فصل ۱۳ اصول و مفاهیم طراحی

مفاهیم کلیدی (مرتب بر حروف الفبا)

ابداعات و ابتکارات طراحی ، استقلال کارکردی ، پالایش ، پنهان سازی اطلاعات ، پیمانه سازی ، پیوستگی ، تجرید و انتزاع ، تجزیه و تفکیک ، چسبندگی ، ساختار داده ها ، مفاهیم طراحی ، معماری ، معیارهای کیفیت

KEY CONCEPTS

Abstraction , architecture , coupling , cohesion , data structure , design concepts , design heuristics , design principles , functional independence , information hiding , modularity , partitioning , quality criteria , refinement

نگاه اجمالی

طراحی چیست؟ طراحی بازنمایی مهندسی و هدفمند چیزی است که قرار است ساخته شود. طراحی را می‌توان مطابق با خواسته‌های مشتری پیش برد و در عین حال بر اساس مجموعه معیارهای از پیش تعریف شدهٔ طراحی "خوب"، کیفیت آن را ارزیابی کرد. در زمینهٔ مهندسی نرم‌افزار، طراحی بر چهار کانون اصلی متمرکز است: داده‌ها، معماری، رابط‌ها و اجزاء. مفاهیم و اصول مورد بحث در این فصل به هر چهار حوزه مربوط می‌شوند.

طراحی کار کیست؟ مهندسین نرم‌افزار، طراحی سیستم‌های کامپیوتری را برعهده دارند. اما مهارت‌های لازم در هر یک از سطوح کار طراحی متفاوتند. در سطح داده‌ها و معماری، طراحی بر الگوهای مرتبط با برنامهٔ کاربردی متمرکز می‌گردد. در سطح رابط، کار پژوهشی انسانی اغلب نحوهٔ طراحی را تعیین می‌کند. در سطح اجزاء، یک "رهیافت برنامه‌سازی" ما را به سمت داده‌های مؤثر و طراحی‌های رویه‌ای هدایت می‌کند.

دلیل اهمیت طراحی چیست؟ شما هیچ‌گاه تلاش نمی‌کنید خانه‌ای را بدون طرح و نقشه بسازید، این‌طور نیست؟ در این‌صورت امکان بروز اشتباه و خطا، نامتناسب بودن نقشهٔ کف، قرار گرفتن پنجره‌ها و درها در جای نادرست و هرگونه خراب‌کاری و افتضاح دیگر وجود دارد. نرم‌افزار کامپیوتری به مراتب پیچیده‌تر از یک خانه است. از این‌رو ضرورت نقشه یا طراحی احساس می‌شود.

طراحی کدامند؟ طراحی با مدل ضرورت‌ها و نیازها شروع می‌شود. ما سعی می‌کنیم این مدل را در هر چهار سطح طراحی یعنی: ساختار داده‌ها، معماری سیستم، نمایش رابط و سطح اجزا به اجرا درآوریم. در هر یک از فعالیت‌های طراحی، مفاهیم و اصول اساسی که به کیفیت بالا منجر می‌شوند را رعایت می‌کنیم.

حاصل کار چیست؟ در نهایت مشخصهٔ طراحی ایجاد می‌گردد. این مشخصه شامل مدل‌های طراحی است که داده‌ها، معماری، رابطها و اجزا را توصیف می‌کنند و هر یک محصول کاری روند طراحی محسوب می‌شوند.

تضمین درستی طراحی چگونه ممکن است؟ در هر مرحله، محصولات کار طراحی نرم‌افزار از جهت وضوح، صحت، تکمیل و هماهنگی با نیازمندیها و با یکدیگر مورد بررسی قرار می‌گیرند.

هدف طراح، ایجاد یک مدل یا نمایش موجودیتی است که بعداً ساخته خواهد شد. روند توسعه و تکمیل مدل طراحی توسط بلیدی [BEL81][1] توصیف می‌گردد:

در هر روند طراحی، دو مرحله اصلی وجود دارد، گوناگونی و یکپارچگی. گوناگونی یعنی فراگیری[2] *مجموعه گزینه‌ها و مواد خام طراحی، اجزاء، طرحهای اجزا، و اطلاعاتی که همگی در کاتالوگ‌ها، کتاب‌های درسی و ذهن وجود دارند. در مرحلهٔ یکپارچگی، طراح برای تأمین اهداف طراحی، آن‌طور که در توضیح نیازمندی‌ها بیان شده و مورد نظر مشتری است، عناصر مناسب را از این مجموعه انتخاب و ترکیب می‌کند. مرحلهٔ دوم حذف*[3] *تدریجی همهٔ عناصر اما ترکیب‌بندی خاص آنها و در نتیجه ایجاد محصول نهایی می‌باشد.*

گوناگونی و یکپارچگی، ترکیبی است از ادراک و قضاوت مبتنی بر تجربه در ساختن موجودیت های مشابه، یک مجموعهٔ اصول و یا ذهنیاتی که راهنمای سیر تکاملی مدل هستند، مجموعه معیارهایی ارزیابی کیفی را ممکن می‌سازند و روند تکراری که در نهایت منجر به نمایش طراحی نهایی می‌گردد.

طراحی نرم‌افزار مانند شیوه‌های طراحی مهندسی در سایر رشته‌ها، در قالب روش‌های جدید، تحلیل بهتر و درک عمیق‌تر، پیوسته در حال تغییر است.

روش‌های طراحی نرم‌افزار از عمق، انعطاف‌پذیری و ماهیت، از ارتباط اندکی با شیوه‌های کلاسیک طراحی مهندسی برخوردارند. با این وجود شیوه‌هایی برای طراحی نرم‌افزار وجود دارند؛ معیارهایی برای کیفیت طراحی در دسترسند و می‌توان از نشان‌گذاری طراحی استفاده کرد. در این فصل، مفاهیم و اصول اساسی قابل اجرا در تمامی طراحی‌های نرم‌افزاری را شناسایی خواهیم کرد. فصول ۱۴، ۱۵، ۱۶و۲۲،

---

1. Belay, L.
2. Acquisition
3. el[illegible]

مجموعهٔ گسترده‌ای از روش‌های طراحی نرم‌افزار را که در طراحی معماری، رابط و سطح اجزاء، کاربرد دارند مورد بررسی و مطالعه قرار می‌دهند.

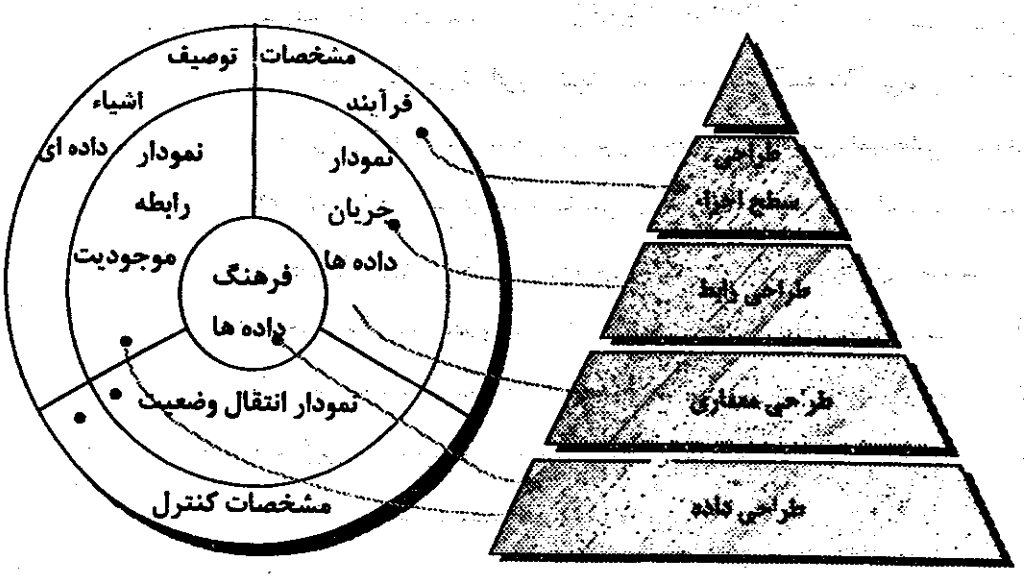
۱۳-۱ طراحی نرم افزار و مهندسی نرم افزار

نقل قول

مهمترین اعجاز مهندسی نرم افزار انتقال از تحلیل به طراحی و از طراحی به برنامه نویسی است.
ریچارد داو

طراحی نرم‌افزار جزء بخش اصلی و فنی مهندسی نرم‌افزار بوده و بدون توجه به مدل به‌کار رفتهٔ فرآیند نرم‌افزار، اعمال می‌گردد. در آغاز و پس از تحلیل و تعیین نیازمندی‌های نرم‌افزار، طراحی نرم‌افزار در بین سه فعالیت فنی یعنی طراحی، تولید برنامه، و آزمون، که در ساخت و تأیید نرم‌افزار ضرورت دارند، اولین آنها به‌شمار می‌رود. هر یک از این فعالیت‌ها، اطلاعات را به‌گونه‌ای تغییر می‌دهد که در نهایت به نرم‌افزار معتبر کامپیوتری منجر می‌گردد.

هر یک از عناصر مدل تحلیلی (فصل ۱۲)، اطلاعات لازم را در ایجاد چهار مدل طراحی به‌منظور تعیین کامل طراحی فراهم می‌آورند. گردش اطلاعات حین طراحی نرم‌افزار، در شکل ۱۳-۱ به نمایش درآمده است. نیازمندی‌های نرم‌افزاری که با مدل‌های داده‌ها، کاربردی و رفتاری نشان داده شده‌اند، کار طراحی را پیش می‌برند. با کاربرد یکی از چند شیوهٔ طراحی (که در فصل‌های بعدی مورد بحث قرار گرفته)، وظیفه طراحی، طراحی داده‌ها، معماری، طراحی رابط و اجزاء را ایجاد می‌کند.



شکل ۱۳ـ۱ برگردان مدل تحلیلی به طراحی نرم

طراحی داده‌ها[1]، مدل اطلاعاتی تولید شده در طول تحلیل را به ساختارهای داده‌ای لازم در اجرای نرم‌افزار تبدیل می‌کند. اشیاء و روابط تعیین شدهٔ داده‌ها در نمودار موجودیت - رابطه 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. طراحی جزیی‌تر داده‌ها همراه با طراحی هر یک از اجزای نرم‌افزار صورت می‌گیرد.

طراحی معماری[2]، رابطهٔ بین عناصر اصلی ساختاری نرم‌افزار را تعیین می‌کند، یعنی رابطهٔ "الگوهای طراحی" به‌کار رفته در تحقق نیازمندی‌های تعیین شدهٔ سیستم و محدودیت‌های مؤثر بر شیوهٔ اجرای الگوهای طراحی معماری [SHA96][3] نمایش طراحی معماری یا ـ چارچوب یک سیستم کامپیوتری ـ حاصل تعیین سیستم، مدل تحلیلی و ارتباط سیستم‌های فرعی تعیین شده در مدل تحلیلی است.

طراحی رابط[4]، توصیف‌کنندهٔ نحوهٔ ارتباط نرم‌افزار در محدودهٔ خود، با سیستم‌هایی که با آن عمل‌پذیری درونی دارند و افرادی است که آن را به‌کار می‌برند. یک رابط بر گردش اطلاعات (مثل داده‌ها و یا کنترل) و نوعی خاصی از رفتار دلالت دارد. بنابراین نمودارهای جریان داده‌ها و کنترل، اکثر اطلاعات لازم برای طراحی رابط را ارائه می‌کنند.

طراحی اجزاء[5]، عناصر ساختاری معماری نرم‌افزار را به توصیف رویه‌ای اجزاء نرم‌افزاری تبدیل می‌کند. اطلاعات به‌دست آمده از STD, CSPEC, PSPEC پایه و اساس طراحی اجزاء به‌شمار می‌روند.

اهمیت طراحی نرم‌افزار را تنها با یک کلمه یعنی "کیفیت"[6] می‌توان بیان کرد. طراحی روندی است که طی آن کیفیت در مهندسی نرم‌افزار، بهبود می‌یابد. طراحی، نمونه‌هایی از نرم‌افزار را که از لحاظ کیفی قابل ارزیابی هستند، در اختیار ما قرار می‌دهد. طراحی تنها راهی است که به‌واسطهٔ آن می‌توانیم به‌طور صحیح نیازمندیها و خواسته‌های مشتری را به یک محصول نرم‌افزاری یا سیستم تکمیل شده تبدیل کنیم. طراحی نرم‌افزار، شالوده و اساس کل مهندسی نرم‌افزار و مراحل بعدی پشتیبانی از آن است. بدون طراحی، خطر ساخت یک سیستم ناپایدار وجود دارد، یعنی سیستمی که پذیرای تغییرات کوچک و جزیی نخواهد بود. آزمون آن دشوار بوده و ارزیابی کیفی آن تا مراحل نهایی فرآیند نرم‌افزاری و با وجود محدودیت زمانی و پس از صرف هزینهٔ بسیار، ممکن نخواهد بود.

1.data design

2.architectural design

3.Shaw, M. and D. Garlan

4.interface design

5.component_level design

6.quality

۱۳-۲ فرآیند طراحی

نقل قول

برای رسیدن به یک طراحی خوب افراد باید به این مهم بیاندیشند که چگونه فعالیتهای فاز طراحی را به خوبی اداره کنند. کاتارین وایت هد

طراحی نرم‌افزار یک فرآیند تکراری است که به‌موجب آن نیازمندیها و ضرورتها برای ساخت نرم‌افزار، تبدیل به یک "طرح یا نقشه" می‌شوند. در ابتدا، طرح یک دید کلی از نرم‌افزار را نشان می‌دهد. یعنی آن‌که طراحی در سطح بالایی از انتزاع ارائه می‌شود ـ سطحی که ردیابی مستقیم آن به هدف خاص سیستم و اطلاعات جزیی‌تر، نیازمندیهای کاربردی و رفتاری منتهی می‌شود. با انجام تکرار در طراحی، پالایش بعدی، به‌نمایش طراحی در سطوح بسیار پایین‌تر انتزاع منجر می‌گردد. این سطوح نیز برای دستیابی به نیازمندیها قابل ردیابی است، اما این نوع ارتباط ظریف‌تر می‌باشد.

