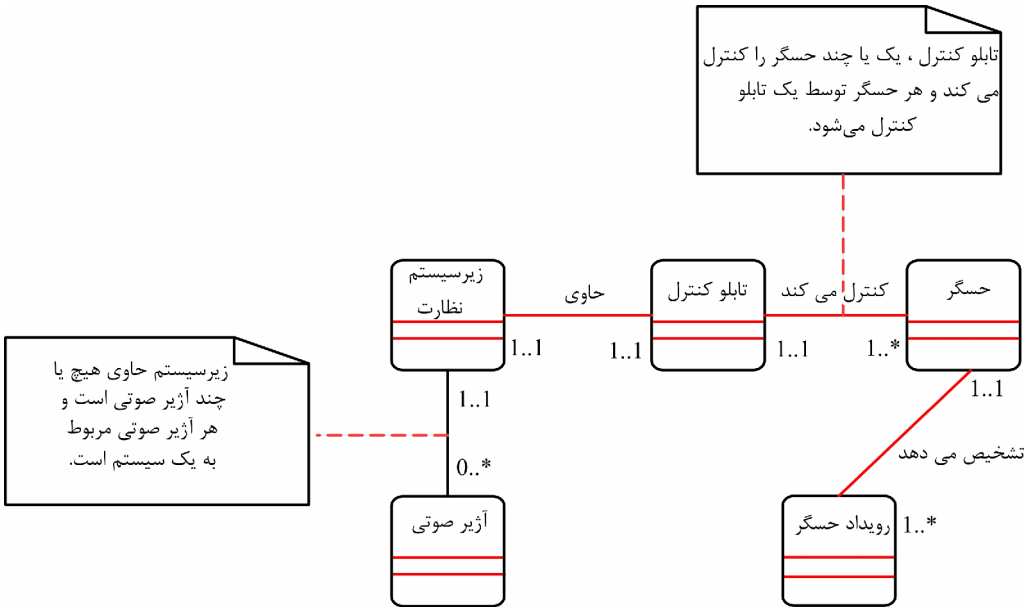
۱۱ـ۲ـ۴ مدل روابط بین اشیا

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

۱ـ با استفاده از کارت‌های شاخص مدل CRC، مجموعه‌ای از اشیای مشارکت‌کننده رسم می‌شوند.

۲ـ با بازبینی کارت‌های شاخص مدل CRC، مسئولیت‌ها و مشارکت‌کننده‌ها مورد ارزیابی قرار گرفته و ارتباطات نامگذاری می‌شوند.

۳ـ هنگامی که روابط با نام مشخص شد، نحوهٔ مشارکت کلاس‌ها در ارتباط، ارزیابی و مدل نهایی به دست می‌آید.



شکل ۱۱ـ۹ نمودار روابط بین اشیا

۱۱ـ۲ـ۵ مدل رفتار اشیا[1]

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**

۱ـ ارزیابی تمام Use-Case ها برای درک کامل ترتیب تعامل‌ها در سیستم

۲ـ شناسایی رویدادهایی که ترتیب تعامل‌ها را هدایت می‌کنند و درک چگونگی ارتباط این رویدادها با اشیای خاص

۳ـ ایجاد یک پیگرد رویداد برای هر Use-Case

۴ـ ساختن نمودار توالی و حالت

۵ـ بازبینی مدل رفتار اشیا برای اعتبارسنجی، صحت و سازگاری

۱۱ـ۲ـ۶ 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[2] می‌نامند.

1 - Object –Behavior Model
2 - Package

فصل دوازدهم

طراحی شیءگرا

برخلاف مدل‌های طراحی نرم‌افزار سنتی، OOD منجر به طرحی می‌شود که شامل سطوح متفاوتی از پیمانه‌ها است. مؤلفه‌های سیستم اصلی به صورت زیرسیستم یا **پیمانه** سطح سیستمی سازماندهی می‌شوند. داده‌ها و عملیاتی که داده‌ها را دستکاری می‌کنند، در اشیا(یک شکل پیمانه‌ای که مؤلفهٔ 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.

۱۲ـ۱ لایه‌های طراحی شیءگرا

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 شیءگرا نیز می‌توان یک هرم طراحی تعریف کرد، ولی در اینجا لایه‌های متناظر قدری تفاوت دارند. چهار لایه هرم طراحی شیءگرا عبارت‌اند از:

لایهٔ 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.

لایهٔ 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 کلیهٔ اشیا است.

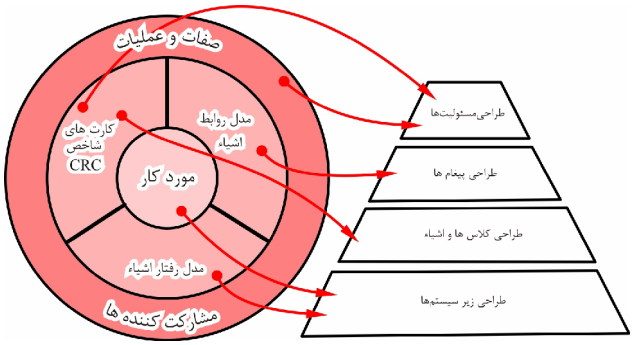
لایهٔ پیغام‌رسانی: شامل جزئیاتی از طراحی است که هر کلاس را قادر به برقراری ارتباط با مشارکت‌کننده‌های خود می‌سازد. این لایه واسط‌های داخلی و خارجی را برای سیستم ایجاد می‌کند.

لایهٔ 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 لایهٔ دیگری از طراحی نیز وجود دارد که **لایهٔ مبنا**یی نامیده شده و هرم بر آن استوار است. **لایهٔ مبنا** بر **طراحی اشیای دامنه** تأکید دارد. اشیای 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. اشیای دامنه را می‌توان برای افزودن اطلاعات بیشتر به برنامهٔ کاربردی نیز به کار برد.

۱۲ـ۲ تبدیل مدل تحلیل شیءگرا به مدل طراحی شیءگرا

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



شکل ۱۲ـ۱ رابطه بین مدل‌های تحلیل و طراحی شیءگرا

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

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

۱۲ـ۳ روش یکنواخت برای طراحی شیءگرا

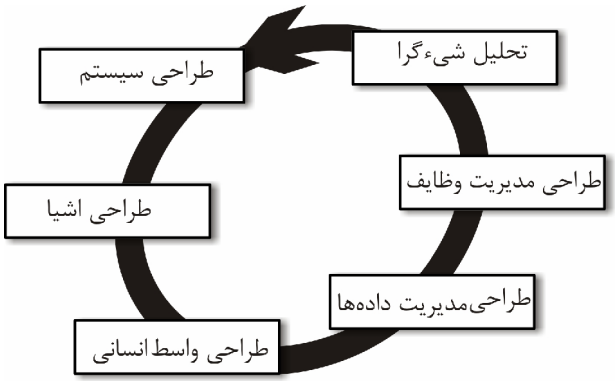
Grady، Booch، Rumbough و Jacobson بهترین ویژگی‌های روش‌های تحلیل و طراحی شیءگرا را کنار هم قرار دادند و روشی یکنواخت حاصل شد. نتیجهٔ کار که زبان مدل‌سازی یکنواخت (UML)[1]، خوانده می‌شود، در سرتاسر صنعت نرم‌افزار، کاربردی گسترده یافته است.

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.

1 - Unified Modeling Language



شکل ۱۲ـ۲ جریان فرایند طراحی شیءگرا

۱۲ـ۴ فرایند طراحی شیءگرای سیستم

فرایند طراحی شیءگرای 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 ازجمله مدیریت وظایف
- در نظر گرفتن چگونگی پرداختن به شرایط مرزی
- بازبینی و بازنگری مدل طراحی

۱۲ـ۵ فرایند طراحی اشیا

طراحی شیءگرا را با توجه به استعاره‌ای که در این متن به کار گرفته شد، می‌توان به عنوان نقشهٔ طرح یک ساختمان در نظر گرفت. نقشهٔ 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 حیطهٔ طراحی شیءگرا، **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 نشان‌دهندهٔ 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 دنبالهٔ 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. ازآنجاکه 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 نرم‌افزار و راهبردها

آزمایش نرم‌افزار عنصری حیاتی از تضمین کیفیت نرم‌افزار است و بیانگر مرور و بازنگری نهایی مشخصهٔ سیستم، تحلیل، طراحی و برنامه‌های مبدأ است.

۱۳ـ۱ شیوهٔ راهبردی آزمایش نرم‌افزار

آزمایش، شامل مجموعه‌ای از فعالیت‌ها است که می‌تواند از قبل به صورت سیستماتیک برنامه‌ریزی و هدایت شود.

راهبردهای آزمایش نرم‌افزار همگی خصوصیات زیر را دارند:

- آزمایش از سطح مؤلفه شروع می‌شود، به سمت خارج در جهت یکپارچه‌سازی کل سیستم رایانه‌ای پیش می‌رود.
- روش‌های متفاوت آزمایش، در نقاط زمانی مختلف مناسب هستند.
- آزمایش توسط توسعه‌دهندهٔ نرم‌افزار و برای پروژه‌های بزرگ توسط گروه مستقل آزمایش، هدایت می‌شود.
- آزمایش و اشکال‌زدایی فعالیت‌های متفاوتی هستند، اما اشکال‌زدایی باید با هر راهبرد آزمایش همراه باشد.

یک راهبرد برای آزمایش نرم‌افزار باید آزمایش‌های سطح پایینی را هدایت کند که برای بازبینی صحت پیاده‌سازی یک قطعه کُد کوچک لازم هستند. همچنین این راهبرد باید آزمایش‌های سطح بالایی را سازماندهی کند که اکثر توابع سیستم را در رابطه با نیازهای مشتری اعتبارسنجی می‌کنند. یک راهبرد باید رهنمودهایی را برای مجری و مجموعه‌ای از علایم نشان‌دهنده را برای مدیر فراهم کند.

۱۳ـ۲ راهبرد آزمایش نرم‌افزار

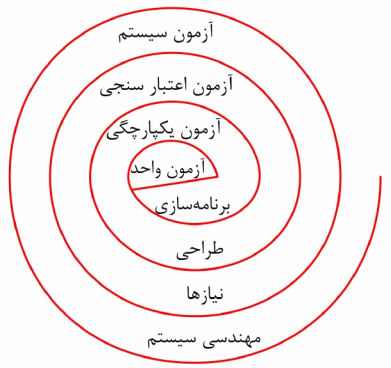
فرایند مهندسی نرم‌افزار را می‌توان به صورت یک مارپیچ مانند شکل زیر در نظر گرفت. در ابتدا، مهندس سیستم، نقش نرم‌افزار را تعریف می‌کند و به تحلیل نیازهای نرم‌افزار وارد می‌شود که دامنهٔ اطلاعات، عملکرد، رفتار، کارایی، محدودیت‌ها و معیارهای اعتبارسنجی برای نرم‌افزار ایجاد می‌شود. با حرکت در مارپیچ به سمت طراحی و درنهایت به برنامه‌نویسی می‌رسیم. به منظور توسعهٔ نرم‌افزار، به سمت درون مارپیچ حرکت می‌کنیم تا در هر دور، سطح انتزاع کاسته شود.

یک راهبرد برای آزمایش نرم‌افزار می‌تواند در این مارپیچ مورد توجه قرار گیرد. **آزمایش واحد،** از مرکز مارپیچ شروع می‌شود و بر روی هر واحد (یعنی مؤلفه) نرم‌افزار متمرکز می‌شود که با برنامه‌های مبدأ پیاده‌سازی شده است. با حرکت به سمت خارج در مارپیچ، آزمایش به سمت

آزمایش **یکپارچه‌سازی** ادامه می‌یابد. این آزمایش بر طراحی و ساخت معماری نرم‌افزار تأکید دارد. با حرکت به اندازه یک دور دیگر به سمت خارج مارپیچ، به آزمایش **اعتبارسنجی** می‌رسیم.

نیازهایی که به عنوان بخشی از تحلیل نیازهای نرم‌افزار ایجاد شده‌اند، در مقابل نرم‌افزاری که ساخته شده اعتبارسنجی می‌شوند. درنهایت باید آزمایش **سیستم** مورد توجه قرار گیرد که در آن نرم‌افزار و عناصر دیگر سیستم به صورت یک مجموعهٔ یکپارچه، آزمایش می‌شوند. برای آزمایش نرم‌افزار، در طول مارپیچ حرکت کرده و در هر دور، محدودهٔ 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 جعبهٔ سفید** بهره می‌برد و مسیرهای خاصی را در ساختار کنترلی پیمانه بررسی می‌کند تا از صحت آن اطمینان حاصل شود.



شکل ۱۳ـ۱ فرایند راهبردی آزمایش نرم‌افزار

سپس مؤلفه‌ها باید مونتاژ یا یکپارچه شوند تا بستهٔ نرم‌افزاری کامل را تشکیل دهند. آزمایش یکپارچه‌سازی، مسایل مربوط به مشکلات دوگانهٔ بازبینی و ساخت برنامه را مورد توجه قرار می‌دهد. تکنیک‌های طراحی نمونه‌های آزمایش **جعبهٔ سیاه**، در زمان یکپارچه‌سازی بیشترین استفاده را دارند.

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

آخرین مرحلهٔ آزمایش مرتبهٔ بالا، خارج از مرز مهندسی نرم‌افزار است و در چارچوب مهندسی سیستم رایانه‌ای مطرح است. نرم‌افزار، پس از اعتبارسنجی، باید با عناصر دیگر سیستم ترکیب شود (برای مثال، سخت‌افزار، 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.

۱۳ـ۳ مبانی آزمایش نرم‌افزار

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

۱۳ـ۳ـ۱ اهداف آزمایش

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

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

۱۳ـ۳ـ۲ اصول آزمایش

قبل از به‌کارگیری 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[1] 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.

۱۳ـ۳ـ۳ 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**[2]: "نرم‌افزار هرچه بهتر کار کند، با کارایی بالاتری آزمایش می‌شود."
- **قابلیت مشاهده**[3]: "آنچه می‌بینید همان است که آزمایش می‌کنید."
- **قابلیت کنترل**[4]: "هرچه نرم‌افزار بهتر کنترل شود، آزمایش بیشتر به طور خودکار و بهینه‌پذیر انجام‌پذیر است."
- **تجزیه‌پذیری**[5]: "با کنترل دامنهٔ کاربرد آزمایش، می‌توان مسایل را با سرعت بیشتری تجزیه کرد و آزمایش‌های مجدد را با هوشمندی انجام داد."
- **سادگی**[6]: "هرچه مورد برای آزمایش کمتر باشد، آزمایش با سرعت بیشتری انجام می‌گیرد."
- **پایداری**[7]: "هرچه تغییرات کمتر باشد، انحراف از آزمایش کمتر است."
- **قابلیت فهم**[8]: "هرچه اطلاعات بیشتری در اختیار داشته باشیم، آزمایش هوشمندانه‌تر انجام می‌شود."

۱۳ـ۳ـ۴ طراحی نمونه‌های آزمایش

طراحی نمونه آزمایش‌هایی برای نرم‌افزار و محصولات مهندسی دیگر می‌تواند به اندازهٔ طراحی اولیهٔ خود محصول متغیر باشد. با این وجود، مهندسین 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:

۱ـ 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 درعین‌حال در هر تابع برای یافتن خطاها جستجو صورت می‌گیرد (**آزمایش جعبه سیاه**).

۲ـ با دانستن عملکرد داخلی محصول، آزمایش‌ها به‌گونه‌ای طراحی می‌شوند که تعیین کنند اعمال داخلی مطابق با مشخصه‌های تعریف‌شده انجام می‌شوند و تمام مؤلفه‌های داخلی به شکل مناسبی آزمایش می‌شوند (**آزمایش جعبهٔ سفید**).

