مهندسی نرم افزار

مدرس: یوسفی

دانشگاه آزاد واحد مرند

منابع

- مهندسی نرم افزار، اثر روگراس پرسمن، ترجمه جعفر نژاد قمی، ابراهیم عامل محرابی
- مهندسی نرم افزار اثر یان سامرویل، ترجمه عین الله جعفر نژاد قمی

محصول

نرم افزار کامپیوتری چیست؟

محصولی است که مهندسین نرم افزار آن را طراحی و تولید می کنند.

برنامه هایی است که در یک کامپیوتر با هر اندازه و ساختاری اجرا می شوند.

مستنداتی هستند که کد برنامه ها، شکل ها و چارت ها را در بر گرفته است.

داده هایی هستند که متشکل از ارقام و متون می باشند اما مشتمل بر نمایش تصویری، ویدئویی و اطلاعات نیز هستند.

محصول

دید مهندس نرم افزار و دید کاربر به محصول ساخته شده چگونه است؟

از دیدگاه مهندس کامپیوتر محصول به دست آمده برنامه ها، مستندات و داده هایی است که نرم افزار کامپیوتر را تشکیل می دهد.

اما از دیدگاه کاربر، محصول، اطلاعات به دست آمده است که به نوعی به درد کاربرمی خورد.

خصوصیات نرم افزار

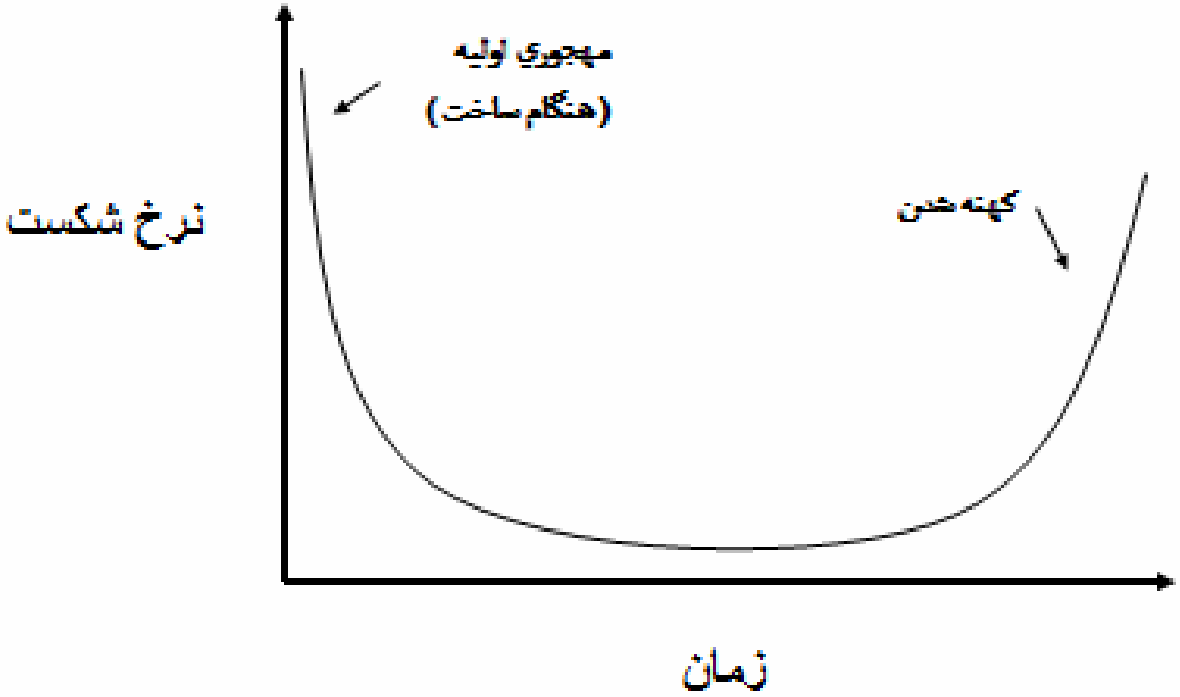
نرم افزار بیشتر یک عنصر منطقی است تا یک سیستم فیزیکی بنابراین دارای مشخصه هایی است که تا حد زیادی از مشخصه های یک سخت افزار متفاوتند.

۱- نرم افزار توسعه می یابد یا طراحی می شود، اما به مفهوم کلاسیک ساخته نمی شود.

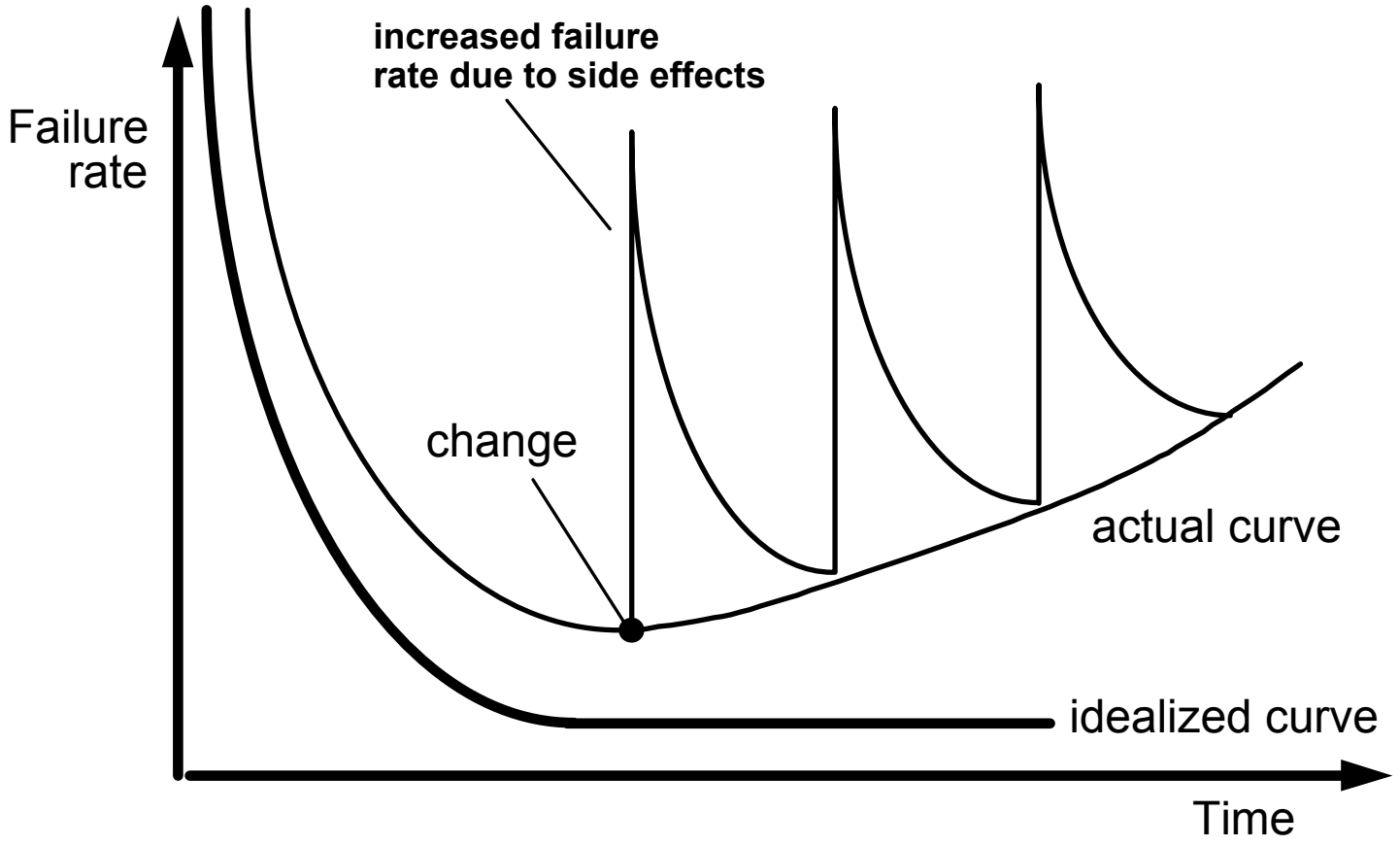
۲- نرم افزار فرسوده نمی شود.

۳- گرچه صنعت به سمت مونتاژ اجزاء حرکت می کند، اما نرم افزارهمچنان بصورت سفارشی ساخته می شود.

منحنی شکست سخت افزار



نمودار نرخ شکست نرم افزار



محتوا و قطعیت اطلاعات

محتوای نرم افزار اشاره به معنی و شکل اطلاعات ورودی و خروجی دارد.

قطعیت اطلاعات به قابلیت پیش بینی سفارش و زمانبندی اطلاعات اشاره دارد.

انواع نرم افزار

۱- نرم افزار سیستم

مانند سیستم عامل که برای سرویس دهی به سایر نرم افزار ها استفاده می شود.

۲- نرم افزار بلادرنگ

نرم افزاری که نظارت، تحلیل و کنترل رویدادهای جهان واقعی را برعهده دارد نرم افزار بلادرنگ نامیده می شود.

۳-نرم افزار تجاری

بیشتر پردازش اطلاعات تجاری و بازرگانی را انجام می دهد.

انواع نرم افزار

۴- نرم افزار مهندسی و علمی

برای حل یک مساله علمی

۵- نرم افزار تعبیه شده

مانند نرم افزارهایی که در ماکروفر و یا کلاً دستگاههای ROMدار استفاده می شود.

نرم افزار جاسازی شده می تواند کارهای بسیار محدود و سریع را انجام دهد (مثل صفحه کلید کنترل در ماکروفر، صفحه کلید داشبورد، کنترل سوخت خودرو و غیره).

انواع نرم افزار

۶- نرم افزار کامپیوتر شخصی

مانند WORD

۷- نرم افزار مبتنی بر وب

نرم افزارهایی که دستورهای قابل اجرا و پروتکل های اینترنتی را اجرا

می کنند مانند CGI HTML, Perl, Java

انواع نرم افزار

۸- نرم افزار هوش مصنوعی

نرم افزار هوش مصنوعی یا AI از الگوریتم های غیر عددی برای حل مسائل پیچیده استفاده می کند.

نرم افزارهایی که مبتنی بر دانش بوده و الگوشناسی می کند. شناسایی الگو(تصویر و صدا)، شبکه های عصبی مصنوعی، حل قضیه و بازیهای کامپیوتری نمونه هایی از برنامه های کاربردی در این گروه می باشند.

شرایط نیاز به تغییر سیستم

- تغییر اهداف سازمان
- تغییر وظایف سازمان
- تغییر تکنولوژی (سخت افزار، نرم افزار، ارتباطات)
- تغییرات محیطی

نرم افزار های موجود چرا باید تغییر کنند؟

- باید با محیطهای محاسباتی و فناوریهای جدید مطابقت داشته باشند
- نیازمندیهای جدید سازمان را برآورده کنند
- باید توسعه داده شوند تا قادر به همکاری با سیستمها و بانکهای اطلاعاتی جدید باشند
- باید در معماری نرم افزار تجدید نظر شود تا در محیط شبکه قابلیت سرویس دهی را داشته باشد

ضوابط ارزیابی نرم افزار

عوامل موثر در خوب بودن نرم افزار:

- عوامل خارجی: توسط کاربر نرم افزار تشخیص داده می شود.
- عوامل داخلی: برای متخصصین کامپیوتر قابل درک است.

عوامل موثر در خوب بودن نرم افزار

- عوامل خارجی:
  - صحت برنامه (Correctness)
  - استحکام (Robustness): جوابگویی در شرایط غیرعادی
  - قابلیت توسعه (Extendibility)
  - قابلیت مصرف مجدد (Reusability)
  - سازگاری (Compability): رعایت استاندارد
  - قابلیت حمل (Portability): اجرا در سیستم عامل و سخت افزارهای گوناگون
  - کارآیی (Efficiency): سرعت بالا و حافظه پایین
- عوامل داخلی: واحد بندی

متدلوژی

به مجموعه ای از قواعد و رویه ها که به چرخه زیست سیستم ساختار می دهد متدلوژی می گویند .

فرایند مهندسی نرم افزار

مجموعه ای از قدمهای قابل پیش بینی برای توسعه نرم افزار را مشخص می کند.

فصل دوم

فرآیند

نگاه اجمالی

- فرایند چیست؟
- چه کسی این کار را انجام می دهد؟
- چرا مهم است؟
- مراحل کار کدام هست؟
- محصول کار چیست؟
- چگونه مطمئن شوم که کاری که انجام داده ام درست است؟

محصول کار چیست؟

از نقطه نظر مهندسی نرم افزار، محصولات کاری، برنامه ها، اسناد و داده هایی است که در نتیجه فعالیتهای مهندسی نرم افزاری که در طول فرآیند صورت گرفته اند، تولید شده اند.

چگونه مطمئن شوم که کاری که انجام داده ام درست است؟

یک سری فرآیندها و مکانیزمهای ارزیابی نرم افزار وجود دارد که سازمانها را قادر می سازد تکامل فرآیند نرم افزاری خود را تعیین کنند.

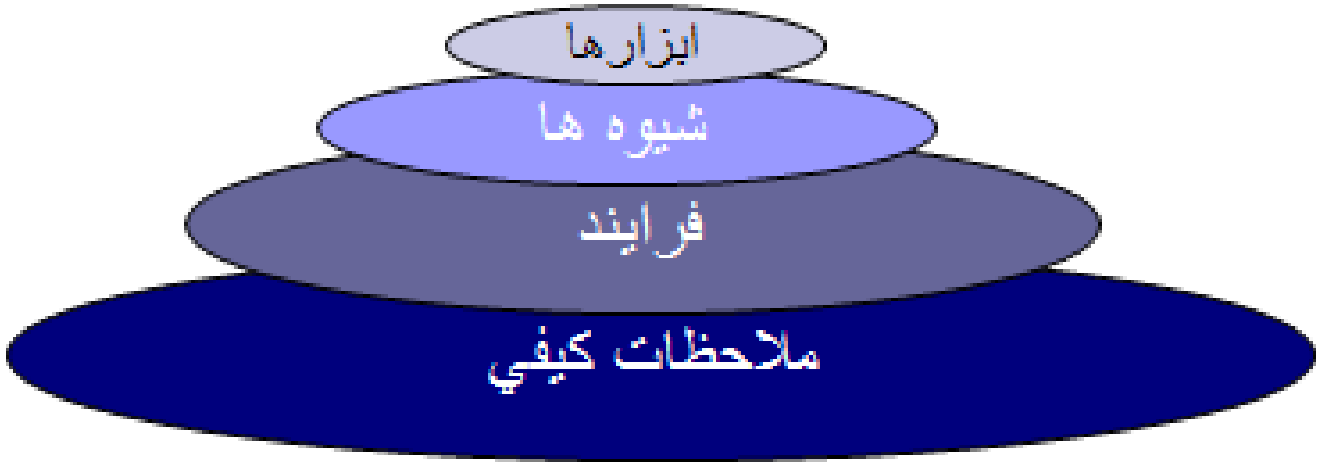
مهندسی نرم افزار

آیا مفهوم فرآیند با مهندسی نرم افزار یکسان است؟ پاسخ هم «بله» و هم «خیر» است فرآیند نرم افزار روشی را تعریف می کند که در هنگام طراحی نرم افزار بکار گرفته می شود.اما مهندسی نرم افزار نیز در بر گیرنده فناوریهای است که در فرآیند وجود دارند یعنی روشهای فنی و ابزار خودکار

مهندسی نرم افزار عبارتست از:

- ایجاد و استفاده از اصول ساده مهندسی به منظور رسیدن به یک نرم افزار مقرون به صرفه که قابل اطمینان بوده و روی دستگاههای واقعی کارآمد باشد.
- بکارگیری یک روش سیستماتیک، منظم و کمیت پذیر برای بسط، راه اندازی و نگهداری نرم افزار. یعنی بکارگیری مهندسی و شیوه های آن در نرم افزار.

لایه های مهندسی نرم افزار



فرایند، پایه و اساس لایه های مهندسی نرم افزار می باشد.

۱- روشها

عبارت است از شیوه های فنی برای ساخت نرم افزار که شامل وظایف زیر می باشد:

- تحلیل خواسته ها
- طراحی
- ساخت برنامه ها
- آزمایش و پشتیبانی

۲- ابزارها

متضمن پشتیبانی خودکار از فرایند و روشها می باشد و هنگامی که اطلاعات ایجاد شده به وسیله یک ابزار، توسط ابزارهای دیگر قابل استفاده باشد سیستمی برای پشتیبانی بسط نرم افزار شکل می گیرد که مهندسی نرم افزار به کمک کامپیوتر(case) نام دارد.

دید کلی از مهندسی نرم افزار

کارهای مربوط به مهندسی نرم افزار را بدون توجه به حیطه برنامه کاربردی، اندازه پروژه و پیچیدگی آن می توان به سه گروه کلی تقسیم کرد

- مرحله تعریف (چیستی)
- مرحله توسعه(چگونگی)
- مرحله پشتیبانی(تغییرات)

مرحله تعریف

این مرحله بر چیستی متمرکز می شود. یعنی در طول تعریف، مهندس نرم افزار سعی می کند تا موارد زیر را شناسایی کند:

- نوع اطلاعاتی را که باید پردازش شوند
- عملکرد مطلوب
- نوع وضعیت سیستم مورد انتظار
- نوع رابطه هایی که باید ایجاد شوند
- محدودیتهای موجود در طرح و معیارهای صحت را که برای تعریف یک سیستم موفق لازمند

اما سه کار اصلی که در مرحله تعریف به نحوی انجام می شوند:

۱.مهندسی سیستم یا اطلاعات

۲.طراحی پروژه نرم افزاری

۳.تحلیل مربوط به موارد مورد نیاز

مرحله توسعه

مرحله توسعه بر چگونگی (how) متمرکز است یعنی در طول این مرحله مهندس سعی دارد تا موارد زیر را توصیف کند.

- ساختارداده ها
- نحوه اجرای کار در معماری نرم افزار
- چگونگی توصیف واسط ها
- چگونگی تبدیل طرح به زبان برنامه نویسی یا زبان غیر رویه ای
- چگونگی انجام آزمون

روشهای بکار گرفته شده در طول مرحله توسعه متفاوتند اما سه کار فنی خاص همیشه رخ می دهند:

- **طراحی نرم افزار**
- **تولید کد**
- **آزمون نرم افزار**

مرحله پشتیبانی

روی تغییراتی متمرکز است که با اصلاح خطا، ایجاد تطابقات در حد لازم در رابطه با محیط نرم افزاری و تغییرات ناشی از بهبود کار بخاطر تغییر نیاز مشتری، در رابطه است.

در طول این مرحله چهار نوع تغییر را مشاهده می کنیم:

۱. اصلاح
۲. تطا بق
۳. بهبود وضعیت
۴. پیشگیری

سطوح بلوغ فرآیند نرم افزار تعریف شده توسط موسسه مهندسی نرم افزار(SEI)

- سطح ۱: شروع

فرآیند نرم افزاری بصورت موقتی و حتی بعضی اوقات بسیار درهم و برهم توصیف شده است. چند فرآیند تعریف می شود و موفقیت به تلاش های فردی بستگی دارد.

- سطح ۲: قابل تکرار

فرآیندهای اولیه مدیریت پروژه برای مشخص کردن هزینه، زمان بندی و قابلیت کارایی آن انجام می گیرند. اصل ضروروی فرآیند در جایی است که تکرار موفقیتهای قبلی روی پروژه هایی با کاربرد مشابه ضروری می باشد.

سطوح بلوغ فرآیند نرم افزار تعریف شده توسط SEI

- سطح ۳: تعریف شده

فرآیند نرم افزاری برای فعالیت های مدیریتی و مهندسی مستندسازی و استاندارد شده و همه پروژه ها از یک نسخه مستند و به تصویب رسیده استفاده می کنند.

- سطح ۴: مدیریت شده

موازین مفصلی از فرایند نرم افزار و کیفیت محصول جمع آوری شده و فرایند و نرم افزار از لحاظ کمی درک شده و همگی کنترل می شوند.

- سطح ۵: بهینه سازی

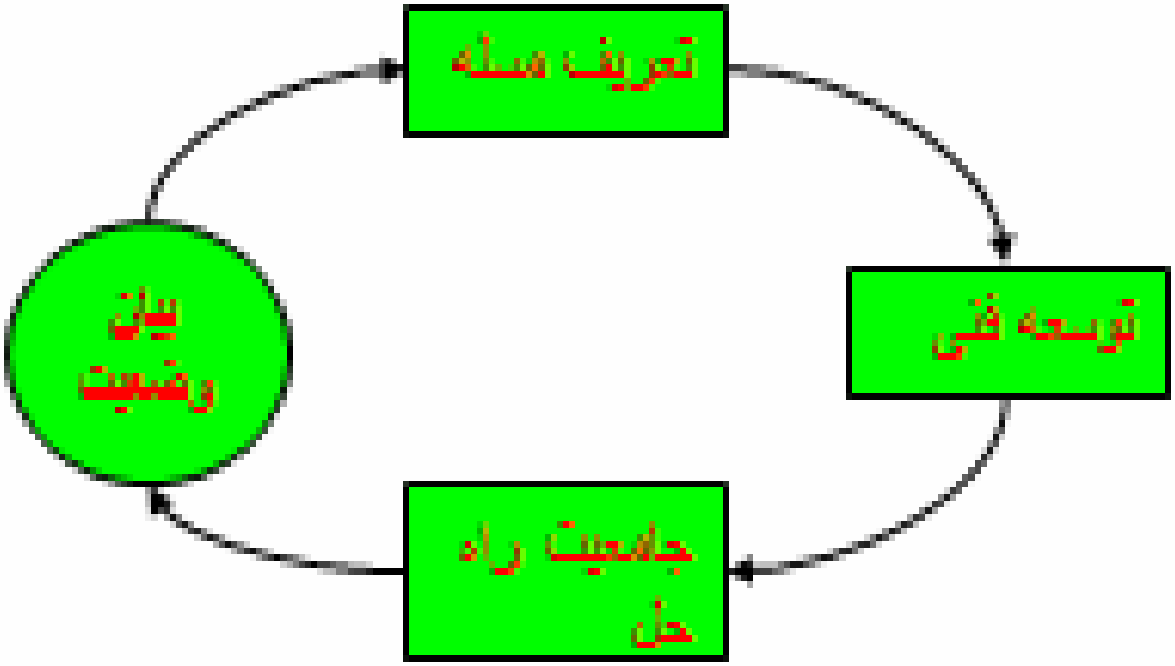
بهبود مکرر فرآیند با توجه به بازخورد کمی فرآیند و از روی آزمون ایده های نوآورانه و فناوریهای جدید، امکان پذیر است.

