

فرآیند

فصل ۲

مفاهیم کلیدی (مرتب بر حروف الفبا)

ابزارهای نسل چهارم (4GT)، توسعه سریع برنامه ها (RAD)، توسعه مبتنی بر اجزاء، توسعه همروند، چارچوب مشترک فرآیند، سطوح بلوغ فرآیند، شیوه های رسمی، فعالیتهای نگهداری، مدل های فرآیند تکاملی، مهندسی نرم افزار، نمونه سازی

KEY CONCEPTS

common process framework , component-based development , Concurrent development , evolutionar, process models , formal methods , 4GT , maintenance activities, prIKess maturity levels , prototyping , RAD , software engineering

نگاه اجمالی

فرآیند چیست؟ هنگامی که شما تولید یک محصول یا سیستم را مدنظر دارید، نکته مهم این است که وارد یک سری مراحل قابل پیش بینی شوید. نقشه ای که به شما کمک می کند، به نتیجه ای سریع و درخشان دست یابید. نقشه ای که شما پیگیری می کنید، یک فرآیند نرم افزاری نامیده می شود.

چه کسی این کار را انجام می دهد؟ مهندسین نرم افزار و مدیران آنها، این فرآیند را مطابق نیازهای خود ساخته و سپس آن را دنبال می کنند. علاوه بر آن، مردمی که متقاضی این نرم افزار هستند، نقش بزرگی در فرآیند نرم افزاری دارند.

چرا مهم است؟ زیرا ثبات، کنترل و سازماندهی را برای یک فعالیت، فراهم می نماید و در غیر این صورت خارج از کنترل بوده و به یک هرج و مرج تبدیل می شود.

مراحل کار کدام است؟ فرآیندی که شما انتخاب می کنید دقیقاً به نرم افزاری که تولید می کنید، بستگی دارد. ممکن است که یک فرآیند برای تولید سیستم الکترونیکی هواپیما مناسب باشد درحالی که فرآیندی کاملاً متفاوت، برای ایجاد سایت وب استفاده می شود.

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 فرایندها و مکانیزمهای ارزیابی نرم افزار وجود دارد که سازمانها را قادر می سازد «تکامل فرآیند نرم افزاری» خود را تعیین کنند.

کیفیت، بدون زمان بودن، و کارایی درازمدت محصول شما بهترین شاخصهای کارایی فرآیند مورد استفاده شما هستند.

اما از دیدگاه فنی یک فرآیند نرم افزاری^۱ دقیقاً چیست؟ در مطالب این کتاب، ما فرآیند نرم افزاری را بعنوان چارچوب کاری تعریف می‌کنیم که برای ایجاد یک نرم افزار دارای کیفیت مطلوب لازم است. آیا مفهوم فرآیند با مهندسی نرم افزار یکسان است؟ پاسخ هم «بله» و هم «خیر» است. فرآیند نرم افزاری روشی را تعریف می‌کند که در هنگام طراحی نرم افزار بکار گرفته می‌شود. اما مهندسی نرم افزار نیز دربرگیرنده فناوریهای است که در فرآیند وجود دارند یعنی روشهای فنی و ابزار خودکار.

مهمتر اینکه، مهندسی نرم افزار بوسیله افراد مبتکر و مطلع صورت می‌گیرد که در چارچوب یک فرآیند معین و کامل کار می‌کنند که برای محصولی که تولید می‌کنند و نیازهای بازار مناسب است. هدف از این بخش مهیا ساختن بررسی وضعیت کنونی فرآیند نرم افزار و مهیا کردن نکاتی برای بحث دقیق‌تر در مورد مدیریت و عناوین فنی است که بعداً در این کتاب ارائه خواهند شد.

در کتاب جالبی که نظرات یک اقتصاددان را درمورد نرم افزار و مهندسی نرم افزار بیان می‌کرد، هاولد با تاجر [BAE98]^۲ درمورد فرآیند نرم افزاری بیان می‌دارد:

از آنجا که نرم افزار چون هر سرمایه دیگری، یک دانش نهفته است و از آنجا که دانش، پراکنده، تلویحی، نهفته و در مقیاس بزرگ ناقص است، تولید نرم افزار یک فرآیند یادگیری اجتماعی است. فرآیند، گفتگویی است که در آن دانشی گردآوری شده که باید نرم افزار شود. این فرآیند، ارتباطی دو سویه و محاوره‌ای بین کاربران و طراحان، بین کاربرد ابزار مورد استفاده و بین طراح و ابزار مورد استفاده او T[فناوری]، برقرار می‌سازد. این یک فرآیند تکرار شونده است که در آن خود ابزار مربوطه، بعنوان رابطی برای برقراری ارتباط می‌باشد و هر دور جدید گفتگو باعث دریافت بیشتری از دانش افراد درگیر در کار می‌شود.

در واقع تولید نرم افزار کامپیوتری یک فرآیند یادگیری تکرار شونده است و حاصل آنچه که باجر «سرمایه نرم افزاری» نام نهاده، قرار دادن دانش جمع‌آوری شده، گزینش و سازماندهی آن به عنوان فرآیند در دست اقدام است.

نقل قول

مهندسی بیشتر از آنکه یک دبسلین دانش و معلومات باند یک فعل است، راهی است عملی برای رفع و رجوع مشکلات و حل مسائل. اسکاوت وایت مایر

۱-۲ مهندسی نرم افزار : یک فناوری لایه ای

گرچه صدها نویسنده تعاریف شخصی از مهندسی نرم افزار^۳ ارائه داده‌اند اما تعریف فریتز بوئر [NAU69]^۱ در کنفرانس نیم سال درمورد این موضوع هنوز اساس بحث و گفتگو قرار داد:

1. software process

2. Baetjer, jr.

3. software engineering

[مهندسی نرم‌افزار عبارتست از] ایجاد و استفاده از اصول ساده مهندسی به منظور رسیدن به یک نرم‌افزار مقرون به صرفه که قابل اطمینان بوده و روی دستگاه‌های واقعی کارآمد باشد.

تقریباً هر خواننده‌ای سعی دارد چیزی به این تعریف بیافزاید. این تعریف چیز کمی در مورد جنبه‌های فنی کیفیت نرم‌افزار می‌گوید و مستقیماً نیاز به رضایت مشتری یا تحویل به موقع آن را مورد خطاب قرار نمی‌دهد. این تعریف اهمیت ارزیابی و متریک را حذف می‌کند و بیانگر میزان اهمیت فرایند تکمیل شده نیست. و هنوز تعریف بوئر خط مبنای کار ماست. اصول ساده مهندسی که می‌توان آنها را در ارائه نرم‌افزار کامپیوتری بکار گرفت کدامند؟ چگونه می‌توانیم بصورت مقرون به صرفه نرم‌افزاری بسازیم که مورد اطمینان باشد؟ چه چیزهایی برای خلق برنامه‌های کامپیوتری لازم است که بصورت کارآمد نه تنها روی یک دستگاه بلکه چند دستگاه واقعی کارکنند؟ این سوالات، سوالاتی هستند که مهندسين کامپیوتر را در قسمت نرم‌افزار به چالش فرا می‌خوانند.



مهندسی نرم‌افزار را چگونه تعریف کنیم؟

IEEE^۱ تعریف جامع‌تری ارائه داده که می‌گوید: [IEE93]

مهندسی نرم‌افزار: ۱- بکارگیری یک روش سیستماتیک، منظم و کمیت پذیر برای ارائه، عملیات و حفظ و نگهداری نرم‌افزار است یعنی بکارگیری مهندسی و شیوه‌های آن در نرم‌افزار ۲- بررسی رهیافتهای آورده شده در قسمت (۱).

۱-۱-۲ فرآیند، شیوه‌ها و ابزارها

مهندسی نرم‌افزار یک فناوری لایه لایه است. با رجوع به شکل ۱-۲، هرگونه رهیافت مهندسی (از جمله مهندسی نرم‌افزار) باید به یک تعهد سازمانی در مورد کیفیت تکیه داشته باشد.

اساس مهندسی نرم‌افزار یک لایه به نام فرآیند^۲ است. فرآیند مهندسی نرم‌افزار مانند چسبی است که لایه‌های فناوری را با هم نگاه داشته، توسعه به موقع و منطقی نرم‌افزار کامپیوتری را ممکن می‌سازد. فرآیند چارچوبی برای مجموعه‌ای از حوزه‌های کلیدی و اصلی فرآیند خود مهیا می‌سازد که باید برای تحویل مؤثر فناوری مهندسی نرم‌افزار بوجود آید.

حوزه‌های اصلی پردازش [PAU93]^۳ (KPAs)^۴ مبنای کنترل مدیریت پروژه‌های نرم‌افزاری را تشکیل داده و بستری فراهم می‌کنند که در آن روش‌های فنی بکار گرفته می‌شود. محصولات تولید می‌شوند، (مدلها، اسناد، داده‌ها، گزارشات، فرمها و غیره)، محکمایی بوجود می‌آیند، کیفیت سنجیده می‌شود و تغییرات احتمالی ترتیب داده می‌شوند.

1. Naur, P. and B.

2. IEEE Standards Collection : Software Engineering , IEEE Standard 610.12-1990, IEEE, 199


3. process layer

4. Paulk, M.

5. key process areas

روشهای^۱ مهندسی نرم افزار چگونگی انجام کار را از نظر فنی برای ساخت نرم افزار مهیا می کنند. روشها دربر گیرنده طیف وسیعی از مشاغل هستند که شامل نیاز به تجزیه و تحلیل، طراحی، ساختار برنامه، آزمون و پشتیبانی است. روشهای مهندسی نرم افزار به مجموعه ای از اصول اولیه تکیه دارند که بر هر حوزه فناوری حاکم بوده و شامل فعالیتهای مدل سازی و دیگر فنون توصیفی است.

ابزارهای مهندسی نرم افزار^۲ پشتیبانی تمام خودکار یا نیمه خودکار را برای فرآیند و روشها مهیا می سازند. وقتی ابزارها منسجم می شوند بطوریکه اطلاعاتی که بوسیله یک ابزار حاصل می شود را می توان توسط ابزار دیگر مورد استفاده قرار داد، آن وقت سیستمی بوجود می آید که مهندسی نرم افزار به کمک کامپیوتر نامیده می شود (CASE).^۳


مهندسی نرم افزار مشتمل بر فرآیند، مدیریت، و روشهای تکنیکی و ابزار می باشد




شکل ۱-۲ لایه های مهندسی نرم افزار

۲-۱-۲ یک دید کلی از مهندسی نرم افزار

مهندسی عبارتست از تحلیل، طراحی، ساخت، تأیید و مدیریت امور فنی (یا اجتماعی). بدون توجه به ماهیت چیزی که مورد کارهای مهندسی قرار می گیرد، سوالات زیر را باید پرسید و پاسخ گرفت:

- مشکلی که باید حل شود چیست؟
- مشخصه های چیزی که مشکل را حل می کند چه می باشد؟
- چگونه باید آن را شناخت؟
- چگونه این ماهیت رفع کننده مشکل ایجاد می شود؟
- چه روشی باید برای مشخص شدن مشکلات استفاده شود یعنی مشکلاتی که در طراحی بوجود آمده و همینطور چه روشی برای ساخت ماهیت رفع کننده مشکل استفاده می شود؟
- این وسیله چگونه در طولانی مدت پشتیبانی می شود وقتی که اطلاعات، تطبیقات و بهبود کارها از سوی استفاده کننده این ماهیت مورد درخواست می باشد؟


ارجاع به وب
روستاک مجله ای است که راهکارها و توصیه های عملی مهندسی نرم افزار را ارائه می کند. این مجله به طور لکترونیکی در سایت www.Stsc.hill.af.mil راهم است.

1. methods

2. software engineering tools

3. computer-aided software engineering

در سراسر این کتاب ما تنها روی یک چیز تأکید داریم و آن هم نرم‌افزار کامپیوتری است. باید فرایند مهندسی نرم‌افزار برای یک مهندس به خوبی تعریف شود. در این بخش، مشخصه‌های کلی فرایند نرم‌افزار در نظر گرفته شده‌اند. بعدها در این فصل، مدل‌های خاصی از فرایند مورد بررسی قرار می‌گیرند.

کارهای مربوطه به مهندسی نرم‌افزار را می‌توان به سه گروه کلی تقسیم کرد بدون توجه به حوزه کاربرد، اندازه پروژه یا پیچیدگی آن. هر مرحله یک یا چند سوال فوق را مورد خطاب قرار می‌دهد.

مرحله تعریف^۱ روی آنچه که این کارها هستند (what)، متمرکز می‌شود. یعنی در طول تعریف مهندس نرم‌افزار سعی می‌کند نوع اطلاعاتی را که باید پردازش شوند، عملکرد مطلوب، نوع وضعیت سیستم مورد انتظار، نوع رابط‌هایی که باید ایجاد شوند. محدودیتهای موجود در طرح و معیارهای صحت را که برای تعریف یک سیستم موفق لازمند، شناسایی کند. نکات اصلی مورد نیاز در سیستم و نرم‌افزار شناسایی شده‌اند. گرچه روشهای بکار گرفته شده در طول مرحله تعریف، بسته به معیار مهندسی نرم‌افزاری که بکار رفته بسیار متفاوتند اما سه کار اصلی در بعضی موارد به وقوع می‌پیوندد: مهندسی سیستم یا اطلاعات (فصل ۱۰)، طراحی پروژه نرم‌افزاری (فصول ۳، ۵، ۶، ۷) و تحلیلهای مربوط به موارد مورد نیاز (فصول ۱۱، ۱۲، ۲۱).

مرحله توسعه^۲ و ارائه روی چگونگی (how) متمرکز است. یعنی در طول این مرحله مهندس سعی دارد چگونگی ساخته شدن داده‌ها، اجرای کار در ساختار نرم‌افزاری، چگونگی اجرای جزئیات کار، چگونگی توصیف رابط‌ها چگونگی تبدیل طرح به زبان برنامه‌نویسی (یا زبان غیر روبه‌ای) و چگونگی انجام آزمون را توصیف کند. روشهای بکار گرفته شده در طول مرحله توسعه متفاوتند اما سه کار فنی خاص همیشه رخ می‌دهد: طراحی نرم‌افزار (فصول ۱۴، ۱۵ و ۲۱) تولید کد و آزمون نرم‌افزار (فصول ۱۶، ۱۷ و ۲۲).

مرحله پشتیبانی^۳ روی تغییر (change) متمرکز است که با اصلاح خطا، ایجاد تطابقات لازم در حد لازم در رابطه با محیط نرم‌افزاری و تغییراتی ناشی از بهبود کار بخاطر تغییر نیاز مشتری، رابطه دارد. در طول این مرحله چهار نوع تغییر را شاهد هستیم:

اصلاح^۴: حتی در مورد بهترین کارهایی که کیفیتشان تضمین شده نیز این احتمال وجود دارد که مشتری متوجه نقایصی در نرم‌افزار شود. کارهای حفظ و نگهداری اصلاحی^۵، نرم‌افزار را به سوی اصلاح نواقص سوق می‌دهد.



مهندسی نرم‌افزار یعنی تأکید بر سه موضوع مستقل: تعریف، توسعه، و پشتیبانی و نگهداری

نقل قول

انجمن معتقد است که باید نسبت به طبیعت دید و بینش ساده‌ای داشت، از آنرو که خدا بولهوس یا مستبد نمی‌باشد. چنین عقیده‌ای برای مهندسی نرم‌افزار مطلوب نخواهد بود. چرا که با پیچیدگی‌های بسیار روبرو می‌باشیم. فردبوکر

1.definition phase

2.development phase

3.support phase

4.change

5.Correction

6.Corrective maintenance

تطابق^۱ به مرور زمان، محیط اصلی که نرم افزار برای آن طراحی شده (مثل CPU، سیستم عامل، قواعد تجاری، مشخصه های محصول خارجی) تغییر می کند. کارهای تطبیقی منجر به تغییراتی در نرم افزار می شود که آن را با محیط خارجی اش مطابقت می دهد.^۲

بهبود وضعیت^۳ با استفاده از نرم افزار، مشتری / کاربر متوجه کارهای دیگری می شود که سودمند خواهند بود. این کارهای بی عیب و نقص سازی^۴، نرم افزار را چیزی فراتر از نیازهای کاربردی اصلی خود می سازد.

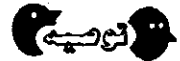
پیشگیری^۵ نرم افزار کامپیوتر بر اثر تغییر سودمندی خود را از دست می دهد و به همین خاطر، کارهای پیشگیرانه^۶ که اغلب مهندسی مجدد نرم افزار^۷ نامیده می شوند، باید صورت بگیرند تا نرم افزار بتواند نیازهای کاربر نهایی خود را مرتفع سازد. در اصل، کارهای پیشگیرانه تغییراتی در برنامه کامپیوتری ایجاد می کند بطوریکه آنها را می توان به صورت آسان تری اصلاح نموده، سازگار کرد و بهبود بخشید.