---

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

۱۳ـ۴ آزمایش جعبهٔ سفید

آزمایش جعبهٔ سفید که گاهی آزمایش **جعبهٔ 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 جعبهٔ سفید، مهندس نرم‌افزار می‌تواند نمونه‌های آزمایشی را طراحی کند که:

۱ـ تضمین کنند تمام مسیرهای مستقل در پیمانه حداقل یک‌بار آزمایش می‌شوند.

۲ـ تمام تصمیمات شرطی را در دو بخش درست و غلط بررسی کنند.

۳ـ تمام حلقه‌ها را در شرایط مرزی و در محدوده‌های عملیاتی اجرا کنند.

۴ـ ساختمان داده‌های داخلی را بررسی کنند تا از اعتبار آن‌ها مطمئن شوند.

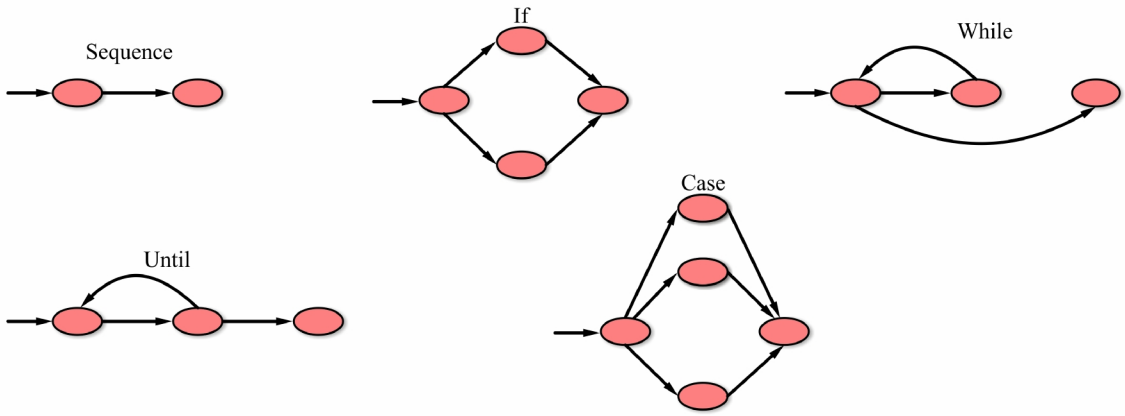
آزمایش جعبه سفید مرتبط با دو بحث **مسیر پایه و گراف جریان** است.

۱۳ـ۴ـ۱ آزمایش مسیر پایه

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

۱۳ـ۴ـ۲ گراف جریان

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



شکل ۱۳ـ۲ نشانه‌گذاری گراف جریان

۱۳ـ۴ـ۳ پیچیدگی دورانی

پیچیدگی دورانی به یکی از سه شکل زیر محاسبه می‌شود:

۱ـ تعداد نواحی گراف جریان

۲ـ با استفاده از تعداد یال‌ها و گره‌ها

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

$$V(G) = E - N + ۲$$

۳ـ با استفاده از گره‌های گزاره‌ای

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

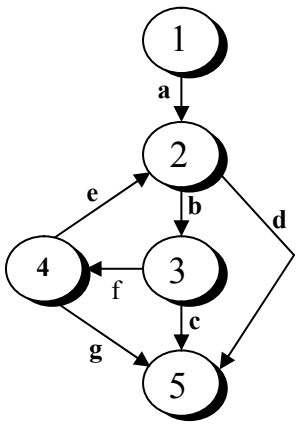
$$V(G) = P + ۱$$

۱۳ـ۴ـ۴ 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 توسعهٔ 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. به‌هرحال 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 ۱۳ـ۳ ماتریس گراف متناظر با گراف جریان

۱۳ـ۴ـ۵ 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.

۱۳ـ۴ـ۵ـ۱ آزمایش شرط

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

E_1 < **عملگر رابطه‌ای** < E_2

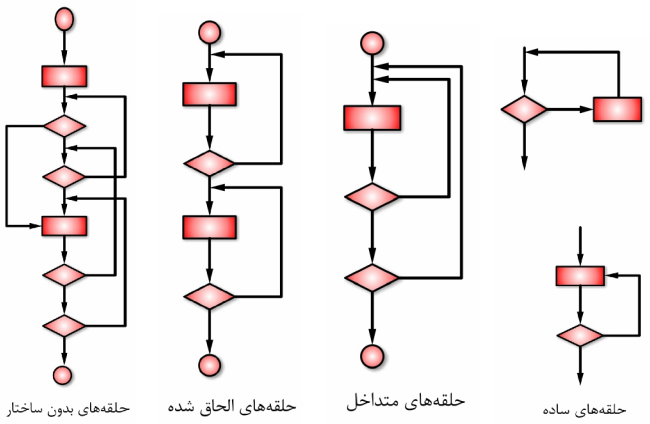
که E_1 و E_2 عبارت‌های محاسباتی هستند و < عملگر رابطه‌ای> شامل یکی از عملگرهای $>$ ، $\leq$ ، $=$ $\neq$ (نامساوی)، $<$ یا $\geq$ است. یک شرط مرکب از دو یا چند شرط ساده، عملگر بولی و پرانتزها تشکیل شده است. فرض می‌کنیم عملگرهای بولی امکان شرط‌های ترکیبی را با اضافه کردن OR (|)، AND (&) و NOT (¬) می‌دهند.

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/ حذف‌شده/ عملگرهای بولی اضافی)
- خطای متغیر بولی
- خطای پرانتزهای بولی
- خطای عملگر رابطه‌ای
- خطای عبارت محاسباتی

۱۳ـ۴ـ۵ـ۲ 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 ۱۳ـ۴ انواع حلقه‌ها

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

۱۳ـ۵ آزمایش جعبهٔ سیاه

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

۱ـ توابع غلط یا حذف‌شده

۲ـ خطاهای واسط‌ها

۳ـ خطا در ساختمان داده‌ها یا دسترسی به بانک اطلاعاتی خارجی

۴ـ خطاهای رفتاری یا کارایی

۵ـ خطاهای آماده‌سازی و اختتامیه

برخلاف آزمایش جعبهٔ سفید که در اوایل فرایند آزمایش انجام می‌شود، آزمایش جعبهٔ سیاه در مراحل آخر آزمایش به کار گرفته می‌شود؛ چون آزمایش جعبهٔ سیاه عمداً به ساختار کنترلی توجهی ندارد و بر دامنهٔ 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.

۱۳ـ۵ـ۱ تحلیل مقادیر مرزی

به دلایلی که کاملاً واضح نیست، تعداد خطاهای نرم‌افزار در مرزهای دامنهٔ ورودی نسبت به مقادیر مرکزی بیشتر است. به همین دلیل روش تحلیل مقادیر مرزی (BVA)[1] 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 ردهٔ مساوی، انتخاب نمونه‌های آزمایش را به لبه‌های این رده هدایت می‌کند. به‌جای تمرکز بر شرایط ورودی، BVA، نمونه‌های آزمایش را از دامنهٔ 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.

۱۳ـ۵ـ۲ آزمایش محیط‌ها، معماری‌ها

نرم‌افزارهای رایانه‌ای پیچیده‌تر شده و نیاز برای شیوه‌های آزمایش خاص نیز رشد کرده است. روش‌های آزمایش جعبهٔ سیاه و جعبهٔ 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) [2] زمینهٔ جالبی را برای مهندسین نرم‌افزار ارائه می‌کنند. به علت اجزای قابل استفادهٔ مجددی که به عنوان بخشی از محیط‌های توسعهٔ GUI فراهم شده‌اند، ایجاد واسط کاربر زمان کمتری نیاز دارد و دقیق‌تر است. اما درعین‌حال، پیچیدگی GUIها نیز افزایش یافته است و باعث مشکلات بیشتر در طراحی و اجرای نمونه‌های آزمایش می‌شود.

چون بسیاری از GUIهای مدرن، احساس و جلوهٔ یکسانی دارند، یک‌سری از آزمایش‌های استاندارد قابل انجام است. گراف‌های مدل‌سازی حالت محدود[3] 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)[4] موارد مهمی را برای آزمایش‌کننده‌های نرم‌افزار نشان می‌دهند. ماهیت توزیع‌شدهٔ محیط‌های C/S، موارد کارایی مربوط به پردازش تراکنش، حضور بالقوهٔ سکوهای 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 هزینهٔ آزمایش محیط‌های C/S است.

---

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

۱۳ـ۵ـ۳ آزمایش مستندات و امکانات راهنما

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

خطاها در مستندات می‌توانند به همان اندازه خطاها در برنامه‌های مبدأ یا داده‌ها، باعث اشکالاتی در پذیرش برنامه شوند. هیچ‌چیز نگران‌کننده‌تر 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 پیش‌بینی شده نباشد. به همین دلیل آزمایش مستندات باید بخش معنی‌داری برای هر طرح آزمایش نرم‌افزار 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 می‌کند. مرحله دوم که آزمایش زنده مستندات نام دارد، مستندات با استفاده از برنامهٔ واقعی، مورد استفاده قرار می‌دهد. استفاده از برنامه از طریق مستندات پیگیری می‌شود.

۱۳ـ۵ـ۴ آزمایش یکپارچه‌سازی

کنار هم قرار دادن مؤلفه‌ها، به معنی ایجاد ارتباط بین آن‌هاست. داده ممکن است در یک ارتباط ناپدید شود. یک پیمانه ممکن است اثر نامطلوب و ناخواسته‌ای را بر دیگری داشته باشد. زیرتوابع، زمانی که با هم ترکیب می‌شوند، ممکن است تابع بزرگ‌تر مورد نظر را به وجود نیاورند. مقادیر غیر دقیقی که در هر یک به‌تنهایی پذیرفته شده‌اند، ممکن است بزرگ شده و به سطوح غیر قابل قبولی برسند. ساختمان داده‌های سراسری ممکن است مشکل‌ساز شود. این لیست همچنان به طور نگران‌کننده‌ای ادامه دارد.

آزمایش یکپارچه‌سازی[1]، روشی سیستماتیک برای ایجاد ساختار برنامه است. هدف، آزمایش مؤلفه‌ها و ایجاد یکپارچه ساختار برنامه‌ای است که توسط طراح دیکته شده است.

یکپارچه‌سازی بالا به پایین

آزمایش یکپارچه‌سازی بالا به پایین، روشی افزایشی برای ایجاد ساختار برنامه است. پیمانه‌ها با حرکت به سمت پایین در سلسله‌مراتب کنترل، یکپارچه می‌شوند. کار، با پیمانهٔ کنترل اصلی[2] شروع می‌شود. پیمانه‌های پایین‌تر نسبت به پیمانهٔ کنترل اصلی در این ساختار به صورت عمقی یا سطحی یکپارچه می‌شوند.

یکپارچه‌سازی پایین به بالا

آزمایش یکپارچه‌سازی پایین به بالا، ساخت و آزمایش نرم‌افزار را با پیمانه‌های ساده (یعنی مؤلفه‌های پایین‌ترین سطوح ساختار برنامه) شروع می‌کند. چون مؤلفه‌ها از پایین به بالا کنار هم قرار می‌گیرند، پردازش‌های لازم برای مؤلفه‌های سطح بعدی همیشه در دسترس هستند و نیاز به جانگهدار از بین می‌رود.

۱۳ـ۵ـ۵ آزمایش رگرسیون[3]

هر بار که پیمانهٔ جدیدی به عنوان بخشی از آزمایش یکپارچه‌سازی به مجموعه نرم‌افزار افزوده می‌شود، نرم‌افزار تغییر می‌کند. مسیرهای جریان دادهٔ جدیدی ایجاد می‌شوند، I/O جدیدی انجام می‌گیرد و منطق کنترل جدیدی فراخوانی می‌شود. این تغییرات ممکن است باعث بروز مشکلات با توابعی شوند که قبلاً بدون خطا کار می‌کردند. آزمایش رگرسیون، اجرای مجدد زیرمجموعه‌ای از آزمایش‌هایی است که قبلاً انجام شده‌اند تا اطمینان حاصل شود که تغییرات، باعث انتشار اثرات جانبی ناخواسته نمی‌شوند.

۱۳ـ۵ـ۶ آزمایش دود

آزمایش دود[1] یک روش یکپارچه‌سازی است و به شکل متداول زمانی استفاده می‌شود که محصولات نرم‌افزاری کم‌اهمیت توسعه داده می‌شوند. درنتیجه، روش آزمایش دود شامل فعالیت‌های 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.

۱۳ـ۵ـ۷ صحت و اعتبارسنجی

آزمایش نرم‌افزار یک فعالیت از عنوان گسترده‌تری است که اغلب با **صحت** و **اعتبارسنجی (V&V)**[2] 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 محصولِ درستی را ایجاد می‌کنیم؟

۱۳ـ۵ـ۸ 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 مشخصهٔ نیازهای 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 توسعهٔ مناسب تمام عناصر پیکربندی نرم‌افزار، مستندسازی و ثبت آن‌ها و ارائه جزئیات لازم برای مرحلهٔ پشتیبانی در دورهٔ زندگی نرم‌افزار است. مرور پیکربندی گاهی تطبیق نیز نامیده می‌شود.

۱۳ـ۵ـ۹ 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**[3] به‌جای مهندسین نرم‌افزار، توسط کاربر نهایی انجام می‌شود و می‌تواند شامل هدایت غیر رسمی یا یک‌سری آزمایش‌های برنامه‌ریزی شده و منظم باشد. درواقع، آزمایش پذیرش می‌تواند در بازهٔ 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**[4] 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 کنترل‌شده** 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 زندهٔ 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.

۱۳ـ۵ـ۱۱ 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.

۱۳ـ۵ـ۱۱ آزمایش احیا

بسیاری از سیستم‌های رایانه‌ای باید بعد از بروز خطا قابل بازیابی باشند و پردازشی را در زمان مشخص‌شده ادامه دهند. سیستم باید در برابر بروز مشکل مقاوم باشد؛ یعنی خطاهای پردازشی نباید باعث خاتمهٔ کل عملکرد سیستم شود. در موارد دیگر، شکست سیستم باید در بازهٔ زمانی مشخصی اصلاح شود؛ زیرا در غیر این صورت ممکن است ضربه اقتصادی شدیدی به وجود آید. **آزمایش احیا**[1] 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؟

۱۳ـ۵ـ۱۲ 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 محدودهٔ 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[2] سعی در بازبینی مکانیزم‌های امنیتی تعبیه‌شده در سیستم را دارد تا از نفوذ غیر قانونی به سیستم محافظت کند.

۱۳ـ۵ـ۱۳ آزمایش فشار

در اثنای مراحل اولیه آزمایش نرم‌افزار، روش‌های آزمایش جعبهٔ سفید و جعبهٔ سیاه، عملکردهای معمول و کارایی متداول سیستم را ارزیابی می‌کنند. **آزمایش‌های فشار**[3] طوری طراحی می‌شوند تا برنامه‌ها را با موقعیت‌های غیر معمول مواجه کنند. درنتیجه 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.