۱۳-۲-۱ طراحی و کیفیت نرم افزار

در طول فرآیند طراحی، کیفیت طراحی در حال تکمیل با مجموعه‌ای از بررسی‌های فنی رسمی یا بررسی‌های دقیق طراحی مورد بحث در فصل ۸، ارزیابی می‌گردد. مک‌گلاولین [MCG91][1] سه مشخصه‌ای را که به‌عنوان راهنمای ارزیابی یک طراحی خوب به‌کار می‌آیند، پیشنهاد می‌کند:

پرسش

آیا رهنمود فراگیری برای هدایت ما به یک طراحی خوب وجود دارد؟

- طراحی باید همهٔ ضرورت‌های آشکار موجود در مدل تحلیل را تحقق بخشیده و با تمامی خواسته‌ها و نیازهای ضمنی و مطلوب مشتری سازگار باشد.
- طراحی باید برای کسانی‌که برنامه‌نویسی می‌کنند و نیز اشخاصی که نرم‌افزار را آزمون نموده و بعداً آن را پشتیبانی می‌کنند، راهنمایی خوانا و قابل درک باشد.
- طراحی باید تصویر کاملی از نرم‌افزار را ارائه کرده و حوزه‌های داده‌ای، کاربردی و رفتاری را از دید پیاده‌سازی مورد توجه قرار دهد.

به‌منظور ارزیابی کیفیت یک طراحی، باید معیارها و ضوابط فنی یک طراحی خوب را تعیین کنیم. در بخش‌های بعدی این فصل، معیارهای کیفیت طراحی را به‌طور جزیی و مفصل مورد بحث قرار خواهیم داد. در حال حاضر، رهنمودهای زیر را معرفی می‌کنیم:

نقل قول

دو راه برای طراحی نرم افزار در پیش روست: یکی آنکه آن را آنگونه ساده طراحی کنیم که آشکارا هیچ نقصی نداشته باشد و دیگر آنکه آنطور پیچیده باشد که هیچ نقصی آشکار نباشد و البته مسیر نخست بسیار سخت است.
سی.آ.ارهوار

۱- طراحی باید یک ساختار معماری را 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 بدین‌ترتیب اجرا و آزمون را تسهیل کند.

۲- طراحی باید پیمانه ای (ماژولار) باشد یعنی نرم‌افزار باید به‌طور منطقی به اجزایی تقسیم شود که اعمال اصلی و فرعی خاصی را انجام دهند.

۳- طراحی بایستی نمایش‌های مجزایی از داده‌ها، معماری، رابط‌ها و اجزاء (پیمانه ها) را در برداشته باشد.

۴- طراحی باید به ساختارهای داده‌ای منجر شود که برای پیاده‌سازی اشیاء مناسب بوده و از الگوهای قابل تشخیص داده‌ها ناشی می‌شوند.

۵- طراحی باید به اجزایی منتهی گردد که خصوصیات مستقل کاربردی را نمایش می‌دهد.

۶- طراحی باید به رابط‌هایی ختم شود که از پیچیدگی روابط بین پیمانه ها و محیط خارجی می‌کاهند.

۷- طراحی باید حاصل کاربرد یک شیوهٔ قابل تکرار با استفاده اطلاعات به‌دست آمده در طول تحلیل نیازمندیهای نرم‌افزاری باشد.

این معیارها، تصادفی به‌دست نمی‌آیند. فرآیند طراحی نرم‌افزار، به‌واسطهٔ کاربرد اصول اساسی طراحی، روش منظم و مرور کامل، طراحی خوب را مورد تأیید و حمایت قرار می‌دهد.

۱۳-۲-۲ سیر تکاملی طراحی نرم افزار

تکامل طراحی نرم‌افزار یک فرآیند پیوسته است که در چهار دههٔ گذشته گسترش یافته است. اولین کارهای طراحی بر معیارهایی جهت توسعه برنامه‌های پیمانه ای[DEN73][2] و روش‌های اصلاح ساختارهای نرم‌افزاری به شیوهٔ بالا به پایین یا (کل به جزء) متمرکز بودند. [WIR71][3]. جنبه‌های عملی تعریف طراحی به نظریه‌ای به‌نام "برنامه نویسی ساختیافته"[4] تحول یافت [MIL72] و [DAH71][6]

کارهای بعدی، روش‌هایی را برای انتقال گردش داده‌ها[STE74][7] یا ساختار داده‌ها [WAR74][8] و [JAC75][9] در توصیف طراحی، پیشنهاد دارند. روش‌های جدیدتر طراحی، رهیافت شیءگرا را در اشتقاق طراحی پیشنهاد می‌کنند. امروزه تأکید در طراحی نرم‌افزار بر معماری نرم‌افزار[SHA96][10] [BAS98][11] و الگوهای طراحی است که در انجام معماری‌های نرم‌افزاری کاربرد دارند. [BRO98][12] و [BUS96][13] و

پژوهش

کدام ویژگیها در تمام شیوه های طراحی مشترکند؟

1 McGlaughlin, R.
2.Dennis, J
3.Wirth, N.
4:structured programming
5.Mills, H.D.
6.Dahl. O., E. Dijkstra, and C. Hoare
7.Stevens, W., G. Myers, and L. Constantine
8.Warnier,J.
9.Jackson, M.A.
10.Shaw, M. and D. Garlan
11.Bass, L., P. Clements, and R. Kazman
12.Brown W J et al.
13 Buschmann, F. et al.

[GAM95][1] بدون توجه به شیوه طراحی به‌کار رفته، مهندس نرم‌افزار باید مجموعه‌ای از اصول اساسی و مفاهیم اصلی را در طراحی داده‌ها، معماری، رابط و اجزاء اعمال کند. این اصول و مفاهیم در بخش‌های بعدی مورد توجه قرار می‌گیرند.

۱۳-۳ اصول طراحی

طراحی نرم‌افزار، هم یک فرآیند است و هم یک مدل. فرآیند[2] طراحی سلسله اعمالی است که به طراح امکان می‌دهند تا تمامی جنبه‌های نرم‌افزار تحت ساخت را توصیف کند. گرچه باید یادآور شد که فرآیند طراحی صرفاً یک کتاب آشپزی نیست، بلکه مهارت خلاقانه، تجارب نوشته و شم ساخت یک نرم‌افزار خوب و تعهد کلی در قبال کیفیت، از عوامل مهم موفقیت در طراحی خوب و قابل قبول به‌شمار می‌رود.

مدل طراحی[3]، معادل طرح‌ها یا نقشه‌های یک معمار برای خانه است. این مدل با نمایش کل نهاد در دست ساخت (مثل تصویر سه‌بعدی خانه) شروع شده و به‌تدریج آن را برای راهنمایی در ساخت هر یک از جزییات (مثل طرح لوله‌کشی) پالایش می‌کند. مدل طراحی ایجاد شده برای نرم‌افزار نیز به‌همین ترتیب چشم‌اندازهای متفاوت و تنوعی از نرم‌افزار کامپیوتری ارائه می‌دهد.

اصول اولیه طراحی به مهندس نرم‌افزار امکان می‌دهند تا فرآیند طراحی را پیش ببرد. داویس [DAV95][4] مجموعه اصولی[5] را برای طراحی نرم‌افزار پیشنهاد می‌کند که این اصول در لیست زیر تنظیم شده و ارائه می‌شوند:

- **باریک بینی نباید در فرآیند طراحی وجود داشته باشد.** یک طراح خوب بایستی رهیافت‌های دیگر را در نظر داشته و هر یک را براساس ضرورت‌های مسئله، منابع موجود برای انجام کار و مفاهیم ارائه شده طراحی در بخش ۱۳-۴ مورد قضاوت قرار دهد.
- **طراحی باید قابل ردیابی به مدل تحلیلی باشد.** از آن‌جا که هر یک از عناصر مدل طراحی اغلب قابل ردیابی به ضروریات چندگانه است، بنابراین وجود روشی برای پی‌گیری چگونگی رفع نیازمندیها در مدل طراحی لازم است.
- **طراحی نباید دوباره‌کاری باشد.** سیستم‌ها با استفاده از مجموعه الگوهای طراحی ساخته می‌شوند که احتمالاً با خیلی از آنها قبلاً هم برخورد داشته‌اند. این الگوها باید همواره به‌عنوان شق دیگر

سازگاری و یکنواختی طراحی، در ساخت سیستم های بزرگ امری خطیر است. برای این مهم، تیمهای نرم افزاری باید پیش از شروع به کار، مجموعه ای از قواعد طراحی را تعیین نموده، به آن وفادار باشند.

---

1. Gamma, E. et al.
2. Design process
3. Design Model
4. Davis, A.

۵. در اینجا تنها زیرمجموعه کوچکی از اصول طراحی دیویس آورده شده است. برای اطلاعات بیشتر به [DAV95] مراجعه کنید.

دوباره‌کاری انتخاب گردند. فرصت، کوتاه و منابع محدودند! زمان طراحی باید صرف ارائه نظرات واقعاً جدید و یکپارچه کردن الگوهای از قبل موجود شود.

- **طراحی باید فاصلهٔ عقلانی بین نرم‌افزار و مسئله موجود در جهان واقعی را به حداقل برساند.** یعنی، ساختار طراحی نرم‌افزار باید (در صورت امکان) ساختار میدان مسئله را تقلید نماید.
- **طراحی باید از یکنواختی و یکپارچگی برخوردار باشد.** اگر این‌طور به‌نظر برسد که توسعهٔ کل کار برعهدهٔ یک شخص بوده، در این‌صورت طراحی یکسان و یکنواخت است. قبل از شروع کار طراحی، قوانین، سبک و قالب باید برای تیم طراحی تعریف و تعیین گردد. توجه و دقت در تعیین رابطهای بین اجزای طراحی، یکپارچگی طراحی را به‌دنبال خواهد داشت.
- **ساختار طراحی باید پذیرای تغییر باشد.** مفاهیم مورد بحث طراحی در بخش بعدی، باعث تطابق آن با این اصل می‌باشند.
- **ساختار طراحی حتی در صورت مواجعه با داده‌های غیرعادی، رویدادها یا شرایط کاری، باید به آرامی از کار بایستد.** یک نرم‌افزار خوب طراحی شده نباید هرگز ناگهان متوقف گردد. بلکه باید طراحی آن به‌گونه‌ای باشد که با شرایط غیرعادی سازگار بوده و اگر قرار شد پردازش را پایان دهد این کار را به‌طور ملایم انجام دهد.
- **طراحی به‌معنای برنامه‌نویسی نیست و برنامه‌نویسی نیز معادل طراحی نمی‌باشد.** حتی پس از ایجاد طراحی‌های جزیی رویه‌ای برای اجزای برنامه، سطح انتزاع مدل طراحی بالاتر از برنامهٔ منبع است. تنها تصمیمات طراحی اتخاذ شده در سطح برنامه‌نویسی، جزییات ظریف پیاده‌سازی را که موجب برنامه نویسی طراحی رویه‌ای می‌شوند را مطرح می‌کند.
- **طراحی باید ضمن شکل‌گیری، از نظر کیفی مورد ارزیابی قرار گیرد نه بعد از اتمام.** مجموعه‌ای از مفاهیم طراحی (بخش ۱۳-۴) و اقدامات طراحی (فصل‌های ۱۹ و ۲۴) برای کمک بر طراح جهت ارزیابی کیفی، در دسترس می‌باشند.
- **طراحی باید به‌منظور به حداقل رساندن خطاهای مفهومی (معنایی) مرور و بررسی شود.** گاه هنگام بررسی و مرور طراحی، تمایل به تأکید روی جزییات وجود دارد. یعنی در جنگل فقط به‌دنبال درخت بودن. تیم طراح بایستی قبل از نگرانی دربارهٔ نحوه مدل طراحی، تضمین کند که عناصر اصلی مفهومی در طراحی (حذف و از قلم افتادگی، ابهام و ناسازگاری) مورد توجه قرار گرفته و رفع و رجوع گشته‌اند.

رهنمودهایی در خصوص طراحی رابط گرافیکی کاربر (GUI) در فصل ۱۶ ارائه شده اند.

با کاربرد به‌جا و مناسب اصول طراحی توصیف شدهٔ فوق‌الذکر، مهندس نرم‌افزار طراحی را به‌وجود می‌آورد که عوامل کیفی داخلی و خارجی را نشان می‌دهد. [MEY88][1]. عوامل بیرونی کیفیت[1]، آن‌دسته

1.Meyer, B.

از خصوصیات نرم‌افزاری هستند که به راحتی توسط کاربران قابل مشاهده است (مثل سرعت، اعتبار، صحت و قابلیت کارآیی).[۲] عوامل داخلی کیفیت[۳] برای مهندسین نرم‌افزار حائز اهمیت است و از دید فنی به طراحی با کیفیت بالا منجر می‌شوند. برای دستیابی به عوامل داخلی کیفی، طراح باید مفاهیم اصلی طراحی را درک کند.

۱۳-۴ مفاهیم طراحی

مجموعه‌ای از مفاهیم اساسی طراحی نرم‌افزار طی چهار دههٔ گذشته گسترش یافته است. گرچه میزان اهمیت هر مفهوم در طول سال‌ها متغیر بوده، هر یک اعتبار و ماندگاری خود را اثبات کرده‌اند. این مفاهیم را در اختیار طراح نرم‌افزار می‌گذارند که براساس آن روش‌های پیچیده‌تر طراحی قابل کاربرد است و هر یک از آنها به مهندس نرم‌افزار کمک می‌کنند تا به سؤالات زیر پاسخ دهند:

- برای تقسیم‌بندی نرم‌افزار به اجزای جداگانه، چه معیارهایی قابل استفاده‌اند؟
- چگونه عمل یا جزییات ساختار داده‌ای، از نمایش مفهومی و ذهنی نرم‌افزار مجزا می‌گردد؟
- آیا معیارهای یکسانی برای تعریف کیفیت فنی طراحی نرم‌افزار وجود دارند؟

مایکل جکسون (M.A.Jackson) می‌گوید: "اولین کار معقول یک (مهندس نرم‌افزار) شناخت تفاوت بین اجرای یک برنامه و درست اجرا شدن آن است." [JAC75]. مفاهیم اصولی طراحی نرم‌افزار، چارچوب لازم برای طراحی درست را فراهم می‌آورند.