مدلهای فرآیند نرم افزار

انتخاب یک مدل فرایند برای مهندسی نرم افزار، بر اساس موارد زیر صورت می گیرد:

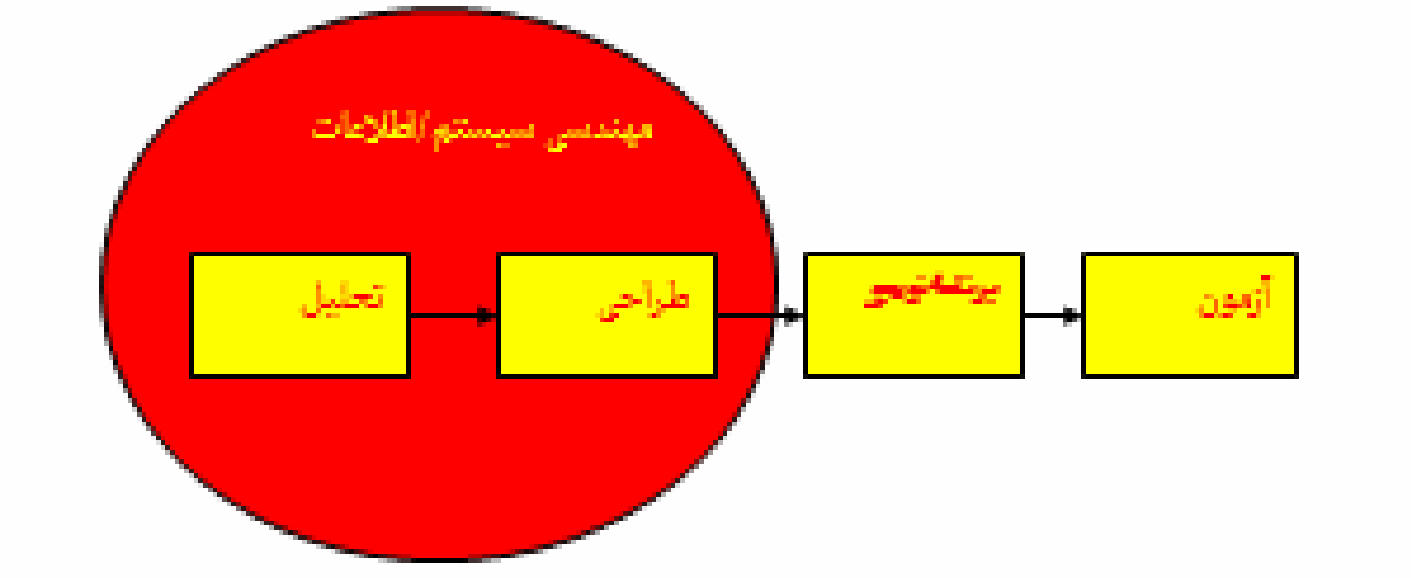
- ماهیت پروژه و نوع کاربرد
- روشها و ابزارهای مورد استفاده
- کنترل ها و قطعات قابل تحویل

مراحل توسعه نرم افزار



مدل ترتیبی خطی

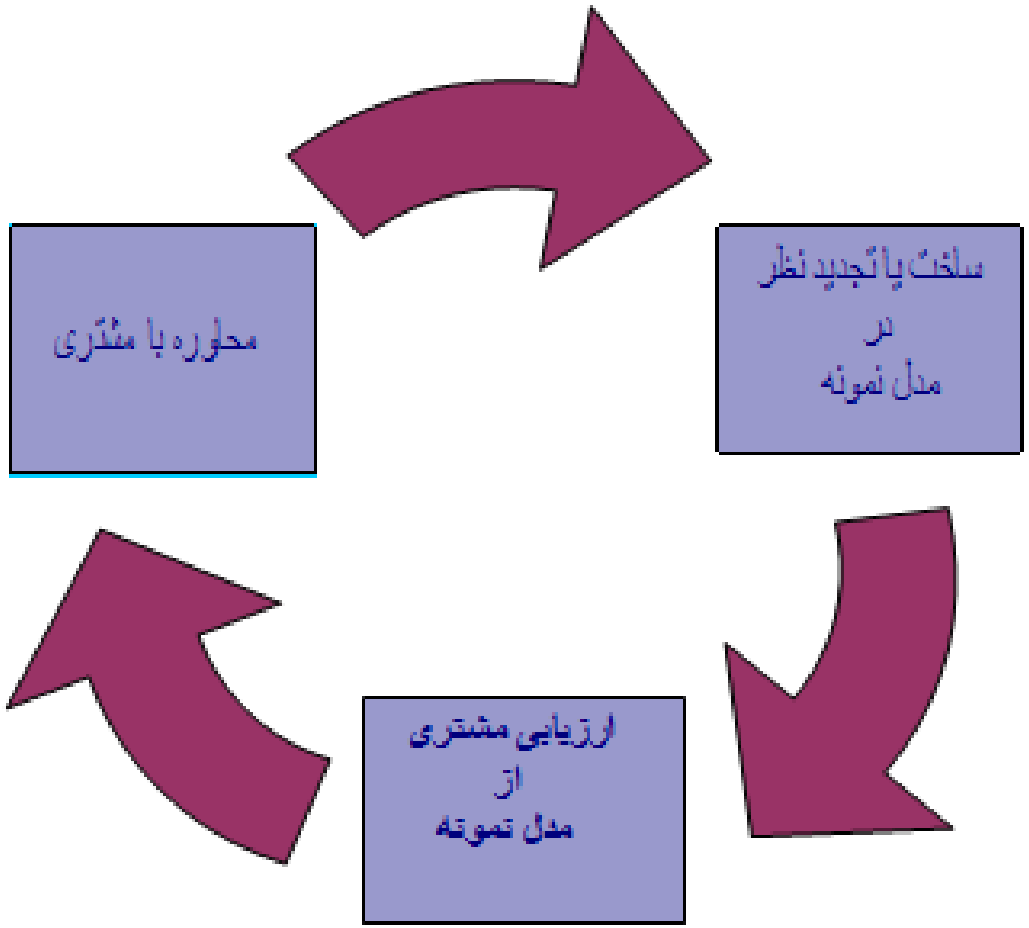
این مدل را که گاهی"مدل چرخه حیات کلاسیک" یا "مدل آبشاری" می نامند، بیانگر یک نگرش نظام مند و ترتیبی نسبت به تولید نرم افزار است که در سطح سیستم شروع شده و با تحلیل، طراحی، کد نویسی، آزمون و پشتیبانی نرم افزاری پیشروی می کند.



مشکلات مدل ترتیبی خطی

- پروژه های واقعی به ندرت از جریان ترتیبی و متوالی که مدل ارائه می دهد، پیروی می کند. گرچه مدل خطی می تواند با تکرار تطابق یابد، اما اینکار را به طور غیر مستقیم انجام می دهد. در نتیجه، با انجام کار توسط اعضای پروژه ممکن است موارد گیج کننده ای رخ دهد.
- اغلب برای مشتری مشکل است که تمام نیاز های خود را به طور مشخص بیان کند. مدل ترتیبی خطی به بیان واضح نیاز داشته و در صورت وجود موارد غیر قطعی که در آغاز بسیاری از پروژه ها وجود دارد، دارای مشکل است.
- مشتری باید صبور باشد. نسخه کاری برنامه تا اواخر مدت زمان کاری پروژه در دسترس نخواهد بود. اگر مشکلی وجود داشته و تا اواخر کار برنامه نویسی که برنامه مورد بازبینی قرار می گیرد مشخص نشود، می تواند فاجعه آمیز باشد.

مدل ساخت نمونه اولیه



مدل ساخت نمونه اولیه

مدل ساخت نمونه اولیه، بهترین روش برای حل مشکلاتی از قبیل:

- نامطمئن بودن سازنده از کارایی الگوریتم، قابلیت تطابق آن با یک سیستم عامل یا شکل ارتباط متقابل دستگاه و انسان.

- نمونه اولیه، نخست توسط مشتری /کاربر ارزیابی شده و از آن برای رفع نیازهای نرم افزاری که قرار است تولید شود استفاده می شود وقتی نمونه اولیه طوری تنظیم شد که نیازهای مشتری را برآورده سازد، تکرار رخ می دهد، در حالیکه در همان زمان تولید کننده نرم افزار را قادر می سازد تا نیاز های مشتری را بهتر درک کند.

مدل توسعه سریع برنامه ها(RAD)

Rapid Application Development

یک مدل فرایند افزایشی تولید نرم افزار است که تنها بر یک سیکل تولید بسیار کوتاه تاکید می کند.

مدل (RAD) یک نسخه پرسرعت از مدل ترتیبی خطی است که در آن توسعه سریع، با استفاده از ساخت مبتنی بر مولفه ها صورت می گیرد

فرایند RAD این امکان را به تیم می دهد که سیستم کاملا عملیاتی را ظرف مدت بسیار (مثلا ۶۰ تا ۹۰ روز) کوتاهی ایجاد کند.

شرایط مورد نیاز برای کاربرد مدل RAD

- موارد مورد نیاز به خوبی شناسایی شده باشد
- دامنه پروژه محدود باشد

- **مراحل مدل RAD**

•مدلسازی تجاری

جریان اطلاعات را در میان کارهای تجاری مدلسازی می کند.

•مدلسازی داده ای

جریان اطلاعاتی که به عنوان بخشی از مرحله مدلسازی تجاری تعریف شده در مجموعه ای از اهداف داده ای که برای پشتیبانی تجاری لازمند، پالایش می شود. مشخصه های هر هدف شناسایی شده و ارتباط بین این اهداف تعریف می شود.

•مدلسازی فرآیند

اهداف داده ای که در مرحله مدلسازی داده ها تعریف شده اند تغییر می یابند تا به جریان اطلاعاتی لازم برای اجرای یک کار تجاری، دست پیدا کنند. توضیحات عمل پردازش برای افزودن، شناسایی، حذف یا بازیابی یک شیء داده ای ارائه می شوند.

مراحل مدل RAD

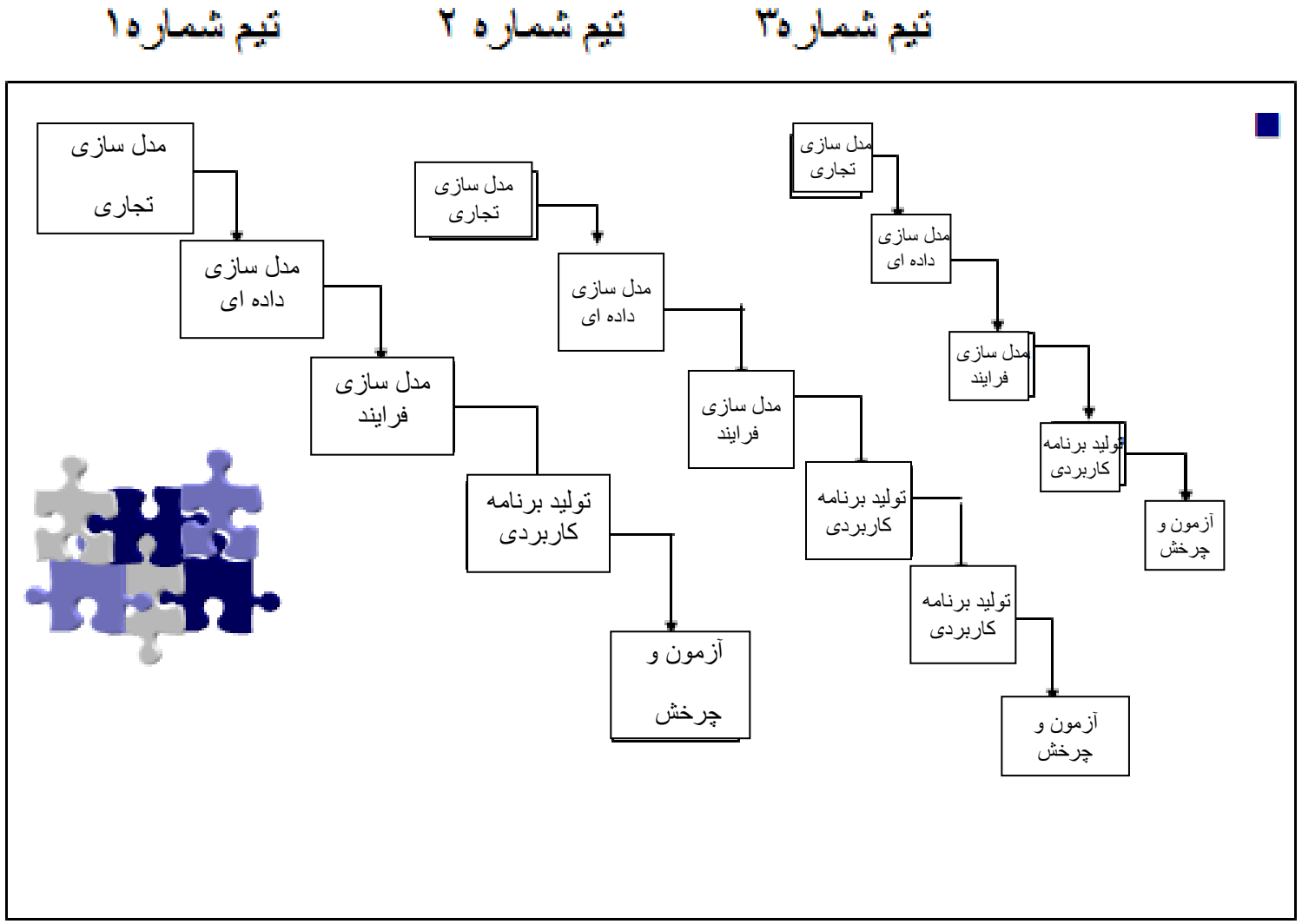
- **تولید برنامه کاربردی**

به جای تولید نرم افزار با استفاده از زبانهای متعارف برنامه نویسی متعارف نسل سوم فرآیندRAD تلاش می کند تا از جزء های برنامه موجود مجددا استفاده کرده یا جزء های قابل استفاده مجددی را تولید کند.

- **آزمون و گردش**

از آنجا که فرایند RAD براستفاده مجدد تأکید می کند بسیاری از اجزای برنامه مورد آزمون قرار گرفته اند.این کار در زمان کلی آزمون صرفه جویی می کند.

مدل RAD

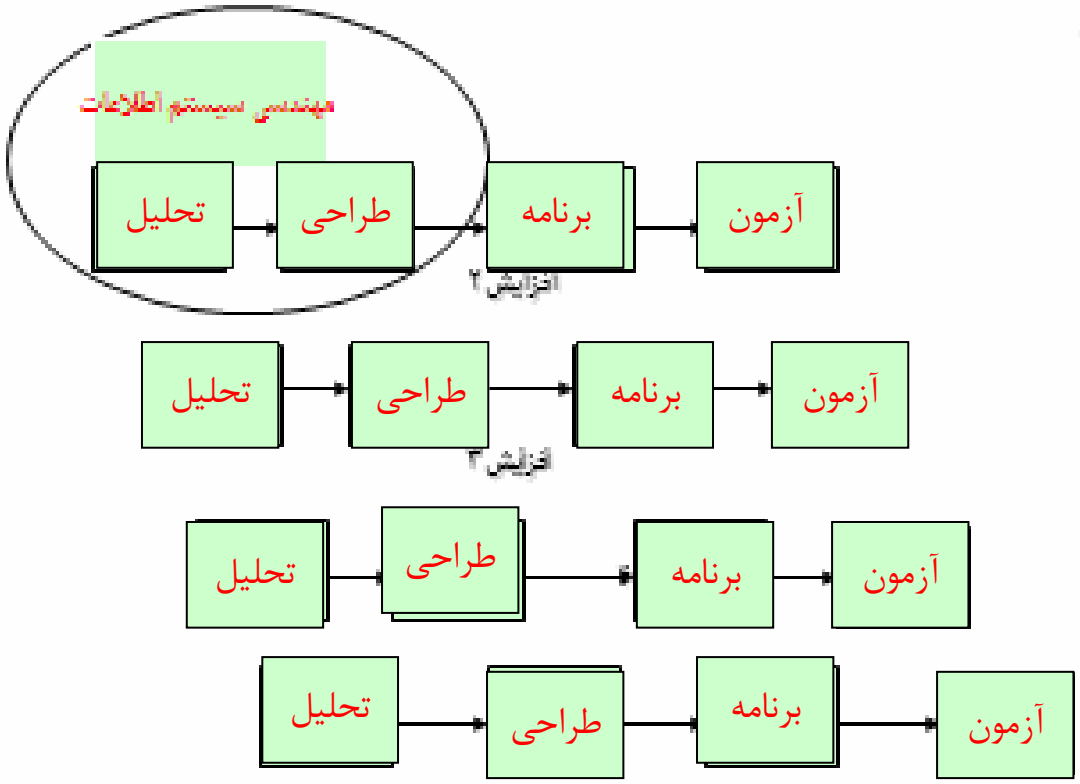

تیم شماره۱
تیم شماره ۲
تیم شماره۳
مدل سازی تجاری
مدل سازی داده ای
مدل سازی فرایند
تولید برنامه کاربردی
آزمون و چرخش
مدل سازی تجاری
مدل سازی داده ای
مدل سازی فرایند
تولید برنامه کاربردی
آزمون و چرخش
مدل سازی تجاری
مدل سازی داده ای
مدل سازی فرایند
تولید برنامه کاربردی
آزمون و چرخش

نقایص مدل فرایند RAD

- در مورد پروژه های بزرگ اما قابل اندازه گیری، RAD نیازمند منابع انسانی کافی است تا تعداد تیمهای درست RAD را تشکیل دهد.
- RAD نیازمند تولید کننده و مشتری هایی است که بتوانند فعالیت های سریع لازم را برای رسیدن به یک سیستم کامل در یک چارچوب زمانی کوتاه انجام دهند.

- همه نوع برنامه کاربردی برای RAD مناسب نیست. اگر سیستم را نتوان بطور مناسب تقسیم بندی کرد، ساختن اجزای لازم برای RAD مشکل ساز خواهد بود. اگر کارایی بالایی مد نظر باشد و نیل به کارایی از طریق تنظیم واسط ها به مولفه های سیستم تأمین شود روش RAD ممکن است جوابگو نباشد.
- RAD وقتی که خطرات فنی بسیار زیاد است مناسب نخواهد بود.

مدل گام به گام



افزایش۱

- تحویل نخستین افزایش
- تحویل دومین افزایش
- تحویل سومین افزایش
- تحویل چهارمین افزایش
- مدل افزایشی

مدل های تکاملی فرایند نرم افزار

- مدل گام به گام
- مدل پیچشی یا حلزونی
- مدل پیچشی بردبرد یا WINWIN
- مدل بسط همزمان
- مدل بسط مبتنی بر مولفه ها
- مدل روشهای رسمی
- تکنیک های نسل چهارم

مدل گام به گام

مدل گام به گام، عناصر مدل ترتیبی خطی را با دیدگاه تکرار مدل ساخت نمونه اولیه، ترکیب می کند.

اولین گام، ساخت هسته محصول است. یعنی نیازهای اولیه مورد توجه قرار می گیرند اما بسیاری از مشخصه های تکمیلی ارائه نمی گردند. هسته محصول به وسیله مشتری مورد استفاده قرار می گیرد.

این فرآیند بدنبال تحویل هر بخش تکرار می شود تا وقتی که محصول کامل تولید شود.

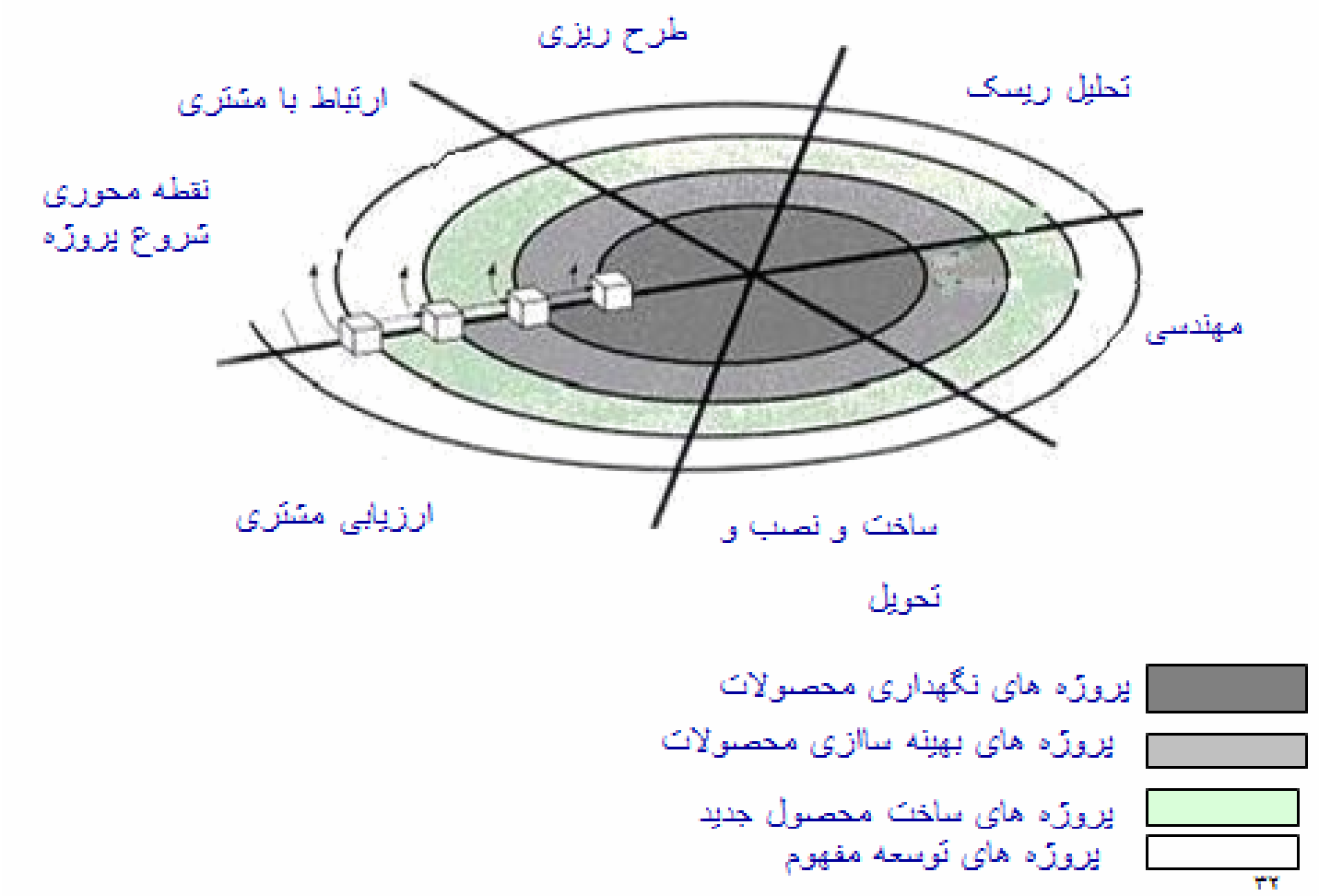
ویژگیهای مدل گام به گام

- مانند مدل ساخت نمونه اولیه و دیگر روش های ارزیابی و تکمیل، ماهیت تکرار شونده دارد.
- بر خلاف مدل ساخت نمونه اولیه، مدل گام به گام روی تحویل یک محصول عملیاتی در هر گام تأکید می کند.
- افزایشهای اولیه، نسخه های محدود شده محصول نهایی هستند.

مدل پیچشی یا مارپیچ(حلزونی)

- این مدل توسط بوهم پیشنهاد شد که ماهیت تکراری مدل ساخت نمونه اولیه را با جنبه های نظام مند و کنترل شده مدل ترتیبی خطی ترکیب می کند.
- با استفاده از مدل حلزونی، نرم افزار در یک سری نسخه های تکراری تولید می شود. در طول تکرارهای اولیه ممکن است نسخه اولیه، یک مدل روی کاغذ یا تنها الگوی اولیه باشد. در طول تکرارهای بعدی نسخه های نسبتا کاملتری از سیستم مهندسی شده و تولید می شود.

مدل حلزونی یا مارپیچ



شش منطقه کاری در مدل حلزونی

۱-ارتباط با مشتری: کارهای لازم برای ایجاد ارتباط مؤثر بین تولید کننده و مشتری.

۲- برنامه ریزی: کارهای لازم برای تعریف منابع، محدوده های زمانی و دیگر اطلاعات مربوط به پروژه.

۳- تحلیل خطر: کارهای لازم برای ارزیابی فنی و خطرات مدیریتی.

شش منطقه کاری در مدل حلزونی

٤- مهندسی و طراحی- کارهای لازم برای ساخت یک یا چند نمونه از برنامه کاربردی.

٥- ساخت و ارائه- کارهای لازم برای ساخت،آزمون، نصب و تهیه پشتیبانی برای کاربر(مثل مستند سازی و آموزش)

٦- ارزیابی مشتری- کارهای لازم برای بدست آوردن واکنش مشتری بر اساس ارزیابی نمونه های نرم افزاری تولید شده در طول مرحله مهندسی و اجرا شده در طول مرحله نصب و راه اندازی

مدل حلزونی WINWIN

۱-شناسایی سیستم یا دارندگان اصلی سیستم

۲-تعیین شرایط برد دارندگان سهم.

۳-مذاکره در مورد شرایط برد سهامداران برای تطابق دادن آنها درمجموعه ای از شرایط بردبرد برای همه افراد درگیر در پروژه.