علاوه بر این فعالیتهای پشتیبانی، کاربران نرم افزاری به پشتیبانی مستمر نیاز دارند. در داخل اغلب تکنسین های فنی، مراکز کمک تلفنی و سایتهای شبکه ای خاص بعنوان بخشی از کار پشتیبانی عمل می کنند.

امروزه، تعداد زیادی برنامه های بجا مانده (به ارث رسیده)، بسیاری از شرکتها را مجبور می سازند که راهبردهای مهندسی مجدد در نرم افزار را دنبال کنند. (فصل ۳۰). در حال کلی، این کار اغلب بخشی از فرایند تجاری مهندسی مجدد در نظر گرفته می شود.

این فازها و مراحل توصیف شده در دیدگاه کلی ما نسبت به مهندسی نرم افزار، بوسیله یک سری فعالیتهای پوششی^۸ تکمیل می شوند. فعالیتهای نمونه در این گروه شامل موارد زیر است:

- کنترل و پیگیری پروژه نرم افزاری.
- بازنگریهای فنی رسمی.
- تضمین کیفی نرم افزار.
- مدیریت طرح نرم افزاری.
- آماده سازی اسناد و تولید.



هنگامی که شما اصطلاح "نگهداری" را بکار می برید، معنای بسیار فراتر از رفع اشکال را اراده کرده اید.



فعالیت های چترگونه

1. Adaptation
2. Adaptive maintenance
3. Enhancement
4. Perfective maintenance
5. Prevention
6. preventive maintenance
7. software reengineering
8. umbrella activities

• مدیریت قابلیت استفاده مجدد.

• ارزیابی.

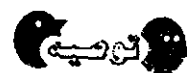
• مدیریت خطر.

فعالیت‌های پوششی در سراسر فرآیند نرم‌افزار بکار گرفته شده و در بخش‌های ۲ و ۵ این کتاب تشریح می‌شوند.

۲-۲ فرآیند نرم‌افزار

می‌توان یک فرآیند پردازش نرم‌افزاری را به صورت شکل ۲-۲ نشان داد. با تعریف تعداد کمی از فعالیت‌های معینی که در همه پروژه‌های نرم‌افزاری بدون توجه به اندازه یا پیچیدگی‌شان قابل اجرا هستند، یک چارچوب کلی برای فرآیند پردازش^۱ بوجود می‌آید. بعضی از کارها که هرکدام مجموعه‌ای از کارهای مهندسی نرم‌افزاری^۲، معیارهای پروژه محصول کار و نقاط تضمین کیفیت هستند، باعث می‌شود که این مجموعه‌های معین فعالیتی در مشخصه‌های پروژه نرم‌افزاری بکار گرفته شده و در مجموعه تقاضاهای تیم پروژه نیز باشند. نهایتاً، فعالیت‌های پوششی مثل تضمین کیفیت نرم‌افزار، مدیریت طراحی نرم‌افزار و ارزیابی، بر کل مدل فرآیند پردازش حاکم می‌شود. فعالیت‌های پوششی مستقل از هرگونه فعالیت معین هستند و در طول فرآیند رخ می‌دهند.^۳

در سالهای اخیر، تأکید خاصی روی «کامل شدن فرآیند» شده است. مؤسسه مهندسی نرم‌افزار (SEI) مدل جامعی ارائه داده که روی مجموعه‌ای از توانایی‌های مهندسی نرم‌افزار که باید برای دسترسی سازمانها به سطوح مختلف بلوغ فرآیند وجود داشته باشد، پیش بینی می‌شود. برای تعیین وضعیت جاری تکمیل فرآیند یک سازمان، SEI از ارزیابی‌ای استفاده می‌کند که منجر به یک طرح درجه‌بندی پنج امتیازی است. این طرح درجه‌بندی میزان تطابق با توانایی مدل کامل^۴ (CMM)^۵ [PAU93] را مشخص می‌کند که فعالیت‌های اصلی لازم در سطوح مختلف فرآیند تکامل را تعریف می‌کند. رهیافت SEI مقیاسی از میزان مؤثر بودن کارهای مهندسی نرم‌افزار یک شرکت را مهیا ساخته و پنج سطح تکمیل فرآیند را بوجود می‌آورد که به شکل زیر تعریف شده‌اند:



مجموعه فعالیت‌های
مشترکی را انتخاب
کنید که محصول افراد
و پروژه را باهم مد نظر
قرار دهد

1. common process framework

2. task sets

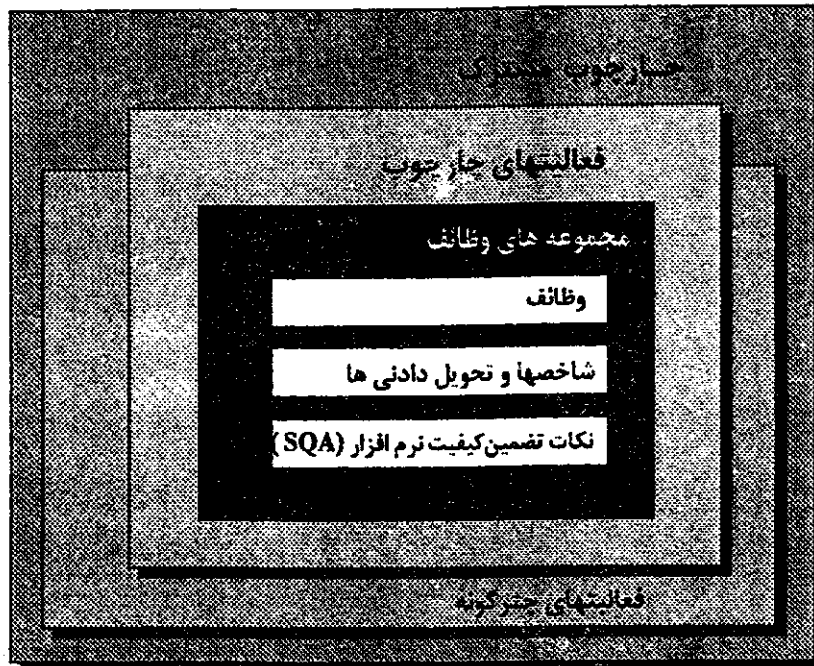
۳. این عناوین در فصل‌های آتی به تفصیل توضیح داده خواهند شد.

4. maturity model (CMM)

5. Paulk, M.

سطح ۱: اولیه. فرایند نرم افزاری بصورت موقتی و حتی بعضی اوقات بسیار درهم و برهم توصیف شده است. چند فرایند تعریف می شود و موفقیت به تلاش های فردی بستگی دارد.

سطح ۲: قابل تکرار. فرایندهای اولیه مدیریت پروژه برای مشخص کردن هزینه، زمانبندی و قابلیت کارایی آن صورت می گیرند. اصل ضروری فرایند در جایی است که موفقیت های پیش تر پروژه هایی با کاربردهای مشابه را تکرار می کند.



شکل ۲-۲ فرایند نرم افزار

سطح ۳: تعریف شده. فرایند نرم افزاری برای مدیریت و فعالیت های مهندسی در طول یک فرایند پردازش نرم افزاری در سازمان ثبت شده، استاندارد و منسجم است. پروژه ها از یک نسخه تأیید شده و ثبت شده در فرایند پردازش سازمان استفاده می کنند که برای توسعه و پشتیبانی نرم افزار می باشد. این سطح شامل تمام مشخصه های تعریف شده در سطح ۲ می باشد.

سطح ۴: مدیریت شده. اقدامات دقیق صورت گرفته در فرایند پردازش نرم افزار و کیفیت محصول همگی کنترل می شوند. فرایند نرم افزاری و محصولات از لحاظ کمی شناسایی شده و با استفاده از اقداماتی دقیق کنترل می شوند. این سطح شامل تمام مشخصه های تعریف شده در سطح ۳ خواهد بود.

سطح ۵: بهینه سازی. بهبود مکرر فرایند با توجه به بازخورد کمی فرایند و از روی آزمون ایده های نوآورانه و فناوری های تازه، مقدور است. این سطح شامل تمام مشخصه های تعریف شده برای سطح ۴ می باشد.



ارجاع به وب

انستیتو مهندسی نرم
افزار (SEI) رشته
ای گسترده از
اطلاعات مرتبط با
فرایند را پیشنهاد می
کند.
www.sei.cmu.edu

نکته حائز اهمیت آن است که هر سطح بالای سطح ۲ زیرمجموعه‌ای از سطح قبلی است. مثلاً یک ارائه کننده نرم‌افزار در سطح ۳ باید به سطح ۲ دست یافته باشد تا بتواند به فرایندهای لازم در سطح ۲ دست پیدا کند.

سطح ۱: از نظر تضمین کیفی و مدیریت پروژه دارای وظیفه خاصی نیست. در این سطح نیم پروژه می‌تواند برای ارائه نرم‌افزار هر راهی را که مایل است با استفاده از روشها، استانداردها و رویه‌هایی که از کیفیت خوب تا خیلی ضعیف قرار دارند، انتخاب کند.

سطح ۲: نمایانگر این حقیقت است که ارائه دهنده نرم‌افزار فعالیت‌های خاصی را همچون گزارش تکمیل کار و گزارش زمان و کارهای صورت گرفته را تعریف کرده است.

سطح ۳: نمایانگر این است که ارائه دهنده نرم‌افزار فرایندهای فنی و مدیریتی را تعریف نموده مثلاً استاندارد برای برنامه‌نویسی بطور دقیق ارائه و بوسیله یک سری رویه‌های مثل حسابرسیها به اجرا درآمده است. این سطح یکی از سطوحی است که اکثر تولیدکنندگان نرم‌افزار از طریق استانداردهای مثل ایزو ۹۰۰۱ آن را مدنظر دارند و موارد معدودی وجود دارد که ارائه دهندگان نرم‌افزار از این سطح تجاوز کنند.

سطح ۴: دربر گیرنده مفهوم اندازه‌گیری و استفاده از متریک است. فصل ۴ این موضوع را به صورت مفصل‌تری مورد بحث قرار می‌دهد. در اینجا این نکته حائز اهمیت است که مفهوم متریک به منظور درک اهمیت تولیدکننده‌ای است که به سطح ۴ و ۵ دست یافته است.

متریک یک کمیت معنادار است که می‌توان آن را از روی بعضی از اسناد یا کدی در داخل پروژه نرم‌افزاری استخراج نمود. مثالی از یک چنین متریکی عبارتست از تعداد شاخه‌های شرطی در یک بخش از کد برنامه. این متریک در صورتی معنی دارد که نشانه‌ای از کارهای لازم را آزمون کد مهیا کند. این متریک مستقیماً با تعداد مسیرهای آزمونی داخل کد مربوط است.

یک سازمان چهار سطحی متریکهای متعددی را جمع‌آوری می‌کند. سپس این متریکها برای مشاهده و کنترل پروژه نرم‌افزار استفاده می‌شوند. مثلاً:

- ممکن است یک متریک آزمونی برای تعیین زمان پایان آزمون یک قلم از کد استفاده شود.

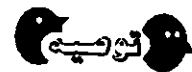
- یک متریک خواندنی برای قضاوت درمورد قابلیت خواندن بعضی از اسناد به زبان طبیعی

استفاده می‌شود.

- از متریک درک و شناسایی برنامه ممکن است برای تهیه آستانه رقمی استفاده شود که

برنامه نویسان نمی‌توانند از آن حد فراتر بروند.

به منظور موفقیت متریکها در این سطح هم اجزای سطح سوم CMM باید در جای خود باشند، مثلاً عبارات دقیقاً تعریف شده برای فعالیت‌هایی مثل مشخصه‌های لازم و طرح بطوریکه متریک بتواند براحتی از وسایل مکانیکی استخراج گردد.



هر سازمانی باید بکوشد تا به بلوغ قابلیت استتیمو مهندسی نرم‌افزار (SEI CMM) نزدیک شود. هرچند پیاده سازی جنبه ای از مدل است که با توجه به موقعیت شما، متفاوت خواهد بود.

سطح ۵ بالاترین سطح قابل دسترسی است. تعداد بسیار کمی از تولیدکنندگان به این سطح دست یافته‌اند. این سطح نمایانگر آنالوگ مکانیزمهای کنترل کیفی نرم‌افزار است که در مکانیزمهای دیگر و صنایع کامل‌تر وجود دارند. مثلاً تولید کننده یک محصول صنعتی مثل بولبرینگ می‌تواند این کیفیت را براساس دستگاهها و فرایندهای تولید مورد استفاده برای تولید بولبرینگ پیش‌بینی کنند. به همین شکل تولیدکننده نرم‌افزار در سطح ۵ می‌تواند نتایج را مثل تعداد اشکالهای باقی مانده در محصول براساس ارزیابی صورت گرفته در طول اجرای پروژه، پیش‌بینی کند. علاوه بر آن، چنین تولیدکننده‌ای می‌تواند اثر فرایندی جدید یا ابزار تولید را از آن پروژه با آزمون متریک‌هایی از پروژه و مقایسه آنها با پروژه‌های قبلی که از این فرایند یا ابزار استفاده نکرده‌اند، تشریح کند.

باید تأکید کرد که برای اینکه تولید کننده نرم‌افزار به سطح ۵ دسترسی پیدا کند باید هر فرایند کاملاً تعریف شده و آن را دقیقاً مورد اجرا قرار داد. این کار نتیجه بودن در سطح ۳ است. اگر تولیدکننده‌ای فرایندها را به صورت تعریف شده و مشخص نداشته باشد، تغییرات زیادی در فرایند حاصل شده و آمار و ارقام مورد استفاده برای فعالیتهای همچون پیش‌بینی را نمی‌توان مورد استفاده قرار داد.

پنج سطح تعریف شده توسط SEI در نتیجه واکنش های ارزیابی در پرسشنامه ارزیابی SEI که براساس CMM است، بدست آمده‌اند. نتایج این پرسشنامه به صورت یک درجه عددی درآمده که نشانه به کمال رسیدن فرایند یک سازمان است.

SEI دارای حوزه‌های اصلی در فرایند است (KPA)^۱ که هر کدام یک سطح رشد و تکامل دارند. KPAها آن گروه از فعالیتهای مهندسی نرم‌افزار را توصیف می‌کنند (مثل برنامه‌ریزی پروژه نرم‌افزاری، نیازمندیهای مدیریتی) که باید برای صورت گرفتن کار درست در سطح بخصوصی موجود داشته باشند. هر KPA با شناسایی مشخصه‌های زیر توصیف می‌شود:

- اهداف^۲ - اهداف کلی که KPA باید به آن دست یابد.
- تعهدات^۳ - نیازمندیهایی (که از طرف سازمان اعمال شده) که باید مرتفع شوند تا به اهداف دست یابیم و دلایلی باید ارائه شود که با اهداف مطابقت داشته باشد.
- توانها^۴ - این چیزها باید درجای خود باشند (چه از نظر سازمانی و چه از نظر فنی) که سازمان بتواند به تعهداتش عمل کند.
- فعالها^۵ - کارهای خاصی که لازمند تا به عملکرد KPA دست یابیم.

1. key process areas (KPAs)

2. Goals

3. Commitments

4. Abilities

5. Activities

• روشهای کنترل پیاده سازی^۱ - شیوه‌ای که با آن فعالیتها در جای خود قرار می‌گیرند، کنترل می‌شوند.

• روشهایی برای ارزیابی درستی پیاده سازی^۲ - شیوه‌ای که با آن عمل درست برای KPA مشخص می‌شود.

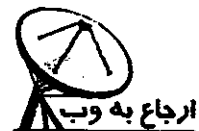
هیچ‌کدام KPA (هرکدام با استفاده از این خصوصیات توصیف شده‌اند) در طول فرایند تکمیل کار تعریف شده و در سطح مختلف فرایند تکاملی بصورت طرح درآمده‌اند. KPAهای زیر را در هر سطح تکمیلی فرایند به دست می‌آوریم:

سطح ۲ فرایند تکمیلی

- مدیریت پیکربندی نرم‌افزار.
- تضمین کیفیت نرم‌افزار.
- مدیریت پیمانکاریهای نرم‌افزار.
- پی‌گیری و نظارت پروژه نرم‌افزاری.
- مدیریت نیازمندیها.

سطح ۳ فرایند تکمیلی

- بازبینی های یکسان.
- همکاری درون گروهی.
- مهندسی محصول نرم‌افزاری.
- مدیریت یکپارچه نرم‌افزاری.
- برنامه آموزشی.
- تعریف فرایند سازمانی.
- نقطه تمرکز فرایند سازمانی.



یک نسخه جدول گونه
از SEI-CMM
کامل مشتمل بر
اهداف، تعهدات،
قابلیتها و فعالیتهایی
است که در آدرس:
sepo.nosc.mil/CM/Matrices.htm
وجود دارد.