۱۳-۴-۱ تجرید

نقل قول

مجرد سازی یکی از راه های اساسی فائق آمدن بر پیچیدگیها است. گریدی بوچ

وقتی برای هر مسئله به‌دنبال راه‌حل پیمانه ای باشیم، بسیاری از سطوح انتزاع نیز مطرح می‌گردند. در بالاترین سطح انتزاع، راه‌حل با به‌کارگیری زبان محیط مسئله و به‌صورت کلی بیان می‌شود. در سطوح پایین‌تر انتزاع، جهت‌گیری و گرایش بیشتر رویه‌ای است. اصطلاحات مسئله‌گرا در تلاش برای بیان یک راه‌حل با اصطلاحات اجرایی تلفیق می‌شوند. و نهایت این‌که در پایین‌ترین سطح انتزاع، راه‌حل به‌گونه‌ای بیان می‌شود که مستقیماً قابل اجرا باشد. واسرمن [WAS83][۴] تعریف مناسب و مفیدی را ارائه می‌دهد:

مفهوم روان‌شناختی در انتزاع این امکان را به‌وجود می‌آورد تا شخص در سطحی کلی و بدون توجه به جزییات نامربوط سطح پایین، روی مسئله‌ای تمرکز کند. هم‌چنین به فرد امکان می‌دهد تا با مفاهیم و اصطلاحات آشنا در محیط مسئله کار کند، بدون آن‌که مجبور باشد آنها را به ساختاری ناآشنا تبدیل نماید.

1.External quality factors

۲. توضیح مفصل عوامل کیفیتی در فصل ۱۹ ارائه شده است.

3.Internal quality factors

4.Wasserman, A.

هر یک از مراحل فرآیند نرم‌افزار، اصلاح یا پالایشی در سطح انتزاعی راه‌حل نرم‌افزار است. در طول مهندسی سیستم، نرم‌افزار یکی از عناصر تخصیصی سیستم کامپیوتری به‌شمار می‌رود. درحین تحلیل نیازمندی‌های نرم‌افزار، راه‌حل نرم‌افزاری براساس اصطلاحاتی که در محیط مسئله آشنا هستند، بیان می‌شود. با پیشروی در روند طراحی سطح انتزاع کاهش می‌یابد. و نهایت این‌که با ایجاد برنامه منبع، پایین‌ترین سطح انتزاع حاصل می‌گردد.

توصیه

به عنوان یک طراح، هم برای رویه ها و هم برای تجزیه داده ها، باید سخت بکوشید، تا مسئله تحت کنترل شما درآید. و البته از استفاده مجدد نیز غافل نباشید.

حین پیشروی در سطوح مختلف انتزاع، تلاش می‌کنیم تا انتزاع‌های رویه‌ای و داده‌ها ایجاد گردد. انتزاع رویه‌ای[1] توالی مشخصی از دستورالعمل‌ها است که عملکرد خاص و محدودی دارد. نمونه‌ای از یک انتزاع رویه‌ای، وجود کلمهٔ "Open" برای یک "Door" می‌باشد. Open بر زنجیرهٔ طولانی مراحل رویه‌ای دلالت دارد: (مثلاً رفتن به سمت در، یا بردن دست و گرفتن دستگیره، چرخاندن دستگیره و کشیدن در، فاصله گرفتن از در و غیره)

انتزاع یا تجرید داده ها[2]، مجموعهٔ مشخصی از داده هاست که یک شی داده ای را توصیف می کند (فصل ۱۲) با توجه به انتزاع رویه‌ای Open (بازکردن) که در بالا اشاره شده، می‌توانیم یک انتزاع داده‌ای بنام Door (در) را تعریف کنیم. مانند تمامی اشیاء داده‌ای، انتزاع داده‌ای Door مشتمل بر مجموعه خصوصیاتی است که در را توصیف می‌کند. (مثل نوع در، جهت چرخش، مکانیزم باز کردن، وزن و ابعاد) نتیجه آن است که انتزاع رویه‌ای Open از اطلاعات موجود در مشخصات انتزاع داده‌ای، Door استفاده می‌کند.

انتزاع کنترل سومین شکل انتزاعی است که در طراحی نرم‌افزار به‌کار می‌رود. همانند انتزاع رویه‌ای و داده‌ای، انتزاع کنترل بر مکانیزم کنترل برنامه بدون مشخص کردن جزییات داخلی اشاره دارد. نمونهٔ انتزاع کنترل "سمافور همزمانی"[3] [KAT83][4] است که برای هماهنگ کردن فعالیتها در سیستم عامل به‌کار می‌رود. مفهوم انتزاع کنترل در فصل ۱۴ به‌طور مختصر مورد بحث و بررسی قرار می‌گیرد.

۱۳-۴-۲ پالایش

پالایش گام به گام[5] یک راهبرد طراحی بالا به پایین است که اولین بار توسط نیکلاس ورث [Kir71] پیشنهاد گردید. برنامه با سطوح پالایشی متوالی جزییات رویه‌ای، توسعه می‌یابد. تا زمان دستیابی به دستورات زبان برنامه‌نویسی، ایجاد توسعهٔ سلسله مراتب با تجزیهٔ دستور ماکروسکوپی عمل (انتزاع رویه‌ای) بر شیوه‌ای گام به گام صورت می‌گیرد. دید کلی این مفهوم توسط ورث ارائه می‌شود:

1.procedural abstraction

2.Data Abstraction

3.synchronization semaphore

4.Kaiser, S.H.

5.Stepwise Refinement

در هر مرحله از پالایش، یک یا چند دستورالعمل برنامهٔ فرضی، به دستورات جزیی‌تر تجزیه می‌گردد. این تجزیه یا پالایش پی‌درپی مشخصات، زمانی پایان می‌یابد که تمامی دستورالعمل‌ها براساس یک کامپیوتر واقعی یا زبان برنامه‌نویسی بیان شوند ... به هنگام پالایش وظایف، ممکن است لازم باشد داده‌ها نیز پالایش، تجزیه یا سازمان‌دهی شوند و پالایش برنامه و خصوصیات داده‌ای به موازات هم، طبیعی است.

هر یک از مراحل پالایش با برخی تصمیمات طراحی همراه است ... بنابراین آگاهی برنامه‌نویس از معیارهای زیربنایی (برای تصمیمات طراحی) و وجود راه‌حل‌های دیگر، اهمیت دارد.

فرآیند پالایش برنامه پیشنهاد شدهٔ ورث با فرآیند پالایش و تقسیم‌بندی به‌کار رفته در تحلیل نیازمندیها، قابل قیاس است و تفاوت آنها در میزان جزییات اجرایی می‌باشد نه شیوه و روش.

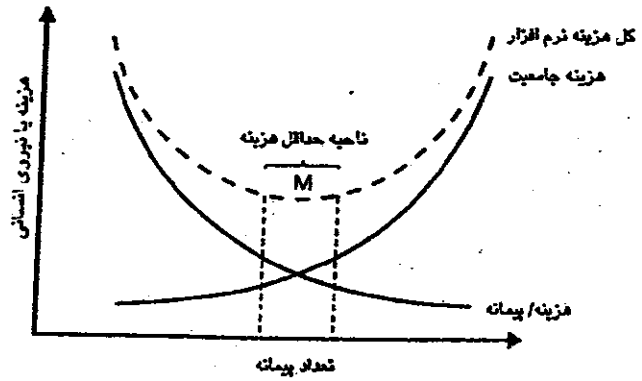
پالایش در واقع یک فرآیند بسط است که با گزارش عمل (یا توصیف اطلاعات) در سطح بالایی از انتزاع تعریف می‌گردد، آغاز می‌شود. یعنی آن‌که این گزارش، عمل یا اطلاعات را به‌طور ذهنی توصیف می‌کند اما هیچ اطلاعاتی دربارهٔ‌عملکردهای داخلی کار یا ساختار درونی اطلاعات ارائه نمی‌دهد. پالایش موجب می‌شود طراح بیان اولیه را بسط و گسترش دهد و در هر پالایش (بسط) متوالی، جزییات بیشتری را فراهم سازد.

توصیه

هرچند تمایلی برای پرداختن سریع به جزئیات با پرش از گامهای طبقه بندی و پالایش وجود دارد، این امر موجب بروز خطاها و از قلم افتادگی و مهمتر از همه دشوار شده بازبینی طراحی خواهد شد. پالایش گام به گام را دنبال کنید.

۱۳-۴-۳ پیمانه سازی

مفهوم پیمانه ای در نرم‌افزار کامپیوتر تقریباً پنج دهه، مورد حمایت واقع شده است. معماری نرم‌افزار (که در بخش ۱۳-۴-۴ توصیف شده) پیمانه‌ای است؛ یعنی آن‌که، نرم‌افزار بر اجزای نشانی پذیر با اسامی جداگانه به‌نام "پیمانه ها"[1] تقسیم می‌شود که برای رفع نیازهای مسئله، یکپارچه و مجتمع می‌باشند.



شکل ۱۳-۲ پیمانه شدن و هزینه نرم افزار

گفته شده است که "پیمانه ای بودن تنها خصوصیت نرم‌افزاری است که امکان کنترل، عقلانی برنامه را فراهم می‌نماید." [MYE78] نرم افزار یکپارچه[1] (یعنی برنامهٔ گستردهٔ شامل یک پیمانه) به راحتی

1.modules

توسط خواننده، قابل درک نیست. تعداد مسیرهای کنترل، گستردگی ارجاع، تعداد متغیرها و پیچیدگی کمی، درک را تقریباً ناممکن می‌سازند. برای روشن شدن این نکته، به استدلال زیر براساس مشاهدات حل مسئله انسانی توجه کنید:

فرض می‌کنیم C(x) تابعی است که پیچیدگی محسوس یک مسئله را با مقدار x بیان کرده و E (x) نیز تابع بیانگر میزان تلاش لازم در واحد زمان برای حل یک مسئله با مقدار x می‌باشد. در مورد دو مسئله p_1 و p_2 :

نقل قول

همواره برای هر مسئله انسانی یک راه حل ساده وجود دارد راه حلی راحت با ظاهری آراسته و البته اشتباه.
اچ.ال.منکن

$$C(p_1) > C(p_2) \qquad \text{(الف ۱-۱۳)}$$

که در ادامه خواهیم داشت :

$$E(p_1) > E(p_2) \qquad \text{(ب ۱-۱۳)}$$

به‌عنوان یک مورد کلی، این نتیجه از لحاظ عقلی واضح است. یعنی هر چه مسئله سخت‌تر و دشوارتر باشد، حل آن به زمان بیشتری نیاز دارد.

خصوصیت جالب دیگری نیز در آزمون حل مسئله انسانی کشف شده است. یعنی:

$$C(P_1+P_2) > C(P_1) + C(P_2) \qquad (۲-۱۳)$$

معادلهٔ فوق نشانگر آن است که پیچیدگی ملموس ترکیب دو مسئله P_2 و P_1 بیشتر از پیچیدگی و دشواری هر یک از p_1 و p_2 به‌طور جداگانه است. با در نظر گرفتن معادلهٔ (۱۳-۲) و شرط موجود در معادلهٔ (۱۳-۱) ، نتیجهٔ زیر حاصل می‌شود:

$$E(P_1+P_2) > E(P_1) + E(P_2) \qquad (۳-۱۳)$$

که این نتیجه به راه‌حل "تقسیم و غلبه " منجر می‌گردد ـ یعنی حل یک مسئله دشوار و پیچیده با تقسیم آن به بخش‌های قابل کنترل، راحت‌تر و ساده‌تر خواهد بود. نتیجهٔ بیان شده در معادلهٔ (۱۳-۳) تضمین‌های مهمی را در ارتباط با پیمانه ای بودن نرم‌افزار در بردارد و در واقع استدلالی برای پیمانه‌ای شدن به‌شمار می‌رود.

توصیه

پیمانه نمودن را به طور افراطی دنبال نکنید، زیرا سادگی پیمانه ها در سایه پیچیدگی انسجام و پیوستگی آنها قرار خواهد گرفت.

از معادلهٔ (۱۳-۳) می‌توان چنین نتیجه گرفت که اگر نرم‌افزار به‌طور نامعین و مجدد تقسیم‌بندی کنیم، تلاش لازم برای توسعهٔ بسیار ناچیز خواهد بود! متأسفانه عوامل دیگری نیز در کارند که باعث بی‌اعتباری این نتیجه می‌گردند. با توجه به شکل ۱۳-۲، تلاش یا (هزینهٔ) لازم برای توسعهٔ یک پیمانه جداگانهٔ نرم‌افزاری، با افزایش تعداد کل پیمانه ها، کاهش می‌یابد. با فرض مجموعهٔ یکسانی از نیازمندیها، پیمانه‌های بیشتر، به معنای اندازه‌های کوچکتر است، حال آنکه ازدیاد تعداد پیمانه ها، تلاش یا (هزینهٔ) مربوط به یکپارچه‌سازی آنها را افزایش می‌دهد. این خصوصیات با منحنی هزینه یا تلاش کل در تصویر به

1.Monolitic Software

نمایش درآمده است. تعداد مشخصی از پیمانه ها (M) منجر به هزینهٔ حداقل توسعه می‌گردد. اما برای پیشبینی M با اطمینان، از پیشرفت فنی لازم برخوردار نیستیم.

منحنی‌های نشان داده شده در شکل ۱۳-۲ راهنمای مفیدی برای پیمانه ای کردن می‌باشند. پیمانه ای کردن نرم‌افزار الزامی است، اما حفظ اعتدال M نیز بایستی مورد توجه و دقت قرار گیرد. یعنی از پیمانه ای کردن کمتر یا بیش از حد باید اجتناب کرد. اما دانستن "میانگین M" چگونه ممکن است؟ و چطور باید نرم‌افزار را پیمانه ای کرد؟ جواب این سؤالات مستلزم درک سایر مفاهیم طراحی است که در بخش‌های بعدی این فصل مورد بررسی قرار می‌گیرند. سؤال مهم بعدی هنگام در نظر گرفتن پیمانه ای کردن مطرح می‌شود. چگونه پیمانه‌ای مناسب با اندازه‌ای مشخص را تعریف می‌کنیم؟ جواب این سؤال بسته به شیوه هایی است که برای تعریف پیمانه ها در یک سیستم به‌کار می‌رود. مایز[MEY88][1] پنج معیار را در ارزیابی یک شیوهٔ طراحی و بر اساس توانایی آن در تعریف یک سیستم مؤثر پیمانه ای معرفی می‌کند:

ارجاع به فصلها

شیوه های طراحی در فصل های ۱۴، ۱۵، ۱۶ و ۲۲ تشریح شده اند.

تجزیه‌پذیری پیمانه‌ای: اگر شیوهٔ طراحی مکانیزم منظمی را برای تجزیهٔ مسئله به مسایل فرعی ارائه دهد، در آن‌صورت پیچیدگی مسئله کلی کاهش یافته و بدین‌ترتیب تحقق یک راه‌حل مؤثر پیمانه ای میسر می‌گردد.