مدل حلزونی WINWIN

۱-شناسایی واگذارندگان

سطح بعد

۲-شناسایی شرایط برد

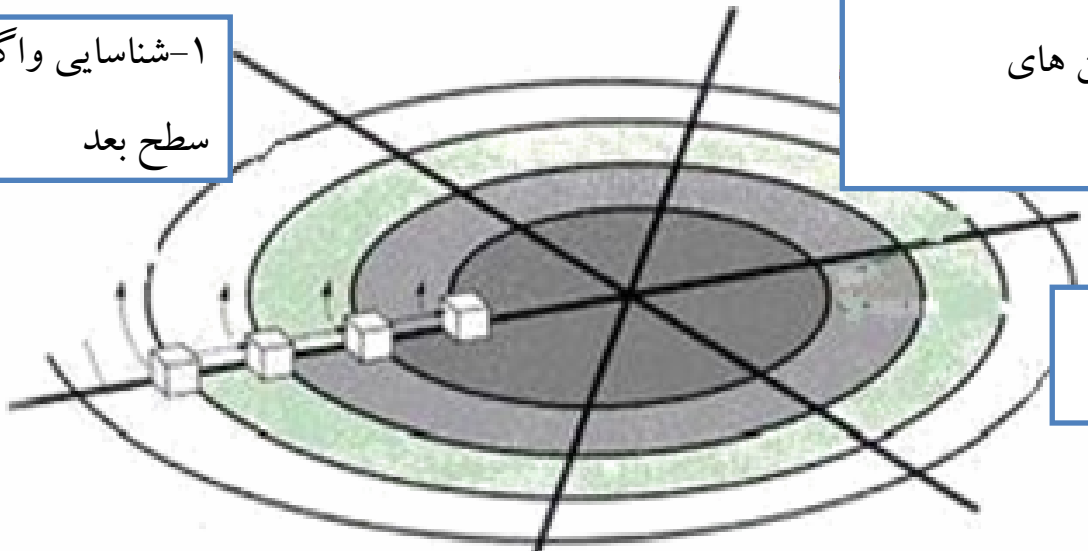
واگذارندگان

۳-الف-تطابق شرایط برد

۳-ب-برقراری اهداف اصلی

محدودیتها و قیود و جایگزین های

سطح بعد



۴-ارزیابی فرایند و محصول های

جایگزین و مرتفع کردن ریسک ها

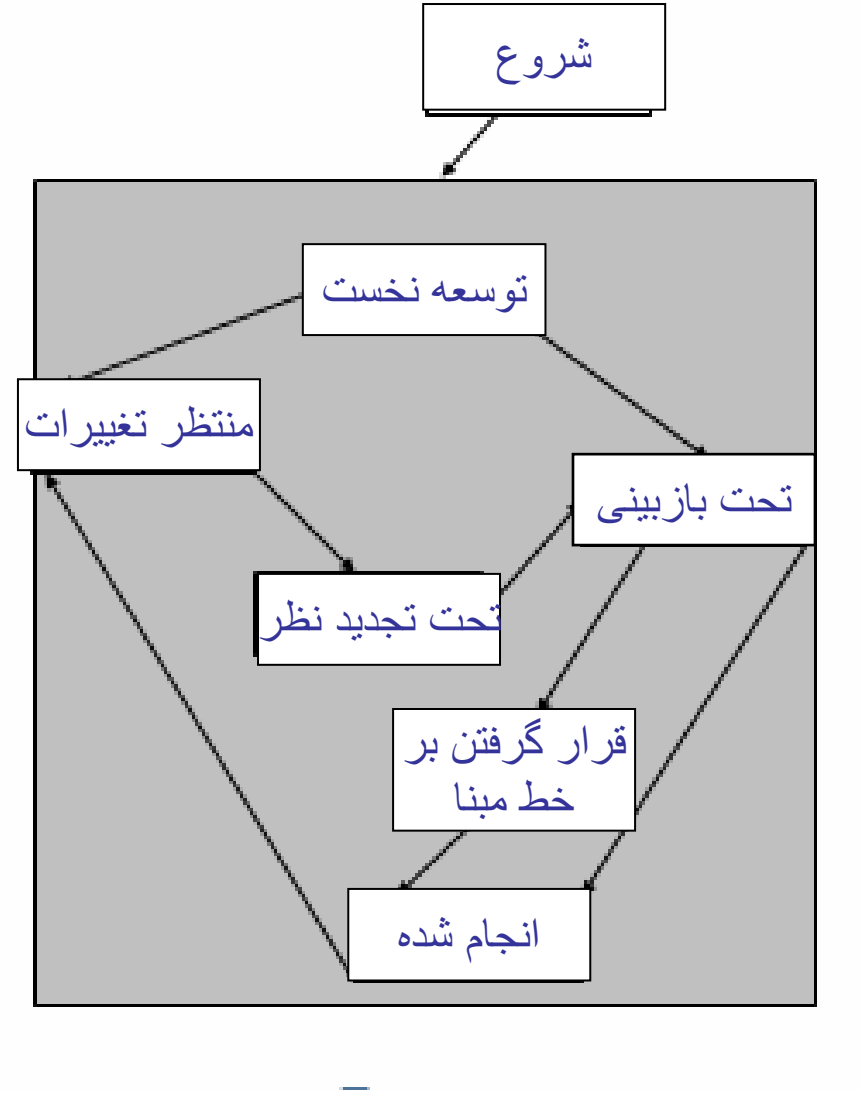
۵- تعریف سطح بعدی محصول و فرایند

مشتمل بر بخشها و قسمتها

۶- اعتبار سنجی محصول

و تعاریف فرایند

مدل فرآیند بسط همزمان



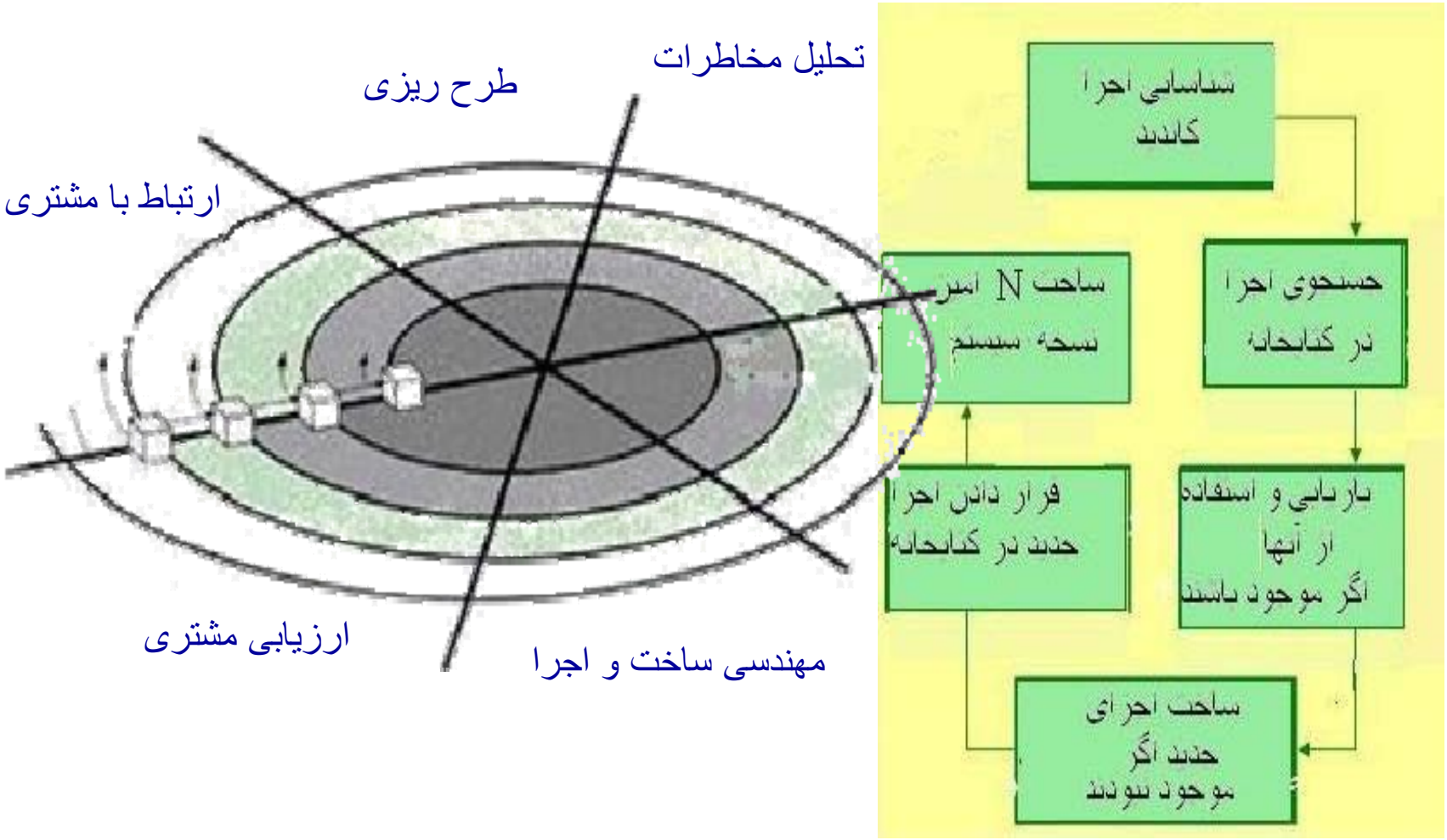
•مدل فرآیند همزمان یک سری رویدادهایی را مشخص می سازد که انتقال از حالتی به حالت دیگر را برای هر کار مهندسی نرم افزار آغاز می کنند.

• مدل فرآیند همزمان اغلب بعنوان معیاری برای توسعه و تولید برنامه های کاربردی مشتری/کارگزار استفاده می شود.

•وقتی مدل فرآیند همزمان در مشتری/کارگزار استفاده می شود، فعالیت هایی را در دو بعد تعریف می کند بعد سیستم و بعد مؤلفه

•موضوعات مربوط به سیستم با استفاده از سه فعالیت مورد توجه قرار می گیرند، طراحی، مونتاژ و کاربرد.

توسعه مبتنی بر مولفه ها



مدل روش های رسمی

این مدل دربرگیرنده یک سری فعالیت هایی است که منجر به تعیین رسمی و ریاضی مشخصه های نرم افزار می شود. روش های رسمی، مهندس نرم افزار را قادر می سازند که یک سیستم مبتنی بر کامپیوتر را با بکارگیری روابط ریاضی بسیار دقیق و مشخص توسعه داده و تغییر دهد. یک نوع از این روش به نام مهندسی نرم افزار اتاق تمییز نام دارد.

با اینکه مدل روش های رسمی بصورت یک روش حتمی و رایج در نیامده اما یک نرم افزار بدون مشکل را تضمین می کند. با این وجود مسئله توانایی آن در محیط تجاری مورد بحث قرار دارد.

مشکلات مدل روشهای رسمی

۱-توسعه روش های رسمی در حال حاضر کاملاً وقت گیر و گران است.

۲-از آنجا که تولید کنندگان کمی در مورد نرم افزار دارای پیش زمینه کافی برای استفاده از روش های رسمی هستند یک دوره آموزش پرهزینه لازم است.

۳-استفاده از مدل ها بعنوان یک مکانیزم ارتباطی برای مشتریان بی تجربه از نظر فنی مشکل است.

تکنیکهای نسل چهارم

اصطلاح تکنیکهای نسل چهارم (4GT)در بر گیرنده طیف گسترده ای از ابزارهای نرم افزاری است که در یک چیز مشترک هستند: هر کدام به مهندس نرم افزار این قدرت را می دهند که یک مشخصه نرم افزار را در سطح بالایی مشخص سازند. سپس ابزار به طور خودکار، کد منبع را بر اساس مشخصه های تولید کننده تولید می نماید.

یک محیط تولید نرم افزار که معیار 4GT را مورد پشتیبانی قرار می دهد شامل یک یا تعدادی از ابزارهای زیر است:

- زبانهای غیر رویه ای برای درخواست پایگاه داده
- ارائه گزارش
- دستکاری و تغییر داده ها
- تعریف صفحه نمایش و تعامل با آن
- تولید کد
- قابلیت گرافیکی سطح بالا
- قابلیت صفحه گسترده و تولید خودکار کد HTML و زبانهای دیگر مورد استفاده برای ایجاد سایت وب با استفاده از ابزارهای پیشرفته نرم افزاری.

تکنیکهای نسل چهارم یا 4GT

۱-استفاده از روش4GT روش بسیار ارزشمندی برای بسیاری از حوزه های مختلف در برنامه های کاربردی است.4GT همراه با ابزارهای مهندسی نرم افزار با کمک کامپیوتر(CASE) و مولدهای کد، یک راه حل خوب برای بسیاری از مشکلات نرم افزاری ارائه می دهد.

۲-اطلاعات جمع آوری شده از شرکتهایی که از4GT استفاده می کنند نشانگر این است که زمان لازم برای تولید نرم افزار برای برنامه های کاربردی کوچک و متوسط تقریباً کاهش یافته و میزان طراحی و تحلیل برای برنامه های کوچک نیز کم شده است.

۳-استفاده از4GT برای کارهای تولید نرم افزار در مقیاس بزرگ نیازمند تحلیل، طراحی و آزمون بیشتر است ولی از طریق حذف کدنویسی می توان به هدف خود دست یافت.

فصل سوم

مفاهیم مدیریت پروژه

نگاه اجمالی:

- چه کسی آن را انجام می دهد؟
- چرا مدیریت مهم است؟
- مراحل مدیریت کدامند؟
- محصول چیست؟
- چگونه می توانم مطمئن شوم که کار را درست انجام داده ام؟

بازیگران

- **مهندس نرم افزار:**

فعالیت های روزمره طرح ریزی و کنترل کار های فنی خود را اداره می کند.

- **مدیران ارشد:**

ارتباط بین امور و متخصین نرم افزار را فراهم می کند.

- **مدیران پرو ژه:**

گروه یا تیم مهندسین نرم افزار را کنترل و طرح ریزی می کند

بازیگران

- **مشتریان**

نیازمندیهای نرم افزار و سایر واگذارندگان را مشخص می کنند.

- **کاربران نهایی**

در زمان روانه شدن نرم افزار به بازار، با آن به تعامل می پردازند.

علل اهمیت مدیریت

ساختن نرم افزار، کاری پیچیده است بخصوص اگرافراد بسیاری درآن دخیل باشند و برای مدت طولانی برروی آن کار کنند.

مراحل مدیریت

۱. **افراد:** باید برای انجام درست کارها سازمان دهی شوند.

مدل رشد مدیریت افراد عملکرد های کلیدی زیرا برای پرسنل نرم افزار تشریح می کند:

استخدام، مدیریت عملکرد، جبران آموزش و پیشرفت شغلی و طراحی سازمان و کار تو سعه

مراحل مدیریت

۲. محصول: ارتباط با مشتری باید به گونه ای باشد که حیطه محصول و نیازمندی ها درک شود.

۳. فرایند: به نحوی باید انتخاب شود که با نرم افزار و محصول متناسب با شد

٤. پروژه: باید با برآورده کردن نیرو و زمان مورد نیاز برای انجام کار، تشریح محصولات، انجام کنترل کیفیت و تعیین مکانیزم های نظارت در طرح، طراحی شود.

محصول چیست؟

طرح پروژه با شروع فعالیت های مدیریت، تهیه و تولید می شود. این طرح، رونـد و کارهایی که باید انجام شود افرادی که کارها را انجام می دهند و مکانیزم ارزیابی خطرات، کنترل تغییر و ارزیابی کیفیت را تشریح می کند.

نکته: هرگز به طور قطعی نمی توانید اطمینان حاصل کنید که طرح پروژه درست است مگر آنکه محصول با کیفیت را در زمان مقرر و با بودجه تعیین شده ارائه کنید.

رهبران تیم

مدل MOI (Motivation – Organization - Idea):

۱. انگیزه

۲. سازماندهی

۳. ایده ها یا ابداع و نوآوری

چهار ویژگی مدیرپروژه کارآمد:

۱.حل مسئله

۲.هویت مدیریتی

۳.موفقیت

۴.ساخت تیم و تاثیرگذاری

۱.**حل مسئله:** مدیر پروژه نرم افزار کارآمد می تواند مسائل فنی و سازمانی شایع را تشخیص دهد و راه حل اصولی را سازماندهی و مطالب اندوخته شده ازپروژه های قبلی را در موقعیت های جدید اعمال کند و در صورت بی ثمر بودن راه حل های اولیه برای تغییر جهت انعطاف داشته باشد.

۲. **هویت مدیریتی:** مدیر پروژه خوب باید مسئولیت پروژه را بر عهده بگیرد او باید برای بدست گرفتن کنترل در زمان مقتضی و مجاز کردن پرسنل فنی خوب از اتکا به نفس خوبی برخوردار باشد.

۱. **موفقیت**: برای بهینه کردن بهروری تیم پروژه، مدیر باید موفقیت و نوآوری را تشویق کند و با اعما ل خود نشان دهد که اگر افرادی به طور کنترل شده ریسک کنند با مجازات روبرو نمی شوند.

۲. **ساخت تیم و تاثیر گذاری:** یک مدیرموفق و موثر باید مردم شناس باشد و تحت شرایط فشار بتواند اوضاع را تحت کنترل در آورد.

مد ل MIO (برای رهبری)

- **انگیزه:** قابلیت ترغیب پرسنل فنی در ایجاد بهترین قابلیت ها
- **سازماندهی:** توانایی ایجاد و برقراری فرایندهای موجود که امکان تبدیل مفهوم اولیه به محصول نهایی را فراهم می کند.
- **ایده ها یا ابداع:** توانایی ترغیب افراد به خلاقیت به هنگام کار در محدوده تعیین شده برای محصول

تیم نرم افزار

تعداد ساختارهای سازمانی انسانی برای توسعه نرم افزار به اندازه تعداد سازمانهای توسعه دهنده نرم افزاری می باشد. ساختار سازمانی نمی تواند به راحتی تغییر داده شود. نتایج عملی و سیاسی تغییر سازمانی در حیطه مسئولیت های مدیر پروژه نرم افزاری نمی باشد اما تشکیلات مردمی که مستقیما در پروژه نرم افزاری دخیل هستند درحیطه کاری مدیر پروژه قراردارد.

سه ساختار کلی تیمی مانتی:

ساختار غیر متمرکز دموکراتیک(DD): Democratic decentralized

دارای یک رهبر دائمی و ثابت نمی باشد، هماهنگ کننده های کار، برای دوره کوتاهی در این سمت منصوب می شوند و سپس افراد دیگری که ممکن است کارهای مختلفی را هماهنگ کنند جایگزین این هماهنگ کننده ها می شوند.

ساختار غیر متمرکز کنترل شده(CD): Controlled decentralized

دارای یک رهبر مشخص می باشد که کارهای خاص را هماهنگ و مشخص می کند و مدیران ثانوی که مسئولیت کارهای فرعی را بر عهده دارند.

سازختار متمرکز کنترل شده(CC): Controlled centralized

حل مشکلات در سطوح بالا و ایجاد هماهنگی داخلی در میان تیم تحت نظارت و کنترل رهبر تیم انجام می شود. ارتباط موجود میان رهبر تیم و اعضا تیم به صورت عمودی می باشد.

مانتی هفت فاکتور پروژه که باید در هنگام برنامه ریزی ساختار تیمهای مهندسی نرم افزاری در نظر گرفته شود را توصیف کرد:

۱- دشواری مساله ای که باید مورد حل و فصل قرار گیرد

۲- اندازه برنامه حاصل برحسب تعداد خطوط کد یا تعداد نقاط تابع

۳- مدت اشتغال اعضای تیم

۴- میزان قابلیت پیما نه ای مساله

۵- میزان اعتبار و کیفیت مورد نیاز برای سیستمی که باید ایجاد شود

۶- قطعیت تاریخ تحویل

۷- میزان ارتباطات مورد نیاز برای پروژه

چهار الگوی سازمانی تعریف شده توسط کنستانتین:

۱. الگوی بسته

۲. الگوی تصادفی

۳. الگوی باز

۴. الگوی همگام

الگوی بسته

- در این الگو، تیم در راستای یک سلسله مراتب سنتی از مسولیتها سازماندهی می شود.
- برای کارهای روتین مناسب است.
- برای نوآوری و کارهای جدید، احتمال موفقیت کم است.

الگوی تصادفی

- تیم ساختار سستی دارد و به ظرفیت تک تک افراد تیم بستگی دارد.
- برای کارهایی که نیاز به نوآوری دارند بهترین نتیجه را می دهد.
- در صورتی که کارکردی منظم مورد نیاز باشد چنین تیمی به دردسر می افتد.

الگوی باز

•سعی می شود در عین دستیابی به کنترلهای خاص الگوی بسته، به حداکثر نوآوری هایی که در الگوی تصادفی وجود دارد دستیابی شود.

•کارها از طریق ارتباطات سنگین و تصمیم گیری های مبتنی بر اجماع انجام می شود.

•برای مسائل پیچیده مناسب است ولی ممکن است بازدهی تیم های دیگر را نداشته باشد.

الگوی همگام

اعضای تیم به شیوه ای قطعه قطعه و سازماندهی می شوند و روی قطعات مساله کار می کنند طوری که ارتباط فعال میان آنها بسیار کم باشد.

جکمن پنج عامل را موجب مسمومیت تیم نرم افزاری معرفی می کند:

۱.جو کاری آشفته

۲.ناامیدی شدید

۳.شیوه های تفکیک شده یا ناقص

۴.تعریف نامشخص نقش ها

۵.تعریف قرارگیری در معرض شکست

مسائل هماهنگی و ارتباط

- قابلیت استفاده داخلی به صورت یکی از مشخصه های اصلی بسیاری از سیستم ها درآمده است. نرم افزار جدید باید با نرم افزار موجود ارتباط برقرار کند و با محدودیت های از پیش تعیین شده ای که از سویی به سیستم یا محصول تحمیل می شود، مطابقت داشته باشد.

- خصیصه های نرم افزاری مدرن، یعنی مقیاس، عدم اطمینان و قابلیت استفاده داخلی حقایق زندگی هستند. برای کنترل موثر این خصیصه ها، تیم مهندسی نرم افزار باید شیوه های مناسب و موثری را برای هماهنگ کردن افرادی که کار را با هم انجام می دهند ایجاد کند. برای تکمیل هماهنگی، مکانیسم های لازم برای روابط رسمی و غیر رسمی میان اعضا تیم و روابط میان چند تیم مختلف باید مشخص شوند. روابط رسمی از طریق نوشتن نامه، ملاقات رسمی و تقریبا از طریق سایر کانالهای ارتباطی غیر شخصی به وجود می آید.

فنون هماهنگ سازی و ارتباطات

- رهیافت رسمی و غیررسمی
- شیوه های رسمی و میان فردی
- شیوه های غیررسمی و میان فردی
- روابط الکترونیکی
- شبکه میان فردی

- **رهیافت های رسمی و غیر رسمی شامل موارد زیر می باشد:**

- اسناد و مدارک قابل تحویل مهندسی نرم افزار
- نوشته های فنی
- زمان ارزیابی پروژه
- برنامه ها و ابزار کنترل پروژه
- بوجود آمدن تغییرات مجدد و مدارک مربوطه
- گزارشهای مربوط به پیگیری خطاها و اطلاعات سری.

- **شیوه های رسمی ومیان فردی**

بر فعالیت های تضمین کیفیت که در مورد محصول کاری مهندسی نرم افزار به کار برده شده است تأ کید دارد این شیوه ها شامل جلسات تجدید نظر رسمی، بررسی مجدد طرح ها و کدها می باشد.

•شیوه های غیر رسمی و میان فردی

این شیوه ها معمولا شامل جلسات گروهی می باشد که به منظور پخش اطلاعات و حل مشکلات و جمع آوری اطلاعات در مورد نیازهای کارمندان و توسعه انجام شده برگزار می شود.

- **روابط الکترونیکی**

این روابط شامل پست الکترونیک، بردهای الکترونیکی پژوهشی و در حالت گسترده و پیشرفته شامل سیستم های گفتگوی ویدئویی می باشد.

- **شبکه میان فردی**

مذاکرات رسمی با اعضا تیم و مذاکرات رسمی با افرادی که عضو پروژه نمی باشند اما تجارب یا اطلاعاتی دارند که می تواند به اعضا تیم کمک کند.

تلفیق محصول و فر آیند

برنامه ریزی پروژه با تلفیق محصول و فرآیند آغاز می شود.

- ارتباط با مشتری
- طرح ریزی
- تجزیه و تحلیل خطرهای احتمالی
- مهندسی
- ساخت و واگذاری
- ارزیابی مشتری

- **ارتباط با مشتری:** کارهایی که برای کشف موثر نیازمندیها میان سازنده و مشتری لازم می باشند.