سطح ۴ فرایند تکمیلی

- مدیریت کیفیت نرم‌افزار.

1. Methodes for monitoring implementation

2. Methodes for verifying implementation

- مدیریت فرآیند کمی.

سطح ۵ فرآیند تکمیلی

- مدیریت تغییر فرآیند.
- پیشگیری از نقیصه و اشکال.

هر یک از KPAها بوسیله یک سری روشهای مهم و اصلی^۱ تعریف شده که به امر دستیابی به اهداف کمک می‌کنند.

روشهای اصلی عبارتند از سیاستها، رویه‌ها و فعالیتهایی که باید قبل از وقوع یک حوزه فرآیند اصلی، اجرا شوند. SEI شاخص‌های اصلی^۲ را بعنوان روشهای اصلی یا جزمهای روشهایی اصلی تعریف می‌کند که بیشترین نظر را درمورد اینکه آیا اهداف یک حوزه فرآیند اصلی حاصل شده‌اند یا خیر، ارائه می‌دهد. سوالات ارزیابی برای تحقیق درمورد وجود (یا نبود) شاخص اصلی استفاده می‌شوند.

۲-۲ مدل‌های فرآیند نرم افزار

به منظور حل مشکلات صنعتی باید یک مهندس نرم‌افزار یا یک تیم از مهندسين نرم‌افزار در تولید یک راهبرد کمک کنند که دربر گیرنده لایه‌های فرآیند، روشها و ابزار است که در بخش ۱-۲ توصیف شده و مراحل کلی مورد بحث در بخش ۲-۱-۲ را نشان می‌دهد. اغلب این راهبرد را مدل فرآیند^۳ یا پارادایم مهندسی نرم‌افزار^۴ می‌نامند. یک مدل فرآیند برای مهندسی نرم‌افزار براساس ماهیت پروژه و کاربرد آن، روشها و ابزار مورد استفاده و کنترل‌ها و موارد قابل ارائه که لازمند، انتخاب می‌شود. در یک گزارش حالت درمورد ماهیت فرآیند نرم‌افزاری، ال. بی. اس. راکون [RAC95]^۵ از ابعاد کسری بعنوان مبداء بحث درمورد ماهیت حقیقی فرآیند نرم‌افزاری، استفاده می‌کند.

تمام کارهای توسعه و ارائه یک نرم‌افزار را می‌توان بعنوان یک حلقه حل مشکل^۶ توصیف کرد (شکل ۲-۲ الف) که در آن چهار مرحله مشخص دیده می‌شوند: وضع موجود، تعریف مشکل، توسعه فنی و منسجم کردن راه حل. وضع موجود نمایانگر وضعیت موجود امور است [RAC95]، تعریف مسئله، مشکل خاصی را که باید حل شود تعریف می‌کند. توسعه فنی، مشکل را از طریق بکارگیری فناوری حل می‌کند و یکپارچه‌سازی راحل، حل نتایج را (مثل اسناد، برنامه‌ها، داده‌ها، عملکرد تجارت جدید، محصول جدید) به

نقل قول

اغلب موارد نرم افزار از اولین قانون دوچرخه سواری تبعیت می‌کند: مهم نیست که به چه سمتی می‌روی، سربالا می‌روی یا رو به باد. یک ناشناس

1. key practices

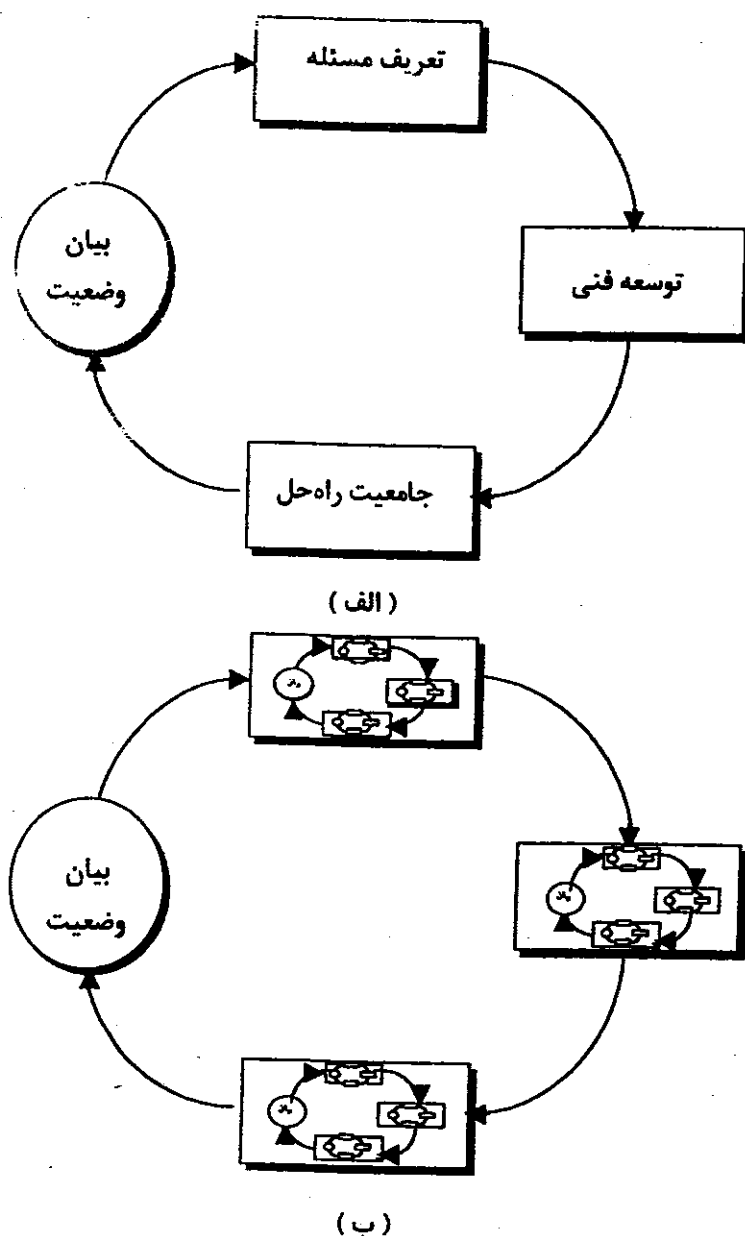
2. key indicators

3. process model

4. software engineering paradigm

5. Raccon. L. b. S.

کسانی که راه حل را در مرحله اول تقاضا کردند، ارائه می‌دهد. کل مراحل و فازهای مهندسی نرم‌افزاری که در بخش ۲-۱-۲ مشخص شده‌اند را می‌توان به راحتی در این مراحل ترسیم نمود.



شکل ۲-۲ الف) مراحل یک چرخه حل مسئله [RAC 95]
ب) مراحل میانی چرخه حل مسئله [RAC 95]

درواقع، تفکیک فعالیتها آنطور که شکل ۲-۳ ب نشان می‌دهد سخت است زیرا در هر مرحله و در طول مراحل صحبت‌های ضد و نقیضی رخ می‌دهد. این نظریه ساده منجر به ایده بسیار مهمی می‌شود: بدون توجه به مدل فرآیند انتخاب شده برای پروژه نرم‌افزاری، هم این مراحل همزمان با هم با مقداری جزئیات

وجود دارند. با فرض ماهیت بازگشتی شکل ۲-۳ ب، چهار مرحله مورد بحث بالا بطور یکسان در تحلیل یک شیوه کاربردی کامل و در تولید بخش کوچکی از کُد استفاده می‌شوند.

راکون [RAC95] یک مدل نامرتب ارائه می‌دهد که تولید نرم‌افزار را بعنوان یک پیوستار از کاربر به تولیدکنند تا فناوری ادامه می‌دهد. با پیشرفت کار به سوی به یک سیستم کامل، مراحل توصیف شده فوق به صورت بازگشتی نسبت به نیازهای کاربر و مشخصه فنی نرم‌افزار که توسط تولید کننده ارائه شده، بکار گرفته می‌شوند.

در بخشهای بعدی، یک سری مدلهای فرایند متفاوت برای مهندسی نرم‌افزار مورد بحث قرار می‌گیرند. هر یک نمایانگر تلاش برای ایجاد نظم در یک فعالیت نامنظم و آشفته است. نکته مهمی که باید به خاطر آورد این است که هر مدل به شکلی توصیف شده که به کنترل و هماهنگی یک پروژه نرم‌افزاری واقعی کمک می‌کند. و با اینحال، در مرکز همه آنها، مدلهای مشخصه‌های یک مدل نامنظم را نشان می‌دهند.



تمام مراحل یک فرایند
نرم افزار - تعیین
وضعیت، تعریف مسئله،
توسعه فنی و مجموعه
راه حل ها - در برخی
سطوح تفصیلی و
جزئیات، همزیستی و
اشتراک دارند

۲-۴ مدل ترکیبی خطی^۱

این مدل را که گاهی «چرخه حیات کلاسیک»^۲ یا «مدل آبشاری»^۳ می‌نامند، بیانگر یک نگرش نظام مند و زنجیری^۴ نسبت به تولید نرم‌افزار است که در سطح سیستم شروع شده و با تحلیل، طراحی، کدگذاری، آزمون و پشتیبانی نرم‌افزاری پیشروی می‌کند. شکل ۲-۴ مدل زنجیری خطی را برای مهندسی نرم‌افزار مدلسازی شده بعد از سیکل متعارف مهندسی و طراحی نشان می‌دهد، این مدل دربر گیرنده فعالیتهای زیر است:

مهندسی اطلاعات / سیستم و مدلسازی.

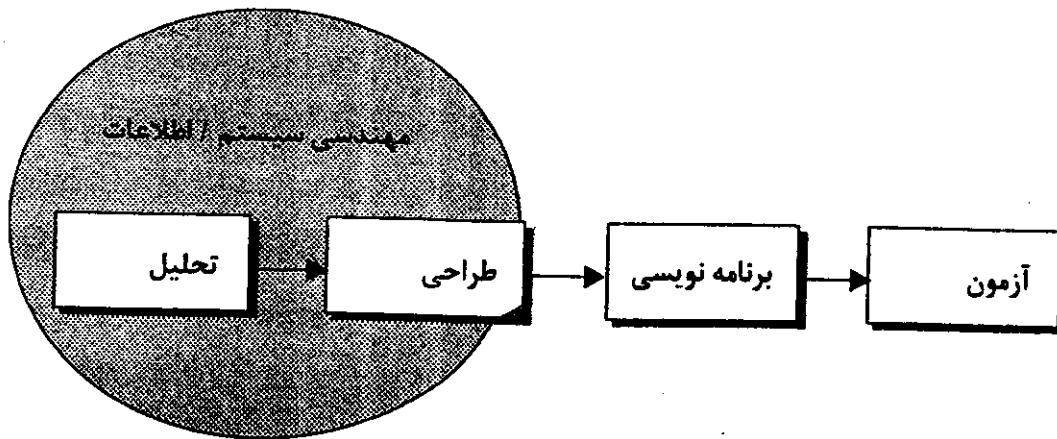
از آنجا که نرم‌افزار همیشه بخشی از یک سیستم بزرگ تر است، کار با ارائه تقاضا برای همه عناصر سیستم آغاز شده و سپس با تخصیص زیرمجموعه‌ای از این نیازمندیها به نرم‌افزار ادامه می‌یابد. نگرش به این سیستم وقتی ضروری است که نرم‌افزار با دیگر عناصر مثل سخت افزار، افراد و پایگاههای داده‌ای رابطه برقرار می‌کند. مهندسی و تحلیل سیستم دربر گیرنده نیازمندیهای جمع آوری شده در سطح سیستم با مقدار کمی تحلیل و طراحی است. مهندسی اطلاعاتی شامل موارد موردنیاز جمع آوری شده در سطح تجاری راهبردی و در سطح حوزه تجارت است.

1. linear sequential model

2. classic life cycle

3. waterfall model

۱. اگر چه مدل آبشاری اصلی ارائه شده توسط وینستن رویس [ROY70] باید دیدگاهی مبتنی بر حلقه های بازخور ساخته شده بود، بسیاری از سازمانها این مدل فرایند را به گونه ای خطی مورد استفاده قرار دادند.



شکل ۲-۴ مدل ترتیبی خطی

تحلیل نیازمندیهای نرم افزاری

فرآیند جمع‌آوری نیازمندیها تشدید یافته بطور خاص روی نرم‌افزار متمرکز شده است. به منظور درک ماهیت برنامه‌های تولیدی، مهندسی نرم‌افزار (تحلیلگر) باید دامنه اطلاعات را درمورد نرم‌افزار بعلاوه عملکرد لازم، وضعیت، عمل و برقراری ارتباط آن، بشناسد.

طراحی

طراحی نرم‌افزار در حقیقت یک فرآیند چند مرحله‌ای است که روی چهار صفت متمایز یک برنامه متمرکز می‌شود یعنی ساختار داده‌ها، معماری نرم‌افزار از نظر ساختمانی، نمایش رابط و جزئیات رویه‌ای (الگوریتم)، فرآیند طراحی نیازها را به نمایش نرم‌افزاری تبدیل می‌کند که می‌توان آن را قبل از شروع کدگذاری از نظر کیفیت ارزیابی کرد.

ایجاد کد

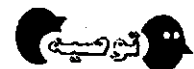
طراحی باید به صورت شکلی در آید که برای دستگاه قابل خواندن باشد. مرحله ایجاد کد این کار را انجام می‌دهد. اگر طراحی به صورت دقیق صورت گیرد، می‌توان بصورت مکانیکی تولید کد را انجام داد.

آزمون

وقتی کد ایجاد شد، آزمون برنامه شروع می‌شود. کار آزمون روی فواصل زمانی منطقی در نرم‌افزار متمرکز می‌شود و تضمین می‌کند که تمام جملات آزمون شده و درمورد خروجیهای عملیاتی یعنی اجرای آزمون برای نشان دادن خطاها و تضمین اینکه داده‌های تعریف شده نتایج واقعی بوجود می‌آورند. با نتایج لازم همخوانی دارند.

پشتیبانی

بدون شک نرم‌افزار بعد از اینکه در اختیار مشتری قرار می‌گیرد دچار تغییراتی می‌شود (یک استثناء احتمالی در نرم‌افزار وجود دارد). تغییر برای این رخ می‌دهد که با خطا مواجه شده‌ایم زیرا نرم‌افزار باید آراسته شود تا با تغییرات محیط خارجی تطابق پیدا کند (مثل تغییر بخاطر یک سیستم عامل جدید یا یک



هرچند مدل خطی اغلب به عنوان روشی از منافته قلمداد می‌شود، هنوز دلایلی برای استفاده از آن رهیافت وجود دارد، خصوصاً هنگامی که نیازمندیها به خوبی قابل درک باشند

وسیله جانبی تازه) یا بخاطر اینکه نیازهای عملیاتی یا کارایی مشتری ارتقاء یافته است. پشتیبانی و نگهداری نرم‌افزار دوباره در هر یک از فازهای قبلی یک برنامه موجود بکار گرفته می‌شوند تا یک برنامه جدید برای آن ایجاد شود.

مدل زنجیری خطی قدیمی‌ترین و رایج‌ترین پارادایم مورد استفاده برای مهندسی نرم‌افزار است. انتقاد از این پارادایم باعث شده که حتی افراد پشتیبانی دهنده نیز در مورد کارایی [HAN95]^۱ آن شک کنند. در میان مشکلاتی که گاهی وقتی که از مدل زنجیری خطی استفاده می‌کنیم با آن مواجه می‌شویم می‌توان به موارد زیر اشاره کرد:

۱- پروژه‌های واقعی به ندرت از جریان زنجیری و متوالی که مدل ارائه می‌دهد، پیروی می‌کنند. گرچه مدل خطی می‌تواند با تکرار تطبیق یابد، اما آن را به طور غیرمستقیم انجام می‌دهد. در نتیجه، با انجام کار توسط اعضای پروژه ممکن است موارد گیج کننده‌ای رخ دهد.

۲- اغلب برای مشتری مشکل است که تمام نیازهای خود را به طور مشخص بیان کند. مدل زنجیری خطی به این امر نیاز داشته و در تطبیق عدم قطعیت طبیعی که در آغاز بسیاری از پروژه‌ها وجود دارد، دارای مشکل است.

۳- 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 می‌گیرد مشخص نشود، می‌تواند فاجعه آمیز باشد.

پاراداک [BRA94] در یک تحلیل جالب از پروژه‌های واقعی دریافت که طبیعت خطی چرخه حیات سنتی "وضعیت بلاکی و بلوکه" را به دنبال دارد به این معنا که برخی از اعضای تیم پروژه باید منتظر دیگر اعضا باشند تا کارشان تکمیل شود. به عبارت دیگر مدت زمانی که در وضعیت انتظار سپری می‌شود از زمان تولید نرم‌افزار پیشی می‌گیرد! این وضعیت بلاکی در ابتدا و انتهای فرآیند خطی ترتیبی نمود بیشتری می‌یابد.