قابلیت ترکیب پیمانه‌ای: اگر یک شیوهٔ طراحی امکان هم‌گذاری اجزای موجود (قابل استفاده مجدد) طراحی را در یک سیستم جدید به‌وجود بیاورد، آن‌گاه یک راه‌حل پیمانه ای را به‌دست خواهد داد که دوباره‌کاری نخواهد داشت.

پرسش

چگونه می توانیم موثر بودن یک شیوه طراحی را در این امر که به پیمانه سازی کمک می کند، ارزیابی کنیم.

قابلیت درک پیمانه‌ای: اگر پیمانه ای به‌عنوان یک پیمانه مستقل قابل درک باشد (بدون ارجاع به پیمانه های دیگر) ساختن و تغییر آن آسان‌تر خواهد بود.

استمرار پیمانه‌ای: اگر تغییرات کوچک در نیازمندی‌های سیستم، بیش از تغییر سیستم، منجر به تغییرات در پیمانه های جداگانه شود، تأثیر اثرات جانبی حاصل از تغییر به حداقل خواهد رسید.

محافظت پیمانه‌ای: اگر در یک پیمانه شرایط غیرعادی پیش بیاید و تأثیرات آن به همان پیمانه محدود شود، تأثیر اثرات جانبی ناشی از خطا، به‌حداقل خواهد رسید.

در پایان، ذکر این نکته مهم است که در صورت لزوم اجرای یکپارچهٔ سیستم، طراحی آن ممکن است به‌صورت پیمانه ای باشد. موقعیت‌هایی وجود دارند (مثل نرم‌افزار بدون درنگ، نرم‌افزار جاسازی شده) که در آنها سرعت تقریباً حداقل و سرریز حافظه ارائه شده توسط زیر برنامه‌ها، (مثل زیر روال‌ها، زیربرنامه‌ها) قابل قبول نمی‌باشد. در چنین شرایطی، طراحی پیمانه ای نرم افزار به‌عنوان یک نظریهٔ فوق‌العاده و درجه اول، الزامی است. برنامه‌نویسی ممکن است به‌طور خطی توسعه یابد. گر چه امکان دارد کد منبع برنامه در

ارجاع به وب

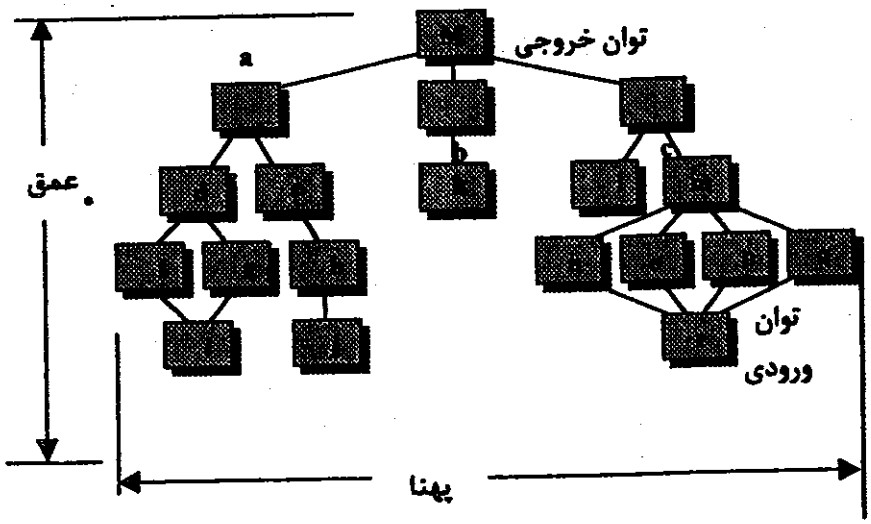
اطلاعات کامل و منابع مربوط به فن آوری معماری نرم افزاری " استار " در آدرس زیر قرار دارد:
www.ast.tds-gn.inco.com/arch/guid.html

[1] I.Meyer, B.

وهلهٔ نخست پیمانه ای به‌نظر نرسد، اما این نظریه برقرار بوده و برنامه، مزایای یک سیستم پیمانه‌ای را فراهم خواهد کرد.

۱۳-۴-۴ معماری نرم افزار

معماری نرم‌افزار به "ساختار کلی نرم‌افزار و راههای ایجاد یکپارچگی ذهنی سیستم از طریق این ساختار" اشاره می‌کند. [SHA95a][1]. معماری در ساده‌ترین شکل خود عبارت است از ساختار سلسله مراتبی اجزاء برنامه (پیمانه ها)، شیوهٔ ارتباط این اجزاء و ساختار داده‌هایی که توسط اجزاء مورد استفاده قرار می‌گیرند. هر چند در مفهوم گسترده‌تر، می‌توان "اجزاء" را برای نشان دادن عناصر اصلی سیستم و ارتباطات آنها، تعمیم داد.[2]



شکل ۱۳ـ۳ شناخت اصطلاحات ساختار برای یک سبک معماری فراخوانی و بازگشت

یکی از اهداف طراحی نرم‌افزار، نقشهٔ معماری سیستم است. این نقشه چارچوب و مبنای‌ست که فعالیت‌هایی جزیی‌تر طراحی براساس آن انجام می‌گیرند. مجموعه الگوهای معماری، مهندس نرم‌افزار را قادر می‌سازد تا مفاهیم سطح طراحی را مجدداً قابل استفاده نماید.

شاو و گارلان [SHA95a] مجموعه خصوصیاتی را توصیف می‌کنند که باید به‌عنوان بخشی از طراحی معماری تصریح شوند:

نقل قول

یک معماری نرم افزار یعنی آنکه کار توسعه محصول بازگشتی عالی از سرمایه همراه با کیفیت،زمان بندی و هزینه داشته باشد

1.Shaw, M. and D. Garlan

۲. برای مثال، اجزاء معماری یک سیستم خادم/ مخدوم به عوان سطح تجریدی متفاوتی بازنمایی می شوند. به فصل ۲۸ برای جزئیات بیشتر مراجعه نمایید.

خصوصیات ساختاری. این جنبه از نمایش طراحی معماری، اجزاء یک سیستم (مثل پیمانه ها، اشیاء، فیلترها) و نحوهٔ بسته‌بندی و ارتباط این اجزاء با یکدیگر تعریف می‌کند. به‌عنوان مثال، بسته‌بندی اشیاء به‌گونه‌ای است که داده‌ها و نیز پردازش و دست‌کاری آنها را در خود جای داده و ارتباط آنها از طریق احضار شیوه‌ها صورت می‌گیرد. (فصل ۲۰)

ویژگی‌های اضافی کاربردی. توصیف طراحی معماری باید توضیح دهد که چگونه معماری طراحی نیازمندیهای عملکرد، ظرفیت، قابلیت اطمینان، امنیت، وقفه‌پذیری و سایر خصوصیات سیستم را فراهم می‌آورد.

خانواده‌های سیستم‌های وابسته. طرح معماری باید از الگوهای قابل تکرار و رایج در طراحی خانواده سیستم‌های مشابه استفاده کند. اساساً طراحی باید توانایی استفادهٔ مجدد از عناصر سازندهٔ معماری را داشته باشد.

پس از تعیین این مشخصات، طراحی معماری با استفاده از یک یا چند مدل مختلف قابل نمایش است. [GAR95][1]. مدل‌های ساختاری، معماری را به‌صورت مجموعهٔ سازمان یافتهٔ اجزاء برنامه نشان می‌دهند. مدل‌های چارچوب، در تلاش برای شناسایی چارچوب‌ها (الگوهای) قابل تکرار طراحی معماری در انواع مشابه برنامه‌های کاربردی، سطح انتزاعی طراحی را افزایش می‌دهند. مدل‌های پویا به جنبه‌های عملی معماری برنامه پرداخته و بیانگر آنند که چگونه ممکن است ساختار یا پیکربندی سیستم به تبعیت از رویدادهای خارجی تغییر کند. مدل‌های فرآیند بر طراحی فرآیندکاری یا فنی سیستم تأکید دارند. و در آخر این‌که مدل‌های کارکردی[2] برای ارائه سلسله مراتب کارکردی سیستم، مورد استفاده قرار می‌گیرند.

پنج گونه مختلف از مدل ها برای بازنمایی طراحی معماری، مورد استفاده دارند.

تعدادی زبان‌های مختلف توصیف معماری[3] (ADLs) برای نمایش مدل‌های فوق‌الذکر توسعه یافته‌اند. [SHA95b][4]. با وجود پیشنهاد بسیاری از ADLs مختلف، اما اکثریت آنها مکانیزم‌هایی را برای توصیف اجزای سیستم و نحوهٔ ارتباط آنها با یکدیگر، عرضه می‌کنند.

در فصل ۱۴، توصیفی کامل از الگوها و سبک های معماری ارائه گردیده است.

۱۳-۴-۵ سلسله مراتب کنترل

"سلسله مراتب کنترل" که "ساختار برنامه"[5] نیز نام دارد، بیانگر سازمان‌دهی اجزاء برنامه (پیمانه ها) بوده و بر سلسله مراتب کنترل دلالت دارد. سلسله مراتب کنترل نشانهٔ جنبه‌های رویه‌ای نرم‌افزار مثل توالی

1.Garlan, D. and M. Shaw
2.Functional Models
3.Architectural Description Languages (ADL)
4.Shaw, M.
5.Program Structure

فرآیندها، وقوع/ ترتیب تصمیمات یا تکرار عملکردها نبوده و لزوماً در تمامی سبک‌های معماری قابل کاربرد نمی‌باشد.

در آن‌دسته از سبک‌های معماری که قابل نمایش هستند، نشان‌گذاری‌های مختلفی برای نمایش سلسله مراتب کنترل به‌کار می‌روند. رایج‌ترین آنها، نمودار درختی (شکل ۱۳-۳) است که کنترل سلسله مراتبی را در معماری‌های "فراخوانی و بازگشت"[1] هر چند سایر نشان‌گذاری‌ها نیز از قبیل نمودارهای وارنیر اور[ORR77][2] و جکسون [JAC83][3] ممکن است با تأثیر یکسان به‌کار روند و برای تسهیل بحثهای بعدی دربارهٔ ساختار، برخی مقیاس‌ها و اصطلاحات ساده را معرفی می‌کنیم. با توجه به شکل ۱۳-۳ عمق و پهنا به‌ترتیب بیانگر تعداد سطوح کنترل و گسترهٔ کلی کنترل[4] می‌باشند. توان خروجی[5] مقیاس تعداد پیمانه هایی است که مستقیماً توسط پیمانه ای دیگر کنترل می‌شوند. توان ورودی[6] نیز بیانگر تعداد پیمانه هایی است که یک پیمانه خاص را به‌طور مستقیم کنترل می‌کنند.

رابطهٔ کنترلی میان پیمانه ها به شیوهٔ زیر بیان می‌شود. پیمانه ای که پیمانه دیگری را کنترل می‌کند، پیمانه حاکم[7] نام دارد و برعکس پیمانه تحت کنترل پیمانه دیگر، تابع‌پیمانه کنترل‌کننده می‌باشد. [YOU79][8] به‌عنوان مثال با در نظر داشتن تصویر ۱۳-۳، پیمانه M نسبت به پیمانه های c, b, a پیمانه حاکم بوده، پیمانه h تابع پیمانه e و در نهایت تابع پیمانه M میباشد. گرچه عملاً امکان بیان روابط عرضی (مثلاً بین پیمانه های e, d) وجود دارد، توصیف این روابط با اصطلاحات صریح لازم نیست.

سلسله مراتب کنترل هم‌چنین بیانگر دو خصوصیت اندک متفاوت معماری نرم‌افزار است : یعنی وضوح[9] و اتصال[10]. وضوح بر مجموعه اجزای برنامه‌ای اشاره دارد که ممکن است احضار شده یا به‌عنوان داده‌ها توسط یک جزء فرضی مورد استفاده قرار گیرند، حتی وقتی این کار به‌صورت غیرمستقیم انجام می‌شود. به‌عنوان مثال، یک پیمانه در سیستم شی‌گرا ممکن است به گروه وسیعی از اشیاء داده‌ای که به ارث برده است، دسترسی داشته باشد. اما تنها از تعداد کمی از این اشیاء داده‌ای استفاده می‌کند. تمامی اشیا برای پیمانه قابل رؤیتند. اتصال بیانگر مجموعه اجزایی است که مستقیماً احضار شده یا به‌عنوان داده

توصیه

اگر شما به توسعه نرم افزاری به شیوه شی گرا پرداخته اید، معیارهای ساختاری ارائه شده در اینجا به کارتان نخواهد آمد، و البته معیارهای دیگری (در بخش چهارم) قابل استفاده اند

---

۱. یک معماری فراخوانی و بازگشت (فصل ۱۴) یک ساختار سنتی برنامه است که به سلسله مراتبی از کنترل تجزیه می شود. یک برنامه اصلی، اجزاء دیگر را فرا می خواند در حالی که هریک از آنها نیز می توانند چنین کنند.

2.Orr, K.T.

3.Jackson, M.A.

4.span of control

5.Fan – out

6.Fan – in

7.Subordinate

8.Yourdon, E., and L.

9.visibility

10.Conectivity

توسط یک جزء خاص مورد استفاده واقع می‌شوند. مثلاً، پیمانه که مستقیماً موجب می‌شود پیمانه دیگر اجرا را آغاز کند، بر آن متصل است.[1]

۱۳-۴-۶ تجزیه ساختاری

پرسش

تجزیه افقی چه مزایایی دارد؟

اگر سبک معماری یک سیستم سلسله مراتبی باشد، می‌توان ساختار برنامه را هم به‌صورت افقی و هم به‌طور عمودی تقسیم‌بندی کرد. با توجه به شکل ۱۳-۴ الف، تقسیم‌بندی افقی[2]، شاخه‌های جداگانهٔ سلسله مراتب پیمانه ای را برای هر یک از وظایف اصلی برنامه تعیین می‌کند. پیمانه های کنترل[3] که با سایهٔ تیره‌تر نشان داده شده‌اند، برای هماهنگی ارتباط بین وظایف و اجرای آنها به‌کار می‌رود. ساده‌ترین شیوهٔ تقسیم‌بندی افقی، سه بخش را تعیین می‌کند که عبارتند از: ورودی، تغییر و تبدیل داده‌ها (که اغلب فرآیند نام دارد) و خروجی تقسیم‌بندی معماری به‌صورت افقی، مزایای روشنی را به‌همراه دارد.

- منجر به نرم‌افزاری می‌شود که آزمون آن آسان‌تر است.
- منجر به نرم‌افزاری می‌شود که نگهداری و حفظ آن آسان‌تر است.
- منجر به نرم‌افزاری می‌شود که انتشار آن از پیامدهای جنبی کمتری برخوردار است.
- نرم‌افزاری را ایجاد می‌کند که بسط و توسعهٔ آن ساده‌تر است.