- **طرح ریزی:** کارهایی که برای تعریف منابع، خطوط زمانی وسایر اطلاعات مربوط به پروژه نیاز می باشد.

- **تجزیه و تحلیل خطرهای احتمالی:** کار هایی که برای ارزیابی خطرهای احتمالی فنی و مدیریتی ضروری می باشند.

- **مهندسی:** کارهایی که برای ساخت یک یا چند مورد از کاربرد های طرح لازم می باشند.

- **ساخت و واگذاری:** کارهایی که برای ساخت، آزمایش، نصب و ارائه خدمات پشتیبانی به کاربر ضروری می باشند.

- **ارزیابی مشتری:** کارهایی برای دستیابی به بازخورد مشتری بر اساس ارزیابی نرم افزار که در طول مرحله طراحی ایجاد و در مرحله نصب، پیاده سازی شده اند.

فعالیتهای چارچوب مشترک فرآیند	ارتباط با مشتری	طرح ریزی	تحلیل ریسک	مهندسی
فعالیتهای مهندسی نرم افزار				
کارکردهای محصول				
ورودی متن				
ویرایش و قالب بندی				
اصلاح خودکار				
آرایش صفحه و صفحه بندی				
شاخص بندی خودکار وToc				
مدیریت فایل				
تولید مستندات				

مرتبط ساختن مسئله و فرایند

تجزیه فرآیند

پس از انتخاب مدل فرآیند، چارچوب فرایند مشترک نیز برآن مطابقت داده می شود. در هرمورد CPF،

- **ارتباط با مشتری**
- **برنامه ریزی**
- **تجزیه و تحلیل احتمال خطر**
- **طراحی**
- **ساخت**
- **انتشار**
- **ارزیابی مشتری**

را می توان با الگو انطباق داد.

پروژه

به منظور دستیابی به یک پروژه نرم افزاری موفق، ابتدا باید به این مطلب پی برد که چه مشکلی وجود دارد و چگونه می توان آن مشکل را برطرف نمود.

علائمی که از نظر جان ریل، مشخص می کند که پروژه سیستم های اطلاعاتی در معرض خطا قرار گرفته اند عبارتند از:

۱. متخصصان نرم افزاری، نیازهای مشتری خود را درک نمی کنند.

۲.حوزه عملکرد محصول خوب تعیین نشده است.

۳.تغییرات به خوبی کنترل نمی شوند.

۴.فناوری انتخاب شده تغییر می یابد.

۵.نیاز های تجاری تغییر می کنند.

۶.مهلت ها واقع بینانه نیستند.

۷.کاربران مقاومت می کنند.

۸. حمایت مالی هرگز به صورت مناسب و درست حاصل نشده است.

۹.افراد متخصص با مهارتهای مناسب در تیم پروژه وجود ندارد.

۱۰.مدیران وسازندگان از به کار بردن بهترین تمرین ها و درسهای فرا گرفته شده اجتناب می کنند.

ریل رهیافت پنج قسمتی داوری صحیح را در مورد پروژه های نرم افزاری پیشنهاد می دهد:

۱. کار را به روشی صحیح آغاز کنید.

۲.نیروی سوق دهنده پروژه را حفظ کنید.

۳.پیشرفت کار را دنبال کنید.

۴.در پروژه خود تصمیم گیری صحیح و عاقلانه انجام دهید.

۵.یک تجزیه وتحلیل نیز پس از اتمام پروژه انجام دهید.

اصل W^5HH باری بوهم

”شما به یک اصل سازمان یافته نیاز دارید که طرح های ساده برای پروژه های ساده با مشخص کند.“ بوهم روشی پیشنهاد می کند که

- اهداف پروژه
- زمانبندی ها و نقاط عطف پروژه
- مسئولیت ها
- روش مدیریتی و فنی و منابع مورد نیاز برای پروژه

را دربرمی گیرد. او این اصل را اصل WWWWWHH نامید.

این مجموعه سوالات منجر به تعریف ویژگی های کلیدی پروژه می شود:

- چرا سیستم توسعه می یابد؟
- چه کاری و چه زمانی انجام خواهد شد؟
- چه کسی مسئول یک عمل است؟
- این مسئولیت ها به صورت سازمانی چه جایگاهی دارند؟
- کار از نظرمدیریتی و فنی چگونه انجام خواهد شد؟
- چه میزان از هر یک از منابع مورد نیاز می باشد؟
- اصل W^5HH بوهم، برای هر پروژه نرم افزاری صرف نظر از پیچیدگی پروژه قابل استفاده است.

اقدامات بحرانی

شورای Air Lie لیستی از کارهای حساس نرم افزاری را برای مدیریت مبتنی بر عملکرد تهیه کرده است. برای اینکه سازمانی قادر باشد تعیین کند که آیا پروژه ای اعمال بحرانی را پیاده سازی کرده است یک مجموعه پرسش تهیه کرده است.

اقدامات بحرانی

- مدیریت رسمی ریسک و خطر
- برآورد هزینه و زمانبندی
- مدیریت پروژه مبتنی بر معیار ها
- ردگیری مقادیر بدست آمده
- پیگیری نقایص در مقابل اهداف کیفیتی
- مدیریت برنامه های آگاهی از افراد

فصل چهارم

معیارهای های پروژه و فرایند نرم افزار

معیار و اندازه گیری چیست؟

معیار های فرایند و محصول نرم افزاری اندازه گیری های کمیتی هستند که به افراد نرم افزاری امکان می دهد تا نسبت به کارآمد بودن فرایند و پروژهای نرم افزاری آگاهی پیدا کنند.

- چه کسی این کار را انجام می دهد ؟
- چرا این کار اهمیت دارد ؟
- این مراحل و اقدامات کدامند ؟
- حاصل کار چیست ؟
- چگونه مطمئن شوم که آن کار را درست انجام داده ام ؟

چه می کند؟

معیارهای نرم افزاری توسط مدیران نرم افزار تحلیل و ارزیابی می شوند میزان ها غالبا توسط مهندسان نرم افزار جمع آوری می شوند.

چرا این کار اهمیت دارد ؟

اگر اندازه گیری نکنیم، داوری ما فقط مبتنی بر ارزیابی موضوعی خواهد بود با اندازه گیری روند امور (چه خوب چه بد) را می توان مشخص کرد برآوردها را بهتر می توان انجام داد، و با گذشت زمان می توان به بهبود واقعی دست یافت.

این مراحل و اقدامات کدامند ؟

با تعریف یک مجموعه محدود از میزان های فرآیند، پروژه و محصول شروع می کنیم که جمع آوری آنها آسان است این میزان ها غالبا معیارهای اندازه گرا یا عمل گرا هستند نتیجه تحلیل می شود و با میانگین های گذشته برای پروژه های مشابه انجام شده در همان سازمان، مقایسه می شوند، روندها ارزیابی می شود و نتایج استنباط می شود.

حاصل کار چیست ؟

مجموعه ای از معیارهای نرم افزاری که منجر به درک فرآیند و پروژه می شود

چگونه مطمئن شوم که آن کار را درست انجام داده ام ؟

با اعمال یک طرح اندازه گیری سازگار و در عین حال ساده

چهار دلیل برای اندازه گیری منابع، محصولات و فرایندهای نرم افزاری عبارتند از:

- مشخص کردن ویژگی ها
- ارزیابی کردن
- پیش بینی کردن
- پیشرفت کردن

مشخص کردن ویژگی ها

ویژگی های فرآیندها، محصولات، منابع و محیط ها را مشخص می کنیم تا آنها را درک کنیم و مبناهایی برای مقایسه ارزیابی های آینده بنا کنیم.

ارزیابی کردن

ارزیابی می کنیم تا وضعیت طرح ها را معین کنیم. موازین، حسگرهایی هستند که به ما اطلاع می دهند فرآیندها و پروژه های ما چه زمانی از مسیر اصلی خود منحرف می شوند و می توانیم آنها را دوباره تحت کنترل در آوریم. همچنین عمل ارزیابی را انجام می دهیم تا بتوانیم میزان دستیابی به اهداف کیفیتی و تاثیرات بهبود فناوری و فرآیند بر محصول را بسنجیم.

پیش بینی کردن

- بتوانیم طرح ریزی کنیم
- می خواهیم اهداف قابل دستیابی برای هزینه، زمانبندی و کیفیت بنا کنیم به طوری که منابع مناسبی را بتوانیم به کار ببریم.

اندازه گیری برای پیش بینی، شامل درک روابط میان فرآیندها و محصولات و ساخت مدل هایی از این روابط می شود به طوری که مقادیر مشاهده شده برای برخی صفات را بتوان برای پیش بینی صفات دیگر به کار برد.

اندازه ها، معیار ها

اندازه: یک شاخص کمیتی از مقدار، میزان، ابعاد، ظرفیت یا یکی از ویژگی های یک محصول یا فرایند را فراهم می آورد.

اندازه گیری: همان عمل تعیین یک اندازه است.

معیار: میزان کمی از حدی که یک سیستم، مولفه یا فرآیند می تواند دارای یک صفت مفروض باشد.

میزان

جمع آوری یک نقطه داده ای منفرد مانند:

تعداد خطاهای کشف شده در بازبینی یک پیمانه منفرد

معیار

معیار نرم افزاری به نحوی با تک تک میزان ها در ارتباط است مانند:

- تعداد میانگین خطاهای یافت شده به ازای هر بازبینی
- تعداد میانگین خطاهای یافت شده به ازای هر نفر ساعت صرف شده روی بازبینی ها

شاخص

مهندس نرم افزار معیارهای را جمع آوری و ایجاد می کند، به طوری که شاخص هایی از آنها به دست می آید.

شاخص، معیار یا ترکیبی از معیارهاست که درکی از فرآیند نرم افزار، پروژه نرم افزاری یا خود محصول به دست می دهد.

شاخص، دیدی فراهم می آورد که مدیر پروژه یا مهندسان نرم افزار را قادر به تنظیم فرآیند می سازد تا شرایط پروژه یا فرآیند بهتر شود.

شاخص های پروژه مدیر پروژه را قادر می سازند تا:

۱) به وضعیت پروژه در حال پیشرفت دست پیدا کند.

۲) خطرات بالقوه را دنبال کند.

۳) نواحی مشکل آفرین را پیش از بحرانی شدن آنها کشف کند.

۴) جریان کار یا وظایف را تنظیم کند.

۵) توانایی تیم پروژه در کنترل کیفیت محصولات کاری نرم افزار را ارزیابی کند.

موارد تاثیرگذار بر کیفیت فرایند و کارایی سازمانی

موارد تاثیرگذار بر کیفیت فرایند و کارایی سازمانی

۱. انگیزش و مهارت افراد

۲. پیچیدگی محصول می تواند اثری چشمگیر بر کیفیت و کارآیی تیم بگذارد

۳. فناوری (یعنی روش های مهندسی نرم افزار) که فرآیند را تشکیل می دهد

مثلث فرآیند در داخل دایره ای از شرایط محیطی قرار گرفته است که شامل:

۱. محیط توسعه (یعنی ابزار کیس)

۲. شرایط بازار کار (مثل مهلت ها و قواعد تجاری)

۳. خصوصیات مشتری (مثل سهولت برقراری ارتباط) می شود.

موازین غیر مستقیم

- موازین خطاهای کشف شده، پیش از ارائه نرم افزار
- نقایص گزارش شده توسط کاربران نهایی
- محصولات کاری تحویل شده
- انرژی انسانی صرف شده
- زمان صرف شده
- مطابقت با برنامه زمانبندی و موازین دیگر

انواع معیارهای فرایند

- معیارهای خصوصی
- معیارهای عمومی

معیارهای خصوصی عبارتند از :

- تعداد نقایص (توسط فرد)
- تعداد نقایص (توسط پیمانه)
- خطاهای یافته شده در اثنای توسعه

معیارهای عمومی عبارتند از :

- نقایص گزارش شده برای اعمال اصلی نرم افزار (که توسط چند تن از سازندگان، توسعه یافته اند)
- خطاهای یافته شده در اثنای بازبینی های فنی رسمی
- تعداد خطوط کد یا نقاط تابع به ازای هر پیمانه یا هر عمل

تحلیل شکست به چه روشهایی صورت می گیرد؟

۱. همه خطاها و معایب بسته به منشاء شان طبقه بندی می شوند.

۲. هزینه تصحیح هر خطا و نقص، ثبت می شود.

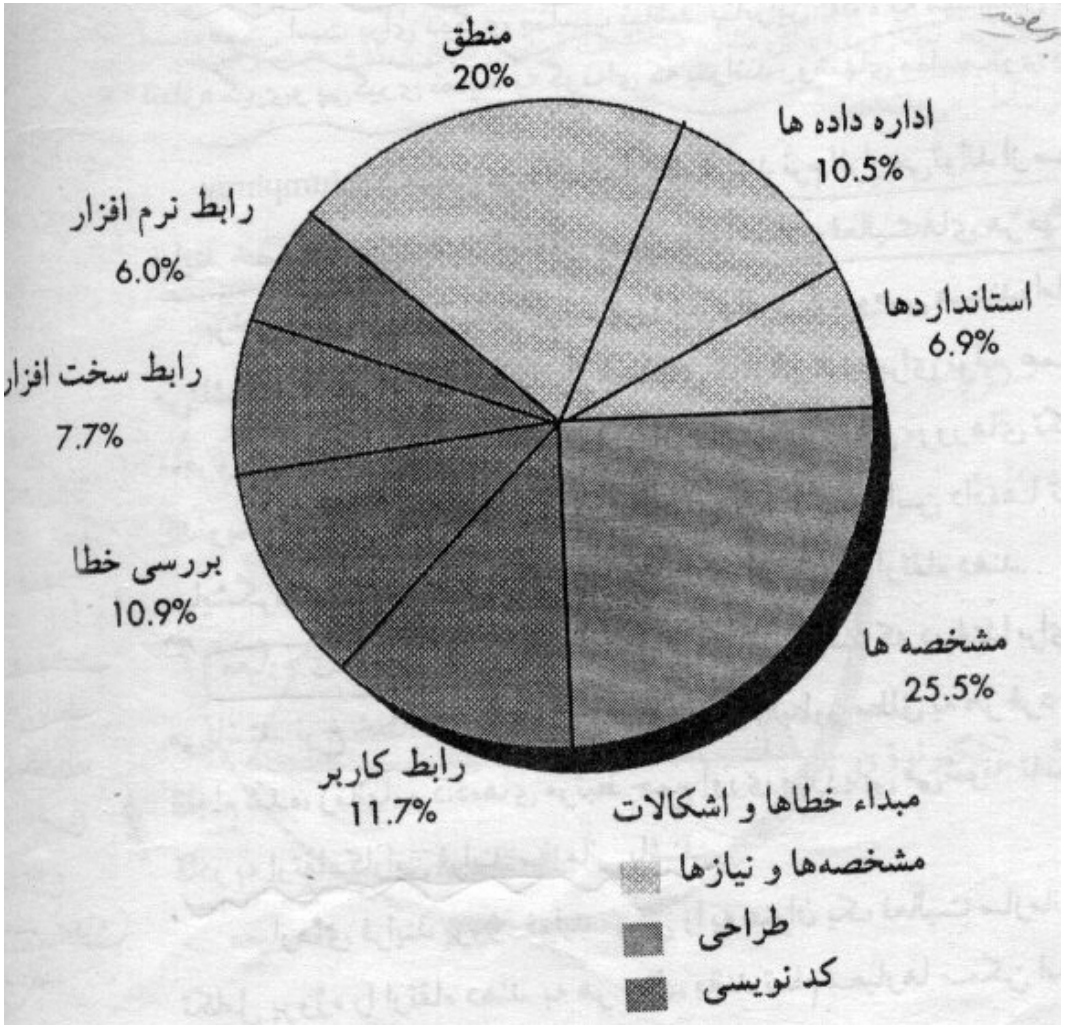
۳. تعداد خطا ها و نقایص در هر دسته شمارش و به ترتیب از بالا به پایین طبقه بندی می شوند.

۴. هزینه کلی خطاها و نقایص در هر طبقه محاسبه می شود.

۴- داده های حاصل، مورد تحلیل قرار می گیرند تا گروه هایی که منجر به بالاترین هزینه ها در سازمان می شوند کشف شوند.

۵- تدابیری برای اصلاح فرآیند با هدف حذف یا کاهش دسته ای از خطاها و نقایص که بیشترین هزینه را دارند اندیشیده می شود.

علل نقایص و منشا آنها برای چهار پروژه نرم افزاری

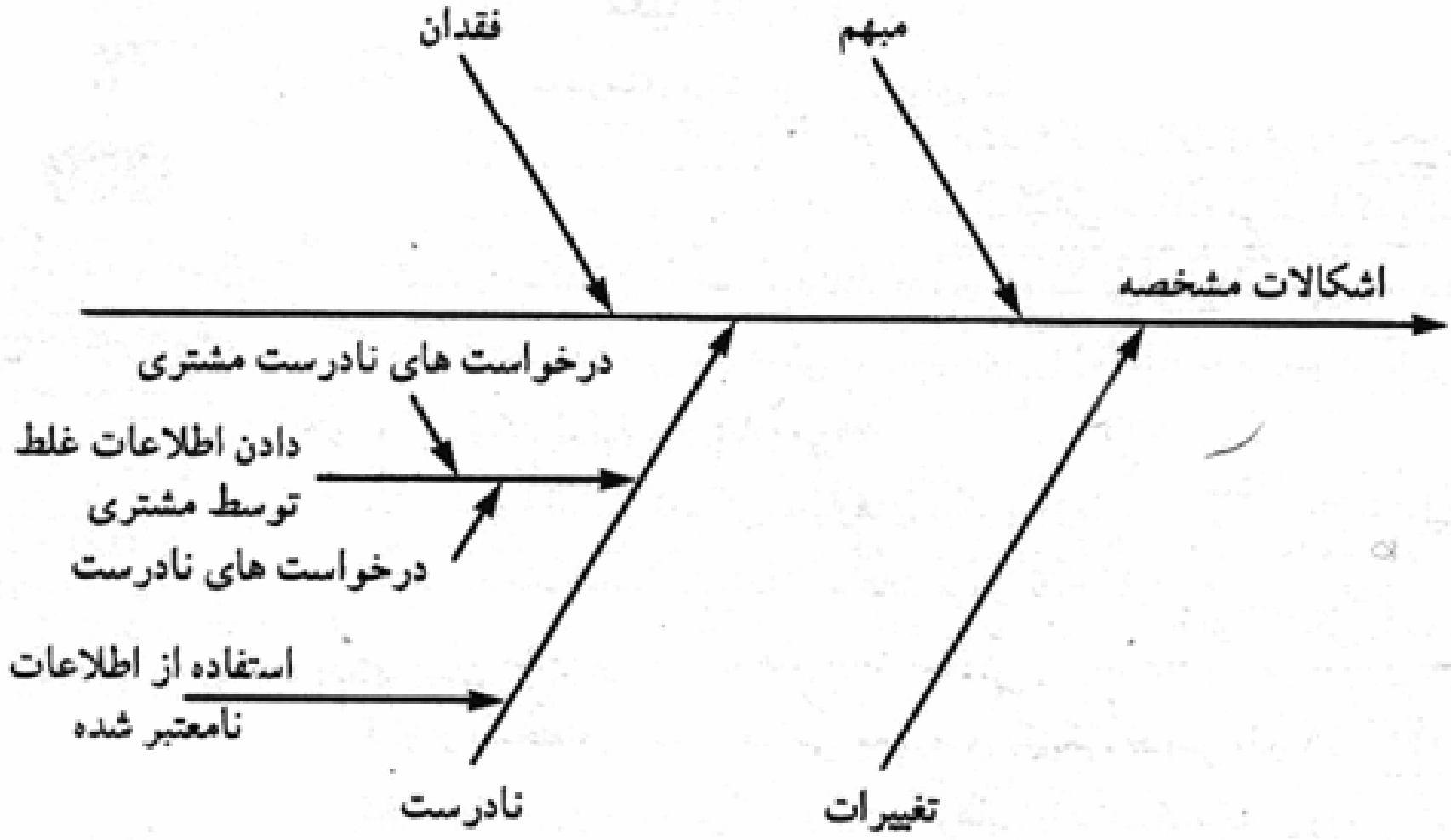


نمودار استخوان ماهی پیشنهاد شده توسط گریدی برای کمک به بررسی داده های ارائه شده در نمودار فراوانی:

•ستون فقرات نمودار (خط مرکزی) نشانگر عامل کیفیت مورد نظر است.

•هر یک از دنده ها (خطوط مورب) نشانگر علل بالقوه برای مشکل کیفیتی (از قبیل کمبود خواسته ها، مشخصه مبهم، خواسته های نادرست، تغییر خواسته ها)

•سپس ستون فقرات و دنده ها به هر یک از دنده های اصلی نمودار افزوده می شوند تا علت آن گسترش داده شود.



نمودار استخوان ماهی

یکی از مدل های معیارهای پروژه می گوید که هر پروژه باید موارد زیر را اندازه گیری کند:

- ورودی ها: میزان هایی از منابع (مثل افراد، محیط) که برای انجام کار لازم است.
- خروجی ها: میزان هایی از قطعات قابل تحویل یا محصولات کاری ایجاد شده طی فرآیند مهندسی نرم افزار.
- نتایج: میزان هایی که نشان دهنده میزان کارآمدی قطعات تحویل شده اند.

اندازه گیری نرم افزار

اندازه گیری در دنیای مادی (فیزیکی)

- موازین مستقیم
- موازین غیر مستقیم

موازین مستقیم

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

معیارهای اندازه گرا

موازین مستقیم فرآیند مهندسی نرم افزار

- خطاها در هر هزار خط برنامه (KLOC)
- تعداد نقایص در هر هزار خط برنامه (KLOC)
- هزینه هر خط برنامه (LOC) (بر حسب دلار یا واحد پول)
- تعداد صفحات مستندات در هر هزار خط برنامه (KLOC)

علاوه بر موارد فوق:

- تعداد خطاها به ازای هر نفر – ماه
- تعداد خطوط برنامه (LOC) به ازای هر نفر – ماه
- هزینه هر صفحه از مستندات (بر حسب دلار یا واحد پول)

موازین غیر مستقیم فرآیند مهندسی نرم افزار

موازین غیر مستقیم از میزان عملکرد ارائه شده توسط برنامه کاربردی، به عنوان مقداری برای نرمال سازی استفاده می کنند چون عملکرد را مستقیما نمی توان اندازه گیری کرد باید آنرا به طور غیر مستقیم با استفاده از موازین مستقیم دیگر به دست آورد.

معیارهای عملکرد گرا نخستین بار توسط آلبرشت پیشهاد شدند او میزانی موسوم به نقطه عملکرد را پیشنهاد کرد.

نقاط عملکرد با استفاده از یک رابطه تجربی مبتنی بر موازین قابل شمارش (مستقیم) از دامنه اطلاعات نرم افزاری و ارزیابی پیچیدگی نرم افزار به دست می آیند.

نقاط عملکرد با کامل کردن جدول ۴-۵ محاسبه می شوند پنج ویژگی از دامنه اطلاعاتی تعیین می شوند و شمارش هایی در محل مناسب جدول صورت می پذیرد مقادیر دامنه اطلاعات به شیوه زیر تعیین می شوند.

- **تعداد ورودی های کاربر.** هر ورودی کاربر که داده های کاربردی متمایز نسبت به نرم افزار داشته باشد.
- **تعداد خروجی های کاربر.** هر خروجی کاربر که اطلاعاتی با جهت گیری کاربردی برای کاربر فراهم آورد شمارش می شود در این معنا، خروجی به گزارش ها، صفحات نمایشی، پیام های خطا و غیره اطلاق می شود.
- **تعداد درخواست های کاربر.** درخواست به عنوان یک ورودی on-line تعریف می شود که منجر به تولید پاسخ بی درنگ به شکل خروجی on-line می شود.

- **تعداد فایلها**. هر فایل اصلی منطقی، یعنی گروهی منطقی از داده ها که ممکن است بخشی از یک بانک اطلاعاتی بزرگ یا یک فایل مجزا باشد شمارش می شود.
- **تعداد واسط های خارجی.** همه واسطهای خواندنی ماشین (مانند فایل های داده ای روی نوار باریک) که برای انتقال دادن اطلاعات به سیستم دیگری به کار می روند.