۱۳ـ۵ـ۱۴ 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**[4] 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

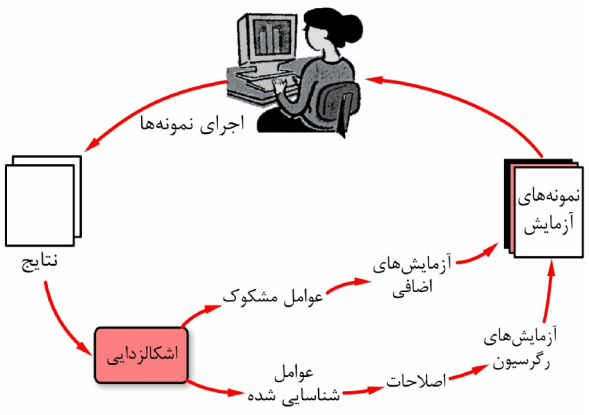
حتی در سطح واحد، کارایی هر پیمانه ممکن است با آزمایش‌های جعبهٔ سفید نیز به دست آید. به‌هرحال تا زمانی که تمام عناصر سیستم به طور کامل یکپارچه نشده باشند، اندازه‌گیری کارایی کامل سیستم قابل دستیابی نیست.

۱۳ـ۶ هنر اشکال‌زدایی

اشکال‌زدایی[1] 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.

۱۳ـ۶ـ۱ 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 ۱۳ـ۵ فرایند اشکال‌زدایی

۱۳ـ۶ـ۲ 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[2] 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[1] 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[2]: 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 ظاهرشده 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)[1] 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 **پیکرهٔ دانش مدیریت پروژه** (PMBoK)[2] برای مدیریت پروژه ارائه داده است. این کتاب استانداردی 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.

۱۴ـ۱ حوزه‌های پیکره دانش مدیریت پروژه

۱۴ـ۱ـ۱ مدیریت حوزه عملکرد[3]

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:

۱۴ـ۱ـ۲ مدیریت یکپارچگی[4]

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

۱۴ـ۱ـ۳ مدیریت زمان[5]

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

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

۱۴ـ۱ـ۴ مدیریت هزینه[1]

توصیف‌کنندهٔ فرایندهای مورد نیاز برای حصول اطمینان از اینکه پروژه در چارچوب بودجه مصوب به اتمام می‌رسد. شامل:

۱۴ـ۱ـ۵ مدیریت منابع انسانی[2]

شامل فرایندهایی است که بهترین شکل به‌کارگیری افراد در پروژه مانند نحوهٔ مدیریت منابع انسانی، سازماندهی، آموزش، ارتباط، تشویق، جذب و ... را تضمین می‌کند و شامل فعالیت‌های زیر است:

۱۴ـ۱ـ۶ مدیریت ریسک[3]

فرایندهای مورد نیاز برای تعیین، تجزیه و تحلیل ریسک و واکنش مناسب در مقابل ریسک پروژه است. مدیریت ریسک با انجام فعالیت‌های زیر صورت می‌گیرد:

۱۴ـ۱ـ۷ مدیریت کیفیت[4]

شامل فرایندهای مورد نیاز برای حصول اطمینان از برآورده شدن ضرورت‌های کیفی تعریف‌شده توسط پروژه است و شامل موارد زیر است:

۱۴ـ۱ـ۸ مدیریت ارتباطات[5]

فرایندهای مورد نیاز برای حصول اطمینان از تولید، جمع‌آوری، ذخیره، انتشار و توزیع مناسب و به‌موقع اطلاعات پروژه است. شامل:

۱۴ـ۱ـ۹ مدیریت تدارکات[6]

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

۱۴ـ۲ طیف مدیریتی

در مدیریت مؤثر پروژه‌های نرم‌افزاری بر چهار محور اساسی تأکید شده است: **افراد، محصول، فرایند و پروژه**.

۱۴ـ۲ـ۱ افراد[7]

عامل انسانی چنان اهمیتی دارد که مؤسسهٔ مهندسی نرم‌افزار، مدل بلوغ قابلیت‌های مدیریت افراد (PM-CMM)[8] را توسعه داده است تا آمادگی سازمان‌های نرم‌افزاری برای به عهده گرفتن کاربردهای پیچیده را بهبود بخشد. مدل بلوغ قابلیت‌های مدیریت افراد، زمینه‌های کلیدی عملی زیر را برای نرم‌افزارنویسان تعیین می‌کند: یافتن افراد جدید، انتخاب، مدیریت اجرا، آموزش، جبران خدمت، توسعه حرفه‌ای، سازماندهی و طراحی کار و توسعه تیمی/ فرهنگی.

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

- M= انگیزش[9]
- O= سازماندهی[10]
- I= نوآوری و خلاقیت[11]

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 Muturity Model
9 - Motivation
10 - Organization
11 - Ideas or Innovation

به‌هرحال بحث در اصول مدیریت و نحوهٔ سازماندهی و مدیریت تیم‌های نرم‌افزاری گسترهٔ وسیعی دارد که در این کتاب به آن‌ها پرداخته نشده است.

- **سازماندهی تیم‌های کاری**

ساختار بهترین تیم کاری به شیوهٔ 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:

- کنترل‌نشدهٔ غیر متمرکز (DD)[1]
- کنترل‌شدهٔ غیر متمرکز (CD)[2]
- کنترل‌شدهٔ متمرکز (CC)[3]

- **عوامل تأثیرگذار بر ساختار و سازماندهی تیم**

	نوع / تیم / ویژگی	ساختار DD	ساختار DC	ساختار CC
۱	دشواری			
	زیاد	✓		
	کم		✓	✓
۲	اندازهٔ برنامه			
	بزرگ		✓	✓
	کوچک	✓		
۳	عمر ماندگاری تیم			
	کوتاه		✓	✓
	درازمدت	✓		
۴	میزان تقسیم‌پذیری مسئله			
	زیاد		✓	✓
	کم	✓		
۵	کیفیت و قابلیت اطمینان			
	زیاد	✓	✓	
	کم			✓
۶	قطعیت تاریخ تحویل			
	غیر قابل انعطاف			✓
	انعطاف‌پذیر	✓	✓	
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 Decenteralized
2 - Controled Decenteralized
3 - Controled Centeralized

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

۱۴ـ۲ـ۲ محصول[1]

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

- **دامنهٔ کاربرد محصول**

نخستین فعالیت مدیریت پروژه نرم‌فزار، تعیین دامنهٔ کاربرد محصول است. دامنهٔ کاربرد با مشخص کردن مؤلفه‌های زیر تعیین می‌شود:

- **محیط:** چگونه نرم‌افزار ایجادشده با سیستم‌های موجود یا محیط حرفه منطبق می‌شود و محدودیت‌ها در این محیط چیست؟
- **اهداف اطلاعاتی:** موجودیت‌ها و اقلام داده‌ای برای خروجی مطلوب و مورد نظر کدام هستند.
- **عملکرد و کارایی:** نحوهٔ تبدیل داده‌های ورودی به اطلاعات خروجی و کارایی مورد نظر چیست؟

- **تجزیه مسئله**

تجزیه مسئله در دو زمینهٔ اصلی صورت می‌پذیرد:

- 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 راه‌های اصولی ارتباط با مشتری استفاده کرد.

۱۴ـ۲ـ۳ فرایند[2]

مدیر پروژه باید در مورد انواع مدل‌های فرایندی توسعهٔ نرم‌افزار (فصل ۲) تصمیم‌گیری کند به طوری که مدل فرایندی برای حالت‌های زیر مناسب باشد:

- مشتری و افرادی که کار را انجام می‌دهند.
- ویژگی‌های محصول
- محیط پروژه که در آن تیم نرم‌افزاری کار می‌کند.

پس از انتخاب مدل فرایندی توسعهٔ نرم‌افزار، مدیر پروژه باید برنامهٔ توسعهٔ نرم‌افزار را تهیه و ارائه کند. به همین منظور وی باید اقدام به **"تلفیق فرایند و محصول"** و **"تجزیه فرایند"** اجرای پروژه کند.

۱۴ـ۲ـ۴ پروژه[3]

برای مدیریت یک پروژه نرم‌افزاری موفق، باید بدانیم چه چیزهایی ممکن است اشتباه شود و چگونه می‌توان آن‌ها را به راحتی رفع کرد. John Reel علایم در خطر بودن یک پروژه را به شرح زیر اعلام کرده است:

- عدم درک نیازهای مشتری

1 - Product
2 - Process
3 - Project

- عدم تعریف خوب از حوزه محصول
- عدم مدیریت مطلوبِ تغییرات
- تغییر فن‌آوری انتخاب‌شده
- 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 پایهٔ نظام مهندسی است و مهندسی نرم‌افزار نیز از این قاعده مستثنی نیست.
معیارهای سنجش نرم‌افزار به گسترهٔ 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 حوزهٔ مدیریت پروژه‌های نرم‌افزاری، ابتدا با معیارهای بهره‌وری و سنجش کیفیت سر و کار داریم که شامل اندازه‌های خروجی فرایند توسعهٔ 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. علاقهٔ ما به برنامه‌ریزی و انجام برآورد، سابقهٔ تاریخی دارد. به عنوان مثال:

- بهره‌وری پروژه‌های توسعهٔ 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؟

۱۵ـ۱ اندازه، معیار سنجش و شاخص

فرهنگ واژه‌های مؤسسه مهندسی نرم‌افزار 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 پیمانهٔ[1] مجزا.
معیار سنجش نرم‌افزار 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 نرم‌افزار، پروژهٔ نرم‌افزاری و یا خود محصول را فراهم می‌کند.
ابزارهای سنجش، بینشی در اختیار مدیر قرار می‌دهد که منجر به تصمیم‌گیری‌های آگاهانه می‌شود. بنابراین:

- **اندازه**[2] : نمایش مقداری از میزان، ابعاد، ظرفیت و ویژگی‌های نرم‌افزار است.
- **معیار**[3] : اندازهٔ مقداری درجه یک سیستم، مؤلفه یا فرایند است.
- **شاخص**[4] : مجموعه‌ای از معیارهای سنجش برای تصحیح عملکرد پروژه است.

۱۵ـ۲ معیارهای سنجش فرایند و پروژه

معیارهای سنجش فرایند در طول تمام پروژه‌ها و در طی یک دورهٔ زمانی طولانی گردآوری می‌شوند. هدف از تهیه معیارهای سنجش فرایند فراهم آوردن شاخص‌هایی برای بهبود فرایند نرم‌افزار در بلندمدت است. شاخص‌های پروژه، یک مدیر پروژه نرم‌افزاری را قادر می‌کند که:
۱ـ وضعیت یک پروژه در حال جریان را ارزیابی کند.
۲ـ ریسک‌های بالقوه را پیگیری کند.
۳ـ زمینه‌های مشکل را پیش از آنکه «بحرانی» شوند، معلوم کند.
۴ـ جریان کار یا فعالیت‌ها را تعدیل کند.
۵ـ قابلیت گروه پروژه را در مورد کنترل کیفیت محصولات کاری مهندسی نرم‌افزار مورد ارزیابی قرار دهد.
بسیاری از معیارهای سنجش هم در حوزهٔ پروژه و هم در فرایند نرم‌افزار 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 پیمانه)
- خطاهای پیداشده در طول توسعه نرم‌افزار

هدف: یافتن شاخص‌های بهبود و افزایش کارایی فرد است

- **معیارهای عمومی**

- تعداد نقص‌های گزارش‌شده در مورد توابع و عملکردهای عمده نرم‌افزار
- تعداد خطاهای مشاهده‌شده در حین بازبینی‌های فنی رسمی
- تعداد خطوط برنامه (LOC)[5] یا ارزش تابعی (FP)[6]

هدف: یافتن شاخص‌های بهبود و افزایش کارایی فرایند سازمانی است.

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

۱۵ـ۲ـ۲ معیارهای سنجش پروژه

معیارهای سنجش فرایند نرم‌افزاری برای مقاصد راهبردی به کار می‌روند. اندازه‌های مقداری پروژه نرم‌افزاری بُعد فنی دارند. یعنی معیارهای سنجش و شاخص‌های پروژه از آن‌ها حاصل شده و از سوی مدیر پروژه و گروه نرم‌افزاری 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 نرم‌افزار
- **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.

۱۵ـ۳ اندازه‌گیری نرم‌افزاری

در دنیای واقعی اندازه‌گیری را می‌توان به دو دستهٔ 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 نرم‌افزار دربرگیرندهٔ سنجش هزینه و تلاش است. اندازه‌گیری مستقیم نرم‌افزار شامل تعداد خطوط برنامه (LOC) تولیدشده، سرعت اجرا، میزان حافظه و خرابی‌های گزارش‌شده در مدت یک دورهٔ زمانی است. سنجش‌های غیر مستقیم نرم‌افزار شامل قابلیت تابعی بودن[1]، کیفیت، پیچیدگی، کارایی، قابلیت اعتماد، قابلیت نگهداری و «قابلیت»های بسیار دیگر است.

۱۵ـ۳ـ۱ معیارهای سنجش مبتنی بر اندازه

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

- تعداد افراد مشارکت‌کننده در توسعه نرم‌افزار (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 پروژهٔ دیگر باشد، شاخص تعداد خطوط برنامه (LOC) را به عنوان ارزش هنجارسازی در نظر گرفته‌ایم.

- **معیارهای نرمال‌شده**

- 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

1 - Functionality

- نسبت LOC به نفر- ماه
- نسبت هزینه به حجم مستندات

البته این روش دارای مجادلات خاص خود است. مخالفین مدعی‌اند که معیارهای LOC، معیارهایی وابسته به زبان برنامه‌نویسی هستند، چون زبان‌هایی وجود دارد که به خوبی طراحی شده‌اند و همین امر موجب کوتاه شدن برنامه‌نویسی 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).

۱۵ـ۳ـ۲ معیارهای سنجش مبتنی بر تابع (عملکرد)

معیارهای سنجش نرم‌افزار مبتنی بر عملکرد از یک معیار اندازه‌گیری تابعی برنامه به عنوان ارزش نرمال‌شده استفاده می‌کنند. چون تابعی بودن یک نرم‌افزار را نمی‌توان مستقیما 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 رابطهٔ تجربی مبتنی بر معیارهای قابل اندازه‌گیری (مستقیم) **دامنه اطلاعات نرم‌افزار** و ارزیابی‌های مربوط به پیچیدگی نرم‌افزار حاصل می‌شود. برای این کار پنج ویژگی از دامنهٔ اطلاعاتی به شرح زیر تعیین و شمارش می‌شوند:

تعداد ورودی‌های کاربران: 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 کاربران: یک درخواست به عنوان یک ورودی لحظه‌ای تعریف می‌شود که حاصل آن تولید پاسخ توسط نرم‌افزار به شکل خروجی‌های لحظه‌ای است.

تعداد پرونده‌ها: هر پروندهٔ اصلی منطقی (نظیر گروه‌بندی منطقی داده‌ها به عنوان بخشی از یک پایگاه دادهٔ 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 رابطهٔ تجربی ارائه‌شدهٔ زیر به دست می‌آید:

ضرایب وزنی پیچیدگی

پارامترهای اندازه گیری	تعداد		ساده	متوسط	پیچیده		
تعداد ورودیهای کاربر		×	۳	۴	۶	=	
تعداد خروجیهای کاربر		×	۴	۵	۷	=	
تعداد درخواستهای کاربر		×	۳	۴	۶	=	
تعداد فایلها		×	۴	۱۰	۱۵	=	
تعداد واسطهای خارجی		×	۵	۷	۱۰	=	
					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:

$$FP = \text{شمارش کل} * \left[۰/۶۵ + ۰/۰۱ \sum F_i \right]$$

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
بی‌تأثیر	ضعیف	معقول	متوسط	با اهمیت	اساسی

- **نرمال‌سازی ارزش تابعی**

وقتی ارزش تابعی محاسبه شد، به طریقی شبیه به روش LOC مقادیر حاصله نرمال‌سازی می‌شوند تا برای سنجش بهره‌وری، کیفیت و سایر مشخصه‌های نرم‌افزار به کار روند.