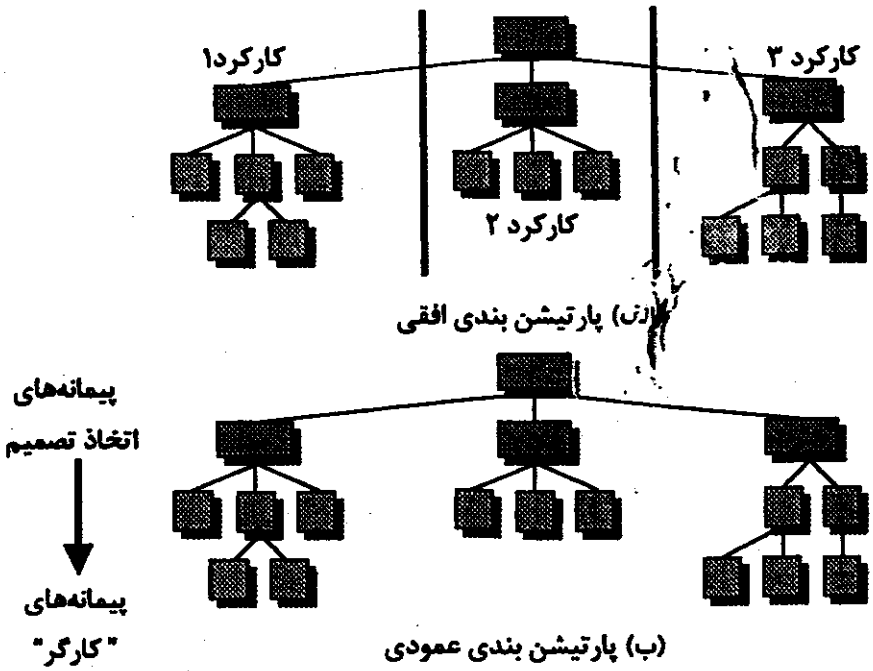
از آن‌جا که کارکرد اصلی از یکدیگر جدا می‌گردند، تغییرات پیچیدگی کمتری دارند و بسط و توسعهٔ سیستم (که امری رایج است) بدون تأثیرات جانبی، راحت‌تر انجام می‌شود. از جنبهٔ منفی، تقسیم‌بندی افقی اغلب باعث می‌شود داده‌های بیشتری از رابطهای پیمانه عبور کرده و کنترل کلی گردش برنامه را دشوار و پیچیده می‌سازد. (در صورتی‌که فرآیند مستلزم جابه‌جایی سریع از یک کارکرد به کارکردی دیگر باشد).

---

۱. در فصل ۲۰، مفهوم وراثت را در نرم افزار شیء گرا توضیح داده ایم. یک جزء برنامه می تواند داده یا کنترل را از دیگر جزء به ارث برد بدون آنکه صریحاً" در کد برنامه آمده باشد. اجزاء با چنین خاصیتی مرتب شده اند اما اتصال مستقیم آنها مشهود نمی باشد. یک چارت ساختاری (فصل ۱۴) اتصال را معلوم می سازد.

2.horizental partitioning

3.control modules



شکل ۱۳ـ۴ پارتیشن بندی ساختاری

تقسیم‌بندی عمودی[1] (شکل ۱۳-۴ـب) که اغلب تجزیهٔ عاملی (فاکتورگیری)[2] نام دارد، بیانگر آنند که کنترل (تصمیم‌گیری) و کار باید در ساختار برنامه توزیعی بالا به پایین داشته باشد. پیمانه های سطح بالا باید اعمال کنترلی را انجام داده و کار فرآیندی کمی را عهده‌دار شوند. پیمانه های واقع در قسمت پایین ساختار باید کارگرائی باشند که همهٔ کارهای ورودی، محاسبه و خروجی را انجام می‌دهند.

ویژگی تغییر در ساختارهای برنامه، ضرورت تقسیم‌بندی عمودی را توجیه می‌کند. با توجه به (تصویر ۱۳-۴ ب) می‌توان مشاهده کرد که تغییر در پیمانه کنترل (در بالای ساختار)، احتمال تولید تأثیرات جانبی را در پیمانه های تابع بالاتر می‌برد. امّا تغییر در یک پیمانه کارگر، با فرض سطح پایین آن در ساختار، احتمال ایجاد اثرات جانبی را کمتر می‌کند. به‌طور کلی، تغییرات در برنامه‌های کامپیوتر، تغییرات ورودی و محاسبه یا تبدیل، و خروجی را در پی دارد. احتمال تغییر در ساختار کنترل کلی برنامه (یعنی عملکرد اصلی آن) بسیار کمتر است. به‌همین دلیل، ساختارهایی با تقسیم‌بندی عمودی به هنگام تغییرات، کمتر در معرض تأثیرات جانبی بوده و از این‌رو قابلیت حفظ و نگهداری آنها بیشتر است ـ و این یک عامل کلیدی کیفیت به‌شمار می‌رود.

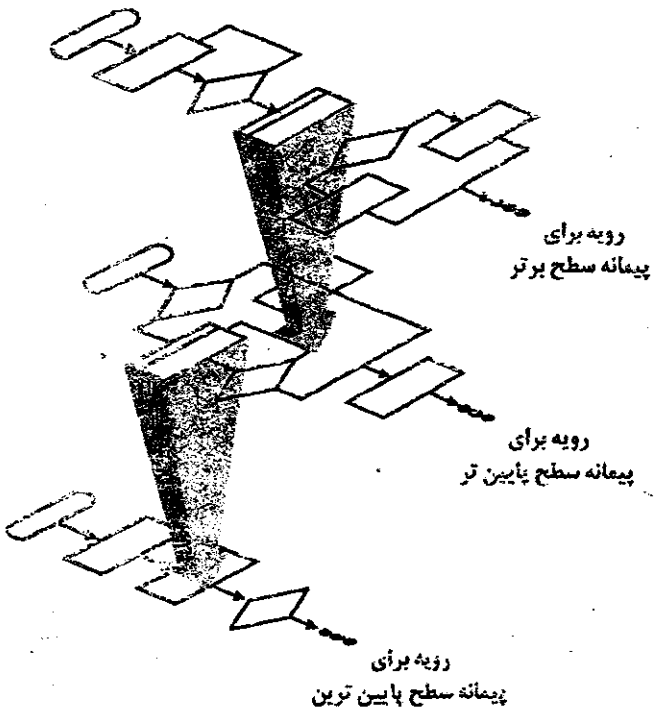
پیمانه های " کارگر " بیش از پیمانه های کنترلی، حار تغییر و تحول می شوند، هرقدر " ورکر " در ساختار کمتر مور استفاده قرار دهیم عواقب و پیامدهای مبیر، کاهش ـ هد یافت.

1.vertical partitioning
2.factoring

۱۳-۴-۷ ساختار داده ها

ساختار داده‌ها[1]، نمایش رابطهٔ منطقی میان عناصر جداگانهٔ داده‌ها است. از آن‌جا که ساختار اطلاعات بر طراحی نهایی رویه‌ای تأثیر خواهد داشت، ساختار داده‌ها به اندازهٔ ساختار برنامه ماهیت متصاوری برنامه‌نویس اهمیت دارد.

ساختار داده‌ها تعیین‌کنندهٔ سازمان‌دهی، روش‌های دستیابی، میزان سرکت‌پذیری و جابگزینی‌های فرآیند اطلاعات می‌باشد. متون بسیاری (مثل [AHO83][2]، [KRU84][3] و [GAN89][4]) به این موضوعات اختصاص یافته و بحث کامل در این‌باره از حوزهٔ این کتاب خارج است. هر چند، درک شیوه‌های کلاسیک موجود برای سازمان‌دهی اطلاعات و مفاهیم زیربنایی سلسله مراتب اطلاعات، از اهمیت برخوردار است.



شکل ۱۳ - ۵ رویه لایه لایه شده

1.data structure

2.Aho, A.V., J. Hopcroft, and J. Ullmann

3.Kruse, R.L.

4.Gonnet, G.

سازمان‌دهی و پیچیدگی ساختار داده‌ها تنها به‌واسطهٔ خلاقیت و مهارت طراح محدود می‌گردد. هر چند، تعداد محدودی ساختارهای داده‌ای کلاسیک وجود دارند که عناصر سازندهٔ ساختارهای پیچیده‌تر بشمار می‌روند.

عنصرعددی[1]، ساده‌ترین شکل ساختارهای داده‌ای است و همان‌طور که از اسم آن پیداست، نشان‌گر یک عنصر پیمانه اطلاعاتی است که ممکن است توسط یک شناسه احضار شده و بدین‌ترتیب دستیابی با مشخص کردن یک آدرس پیمانه در حافظه، تحقق می‌یابد. اندازه و قالب عنصرعددی ممکن است در حد و مرزهای تعیین شده توسط یک زبان برنامه‌نویسی متغییر باشد. به‌عنوان مثال، یک عنصرعددی امکان دارد یک عنصر منطقی با سایز یک بیت، عدد صحیح یا ممیز شناور با اندازهٔ ۸ تا ۶۴ بیت یا یک رشتهٔ نویسه‌ای با طول صد یا هزاران بایت باشدوقتی اقلام عددی بصورت یک لیست یا گروه پیوسته، سازمان‌دهی می‌شوند، یک بردار ترتیبی[2] تشکیل می‌شود. بردارها رایج‌ترین شکل ساختارهای داده‌ها هستند و در را به روی شاخص‌دهی متغیر اطلاعات باز می‌کند.

هنگامی‌که بردار ترتیبی به دو، سه یا نهایتاً تعداد دلخواهی ابعاد توسعه می‌یابد. یک فضای n بعدی ایجاد می‌گردد. رایج‌ترین فضای n بعدی[3]، ماتریس دو بعدی می‌باشد. در بسیاری از زبان‌های برنامه‌نویسی، یک فضای n بعدی، آرایه[4] نام دارد.

اقلام، بردارها و فضاها ممکن است با قالب‌های بسیاری سازمان‌دهی شوند. لیست پیوندی[5]، یک ساختار داده‌ای است که اقلام عددی غیرپیوسته، بردارها یا فضاها بنحوی سازمان‌دهی می‌کند (بنام گره‌ها) که فرآیند آنها بصورت یک لیست میسر گردد. هر گره مشتمل بر سازمان‌دهی مناسب داده‌ها (مثل یک بردار) و یک یا چند اشاره‌گر است که دال بر نشانی در ذخیرهٔ گرهٔ بعدی لیست می‌باشند. می‌توان با تعریف دوبارهٔ اشاره‌گرها برای پذیرفتن ورودی جدید لیست، گره‌ها را در سر نقطه‌ای از لیست، افزایش داد.

سایر ساختارهای داده‌ای با استفاده از ساختارهای کلی داده‌ای اصلی توصیف شدهٔ فوق ساخته می‌شوند. مثلاً، ساختار داده‌ای سلسله مراتبی[6] با کاربرد لیست چند اتصالی شامل اقلام عددی، بردارها و احتمالاً فضاهای n بعدی تحقق می‌یابند. ساخت سلسله مراتبی غالباً در برنامه‌های کاربردی به چشم می‌خورد که مستلزم شرکت‌پذیری و دسته‌بندی اطلاعات می‌باشند.

نقل قول

ترتیب و اتصال ایده ها مشابه ترتیب و اتصال اشیا است. باروخ اسپینوزا

توصیه

همان قدر که برای طراحی الگوریتم ها وقت می گذارید، زمانی را به طراحی ساختار داده ها اختصاص دهید. اگر اینگونه عمل نمایید در اجرای طولانی مرقه جوبی زمانی خواهید داشت.

1.scalar item
2.sequential vector
3.n-dimensional space
4array
5.linked list
6.hierachical data structure

لازم به یادآوری است که ساختارهای داده‌ها مانند ساختار برنامه، در سطوح مختلف انتزاع قابل نمایش است. به‌عنوان مثال پشته، مدل ذهنی یک ساختار داده‌ای است که می‌تواند به‌صورت بردار یا لیست پیوندی پیاده گردد. بسته به سطح جزییات طراحی، عملکرد داخلی پشته ممکن است مشخص شده یا نشود.

۱۳-۴-۸ رویه نرم افزار

ساختار برنامه[1]، سلسله مراتب کنترل را بدون توجه به توالی فرآیند و تصمیمات تعیین می‌کند. رویهٔ نرم‌افزار بر جزییات پردازشی هر پیمانه به‌طور جداگانه تأکید دارد. رویه باید تعیین دقیق پردازش از جمله توالی رویدادها، نقاط دقیق تصمیم‌گیری، اعمال تکراری و حتی سازمان‌دهی/ ساختار داده‌ای را ارائه دهد.

البته، بین ساختار و رویه ارتباطی وجود دارد. فرآیند تعیین شده برای هر پیمانه ، باید ارجاع به تمامی پیمانه های تابع آن را در بر داشته باشد. یعنی نمایش رویه‌ای نرم‌افزار، همان‌طور که در شکل ۱۳-۵ نشان داده شده، لایه‌ای است.[2]

۱۳-۴-۹ پنهان سازی اطلاعات

مفهوم پیمانه ای بودن یک سؤال اساسی را برای هر طراح نرم‌افزاری مطرح می‌کند. «برای دستیابی به بهترین مجموعهٔ پیمانه ها، چگونه یک راه‌حل نرم‌افزاری را تجزیه کنیم؟» اصل اختفای اطلاعات[3] [PAR72][4] بیانگر آن است که "وجه مشخصهٔ پیمانه ها، تصمیمات طراحی است که هر پیمانه از پیمانه های دیگر مخفی می‌سازد." به‌عبارتی دیگر، پیمانه ها باید طوری طراحی و مشخص شوند که اطلاعات (رویه و داده‌ها) موجود در هر پیمانه برای پیمانه های دیگری که به چنین اطلاعاتی نیاز ندارند، غیرقابل دسترسی باشد.

اختفا (یا پنهان‌سازی) یعنی تعیین مجموعه‌ای از پیمانه های مستقل که تنها اطلاعات لازم برای تحقق عملکرد نرم‌افزاری را با یکدیگر مبادله می‌کنند و بدین‌ترتیب پیمانه ای شدن به‌طور مؤثر و سودمند تحقق می‌یابد. انتزاع، به تعیین عناصر رویه‌ای (یا اطلاعاتی) تشکیل‌دهندهٔ نرم‌افزار کمک می‌کند. اختفا محدودیت‌های دستیابی به جزییات رویه‌ای موجود در یک پیمانه و هر یک از ساختار داده‌ای محلی مورد استفادهٔ آن را تعین نموده و اعمال می‌کند.[ROS75] [1]

1.program structure

۲. این امر برای تمام سبک های معماری صادق نخواهد بود. برای مثال، لایه بندی سلسله مراتبی رویه ها، در معماری های شیٔ گرا وجود ندارند.