امتیازات عملکردی با استفاده از یک رابطه تجربی بر اساس اندازه های قابل شمارش (مستقیم) دامنه اطلاعات نرم افزاری و ارزیابی پیچیدگی نرم افزاری بدست می آیند.

ضرایب وزنی

پارامتر های اندازه گیری	تعداد		ساده	متوسط	پیچیده	
تعداد ورودی های کاربر		*	۳	۴	۶	=
تعداد خروجی های کاربر		*	۴	۵	۷	=
تعداد درخواستها		*	۳	۴	۶	=
تعداد فایلها		*	۷	۱۰	۱۵	=
تعداد واسطهای خارجی		*	۵	۷	۱۰	=
تعداد کل						

تعیین پیچیدگی

وقتی داده ها جمع آوری شدند یک مقدار پیچیدگی با هر شمارش همراه می شود که تعیین آن تا حدی ذهنی است.

برای محاسبه امتیازات عملکردی (FP)، رابطه ذیل به کار می رود:

$$FP = (\text{total count}) * [\, 0.65 + 0.01 * \sum (Fi) \,]$$

$i = ۱, .. ۱۴$, مجموع همه مدخلهای FP

سوالهایی برای تعیین مقادیر تنظیم پیچیدگی (Fi)

۱. آیا سیستم به پشتیبانی و بازیابی قابل اطمینان نیاز دارد؟
۲. آیا ارتباطات داده ای مورد نیاز است؟
۳. آیا عملیات پردازشی توزیع شده وجود دارند؟
۴. آیا کارآیی اهمیت دارد؟
۵. آیا سیستم در یک محیط عملیاتی موجود و پرکاربرد اجرا می شود؟
۶. آیا سیستم به مدخل داده های on-line نیاز دارد؟
۷. آیا مدخل داده های on-line نیاز به ساخت تراکنش ورودی روی عملیات یا صفحات نمایش چندگانه دارد؟
۸. آیا فایل های اصلی به صورت on-line به هنگام می شوند؟
۹. آیا ورودی ها خروجی ها، فایل ها و در خواست های پیچیده اند؟
۱۰. آیا پردازش داخلی پیچیده است؟
۱۱. آیا کد طراحی شده است که دوباره قابل استفاده باشد؟
۱۲. تبدیل و نصب در طراحی لحاظ شده است؟
۱۳. آیا سیستم برای نصب چندگانه در سازمان های متفاوت طراحی شده است؟
۱۴. آیا برنامه کاربردی طوری طراحی شده است که تغییرات را تسهیل کند و به آسانی توسط کاربر استفاده شود؟

ایجاد ارتباط بین دو میزانFP و LOC:

آلبرشت و گافنی می گویند:

غرض از این کار پژوهشی آن است که مقدار عملکرد ارائه شده توسط برنامه کاربردی را بتوان از روی تک تک قطعات اصلی داده های مورد استفاده آن یا تولید شده توسط آن برآورد نمود. به علاوه این برآورد، عملکرد باید هم با LOC تولید شده و هم انرژی صرف شده همبستگی داشته باشد.

ایجاد ارتباط میان معیارهای متفاوت

زبان برنامه سازی	میانگین تعداد خطوط به ازای هرامتیاز کارکردی
Assembly Language	320
C	128
COBOL	106
FORTRAN	106
Pascal	90
C++	64
Ada 95	53
Visual Basic	32
Smaltalk	22
Powerbuilder (code	16

برآورد خاصی از میانگین تعداد خطوط کد مورد نیاز برای ساخت یک نقطه عملکرد

تعداد خطوط C++ تقریبا ۱,۶ برابر خطوط FORTRAN، امتیاز و قابلیت عملکردی را ایجاد می کند.

مجموعه ای از عوامل کیفیتی تعریف شده توسط مک کال و کارانو برای توسعه معیارهایی برای کیفیت نرم افزار:

۱. کارکرد محصول (استفاده از آن)

۲. بازبینی محصول (تغییر دادن آن)

۳. گذار محصول (اصلاح آن برای کارکردن در محیطی متفاوت)

تعاریف ارائه شده برای موازین کیفیت نرم افزار توسط گلیب:

- **درستی**

یک برنامه باید درست کار کند و گرنه برای کاربران خود ارزشی ندارد

درستی، حدی از کارکرد نرم افزار برای انجام وظایف آن است،

متداولترین میزان برای درستی، تعداد نقایص به ازای هر KLOC است.

- **قابلیت نگهداری**

قابلیت نگهداری عبارت از سهولت تصحیح یک برنامه در صورت مواجهه با خطا، تطابق با تغییرات محیط یا بهبود بخشیدن به برنامه در صورت تغییر یافتن خواسته های مشتری است.

نگهداری نرم افزار از هر فعالیت مهندسی نرم افزار دیگر بیشتر کار می برد.

- **اندازه گیری قابلیت نگهداری**
- میانگین زمان لازم برای تغییر (MTTC)، یعنی زمان لازم برای تحلیل تقاضای تغییر
- طراحی یک اصلاح مناسب
- پیاده سازی تغییر
- آزمون آن و توزیع تغییر در میان همه کاربران.

به طور میانگین، برنامه هایی که قابل نگهداری هستند از برنامه هایی که قابل نگهداری نیستند MTTC کمتری دارند.

ریخت و پاش

هیتاپی یک معیار مبتنی بر هزینه را برای قابلیت نگهداری به کار برده است که ریخت و پاش نام دارد و عبارت از هزینه ای است که برای رفع نواقصی صرف می شود که پس از ارائه سیستم به کاربران شناسایی شدند.

هنگامی که نسبت ریخت و پاش به هزینه کل پروژه (برای چند پروژه) به صورت تابعی از زمان رسم شود، مدیر می تواند تعیین کند که آیا قابلیت نگهداری کلی نرم افزار تولید شده، توسط سازمان بهبود یافته است یا خیر.

- **یکپارچگی**

توانایی یک نرم افزار را برای مقاومت در برابر دستبردها به برنامه ها، داده ها و اسناد، می سنجد. که دو ویژگی باید بررسی شود:

۱- تهدید

بروز یک حمله از نوعی خاص و در محدوده زمانی مشخص

۲-امنیت

احتمال دفع نوعی خاص از حمله

یکپارچگی = Σ[(تهدید -1) * (امنیت -۱)]

میزان تهدید و امنیت را از روی شواهد تجربی می توان تخمین زد.

کارایی رفع نقص DRE

میزانی از توانایی فیلتر کردن فعالیت های مربوط به کنترل و تضمین کیفیت است که در سرتاسر فعالیت های چارچوبی اجرا می شوند.

به طور ایده آل مساوی است با **DRE = E / (E + D)**

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

D خطاهایی است که بعد از تحویل یافته شده اند و در حالت عادی بزرگتر از صفر است.

اگر DRE به عنوان شاخصی از توانایی فیلتر کردن فعالیت های کنترل و تضمین کیفیت در نظر گرفته شود تیم نرم افزاری را به وضع تکنیک هایی تشویق می کند که خطاها را پیش از تحویل سیستم به کاربران نهایی بیابند.

آن دسته از خطاها که طی بازبینی مدل تحلیل یافت نمی شوند به وظیفه طراحی راه پیدا می کنند (که در آنجا ممکن است یافته شوند یا نشوند) در این حالت DRE به صورت زیر تعریف می شود.

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

که در آن E_i تعداد خطاهای یافت شده طی فعالیت مهندسی نرم افزار i و E_i+1 تعدا خطاهای یافت شده طی فعالیت مهندسی نرم افزار i+1 است که در فعالیت مهندسی نرم افزار i کشف نشده اند.

اصول و مفاہیم طراحی

طراحی چیست؟

طراحی، نمایش مهندسی معناداری از چیزی است که باید ساخته شود. باید بر اساس خواسته های کاربر و ضوابط موجود ردیابی شود.

در زمینه مهندسی نرم افزار، طراحی به چهار مورد می پردازد:

۱. داده ها

۲.معماری

۳. واسط ها

۴.مؤلفه ها

چه کسی طراحی را انجام می دهد؟

مهندسین نرم افزار سیستم های کامپیوتری را طراحی می کنند ولی مهارت های مورد نیاز در هر سطح متفاوت است.

در سطوح داده ها و معماری، طراحی بر روی الگوهایی تأکید دارد که به برنامه کاربردی که باید ساخته شود اعمال می شوند.

در سطح واسط، مسائل آرگونومیک حرف اول رامی زنند.

در سطح مؤلفه « روش برنامه نویسی » ما را به طرح های مؤثر داده و رویه ها هدایت می کند.

چرا طراحی مهم است؟

هیچ کس خانه ای را بدون نقشه نمی سازد. نرم افزار کامپیوتر به مراتب پیچیده تر از ساختمان است بنابرین نیاز به نقشه است.

مراحل طراحی چیست؟

طراحی با مدل خواسته ها شروع می شود. این مدل را به چهار سطح طراحی تقسیم می کنیم:

۱. ساختمان داده

۲.معماری سیستم

۳.نمایش واسط

۴.جزئیات سطح مؤلفه

در اثنای هر فعالیت طراحی از مفاهیم و قواعدی استفاده می کنیم تا کیفیت کار بالا باشد.

محصول کار چیست؟

معمولا یک مشخصات طراحی ایجاد می شود. این مشخصات حاوی مدل های طراحی است که داده ها، معماری، واسط ها و مؤلفه ها را توصیف می کند، هر کدام از این ها یک محصول طراحی است.

چگونه مطمئن شویم که طراحی به درستی انجام شد؟

در هر مرحله، محصولات طراحی نرم افزار مرور می شوند تا وضوح، صحت، تمامیت وسازگاری آنها درست باشد.

طراحی واسط

چگونگی برقراری ارتباط داخلی سیستم، ارتباط آن با سیستم های وابسته به آن و انسانهایی که با آن کار می کنند را توصیف می کند.

یک واسط حاکی از جریان اطلاعات (مثلا داده ها و یا کنترل) و نوع خاصی از رفتار است از این رو، نمودارهای جریان داده ها و کنترل، اکثر اطلاعات لازم برای طراحی واسط را فراهم می آورند.

اهمیت طراحی نرم افزار

- طراحی تنها راهی است که خواسته های مشتری به سیستم یا محصول نرم افزاری تبدیل می کند.
- طراحی نرم افزار به عنوان مبنایی برای تمام مراحل مهندسی نرم افزار و پشتیبانی نرم افزار عمل می کند.
- بدون طراحی، خطر ساخت بنایی ناپایدار وجود دارد یعنی بنایی که با مختصر تغییراتی فرو می ریزد ؛ بنایی که آزمایش آن دشوار است و کیفیت آن را نمی توان ارزیابی کرد، مگر در واپسین قدمهای فرآیند نرم افزار که زمان کوتاه است و مبالغ هنگفتی هزینه شده است.

تبدیل تحلیل به طراحی نرم افزار

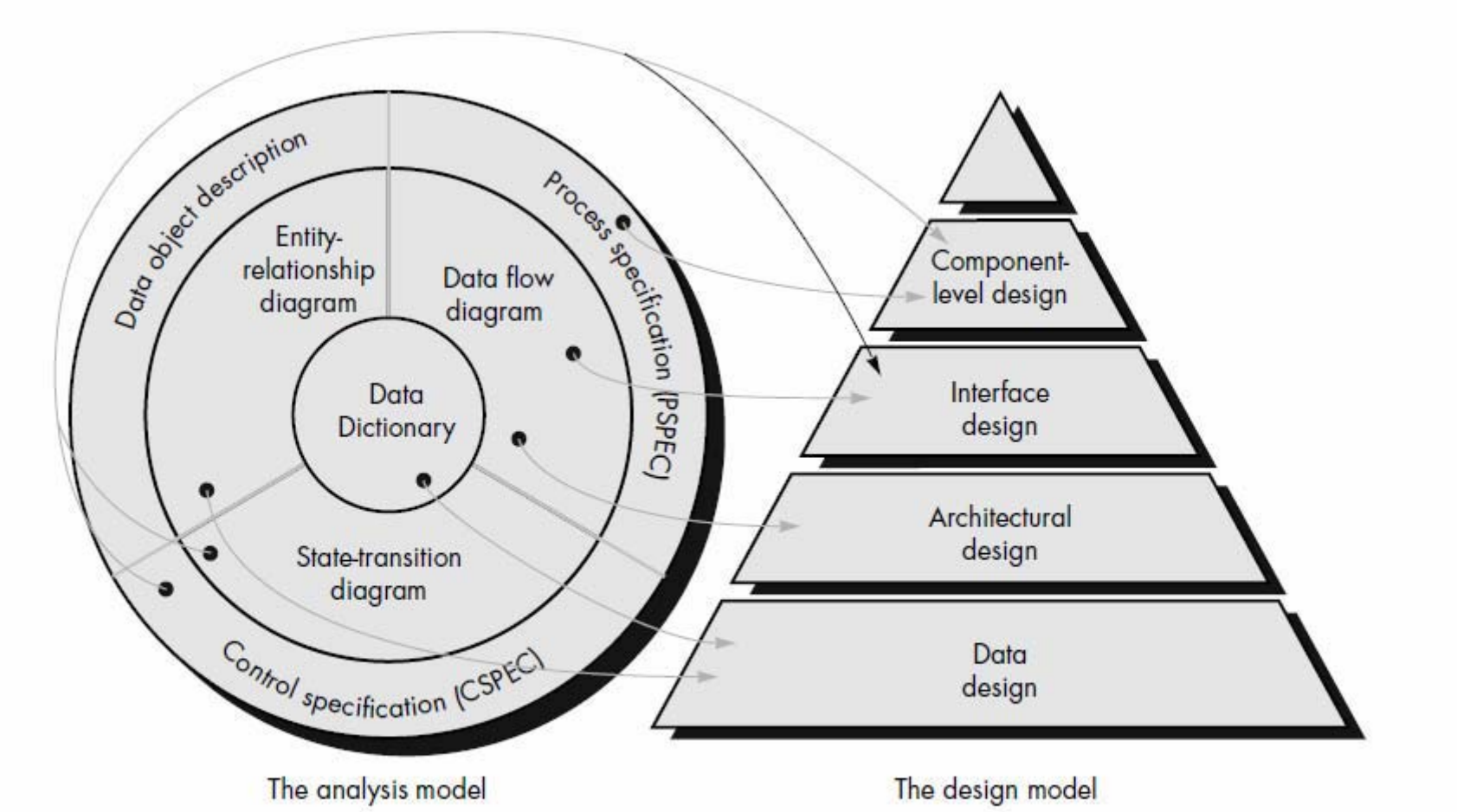


FIGURE 13.1 Translating the analysis model into a software design

مک گلوگلین سه ویژگی را به عنوان راهنمایی برای ارزیابی یک طراحی خوب پیشنهاد می کند:

- طراحی باید کلیه خواسته های واضح موجود در مدل تحلیل را پیاده سازی کرده و جایی برای تمامی خواسته های مبهم مشتری در نظر بگیرد.
- طراحی باید برای کسانی که کد را تولید می کنند، کسانی که نرم افزار را آزمایش می کنند و کسانی که آن را پشتیبانی می کنند راهنمایی قابل فهم باشد.
- طراحی باید تصویر کاملی از نرم افزار ارائه دهد که در برگیرنده دامنه های رفتاری، عملیاتی و داده ای از دیدگاه پیاده سازی باشد.

ملاکهای فنی جهت ارزیابی کیفیت نمایش طراحی

۱.طراحی باید ساختار معماریی داشته باشد که

- با استفاده از الگوهای طراحی شناخته شده ایجاد شده باشد
- از مؤلفه هایی تشکیل شده باشد که ویژگی های طراحی خوبی از خود نشان می دهند
- به شیوه ای تکاملی قابل پیاده سازی بوده در نتیجه پیاده سازی و آزمون آن آسان باشد.

۲. طراحی باید پیمانه ای باشد؛ یعنی نرم افزار باید به طور منطقی به عناصری افراز شده باشد که عملیات خاصی را اجرا می کنند.

۳. طراحی باید حاوی نمایش برجسته ای از داده ها، معماری، واسط ها و مؤلفه ها (پیمانه ها) باشد.

۴. طراحی باید منجر به ساختمان داده ای شود که مناسب اشیایی باشد که قراراست پیاده سازی شوند و از الگوهای داده ای قابل تشخیص گرفته شود.

۵. طراحی باید منجر به مؤلفه هایی شود که ویژگی های عملیاتی مستقلی از خود نشان دهند.

۶. طراحی باید به واسطهایی منجر شود که پیچیدگی ارتباطات میان پیمانه ها و محیط خارجی را کاهش دهد.

۷. طراحی باید با استفاده از یک روش تکرار به دست آید که توسط اطلاعات حاصل از تحلیل خواسته های نرم افزار هدایت می شوند.

پیمانه ای کردن

پیمانه ای، تنها صفتی از نرم افزار است که امکان مدیریت هوشمندانه یک برنامه را فراهم می کند.

قابلیت خوانایی نرم افزار یکپارچه (یعنی برنامه بزرگی که تنها از یک پیمانه تشکیل شود) پایین است.

تعداد مسیرهای کنترلی، تعداد متغیرها و پیچیدگی کلی باعث می شوند که درک برنامه تقریبا غیر ممکن شود.

فرض کنیم C(X) تابعی باشد که پیچیدگی مورد انتظار مسئله x و E(X) تابعی باشد که کار لازم برای حل مسئله x (بر حسب زمان) را تعیین می کند.

برای دو مسئله p1 و p2 اگر: **C(P1) > C(P2)**

خواهیم داشت: **E(P1)> E(P2)**

یک ویژگی جالب دیگردرحل مسئله:

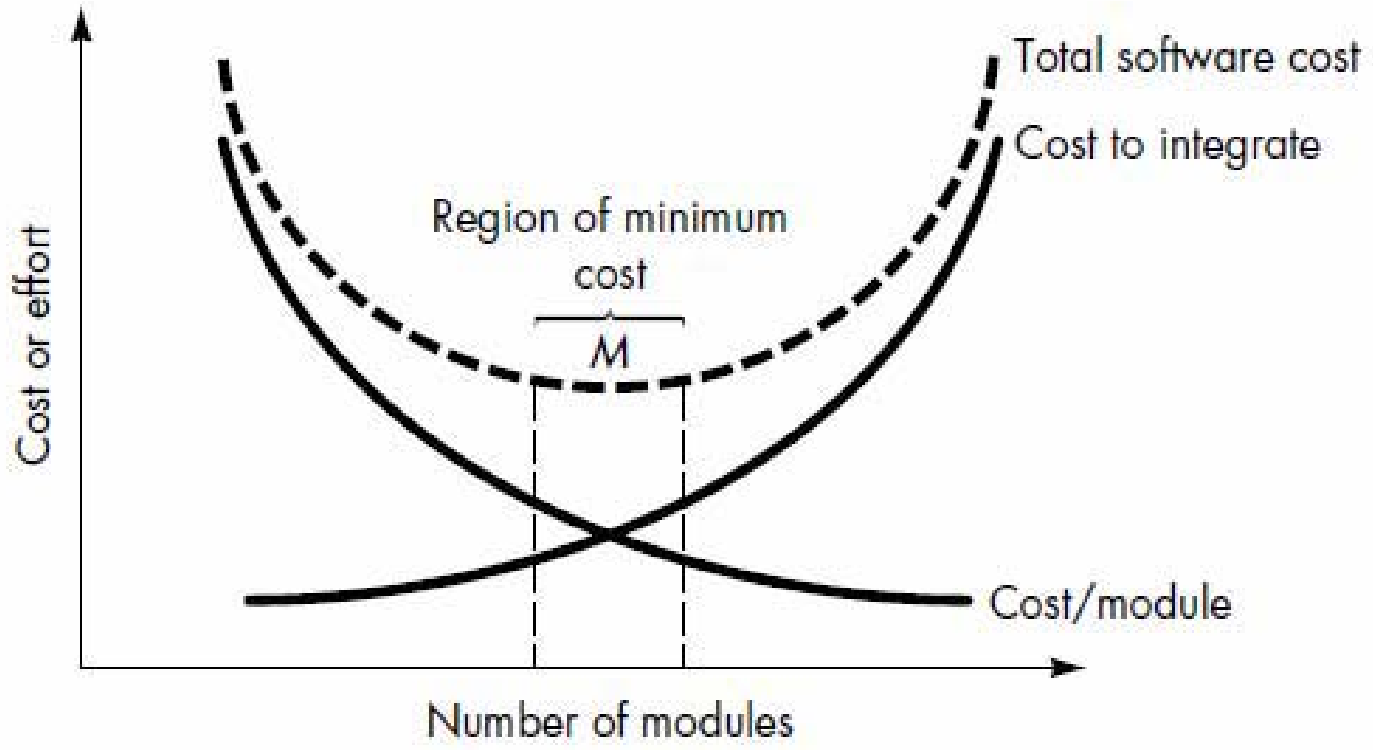
C(P1+ P2) > C(p1)+ C(P2)

پیچیدگی مسئله ای مرکب ازp1 وp2 بیشتر پیچیدگی تک تک این مسائل می باشد.

E(P1 + P2) > E(P1)+ (P2)

مدولاریته و هزینه نرم افزار

FIGURE 13.2
Modularity and software cost



شکل نشان می دهد که کار (هزینه) لازم برای یک پیمانهٔ نرم افزاری با افزایش تعداد کل پیمانه ها حتما کاهش می یابد.

اگر مجموعه خواسته ها یکسان نباشند تعداد بیشتر پیمانه ها به معنای کاهش اندازهٔ هر پیمانه است.

به موازات رشد تعداد پیمانه ها، کار (هزینه) مربوط به جامعیت پیمانه نیز افزایش می یابد.

یک تعداد معین m وجود دارد که به ازای آن هزینه توسعه کمینه می شود ولی لازم نیست خود را درگیر تعیین دقیق m کنیم.

مایر پنج ملاک تعریف می کند که با استفاده از آن می توانیم توانایی یک روش طراحی را در تعریف سیستم پیمانه ای ارزیابی کنیم:

- **تجزیه پذیری پیمانه ای**
- **ترکیب پذیری پیمانه ای**

درک پذیری پیمانه ای

تداوم پیمانه ای

حفاظت پیمانه ای

تجزیه پذیری پیمانه ای

اگر یک روش طراحی راهکاری سیستماتیک برای تجزیهٔ مسئله به مسائل کوچکتر فراهم آورد پیچیدگی کل مسئله کاهش یافته در نتیجه یک راهکار پیمانه ای کار آمد به دست می آید.

ترکیب پذیری پیمانه ای

اگر در یک روش طراحی بتوان مؤلفه های طراحی موجود (قابل استفاده ای مجدد) را در سیستم جدیدی مونتاژ کرد راهکاری پیمانه ای نتیجه می شود که در آن از دوباره کاری پرهیز شده است.

درک پذیری پیمانه ای

اگر پیمانه را بتوان به عنوان واحدی مستقل درک کرد (بدون رجوع به پیمانه های دیگر) آسانتر می توان آن را ساخت و تغییر داد.

تداوم پیمانه ای

اگر تغییرات کوچک در خواسته های سیستم منجر به تغییر دادن پیمانه های منفرد شوند (نه به تغییرات گسترده در کل سیستم) اثرات جانبی حاصل ازآن تغییر، کمترین ضربه را به پیکر سیستم وارد می آورد.

حفاظت پیمانه ای.

اگر در داخل پیمانه ای یک شرط نامطلوب رخ دهد و اثرات آن در آن پیمانه محدود شده باشد اثرات جانبی کاهش می یابد.

تکنیک های آزمون نرم افزار

آزمون نرم افزار چیست؟

هنگامی که کد منبع تولید شد، نرم افزار باید آزموده شود تا هر تعدادی از خطاها را که امکان داشته باشد، قبل از تحویل به مشتری کشف کرد. هدف، طراحی موارد آزمون است که احتمال یافتن خطا را بالا ببرند.

چه کسی آن را انجام می دهد؟

طی مراحل اولیه آزمون، مهندس نرم افزار کلیه آزمونها را انجام می دهد. ولی به موازات پیشرفت فرآیند آزمون، ممکن است کارشناسان آزمون نیز در این امر شرکت کنند.