۱۵ـ۳ـ۳ معیارهای سنجش ارزش تابعی توسعه‌یافته

معیارهای سنجش مبتنی بر ارزش تابعی در اصل برای استفاده در برنامه‌های کاربردی سیستم‌های اطلاعاتی تجاری طراحی شده‌اند. به همین دلیل معیارهای سنجش ارزش تابعی برای بسیاری از سیستم‌ها مهندسی و توکار (که بر عملکرد و کنترل تأکید دارند) نامناسب است. برای بهبود آن، از معیار سنجش **ارزش تابعی توسعه‌یافته** با ارزش تابعی سه بعدی استفاده می‌شود.

ویژگی‌های هر سه بُعد نرم‌افزار محاسبه شده و مقداری می‌شود و سپس تبدیل به مقیاسی می‌شود که بر تابع‌گرایی نرم‌افزار دلالت دارد.

بُعد داده‌ای اکثراً با معیارهای سنجش ارزش تابعی مورد ارزیابی قرار می‌گیرد. شمارش داده‌های شناسایی‌شده (ساختار داده‌های برنامه نظیر پرونده‌ها) و داده‌های خارجی (ورودی‌ها، خروجی‌ها، پرس‌وجوها و ارجعات خارجی) به همراه مقیاس‌های پیچیدگی برای استخراج شاخص بُعد داده‌ای مورد استفاده قرار می‌گیرند.

بُعد تابعی با در نظر گرفتن «تعداد عملیات داخلی مورد نیاز برای تبدیل داده‌های ورودی به خروجی» اندازه‌گیری می‌شود. در محاسبات ارزش تابعی سه‌بعدی، یک تبدیل به عنوان رشته‌ای از گام‌های پردازشی محدودشده توسط دستورات مبنایی در نظر گرفته می‌شود.

بُعد کنترلی با شمارش تعداد تبدیلات بین وضعیت‌های مختلف اندازه‌گیری می‌شود. یک وضعیت نشان‌دهندهٔ برخی از حالات رفتاری قابل رؤیت خارجی است. یک تبدیل به عنوان نتیجهٔ وقایعی است که منجر به تغییر حالت رفتاری نرم‌افزار می‌شود. در حین محاسبه ارزش تابعی سه‌بعدی (3D)، درجهٔ پیچیدگی به انتقال تخصیص داده نمی‌شود.

- **محاسبه ارزش تابعی سه‌بعدی**

برای محاسبهٔ ارزش تابعی سه‌بعدی از رابطهٔ زیر استفاده می‌شود:

$$Index=I+O+Q+F+E+T+R$$

که در آن I ، O ، Q ، F ، E ، T و R به ترتیب نشان‌دهندهٔ مقادیر وزنی پیچیدگی یعنی ورودی‌ها، خروجی‌ها، درخواست‌ها، ساختار داده‌ای داخلی، پرونده‌های خارجی، تبدیلات و انتقالات است. ارزش وزنی پیچیدگی هر کدام نیز با استفاده از رابطهٔ زیر محاسبه می‌شود:

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

که در آن N_{ij} نشان‌دهندهٔ تعداد فراوانی رخداد مؤلفه i (نظیر خروجی‌ها) در هر سطح از پیچیدگی (کم ، متوسط و بالا) و W_{ij} وزن‌های متناظر مربوط به هر یک از آن‌هاست.

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

۱۵ـ۴ تلفیق راهکارهای متفاوت سنجش

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

جدول ۱۵ـ۱ متوسط ارزش تابعی در محیط‌های مختلف

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

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.

۱۵ـ۴ـ۱ اندازه‌گیری کیفیت

اگرچه معیارهای بسیاری در مورد کیفیت، مانند **صحت (درستی عملکرد)** ، **قابلیت نگهداری** ، **یکپارچگی و قابلیت استفادهٔ 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**[1] درجه‌ای است که نشان‌دهندهٔ 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 درصورتی‌که **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)**[2] است که مدت‌زمان لازم برای تحلیل درخواست تغییر، آزمایش و توزیع نسخه تغییریافته 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 نرم‌افزار در عصر سرقت‌ها و ویروس‌های رایانه‌ای به طور فزاینده‌ای اهمیت یافته است. این مشخصه، **قابلیت** یک سیستم را **در مقابله با دستکاری‌های تصادفی یا عمدی** در راستای ایمنی آن اندازه‌گیری می‌کند. این دستکاری‌ها می‌توانند در مورد سه جزء نرم‌افزار (برنامه، داده و مستندات) انجام گیرند.

برای سنجش یکپارچگی نرم‌افزار دو مشخصه **تهدید** و **ایمنی** به صورت زیر تعریف می‌شود:

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

در این صورت **یکپارچگی سیستم** به صورت زیر تعریف می‌شود:

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

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

قابلیت استفاده: بحث سازگاری کاربر با محصولات نرم‌افزاری و به عبارت دیگر کاربرپسندی[1] نرم‌افزار جایگاه خاص خود را پیدا کرده است. اگر برنامه‌ای کاربرپسند نباشد علی‌رغم دارا بودن توابع و عملکردهای با ارزش، اغلب با شکست مواجه می‌شود. **قابلیت استفاده** تلاشی برای **مقداری کردن** کاربرپسندی نرم‌افزار است که از طریق چهار ویژگی زیر اندازه‌گیری می‌شود:

۱ـ مهارت فیزیکی یا ذهنی مورد نیاز برای یادگیری نرم‌افزار

۲ـ مدت‌زمان مورد نیاز برای کسب کارایی بالا در استفاده از سیستم

۳ـ خالص افزایش بهره‌وری در مقایسه با سیستم قبلی

۴ـ ارزیابی شهودی کاربر از نرم‌افزار (گاهی این معیار از طریق پرسشنامه و نظرسنجی از کاربران نسبت به سیستم به دست می‌آید).

چهار معیاری که در بالا مطرح شدند تنها نمونه‌هایی از عواملی هستند که به عنوان معیارهای کیفیت نرم‌افزار پیشنهاد شده‌اند.

۱۵ـ۵ 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 می‌شود:

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

که در آن:

E : تعداد خطاهای پیداشده قبل از تحویل نرم‌افزار به کاربر نهایی

D : تعداد شکست‌هایی که پس از تحویل نرم‌افزار به مشتری مشخص می‌شوند.

مقدار ایده‌آل DRE برابر با ۱ است که نشان‌دهنده عدم وجود خرابی در نرم‌افزار است.

اگر شاخص DRE به عنوان معیاری برای قابلیت فیلترکردن فعالیت‌های تضمین و کنترل کیفیت به کار رود، می‌تواند یک گروه پروژه نرم‌افزاری را به برقراری روش‌هایی برای یافتن خطاهای هرچه بیشتر قبل از تحویل نرم‌افزار به مشتری تشویق کند.

همچنین DRE را می‌توان در یک پروژه برای ارزیابی قابلیت خطایابی فعالیت‌های پشتیبانی یا سایر فعالیت‌های مهندسی نرم‌افزار 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 نرم‌افزار پیدا شده‌اند.

E_{i+1}: تعداد خطاهایی است که در طی فعالیت $i+1$ ام مهندسی نرم‌افزار پیدا شده‌اند و در فعالیت i مشخص نشده‌اند.

هدف کیفی برای یک گروه نرم‌افزاری (یا یک مهندس نرم‌افزار) دستیابی به DRE نزدیک به ۱ است.

1 - User Friendly

۱۵ـ۶ 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. ازآنجاکه یکی از اهداف پروژه‌های مبنی بر شیءگرایی استفادهٔ 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.

۱۵ـ۶ـ۱ مجموعه پیشنهادی از معیارهای پروژه

تعداد متون سناریو: یک متن سناریو، مراحلی 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 **شرکت‌کننده**، 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. ازآنجاکه کلاس‌های کلیدی، در دامنه مسئله اهمیت اساسی دارند، تعداد این‌گونه کلاس‌ها شاخصی از مقدار کار لازم برای توسعهٔ نرم‌افزار و نیز شاخصی از مقدار بالقوه استفادهٔ مجدد در اثنای توسعهٔ سیستم است.

تعدادکلاس‌های پشتیبان: کلاس‌های پشتیبان برای پیاده‌سازی سیستم لازم‌اند، ولی ارتباط مستقیم با دامنهٔ 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 برنامهٔ 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

برنامه‌ریزی 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.

۱۶ـ۱ برآوردها

برآورد منابع، هزینه و زمان‌بندی برای کارهای مهندسی نرم‌افزار نیاز به داشتن تجربه، دستیابی به اطلاعات تاریخی خوب و جرأت انجام پیش‌بینی‌های کمّی 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 توسعهٔ 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

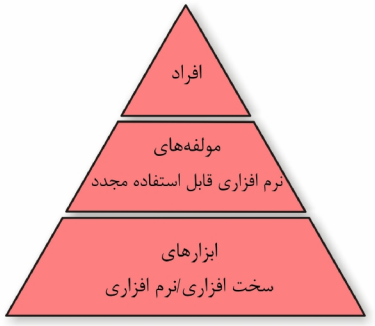
۱۶ـ۱ـ۱ عوامل تأثیرگذار بر برآوردها

- **پیچیدگی پروژه**
- **اندازهٔ پروژه**
- **درجه یا میزان عدم قطعیت ساختاری**

۱۶ـ۲ منابع

دومین کار برنامه‌ریزی پروژهٔ 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 فراهم‌کنندهٔ زیربنایی برای حمایت از تلاش‌های توسعه نرم‌افزار است. در سطح بالاتر، اجزای نرم‌افزاری قابل استفادهٔ 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 ۱۶ـ۰ منابع پروژه

۱۶ـ۲ـ۱ 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.

۱۶ـ۲ـ۲ منابع نرم‌افزاری قابل استفادهٔ مجدد

مهندسی نرم‌افزار مبتنی بر مؤلفه‌ها (CBSE)[1] 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

۱۶ـ۲ـ۳ منابع محیطی

محیطی که پروژه نرم‌افزار را حمایت می‌کند، اغلب محیط مهندسی نرم‌افزار (SEE)[2] 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.

۱۶ـ۳ برآورد پروژه نرم‌افزاری

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

۱ـ تأخیر در برآورد حتی تا اواخر پروژه

۲ـ استفاده از بهترین برآوردهای انجام‌شده روی پروژه‌هایی که قبلاً کامل شده‌اند (مشابه‌سازی)

1 - Component Base Software Engineering
2 - Software Engineering Enviroment

۳ـ استفاده از روش‌های تجزیه نسبتاً ساده برای برآوردهای هزینه‌ای، زمانی و تلاش پروژه
۴ـ استفاده از مدل‌های تجربی برآورد هزینه و تلاش مورد نیاز
مدل‌های تجربی را می‌توان برای تکمیل روش‌های تجزیهٔ 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 پروژهٔ مشابه قابل قبول است. حالت‌های باقی‌مانده، روش‌های عملی برای برآورد پروژهٔ نرم‌افزار هستند.

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

۱۶ـ۴ روش‌های تجزیهٔ برآورد پروژهٔ نرم‌افزار

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

۱۶ـ۴ـ۱ تعیین اندازه نرم‌افزار

دقت برآورد پروژهٔ نرم‌افزار به چند عامل بستگی دارد: (۱) درجه‌ای که برنامه‌ریز اندازهٔ محصول را تخمین زده است، (۲) توانایی ترجمهٔ 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 عمدهٔ برنامه‌ریز پروژه است. در برنامه‌ریزی پروژه، منظور از اندازه به نتیجهٔ کمّیت‌پذیر 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.

۱۶ـ۴ـ۲ 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 اندازهٔ هر عنصر نرم‌افزاری و (۲) به عنوان معیارهای بنیادی جمع‌آوری‌شده از پروژه‌های قبلی و استفاده به همراه متغیرهای تخمینی از هزینه و کار.

برآورد LOC وFP روش‌های برآورد متمایزی هستند. با این وجود هر دو ویژگی‌های مشترکی دارند. برنامه‌ریز پروژه اقدام به تجزیهٔ 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 محدودهٔ اطلاعات، کار را شروع می‌کند. با استفاده از داده‌های سابقه‌ای یا (در صورت شکست موارد دیگر) با استفاده از ادراک، برنامه‌ریز مقدار اندازهٔ **خوشبینانه**، **محتمل** و **بدبینانه** را برای هر تابع یا هر مقدار موجود در محدودهٔ 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})[1] و محتمل (S_m)[2] و بدبینانه (S_{pess})[3] برآوردها به صورت زیر محاسبه می‌شود:

$$S = (S_{opt} + ۴S_m + S_{pess})/۶$$

۱۶ـ۴ـ۳ 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.

1 - Optimistics
2 - Probabilistics
3 - Pessimistics

۱۶ـ۵ مدل‌های تجربی برآورد هزینه و تلاش نرم‌افزار

یک مدل برآورد تلاش نرم‌افزار، از مشاهدات به‌دست‌آمده از فرمول‌های تجربی برای پیش‌بینی کار و تلاش به صورت تابعی از LOC یا FP تعریف شده است.

۱۶ـ۵ـ۱ ساختار مدل‌های برآورد

$$E = A + B \bullet (ev)^{C}$$

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

- **مدل‌های پیشنهادی بر مبنای LOC**

$E = ۵/۲*(KLOC)^{۰/۹۱}$, Walston- Felix

$E = ۳/۲*(KLOC)^{۱/۰۵}$, Boehm Simple

$E = ۵/۵ + ۰/۷۳*(KLOC)^{۱/۱۶}$, Baily- Basili

- **مدل‌های پیشنهادی بر مبنای FP**

$E = -۱۳/۳۹ + ۰/۵۴۵*FP$, Albrecht-Gaffney

$E = ۵۸۵/۷ + ۱۵/۱۲*FP$, Matson-Barnett-Mellichamp

۱۶ـ۵ـ۲ مدل‌های COCOMO[1]

Boehm سه رده از مدل‌های COCOMO را تعریف می‌کند:

- مدل‌های مقدماتی
- مدل‌های میانی
- مدل‌های پیشترفته

هر یک از مدل‌ها در حالت‌های اولیه، نیمه‌جدا و توکار برای پارامترهای A ، B و C برآورد شده است.

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

$PM = ۲/۴(KIOC)^{۱/۰۵}$ (نفر ـ ماه)

$TD_{ev} = ۲/۵(PM)^{۰/۳۸}$ (مدت‌زمان برحسب ماه)

۱۶ـ۵ـ۳ مدل‌های COCOMOII

این مدل شامل سلسله‌مراتبی از مدل‌هاست که زمینه‌های زیر را مورد توجه قرار می‌دهد:

مدل ترکیب کاربرد در مراحل اولیهٔ مهندسی نرم‌افزار استفاده می‌شود، هنگامی که نمونه‌سازی واسط‌های کاربر، توجه به ارتباط نرم‌افزار و سیستم، تشخیص کارایی و ارزیابی تکامل تکنولوژی، اهمیت دارند.

مدل مرحله اولیه طراحی زمانی استفاده می‌شود که نیازها تثبیت شده و ساختار پایهٔ معماری ایجاد شده است.

مدل مرحله بعد از معماری در ایجاد نرم‌افزار استفاده می‌شود.