3.information hiding

4.Parnas, D.L.

۱۳-۵ طراحی موثر پیمانه ای (ماجولار)

مفاهیم اصلی طراحی که در قسمت قبلی توصیف شدند. همگی در خدمت طراحی‌های پیمانه ای هستند. در واقع، پیمانه ای شدن به شیوه‌ای قابل قبول در تمام رشته‌های مهندسی،تبدیل گشته است. طراحی پیمانه‌ای پیچیدگی را کاهش داده (رجوع به قسمت ۱۳-۴-۳)، تغییر را تسهیل نموده (جنبهٔ مهم قابلیت نگهداری نرم‌افزار) و با کمک به توسعهٔ موازی قسمت‌های مختلف سیستم، منجر به اجرای راحت‌تر می‌گردد.

۱۳-۵-۱ استقلال کارکردی

توصیه

یک پیمانه " یک فکر مجرد " است.اگر شما بتوانید با یک جمله، موضوع، گزاره یا شئ ... آن را تشریح کنید

مفهوم" استقلال کارکردی" پیامد مستقیم پیمانه ساختن و مفاهیم انتزاع و اختفای اطلاعات است. پارناس[PAR72] [2] و ورث [WIR71][3] در مقالات برجسته‌ای راجع به طراحی نرم‌افزار، به فنون پالایشی تقویت‌کنندهٔ استقلال پیمانه‌ای اشاره می‌کنند. کارهای بعدی انجام شده توسط کنستانتین [STE74][4]، مایرز و استیون این مفهوم را تثبیت نمود.

استقلال کارکردی از طریق ایجاد پیمانه هایی با عملکرد یک منظوره و عدم ارتباط بیش از حد با پیمانه های دیگر، تحقق می‌یابد. به بیان دیگر، ما قصد داریم نرم‌افزاری را طراحی کنیم که هر پیمانه یک وظیفهٔ خاص فرعی از ضرورت‌ها را انجام داده و از دید سایر بخش‌های ساختار برنامه، رابط ساده‌ای داشته باشد. به‌جا است که بدانیم استقلال چه اهمیتی دارد؟ توسعهٔ نرم‌افزاری با پیمانه‌ای شدن کارآمد یعنی پیمانه های مستقل، آسان‌تر است زیرا کار تقسیم‌بندی شده و رابط‌ها ساده شده‌اند. (انشعابات را هنگام انجام توسعه توسط یک تیم، در نظر بگیرید). نگهداری و آزمون پیمانه های مستقل ساده‌تر است زیرا تأثیرات ثانویه ایجاد شده به‌واسطهٔ تغییر طراحی/ برنامه، محدود می‌شوند. انتشار خطا کاهش می‌یابد و پیمانه هایی با قابلیت استفادهٔ مجدد امکان‌پذیر می‌گردند. به‌طور خلاصه، استقلال کارکردی رمز طراحی خوب و طراحی، رمز کیفیت نرم‌افزار است. استقلال با دو معیار کیفی ارزیابی می‌گردد: انسجام و اتصال (چسبیدگی و پیوستگی). "انسجام"[5] قدرت کارکردی نسبی یک پیمانه است و اتصال[6] مقیاس وابستگی نسبی پیمانه ها به‌هم می‌باشد.

1.Ross, D.

2.Parnas, D.L.

3.Wirth, N.

4.Stevens, W., G. Myers, and L. Constantine

5.Cohesion

6.Coupling

۱۳-۵-۲ چسبندگی

انسجام یا چسبندگی، بسط طبیعی مفهوم اختفای اطلاعات است که در قسمت ۱۳-۴-۸ توصیف گردید. یک پیمانه یکپارچه، یک وظیفه منفرد را در یک رویه نرم‌افزاری و با برقراری ارتباط محدود با رویه‌های در حال اجرای سایر بخش‌های برنامه، انجام می‌دهد. به بیان ساده‌تر، یک پیمانه منسجم (به‌طور ایده‌ال) باید تنها یک کار را انجام دهد.

چسبندگی، معیاری کیفی است که درجه تمرکز یک پیمانه بر یک چیز را مشخص می سازد

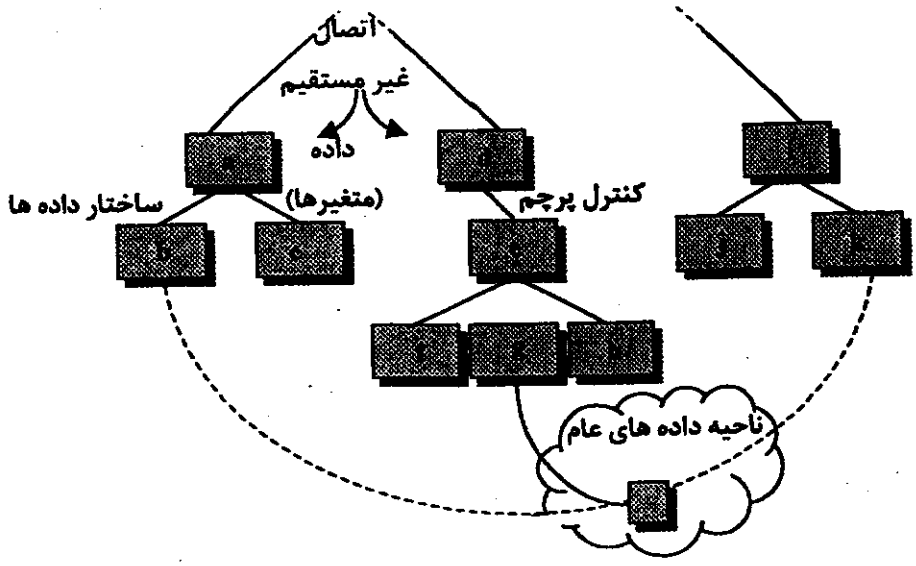
انسجام را می‌توان به‌صورت یک "طیف" نشان داد. ما همیشه تلاش می‌کنیم تا به انسجام و یکپارچگی زیاد دست یابیم. هر چند که دامنهٔ متوسط طیف نیز اغلب قابل قبول است. مقیاس انسجام، غیرخطی است. یعنی آن‌که، انسجام انتهای پایانی به مراتب بدتر از دامنهٔ متوسط (اواسط طیف – م.) است که تقریباً به‌اندازهٔ انسجام انتهای بالایی، قابل قبول می‌باشد. عملاً لزومی ندارد که طراح به طبقه‌بندی انسجام در یک پیمانه خاص بپردازد، بلکه باید مفهوم کلی را درک نموده و هنگام طراحی پیمانه ها بایستی از سطوح پایین انسجام اجتناب کرد.

در انتهای پایینی (نامطلوب) طیف، ما با پیمانه ای روبه‌رو هستیم که مجموعه وظایفی را انجام می‌دهد که یا ربطی به یکدیگر نداشته و یا ارتباط میان آنها ضعیف است. چنین پیمانه هایی "منسجم اتفاقی"[1] نام دارند. پیمانه ای که مجموعه وظایفی را که به‌طور منطقی به‌هم مرتبطند، انجام می‌دهد (مثل پیمانه ای که بدون توجه به نوع، همهٔ خروجی را تولید می‌کند)، "منسجم منطقی"[2] نامیده می‌شود. اگر وجه مشترک وظایف و اعمال موجود در یک پیمانه ، اجرای آنها در مدت زمانی یکسان باشد، چسبندگی یا انسجام پیمانه از " نوع گذرا " یا موقتی[1] است.

به‌عنوان نمونهٔ انسجام کم، پیمانه ای را در نظر بگیرید که فرآیند خطا را برای یک بسته تحلیل مهندسی انجام می‌دهد. وقتی داده‌های محاسبه شده از حد و مرزهای قبلاً تعیین شده فراتر می‌روند، پیمانه احضار می‌شود و کارهای زیر را انجام می‌دهد: (۱) داده‌های مکمل را براساس داده‌های محاسبه شده اولیه، محاسبه می‌کند. (۲) گزارش خطا را (با محتوای گرافیکی) روی ایستگاه کاری کاربر تولید می‌کند. (۳) محاسبات بعدی مورد درخواست کاربر را انجام می‌دهد. (۴) پایگاه داده‌ای را به روز می‌کند. (۵) انتخاب فوری برای فرآیند بعدی، امکان‌پذیر می‌سازد. گرچه وظایف قبلی ارتباط چندانی با یکدیگر ندارند، اما هر یک از آنها موجودیت کاربردی مستقلی است که در یک پیمانه جداگانه به بهترین شکل قابل اجراست. تلفیق این وظایف داخل یک پیمانه منفرد، تنها موجب افزایش احتمال پخش خطا به هنگام اعمال تغییر در یکی از اعمال پردازشی فوق‌الذکر می‌گردد.

1.coincidentally cohesive

2.Logically cohesive



شکل ۱۳ـ۶ انواع اتصال و چسبندگی

سطوح متوسط انسجام از نظر میزان استقلال پیمانه ، نسبتاً به یکدیگر نزدیکند. وقتی عناصر پردازش یک پیمانه به‌هم مرتبط بوده و باید با ترتیب خاصی اجرا شوند، آن‌گاه "انسجام رویه‌ای"[2] به‌وجود می‌آید. هنگامی‌که تمام عناصر پردازشی هر یک حوزهٔ ساختار داده‌ای متمرکز می‌گردند. "انسجام ارتباطی"[1] بروز می‌یابد. شاخص انسجام و یکپارچگی زیاد، پیمانه ای است که یک کار رویه‌ای مجزا را انجام می‌دهد.

همان‌طور که قبلاً هم اشاره کردیم، تعیین میزان دقیق انسجام، ضرورتی ندارد. بلکه تلاش در جهت انسجام بیشتر و تشخیص یکپارچگی کم مهم است، طوری‌که بتوان با تغییر و اصلاح طراحی نرم‌افزار، به استقلال کارکردی بیشتر دست یافت.

۱۳-۵-۳ پیوستگی

پیوستگی، معیاری کیفی است که درجه اتصال یک پیمانه به دیگر پیمانه ها و جهان خارج را مشخص می سازد.

اتصال و پیوستگی، مقیاس ارتباط بین پیمانه ها در ساختار نرم‌افزار است. اتصال به پیچیدگی رابط بین پیمانه ها، نقطه ورود(دخول) و ارجاع به یک پیمانه و نوع داده‌های عبوری از رابط، بستگی دارد. در طراحی نرم‌افزار، تلاش ما در جهت پایین‌ترین سطح ممکن اتصال است. اتصال سادهٔ بین پیمانه ها باعث ایجاد نرم‌افزاری می‌شود که فهم آن ساده‌تر بوده و به‌هنگام وقوع خطاها در یک محل و پخش آنها در سیستم، کمتر در معرض «اثر فزونگر» ایجاد شده قرار دارد. [STE75]

1.temporal cohesion

2.procedural cohesion

شکل ۱۳-۶ نمونه‌هایی از انواع مختلف اتصال پیمانه را ارائه می‌دهد. پیمانه های a و b تابع پیمانه های مختلفی هستند. این دو به یکدیگر مرتبط نبوده و از این‌رو اتصال مستقیم به‌وجود نمی‌آید. پیمانه c تابع پیمانه a بوده و از طریق لیست قراردادی ارگومان که داده‌ها از طریق آن منتقل می‌شوند، قابل دسترسی است. تا مادامی‌که لیست به آرگومان ساده وجود دارد (یعنی داده‌های ساده انتقال یافته و ارتباطی یک به یک بین اقلام برقرار است)، اتصال ضعیف یا (اتصال داده‌ها)[2] در این بخش از ساختار به نمایش گذاشته می‌شود. نوعی اتصال داده‌ها به نام "اتصال استمپ"[3] زمانی ظاهر می‌شود که بخشی از ساختار داده‌ای (به‌جای آرگومان‌های ساده) از طریق رابط پیمانه ، منتقل می‌گردد. این اتصال بین پیمانه های a و b به‌وجود می‌آید.

در سطوح متوسط "اتصال با انتقال کنترل بین پیمانه ها" توصیف می‌گردد. "اتصال کنترل"[4] در اکثر طراحی‌های نرم‌افزاری بسیار رایج است و در شکل ۱۳-۶ با عبور "نشانه نمای کنترل"[5] (متغیری که تصمیمات را در یک پیمانه تابع یا حاکم کنترل می‌کند) بین پیمانه های d و e نشان داده شده است.

در صورت ارتباط پیمانه ها با محیط خارج از نرم‌افزار، سطوح نسبتاً بالای اتصال، به‌وجود می‌آیند. به‌عنوان مثال I/O، پیمانه را به دستگاه‌های خاص، قالب ها و پروتوکل‌های ارتباطاتی، متصل می‌کند. "اتصال خارجی"[6] ضروری است اما باید به تعداد کمی از پیمانه ها محدود شود. اتصال سطح بالا نیز زمانی پدید می‌آید که تعدادی از پیمانه ها به ناحیهٔ سراسری داده‌ها ارجاع می‌کنند" اتصال مشترک" که نام این حالت می‌باشد، در شکل ۱۳-۶ به نمایش درمی‌آید. پیمانه های c, g و k هر کدام به یک عنصرداده‌ای در یک ناحیهٔ سراسری داده‌ها دسترسی دارند (مثل پروندهٔ دیسک یا ناحیه‌ای از حافظه که به‌طور سراسری قابل دستیابی است). پیمانه c به عنصرداده‌ای مقدار اولیه می‌دهد. سپس پیمانه g آن را مجدداً محاسبه کرده و به روز درمی‌آورد. فرض کنیم که خطایی رخ داده و g عنصرداده‌ای را به طرزی نادرست به روز می‌کند. در مراحل بعدی فرآیند، پیمانه k عنصر را می‌خواند، سعی می‌کند آن را پردازش نماید ولی مؤفق نمی‌شود و موجب توقف ناگهانی نرم‌افزار می‌گردد. عامل ظاهری این توقف پیمانه k است، اما علت واقعی پیمانه g می‌باشد. تشخیص مسایل در ساختارهایی با اتصال مشترک فراوان، وقت‌گیر و دشوار است. هر چند، این بدان معنا نیست که کاربرد داده‌های سراسری، الزاماً بد است. بلکه منظور آن است که طراح

1.communicational cohesion
2.data coupling
3.Stamp coupling
4.control coupling
5.Control Flag
6.External coupling

نرم‌افزار بایستی از پیامد های احتمالی "اتصال مشترک"[1] آگاه بوده و برای رویارویی با آنها، محتاط و مراقب باشد.