چرا اهمیت دارد؟

بازبینی ها و فعالیتهای SQA دیگر می توانند خطاها راکشف کنند و می کنند ولی کافی نیستند. هر بار که برنامه اجرا شود، مشتری آن را می آزماید ! بنابراین، باید برنامه را پیش از آنکه به دست مشتری برسد با هدف یافتن و حذف خطاها اجرا نمود. برای یافتن حداکثر تعداد ممکن خطاها، باید آزمونهایی را به طور سیستماتیک اجرا کرد و موارد آزمودنی را با استفاده از تکنیکهای منظم، طراحی نمود.

مراحل کار کدام است؟

نرم افزار از دو دیدگاه متفاوت آزمایش می شود:

۱-با استفاده از تکنیکهای طراحی موارد آزمون با منطق داخلی برنامه (جعبه سفید) تمرین می شود.

۲- با خواسته های نرم افزار با استفاده از تکنیکهای طراحی مورد آزمون « جعبه سیاه » تمرین می شود. در هر دو حال، هدف، یافتن حداکثر تعداد خطاها با حداقل مقدار کار و زمان است.

محصول کاری چیست؟

مجموعه ای از موارد آزمون که برای تمرین با منطق داخلی و خواسته های خارجی، طراحی و مستندسازی شده اند؛ نتایج مورد انتظار تعیین و نتایج واقعی ثبت می شوند.

چگونه مطمئن شوم که درست از عهده کارها برآمده ام؟

هنگامی که آزمون را شروع می کنید، دیدگاه خود را تغییر دهید. موارد آزمون را به شیوه ای منظم طراحی کنید و آنها را بازبینی کنید تا مطمئن شوید که کامل هستند.

اهداف آزمون

گلن مایرز درباره چند قاعده در مورد اهداف آزمون بیان می کند.

۱.آزمون، فرآیند اجرای برنامه به قصد یافتن خطاست.

۲.مورد آزمون خوب، موردی است که احتمال یافتن خطاهای کشف شده در آن، بالا باشد.

۳. آزمون موفق، آزمونی است که خطاهای کشف نشده را کشف می کند.

اهداف بالا نشانگر یک تغییر دیدگاه زیبا هستند و برخلاف این دیدگاه عامیانه که آزمون موفق، آزمونی است که در آن خطایی یافته نشود.

مجموعه ای از اصول پیشنهاد شده توسط دیویس در مورد موارد آزمون موثر:

- همه آزمونها باید تا حد خواسته های مشتری قابل ردیابی باشند.
- آزمون باید مدتها قبل از شروع آزمون، طرح ریزی شود.
- اصل پارتو در آزمون نرم افزار صدق می کند.
- آزمون باید در ابعاد کوچک آغاز شود و به ابعاد بزرگتر گسترش یابد.
- برای آنکه آزمون بیشترین بازدهی را داشته باشد باید توسط یک شخص ثالث بی طرف انجام شود.

همه آزمونها باید تا حد خواسته های مشتری قابل ردیابی باشند.

هدف آزمون نرم افزار، کشف خطاها است. یعنی اکثر نقایص شدید (از دیدگاه مشتری) آنهایی هستند که باعث می شوند برنامه نتواند خواسته های او را برآورده سازند.

آزمون باید مدتها قبل از شروع آزمون، طرح ریزی شود.

طرح ریزی آزمون می تواند به محض کامل شدن مدل خواسته ها آغاز شود.

تعریف مشروح موارد آزمون می تواند به محض منسجم شدن مدل طراحی آغاز شود.

همه آزمونها را می توان پیش از تولید هرگونه کد، برنامه ریزی و طراحی کرد.

اصل پارتو در آزمون نرم افزار صدق می کند.

اصل پارتو بیان می کند که ۸۰ درصد همه خطاهای کشف شده طی آزمون، احتمالاً در ۲۰ درصد همه مولفه های برنامه قابل کشف هستند. مسئله، جداسازی مولفه های مظنون و آزمودن کامل آنهاست.

آزمون باید در ابعاد کوچک آغاز شود و به ابعاد بزرگتر گسترش یابد.

اولین آزمونها بر روی هر یک از مولفه های انجام می شوند.

با پیشرفت آزمون، خطاهای مجموعه ای از مولفه های مجتمع و سپس کل سیستم یافته می شود.

برای آنکه آزمون بیشترین بازدهی را داشته باشد، باید توسط یک شخص ثالث بی طرف انجام شود.

منظور از بیشترین بازدهی آن است که خطاها را با احتمال بیشتری بیابد، مهندس نرم افزاری که سیستم را ایجاد کرده است، بهترین کسی نیست که باید همه آزمونها را انجام دهد.

لیست کنترلی از نکات طراحی و ویژگیها ی آزمون پذیری

- قابلیت کار
- قابلیت مشاهده
- کنترل پذیری
- تجزیه پذیری
- سادگی
- پایداری
- درک پذیری

قابلیت کار

« هر چه بهتر کار کند، آزمون آن کارآمدتر است.»

•سیستم چند اشکال معدود دارد (این اشکالات باعث افزایش تحلیل و گزارش دهی به فرآیند آزمون می شود.)

•هیچ اشکالی مانع اجرای آزمونها نمی شود.

•محصول در مراحل عملیاتی تکامل می یابد (امکان توسعه و آزمون همزمانی وجود دارد.)

قابلیت مشاهده

« آنچه می بینید، همان است که آزمایش می کنید. »

- برای هر ورودی یک خروجی متمایز تولید می شود.
- متغیرها و حالتهای سیستم در اثنای اجرا قابل مشاهده و استفسارند.
- متغیرها و حالتهای گذشته سیستم قابل مشاهده و استفسارند. (مثل ثبت وقایع تراکنشی)
- همه وقایعی که خروجی را تحت تاثیر قرار می دهند، قابل مشاهده اند.
- خروجی نادرست به آسانی قابل شناسایی است.
- خطاهای داخلی به طور خودکار از طریق راهکارهای خودآزمایی آشکار می شوند.
- خطاهای داخلی به طور خودکار گزارش می شوند.
- کد منبع قابل دستیابی است

کنترل پذیری

«هر چه بهتر بتوان نرم افزار را کنترل کرد، آزمون را بیشتر می توان خودکار و بهینه کرد.»

- همه خروجیهای ممکن را می توان از طریق ترکیبی از ورودیها تولید نمود.
- همه کدها از طریق ترکیبی از ورودی ها قابل اجرا هستند.
- متغیرها و حالتهای سخت افزاری و نرم افزاری مستقیماً توسط مهندس آزمودن قابل کنترل هستند.
- فرمتهای خروجی و ورودی، سازگار و ساخت یافته هستند.
- آزمونها را به راحتی می توان مشخص، خودکار و بازسازی کرد.

تجزیه پذیری

« با کنترل دامنه کاربرد آزمون، می توان مسائل را سریعتر جداسازی کرد و آزمونهای مجدد رابا هوشمندی بیشتر انجام داد. »

- سیستم نرم افزاری از پیمانه های مستقل ساخته می شود.
- پیمانه های نرم افزاری را می توان مستقل از هم آزمود.

سادگی

« هرچه مورد آزمون کوچکتر باشد، سریعتر می توان آن را آزمود. »

•سادگی عملیاتی. مثلاً مجموعه ویژگیها، حداقل مجموعه لازم برای برآوردن خواسته ها است.

•سادگی ساختاری. مثلاً معماری به صورت پیمانه ای درآمد تا انتشار خطاها را به حداقل برساند.

•سادگی کد . مثلاً یک استاندارد کد نویسی رعایت می شود که وارسی و نگهداری آن آسان باشد.

پایداری

« هر چه تعداد تغییرات کمتر باشد، آزمون کمتر با مانع مواجه می شود. »

- تغییرات نرم افزار چندان زیاد نیست.
- تغییرات نرم افزار کنترل شده هستند.
- تغییرات نرم افزار آزمونهای موجود را بی اعتبار نمی کنند.
- نرم افزار به خوبی از پس شکستها بر می آید.

درک پذیری

« هر چه اطلاعات بیشتری داشته باشیم، آزمون هوشمندانه تر انجام می شود»

- طراحی به خوبی درک شده است
- وابستگی میان مولفه های داخلی، خارجی و مشترک به خوبی درک شده است.
- تغییرات در طراحی مورد تبادل قرار گرفته اند.
- مستندات فنی بلافاصله قابل دستیابی اند.
- مستندات فنی به خوبی سازمان یافته اند.
- مستندات فنی مشخص و مفصل هستند.
- مستندات فنی صحیح هستند.

کینر، فالک و نگورین صفات زیر را برای یک آزمون « خوب » بر می شمرند.

- آزمون خوب با احتمال زیادی خطاها را می یابد.
- آزمون خوب دارای زواید نیست.
- یک آزمون خوب باید « بهترین » باشد.
- آزمون خوب نباید بیش از حد ساده و نه بیش از حد پیچیده باشد.

آزمون خوب با احتمال زیادی خطاها را می یابد

برای حصول این منظور، آزمونگر باید نرم افزار را بشناسد و کوشش کند تا یک تصویر ذهنی از چگونگی شکست احتمالی نرم افزار بسازد.

آزمون خوب دارای زواید نیست.

زمان و منابع آزمون، محدود است. در اجرای آزمونی که هدف آن با هدف یک آزمون دیگر یکی است هیچ نکته ای وجود ندارد.

مثال: پیمانه ای از یک نرم افزار، برای شناسایی کلمه عبور کاربر جهت فعال و غیر فعال کردن سیستم طراحی می شود. برای مثال، کلمه عبور نامعتبر 1234 نباید توسط سیستمی پذیرفته شود که طبق برنامه ریزی، کلمه عبور معتبر 8080 را می پذیرد. یک ورودی آزمایشی دیگر مثلاً ۱۲۳۵ همان هدف ۱۲۳۴ را دنبال می کند لذا زاید است،

ورودیهای نامعتبر همچون ۸۰۸۱ یا ۸۱۸۰ دارای تفاوتی ظریف بوده کوشش می کنند تا نشان دهند که برای کلمات عبور نزدیک به کلمه عبور معتبر، ولی نه همسان با آنها، خطایی وجود دارد.

یک آزمون خوب باید « بهترین » باشد.

در گروهی از آزمونها که هدف و قصدی مشابه دارند، محدودیتهای منابع و زمان، ممکن است فقط با اجرای فقط زیر مجموعه ای از این آزمونها تعدیل شوند. در چنین مواردی، آزمونی به کار گرفته می شود که با احتمال بیشتری خطا را می یابد.

آزمون خوب نباید بیش از حد ساده و نه بیش از حد پیچیده باشد.

گرچه ممکن است تعدادی آزمون را در یک آزمون خلاصه کرد، اثرات جانبی این روش ممکن است خطاها را نادیده بگیرد. به طور کلی، هر آزمون باید جداگانه اجرا شود.

طراحی موارد آزمون

طراحی آزمونها برای نرم افزار و هر محصول مهندسی دیگر، می تواند به اندازه طراحی خود محصول اولیه دشوار باشد. با به خاطر داشتن اهداف آزمون، باید آزمونهایی را طراحی کنیم که احتمال یافتن خطاها در حداقل زمان، بیشتر باشد.

هر محصول مهندسی را می توان به یکی از دو روش زیر آزمایش کرد:

- با دانستن عملکرد خاصی که نرم افزار برای انجام آن طراحی شده است، می توان آزمونهایی طراحی کرد که نشان می دهند هر عملکرد به طور کامل درست است، ودر عین حال، در هر عملکرد به دنبال یافتن خطاها هستند. این روش اول را آزمون جعبه سیاه می نامند.
- با دانستن طرز کار داخلی محصول، می توان آزمونهایی ترتیب داد که اطمینان دهند عملیات داخلی طبق مشخصه اجرا می شوند و با همه مولفه های داخلی به طور مناسب تمرین شده است. این روش را آزمون جعبه سفید می نامند.

آزمون جعبه سیاه نرم افزار

آزمون جعبه سیاه، جنبه های بنیادی سیستم را بدون در نظر گرفتن ساختار منطقی داخلی نرم افزار، مورد بررسی قرار می دهد. آزمونهایی است که در واسط نرم افزار اجرا می شوند و برای اهداف زیربه کار می روند:

آیا عملکردهای نرم افزار، انجام پذیر هستند،

ورودی به طور مناسب پذیرفته شده است.

خروجی به درستی تولید شده است.

آیا جامعیت اطلاعات خارجی (مثل بانک اطلاعاتی) حفظ شده است؟

آزمون جعبه سفید نرم افزار

آزمون جعبه سفید نرم افزار، بررسی دقیق جزئیات رویه ای را دربر می گیرد، مسیرهای منطقی در سرتاسر نرم افزار آزمون می شود؛ برای این منظور، موارد آزمونی طراحی می شود که روی مجموعه مشخصی از شرایط و یا حلقه ها کار می کنند.

وضعیت برنامه ممکن است در نقاط گوناگون مورد بررسی قرار گیرد تا تعیین شود که آیا وضعیت مورد انتظار یا ارزیابی شده با وضعیت واقعی شباهت دارد یاخیر.

آیا آزمون جعبه سفید، غیر عملی است؟

تعداد محدودی از مسیرهای منطقی را می توان انتخاب و با آنها تمرین کرد.

ساختمان داده های مهم را می توان بررسی کرد و از اعتبار آنها اطمینان یافت.

صفات هر دو روش جعبه سیاه و جعبه سفید را می توان در هم آمیخت و روشی فراهم آورد که واسط نرم افزار را اعتبار سنجی کند و به طور گزینشی اطمینان دهد که عملیات داخلی نرم افزار صحیح هستند.

آزمون جعبه سفید

آزمون جعبه سفید یا جعبه شیشه ای، یک روش طراحی برای موارد آزمونی است که برای به دست آوردن موارد آزمون، از ساختار کنترلی طراحی رویه ای استفاده می کند.

با استفاده از متدهای آزمون جعبه سفید می توان موارد آزمونی به دست آورد که:

۱.تضمین می کنند که همه مسیرهای مستقل در یک پیمانه حداقل یک بار امتحان شده اند.

۲.همه تصمیم گیریهای منطقی را در دو بخش درست و غلط امتحان کنند.

۳.همه حلقه ها را در مرزها و در داخل مرزهای عملیاتی آنها اجرا کنند.

۴.ساختمان داده های داخلی را امتحان کنند تا اعتبار آنها ثابت شود.

چرا باید وقت و انرژی خود را صرف آزمودن جزئیات منطقی کنیم؟

چرا همه انرژی خود را صرف آزمونهای جعبه سیاه نکنیم؟

پاسخ در ماهیت نقایص نرم افزاری نهفته است.

- خطاهای منطقی و فرضیات نادرست، با احتمال اینکه یک مسیر از برنامه ای اجرا شود نسبت عکس دارند. هنگامی که یک عملکرد، شرایط یا کنترل خارج از جریان اصلی را طراحی و پیاده سازی می کنیم، ممکن است دچار خطا شویم. پردازش تکراری (هرروزه) معمولا به خوبی درک می شود، در حالی که در پردازش «موارد خاص » معمولا مشکل پیش می آید.
- غالبا بر این باوریم که یک مسیر منطقی خاص، همواره به طور منظم اجرا می شود. جریان منطقی یک برنامه گاهی هوشمندانه نیست یعنی فرضیات ناخودآگاه ما درباره جریان کنترل و داده ها ممکن است باعث شود که دچار خطاهای طراحی شویم که فقط هنگام شروع آزمون مسیر کشف می شوند.
- خطاهای تایپی ماهیتی تصادفی دارند. هنگامی که برنامه ای به کد منبع زبان برنامه نویسی ترجمه می شود این احتمال وجود دارد که برخی خطاهای تایپی رخ دهد. بسیاری از آنها توسط راهکارهای چک کننده تایپ و فرمت کشف می شوند ولی بقیه ممکن است تا زمان شروع آزمون کشف نشوند. این احتمال وجود دارد که یک خطای تایپی در یک مسیر منطقی مبهم وجود داشته باشد.

آزمون مسیرهای پایه

آزمون مسیر، مبنای یک تکنیک آزمون جعبه سفید است که نخستین بار توسط نام مک کیب پیشنهاد شد.

روش مسیرهای پایه، طراح موارد آزمون را قادر می سازد تا میزانی منطقی از پیچیدگی رویه ای به دست آورد و از این میزان به عنوان راهنمایی جهت تعریف یک مجموعه پایه از مسیرهای اجرا استفاده کند.

موارد آزمون به دست آمده برای امتحان کردن این مجموعه پایه، هر دستور از برنامه را حداقل یک بار در اثنای آزمون اجرا خواهند کرد.

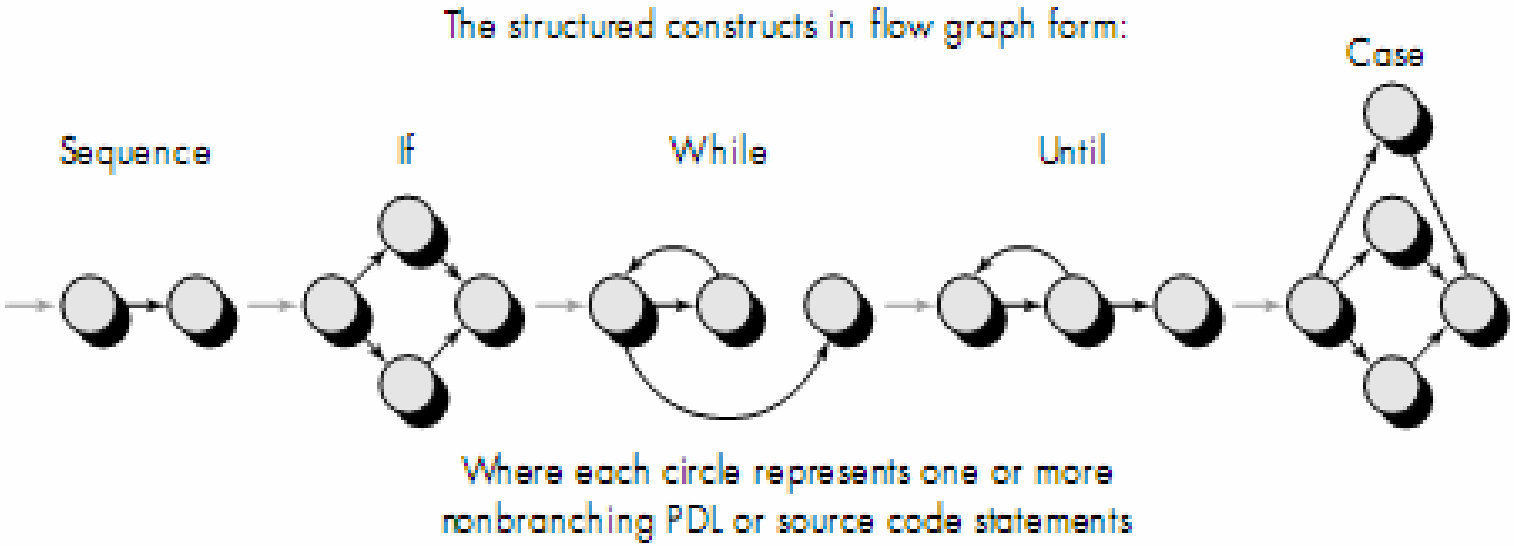
نشانه گذاری گراف جریان

پیش از آنکه بتوان روش مسیرهای پایه را معرفی نمود، یک نشانه گذاری ساده، موسوم به گراف جریان را باید برای نمایش دادن جریان معرفی کرد.

گراف جریان، جریان کنترل منطقی را با استفاده از نشانه گذاری در شکل ۱-۱۷تصویر می کند.

متناظر با هر ساختار ساخت یافته، یک نماد گراف جریان وجود دارد.

FIGURE 17.1
Flow graph notation



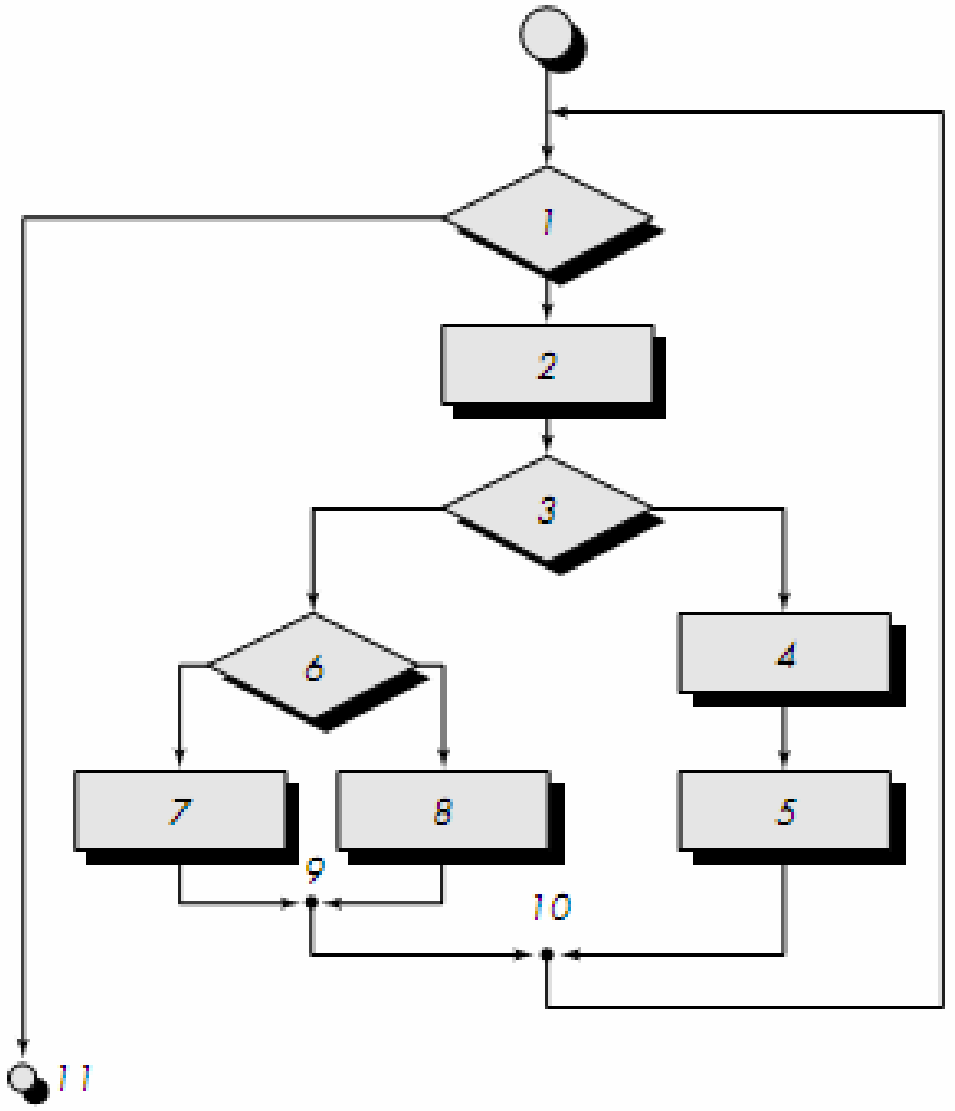
از یک نمودار گردشی برای تصویر کردن ساختار کنترلی برنامه استفاده می شود.