۱۶ـ۵ـ۴ معادله نرم‌افزار

معادلهٔ نرم‌افزار مدلی پویا و چندمتغیره است که فرض می‌کند توزیع خاصی از فعالیت‌ها در دورهٔ زندگی یک پروژهٔ توسعهٔ نرم‌افزار وجود دارد. این مدل از داده‌های قابلیت تولیدی که برای ۴۰۰۰ پروژهٔ نرم‌افزاری جمع‌آوری شده‌اند به دست آمده است. بر مبنای این داده‌ها، مدل برآورد به صورت زیر ارائه شده است:

1 - Constructive Cost MOdel

یا

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

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

که در آن

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

t: مدت‌زمان 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 به‌کارگیری تجارب مهندسی نرم‌افزار خوب
- سطح زبان برنامه‌نویسی استفاده‌شده
- حالت محیط نرم‌افزار
- مهارت‌ها و تجربیات تیم نرم‌افزار
- پیچیدگی کاربرد

مقادیر معمول می‌تواند $P = 2000$ برای توسعهٔ نرم‌افزار بلادرنگ توکار، $P = 10000$ برای نرم‌افزار 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 معادلهٔ نرم‌افزار 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:

$$t_{min} = 8.14 * (LOC / p)^{0.43} \quad , \quad \text{if} \quad t \geq 6$$

$$E = 180 B t^3 \quad , \quad \text{if} \quad E \geq 20$$

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

۱۶ـ۶ رهیافت پیشنهادی برآورد و زمان‌بندی برای پروژه‌های شیءگرا

۱ ایجاد برآوردهایی با استفاده از تجزیهٔ کار، تحلیلFP و هر روش دیگری که برای کاربردهای سنتی قابل اجرا باشد.

۲ با استفاده از مدل‌های تحلیل شیءگرا، 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 شیءگرا تعداد کلاس‌های کلیدی را تعیین کنید.

۴ نوع واسط را برای برنامهٔ کاربردی دسته‌بندی کرده و برای کلاس‌های پشتیبان ضریبی ایجاد کنید.

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

۵ تعداد کل کلاس‌ها (کلیدی + پشتیبان) را در تعداد میانگین واحدهای کاری مورد نیاز ضرب کنید. Lorenz و Kidd 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.

۱۶ـ۷ 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 هزینهٔ 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 هزینهٔ خرید و تطبیق با نیازها کمتر از هزینهٔ توسعهٔ نرم‌افزار به صورت داخلی است؟ (۳) آیا هزینهٔ پشتیبانی خارجی (برای مثال، قرارداد پشتیبانی) کمتر از هزینهٔ پشتیبانی داخلی است؟ این شرایط برای هر یک از حالت‌های توسعه باید در نظر گرفته شوند.

۱۶ـ۷ـ۱ ایجاد درخت تصمیم‌گیری

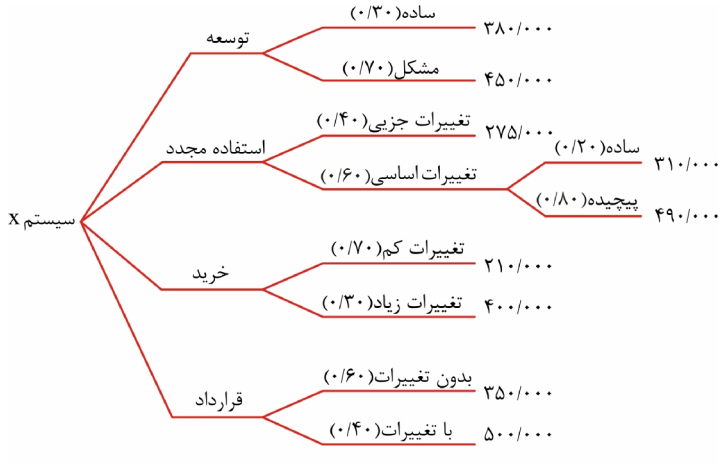
مراحل مطرح‌شده در بالا را می‌توان با استفاده از روش‌های آماری مانند **تحلیل درخت تصمیم** گسترش داد. برای مثال یک درخت تصمیم را برای تولید یک سیستم نرم‌افزاری به صورت زیر در نظر بگیرید.

(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 توسعه‌دهندهٔ 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 توسعهٔ ساده ۳۸۰۰۰۰ واحد پولی هزینه برآورد شده است؛ بنابراین هزینه مورد انتظار هر شاخه از درخت تصمیم عبارت است از:

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

که i مسیری در درخت تصمیم است.



شکل ۱۶ـ۲ درخت تصمیم برای توسعه نرم‌افزار

بنابراین داریم:

(۱) هزینه مورد انتظار برای توسعه $= ۰/۳۰(۳۸۰)+۰/۷۰(۴۵۰)=۴۲۹$

(۲) هزینه مورد انتظار برای استفاده مجدد $= ۰/۴۰(۲۷۵)+۰/۰\left[۰/۲۰(۳۱۰)+۰/۸۰(۴۹۰)\right]=۳۲۰$

(۳) هزینه مورد انتظار برای خرید $= ۰/۷۰(۲۱۰)+۰/۳۰(۴۰۰)=۲۶۷$

(۴) هزینه مورد انتظار برای قرارداد $= ۰/۶۰(۳۵۰)+۰/۴۰(۵۰۰)=۴۱۰$

بنابراین کمترین هزینهٔ مورد انتظار، حالت **"خرید"** است.

۱۶ـ۸ پیمانکاری

استفاده از منابع خارجی از نظر مفهومی، بسیار ساده است. برای انجام فعالیت‌های مهندسی نرم‌افزار با مجموعهٔ دیگری که این کار را با هزینه و زمان کمتر و کیفیت بالاتر انجام می‌دهد قرارداد بسته می‌شود. در این حالت مدیریت توسعهٔ نرم‌افزاری در یک شرکت، به یک فعالیت مدیریت عقد قرارداد کاهش می‌یابد.

تصمیم برای استفاده از منابع خارجی می‌تواند راهبردی یا فنی باشد. در سطح راهبردی، مدیران تجاری در نظر می‌گیرند که آیا بخش مهمی از تمام کار نرم‌افزار می‌تواند به دیگران واگذار شود. در سطح فنی، مدیر پروژه مشخص می‌کند که آیا بخشی یا تمام یک پروژه می‌تواند به بهترین شکل با بستن قرارداد پیمانکاری[1] انجام شود.

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

۱۶ـ۹ 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 می‌دهند:

۱ـ تعیین اندازهٔ عناصر قابل تحویل پروژه

۲ـ انتخاب فعالیت‌های پروژه

۳ـ پیش‌بینی میزان نیاز به منابع انسانی

۴ـ پیش‌بینی تلاش مورد نیاز نرم‌افزار

۵ـ پیش‌بینی هزینه نرم‌افزار

۶ـ پیش‌بینی زمان مورد نیاز برای انجام فعالیت‌های نرم‌افزاری

هنگامی که ابزارهای متفاوت برآورد برای داده‌های یک پروژه به کار گرفته می‌شوند، تفاوت‌های نسبتاً زیادی در نتایج برآورد مشاهده می‌شود. مهم‌تر اینکه، مقادیر پیش‌بینی شده گاه تا حد زیادی متفاوت با مقادیر واقعی هستند؛ بنابراین توصیه شود که خروجی ابزارهای برآورد به عنوان یک **" نقطهٔ 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.**

1 - Outsourcing

فصل هفدهم

تحلیل و مدیریت ریسک

هنگامی که بحث ریسک در مهندسی نرم‌افزار مطرح می‌شود، هر سه فرض مفهومی Pobert Charette همواره قابل مشاهده‌اند. (۱) ما با آینده سر و کار داریم- چه ریسک‌هایی ممکن است باعث ناکامی در پروژهٔ 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:

۱۷ـ۱ راهبردهای ریسک واکنشی در برابر پیش‌واکنشی

۱۷ـ۱ـ۱ 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.

۱۷ـ۱ـ۲ 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 ازآنجاکه همهٔ ریسک‌ها قابل پیش‌بینی و جلوگیری نیستند، تیم کوشش می‌کند تا یک برنامهٔ احتیاطی توسعه دهد و یا به خوبی در برابر آن‌ها عکس‌العمل نشان دهد.

۱۷ـ۲ 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 بیافتد یا نیافتد.
- **ضایعات:** یعنی اگر ریسک به‌وقوع بپیوندد حتماً ضایعاتی دربر خواهد داشت.

۱۷ـ۲ـ۱ 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 حیطهٔ 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.

۱۷ـ۳ شناسایی ریسک

برای هر یک از گروه‌های ارائه‌شده دو نوع متمایز ریسک وجود دارد: **ریسک‌های کلی و ریسک‌های خاص محصول**. ریسک‌های کلی برای همهٔ 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 پروژهٔ مورد نظر دارند. برای شناسایی ریسک‌های خاص پروژه، برنامه‌ریزی پروژه و بیانیهٔ حوزهٔ عملکرد نرم‌افزار مورد بررسی قرار می‌گیرند و پاسخی برای این سؤال به دست می‌آید: *" کدام‌یک 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:

- **اندازهٔ محصول**: ریسک‌های مرتبط با اندازهٔ 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 درجهٔ تعریف و بلوغ سازمان مهندسی نرم‌افزار توسط توسعه‌دهندهٔ 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 توسعهٔ سیستم.
- **تجربه و تعداد پرسنل**: ریسک‌های مرتبط با تجربهٔ 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.

۱۷ـ۳ـ۱ مؤلفه‌های ریسک

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

۱۷ـ۳ـ۲ محرک‌های ریسک

تأثیر مؤلفه‌های ریسک را نشان می‌دهند که به چهار گروه زیر تقسیم می‌شوند:

- قابل چشم‌پوشی
- کم‌اهمیت

- بحرانی
- فاجعه‌بار

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 ۱۷ـ۱ جدول ارزیابی محرک‌ها و مؤلفه‌های ریسک

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

۱۷ـ۴ پیش‌بینی ریسک (برآورد ریسک)

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

۱ـ احتمال وقوع،

۲ـ پیامدهای مرتبط،

۱۷ـ۴ـ۱ 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

۱۷ـ۴ـ۲ تهیه جدول ریسک

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

جدول ۱۷ـ۲ 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، شایستهٔ توجه بیشتر و برنامه‌ریزی هستند. ریسک‌هایی که زیر این خط قرار می‌گیرند، دوباره ارزیابی می‌شوند تا اولویت‌بندی مرتبه دوم صورت پذیرد.

همهٔ 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**[1] 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.

۱۷ـ۵ در معرض ریسک

میزان کلی قرار گرفتن در معرض ریسک[2] 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.

1 - Risk Mitigation, Monitoring and Management
1 - Risk Exposuer

۱۷ـ۶ 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

۱۷ـ۷ سنجش تأثیر ریسک

اگر ریسکی به وقوع بپیوندد، احتمالاً یک‌سری 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

۱۷ـ۸ برنامه تقلیل، نظارت و مدیریت ریسک (برنامه RMMM)

همهٔ فعالیت‌های تحلیل ریسک، یک هدف منفرد دارند، یعنی کمک به تیم پروژه برای توسعهٔ راهبردی در مقابله با ریسک. در یک راهبـرد 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 هزینهٔ پروژه** می‌شود. 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 می‌کند

برای یک پروژهٔ 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 می‌شود! **از همین رو قاعده ۲۰-۸۰ پارتو را در مورد ریسک نرم‌افزاری** اعمال می‌کنیم. به همین دلیل، برخی از ریسک‌های شناسایی‌شده، سنجش و پیش‌بینی‌شده ممکن است به طرح RMMM راه پیدا نکنند. درنتیجه:

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

فصل هجدهم

زمان‌بندی، نظارت و کنترل پروژه

برای انجام هر پروژه نرم‌افزاری نیاز به داشتن برنامهٔ 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 **زمان‌بندی، نظارت و کنترل پروژه** گویند.

۱ـ۱۸ 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 زمان‌بندی توسط مدیر پروژه
- مدیر پروژهٔ بی‌تجربه و نالایق
- عدم تعهد کافی اعضای تیم
- تأثیرات ناشی از رقبا
- و ...

۱۸ـ۲ اصول و مفاهیم پایه‌ای زمان‌بندی

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 زمان‌بندی 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.

زمان‌بندی پروژه‌های نرم‌افزاری فعالیتی است که کارهای برآوردشده را به وظایف مهندسی نرم‌افزار اختصاص می‌دهد. به‌هرحال توجه به این نکته مهم است که این زمان‌بندی با گذشت زمان، تکامل می‌یابد. در **مراحل اولیهٔ برنامه‌ریزی پروژه، یک زمان‌بندی ماکروسکوپی** توسعه داده می‌شود. این نوع زمان‌بندی مشخص‌کنندهٔ تمام فعالیت‌های عمدهٔ مهندسی نرم‌افزار و توابعی از محصول است. با قرار گرفتن پروژه در مسیر خود، هر ورودی زمان‌بندی ماکروسکوپی به صورت زمان‌بندی همراه با جزئیات تعریف می‌شود. در اینجا، کارهای خاص نرم‌افزاری (مورد نیاز برای دستیابی به یک فعالیت) مشخص و زمان‌بندی می‌شوند.

مانند تمام زمینه‌های دیگر مهندسی نرم‌افزار، اصول بنیادی زیر، رهنمودهایی را در رابطه با زمان‌بندی پروژهٔ نرم‌افزاری ارائه می‌دهند:

- **تقسیم:** پروژه باید به چند فعالیت و کار قابل مدیریت تقسیم شود. برای دستیابی به این قطعه‌بندی، محصول و فرایند باید تجزیه شوند.
- **وابستگی**: وابستگی هر یک از فعالیت‌ها یا کارها باید مشخص شود. برخی کارها باید به ترتیب انجام گیرند درحالی‌که بقیه می‌توانند به شکل موازی انجام شوند.
- **تخصیص زمان:** به هر کاری که باید زمان‌بندی 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 رابطهٔ بین کارها و نوع تخصیص کار به صورت تمام‌وقت یا پاره‌وقت است.
- **اعتبارسنجی کارها:** هر پروژه چند سرپرست تعریف‌شده دارد. با تخصیص زمان، مدیر پروژه باید مطمئن شود که در هر مرحله بیشتر از تعداد افراد اختصاص‌یافته برای کار مورد نظر تخصیص داده نشده‌اند.
- **تعیین وظایف:** هر کاری که زمان‌بندی شده است باید به عضو مشخصی از تیم اختصاص یابد.
- **تعیین پیامدها:** هر کاری که زمان‌بندی شده است باید نتیجهٔ تعریف‌شده‌ای نیز داشته باشد. برای پروژه‌های نرم‌افزاری، این نتیجه به طور معمول یک محصول کاری (برای مثال، طراحی یک پیمانه)، 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.

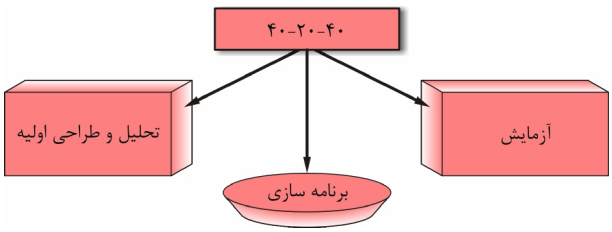
۱۸ـ۳ رابطهٔ بین افراد و کار

یک نکتهٔ کوچک مشترک وجود دارد که هنوز اکثر مدیرانی که مسئول فعالیت توسعه نرم‌افزار هستند به آن باور دارند: **"اگر از زمان‌بندی عقب باشیم، همیشه می‌توانیم برنامه‌نویس‌های 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 زمان‌بندی 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 توسعهٔ نرم‌افزاری موفق ضروری هستند ولی هر مسیر ارتباطی جدید، نیازمند فعالیت و درنتیجه زمان بیشتری است.