هر یک از این مشکلات واقعی هستند. معیار چرخه حیات کلاسیک دارای نقش مشخص و مهمی در کار مهندسی نرم‌افزار است. این معیار الگویی در اختیار ما می‌گذارد که در آن روشهایی برای تحلیل، طراحی، برنامه‌نویسی، آزمون و پشتیبانی قرار دارند. این چرخه رایج‌ترین مدل روبه‌ای مورد استفاده در مهندسی نرم‌افزار باقی مانده است. با اینکه دارای نقاط منفی است اما تا حد زیادی بهتر از یک روش نامنظم در تولید نرم‌افزار است.



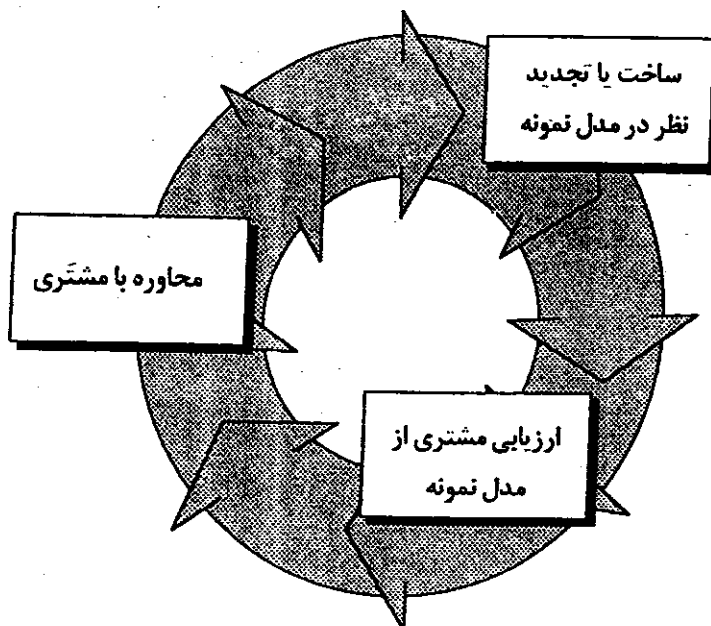
به چه علت برخی اوقات، مدل خطی با شکست مواجه می‌شود؟

۵-۲ مدل نمونه سازی^۱

اغلب، مشتری یک سری اهداف کلی برای نرم افزار تعریف می کند اما ورودیها، پردازش یا نیازهای خروجی را بطور دقیق بیان نمی کند. در سایر موارد، ممکن است تولیدکننده از کارایی الگوریتم، قابلیت تطابق آن با یک سیستم عامل یا شکل ارتباط متقابل دستگاه و انسان نامطمئن باشد. در این موارد و بسیاری از موارد دیگر، ممکن است یک پارادایم نمونه سازی بهترین روش باشد.

پارادایم نمونه سازی نخست (شکل ۵-۲) با جمع آوری نیازمندیها شروع می شود. مشتری و تولیدکننده همدیگر را ملاقات نموده و اهداف کلی خود را در مورد نرم افزار بیان می نمایند، هرچه که مورد نیاز است را شناسایی می نمایند و حوزه هایی را مطرح می کنند که در آن تعریف بیشتر ضروری است. سپس طراحی سریع رخ می دهد. این طراحی روی نمایش آن جنبه هایی از نرم افزار متمرکز می شود که برای مشتری/کاربر مشهودند (مثل رهیافت های داده های ورودی و قالب های خروجی). این طراحی سریع منجر به ساخت یک نمونه می شود.

نمونه 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، عمل تکرار رخ می دهد، درحالی که در همان زمان تولیدکننده نرم افزار را قادر می سازد نیازهای مشتری را بهتر درک کند.



شکل ۵-۲ پارادایم ساخت نمونه

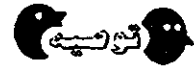
نمونه اولیه بعنوان مکاتیزی برای شناسایی موارد مورد نیاز نرم افزار عمل می کند. اگر یک نمونه کاری ساخته شود، تولیدکننده آن سعی می کند از برنامه موجود یا ابزارهای کاربردی که در اختیار دارد استفاده کند (مثل گزارش دهندگان، مدیران پنجره و غیره) تا برنامه های کاری را قادر به تولید سریع کند. اما وقتی نمونه تولید شده هدف مورد نظر فوق را تأمین کرد، ما چه کاری را انجام می دهیم؟ بروکز [BR075]^۱ پاسخ این سوال را می دهد:

در اکثر پروژه ها، اولین سیستمی که ساخته می شود به ندرت قابل استفاده است. ممکن است بسیار کند، بسیار بزرگ، از نظر کاری مشکل و یا هر سه مورد با هم باشد. هیچ راهی نیست جز تکرار کار، هوشمندانه اما با هوش تر تا نسخه جدیداً طراحی شده ای بسازیم که در آن این مشکلات حل شده باشد ... وقتی مفهوم یک سیستم جدید یا یک فناوری جدید استفاده می شود باید سیستمی ساخت که بتوان آن را از رده خارج کرد زیرا حتی بهترین برنامه ریزی نیز نمی تواند دربر گیرنده همه نکات باشد به طوری که در همان وهله اول درست کار کند. بنابراین سوال درمورد مدیریت این نیست که آیا یک سیستم را همانا ارائه کنیم و آن را دور بیندازیم یا نه شما این کار را می کنید. تنها سوالی که وجود دارد این است که آیا باید پیشاپیش طرحی را کشید که قابل دور انداختن باشد یا تعهد کنیم که آن را به مشتریان ارائه دهیم

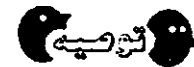
نمونه نخست می تواند به عنوان اولین سیستم عمل کند همانی که بروکز توصیه می کند به دور بیندازیم. اما ممکن است این یک دیدگاه ایده آل باشد. این مسئله حقیقت دارد که هم مشتری و هم تولیدکننده پارادایم نمونه نخستین را دوست دارند. کاربران درمورد سیستم واقعی حالت خاصی دارند و تولیدکننده ها نیز می خواهند فوراً چیزی بسازند. ممکن است بنابه دلایل زیر ارائه نمونه اولیه مشکل ساز باشد:

۱- مشتری آنچه را که به نظر می رسد نسخه کاری نرم افزار باشد می بیند بدون اطلاع از اینکه نمونه اولیه همراه با "آدامس بادکنکی و یو یو" ارائه می شود، و بدون اطلاع از اینکه ما اینقدر با عجله برای خرید نرم افزاری هجوم می بریم که تمام مولود کیفی یا قابلیت نگهداری درازمدت آن را نمی دانیم. وقتی مطلع می شویم که محصول باید مجدداً تولید شود بطوریکه می توان سطح کیفی بالای آن را حفظ نمود، مشتری ناراحت می شود و تقاضا می کند که چد کار دیگر باید روی نمونه صورت گیرد تا آن را به صورت یک محصول کاری در بیاورد. اغلب این مدیریت تولید نرم افزار است که کوتاه می آید.

۲- تولید کننده اغلب تعهدات اجرایی برای ارائه سریع نسخه کاری می دهد. یک سیستم عامل یا زبان برنامه نویسی مناسب ممکن است تنها برای این استفاده شود که موجود و شناخته شده است. ممکن است از یک الگوریتم نه چندان مناسب صرفاً به این خاطر استفاده شود که قابلیت برنامه را



هنگامی که مشتری شما نیازمندی قانونی و درستی دارد اما درگیر جزئیات امر شده است، در اولین گام یک مدل نمونه را بسازید.



در مقابل فشاری که برای تبدیل مدل نمونه کلی به یک محصول تولیدی وجود دارد، مقاومت کنید. آنچه در این میان ضربه می بیند کیفیت است.

تشریح کند. بعد از مدتی، تولیدکننده با این انتخابها آشنا شده و تمام دلایل نامناسب بودن آنها را فراموش می‌کند. انتخابی که کمتر از همه موردتوجه است اکنون یک بخش تکمیل کننده سیستم می‌شود.

گرچه مشکلاتی ممکن است رخ دهند اما تولید نمونه می‌تواند معیار مؤثری برای مهندسی نرم‌افزار باشد. نکته اصلی عبارتست از تعریف قوانینی از بازی در آغاز کار یعنی مشتری و تولید کننده باید هر دو توافق کنند که نمونه اولیه برای خدمت به عنوان مکانیزمی برای تعریف نیازمندیها ساخته می‌شود. و آنگاه می‌توان امید داشت که نرم‌افزار بطور کیفی و با قابلیت نگهداری، مهندسی شود.

۲-۶ مدل ساخت سریع برنامه‌ها (RAD)

Rapid Application Development یا توسعه سریع کاربرد (RAD)، یک مدل فرایند افزایشی تولید نرم‌افزار است که تنها بر یک سیکل تولید بسیار کوتاه تأکید می‌کند. مدل RAD یک نسخه تطابق یافته سرعت بالا از مدل زنجیری خطی است که در آن توسعه سریع با استفاده از ساختمانی مبتنی بر اجزاء تشکیل دهنده حاصل می‌شود. اگر موارد موردنیاز

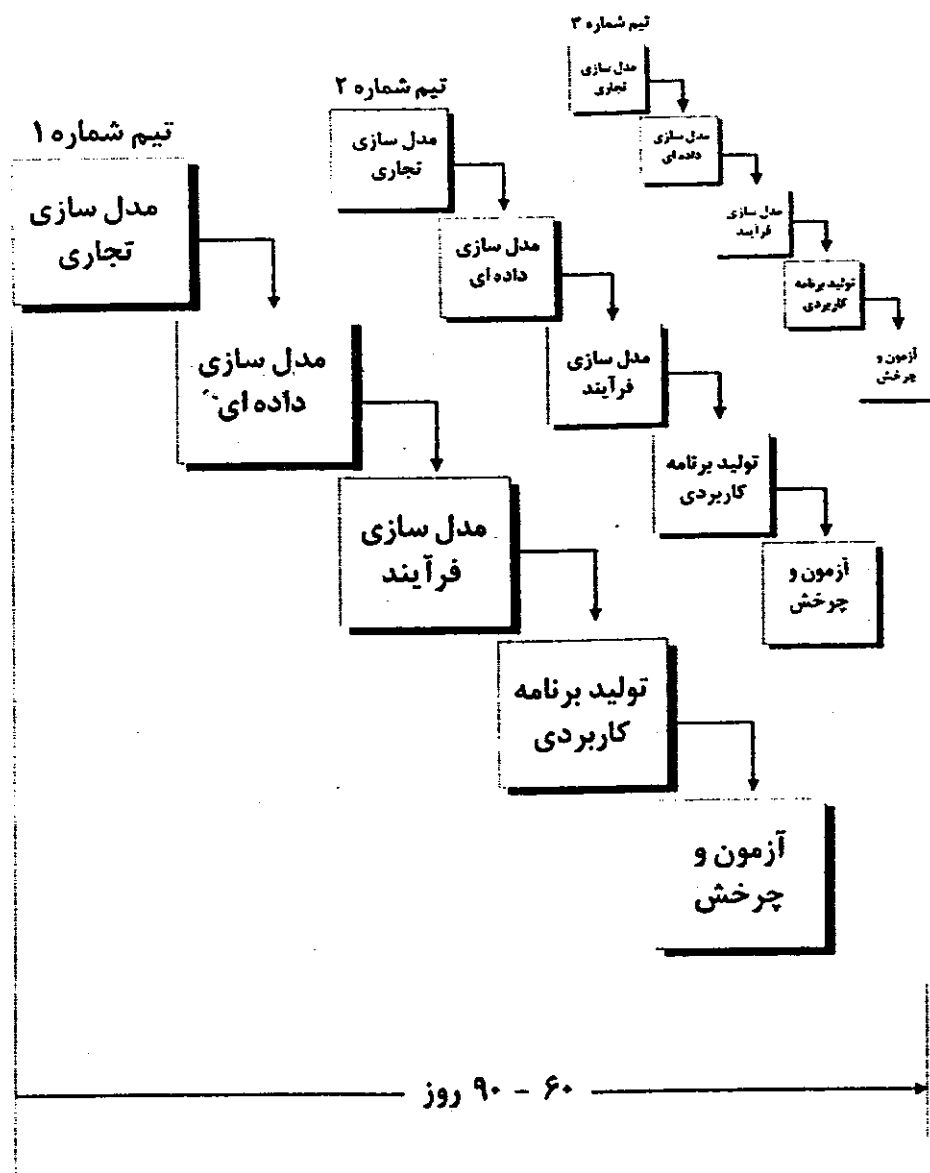
به خوبی شناسایی شده و دامنه پروژه محدود شود، فرایند RAD این امکان را به تیم می‌دهد که یک سیستم تمام کاربردی را ظرف مدت بسیار کوتاهی (مثلاً ۶۰ تا ۹۰ روز) [MAR91] روش RAD که اساساً برای کارهای سیستمهای اطلاعاتی استفاده می‌شود دربر گیرنده مراحل زیر است [KER94]

مدلسازی تجاری.

جریان اطلاعات در میان کارهای تجاری به شکلی مدلسازی می‌شود که به سوالات زیر پاسخ دهد: چه اطلاعاتی لازمه حرکت فرایند تجاری است؟ چه اطلاعاتی تولید می‌شوند؟ چه کسانی این اطلاعات را تولید می‌کنند؟ این اطلاعات به کجا می‌روند؟ چه کسانی آن را پردازش می‌کنند؟ مدلسازی تجاری به تفصیل در فصل ۱۰ توضیح داده شده است.

1. Martin, J.

2. Kerr, J.



شکل ۲-۶ مدل توسعه سریع برنامه‌ها (RAD)

مدلسازی داده‌ای.

جریان اطلاعاتی که بعنوان بخشی از مرحله مدلسازی تجاری تعریف شده در مجموعه‌ای از اهداف داده‌ای که برای پشتیبانی تجاری لازمند، پالایش می‌شود. مشخصه‌های^۱ هر هدف شناسایی شده و ارتباط بین این اهداف تعریف می‌شود. مدلسازی داده‌ای در فصل ۱۲ بررسی می‌شود.

مدلسازی فرآیند.

اهداف داده‌ای که در مرحله مدلسازی داده‌ها تعریف شده اند تغییر می‌یابند تا به جریان اطلاعاتی لازم برای اجرای یک کار تجاری، دست پیدا کنند. توضیحات عمل پردازش برای افزودن، شناسایی، حذف یا بازیابی یک شیء، داده‌ای ارائه می‌شوند.

تولید برنامه کاربردی.

فرض RAD براساس استفاده از فنون نسل چهارم است (بخش ۱۰-۲) بجای تولید نرم افزار با استفاده از زبانه‌های برنامه نویسی متعارف نسل سوم فرایند RAD تلاش می کند تا از جزءهای برنامه موجود مجدداً استفاده کرده (در صورت امکان) یا جزءهای قابل استفاده مجددی را تولید کند (وقتی که لازم است). در همه موارد، ابزارهای خودکار برای تسهیل ساختن نرم افزار استفاده می شوند.

آزمون و گردش.

از آنجا که فرایند RAD بر استفاده مجدد تأکید می کند بسیاری از اجزای برنامه مورد آزمون قرار گرفته اند. این کار در زمان کلی آزمون صرفه جویی می کند. در هر حال، باید جزءهای جدید آزمون شده و همه رابطها کاملاً بکار گرفته شوند.

مدل فرایند RAD در شکل ۲-۶ تشریح شده است. مشخص است که محدودیتهای زمانی اعمال شده بر پروژه RAD نیازمند یک دامنه قابل اندازه گیری [KER94]^۱ است. اگر بتوان یک برنامه کاربردی را به شکلی پیمانه بندی کرد که هر تابع اصلی را در کمتر از ۳ ماه کامل کند (با استفاده از روش توصیف شده فوق)، آن وقت برای RAD کاندید می شود. هر تابع اصلی را می توان با یک تیم جداگانه از RAD مورد بررسی قرار داد و سپس آن را بطور کلی یکپارچه نمود.

مانند همه مدل های پردازش، روش RAD دارای نقایصی است: [BUT94]^۱

- درمورد پروژه های بزرگ اما قابل اندازه گیری، RAD نیازمند منابع انسانی کافی است تا تعداد تیم های درست RAD را تشکیل دهد.

- RAD نیازمند تولید کننده و مشتری هایی است که بتوانند فعالیتهای سریع لازم را برای رسیدن به یک سیستم کامل در یک چارچوب زمانی کوتاه انجام دهند. اگر این تعهد دارای هیچگونه قانونی نباشد، پروژه شکست می خورد.