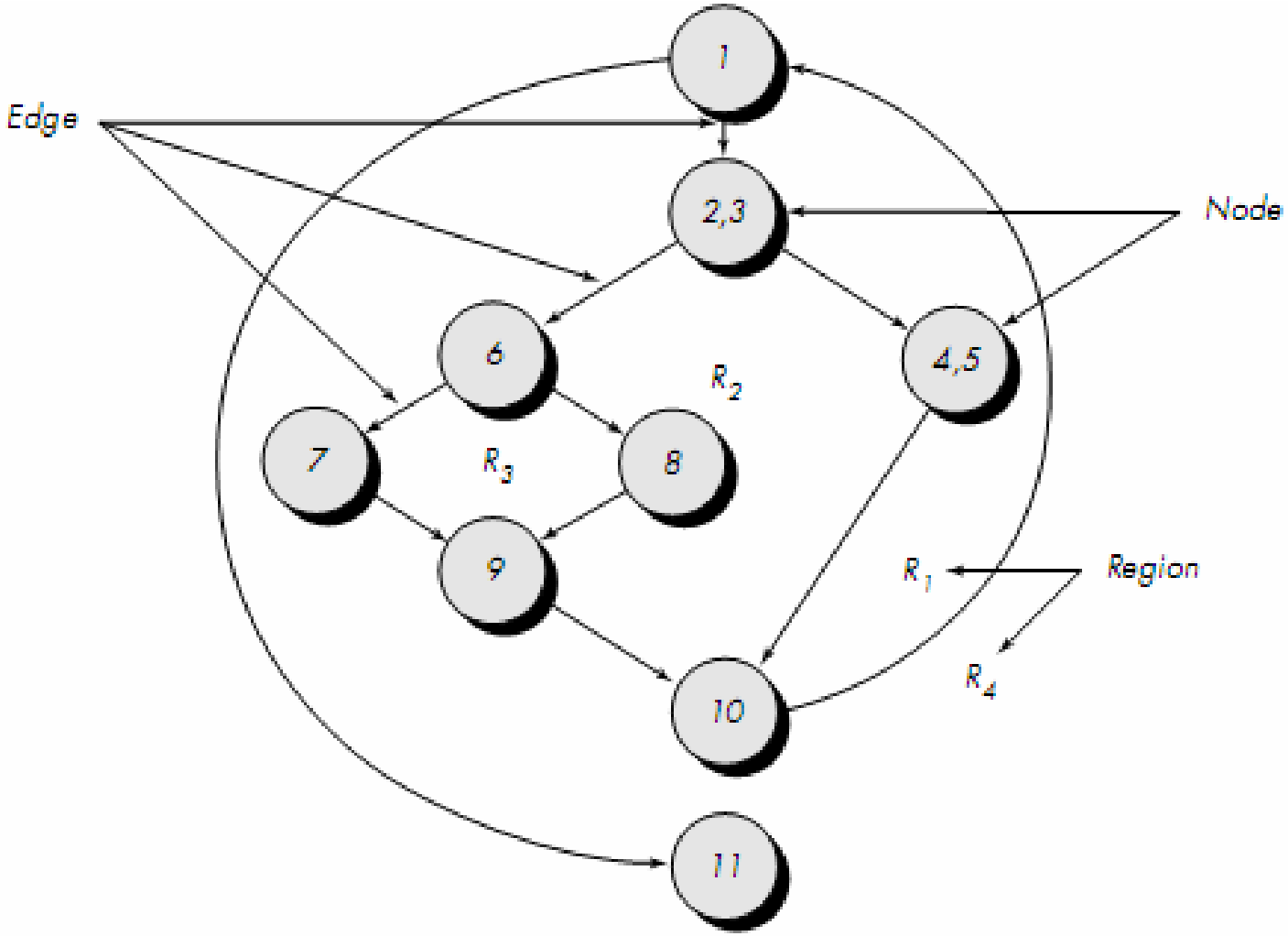
شکل ۲-۱۷-ب نمودار گردشی را به صورت گراف جریان آن در آورده است (با این فرض که هیچ شرط مرکبی در لوزیهای تصمیم گیری نمودار گردشی وجود نداشته باشد).

هردایره که گره گراف جریان خوانده می شود یک یا چند دستور رویه ای را نشان می دهد.

ترتیبی از مستطیلهای پردازشی و یک لوزی تصمیم گیری را می توان در یک گروه منفرد خلاصه نمود.

FIGURE 17.2
Flowchart, (A) and flow graph (B)





(B)

پیکان های روی گراف جریان که یال یا پیوند خوانده می شوند، نشانگر جریان کنترل بوده مشابه پیکان های نمودار گردشی هستند.

هر رویه باید در یک گره پایان یابد، حتی اگر گره هیچ دستور رویه ای را نشان ندهد. مثلا نماد مربوط به ساختمان if-then-else ببینید.

مساحتهای محصور شده توسط یالها و گره ها را ناحیه می نامند.

هنگام شمارش نواحی مساحت خارج از گراف را نیز به عنوان یک ناحیه در نظر گرفته آن را لحاظ می کنیم.

هنگام مواجهه با شرطهای مرکب در یک طراحی رویه ای، تولید گراف جریان قدری پیچیده تر می شود.

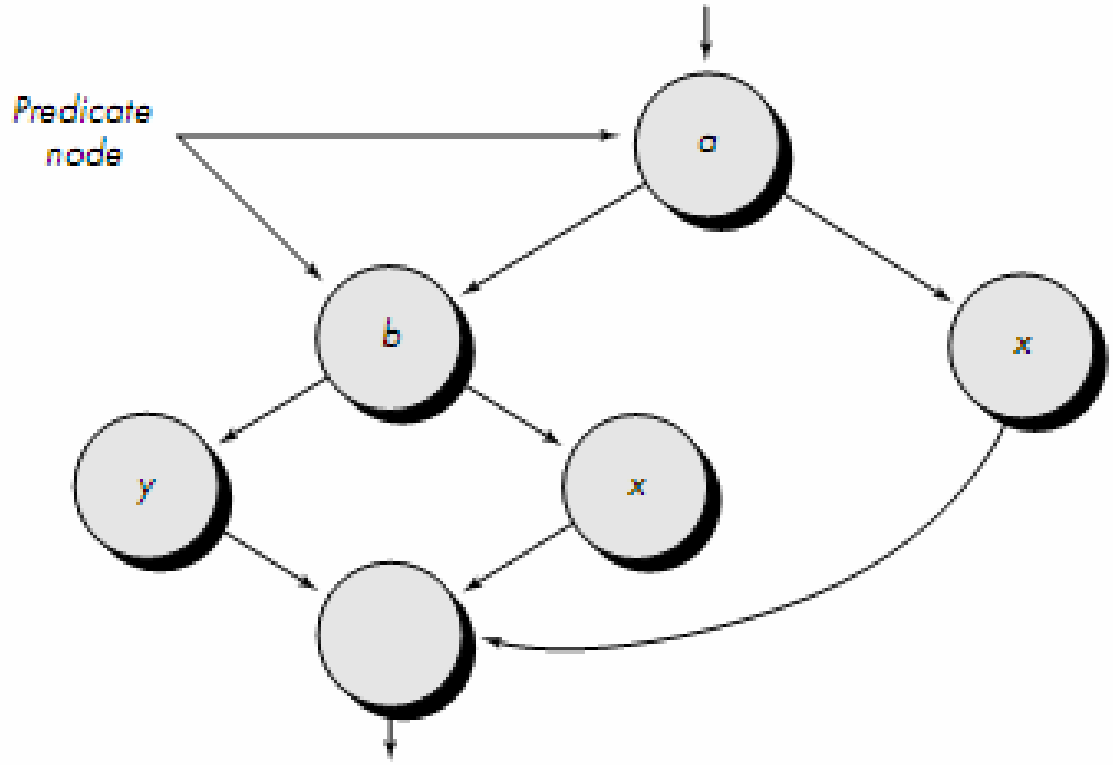
شرط مرکب زمانی رخ می دهد که یک یا چند عملگر بولی (NAND,NOR,AND,OR منطقی) در یک دستور شرطی وجود داشته باشند.

در شکل ۳-۱۷ دستور PDL به گراف جریان ترجمه می شود. توجه داشته باشید که برای هریک از شرایط aو b در دستورIF a OR b یک گروه جداگانه ایجاد می شود.

هرگره که حاوی یک شرط باشد، گره گزاره ای خوانده می شود و با دو یا چند پیکان که از آن بیرون می آیند، مشخص می شود.

FIGURE 17.3
Compound logic

```
.
.
.
IF a OR b
then procedure   x
else procedure   y
ENDIF
```



پیچیدگی سیکلوماتیک

پیچیدگی سیکلوماتیک، یک معیار نرم افزاری است که میزانی کمی از پیچیدگی منطقی یک نرم افزار را ارائه می دهد.

در روشهای آزمون پایه، مقدار پیچیدگی سیکلومات، تعداد مسیرهای مستقل در مجموعه پایه یک برنامه را تعیین می کند و یک حد فوقانی برای تعداد مسیرهایی فراهم می آورد که باید اجرا شوند تا اطمینان حاصل شود که همه دستورها حداقل یک بار اجرا شده اند.

مسیر مستقل

هر مسیری از برنامه است که حداقل یک مجموعه جدید از دستورهای پردازش یا یک دستور شرطی را معرفی کند.

اگر مسیر مستقل بر حسب گراف جریان بیان شود، حداقل باید در راستای یک یال حرکت کند که پیش از تعریف مسیر از آن عبور نشده باشد.

برای مثال، مجموعه ای از مسیرهای مستقل در شکل ۲-۱۷-ب در گراف جریان نشان داده شده اند:

۱-۱۱: مسیر ۱

۱-۲-۳-٤-٥-۱۰-۱-۱۱: مسیر ۲

۱-۲-۳-٦-۸-۹-۱۰-۱-۱۱: مسیر ۳

۱-۲-۳-٦-۷-۹-۱۰-۱-۱۱: مسیر٤

هر مسیر جدید یک یال جدید را معرفی می کند.

مسیر زیر: ۱-۲-۳-۴-۵-۱۰-۱-۲-۳-۶-۸-۹-۱۰-۱-۱۱

به عنوان مسیری مستقل در نظر گرفته نمی شود زیرا صرفا تلفیقی از مسیرهای مشخص شده از قبل است.

مسیرهای ۱،۲،۳،۴ که تعریف شدند یک مجموعه پایه را برای گراف جریان شکل ۲-۱۷-ب تشکیل می دهند.

لازم به ذکر است که مجموعه پایه منحصر بفرد نیست. در حقیقت چند مجموعه پایه متفاوت را می توان برای یک طراحی رویه ای مفروض به دست آورد.

چگونه باید بدانیم که به دنبال چند مسیر باید بگردیم؟

پاسخ را در محاسبه پیچیدگی سیکلوماتیک می توان جستجو کرد.

پیچیدگی سیکلوماتیک ریشه در نظریه گرافها دارد و یک معیار نرم افزاری سودمند را فراهم می آورد.

پیچیدگی به یکی از سه شیوه زیر محاسبه می شود:

۱.تعداد نواحی گراف جریان متناظر با پیچیدگی سیکلوماتیک

۲.پیچیدگی سیکلوماتیک برای یک گراف جریان V(G) به صورت زیر تعریف می شود

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

E تعداد یالهای گراف جریان و N تعداد گره های آن است.

۳.پیچیدگی سیکلوماتیکV(G) برای گراف جریان G به صورت زیر تعریف می شود:

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

P تعداد گره های گزاره ای موجود در گراف جریان G است.

محاسبه پیچیدگی سیکلوماتیک گراف جریان شکل ۲-۱۷-ب

گراف گردشی چهار ناحیه دارد.

V(G) =یال۱۱- گره۹+۲=۴

V(G) = ای گزاره گره۳+۱= ۴

به دست آوردن موارد آزمون

روش آزمون مسیرهای پایه را می توان در یک طراحی رویه ای یا کد منبع به کار برد.

در این بخش آزمون مسیرهای پایه را به صورت چند مرحله ارائه خواهیم داد.

به عنوان مثالی برای نشان دادن هر یک از مراحل این روش، از رویه average (شکل ۴-۱۷) استفاده می کنیم.