۱۸ـ۴ توزیع کار

توزیع توصیه‌شده‌ای برای فعالیت‌های مهندسی نرم‌افزار در مراحل توسعه نرم‌افزار، اغلب قانون ۴۰-۲۰-۴۰ نامیده می‌شود. چهل درصد تمام فعالیت‌ها به تحلیل و طراحی اولیه تخصیص یافته است. یک ۴۰ 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 ۱۸ـ۱ توزیع پیشنهادی فعالیت‌ها

۱۸ـ۵ مجموعه کاری و انواع پروژه

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

۱ـ **پروژه‌های توسعهٔ مفهومی**، برای توصیف و طراحی یک مفهوم جدید تجاری یا کاربردی از یک فن‌آوری جدید تعریف می‌شوند.

۲ـ **پروژه‌های توسعهٔ کاربرد جدید**، بر اساس درخواست مشتری منجر به تولید نرم‌افزار می‌شوند و نتیجهٔ آن تولید یک محصول نرم‌افزاری کاربردی است.

۳ـ **پروژه‌های ارتقای نرم‌افزار کاربردی**، زمانی انجام می‌شوند که نرم‌افزار نیاز به اصلاحات عمده‌ای برای عملکرد، کارایی یا واسط‌های کاربر نهایی دارد.

۴ـ **پروژه‌های پشتیبانی نرم‌افزارهای کاربردی** شامل فعالیت‌های نگهداری نرم‌افزار جاری و در حال کار است.

۵ـ **پروژه‌های مهندسی مجدد**، با هدف باز مهندسی سیستم موجود به صورت کلی یا جزئی اجرا می‌شوند.

۱۸ـ۵ـ۱ درجهٔ دشواری

حتی برای پروژه‌ای خاص غیر ممکن است که درجهٔ دشواری فرایند نرم‌افزار تا حد زیادی تغییر کند. درجهٔ 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 نرم‌افزار هنوز قابل دستیابی هستند.
- **ساخت‌یافته:** فرایند مهندسی نرم‌افزار 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)[1] و مدیریت پیکربندی (SCM)[2] 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 نرم‌افزار 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. توسعهٔ مجموعهٔ کاملی از مستندات، هدایت کردن مرورهای اضافی که Backfilling نامیده می‌شوند، بعد از تحویل محصول به مشتری صورت می‌گیرند.

۱۸ـ۵ـ۲ معیارهای تطبیق

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

به هر یک از معیارهای تطبیق درجه‌ای بین ۱ تا ۵ نسبت داده می‌شود که ۱ نشان‌دهندهٔ پروژه‌ای است که در آن مجموعه‌ای از کارهای فرایند مورد نیاز است و تمام نیازهای فرایندی و مستندسازی حداقل هستند و ۵ نشان‌دهندهٔ 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.

۱۸ـ۵ـ۳ انتخاب وظایف و محاسبهٔ 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 نقطهٔ ورودی مقدار صفر یا ۱ برای هر نوع پروژه دارد و رابطهٔ معیار تطبیق را با نوع پروژه مشخص می‌کند.

۴- نتیجهٔ حاصل‌ضرب زیر در ستون نتیجه درج می‌شود:

ضریب نقطهٔ ورودی × ضریب ارزش × درجه

۵- متوسط تمام مقادیر ستون نتیجه محاسبه می‌شود که انتخاب‌کنندهٔ مجموعه وظایف (TSS)[1] نامیده می‌شود. این مقدار برای کمک به انتخاب مجموعهٔ وظایف مورد نیاز برای یک پروژه مناسب‌ترین معیار است.

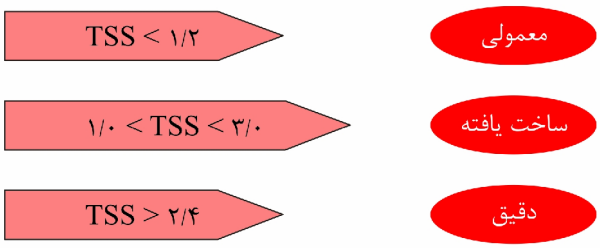
در جدول ۱۸ـ۱ برای توسعه یک نرم‌افزار کاربردی جدید مقادیر درجه و ارزش تعیین شده و با توجه به ضریب نقطهٔ ورودی مربوط به ستون کاربرد جدید، مقادیر مربوط به ستون نتیجه محاسبه شده است.

1 - Task Selector Set

جدول ۱۸-۱ انتخاب مجموعهٔ وظایف پروژه

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

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



شکل ۱۸ـ۲ نحوهٔ 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 مجموعهٔ وظایف غیر ممکن و بستگی به تجربه و مهارت مدیر پروژه و ویژگی‌های محصول دارد.

۱۸ـ۶ مدیریت پروژه‌های نرم‌افزاری شیءگرا

۱۷ـ۶ـ۱ 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.
٤- تعریف یک‌سری نقاط کنترلی برای مدیریت ریسک، تضمین کیفیت و کنترل پروژه
٥- مدیریت تغییرات که به طور ذاتی در پیشرفت پروژه رخ می‌دهند.
٦- پیگیری، نظارت و کنترل پیشرفت

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

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

۱۸ـ۶ـ۲ تعریف چارچوب فرایند مشترک

چارچوب فرایند مشترک (CPF)[1] 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.

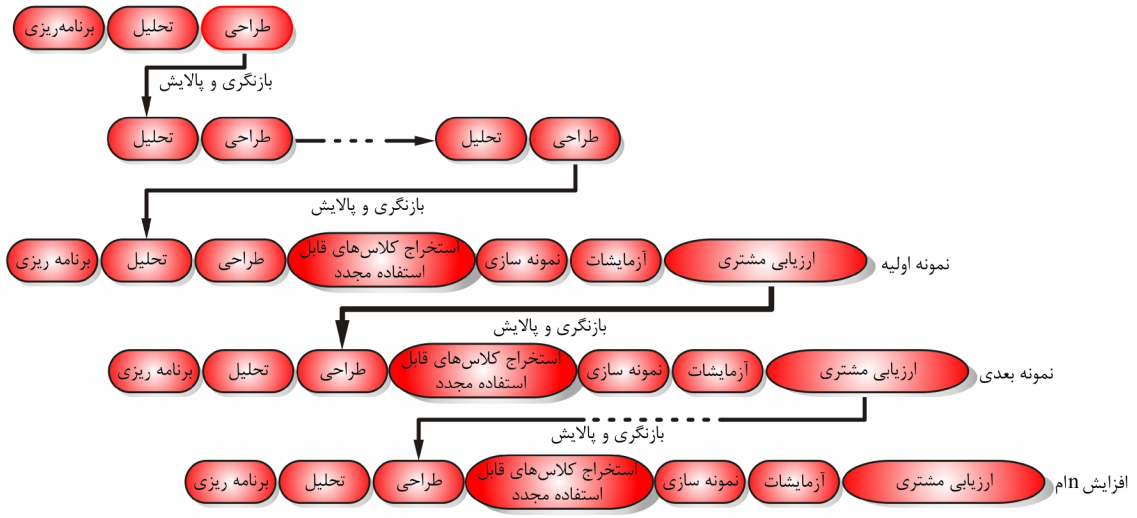
Grady و Booch مدل بازگشتی/ موازی را برای توسعه نرم‌افزارهای شیءگرا پیشنهاد می‌کنند. استفاده از مدل بازگشتی/ موازی در مهندسی نرم‌افزار شیءگرا مستلزم اجرای مراحل زیر است:

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 استفادهٔ مجدد از یک کتابخانه، برای ساختن یک نمونهٔ اولیه تقریبی
۴- اجرای چند آزمون جهت کشف خطاها در نمونهٔ اولیه
۵- دریافت نظرات مشتری در خصوص نمونهٔ اولیه
۶- اصلاح مدل تحلیل بر اساس بازخوردی که از نمونهٔ 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 نمونهٔ اولیه به یک برنامهٔ کاربردی تبدیل شود. مدل بازگشتی/ موازی شبیه به الگوی تکاملی یا حلزونی است زیرا:

1 - Common Process Framework

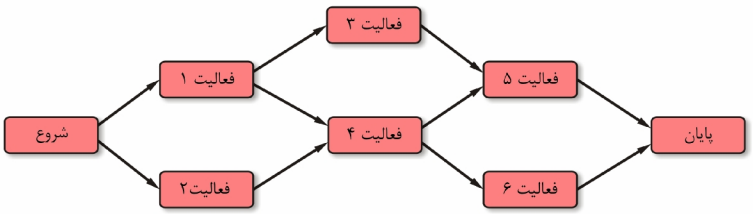
- هر دور تکرار نیاز به برنامه‌ریزی، مهندسی (تحلیل، طراحی، استخراج کلاس‌ها، ایجاد نمونهٔ اولیه و آزمون) و فعالیت‌های ارزیابی دارد.
- در اثنای برنامه‌ریزی، فعالیت‌های مرتبط با هر یک از مؤلفه‌های مستقل، برنامه‌ریزی و زمان‌بندی می‌شود.
- مراحل اولیهٔ مهندسی، تحلیل و طراحی به طور تکراری انجام می‌شود. هدف 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 نتیجهٔ آن بر فعالیت برنامه‌ریزی بعدی تأثیر می‌گذارد، (شکل ۱۸ـ۳ را ببینید).



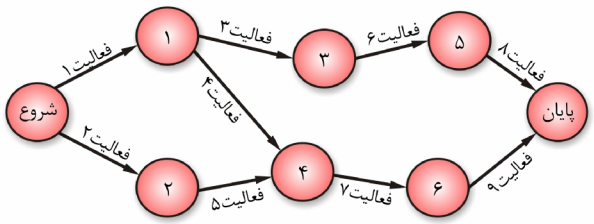
شکل ۱۸ـ۳ فرایندهای متداول برای یک پروژه شیءگرا

۱۸ـ۷ تعریف شبکهٔ وظایف

یک شبکهٔ کاری که **شبکهٔ فعالیت** نیز نامیده می‌شود، نمایش گرافیکی از جریان کاری پروژه است. گاهی از آن به عنوان مکانیزمی استفاده می‌شود که از طریق تعریف دنبالهٔ کاری و وابستگی‌های آن‌ها به عنوان ورودی به یک ابزار خودکار، زمان‌بندی پروژه انجام می‌شود. دو نوع شبکه AON[1] و AOA[2] وجود دارد. شکل‌های زیر یک نمونه از هر نوع شبکه را نشان می‌دهد.



شکل ۱۸ـ۴ شبکه فعالیت‌ها در AON



شکل ۱۸ـ۵ شبکه فعالیت‌ها در AOA

1 - Activity On Node
2 - Activity On Arc

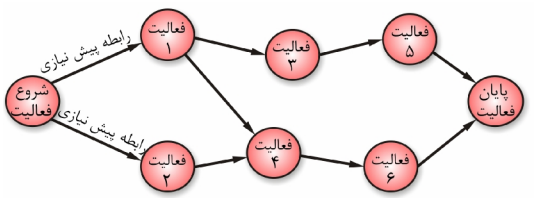
۱۸ـ۸ زمان‌بندی پروژه

فن ارزیابی و بازنگری پروژه/ برنامه (PERT)[1] و **روش مسیر بحرانی** (CPM)[2] دو روش زمان‌بندی پروژه هستند که می‌توانند در توسعهٔ نرم‌افزار به کار گرفته شوند. هر دو روش زمان‌بندی شبکه توسط اطلاعات حاصل از فعالیت‌های برنامه‌ریزی پروژه به دست می‌آیند.
وابستگی‌های بین کارها می‌توانند با استفاده از یک شبکه کاری تعریف شوند. کارها که گاهی ساختار شکست کار (WBS)[3] پروژه نیز نامیده می‌شوند برای کل محصول و برای توابع جزئی تعریف می‌شوند.
PERT و CPM ابزارهایی کمّی را فراهم می‌کنند که به برنامه‌ریز پروژه امکان می‌دهد تا:

۱- مسیر بحرانی را مشخص کند که شامل زنجیره‌ای از فعالیت‌ها برای مشخص کردن مدت پروژه است.
۲- تخمین‌های زمانی با احتمال بالا را برای هر یک از فعالیت‌ها با به‌کارگیری مدل‌های آماری ایجاد کند.
۳- زمان‌های بحرانی، شناوری و مرزی را محاسبه کرده که برای فعالیت‌ها تعریف شده‌اند.

۱۸ـ۸ـ۱ روش مسیر بحرانی

شبکه AON را در نظر بگیرید:



شکل ۱۸ـ۶ مثال شبکه AOA فعالیت‌ها

برای هر گره در شبکه AOA زودترین زمان شروع فعالیت‌های بعد از گرهٔ i را به صورت زیر و به روش پیشرو محاسبه می‌کنیم:

$$\text{Set } ES_{start} = 0 \text{ or Date},$$
$$ES_i = \text{Max}\left\{ES_i + d_{ij}\right\} \quad , \forall i \in \text{شبکه}$$

در انتهای شبکه با قرار دادن مقدار محاسبه‌شده به عنوان آخرین مهلت تکمیل فعالیت‌های پروژه در آخرین گره، آخرین مهلت تکمیل فعالیت‌های قبل از گره j را به روش عقبگرد به شرح زیر محاسبه می‌کنیم:

$$\text{Set } LF_{end} = ES_{end}$$
$$LF_i = \text{Min}\left\{LF_j - d_{ij}\right\}, \quad \forall j \in \text{شبکه}$$

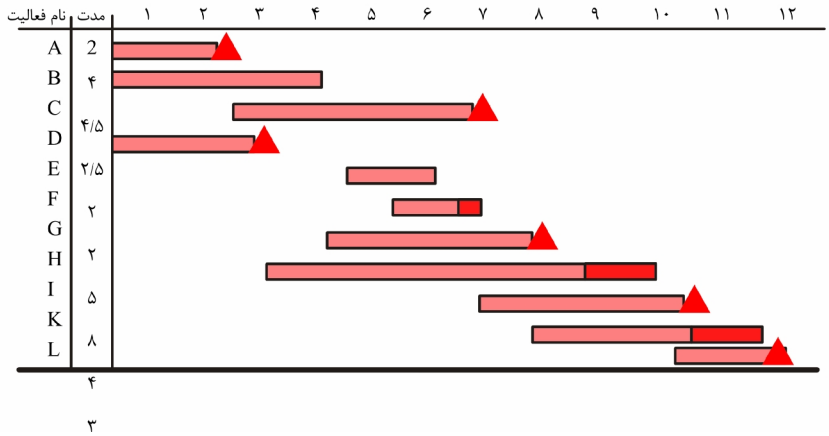
در این صورت، مسیر بحرانی به صورت زیر تعیین می‌شود:

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

۱۸ـ۸ـ۲ نمودار گانت

نمونه‌ای از نمودار گانت که نمودار خط زمانی نیز نامیده می‌شود. در شکل زیر نشان داده شده است.

1 - Project/Program Evaluation and Review Technique
2 - Critical Path Method
3 - Work Breakdown Structure



شکل ۱۸ـ۷ نمونه‌ای از نمودار گانت

۱۸ـ۸ـ۳ پیگیری زمان‌بندی

زمان‌بندی پروژه امکان نظارت بر پیشرفت پروژه را برای مدیر پروژهٔ نرم‌افزاری فراهم می‌کند. اگر زمان‌بندی پروژه‌ای امکان کنترل به شکل مناسب را داشته باشد، کنترل پروژه وجود داشته و موفقیت پروژه تا حد زیادی تضمین می‌شود؛ بنابراین:

- پیگیری زمان‌بندی، نقشهٔ راهنمای مدیر پروژه است.
- پیگیری زمان‌بندی، وظایف و نقاط شاخص‌های زمانی[1] پروژه را مشخص می‌کند.