- همه نوع برنامه کاربردی برای RAD مناسب نیست. اگر سیستم به درستی پیمانه بندی نشود، ساختن اجزای لازم برای RAD مشکل ساز خواهد بود. اگر عملکرد عالی موضوع اصلی

است و قرار است از طریق تنظیم رابطها با اجزای سیستم تأمین شود، روش RAD ممکن است

کارگر نباشد.

- RAD وقتی که خطرات فنی بسیار زیاد است مناسب نخواهد بود. این وقتی رخ می دهد که

یک برنامه کاربردی جدیدی استفاده شدیدی از فناوری تازه نموده یا وقتی که نرم افزار تازه نیازمند

قابلیت میانیگری بین برنامه های کامپیوتر موجود باشد.



توسعه سریع کاربردی (RAD)
اجزاء قابل استفاده مجدد را تقویت می نماید. روی اطلاع بیشتر از توسعه مبتنی بر اجزاء، به فصل ۲۷ مراجعه نمایید.

۷-۲ مدل های تکاملی فرآیند نرم افزار

امروزه معلوم شده که نرم افزار مانند هر سیستم پیچیده دیگر در طول یک مدت زمانی مشخص تکامل می یابد. شرایط تجاری و محصول اغلب با صورت گرفتن توسعه تغییر کرده و باعث می شود مسیر مستقیم تا محصول نهایی غیرواقع بینانه شود. ضرب الاجل های دقیق در بازار، تکمیل یک محصول نرم افزاری جامع را غیرممکن ساخته اما یک نسخه محدود شده باید معرفی گردد تا فشار رقابتی یا تجاری را تعدیل کند. یک سری نیازهای اصلی محصول یا سیستم بخوبی شناخته شده است اما جزئیات محصول یا سیستم هنوز باید مشخص گردند. در این موقعیتهای مشابه مهندسين نرم افزار به مدلهایی نیاز دارند که مشخصاً برای سازگار کردن یک محصول به مرور زمان، طراحی شده اند.

مدل زنجیری خطی برای توسعه خط-مستقیم طراحی شده است. در اصل، فرض روش آبشاری بر این است که یک سیستم کامل بعد از تکمیل زنجیره خطی ارائه می شود. مدل نمونه برای این طراحی شده که به مشتری در درک شرایط و نیازها کمک کند. بطور کلی، این مدل برای این طراحی نشده که یک سیستم تولیدی ارائه دهد. ماهیت تکاملی نرم افزار در هیچ یک از این معیارهای کلاسیک مهندسی در نظر گرفته نمی شود.

• مدلهای تکاملی تکراری هستند. آنها به شیوه ای توصیف می شود که مهندسين نرم افزار را قادر می سازند نسخه های کاملتری از نظر نرم افزاری ارائه دهند.

۱-۷-۲ مدل فزاینده (افزایشی)

مدل فزاینده^۲ عناصر مدل زنجیری خطی را با دیدگاه تکرار مدل نخست، ترکیب می کند. باتوجه به شکل ۷-۲ مدل فزاینده زنجیره های خطی را به شکل متناوب با پیشروی زمانی در روی تقویم، بکار می گیرد. دو زنجیره خطی یک رشد قابل ارائه در نرم افزار پدید می آورد. مثلاً نرم افزار واژه پرداز کلمه که با استفاده از نمونه فزاینده ارائه شده ممکن است مدیریت فایل ها، اصلاح و تولید سند را در اولین رشد بصورت مقدماتی ارائه دهد. کارهای اصلاح و تولید سند پیچیده تر با دومین رشد صورت می گیرد، هجی کردن و بررسی نکات دستوری در سومین و توانایی صفحه بندی بصورت پیشرفته در رشد چهارم صورت می گیرد. باید توجه کرد که جریان فرآیند برای هر رشد می تواند از نمونه اولیه کمک بگیرد.

وقتی از یک مدل فزاینده استفاده می شود، اولین بخش رشد اغلب، هسته محصول اصلی^۳ است. یعنی نیازهای اولیه مورد توجه قرار می گیرند اما بسیاری از مشخصه های تکمیلی (که بعضی ها شناسایی و بعضی



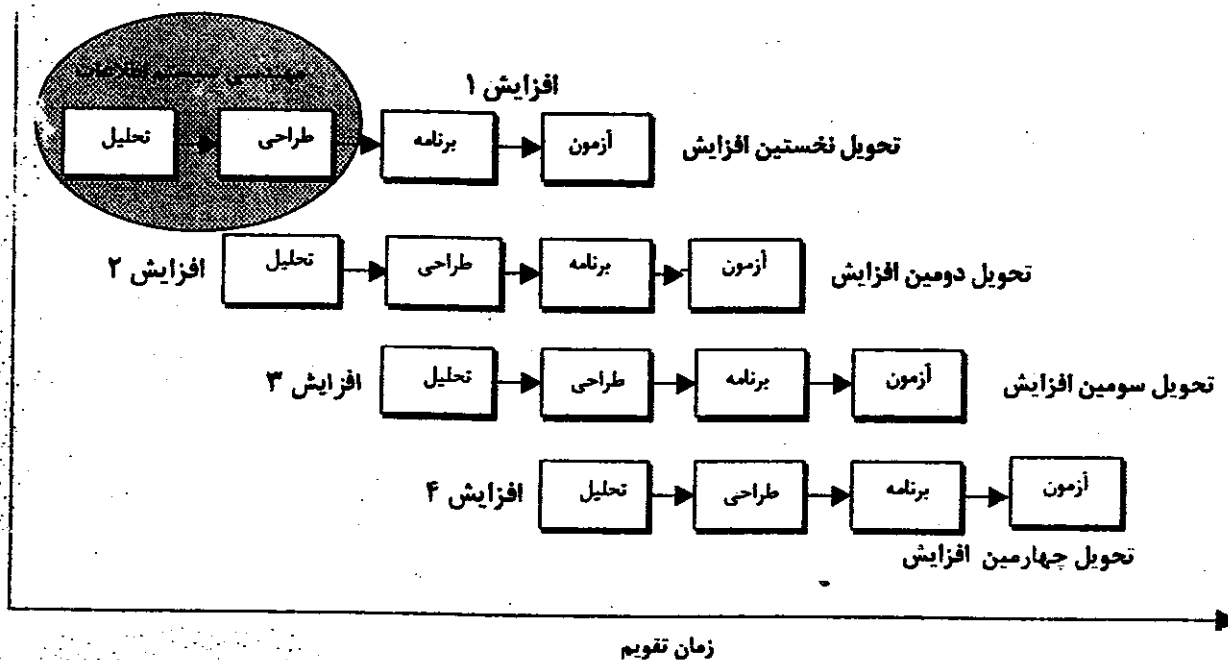
مدل افزایشی، نرم افزار را در بخش های کوچک اما قابل استفاده به پیش می برد، که آنها را "افزایش" می نامیم. به طور کلی ساخت هر افزایش، بر پایه ساخته های انتقال یافته پیشین خواهد بود.

1. Butler, J.

2. incremental model

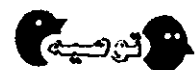
3. core product

دیگر شناسایی نشده‌اند) ارائه نمی‌گردند. هسته محصول بوسیله مشتری مورد استفاده قرار می‌گیرد (یا تحت بازبینی دقیق قرار می‌گیرد). بخاطر استفاده یا ارزیابی، طرحی برای رشد قسمت بعدی ارائه می‌شود. این طرح اصلاح هسته محصول را مورد توجه قرار می‌دهد تا بهتر نیازهای مشتری را برآورده نموده و ارائه مشخصه‌ها و قابلیت‌های کاری اضافی موردنظر مشتری را نیز بهتر سازد. این فرایند بدنبال تحویل هر بخش تکرار می‌شود تا وقتی که محصول کامل تولید شود.



شکل ۲-۷ مدل افزایشی

مدل فرایند فزاینده مثل ساخت نمونه اولیه و دیگر روشهای ارزیابی و تکمیل، ماهیت تکرار شونده دارد. اما برخلاف نمونه اولیه، مدل فزاینده روی تحویل یک محصول عملیاتی با هر بخش رشد یافته تأکید می‌کند. افزایشهای اولیه نسخه‌های محدود شده محصول نهایی هستند، اما آنها این توانایی را دارند که در خدمت کاربر بوده و همچنین محلی برای ارزیابی توسط کاربر فراهم می‌کنند. تولیدات بصورت فزاینده بطور خاص زمانی مفید است که کارکنان مجبورند تا موعد مقرری کار اجرای پروژه را به اتمام برسانند. افزایش ها را می‌توان با تعداد افراد کمی نیز انجام داد.



هنگامیکه با زمان بندی مشکل غیر قابل تغییری سر و کار دارید، مدل افزایشی، می‌تواند به عنوان رهیافتی مناسب مد نظر قرار گیرد.

۲-۷-۲ مدل پیچشی (حلزونی)

مدل حلزونی که در اصل توسط بوهم [BOE88] پیشنهاد شد یک مدل فرایند نرم‌افزار تکاملی است که ماهیت تکراری نمونه اولیه را با جنبه‌های نظام مند و کنترل شده مدل زنجیری خطی ارتباط

می‌دهد. این مدل پتانسیل لازم را برای تولید سریع نسخه‌های فزاینده نرم افزار، فراهم می‌کند. با استفاده از مدل حلزونی نرم‌افزار در یک سری نسخه‌های فزاینده تولید می‌شود. در طول تکرارهای اولیه، ممکن است نسخه اولیه یک مدل روی کاغذ یا تنها الگوی اولیه باشد. در طول تکرارهای بعدی نسخه‌های نسبتاً کاملتری از سیستم مهندسی شده و تولید می‌شود.

مدل حلزونی از نظر فعالیتی به چند قسمت تقسیم می‌شود که به آنها مناطق کاری^۱ نیز می‌گویند.^۲ معمولاً بین ۳ و ۶ منطقه وجود دارد. شکل ۲-۸ یک مدل حلزونی نشان می‌دهد که دارای شش منطقه است:

- ارتباطات مشتری - کارهای لازم برای ایجاد ارتباط مؤثر بین تولیدکننده و مشتری.
- برنامه‌ریزی - کارهای لازم برای تعریف منابع، محدوده‌های زمانی و دیگر اطلاعات مربوط به پروژه.

- تحلیل خطر - کارهای لازم برای ارزیابی فنی و خطرات مدیریتی.
- مهندسی و طراحی - کارهای لازم برای ساخت یک یا چند نمونه از برنامه کاربردی.
- ساخت و ارائه - کارهای لازم برای ساخت، آزمون، نصب و تهیه پشتیبانی برای کاربر (مثل مستندسازی و آموزش).

- ارزیابی مشتری - کارهای لازم برای بدست آوردن واکنش مشتری براساس ارزیابی نمونه‌های نرم‌افزاری تولید شده در طول مرحله مهندسی و اجرا شده در طول مرحله نصب و راه‌اندازی.

هر یک از این مناطق در اختیار یک سری کارهای مختلف قرار دارند که به آنها مجموعه کاری می‌گویند و خود را با مشخصه‌های هر پروژه‌ای که در دست انجام است، مطابق می‌کنند. درمورد پروژه‌های کوچک، تعداد وظائف کاری و فرم‌الیه بودن آنها کم است. درمورد پروژه‌های بزرگتر و مهم تر هر منطقه کاری دارای مشاغل کاری بیشتری است که به منظور رسیدن به سطح فرم‌الیه بالاتر، تعریف و مشخص شده‌اند. در همه موارد، فعالیتهای پوششی (مثل مدیریت پیکربندی نرم‌افزار و تضمین کیفیت آن) که در بخش ۲-۲ مورد توجه بوده، بکار گرفته می‌شوند.



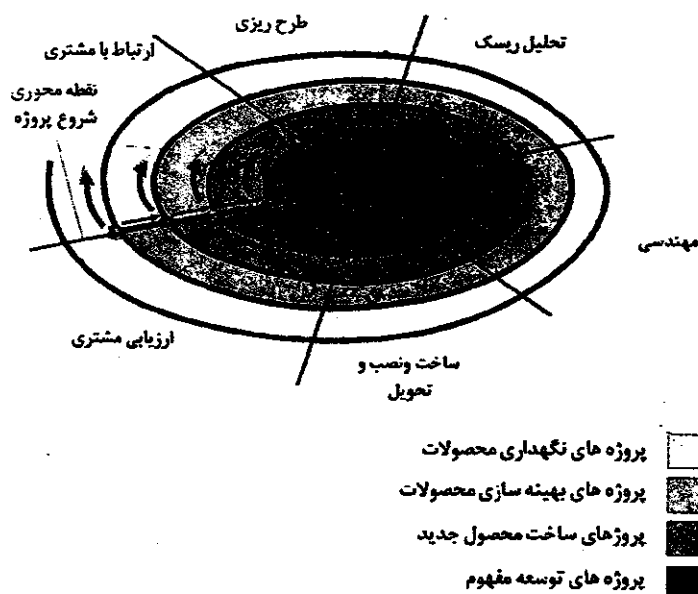
برای هر پروژه نرم
افزاری صرف‌نظر از
سایز (اندازه) یا
پیچیدگی آن،
فعالیت‌های چارچوبی
مورد استفاده دارند



"مجموعه وظائف"
چیست؟

1. task regions

^۲ مدل حلزونی توضیح داده شده در اینجا یکی از مدل‌های متعدد مورد نظر بوهم می‌باشد. برای اطلاع از مدل حلزونی اصلی [BOE88] را ببینید. اطلاعات بیشتر در خصوص مدل حلزونی بوهم، در [BOE98] یافت می‌شود.



شکل ۲-۸ یک مدل حلزونی

با شروع فرایند تکمیلی، تیم مهندسی نرم افزار در جهت حرکت عقربه های ساعت، حرکت در مارپیچ را آغاز می کند و این کار از مرکز شروع می شود. اولین مدار حول مارپیچ ممکن است منجر به تولید مشخصه محصول شود. با طی مسیرهای بیشتری از مارپیچ، از آن ممکن است برای تولید نمونه اولیه استفاده شده و سپس نسخه های پیچیده تری به تدریج تولید شود. با عبور از هر مرحله منطقه برنامه ریزی، کارهای تطبیقی با طرح پروژه صورت می گیرد. هزینه و زمان بندی براساس بازخورد ارزیابی مشتری، تنظیم می گردند. علاوه بر آن، مدیر پروژه تعداد تکرارهای تنظیم شده لازم برای تکمیل نرم افزار را تعیین می کند.



مدل فرایند قابل تطبیق

برخلاف مدل های پردازش کلاسیک که زمانی پایان می پذیرند که نرم افزار به بازار ارائه می شود، مدل حلزونی می تواند کارهای تطبیقی را صورت دهد تا در تمام طول حیات نرم افزار کامپیوتری بکار گرفته شود. دیدگاه دیگر در مورد مدل حلزونی را می توان با بررسی محور نقطه ورودی پروژه^۱ همانطور که در شکل ۸-۲ نشان داده شده، در نظر گرفت. هر مکعبی که در طول محور قرار داده شده است را می توان به عنوان نماینده نقطه شروع برای انواع پروژه های مختلف در نظر گرفت. هر پروژه توسعه مفهوم از مرکز مارپیچ شروع شده و ادامه می یابد (چند تکرار در طول مسیر پیشروی رخ می دهد که منطقه سایه خورده مرکزی را پیوند می دهد) تا وقتی که عمل توسعه و بسط مفهوم کامل می شود. اگر این مفهوم قرار است در یک محصول حقیقی توسعه یابد، فرایند پردازش از طریق مکعب بعدی ادامه می یابد. (نقطه ورود پروژه تولید محصول جدید) و یک پروژه تولیدی جدید آغاز می شود. محصول جدید در طول یک سری تکرار حول

ماربیج مسیر تکمیل می‌شود که این مسیر با منطقهای که سایه کم رنگ‌تری در قسمت مرکزی دارد پیوند می‌خورد. در اصل، مسیر ماربیج وقتی به این شکل طی شد، (محصول) قابل بهره‌برداری باقی می‌ماند تا وقتی که نرم‌افزار دیگر مورد استفاده قرار نگیرد. مولردی وجود دارد که در آن فرایند غیرفعال است اما وقتی تغییری شروع شود، فرایند پردازش از نقطه ورودی مناسب آغاز می‌شود. (مثل بهبود محصول)

مدل پیجشی، روش واقعیت‌های در تولید و توسعه سیستمهای بزرگ و نرم‌افزار است. از آنجا که نرم‌افزار با پیشرفت فرایند تکمیل می‌شود، تولید کننده و مشتری خطرات را بهتر فهمیده و در هر مرحله از کارهای تکمیلی بهتر با خطرات مواجه می‌شوند. مدل پیجشی بعنوان مکانیزمی برای کاهش خطر است اما مهمتر اینکه تولیدکننده را قادر می‌سازد در هر مرحله از تکمیل محصول روش نمونه نخست را بکار گیرد. این کار روش نظام مند مرحله‌ای را که توسط چرخه طول حیات کلاسیک بیان شود حفظ می‌کند، اما آن را در یک چارچوب تکرار شونده بکار می‌گیرد که به شکل واقع بینانه‌تری کلمه واقعی را منعکس می‌سازد. مدل پیجشی نیازمند بررسی مستقیم خطرات فنی در تمام مراحل پروژه است و اگر به درستی بکار گرفته شود باید خطرات را قبل از اینکه مشکل ساز شوند، کاهش دهد.