الگوریتم average حاوی حلقه ها و شرطهای مرکب است.

```
PROCEDURE average;
This procedure computes the average of 100 or fewer numbers that lie between
    bounding values; it also computes the  sum and the total number valid.
INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;
TYPE value[1:100] IS SCALAR ARRAY; TYPE average, total.input, total.valid; minimum,
    maximum, sum IS SCALAR;  TYPE i IS INTEGER;
    i = 1; total.input = total.valid = 0;  sum = 0;
    DO WHILE value[i] <> –999 AND total.input < 100
            increment total.input by 1;
            IF value[i] > = minimum AND value[i] < = maximum
                    THEN increment total.valid by 1;
                        sum = s sum + value[i]
                    ELSE skip
            ENDIF
         increment i by 1;
    ENDDO
    IF total.valid > 0
            THEN average = sum / total.valid;
          ELSE average = –999;
    ENDIF
END average
```

```
PROCEDURE average;

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

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

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

برای به دست آوردن مجموعه پایه باید مراحل زیر را اجرا نمود:

- استفاده از طراحی یا کد به عنوان یک بستر و رسم گراف جریان مربوطه
- تعیین پیچیدگی سیکلوماتیک گراف جریان حاصل
- تعیین مجموعه پایه برای مسیرهای مستقل خطی
- تهیه موارد آزمونی که اجرای همه مسیرها در مجموعه پایه را الزامی کنند.

- **استفاده از طراحی یا کد به عنوان یک بستر و رسم گراف جریان مربوطه**

با توجه به PDLمربو ط به رویه average در شکل ۴-۱۷، گراف جریان با شماره گذاری آن دسته از دستورهای PDL ایجاد می شود که در گره های گراف جریان مربوطه تصویر شده اند.

- **پیچیدگی سیکلوماتیک گراف جریان حاصل را تعیین کنید.**

پیچیدگی سیکلوماتیک V(G) با استفاده از روشهای ذکر شده، تعیین می شود.

V(G) را می توان بدون توسعه یک گراف جریان، با شمارش کلیه دستورهای شرطی در PDL و افزودن یک واحد محاسبه کرد.

V(G)= ۶ ناحیه

V(G)= ۱۷یال -گره ۳+۲=۶

V(G)= ۵+۱=۶ گره گزاره ای

تعیین مجموعه پایه برای مسیرهای مستقل خطی

مقدار V(G)، با استفاده از تعداد مسیرهای مستقل خطی موجود در ساختار کنترلی برنامه مشخص می شود.

در مورد رویه average انتظار داریم شش مسیر مشخص شود:

مسير١: ١-٢-١٠-١١-١٣

مسير٢: ١-٢-١٠-١٢-١٣

مسير٣: ١-٢-٣-١٠-١١-١٣

مسير٤: ١-٢-٣-٤-٥-٨-٩-٢-...

مسير٥: ١-٢-٣-٤-٥-٦-٨-٩-٢-...

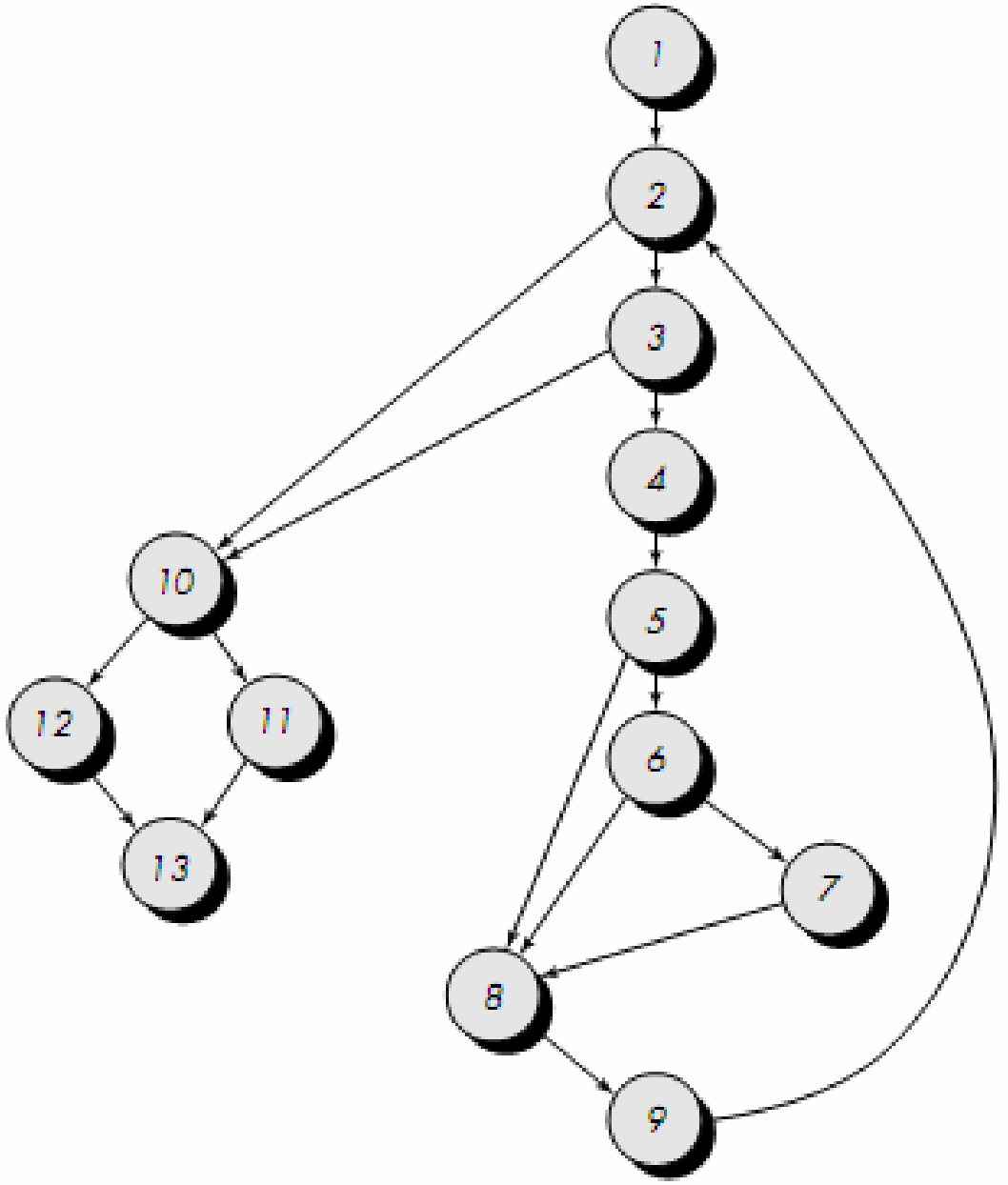
مسير٦: ١-٢-٣-٤-٥-٦-٧-٨-٩-٢-...

سه نقطه ای که بعد از مسیرهای ۴،۵و ۶ می آید نشان می دهد که هر مسیر در باقیمانده ساختار کنترلی، قابل توجه است.

غالبا خوب است در به دست آوردن موارد آزمون از گره های گزاره ای کمک گرفت.

دراین مورد گره های ۳،۵،۶،۱۰و۲ گره های گزاره ای هستند.

FIGURE 17.5
Flow graph for the procedure **average**



•موارد آزمونی را تهیه کنید که اجرای همه مسیرها در مجموعه پایه را الزامی کنند.

داده ها را باید طوری انتخاب کرد که به موازات آزموده شدن هر مسیر، شرایط در گره های گزاره ای بطور مناسب تنظیم شود موارد آزمونی عبارتند از:

مورد آزمون مسیر ۱:

ورودی معتبر، که در آن k<i(i در زیر تعریف شده است)=value(k)

که در آن value(i)=-۹۹۹

نتایج مورد انتظار: میانگین صحیح بر اساسk مقدار و حاصل جمع مناسب.

مورد آزمون مسیر ۲:

value(1) = – ۹۹۹

نتایج موردانتظار:

۹۹۹– =average؛ مقادیر حاصل جمع دیگر، مقادیر اولیه هستند.

مورد آزمون مسیر۳:

کوشش برای پردازش ۱۰۱ مقدار یا بیشتر.

۱۰۰مقدار نخست باید معتبر باشد.

نتایج مورد انتظار: همانند مورد آزمون ۱.

مورد آزمون مسیر ٤:

ورودی معتبر که درآن ۱۰۰>i=value(i)

(که درآن i >k) value(k)>minimum

نتایج مورد انتظار: میانگین صحیح براساس kمقدار و حاصل جمع مناسب.

مورد آزمون مسیر ۵:

ورودی معتبر که درآن ۱۰۰>i=value(i)

(که درآن i >k) value(k)>maximum

نتایج مورد انتظار: میانگین صحیح براساس nمقدار و حاصل جمع کل.

مورد آزمون مسیر ٦:

ورودی معتبر که در آن value(i)=i<۱۰۰

نتایج مورد انتظار: میانگین صحیح بر اساس nمقدار و حاصل جمع کل

ماتریس گراف

یک ماتریس مربعی است که اندازه آن (یعنی تعداد سطر ها و ستونهای آن)برابر تعداد گره های موجود در گراف جریان است.

هر سطر و ستون ماتریس، متناظر با یکی از گره ها است.

مدخلهای ماتریس با اتصالات (یعنی یالهای)میان گره ها متناظرند.

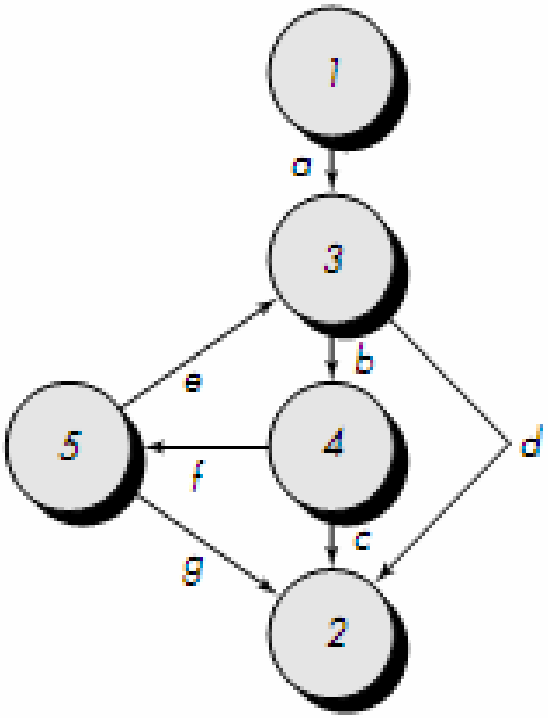
با افزودن وزن پیوند به هر یک از مدخلهای ماتریس می توان آن را به ابزاری پر قدرت برای ارزیابی ساختار کنترلی برنامه در اثنای آزمون تبدیل کرد.

وزن پیوند اطلاعاتی درباره جریان کنترل فراهم می آورد. وزن پیوند در ساده ترین شکل خود برابر ۱(وجود ارتباط)یا۰ (نبود ارتباط) است.

خواص جالب دیگری که می توان به اوزان پیوند نسبت داد:

- احتمال اینکه یک پیوند (یال)اجرا شود
- زمان پردازش صرف شده برای طی کردن یک پیوند
- حافظه لازم برای طی کردن یک پیوند
- منابع لازم برای طی کردن یک پیوند

FIGURE 17.6
Graph matrix



Flow graph

Node \ Connected to node	1	2	3	4	5
1			a		
2					
3		d		b	
4		c			f
5		g	e		

Graph matrix

در شکل زیر هر سطر با دو یا چند مدخل نشانگر یک گره گزاره ای است بنابراین با اجرای محاسبات نشان داده شده در طرف راست ماتریس ارتباط، یک روش دیگر برای تعیین پیچیدگی سیکلوماتیک فراهم می آید.

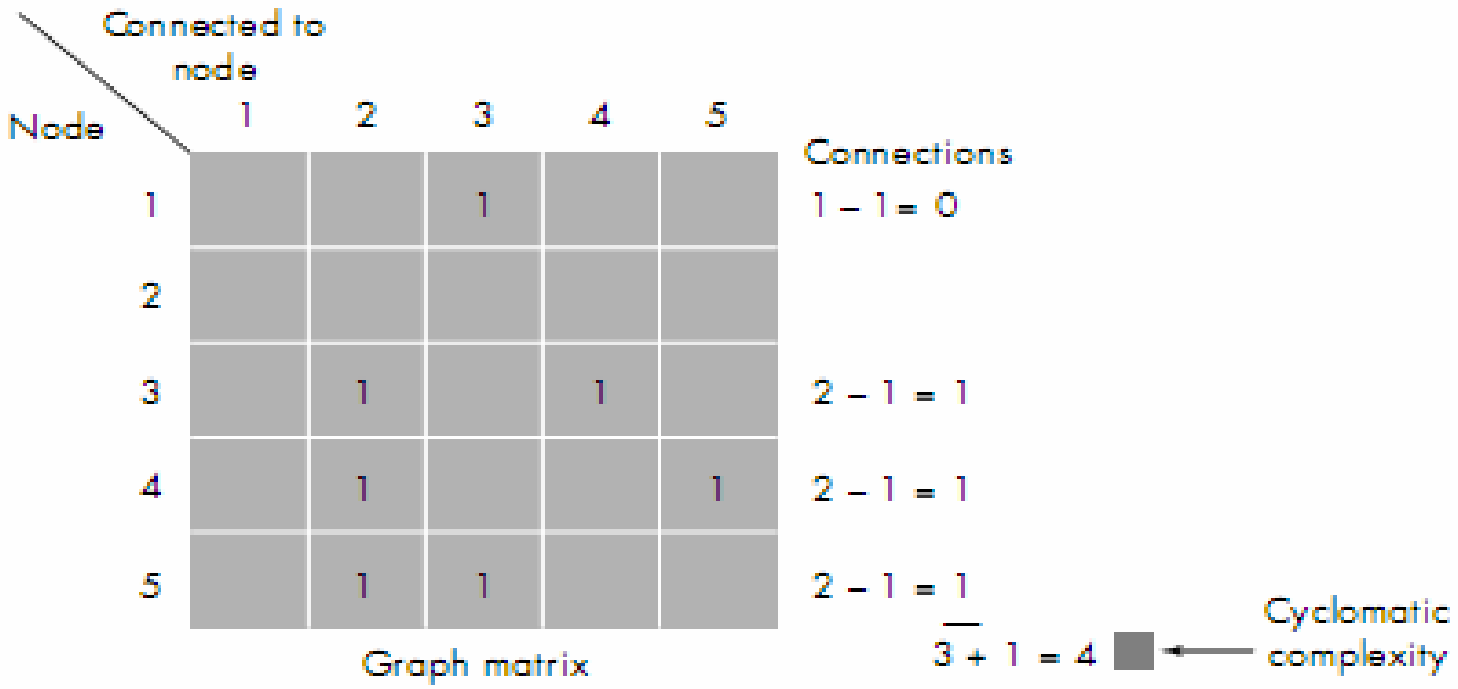


FIGURE 17.7 Connection matrix

آزمون شرط ها

آزمون شرط ها یک روش طراحی موارد آزمون است که شرطهای منطقی موجود در یک پیمانه برنامه را امتحان می کند.

هر شرط ساده، یک متغیر بولی یا یک عبارت رابطه ای است.

عبارت رابطه ای به شکل زیر است:

E_1<عملگر رابطه ای >E_2

که درآن E_1و E_2عبارت های محاسباتی و < > عملگر رابطه ای یکی از موارد "<"، ">"، "="، "≠"، "≥"، "≤"است.

شرط مرکب از دو یا چند شرط ساده، عملگرهای بولی و پرانتز تشکیل می شود. فرض می کنیم که عملرهای بولی مجاز در یک شرط مرکب شاملOR(|)، AND("&")، NOT ("¬") باشد شرط بدون عبارتهای رابطه ای را عبارت بولی می گویند.

انواع ممکن برای عناصر در یک شرط عبارتند از:

- عملگر بولی
- متغیر بولی
- یک جفت پرانتز بولی (که یک شرط ساده یا مرکب را محصور می کند)
- عملگر رابطه ای
- یک عبارت محاسباتی

اگر شرطی نادرست باشد در آن صورت حداقل یک جزء از شرط نادرست است.

انواع خطاها در یک شرط شامل موارد زیر است:

- خطای عملگر بولی(نادرست/جا افتاده/اضافی)
- خطای متغیر بولی
- خطای پرانتزهای بولی
- خطای عبارات محاسباتی

راهبردهای آزمون شرطها

- آزمون شاخه ای
- آزمون دامنه

آزمون شاخه ای

برای شرط مرکب C، شاخه های درست و نادرست C و هر شرط ساده در C باید حداقل یکبار اجرا شود.

آزمون دامنه

آزمون دامنه مستلزم به دست آوردن سه یا چهار آزمون برای یک عبارت گویا است.

آزمون دامنه (مثال)

برای یک عبارت گویا به شکل زیر: E_1< عملگر رابطه ای <E_2

سه آزمون لازم است تا مقدار E_1 بزرگتر از، مساوی با، یا کوچکتر از E_2 باشد.

اگر عملگر رابطه ای > نادرست باشد و E_1 و E_2 درست باشند در آن صورت با سه آزمون می توان اطمینان یافت که خطای عملگر رابطه ای پیدا خواهد شد. برای یافتن خطاها در E_1 و E_2 آزمونی که مقدار E_1 را بزرگتر یا کوچکتر از E_2 می کند باید بین این دو مقدار حداقل اختلاف ممکن را ایجاد کند.

برای یک عبارت بولی با n متغیر، هر 2^nآزمون ممکن لازم است انجام شود ($n>0$).

با این راهبرد می توان خطاهای عملگر بولی متغیر و پرانتزها را یافت، ولی فقط در صورتی عملی است که n کوچک باشد.

برای عبارتهای بولی، آزمونهای حساس به خطا نیز می توان انجام داد. برای هرعبارت بولی منفرد، با n متغیر بولی ($n>0$) می توان یک مجموعه آزمون با کمتر از 2^n آزمون تولید کرد به طوری که این مجموعه آزمون یافتن خطاهای عملگرهای بولی چندگانه را تضمین کند.

تای یک راهبرد برای آزمون شرطها پیشنهاد می کند که بر تکنیکهای فوق الذکر استوار است این تکنیک به اختصار BROخوانده می شود.

این روش، پیدا شدن خطاهای شاخه ای و عملگر رابطه ای را در یک شرط تضمین می کند مشروط بر اینکه همه متغیرهای بولی و عملگرهای رابطه ای فقط یک بار در شرط ظاهر شوند و متغیر مشترکی نداشته باشند.

راهبرد BRO از محدودیتهای شرطی برای شرط C استفاده می کند محدودیت شرطی برای C با n شرط ساده به صورت (D_1وD_2و.....وD_n) تعریف می شود که ($0< i\leq n$)D_i نماد مشخص کننده محدودیت روی نتیجه شرط ساده iام در شرط C است.

اگر طی اجرای C نتیجه هر شرط ساده C، محدودیت مربوطه رادر D برآورده کند گفته می شود که محدودیت شرطی D توسطC پوشش داده می شود.

برای متغیر بولی B یک محدودیت روی نتیجه B مشخص می کنیم که بیان می کند B باید درست (t) یا نادرست (f) باشد به طور مشابه در یک عبارت رابطه ای برای مشخص کردن محدودیتهایی روی نتایج عبارت از نمادهای >، =، < استفاده می شود.

به عنوان مثال شرط زیر را در نظر بگیرید:

$C_1:B_1\&B_2$

که درآن B_1و B_2 متغیرهای بولی هستند.

محدودیت شرطی C_1 به شکل (D_1,D_2) است که درآن D_1 و D_2 "t"یا "f" است. مقدار (t,f) یک محدودیت شرطی برای C_1 بوده، توسط آزمونی که مقدار B_1 را درست و مقدار B_2 را نادرست می کند پوشش داده می شود.

راهبرد آزمون BRO مستلزم آن است که محدودیتهای $\{(t,t),(f,t),(t,f)\}$ توسط اجرای C_1 پوشش داده شود.

C_2: B_1&(E_3=E_4)

B_1 یک عبارت بولی و E_3و E_4 عبارتهای محاسباتی اند.

یک محدودیت شرطی برایC_2 به شکل (D_1,D_2) است که در آن D_1، "t" یا "f" و D_2، >، = یا < است.

مجموعه محدودیت های {(t,t),(f,t),(t,f)}را که برای C_1 تعریف شد، اصلاح می کنیم.

"t"برای (E_3=E_4) به معنای "=" و f برای (E_3=E_4) به معنی ">" یا "<" است.

مجموعه محدودیتهای شرطی برای C_2عبارت است از: {(t,=), (f,=), (t,>), (t,<)}.

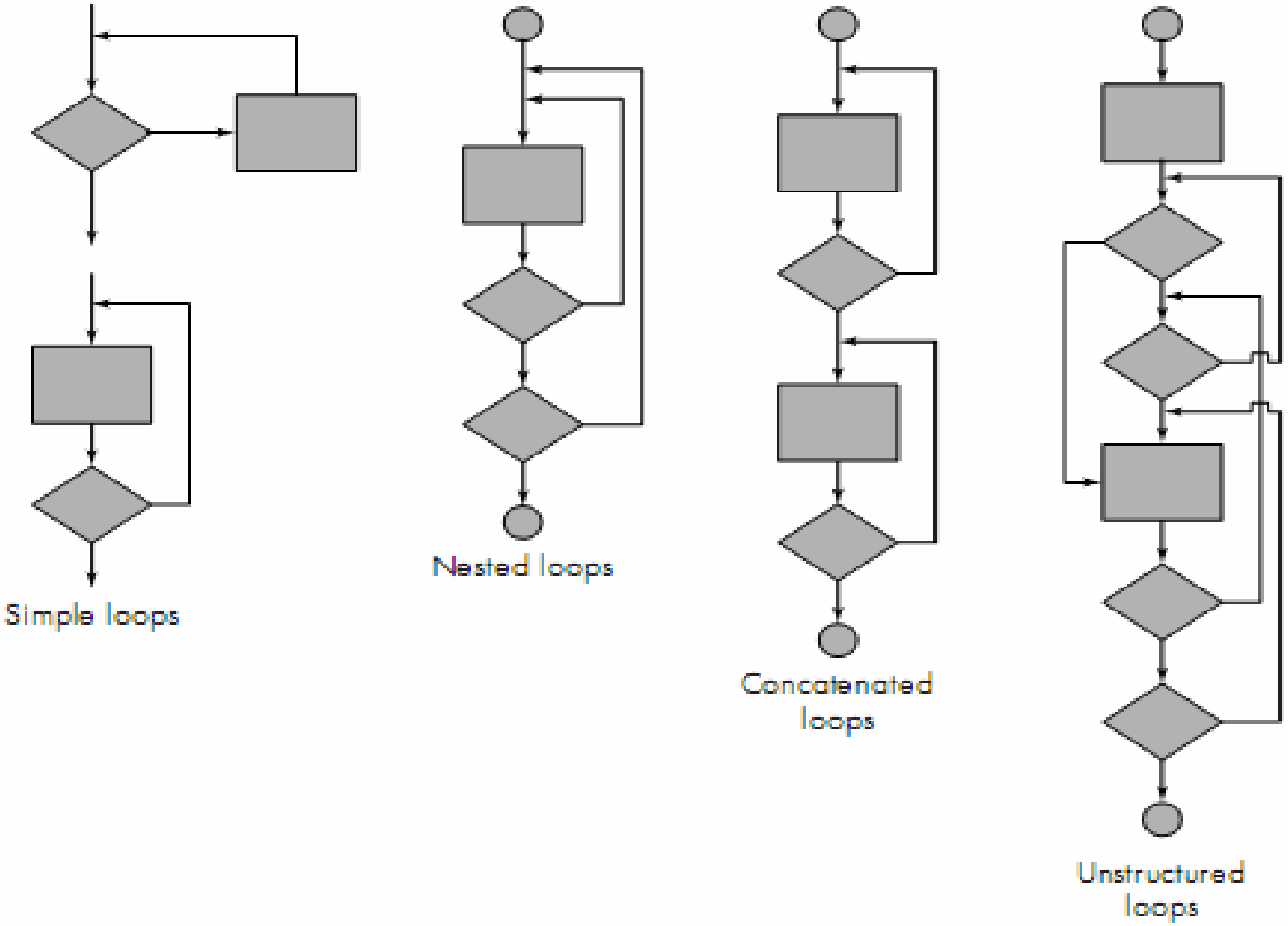
آزمون حلقه ها

آزمون حلقه ها یکی از تکنیکهای آزمون جعبه سفید است که انحصارا بر اعتبار ساختمان حلقه تکیه دارد.

چهار طبقه متفاوت از حلقه ها را می توان تعریف کرد:

- حلقه های ساده
- حلقه های تسلسلی
- حلقه های تودرتو
- حلقه های ساخت نیافته

FIGURE 17.8
Classes of loops



- **حلقه های ساده**

آزمونهای زیر را می توان درمورد یک حلقه ساده اجرا کرد، که درآن n حداکثر تعداد گذرهای مجاز از میان حلقه است.

- عدم اجرای حلقه
- فقط یک بار گذر ازحلقه
- دو بار گذر از حلقه
- mبار گذر ازحلقه که m < n است
- n-1، nو n+1 گذر از حلقه

حلقه های تودرتو

بیزر روشی پیشنهاد می کند که به کاهش دادن تعداد آزمونها کمک می کند:

- از داخلی ترین حلقه شروع کنید همه حلقه های دیگر را در حداقل مقدار قرار دهید.
- آزمونهای حلقه ساده را برای داخلی ترین حلقه اجرا کنید و درعین حال حلقه های خارجی دیگر را در حداقل مقدار پارامتر تکرارشان نگه دارید. آزمونهای دیگری برای مقادیر خارج از محدوده یا مقادیر مستثنی شده اجرا نمایید.
- به سمت بیرون حرکت کنید و همین روند را برای حلقه بعدی تکرار کنید، ولی همه حلقه های بیرونی را در حداقل مقدارشان و حلقه های داخلی را در مقادیر «معمولی» آنها قرار دهید.
- کار را تا آزمون همه حلقه ها ادامه دهید.

حلقه های تسلسلی

حلقه های تسلسلی را می توان با استفاده از روش آزمون برای حلقه های ساده آزمود با این شرط که هر یک از حلقه ها مستقل از دیگری باشد ولی اگر دو حلقه تسلسلی نباشند و شمارنده حلقه ۱ به عنوان مقدار اولیه ای برای حلقه ۲ استفاده شود در آن صورت حلقه ها مستقل از یکدیگر نیستند. در این حالت روش به کار رفته در حلقه های تودرتو را توصیه می کنیم.

حلقه های ساخت نیافته

درصورت امکان این نوع از حلقه ها را باید طوری دوباره طراحی کرد که منعکس کننده کاربرد ساختمانهای برنامه نویسی ساخت یافته باشند.

آزمون جعبه سیاه

آزمون جعبه برای کشف خطاهای موجود در این گروهها بکار می رود:

۱.عملکرد نادرست یا جا افتاده

۲.خطاهای واسط

۳.خطاهای موجود در ساختمان داده ها یا دستیابی به بانک اطلاعاتی خارجی

۴.خطاهای رفتاری یا کارایی

۵.خطاهای مقدار دهی اولیه یا خاتمه برنامه.

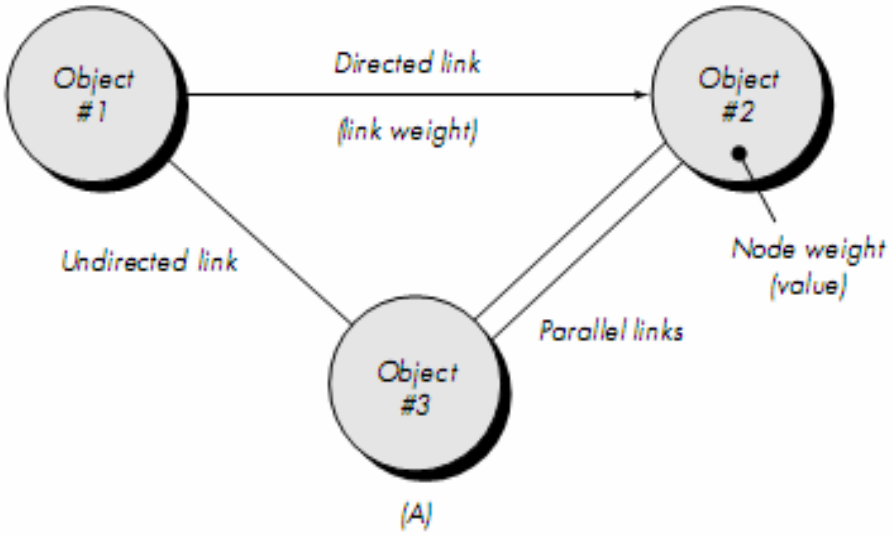
آزمون جعبه سفید از اوایل فرایند آزمون اجرا می شود.

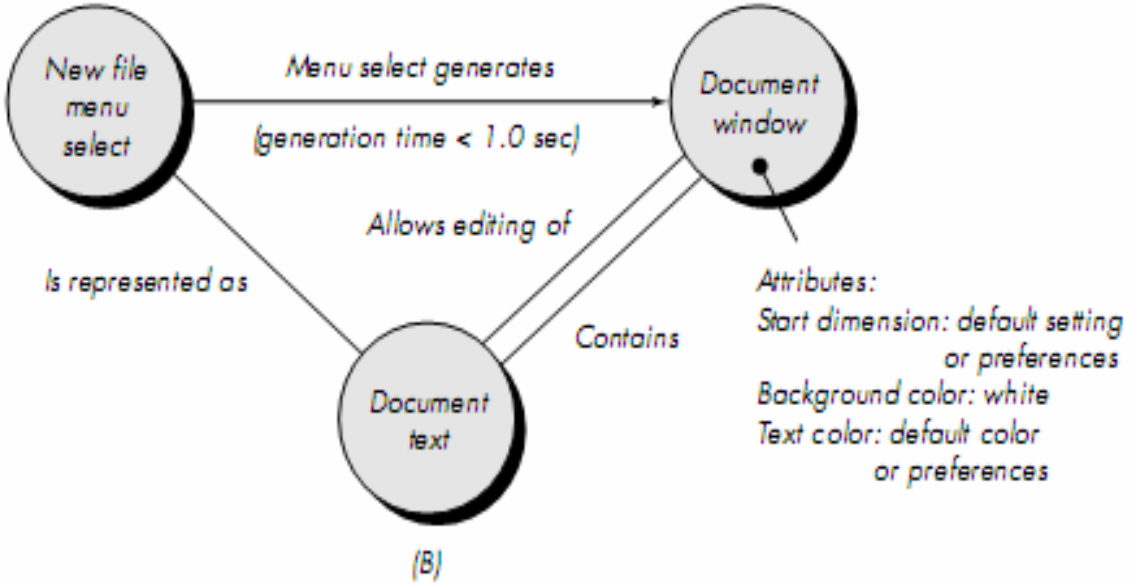
آزمون جعبه سیاه در مراحل آخر آزمون به کار می رود.

روشهای آزمون مبتنی بر گراف

- شناخت اشیایی که در نرم افزار مدلسازی می شوند
- روابط میان این اشیا
- تعیین تعدادی آزمون که ثابت کنند همه اشیاء دارای رابطه مورد انتظار با یکدیگر هستند.

FIGURE 17.9
(A) Graph notation
(B) Simple example





افراز هم ارزی

دامنه ورودی یک برنامه را به طبقاتی از داده ها تقسیم می کند و موارد آزمون را می توان از روی آن به دست آورد. یک مورد آزمون ایده آل یک طبقه از خطاها (مثلا پردازش نادرست همه داده های کاراکتری) را کشف می کند که در غیر این صورت قبل از مشاهده یک خطای عمومی، موارد متعددی باید اجرا شوند. افراز هم ارزی مورد آزمونی را تعریف می کند که طبقاتی از خطاها را کشف می کند و در نتیجه از تعداد کل موارد آزمون می کاهد.

- طبقه هم ارزی نشانگر مجموعه ای از حالتهای معتبر و نامعتبر برای شرایط ورودی است. هر شرط ورودی معمولا با یک مقدار عددی بازه ای از مقادیر مجموعه ای از مقادیر مرتبط با یک شرط بولی است طبقات هم ارزی رامی توان طبق دستورالعملهای زیر تعریف نمود:
- اگر شرط ورودی، بازه ای رامشخص کند یک طبقه هم ارزی معتبر و دو طبقه هم ارزی نامعتبر تعریف می شوند.
- اگر شرط ورودی، نیازمند مقداری مشخص باشد یک طبقه هم ارزی معتبر، دو طبقه هم ارزی نامعتبر تعریف می شوند.
- اگر شرط ورودی، نیازمند عضوی از یک مجموعه باشد یک طبقه هم ارزی معتبر، یک طبقه هم ارزی نامعتبر تعریف می شوند.
- اگر شرط ورودی، بولی باشد یک طبقه معتبر و یک طبقه نامعتبر تعریف می شود.

- به عنوان مثال داده هایی را در نظر بگیرید که به عنوان بخشی از یک برنامه بانکداری نگهداری می شوند. کاربر می تواند با استفاده از کامپیوتر شخصی خودش به بانک دستیابی داشته باشد یک کلمه عبور شش رقمی را وارد کند و سپس عملیات بانکی را با تایپ فرمانها انجام دهد. طی دنباله ثبت رویدادها، نرم افزار مربوطه، داده ها را به شکل زیر می پذیرد:
- کدناحیه –خالی یا ستاره سه رقمی
- پیشوند –شماره ای سه رقمی که با ۰یا ۱شروع نمی شود.
- پسوند–شماره ای چهار رقمی
- کلمه عبور–رشته حرفی –عددی شش رقمی
- فرمانها –(چک کردن)، (واریز کردن)، (پرداخت قبوض)و....

- شرایط ورودی همراه با هر عنصر داده ای در برنامه بانکداری را می توان به صورت زیر مشخص کرد:
- کدناحیه: شرط ورودی، بولی –کد ناحیه ممکن است وجود داشته باشد یا وجود نداشته باشد؛
- شرط ورودی، بازه –مقادیر تعریف شده بین ۲۰۰و ۹۹۹ با استثناهای مشخص؛
- پیشوند: شرط ورودی، بازه –مقدار مشخص شده بزرگتر از ۲۰۰ بدون ارقام صفر؛
- شرط ورودی، مقدار –طول ۴رقم؛
- کلمه عبور: شرط ورودی، بولی-کلمه عبور ممکن است وجود داشته باشد یا وجود نداشته باشد؛
- شرط ورودی، مقدار-رشته شش کاراکتری
- فرمان: شرط ورودی، مجموعه – حاوی فرمانهای فوق الذکر

تحلیل مقادیر مرزی

تعداد خطاهای موجود در مرزهای دامنه ورودی نسبت به مقادیر مرکزی دامنه بیشتر است به همین دلیل تکنیکی موسوم به تحلیل مقادیر مرزی (BVA) توسعه یافته است تحلیل مقادیر مرزی به موارد آزمونی منجر می شود که مقادیر مرزی را امتحان می کنند.

- دستورالعملهای BVAاز بسیاری جهات مشابه با دستور العملهای ذکر شده برای افراز هم ارزی است:
- اگر یک شرط ورودی بازه ای محصور به مقادیر aو bرا مشخص کند موارد آزمونی با مقادیر a وb باید طراحی کرد که به ترتیب درست در پایین و بالای مقدار a وb باشند.
- اگر یک شرط ورودی چند مقدار را مشخص می کند باید موارد آزمونی توسعه یابند که اعداد بیشینه و کمینه را امتحان کنند مقادیری که درست در بالای بیشینه و درست در پایین کمینه قرار دارند نیز باید آزمایش شوند.

دستورالعملهای BVA:

•دستورالعملهای ۱و۲ باید برای شرطهای خروجی نیز اعمال شوند فرض کنید که یک جدول دما در مقابل فشار به عنوان خروجی یک برنامه تحلیل مهندسی مورد نیاز است موارد آزمون باید طوری طراحی شوند که یک گزارش خروجی ایجاد کنند که حداکثر (و حداقل) تعداد مدخلهای ممکن برای جدول را مشخص کند.

•اگر ساختمان داده های داخلی برنامه دارای مرزهای معین باشند (مثلا آرایه ای دارای ۱۰۰مدخل باشد)حتما باید یک مورد آزمون برای امتحان کردن ساختمان داده طراحی شود.

آزمون مقایسه ای

شرایط معینی وجود دارد (مثل کنترل هواپیما و کنترل نیروگاههای هسته ای) که در آنها قابلیت اطمینان نرم افزار اهمیت مطلق دارد. در چنین کاربردهایی غالبا از نرم افزارها و سخت افزارهای اضافی برای به حداقل رساندن امکان خطا استفاده می شود. هنگامی که نرم افزارهای اضافی توسعه می یابد تیمهای مهندسی نرم افزار جداگانه ای، نسخه های مستقلی از برنامه های کاربردی را با استفاده از همان مشخصه ها توسعه می دهند. در چنین شرایطی هر نسخه را می توان با همان داده های آزمایشی مورد آزمون قرار داد تا اطمینان حاصل شود که همگی خروجی یکسان تولید می کنند سپس همه نسخه ها به طور موازی با مقایسه بلادرنگ نتا یج اجرا می شوند تا از سازگاری آنها اطمینان حاصل شود.