نظارت، کنترل و پیگیری زمان‌بندی پروژه را در صورت مشخص بودن معیارهای پیگیری و کنترل پیشرفت پروژه می‌توان به چند روش انجام داد:

- برپایی نشست‌های ادواری
- ارزیابی نتایج بازبینی
- کنترل کردن شاخص‌های زمانی
- مقایسه تاریخ‌ها (شروع و پایان)
- استفاده از تحلیل ارزش حاصله

۱۸ـ۹ استقرار معیارهای پیگیری پیشرفت پروژه شیءگرا

با اجرای همزمان فعالیت‌های یک پروژه شیءگرا، شاخص‌های زمانی تعیین‌شده باید توسط مدیر پروژه کنترل شود. این شاخص‌ها و موارد مربوطه در ادامه ارائه شده است.

۱- شاخص‌های تکمیل شدن تحلیل شیءگرا

- همه کلاس‌ها و سلسله‌مراتب کلاس‌ها تعریف و بازبینی شده‌اند.
- صفات و عملیات کلاس‌های مرتبط با یک کلاس تعریف و بازبینی شده‌اند.
- روابط کلاس‌ها تعیین و بازبینی شده‌اند.
- یک مدل رفتاری ایجاد و بازبینی شده است.
- کلاس‌های قابل استفادهٔ مجدد مشخص شده‌اند.

1 - Mile Stones

۲- شاخص‌های تکمیل شدن طراحی شیءگرا

- 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 شیءگرا

- 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 شیءگرا

- درستی و کامل بودن مدل تحلیل و طراحی شیءگرا 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.

۱۸ـ۱۰ تحلیل ارزش حاصله

آیا روش کمّی برای دستیابی به پیشرفت پروژهٔ نرم‌افزاری که دارای زمان‌بندی است، وجود دارد؟ درواقع بله، این روش، تحلیل ارزش حاصله[1] نام دارد.

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

۱ـ **هزینهٔ بودجه‌ای کار زمان‌بندی شده (BCWS)**[2]: 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:

$$BCWS = \sum_i BCWS_i$$

۲ـ مقادیر $BCWS_i$ برای تمام کارهای پروژه با هم جمع شده تا **بودجهٔ تکمیل پروژه** (BAC)[3] را مشخص کند.

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

۳ـ **هزینهٔ بودجه‌ای کار انجام‌شده (BCWP)**[4] 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

$$BCWP = \sum_i BCWP_i$$

Wilkens اشاره می‌کند که تفاوت بین BCWS و BCWP این است که اولی نشان‌دهندهٔ بودجهٔ فعالیت‌هایی است که برای تکمیل پروژه برنامه‌ریزی شده‌اند و دومی نشان‌دهندهٔ بودجهٔ 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**[1]

$$SPI = BCWP/BCWS$$

- **شاخص واریانس زمان‌بندی**[2]

$$SV = BCWP - BCWS$$

SPI نشان‌دهندهٔ کارایی استفادهٔ پروژه از منابع زمان‌بندی شده در پروژه است. مقدار SPI نزدیک به ۱ نشان‌دهنده اجرای کارآمد زمان‌بندی پروژه است. SV فقط نشان‌دهندهٔ 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 کمّی درصد تکمیل پروژه در نقطه‌ای از پروژه در زمان t است.

این امکان نیز وجود دارد که هزینه واقعی کار انجام‌شده (ACWP)[3]، 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**[4]

$$CPI = BCWP/ACWP$$

- **شاخص واریانس هزینهٔ پروژه**

$$CV = BCWP – ACWP$$

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

برخی نرم‌افزارنویسان همچنان بر این باورند که کیفیت نرم‌افزار چیزی است که پس از تولید برنامه باید نگران آن بود. درصورتی‌که اطمینان مرغوبیت نرم‌افزار (SQA)[1] یک فعالیت پشتیبانی است که در سرتاسر فرایند نرم‌افزار اجرا می‌شود. SQA شامل موارد زیر است:

- روشی برای مدیریت کیفیت
- اطمینان از به‌کارگیری فن‌آوری مهندسی نرم‌افزار مؤثر (روش‌ها و ابزارها)
- بازبینی‌های فنی و رسمی در سرتاسر فرایند توسعه نرم‌افزار
- یک راهکار آزمون چند لایه‌ای
- کنترل مستندسازی نرم‌افزار و تغییرات اعمال‌شده در آن
- رویه‌ای برای حصول اطمینان از مطابقت محصول با استانداردهای توسعهٔ نرم‌افزار (در صورت امکان)
- استفاده از راهکارهای اندازه‌گیری و گزارش‌دهی به مدیریت

۱۹ـ۱ـ۱ کیفیت چیست؟

در فرهنگ لغات **کیفیت** به عنوان "*ویژگی یا صفتی از یک عنصر*" تعریف می‌شود. به عنوان صفت یک شیء، کیفیت عبارت است از ویژگی‌های قابل اندازه‌گیری آن شیء. بر این اساس با دو نوع کیفیت مواجه هستیم:

کیفیت طراحی به ویژگی‌هایی اطلاق می‌شود که طراحی برای یک شیء مشخص می‌کند. پایهٔ مواد اولیه، انحرافات و مشخصات کارایی در کیفیت طراحی مطرح هستند. اگر محصول طبق مشخصات ساخته شود، به موازات استفاده از مواد اولیه با پایهٔ بالاتر، انحرافات محدودتر و با مشخص شدن سطوح بالاتری از کارایی، کیفیت طراحی محصول افزایش می‌یابد.

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 توسعهٔ نرم‌افزار، کیفیت طراحی شامل خواسته‌ها، مشخصه‌ها و طراحی سیستم می‌شود. کیفیت تطبیق مسئله‌ای است که اساساً بر پیاده‌سازی تأکید دارد. اگر پیاده‌سازی از طراحی پیروی کرده و سیستم حاصل، خواسته‌ها و اهداف کارایی را برآورده سازد، کیفیت تطبیق بالا است.

1 - Software Quality assurance

۱۹ـ۱ـ۲ کنترل کیفیت

کنترل گوناگونی ممکن است با **کنترل کیفیت** یکسان باشد. ولی چگونه به کنترل کیفیت دست پیدا کنیم؟ کنترل کیفیت عبارت از یک‌سری بازرسی‌ها، بازبینی‌ها 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.

۱۹ـ۱ـ۳ تضمین کیفیت

تضمین کیفیت شامل بازرسی و گزارش وظایف مدیریت است. هدف تضمین کیفیت فراهم آوردن داده‌های مربوط به کیفیت محصول برای مدیریت و درنتیجه 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.

۱۹ـ۱ـ۴ 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.

۱۹ـ۱ 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 توسعهٔ 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.

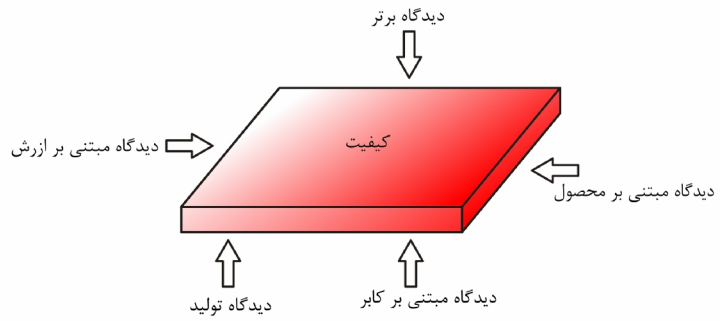
۱۹ـ۲ مدل‌های سلسله‌مراتبی کیفیت

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

۱۹ـ۲ـ۱ دیدگاه‌های کیفیتی Garovin

کیفیت دارای ساختار چندبُعدی 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:



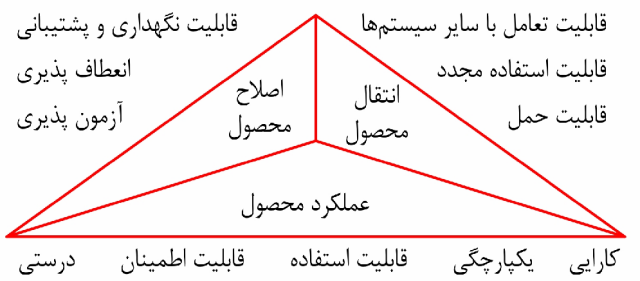
شکل ۱۹ـ۱ دیدگاه‌های کیفیت مطرح‌شده توسط Garovin

۱۹ـ۲ـ۲ 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 ناحیهٔ 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 ۱۹ـ۲ معیارهای کیفیت مدل MacCall

۱۹ـ۲ـ۳ 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**[1]

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

- **قابلیت استفاده**[2]

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

- **قابلیت اعتماد**[3]

از طریق اندازه‌گیری تکرار (توالی) و شدت شکست، درستی و دقت نتایج خروجی، متوسط زمان شکست (MTTF)[4]، 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**[1]

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

- **قابلیت پشتیبانی**[2]

ترکیبی است از توانایی گسترش برنامه[3]، تطبیق‌پذیری[4]، سرویس‌پذیری[5]، به علاوه قابلیت آزمون[6]، سازگاری[7]، قابلیت پیکربندی[8]، سهولت نصب سیستم و سهولت مکان‌یابی شکست‌ها.

۱۹ـ۲ـ۴ مدل Bohem(۱۹۷۸)

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

۱۹ـ۲ـ۵ مدلPerry(۱۹۸۷)

مدلPerry 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.

۱۹ـ۳ 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.

۱۹ـ۳ـ۱ 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 وظیفهٔ 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 برنامه‌ریزی‌شده، 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 پروژهٔ 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

۱۹ـ۴ـ۲ 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.

۱۹ـ۴ 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.

۱۹ـ۵ بازبینی فنی رسمی

بازبینی فنی رسمی (FTR)[1] 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.

۱۹ـ۵ـ۱ 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؟

1 - Formal Technical Rview

۱۹ـ۶ اندازه‌گیری کیفیت

۱۹ـ۶ـ۱ قابلیت اعتماد نرم‌افزار

قابلیت اعتماد نرم‌افزار[1] به صورت احتمال عملکرد بدون شکست یک برنامه رایانه‌ای در محیطی مشخص برای یک زمان معین تعریف می‌شود. یعنی:

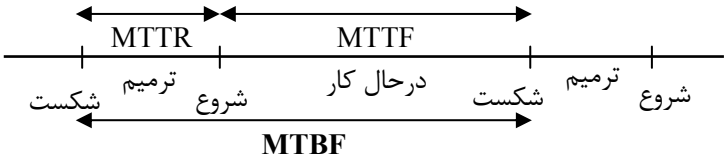
$$SR = ۱ - P \quad \textbf{(شکست یک برنامه رایانه‌ای در محیط مشخص در زمان معین)}$$

اندازه‌گیری قابلیت اعتماد

در یک سیستم رایانه‌ای، یک معیار ساده قابلیت اعتماد، متوسط زمان بین شکست‌ها (MTBF)[2] است که از رابطه زیر به دست می‌آید:

$$\mathbf{MTBF = MTTF + MTTR}$$

MTTF و MTTR به ترتیب متوسط زمان تا وقوع خرابی[3] و متوسط زمان ترمیم یا رفع خرابی[4] 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، MTTRو MTBF در شکل زیر نشان داده شده است.



شکل ۱۹ـ۳ رابطهٔ بین MTBF با MTTR و MTTF

۱۹ـ۶ـ۲ اندازه‌گیری قابلیت دسترسی

در دسترس بودن نرم‌افزار[5] 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. درصورتی‌که معیار در دسترس بودن بیشتر به MTTR حساس است، که نشان‌دهندهٔ قابلیت نگهداری نرم‌افزار است.

۱۹ـ۶ـ۳ امنیت نرم‌افزار

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

اگرچه قابلیت اعتماد و امنیت نرم‌افزار تا حد زیادی به یکدیگر نزدیک هستند، درک این نکته که بین آن‌ها تفاوت‌های پنهان بسیاری وجود دارد مهم است. قابلیت اعتماد نرم‌افزار از تحلیل‌های آماری برای تعیین احتمال شکست نرم‌افزار استفاده می‌کند. به هر حال، وقوع شکست لزوماً خطرساز نیست. امنیت نرم‌افزار 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 نرم‌افزار

تعریف Babich: هنر هماهنگ‌سازی توسعهٔ نرم‌افزار برای به حداقل رساندن سردرگمی را مدیریت پیکربندی نرم‌افزار (SCM)[1] می‌گویند. مدیریت پیکربندی هنر شناسایی، سازماندهی و کنترل تغییرات و اصلاحاتی است که باید در یک نرم‌افزار در حال توسعه توسط تیم مهندسی نرم‌افزار اعمال شوند. هدف به حداکثر رساندن بهره‌وری در کنار به حداقل رساندن اشتباهات است.

مدیریت پیکربندی نرم‌افزار یک فعالیت پشتیبانی است که در سرتاسر فرایند نرم‌افزار قابل اجراست. چون امکان تغییر در هر زمانی وجود دارد بنابراین فعالیت‌های SCM باید به صورت زیر اجرا شوند:

۱ـ شناسایی تغییر

۲ـ کنترل تغییر

۳ـ حصول اطمینان از پیاده‌سازی مناسب تغییر

۴ـ تهیه و اعلام گزارش تغییر به ذینفعان پروژه

۲۰ـ۱ مدیریت پیکربندی نرم‌افزار

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

۱ـ برنامه‌های رایانه‌ای (چه در سطح کُدهای منبع و چه اجرایی)

۲ـ مستنداتی که برنامه‌های رایانه‌ای را توصیف می‌کنند (چه در بُعد فنی و چه در بُعد کاربر)

۳ـ داده‌ها (که یا در داخل برنامه‌ها جا دارند و یا در خارج آن‌ها قرار گرفته‌اند)

تمام موارد بالا را اقلام پیکربندی نرم‌افزار 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.

۲۰ـ۱ـ۱ منشأ تغییرات

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

1 - Software Configuration Management (SCM)

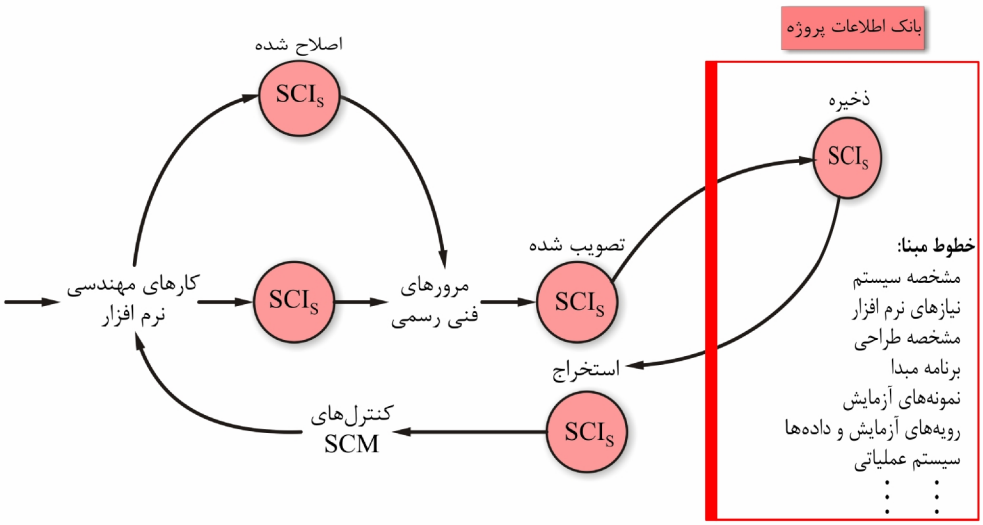
- نیازهای جدید مشتری که اصلاحاتی را در داده‌های تولیدشده توسط سیستم‌های اطلاعاتی، عملکرد تحویل‌شده توسط محصولات یا خدمات تحویل‌شده را توسط یک سیستم رایانه‌ای طلب می‌کنند.
- سازماندهی مجدد یا رشد / زوال تجاری که باعث تعریف مجدد محصول یا سیستم می‌شود.
- شرایط حدی بودجه یا زمان‌بندی که باعث تعریف مجدد محصول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

۲۰ـ۲ خط مبنا

تعریف مبنا (استاندارد IEEE 61012-1940)[1]: مشخصه یا محصولی است که رسماً مورد بازبینی و توافق قرار گرفته و پس از آن به عنوان مبنایی برای توسعهٔ 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 توسعهٔ نرم‌افزار است که با تحویل یک یا چند قلم پیکربندی نرم‌افزار (SCIs)[2] و تصویب آن‌ها از طریق بازبینی فنی رسمی شناسایی می‌شود. برای مثال، عنصر مشخصهٔ طراحی عبارت است از تمام اسناد فنی که مستندسازی و مرور شده‌اند، خطاهایی که پیدا و آشکار شده‌اند و... . پس از اینکه تمام بخش‌های این مشخصه مرور، تصحیح و مورد تأیید قرار گرفت، آن‌گاه مشخصهٔ طراحی به یک مبنا تبدیل می‌شود. تغییرات بیشتر در معماری برنامه (که در مشخصهٔ 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.