اما همچون دیگر الگوها، مدل حلزونی داروی تمام دردها نیست. ممکن است مطمئن ساختن مشتری (پروژه در مواقع قرارداد) سخت باشد که روش تکاملی قابل کنترل است. این کار مستلزم تخصص قابل توجهی در ارزیابی خطر است و برای موفقیت به این مهارت نیاز دارد. اگر خطر بزرگی معلوم نشده و کنترل نگردد، بدون شک مشکلاتی بروز می‌کند. در نهایت، مدل بطور گسترده مثل نمونه‌های زنجیری خطی یا نمونه نخست استفاده نشده است. قبل از اینکه بتوان کارایی این نمونه جدید و مهم را با قطعیت تعیین کرد، به سالها وقت نیاز داریم.

۲-۷-۲ مدل حلزونی WINWIN

مدل حلزونی مورد بحث در بخش ۲-۷-۲ بینگر چارچوب فعالیتی است که ارتباطات مشتری را مورد توجه قرار می‌دهد. هدف از این فعالیت عبارتست از بیرون کشیدن اطلاعات درمورد نیازهای پروژه از مشتری در یک بستر مناسب، تولیدکننده صرفاً از مشتری آنچه را که لازم دارد سؤال می‌کند و مشتری جزئیات کافی را برای عمل در اختیار او قرار می‌دهد. متأسفانه این کار به ندرت رخ می‌دهد. درواقعیت، مشتری و تولیدکننده وارد فرایند مذاکره می‌شوند که در آن ممکن است از مشتری قابلیت کارایی، عملکرد و سایر مشخصه‌های سیستم یا محصول در برابر هزینه و بازار در همان موقع سؤال شود.



مدل های اصطلاحی، مانند مدل حلزونی برای توسعه سیستم شیءگرا . بسیار مناسب می باشند. برای تفصیلات بیشتر به بخش چهارم مراجعه نمایید.

بهترین مذاکرات برای یک نتیجه Win-Win (برنده - برنده) مبارزه می‌کند.^۱ یعنی مشتری با بدست آوردن سیستم یا محصولی که اکثر نیازهایش را برطرف می‌سازد برنده شده و تولیدکننده نیز با کار در مدت مقرر و با بودجه‌ای معقول و واقع‌بینانه برنده می‌شود.

مدل حلزونی WINWIN بوهم [BOE98]^۲ یک سری مذاکرات را در آغاز هر گذر از مارپیچ بیان می‌کند. بجای یک رابطه صرف مشتری، کلیه فعالیتهای زیر تعریف می‌شوند:

۱- شناسایی سیستم یا دارندگان اصلی سیستمهای فرعی.^۲

۲- تعیین "شرایط برد" دارندگان سهم

۳- مذاکره درمورد شرایط برد سهامدار برای تطبیق دادن آنها در مجموعه‌ای از شرایط win-

win برای همه افراد درگیر (از جمله تیم پروژه نرم افزاری).

تکمیل موفقیت آمیز این مراحل اولیه به نتیجه win-win می‌انجامد که معیار اصلی برای ارائه

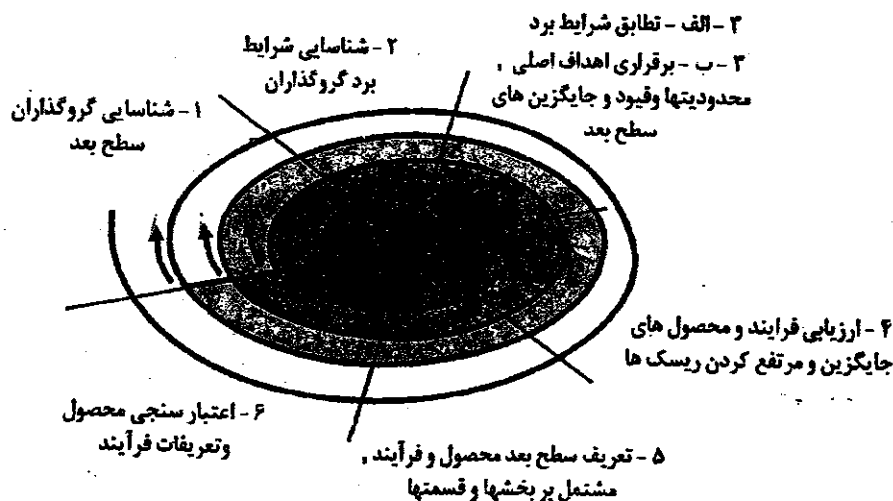
تعریف از سیستم و نرم‌افزار است. مدل حلزونی WINWIN در شکل ۹-۲ تشریح شده است.



استخراج نیازمندیها
نرم افزاری وابسته به
مذاکره است. مذاکرات
موفق یعنی توفیق و
موفقیت طرفین



مهارت های مذاکره



شکل ۲-۹ مدل حلزونی WINWIN

علاوه بر تأکید روی مذاکرات اولیه، مدل حلزونی WINWIN سه معیار فرایند پردازش نیز معرفی می‌کند که نقاط تکیه^۱ یا anchor [BOE96]^۲ نامیده می‌شوند و به تکمیل یک چرخه حول مارپیج کمک نموده و معیارهای تصمیم‌گیری را قبل از انجام پروژه نرم افزاری مهیا می‌کنند.

۱. کتابهای متعددی در خصوص مهارتهای مذاکره به رشته تحریر درآمده (مانند [FIS91] , [DON96] , [FAR97]). یکی از مهمترین چیزهایی که یک مهندس یا مدیر چوآن (یا سالخورده) باید بیاموزد، یکی از آنها را بخوانید.

2. Boehm, B.

۳. یک سهامدار در سازمان، آن کسی است که علاقه تجاری مستقیم و معینی در سیستم یا محصول مورد ساخت دارد. در صورت توفیق شادمان می‌شود و در صورت شکست، نگران و مضطرب می‌گردد.

در اصل، نقاط تکیه نمایانگر سه دیدگاه مختلف از پیشرفت در زمانی هستند که پروژه ماریج را طی می‌کند. اولین تکیه‌گاه که به آن اهداف چرخه حیات^۲ می‌گویند (LCO)، مجموعه‌ای از اهداف را برای هر فعالیت عمده مهندسی نرم‌افزار بیان می‌کند مثل بعنوان بخشی از LCO، مجموعه اهداف مرتبط با تعریف نیازهای سطوح ارشد سیستم/محصول. دومین نقطه

تکیه‌گاه به نام معماری چرخه حیات^۳ (LCA) اهدافی را ارائه می‌دهد که باید با تعریف شدن سیستم و نرم‌افزار مرتفع شوند. مثلاً، بعنوان بخشی از LCA تیم پروژه نرم‌افزاری باید توضیح دهد که قابلیت کارایی اجزای نرم‌افزاری قابل استفاده مجدد و موجود در بازار را برآورده کرده و اثر آنها را روی تصمیمات ساختاری در نظر گرفته است. قابلیت عملیاتی اولیه^۴ (IOC) سومین نقطه تکیه‌گاه است و نمایانگر مجموعه اهداف مربوط به آماده‌سازی نرم‌افزار برای نصب/توزیع، آماده‌سازی سایت قبل از کارهای نصب و راه‌اندازی و کمک موردنیاز از طرف همه طرفه‌ای است که از نرم‌افزار استفاده یا آن را پشتیبانی می‌کنند.

۲-۷-۴ مدل توسعه همروند

مدل تولید همزمان که گاهی به آن مهندسی همروند^۵ گفته می‌شود به شکل زیر توسط دیویس و سیترا^۶ [DAV94] توصیف شده است:

مدیران پروژه‌ای که وضعیت پروژه را از فقط با دنبال کردن مراحل اصلی تعقیب می‌کنند هیچ نظری در مورد وضعیت پروژه‌های خود ندارند. اینها نمونه‌هایی از تلاش برای پیگیری مجموعه فعالیتهای بسیار مشکل با استفاده از مدل‌های بسیار ساده است. توجه داشته باشید که گرچه پروژه در مرحله کدگذاری است اما افرادی در پروژه وجود دارند که در فعالیتهای دیگری مشغولند که معمولاً با بسیاری از مراحل تولید بطور همزمان مرتبط است. مثلاً پرسنل بطور همزمان نیازمندیها را نوشته، کار طراحی، برنامه‌نویسی، آزمون یکپارچه‌سازی را انجام می‌دهند. مدل‌های فرآیند مهندسی نرم‌افزار که توسط هامفری و کلنر^۷ [HUM89]، [KEL89] ارائه شده‌اند، نشانگر همروندی فعالیتهای رخ داده در طول هر مرحله

1. anchor points

2. Boehm, B.

3. Life cycle objective (LCO)

4. Life cycle architecture (LCA)

5. Initial operational capability (IOC)

6. concurrent engineering

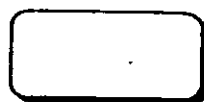
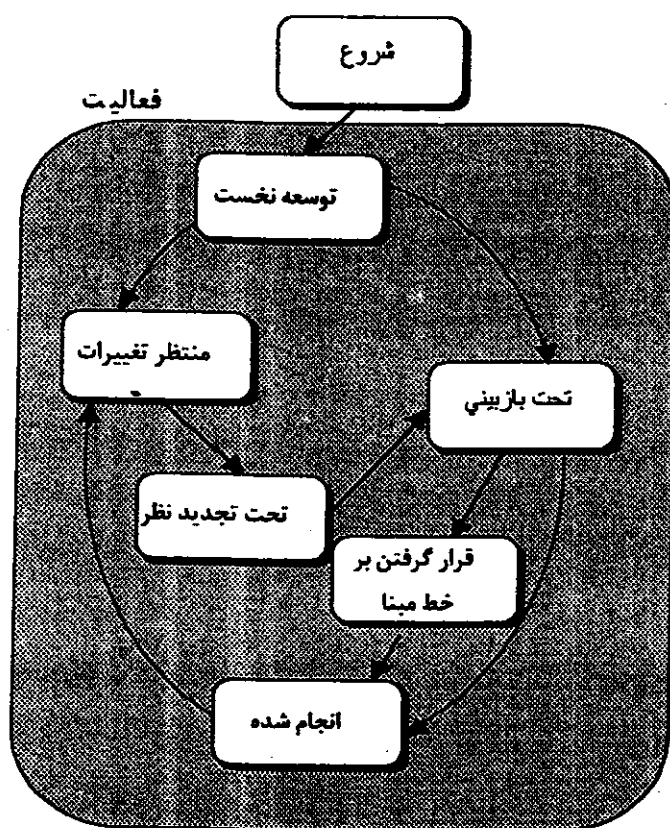
7. Davis, A.

8. Kellner, M.

9. Humphrey, W.

می‌باشند. کار جدیدتر کلنر [KEL91]^۱ از نمودارهایی استفاده می‌کند (عباراتی که بیانگر وضعیت یک فرآیند هستند) که بیانگر ارتباط همزمان موجود بین فعالیتهای مربوط به حادثهای خاص می‌باشند (مثل لزوم تغییر در طول اواخر تولید) اما نمی‌تواند به غنای این همزمانی که در تمام طول تولید نرم‌افزار وجود دارد پی برده و فعالیتهای آن پروژه را مدیریت کند اکثر مدل‌های فرآیند تولید نرم‌افزار مبتنی بر زمان هستند.

هرچه بیشتر تولید طول بکشد، در فرآیند تولید عقبتر هستند. یک مدل فرآیند همزمان ناشی از نیازهای کاربر، تصمیمات مدیریتی و نتایج بازبینی است.



شکل بازنمایی یک وضعیت از یک فعالیت

شکل ۲-۱۰ یک عنصر از مدل فرآیند همروند

مدل فرآیند همزمان را می‌توان به طور شماتیک بصورت مجموعه ای از فعالیتهای عمده فنی، وظایف و سایر موارد مربوطه نشان داد. مثلاً فعالیت مهندسی تعریف شده برای ماریج با انجام کارهای زیر حاصل می‌شود:

مدلسازی تحلیلی و یا نمونه اولیه، تخصیص نیازها و طراحی^۱ شکل ۱-۲۰ نمایش شماتیک از یک فعالیت با مدل فرآیند همزمان است. این فعالیت^۲ که یک تحلیل است ممکن است بصورت هر یک از حالات مورد اشاره در زمان ارائه شده، باشد. به همین شکل، سایر

فعالیت‌های (مثل طراحی یا ارتباط با مشتری) را می‌توان به شکل پیچیده تری نشان داد. تمام فعالیتها بصورت همزمان وجود دارند اما در حالات مختلف قرار می‌گیرند. مثلاً در ابتدای پروژه کار برقراری ارتباط با مشتری^۳ (که در شکل نشان داده نشده) در اولین تکرار تکمیل شد و در وضعیت تغییرات مورد انتظار قرار دارد. کار تحلیل (که در وضعیت شروع none قرار دارد در حالیکه ارتباط اولیه با مشتری تکمیل شده و اکنون وارد وضعیت تحت تولید می‌شود. اگر مشتری اشاره کند که تغییری در نیازها باید صورت بگیرد، کار تحلیل از وضعیت تحت توسعه وارد وضعیت تغییرات مورد انتظار می‌شود.

مدل فرآیند همزمان یک سری رویدادهایی را مشخص می‌کند که انتقال از حالتی به حالت دیگر را برای هر کار مهندسی نرم‌افزار آغاز می‌کنند. مثلاً در طول مراحل اولیه طراحی، یک نوع عدم انسجامی در طول تحلیل مشخص می‌شود. این کار باعث اصلاح مدل تحلیل^۴ حادثه می‌شود که فعالیت تحلیلی^۵ را از وضعیت انجام شده یا شروع وارد وضعیت انتظار می‌کند.

مدل فرآیند همزمان اغلب بعنوان معیاری برای توسعه و تولید برنامه های کاربردی خادم/مخدوم استفاده می‌شود. یک سیستم خادم/مخدوم^۶ متشکل از مجموعه ای اجزای کاربردی است. وقتی مدل فرآیند همزمان در خادم/مخدوم استفاده می‌شود، فعالیت‌هایی را در دو بعد [SHE94] تعریف می‌کند: بعد سیستم و بعد مولفه، موضوعات مربوط به سیستم با استفاده از سه فعالیت مورد توجه قرار می‌گیرند، طراحی، مونتاژ و کاربرد. بعد جزء از نظر دو فعالیت مورد بررسی است: طراحی و شناسایی و همزمانی به دو شکل حاصل می‌شود:

۱- فعالیت‌های سیستم و جزء که بطور همزمان رخ می‌دهند و می‌توان آنها را با استفاده از

رهیافت وضعیت‌گرا که در بالا توصیف شد مدلسازی کرد.

۱. باید توجه داشت که تحلیل و طراحی وظائفی پیچیده هستند که نیازمند توصیفی دقیق و شفاف می‌باشند. بخشهای سوم و چهارم این کتاب این مباحث را به تفصیل ارائه داشته اند.
۲. یک وضعیت، گونه ای قابل رویت از رفتار است.

3. customer communication

4. analysis model correction

5. analysis

۶. در یک برنامه کاربردی خادم / مخدوم، کارکرد نرم افزار بین مخدومها (PC های معمولی) و خادم (یک کامپیوتر قدرتمند) که نوعاً یک پایگاه داده های متمرکز را پشتیبانی می‌کند، تقسیم می‌شود.

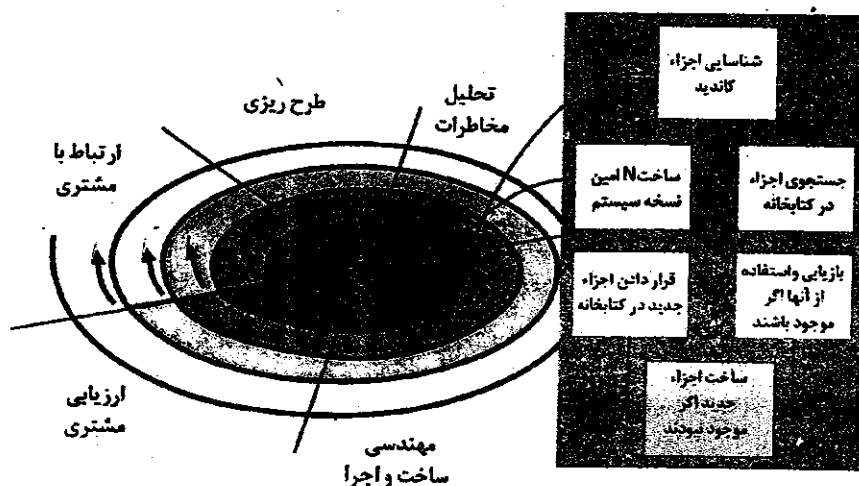
۲- یک برنامه کاربردی معمول خادم/مخدوم بوسیله اجزاء بسیاری به اجرا در می‌آید که هر کدام از آنها را می‌توان بطور همزمان طراحی و شناسایی نمود. در واقعیت، مدل فرایند همزمان در مورد همه نوع تولید نرم‌افزار قابل اجراست و تصویر دقیقی از وضعیت کنونی پروژه را ارائه می‌دهد.