بالاترین میزان اتصال با عنوان "اتصال محتوا"[2] زمانی بوجود می‌آید که یک پیمانه ، از داده‌ها یا اطلاعات کنترلی موجود در حد و مرز پیمانه ای دیگر، استفاده می‌کند. ثانیاً اتصال محتوایی بهنگام ایجاد شاخه‌هایی در میان یک پیمانه نیز اتفاق می‌افتد. این حالت اتصال، قابل پیشگیری بوده و باید از آن اجتناب کرد.

۱۳-۶ابداعات طراحی برای پیمانه سازی موثر و کارآ

پس از ایجاد ساختار برنامه پیمانه‌ای کردن مؤثر و کارآمد به‌واسطهٔ اجرای مفاهیم طراحی که در ابتدای فصل معرفی شدند، تحقق می‌یابد. ساختار برنامه براساس مجموعه ذهنیت‌های زیر، قابل دست‌کاری است:

۱. "اولین تکرار" ساختار برنامه را بمنظور کاهش اتصال و بهبود انسجام، مورد ارزیابی قرار دهید. پس از ساخت برنامه، می‌توان پیمانه ها را از جهت بهبود استقلال آنها، تفکیک یا ترکیب کرد.

یک " پیمانه تفکیکی"[3] در ساختار نهایی برنامه، تبدیل به دو یا چند پیمانه می‌گردد. " پیمانه ترکیبی"[4] حاصل ترکیب پردازش انسجام شده توسط دو یا چند پیمانه است.

پیمانه تفکیکی اغلب زمانی حاصل می‌شود که پردازش مشترک در دو یا چند پیمانه وجود داشته و بصورت یک پیمانه منسجم و جداگانه، مجدداً قابل تعریف است. هنگامی‌که اتصال در سطح بالا موردنظر می‌باشد، می‌توان گاهی پیمانه ها را برای کاهش انتقال کنترل، ارجاع به داده‌های سراسری و پیچیدگی رابط، ترکیب کرد.

۲. سعی کنید ساختارهایی با گنجایش خروجی زیاد را به حداقل رسانده و همان‌طور که عمق افزایش می‌یابد، برای گنجایش ورودی تلاش کنید. ساختار داخل ابر در تصویر ۱۳-۷ از تجزیهٔ عوامل استفادهٔ مفیدی نمی‌کند. تمامی پیمانه ها، زیر یک تک پیمانه کنترل بطور یکنواخت قرار دارند. بطور کلی، توزیع معقول‌تر کنترل در ساختار سمت راست، نشان داده شده است. این ساختار بیضی شکل بوده و تعدادی لایهٔ کنترل و پیمانه های کاملاً سودمند را در سطوح پایین‌تر نمایش می‌دهد.

۳. حوزه تأثیر یک پیمانه را در محدودهٔ دامنهٔ کنترل همان پیمانه حفظ کنید. حوزهٔ تأثیر[1] یک پیمانه e، به تمام پیمانه هایی اطلاق می‌شود که تحت‌تأثیر تصمیم اتخاذ شده در پیمانه e هستند.

نقل قول

عقیده بر این امر که فنون خوب (طراحی) خلاقیت را محدود می سازند به مثابه این است که بگوییم یک هنر پیشه بدون دانستن جزئیات رفتاری می تواند به شهرت برسد یا آنکه یک موسیقیدان نیازی به دانستن دانش موسیقی ندارد. ماروین زلکوویچ

1.commen coupling
2.content coupling
3.exploded modules
4.imploded modules

دامنهٔ کنترل پیمانه e، همه پیمانه هایی هستند که به‌صورت مستقیم یا با واسطه تابع پیمانه e می‌باشند. طبق تصویر ۱۳-۷ اگر تصمیم اتخاذ شده در پیمانه e بر پیمانه r تأثیر بگذارد، ذهنیت ۳ نقض شده، زیرا پیمانه r در حوزهٔ کنترل پیمانه e نمی‌باشد.

۴. رابطهای پیمانه را به‌منظور کاهش پیچیدگی و حشو و بهبود سازگاری مورد ارزیابی قرار دهید. پیچیدگی رابط پیمانه ، عامل عمدهٔ خطاهای نرم‌افزاری است. رابطها باید برای انتقال راحت اطلاعات طراحی شده و با وظیفهٔ پیمانه هماهنگ و سازگار باشند. ناسازگاری رابط (یعنی داده‌های ظاهراً نامربوط که از طریق لیست آرگومان یا تکنیک دیگری منتقل می‌شوند) نشانهٔ انسجام و یکپارچگی است. پیمانه موردنظر بایستی مجدد مورد ارزیابی واقع شود.

۵. پیمانه هایی را تعریف کنید که کارشان قابل پیش‌بینی است اما از پیمانه های بسیار محدودکننده اجتناب نمایید. یک پیمانه زمانی قابل پیش‌بینی است که به‌صورت یک جعبهٔ سیاه تلقی شود، یعنی، داده‌های یکسان خارجی بدون در نظر داشتن جزییات داخلی پردازشی، تولید خواهند شد.[۲] پیمانه هایی که حافظهٔ داخلی دارند، می‌توانند غیرقابل پیش‌بینی باشند، مگر آن‌که در کاربرد آنها دقت شود.

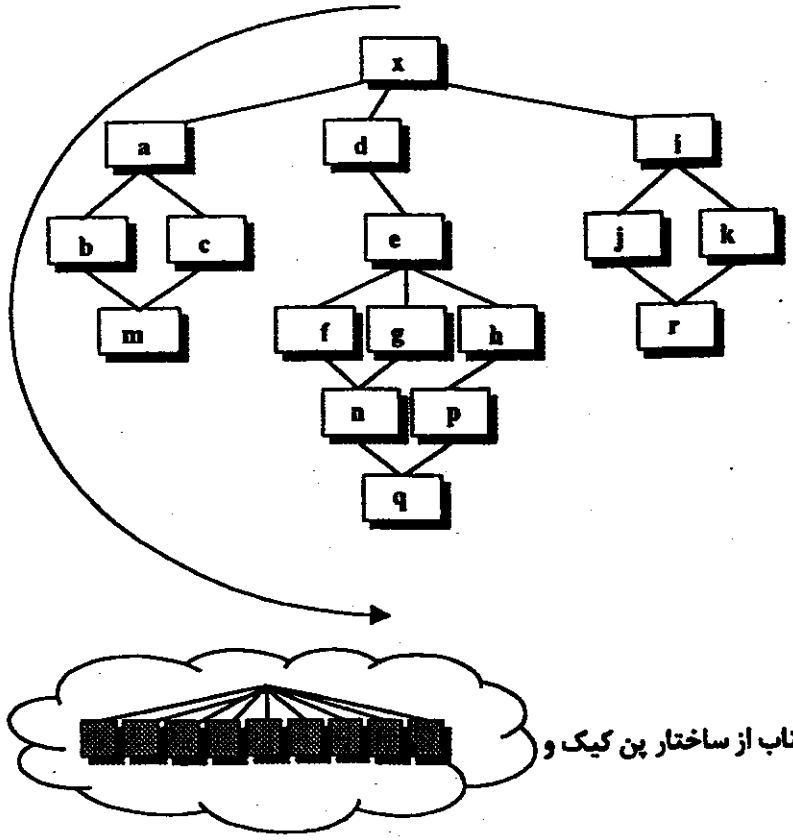
پیمانه ای که پردازش را فقط بر یک عمل فرعی محدود می‌کند، انسجام زیادی را نشان می‌دهد و طراح نسبت به آن تمایل دارد. با این وجود آن پیمانه ای که به‌طور دلخواه اندازهٔ ساختار داده‌ای محلی را و گزینه های جریان کنترل یا حالات رابط خارجی را محدود می‌کند، همواره مستلزم مراقبت و نگهداری خواهد بود تا چنین محدودیت‌هایی رفع گردند.

۶. برای پیمانه هایی با "ورودی کنترل شده" تلاش نموده و از "اتصالات نامعقول" خودداری کنید. این ذهنیت طراحی، دربارهٔ اتصال محتوایی، هشدار می‌دهد. در صورت محدودسازی و کنترل رابط های پیمانه ، درک و در نتیجه نگهداری و تعمیر نرم‌افزار ساده‌تر می‌گردد. اتصال نامعقول[۳]، بر شاخه‌ها (انشعابها) یا مراجع موجود در میان پیمانه اشاره می‌کند.

1.scope of effect

۲. یک پیمانه " جعبه سیاه " تجریدی پیمانه ای است.

3.pathological connection



شکل ۱۳ـ۷ ساختارهای برنامه

۱۳-۷ مدل طراحی

اصول و مفاهیم مورد بحث طراحی در این فصل، مبنایی را برای ایجاد یک مدل طراحی فراهم می‌کنند که این مدل، بازنمایی داده‌ها، معماری، رابطها و اجزاء را در بردارد. مانند مدل تحلیلی قبل از آن، هر یک از این نمودهای طراحی با سایر بازنمایی‌ها در ارتباط بوده و ردیابی تمامی آنها به ضروریات و نیازمندیهای نرم‌افزار برمی‌گردد.

در شکل ۱۳-۱، مدل طراحی به شکل یک هرم نشان داده شد. نمادگری این شکل اهمیت دارد. هرم شی بسیار ثابتی با قاعدهٔ پهن و مرکز ثقل پایین است. ما قصد داریم مانند هرم یک طراحی نرم‌افزاری ثابت را ایجاد کنیم. با کاربرد طراحی داده‌ها در قاعدهٔ وسیع هرم، ساخت یک بخش میانی پایدار با استفاده از طراحی معماری و رابط و اعمال طراحی اجزا در قلهٔ نوک تیز هرم، می‌توانیم مدلی از طراحی ایجاد کنیم که به آسانی و به‌واسطهٔ هر تغییری واژگون نگردد.

شیوه‌هایی که منجر به ایجاد مدل طراحی می‌گردند، در فصول ۱۴، ۱۵، ۱۶ و ۲۲ (برای سیستم‌های شی‌گرا)، ارائه شده‌اند. هر یک از این شیوه‌ها، به طراح امکان می‌دهند تا بر اساس مفاهیم اصلی و مؤثر در کیفیت بالای نرم‌افزار، طراحی ثابت و پایداری را ایجاد نماید.

۱۳-۸ مستندسازی طراحی


مشخصات طراحی نرم افزار

"مشخصات طراحی"[1] جنبه‌های مختلف مدل طراحی را دربرداشته و هنگام پالایش نمایش نرم‌افزار توسط طراح، تکمیل می‌گردد. ابتدا، حوزهٔ کلی کار طراحی توصیف می‌گردد. اغلب اطلاعات ارائه شده در این قسمت، از مشخصات سیستم[2] و مدل تحلیلی (مشخصات نیازمندی‌های نرم‌افزار)[3] مشتق می‌شود. سپس، طراحی داده‌ها تعیین می‌گردد. ساخت پایگاه داده‌ای، هرگونه ساختار فایل خارجی، ساختارهای داخلی داده‌ها و یک ارجاع متقابل که اشیاء داده‌ای را به فایل‌هایی خاص مرتبط می‌کند، همگی تعریف و مشخص می‌شوند. طرح معماری بیانگر چگونگی اشتقاق معماری برنامه از مدل تحلیلی است. به‌علاوه، نمودارهای ساختار، برای نمایش سلسله مراتب پیمانه به‌کار می‌روند.

طراحی رابط‌های خارجی و داخلی برنامه، نمایش داده شده است و طراحی جزیی رابط بین انسان و ماشین توصیف می‌گردد. در برخی موارد، مدل جزیی و مفصل GUI ممکن است نشان داده شود.

اجزاء ب عناصر جداگانه نشانی‌پذیر نرم‌افزار مانند زیر روال‌ها، اعمال یا رویه‌ها ـ در ابتدا با یک شرح پردازشی زبان انگلیسی (یا فارسی – م.) توصیف می‌گردند. شرح پردازشی، عمل رویه‌ای جزء (پیمانه) را توضیح می‌دهد. بعداً ابزار طراحی رویه‌ای برای تبدیل این شرح به یک توصیف ساخت یافته، به‌کار می‌رود.

مشخصات طراحی، شامل ارجاع متقابل نیازمندیها می‌باشد. هدف این ارجاع متقابل (که معمولاً به‌صورت یک ماتریس ساده به نمایش درمی‌آید) عبارت است از: (۱) اثبات این‌که تمامی نیازمندیها در طراحی نرم‌افزار مرتفع شده‌اند. (۲) تعیین اجزای حیاتی و مهم در اجرای نیازمندیها و ضرورت‌های خاص.

اولین مرحله در توسعهٔ مستندسازی آزمون، در مستندسازی طراحی نیز موجود است. پس از ایجاد ساختار برنامه و رابط‌ها، می‌توانیم رهنمودهایی را برای آزمون پیمانه های جداگانه و یکپارچگی کل بسته (Package)، توسعه دهیم. در برخی موارد، مشخصات مفصل مراحل آزمون به موازات طراحی، وجود دارد. در چنین مواردی، ممکن است این بخش از مشخصات طراحی حذف شود.

محدودیت‌های طراحی، مثل محدودیت‌های حافظهٔ حافظی یا ضروریات یک رابط تخصصی خارجی، ممکن است ضرورت‌های خاصی را برای هم‌گذاری یا بسته‌بندی نرم‌افزار، تعیین کند. ملاحظات خاص ناشی از ضرورت جایگذاری برنامه، مدیریت حافظهٔ مجازی، پردازش با سرعت بالا و عوامل دیگر ممکن است باعث تغییراتی در طراحی حاصل از گردش یا ساختار اطلاعاتی گردد. به‌علاوه، این بخش شیوهٔ به‌کار رفته در انتقال نرم‌افزار به سایت مشتری را توصیف می‌کند.

آخرین بخش مشخصات طراحی، داده‌های مکمل را در بردارد. توصیفات الگوریتمی، رویه‌های دیگر، داده‌های جدولی، قطعاتی از سایر اسناد و دیگر اطلاعات مربوطه به‌صورت یک تبصرهٔ خاص یا یک پیوست

1.design specification

2.system specification

3.software requirements specification

جداگانه، ارائه می‌شوند. توسعهٔ "عملیات مقدماتی/ راهنمای نصب"[1] و الحاق آن به‌عنوان یک پیوست، ضمیمه به مستندات طراحی، عاقلانه و صحیح به‌نظر می‌رسد.