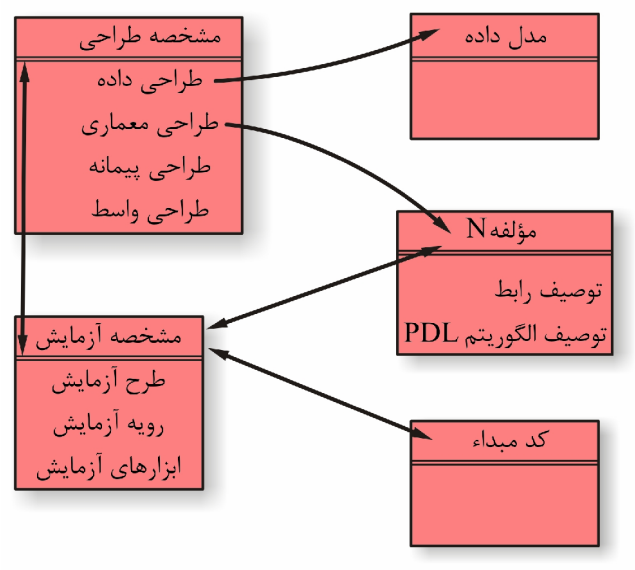
شکل ۲۰ـ۱ SCI های مبنا و بانک اطلاعاتی پروژه

۲۰ـ۳ اقلام پیکربندی

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

درعمل، 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 ۲۰ـ۲ اشیای پیکربندی نرم‌افزار

۲۰ـ۴ فرایند SCM

مدیریت پیکربندی نرم‌افزار عنصر مهمی از تضمین کیفیت نرم‌افزار است. مسئولیت اولیهٔ آن کنترل تغییر است. به هرحال فرایند SCM مسئول شناسایی اقلام پیکربندی نرم‌افزار، کنترل نسخ، کنترل تغییرات، بازرسی 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 نرم‌افزار به مشتری کنترل می‌کند؟
- سازمان چگونه تغییرات را قبل و بعد از ارائه نرم‌افزار 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.

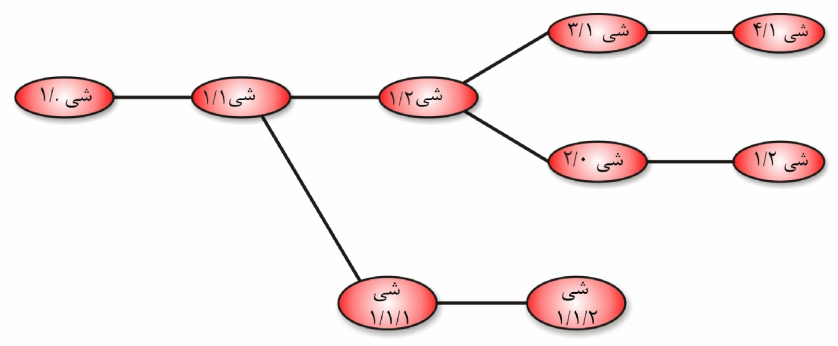
۲۰ـ۴ـ۱ شناسایی اشیای پیکربندی نرم‌افزار

برای کنترل و مدیریت اقلام پیکربندی نرم‌افزار، 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 نرم‌افزار در تحلیل، طراحی، کُد نویسی، یا آزمایش ایجاد شده است. برای مثال، یک شیء پایه می‌تواند یک ویرایش از مشخصهٔ نیازها، یک لیست از برنامه مبدأ یک مؤلفه، یا یک گروه از نمونه‌های آزمایش باشد. یک شیء ترکیبی، گروهی از اشیای پایه و اشیای مرکب دیگر است. با مراجعه به شکل صفحهٔ قبل، مشخصهٔ طراحی، یک شیء ترکیبی است. از نظر مفهومی، می‌تواند به صورت لیستی دارای نام از اشاره‌گرهایی باشد که اشیای پایه‌ای مانند مدل داده و مؤلفهٔ N را مشخص می‌کنند.

۲۰ـ۴ـ۲ کنترل نسخه

مدیریت پیکربندی به کاربر امکان می‌دهد تا پیکربندی‌های جایگزین سیستم نرم‌افزاری را از طریق انتخاب نسخ مناسب اقلام مشخص کند. این عمل با همراه کردن صفات با هر نسخه از نرم‌افزار پشتیبانی شده و سپس یک پیکربندی با توصیف مجموعهٔ مناسب صفات، مشخص و ساخته می‌شود.

یک نمایش از نسخ متعدد یک سیستم، مانند گراف تکاملی ارائه‌شده در شکل زیر است. هر گره در این گراف یک شیء مرکب (ترکیبی) است، یعنی نسخهٔ کاملی از نرم‌افزار. هر نسخه از این نرم‌افزار مجموعه‌ای از SCIها است (کد مبدأ، اسناد و داده‌ها) که ترکیبی از تغییرات مختلف است.



شکل ۲۰ـ۳ گراف تکامل اقلام پیکربندی نرم‌افزار

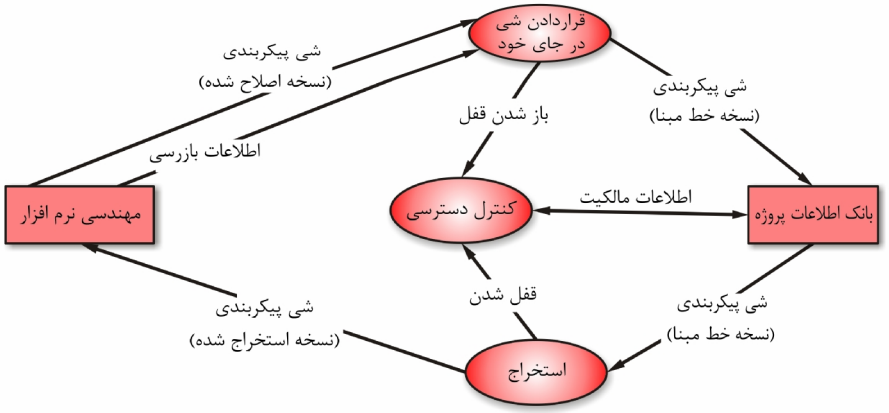
۲۰ـ۴ـ۳ کنترل تغییر

مدیریت پیکربندی این امکان را به کاربر می‌دهد که پیکربندی‌های متفاوتی از سیستم‌های نرم‌افزاری را از طریق گزینش نسخه‌های مناسب، مشخص کند. این امر با نسبت دادن صفاتی به هر نسخه نرم‌افزار و سپس مشخص کردن پیکربندی [و بنا نهادن آن] با توصیف مجموعه‌ای از صفات مطلوب صورت می‌پذیرد.

فرایند استخراج و قرار دادن اقلام پیکربندی در بانک اطلاعاتی پروژه، دو عنصر مهم از کنترل تغییرات محسوب می‌شود.

- **کنترل دسترسی** مشخص می‌کند کدام‌یک از مهندسان نرم‌افزار اجازهٔ 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 ۲۰ـ۴ کنترل دسترسی و همزمانی

بر مبنای درخواست تغییر موافقت‌شده و تهیهٔ دستور تغییر مهندسی (ECO)[1] ، مهندس نرم‌افزار یک شیء پیکربندی را خارج می‌کند. یک تابع کنترل دسترسی مطمئن می‌سازد که این مهندس نرم‌افزار اجازهٔ برداشتن این شیء را دارد و کنترل همزمانی، این شیء را در بانک اطلاعاتی پروژه قفل می‌کند به‌گونه‌ای که دیگر هیچ تغییری بر روی آن قابل انجام نباشد، تا زمانی که نسخهٔ جاری خارج‌شده، مجدد در بانک پروژه جایگزین شود. یک کپی از شیء مبنا، در اختیار مهندس نرم‌افزار، قرار می‌گیرد و اصلاح می‌شود. بعد از انجام SQA مناسب و آزمایش، نسخهٔ اصلاح‌شدهٔ این شیء وارد بانک می‌شود و شیء مبنای جدید از حالت قفل خارج می‌شود.

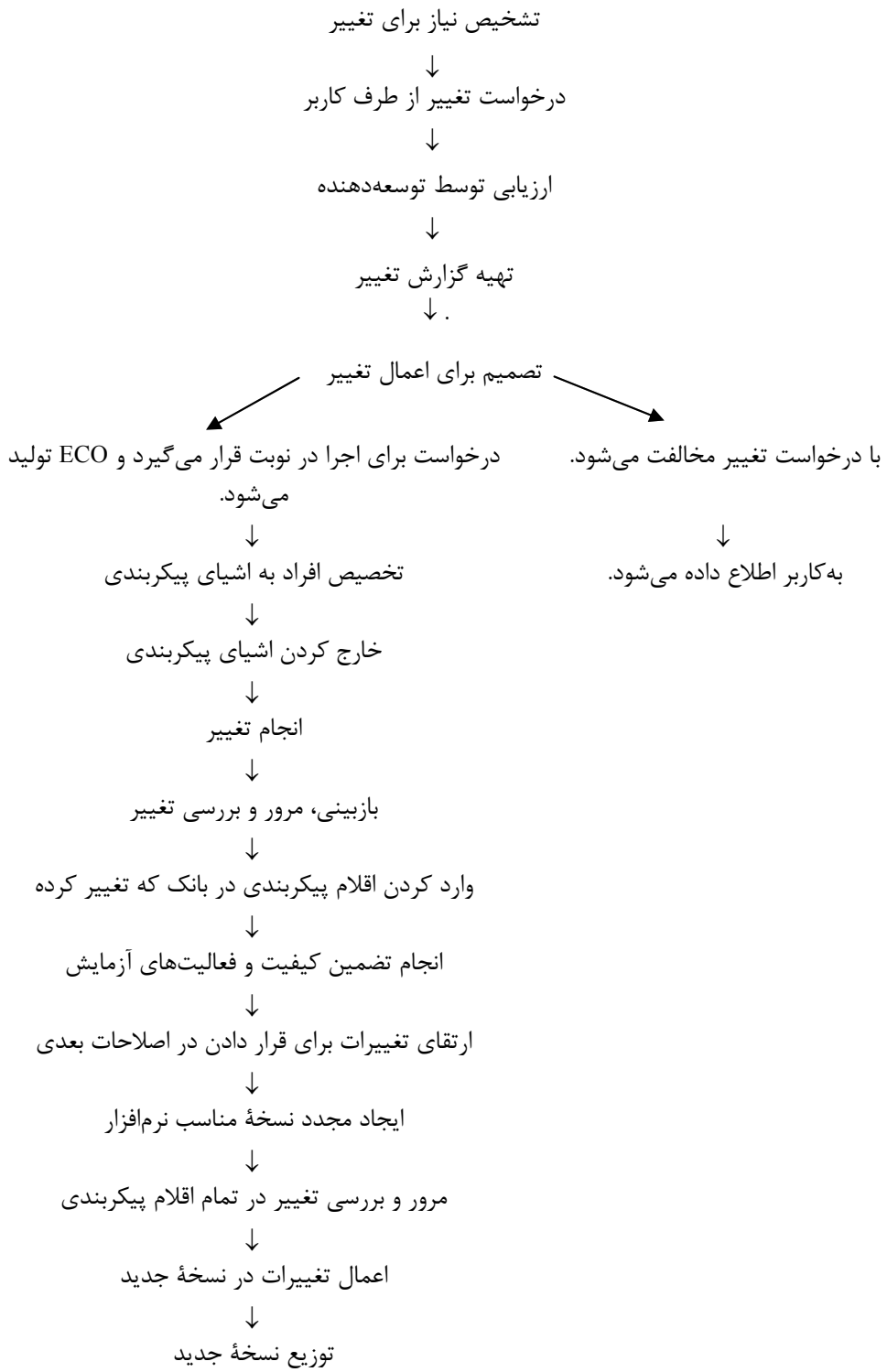
1 - Engineering Change Order

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)[1] قرارمی‌گیرد، 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.

1 - Change Control Authority

راهکارهای کنترل نسخهٔ مناسب برای ایجاد نسخهٔ بعدی نرم‌افزار به کار گرفته می‌شوند.



شکل ۲۰ـ۵ فرایند کنترل تغییر

۲۰ـ۴ـ۴ بازرسی پیکربندی

شناسایی، کنترل نسخه و کنترل تغییرات به نرم‌افزارنویس کمک می‌کند تا نظم امور را حفظ کند. ولی، حتی مؤثرترین راهکارهای کنترل نیز یک تغییر را تا جایی پیکربندی می‌کنند که یک ECO تولید شود. چگونه می‌توان اطمینان یافت که تغییر به طور مناسب پیاده‌سازی شده است؟ این کار با انجام دو فعالیت زیر صورت می‌گیرد:

۱- بازبینی‌های فنی رسمی
۲- بازرسی پیکربندی نرم‌افزار

در حین بازبینی و بازرسی سؤالاتی مطرح و پاسخ آن‌ها داده می‌شود:

۱- آیا تغییر مشخص‌شده 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 نرم‌افزار دنبال و استانداردهای مهندسی نرم‌افزار به طور مناسب به اجرا درآمده‌اند؟
۴- آیا تغییر در SCI نمایان و مشهود است؟ آیا داده‌های تغییر و صاحب تغییر مشخص شده‌اند؟ آیا صفات شیء پیکربندی این تغییر را منعکس می‌کنند؟
۵- آیا روال‌های SCM برای شناسایی تغییر، ثبت آن و گزارش آن دنبال شده‌اند؟
۶- آیا همهٔ SCIهای مربوطه به طور مناسب به‌هنگام‌سازی شده‌اند؟

۲۰ـ۵ـ۵ تهیه گزارش وضعیت پیکربندی

گزارش پیکربندی (که گاه صورت وضعیت تغییرات نیز خوانده می‌شود) یک فعالیت SCM است که به سؤالات زیر پاسخ می‌دهد:

- 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 پروژه‌های نرم‌افزاری بزرگ ایفا می‌کند که مهندسان نرم‌افزار باید به آن توجه کافی داشته باشند.