۲-۸ توسعه مبتنی بر اجزاء

فناوریهای مربوط به شیء (بخش ۴ این کتاب) چهارچوب فنی برای مدل و آینده پردازش بر پایه جزء را در مورد مهندسی نرم‌افزار مهیا می‌کنند. پارادایم شیء گرا بر ایجاد کلاس‌هایی تأکید دارد که دارای داده و الگوریتمهایی هستند که برای دستکاری داده‌ها بکار می‌روند. اگر کلاس‌های شیء گرا به درستی طراحی و اجرا شود در برنامه‌های کاربردی مختلف و ساختارهای سیستمهای مبتنی بر کامپیوتر قابل استفاده مجدد می‌باشند.

مدل تولید بر پایه جزء (CBD) (شکل ۲-۱۱) بسیاری از مشخصه‌های مدل حلزونی را دربر دارد. ماهیت آن تکاملی [NIE92]^۱ است، به یک روش تکراری نیاز دارد تا نرم‌افزار را تولید کند. مدل تولید بر پایه جزء، از برنامه‌های کاربردی تشکیل شده است که از اجزای نرم‌افزاری از پیش بسته‌بندی شده‌اند (که به آنها «کلاس یا رده» گفته می‌شود).

فعالیت مهندسی با شناسایی کلاس‌های کاندید آغاز می‌شود. این کار با بررسی داده‌هایی صورت می‌گیرد که باید توسط برنامه کاربردی مورد دستکاری قرار بگیرند و الگوریتمهایی که برای موفقیت این کارها بکار گرفته می‌شوند.^۲ داده و الگوریتمهای مربوطه در یک کلاس قرار می‌گیرند.



شکل ۲-۱۱ توسعه مبتنی بر اجزاء



فن آوری توسعه بر مبنای اجزاء (CBD) در بخش چهارم کتاب حاضر، تشریح شده است. توضیحات تکمیلی را در فصل ۲۷ دنبال کنید.

1. Nierstrasz, O.

۲. این یک توصیف ساده از تعریف کلاس است. برای توضیح بیشتر فصل ۲۰ را مطالعه فرمائید.

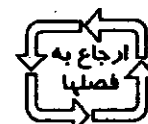
کلاسهایی که در پروژه‌های مهندسی نرم‌افزار پیشین ساخته شده‌اند، در یک کتابخانه یا مخزن کلاس ذخیره می‌شوند (فصل ۳۱). هنگامیکه کلاس‌های کلانید شناسایی شدند، کتابخانه کلاس مورد جستجو قرار می‌گیرد تا معلوم شود چنین کلاس‌هایی وجود دارند یا خیر. اگر آنها وجود دارند از کتابخانه استخراج شده و مجدداً استفاده می‌شوند. اگر کلاس کلانیدی در کتابخانه وجود نداشت بوسیله روشهای شیء گرا ایجاد می‌شود. اولین تکرار هر برنامه کاربردی که باید ساخته شود با استفاده از کلاس‌های استخراج شده از کتابخانه و هرگونه کلاس جدیدی که برای رفع نیازهای منحصر به فرد برنامه کاربردی ایجاد می‌شود، تنظیم می‌گردد. جریان فرآیند به ماریج باز می‌گردد و نهایتاً مجدداً وارد تکرار مونتاژ جزء می‌شود که در حین عبور بعدی از میان کار مهندسی صورت می‌گیرد.

مدل تولید بر پایه جزء منجر به استفاده مجدد نرم‌افزار شده و این قابلیت بکارگیری مجدد تعدادی مزایای قابل توجه در اختیار مهندسين نرم‌افزار قرار می‌دهد. براساس تحقیقات مربوط به قابلیت استفاده مجدد شرکت QSM Associates گزارش می‌کند که مونتاژ جزء منجر به کاهش ۷۰ درصد در زمان چرخه تولید می‌شود، همچنین یک کاهش ۸۴ درصدی در هزینه پروژه و شاخص بهره‌وری ۲۶/۲ درصدی می‌دهد که در مقایسه با نورم صنعتی ۱۶/۹ [YOU94] درصد است. گرچه این نتایج تابع قدرت و ثبات کتابخانه اجزاء است اما شک چندانی وجود ندارد که مدل تولید بر پایه جزء مزایای بسیاری برای طراحان نرم‌افزاری مهیا می‌کند.

فرآیند تولید یکپارچه نرم‌افزار^۲ [JAC99] نماینده تعدادی از مدل‌های تولید بر پایه اجزاء است که در صنعت پیشنهاد شده‌اند. با استفاده از زبان مدلسازی یکپارچه^۳ یا UML، فرآیند یکپارچه جزءهایی را تعریف می‌کند که برای ساختن سیستم استفاده شده و رابط‌هایی که این جزءها را به هم وصل می‌کنند. با استفاده از تولید فزاینده و تکراری، فرآیند یکپارچه تابع سیستم را با بکارگیری رهیافت مبتنی بر سناریو تعریف می‌کند. (از دیدگاه کاربر). سپس تابع را با یک چارچوب معماری و ساختاری مرتبط می‌کند که فرم نرم‌افزار را مشخص می‌سازد.



UML به تفصیل
در بخش‌های ۲۱ و ۲۲
آورده شده است.



شیوه‌های رسمی در
فصل‌های ۲۵ و ۲۶
ارائه گردیده‌اند.

۲-۹ مدل شیوه‌های رسمی

مدلی که بر پایه روشهای رسمی بدست می‌آید دربر گیرنده یک سری فعالیتهایی است که منجر به تعیین رسمی مشخصه‌های نرم‌افزار کامپیوتر با روابط ریاضی می‌شود. روشهای رسمی مهندس نرم‌افزار را قادر می‌سازند که یک سیستم مبتنی بر کامپیوتر را با بکارگیری عبارات ریاضی بسیار دقیق مشخص،

1. Yourdon, E.

2. unified software development procces

3. Jacobson, I, Booch, G.

4. unified modeling language(UML)

توسعه داده و تغییر دهد. یک نوع از این روش به نام مهندسی نرم‌افزار اتاق پاک^۱ [MIL87, DYE92] در حال حاضر بوسیله بعضی از سازمانهای تولید نرم‌افزار مورد استفاده قرار می‌گیرد.

وقتی از روشهای رسمی در طول تولید استفاده می‌شود آنها مکانیزمی برای حذف بسیاری از مشکلات ارائه می‌دهند که فائق شدن بر آنها از طریق بکارگیری سایر پارادایم نرم‌افزاری سخت است. البته، ناقص بودن و نداشتن انسجام و ناسازگاری را می‌توان کشف و راحت‌تر اصلاح نمود. نه از طریق بازبینی بی‌پایه و اساس بلکه از طریق برنامه کاربردی تحلیل ریاضی. وقتی که از روشهای رسمی در طول طراحی استفاده می‌شود آنها بعنوان پایه‌ای برای اصلاح برنامه عمل کرده و بنابراین مهندس نرم‌افزار را قادر می‌سازند خطاهایی که ممکن است ندیده مانده باشند کشف و اصلاح کند.

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

- ۱- توسعه روشهای رسمی در حال حاضر کاملاً وقت گیر و گران است.
 - ۲- از آنجا که تولیدکنندگان کمی درمورد نرم‌افزار دارای پیش زمینه کافی برای استفاده از روشهای رسمی هستند. یک دوره آموزش پرهزینه لازم است.
 - ۳- استفاده از مدلها بعنوان یک مکانیزم ارتباطی برای مشتریان غیر کارگشته از نظر فنی، مشکل است.
- علیرغم این مشکلات، احتمال دارد که روش‌های رسمی در میان تولیدکنندگان نرم‌افزاری که باید نرم‌افزار مهم بی خطری را بسازند (مثل تولیدکنندگان تجهیزات الکترونیکی هواپیما و وسایل پزشکی) و در میان تولیدکنندگانی که در صورت بروز خطا دچار ضرر و زیان اقتصادی شدید می‌شوند، طرفدارانی پیدا کند.

۲-۱۰ فنون نسل چهارم

اصطلاح فنون نسل چهارم^۲ (UGT) در بر گیرنده طیف گسترده‌ای از ابزارهای نرم‌افزاری است که دارای یک چیز مشترک هستند: هرکدام به مهندس نرم‌افزار این قدرت را می‌دهند که یک مشخصه نرم‌افزاری را در سطح بالایی مشخص سازد. سپس ابزار بطور خودکار کد منبع را براساس مشخصه‌های تولیدکننده تولید می‌نماید. درمورد این مسئله اختلاف نظر کمی وجود دارد که هرچه سطح مشخص شده برای نرم‌افزار نسبت به ماشین بالاتر باشد، برنامه به صورت سریعتری نوشته می‌شود. معیار 4GT درمورد برنامه نویسی نرم‌افزاری روی توانایی مشخص کردن نرم‌افزار با استفاده از فرم های زبانی مخصوص یا یک

1. cleanroom software engineering

2. Mills, H.D. Dyer, M.

3. fourth generation techniques

عبارت گرافیکی متمرکز می‌شود که مشکلی را که باید حل شود طوری حل می‌کند که مشتری متوجه آن شود.

در حال حاضر، یک محیط تولید نرم‌افزار که معیار 4GT را مورد پشتیبانی قرار می‌دهد شامل چند یا کل ابزار زیر است: زبانهای غیر رویه‌ای برای تحقیقات در پایگاه داده‌ای، ارائه گزارش، دستکاری و تغییر داده‌ها، تعریف و ارتباط متقابل صفحه‌ای، ارائه کد، توانایی گرافیکی سطح بالا، توانایی صفحه گسترده و تولید خودکار HTML و زبانهای مشابه استفاده شده برای ایجاد سایت وب با استفاده از ابزارهای پیشرفته نرم‌افزاری. در ابتدا، بسیاری از ابزارهای مورد اشاره فوق تنها برای حوزه‌های کاربردی خاصی مهیا بودند، اما امروزه محیطهای 4GT بسط یافته‌اند تا اکثر طبقه‌بندیهای برنامه‌های کاربردی نرم‌افزاری را مورد توجه قرار دهند.

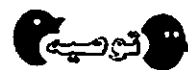
همچون دیگر نمونه‌ها، 4GT با یک مرحله جمع‌آوری نیازها آغاز می‌شود. ایده‌آل این است که مشتری نیازهای خود را شرح دهد و این نیازها مستقیماً به یک نمونه عملیاتی تبدیل شوند. اما این کار در عمل قابل اجرا نیست. ممکن است مشتری از آنچه که لازم دارد مطمئن نباشد، ممکن است در مشخص کردن حقایق شناخته شده ابهام وجود داشته و قادر یا مشتاق به مشخص کردن اطلاعات به شیوه‌ای نباشد که ابزار 4GT بتواند از آنها استفاده کند. به همین دلیل، گفتگوی بین مشتری و تولیدکننده که برای دیگر مدل‌های فرایند توصیف شد بعنوان بخش ضروری از روش 4GT باقی می‌ماند.

در مورد برنامه‌های کاربردی کوچک، این امکان وجود دارد که مستقیماً از سوی نیازهای جمع‌آوری شده در مرحله اول با استفاده از یک زبان غیر رویه‌ای نسل چهارم یا مدل بدست آمده از شبکه ایکونهای گرافیکی به سوی اجرا حرکت کنیم. در کارهای بزرگتر لازم است یک شیوه طراحی را برای سیستم ارائه دهیم حتی اگر قرار است از 4GL استفاده کنیم. استفاده از 4GL بدون طرح (برای پروژه‌های بزرگ) باعث همان مشکلاتی می‌شود که در هنگام تولید نرم‌افزار با استفاده از روشهای متعارف به آنها برخوردیم.

اجرا با استفاده از 4GL تولید کننده نرم‌افزار را قادر می‌سازد که نتایج موردنظر را به شیوه‌ای ارائه دهد که منجر به تولید خودکار کد برای ارائه آن نتایج شود. مشخصاً، ساختار داده‌ای با اطلاعات مربوطه باید وجود داشته و توسط 4GT به راحتی مهیا باشد.

برای تبدیل عملیات اجرایی 4GL به یک محصول، باید تولید کننده از طریق آزمون، تولید اسناد معنادار و انجام همه راه‌حل‌های مجتمع سازی که در سایر نمونه‌های مهندسی نرم‌افزار نیز لازمند، کار را انجام دهد. علاوه بر آن، نرم‌افزار تولید شده با 4GT باید به شکلی ساخته شود که امکان نگهداری و تعمیر سریع آن وجود داشته باشد.

همچون سایر نمونه‌های مهندسی نرم‌افزار، مدل 4GT دارای مزایا و معایبی می‌باشد. طرفداران آن کاهش شدید زمان تولید و میزان بهره‌وری بسیار بیشتری را برای افرادی که نرم‌افزار تولید می‌کنند، ادعا کرده‌اند.



حتی هنگامی که از فنون نسل چهارم بهره می‌برید، بازهم تحلیل و طراحی و تست را به عنوان مباحث مهندسی نرم‌افزار در نظر داشته باشید.

مخالفان نیز ادعا می‌کنند که ابزارهای کنونی 4GT طرز استفاده نه چندان ساده‌ای نسبت به زبانهای برنامه‌نویسی دارند، که منبع منتج از چنین ابزارهایی، کارایی ندارد و قابلیت نگهداری آن در سیستمهای بزرگ ارائه شده با استفاده از 4GL مورد تردید است.

در ادعاهای بیان شده از هر دو طرف امتیازاتی وجود دارد و مزایای و معایب آن را باید در 4GT را به صورت زیر خلاصه نمود:

۱- استفاده از روش 4GT روش بسیار ارزشمندی برای بسیاری از حوزههای مختلف در برنامههای کاربردی است. 4GT همراه با ابزارهای مهندسی نرم‌افزار با کمک کامپیوتر (CASE) و مولدهای کد، یک راه حل خوب برای بسیاری از مشکلات نرم‌افزاری ارائه می‌دهد.

۲- اطلاعات جمع‌آوری شده از شرکتهایی که از 4GT استفاده می‌کنند نشانگر این است که زمان لازم برای تولید نرم‌افزار برای برنامههای کاربردی کوچک و متوسط تا حد زیادی کاهش یافته و میزان طرح و تحلیل برای برنامههای کوچک نیز کم شده است.

۳- استفاده از 4GT برای کارهای تولید نرم‌افزار در مقیاس بزرگ نیازمند تحلیل، طراحی و آزمون بیشتر است تا صرفه‌جویی بیشتری در وقت صورت گیرد که این کار از طریق حذف کدها بوجود می‌آید.

بطور خلاصه، فنون نسل چهارم هم اکنون بخش مهمی از مهندسی نرم‌افزار شده‌اند. وقتی 4GT با روشهای تولید براساس اجزاء همراه شود ممکن است به روش بزرگی برای تولید نرم‌افزار تبدیل شود.

۲-۱۱ فناوری فرایند

مدلهای فرایند مورد بحث در بخشهای قبلی باید تحت انطباقانی قرار گیرند تا در یک پروژه نرم‌افزاری استفاده شوند. برای این کار، ابزارهای فناوری فرایند، تولید شده‌اند تا به سازمانهای نرم‌افزاری کمک کنند فرایند پردازش کنونی خود را تحلیل کرده، کارهای مختلف را سازماندهی کنند، میزان پیشرفت را کنترل و مشاهده نمایند و کیفیت فنی را سامان دهند.

ابزارهای فناوری فرایند به سازمان مربوطه امکان می‌دهد تا یک مدل خود کار در یک چارچوب عادی در فرایند، مجموعه وظایف و فعالیتهای پوششی مورد بحث در بخش ۲-۳ را آماده کنند. مدل که معمولاً بوسیله یک شبکه به نمایش در می‌آید می‌تواند مورد تجزیه و تحلیل قرار گیرد تا جریان کاری را معین نموده و ساختارهای فرایند جایگزین را که ممکن است به کاهش زمان یا هزینه تولید بیانجامد، بررسی کند.