۱۳-۹ خلاصه

طراحی، هستهٔ فنی مهندسی نرم‌افزار است. در طی طراحی، پالایش‌های پیش‌روندهٔ ساختار داده‌ها، معماری، رابطها و جزییات رویه‌ای اجزاء نرم‌افزاری، توسعه یافته، بررسی شده و توضیح داده می‌شوند. طراحی نمایش‌هایی از نرم‌افزاری است که از نظر کیفی قابل ارزیابی است.

در طول چهار دههٔ گذشته، برخی اصول و مفاهیم اساسی طراحی نرم‌افزار پیشنهاد شده‌اند. اصول طراحی در حین پیشروی فرآیند طراحی، مهندس نرم‌افزار را راهنمایی می‌کنند. مفاهیم طراحی، معیارهای اصلی کیفیت طراحی را فراهم می‌نمایند. پیمانه ای کردن (هم در برنامه و هم داده‌ها) و مفهوم انتزاع، به طراح امکان می دهد تا اجزاء نرم‌افزاری را ساده کرده و مجدداً از آنها استفاده نماید. پالایش، مکانیزمی را برای نمایش لایه‌های توالی جزییات کاربردی، ایجاد می‌کند. ساختار برنامه و داده‌ها، دید کلی از معماری نرم‌افزار را به‌دست داده و رویه، جزییات لازم برای اجرای الگوریتم را تأمین می‌نماید. اختفای اطلاعات و استقلال کارکردی، ذهنیاتی برای دستیابی به پیمانه ای شدن کارآمد می‌باشند.

ما بحث خود دربارهٔ مبانی و اصول طراحی را با کلمات گلنفورد مایرز [MYE78][2] به پایان می‌بریم:

"ما سعی می‌کنیم مسئله را با تعجیل در فرآیند طراحی حل کنیم، تا در پایان پروژه، زمان کافی برای کشف خطاها باقی بماند، چرا که ما در فرآیند طراحی عجله کردیم ..."

نتیجهٔ اخلاقی این است که در فرآیند طراحی عجله نکنید؛ طراحی ارزش تلاش را دارد.

ما بحث طراحی را به اتمام نرسانده‌ایم. در فصل‌های بعد، روش‌های طراحی مورد بحث قرار می‌گیرند. این روش‌ها در تلفیق با اصول این فصل، مبنایی را برای بررسی کامل طراحی نرم‌افزار فراهم می‌آورند.

1.Preliminary Operations /Installation

2.Myers, G.

مسایل و نکاتی برای تفکر و تعمق بیشتر

۱۳-۱ آیا شما هنگام نوشتن برنامه، نرم‌افزار را طراحی می‌کنید؟ چه چیز طراحی نرم‌افزار را از برنامه‌نویسی متمایز می‌کند؟

۱۳-۲ سه اصل طراحی دیگر را برای افزودن به موارد ذکر شده در بخش ۱۳-۳، توسعه دهید.

۱۳-۳ نمونه‌هایی از سه انتزاع داده‌ای و انتزاع‌های رویه‌ای برای دست‌کاری آنها، ارائه کنید.

۱۳-۴ جهت گسترش سه سطح متفاوت از انتزاع رویه‌ای برای یک یا چند برنامهٔ زیر، یک "شیوهٔ پالایشی گام به گام" را به‌کار برید.

الف - یک نگارندهٔ کنترلی را توسعه دهید که با دادن مقدار عددی دلار، آن مقدار را با کلماتی که به‌طور معمول در یک بررسی ضرورت دارند، چاپ خواهد کرد.

ب - برای جذرگیری یک معادلهٔ غیرجبری، از حل تکراری استفاده کنید.

پ - یک الگوریتم زمان‌بندی ساده با گردش نوبت را برای سیستم عامل گسترش دهید.

۱۳-۵ آیا موردی هست که در آن معادلهٔ (۱۳-۲) صدق نکند؟ چنین موردی چگونه ممکن است در استدلال پیمانه ای کردن تأثیر بگذارد؟

۱۳-۶ چه وقت یک طراحی پیمانه ای باید به‌صورت نرم‌افزار یکپارچه اجرا گردد؟ انجام این کار چگونه امکان دارد؟ آیا عملکرد تنها توجیه اجرای نرم‌افزار یکپارچه است؟

۱۳-۷ حداقل پنج سطح انتزاع را برای یکی از مسایل نرم‌افزاری زیر توسعه دهید:

الف- یک بازی ویدیویی به انتخاب خودتان

ب - یک بسته تبدیل سه‌بعدی برای برنامه‌های کاربردی گرافیکی کامپیوتر

پ - مفسر زبان برنامه‌نویسی

ت - کنترل‌کننده روبات با دو درجه آزادی

ث - سر مسئله‌ای که به‌طور متقابل برای شما و مربی‌تان، قابل قبول است.

با کاهش سطح انتزاع، توجه شما ممکن است محدود گردد به‌نحوی‌که در سطح پایانی (که منبع) تنها توصیف یک کار لازم است.

۱۳-۸ مقالهٔ اصلی Parnas [PAR72] را مطالعه کرده و نمونهٔ نرم‌افزاری را که او برای توضیح و نمایش تجزیهٔ سیستم به پیمانه ها به‌کار می‌برد، خلاصه کنید. اختفای اطلاعات چگونه برای انجام تجزیه، مورد استفاده قرار می‌گیرد؟

۱۳-۹ رابطهٔ بین مفهوم اختفای اطلاعات را به‌عنوان ویژگی پیمانه ای شدن مؤثر و مفهوم استقلال پیمانه، مورد بحث قرار دهید.

۱۳-۱۰ برخی تلاش‌های اخیر توسعه نرم‌افزاری را مورد بررسی قرار داده و هر پیمانه را درجه‌بندی کنید (در مقیاس ۱ پایین تا ۷ بالا)، نمونه‌هایی از بهترین و بدترین کار خود را گزارش دهید.

۱۳-۱۱ تعدادی از زبان‌های برنامه‌نویسی سطح بالا، رویهٔ داخلی را به‌صورت ساخت پیمانه ای، حمایت می‌کنند. چگونه این ساخت بر اتصال و اختفای اطلاعات تأثیر می‌گذارد.

۱۳-۱۲ مفاهیم اتصال و قابلیت انتقال نرم‌افزار چه ارتباطی با یکدیگر دارند؟ برای تأیید بحث خود، نمونه‌هایی را ارائه دهید.

۱۳-۱۳ چگونگی کمک تقسیم‌بندی ساختاری به قابلیت نگهداری بیشتر نرم‌افزار را مورد بحث قرار دهید.

۱۳-۱۴ هدف از توسعهٔ ساختار برنامه با تجزیهٔ عوامل چیست؟

۱۳-۱۵ مفهوم اختفای اطلاعات را به بیان خودتان توصیف کنید.

۱۳-۱۶ به چه دلیل حفظ حوزه تأثیر یک پیمانه در محدودهٔ دامنهٔ کنترل آن، عقیدهٔ خوبی به‌شمار می‌رود؟

فهرست منابع و مراجع

[AH083] Aho, A.V., J. Hopcroft, and J. Ullmann, *Data Structures and Algorithms*, Addison-Wesley, 1983.

[BAS98] Bass, L., P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.

[BEL8I] Belady, L., Foreword to *Software Design: Methods and Techniques* (L.J. Peters, author), Yourdon Press, 1981.

[BR098] Brown, W.J., et al., *Anti-Patterns*, Wiley, 1998.

[BUS96] Buschmann, F. et al., *Pattern-Oriented Software Architecture*, Wiley, 1996.

[DAH72] Dahl. 0., E. Dijkstra, and C. Hoare, *Structured Programming*, Academic Press, 1972.

[DAV95] Davis, A., *201 Principles of Software Development*, McGraw-Hili, 1995.

[DEN73] Dennis, *J.*, "Modularity," in *Advanced Course on Software Engineering* (F.L. Bauer, ed.), Springer-Verlag, New York, 1973, pp. 128-182.

[GAM95] Gamma, E. et al., *Design Patterns*, Addison-Wesley, 1995.

[GAN89] Gonnet, G., *Handbook of Algorithms and Data Structures*, 2nd ed., Addison-Wesley, 1989.

[GAR95] Garlan, D. and M. Shaw, "An Introduction to Software Architecture," *Advances in Software Engineering and Knowledge Engineering*, vol. I (V. Ambriola and G. Tortora, eds.), World Scientific Publishing Company, 1995.

[JAC75] Jackson, M.A., *Principles of Program Design*, Academic Press, 1975.

[JAC83] Jackson, M.A., *System Development*, Prentice-Hall, 1983.

[JAC92] Jacobson, I., *Object-Oriented Software Engineering*, Addison-Wesley, 1992.

[KAI83] Kaiser, S.H., *The Design of Operating Systems for Small Computer Systems*, Wiley-Interscience, 1983, pp. 594 ff.

[KRU84] Kruse, R.L., *Data Structures and Program Design*, Prentice-Hall, 1984.

[MCG91] McGlaughlin, R., "Some Notes on Program Design," *Software Engineering Notes*, vol. 16, no. 4, October 1991, pp. 53-54.

[MEY88] Meyer, B., *Object-Oriented Software Construction*, Prentice-Hall, 1988.

[MIL72] Mills, H.D., "Mathematical Foundations for Structured Programming," Technical Report FSC 71-6012, IBM Corp., Federal Systems Division, Gaithersburg, Maryland, 1972.

[MYE78] Myers, G., *Composite Structured Design*, Van Nostrand,1978.

[ORR77] Orr, K.T., *Structured Systems Development*, Yourdon Press, 1977.

[PAR72] Parnas, D.L., "On Criteria to Be Used in Decomposing Systems into Modules," *CACM*, vol. 14, no. I, April 1972, pp. 221-227.

[ROS75] Ross, D.,]. Goodenough, and C. Irvine, "Software Engineering: Process, Principles and Goals," *IEEE Computer*, vol. 8, no. 5, May 1975.

[SHA95a] Shaw, M. and D. Garlan, "Formulations and Formalisms in Software Architecture," *Volume /ODD-Lecture Notes in Computer Science*, Springer-Verlag, 1995.

[SHA95b] Shaw, M. et al., "Abstractions for Software Architecture and Tools to Support Them," *IEEE Trans. Software Engineering*, vol. SE-2I, no. 4, April 1995, pp. 314-335.

[SHA96] Shaw, M. and D. Garlan, *Software Architecture*, Prentice-Hall, 1996.

[SOM89] Sommerville, I., *Software Engineering*, 3rd ed., Addison-Wesley, 1989.

[STE74] Stevens, W., G. Myers, and L. Constantine, "Structured Design," *IBM Systemsjournal*, vol. 13, no. 2,1974, pp. 115-139.
[WAR74] Warnier,J., *Logical Construction of Programs*, Van Nostrand-Reinhold, 1974.
[WAS83] Wasserman, A., "Information System Design Methodology," in *Software Design Techniques* (P. Freeman and A. Wasserman, eds.). 4th ed., IEEE Computer Society Press, 1983, p. 43.
[WIR7 I] Wirth, N., "Program Development by Stepwise Refinement," *CACM*, vol. 14, no.4,197I,pp.221-227.
[YOU79] Yourdon, E., and L. Constantine, *Structured Design*, Prentice-Hall, 1979.

خواندنیهای دیگر و منابع اطلاعاتی

Donald Norman has written two books *(The Design of Everyday Things*, Doubleday, 1990, and *The Psychology of Everyday Things*, HarperCollins, 1988) that have become classics in the design literature and "must" reading for anyone who designs anything that humans use. Adams *(Conceptual Blockbusting*, 3rd ed., Addison-Wesley, 1986) has written a book that is essential reading for designers who want to broaden their way of thinking. Finally, a classic text by Polya *(How to Solve It*, 2nd ed., Princeton University Press, 1988) provides a generic problem-solving process that can help software designers when they are faced with complex problems.

Following in the same tradition, Winograd-et al. *(Bringing Design to Software*, Addison-Wesley, 1996) discusses software designs that work, those that don't, and why.

A fascinating book edited by Wixon and Ramsey *(Field Methods Casebook for Software Design*, Wiley, 1996) suggests field research methods (much like those used byanthropologists) to understand how end-users do the work they do and then design software that meets their needs. Beyer and Holtzblatt *(Contextual Design: A Customer-Centered Approach to Systems Designs*, Academic Press, 1997) offer another view of software design that integrates the customer/user into every aspect of the software design process.

McConnell *(Code Complete*, Microsoft Press, 1993) presents an excellent discussion of the practical aspects of designing high-quality computer software. Robertson *(Simple Program Design*, 3rd ed., Boyd and Fraser Publishing, 1999) presents an introductory discussion of software design that is useful for those beginning their study of the subject.

An excellent historical survey of important papers on software design is contained in an anthology edited by Freeman and Wasserman *(Software Design Techniques*, 4th ed., IEEE, 1983). This tutorial reprints many of the classic papers that have formed the basis for current trends in software design. Good discussions of software design fundamentals can be found in books by Myers [MYE78], Peters *(Software Design: Methods and Techniques*, Yourdon Press, 1981), Macro *(Software Engineering: Concepts and Management*, Prentice-Hall, 1990), and Sommerville *(Software Engineering*, Addison-Wesley, 5th ed., 1996).

Mathematically rigorous treatments of computer software and design fundamentals may be found in books by Jones *(Software Development: A Rigorous Approach*, Prentice-Hall, 1980), Wulf *(Fundamental Structures of Computer Science*, Addison-Wes-

ley, 1981), and Brassard and Bratley *(Fundamental of Algorithmics,* Prentice-Hall, 1995). Each of these texts helps to supply a necessary theoretical foundation for our understanding of computer software.

Kruse *(Data Structures and Program Design,* Prentice-Hall, 1994) and Tucker et al. *(Fundamentals of Computing II: Abstraction, Data Structures, and Large Software Systems,* McGraw-Hili, 1995) present worthwhile information on data structures. Measures of design quality, presented from both the technical and management perspectives, are considered by Card and Glass *(Measuring Software Design Quality,* Prentice-Hall,1990).

A wide variety of information sources on software design and related subjects is available on the Internet. An up-to-date list of World Wide Web references that are relevant to design concepts and methods can be found at the SEPA Web site: **http://www.mhhe.com/engcs/compsci/pressman/resources/design -principles.mhtml**