وقتی که یک فرایند قابل قبول بوجود آمد، سایر ابزارهای فناوری فرایند را می‌توان برای تخصیص، کنترل و مشاهده و حتی کنترل تمام کارهای مهندسی نرم‌افزاری تعریف شده بعنوان قسمتی از مدل

فرآیند، استفاده کرد. هر عضو تیم تولید نرم‌افزار می‌تواند از چنین ابزارهایی برای تولید فهرستی از کارهایی که باید انجام دهد، محصولاتی که باید تولید شوند و تضمین کیفی فعالیت‌هایی که باید صورت گیرند، استفاده کند. ابزار فناوری فرآیند باید برای هماهنگی کاربرد سایر ابزارهای مهندسی نرم‌افزار در قسمت کمکی که برای کار بخصوصی لازمند، نیز استفاده شود.

۱۲-۲ محصول و فرآیند

اگر فرآیند ضعیف است، محصول نهایی بدون شک دچار مشکل می‌شود. اما تکیه وسواس گونه به یک فرآیند نیز خطرناک است. در یک مقاله کوتاه، مارگارت دیویس [DAV95]^۱ دوگانگی محصول و فرآیند را شرح می‌دهد:

حدود هر ده یا پنج سال، جامعه نرم‌افزاری مشکلات را با تغییر نقطه اصلی تمرکز از موضوعات مربوط به محصول، مجدداً تعریف می‌کند. بنابراین ما دارای زبانهای برنامه نویسی نظامندی هستیم که به دنبال آن روشهای تحلیل منظم می‌آیند که به دنبال آن احاطه بر داده‌ها را داریم که تأکید کنونی روی مدل کامل توانایی تولید نرم‌افزار در مؤسسه مهندسی است.

تمایل یک آهنگ این است که در نقطه وسط میان دو سر حد بایستد، نقطه تمرکز جامعه نرم‌افزاری مرتباً دستخوش تغییر می‌شود زیرا نیروی جدیدی در هنگام شکست آخرین حرکت، وارد می‌شود. این حرکتها و نوسانات بخودی خود مضرند زیرا با تغییر شدید آنچه که به معنای انجام کار جدا از انجام دادن خوب آن هست، مهارت متخصص نرم‌افزاری را آشفته می‌سازد. همچنین این حرکات مشکل را حل نمی‌کنند. زیرا آنها تا وقتی که محصول و فرآیند بجای یک دوگانگی دارای یک تفکیک هستند، محکوم به شکست می‌باشند.

در جامعه علمی اولویت برای پیشرفت عبارات دوگانه‌ای است که ضد و نقیض بودن آنها در مشاهدات نمی‌تواند بطور کامل با یک تئوری رقابتی یا چیز دیگر تشریح شود. ماهیت دوگانه نور که بنظر می‌رسد بطور همزمان ذره و موج باشد، از سال ۱۹۲۰ پذیرفته شد وقتی که لوئیس دبروئر این مسئله را بیان داشت. من معتقدم مشاهداتی که می‌توانیم از مصنوعات نرم‌افزاری و توسعه آنها داشته باشیم یک دوگانگی اسامی را بین محصول و فرآیند تشریح می‌کند. شما هرگز نمی‌توانید کل مصنوع، بافت، کار، مفهوم و ارزش آن را درک کنید اگر که تنها به آن بعنوان یک محصول یا فرآیند نگاه کنید.

ممکن است تمام فعالیت‌های انسان یک فرآیند باشد، اما هرکدام از ما دارای یک حس خود ارزشی در مورد فعالیت‌هایی هستیم که منجر به نمایش مولردی می‌شوند که می‌توانند توسط یک یا چند نفر مورد استفاده یا ارزشیابی قرار گیرند و بارها و بارها در بستر یا زمانی که در نظر نگرفته‌ایم

1. Davis, M. J.

مجدداً استفاده شوند. یعنی ما با استفاده مجدد از آن محصولات توسط خود یا دیگران، به رضایت خاطر می‌رسیم.

بنابراین، در عین حالیکه تلفیق سریع اهداف ناشی از استفاده مجدد با تولید نرم‌افزار بطور بالقوه میزان رضایت متخصصین نرم‌افزار را از کارشان افزایش می‌دهد، این کار همچنین لزوم پذیرش دوگانگی محصول و فرآیند را نیز بالا می‌برد. تصور مصنوع قبل استفاده مجددی چه بصورت محصول یا چه به صورت یک فرآیند نه تنها بافت و شیوه‌های بکارگیری آن را می‌پوشاند بلکه این حقیقت را نیز مخدوش می‌کند که هر استفاده منجر به محصولی می‌شود که به نوبه خود بعنوان ماده اولیه یک فعالیت تولیدی نرم‌افزاری دیگر استفاده می‌شود.

داشتن نظرات مختلف تا حد زیادی فرصت بکارگیری مجدد یک چیز را کاهش داده و فرصت

افزایش رضایت شغلی را نیز از بین می‌برد.

مردم به اندازه محصول نهایی از یک فرآیند نوآورانه نیز احساس رضایت می‌کنند. هنرمند علاوه بر تصویر نهایی از هرگونه قلمی که بر کاغذ می‌زند احساس خرسندی می‌کند. یک نویسنده از جستجوی برای یافتن کتایه متناسب، به اندازه کل کتاب تمام شده، لذت می‌برد. یک متخصص مبتکر نرم‌افزار باید از فرآیند کار به اندازه محصول نهایی احساس رضایت کند.

در آینده کار افراد متخصص نرم‌افزار تغییر خواهد کرد. دوگانگی محصول و فرآیند عامل مهمی در حفظ افراد خلاق درگیر در کار است و این زمانی است که انتقال از برنامه‌نویسی به مهندسی نرم‌افزار تمام می‌شود.

۲-۱۳ خلاصه

مهندسی نرم‌افزار یک دیسپلین است که فرآیند، مدلها و ابزارهایی را برای تولید نرم‌افزار کامپیوتری ارتقاء می‌بخشد. تعدادی از مدل‌های فرآیندهای مختلف برای مهندسی نرم‌افزار پیشنهاد شده‌اند که هر کدام دارای نقاط ضعف و قوتی هستند اما همگی دارای یک سری مراحل کلی مشترک می‌باشند. اصول، مفاهیم و روشهایی که ما را قادر می‌سازند فرآیندی را انجام دهیم که مهندسی نرم‌افزار آن را در سراسر باقی این کتاب مدنظر دارند.

نقل قول

هر فعالیتی می‌تواند سازنده باشد، مشروط بر آنکه انجام دهنده آن به درستی و بهتر شده آن، بیاندهد. جان ایلبیک

مسایل و نکاتی برای تفکر و تعمق بیشتر

۱-۲ شکل ۱-۲ سه لایه مهندسی نرم‌افزار را در صدر کل مطالب قرار می‌دهد که نقطه تمرکز کیفی است. این کار نشانگر یک برنامه کیفی سازمانی، مدیریت کیفی کل است. کمی تحقیق کنید و یک طرح کلی از اهداف اصلی برنامه مدیریت کیفی کل ارائه دهید.

۲-۲ آیا موردی وجود دارد که در آن مراحل کلی فرآیند مهندسی نرم‌افزار بکار گرفته نشوند؟ اگر اینطور است آن را توصیف کنید

۳-۲ مدل تکامل قابلیت و توانایی در SEI یک سند تکمیل شده است. کمی تحقیق کرده و معلوم کنید که آیا هر گونه KPA جدیدی از تاریخ انتشار این کتاب اضافه شده است یا خیر.

۴-۲ مدل بدون نظم بیانگر این است که یک حلقه حل مشکل را می‌توان در مرحله‌ای بکار گرفت. درمورد شیوه‌ای که شما این چرخه را بکار می‌گیرید بحث کنید. (۱) شرایط را درمورد یک محصول بردار شگر کلمه شناسایی کنید (۲) یک جزء بازبینی کننده نکات دستوری و دیکته‌ای برای آن ارائه دهید. (۳) یک کد برای مدل برنامه بگذارید که فاعل، گزاره و مفعول را در یک جمله انگلیسی مشخص کند.

۵-۲ کدامیک از معیارهای مهندسی نرم‌افزار ارائه شده در این فصل مؤثر از هم هستند و چرا؟

۶-۲ پنج مثال از پروژه‌های تولید نرم‌افزار ارائه دهید که در نمونه اولیه قابل اصلاح باشند. دو یا سه برنامه کاربردی نام ببرید که نسبت به نمونه نخست مشکل تر باشند.

۷-۲ مدل RAD اغلب در رابطه با ابزارهای CASE است. مقاله را مورد تحقیق قرار داده و خلاصه‌ای از ابزار عادی CASE را که RAD را مورد پشتیبانی قرار می‌دهد ارائه دهید.

۸-۲ یک پروژه نرم‌افزاری خاص را پیشنهاد دهید که در مدل فزاینده قابل اصلاح باشد. طرحی برای بکارگیری مدل در نرم‌افزار ارائه دهید.

۹-۲ با حرکت شما به سوی خروجی مسیر جریان ردایش مدل حلزونی، درمورد نرم‌افزار تولید شده یا تعمیر شده چه می‌توانید بگویید؟

۱۰-۲ بسیاری از مردم معتقدند که تنها شیوه بهبود کیفیت نرم‌افزار و میزان بهره‌وری از طریق تولید بر پایه اجزاء است. در این مورد سه یا چهار مقاله یافته و آنها را برای کلاس بصورت خلاصه درآورید.

۱۱-۲ مدل تولید هم زمان را از زبان خود توصیف کنید.

۱۲-۲ سه مثال درمورد فنون نسل چهارم ارائه دهید.

۱۳-۲ یک از این دو مهم تر است - محصول یا فرآیند؟

فهرست منابع و مراجع

- [BAE98] Baetjer, Jr., H., *Software as Capital*, IEEE Computer Society Press, 1998, p.85.
- [BAN95] Bandinelli, S. et al., "Modeling and Improving an Industrial Software Process," *IEEE Trans. Software Engineering*, vol. SE-21, no. 5, May 1995, pp. 440-454.
- [BOE88] Boehm, B., "A Spiral Model for Software Development and Enhance-ment," *Computer*, vol. 21, no. 5, May 1988, pp. 61-72.
- [BOE96] Boehm, B., "Anchoring the Software Process," *IEEE Software*, vol. 13, no. 4, July 1996, pp. 73-82.
- [BOE98] Boehm, B., "Using the WINWIN Spiral Model: A case Study," *Computer*, vol. 31, no. 7, July 1998, pp. 33-44.
- [BRA94] Bradac, M., D. Perry, and L. Votta, "Prototyping a Process Monitoring Experiment," *IEEE Trans. Software Engineering*, vol. SE-20, no. 10, October 1994, pp. 774-784.
- [BRO75] Brooks, F., *The Mythical Man-Month*, Addison-Wesley, 1975.
- [BUT94] Butler, J., "Rapid Application Development in Action," *Managing System Development*, Applied Computer Research, vol. 14, no. 5, May 1994, pp. 6-8.
- [DAV94] Davis, A. and P. Sitaram, "A Concurrent Process Model for Software Development," *Software Engineering Notes*, ACM Press, vol. 19, no. 2, April 1994, pp.38-51.
- [DAV95] Davis, M.J., "Process and Product: Dichotomy or Duality," *Software Engineering Notes*, ACM Press, vol. 20, no. 2, April 1995, pp. 17-18.
- [DON96] Donaldson, M.c. and M. Donaldson, *Negotiating for Dummies*, IOG Books Worldwide, 1996.
- [DYE92] Dyer, M., *The Cleanroom Approach to Quality Software Development*, Wiley, 1992.
- [FAR97] Farber, D.c., *Common Sense Negotiation: The Art of Winning Gracefully* Bay Press, 1997.
- [FIS91] Fisher, R, W. Ury, and B. Patton, *Getting to Yes: Negotiating Agreement Without Giving In*, 2nd edition, Penguin USA, 1991.
- [GIL88] Gilb, T., *Principles of Software Engineering Management*, Addison-

Wesley,
1988.

[HAN95] Hanna, M., "Farewell to Waterfalls," *Software Magazine*, May 1995,
pp.

38-46.

[HUM89] Humphrey, W. and M. Kellner, "Software Process Modeling:
Principles of

Entity Process Models," *Proc. 11 th Intl. Conference on Software
Engineering*, IEEE

Computer Society Press, pp. 331-342.

[IEEE93] *IEEE Standards Collection: Software Engineering*, IEEE Standard
610.12-

1990, IEEE, 1993.

[JAC99] Jacobson, I., Booch, G., and J. Rumbaugh, *The Unified Software
Develop-*

ment Process, Addison-Wesley, 1999.

[KEL89] Kellner, M., *Software Process Modeling: Value and Experience*, SEI
Techni-

cal Review-I 989, SEI, 1989.

[KEL91] Kellner, M., "Software Process Modeling Support for Management
Plan

ning and Control," *Proc. 1st Intl. Conf. on the Software Process*, IEEE
Computer

Society Press, 1991, pp. 8-28.

[KER94] Kerr, J. and R Hunter, *Inside RAD*, McGraw-Hill, 1994.

[MAR91] Martin, J., *Rapid Application Development*, Prentice-Hall, 1991.

[MDE93] McDermid, J. and P. Rook, "Software Development Process
Models," in

Software Engineer's Reference Book, CRC Press, 1993, pp. 15/26-15/28.

[MIL87] Mills, H.D., M. Dyer, and R Linger, "Cleanroom Software
Engineering,"

IEEE Software, September 1987, pp. 19-25.

[NAU69] Naur, P. and B. Randall (eds.), *Software Engineering: A Report on a
Confer-*

ence Sponsored by the NATO Science Committee, NATO, 1969.

[NIE92] Nierstrasz, O., S. Gibbs, and D. Tsichritzis, "Component-Oriented
Soft-

ware Development," *CACM*, vol. 35, no. 9, September 1992, pp. 160-165.

[PAU93] Paulk, M. et al., "Capability Maturity Model for Software," *Software
Engi-*

neering Institute, Carnegie Mellon University, Pittsburgh, PA, 1993.

[RAC95] Raccoon, L.B.S., "The Chaos Model and the Chaos We Cycle," *ACM
Soft-*

ware Engineering Notes, vol. 20., no. 1, January, 1995, pp. 55-66.

[ROY70] Royce, ww, "Managing the Development of Large Software
Systems:

Concepts and Techniques," *Proc. WESCON*, August 1970.

[SHE94] Sheleg, W., "Concurrent Engineering: A New Paradigm for C/S
Develop-

ment," *Application Development Trends*, vol. I, no. 6, June 1994, pp. 28-33.

[YOU94] Yourdon, E., "Software Reuse," *Application Development Strategies*, vol. 6, no. 12, December 1994, pp. 1-16.

خواندنیهای دیگر و منابع اطلاعاتی

The current state of the art in software engineering can best be determined from monthly publications such as *IEEE Software*, *Computer*, and the *IEEE Transactions on Software Engineering*. Industry periodicals such as *Application Development Trends*, *Cutter IT journal* and *Software Development* often contain articles on software engineering topics. The discipline is 'summarized' every year in the *Proceedings of the International Conference on Software Engineering*, sponsored by the IEEE and ACM and is discussed in depth in journals such as *ACM Transactions on Software Engineering and Methodology*, *ACM Software Engineering Notes*, and *Annals of Software Engineering*.

Many software engineering books have been published in recent years. Some present an overview of the entire process while others delve into a few important topics to the exclusion of others. Three anthologies that cover a wide range of software engineering topics are

Keyes, J., (ed.), *Software Engineering Productivity Handbook*, McGraw-Hill, 1993.

McDermid, J., (ed.), *Software Engineer's Reference Book*, CRC Press, 1993.

Marchiniak, J.J. (ed.), *Encyclopedia of Software Engineering*, Wiley, 1994.

Gautier (*Distributed Engineering of Software*, Prentice-Hall, 1996) provides suggestions

and guidelines for organizations that must develop software across geographically dispersed locations.

On the lighter side, a book by Robert Glass (*Software Conflict*, Yourdon Press, 1991)

presents amusing and controversial essays on software and the software engineering process. Pressman and Herron (*Software Shock*, Dorset House, 1991)

consider software and its impact on individuals, businesses, and government.

The Software Engineering Institute (SEI is located at Carnegie-Mellon University)

has been chartered with the responsibility of sponsoring a software engineering mono-

graph series. Practitioners from industry, government, and academia are contribut

این کتاب تنها به خاطر حل مشکل دانشجویان پیام نور تبدیل به پی دی اف شد. همین جا از ناشر و نویسنده و تمام کسانی که با افزایش قیمت کتاب ما را مجبور به این کار کردند و یا متحمل ضرر شدند عذرخواهی می کنم.
گروهی از دانشجویان مهندسی کامپیوتر مرکز تهران