فصل ۳ فرآیند نرم‌افزار

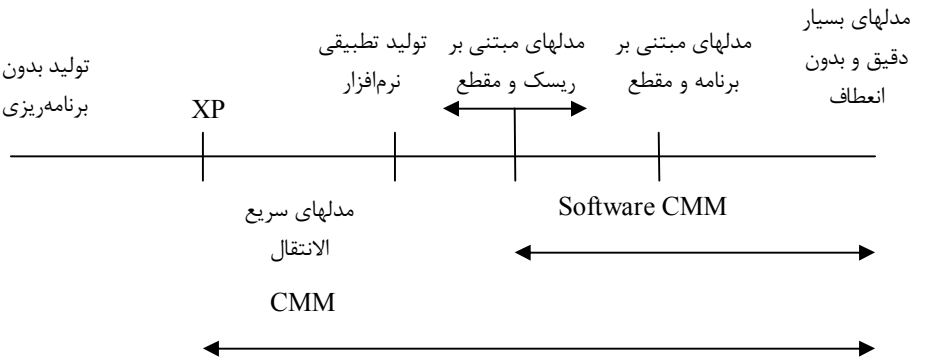
مقدمه

فرآیند نرم‌افزار[1] روشی برای ساخت یا تولید نرم‌افزار است. فرآیند نرم‌افزار مجموعه‌ای از گامهای قابل پیش بینی به منظور ساخت نرم‌افزاری مقرون به صرفه و با کیفیت است. مدل فرآیند نرم‌افزار یا مدل چرخهٔ عمر[2]، استراتژی ساخت نرم‌افزار است که در برگیرندهٔ فرآیند، روشها و ابزارهای مهندسی نرم‌افزار می‌باشد. در هر مدل فرآیند نرم‌افزار، مراحل عمومی فعالیت مهندسی نرم‌افزار یعنی تعریف، ساخت و پشتیبانی تعریف می‌شود. پرسشی که مطرح می‌شود این است که چرا باید از مدلهای فرآیند به منظور تولید نرم‌افزار استفاده نمود. به طور خلاصه می‌توان مزایا و کاربردهای مدل فرآیند نرم‌افزار در انجام پروژه‌های نرم‌افزاری را به شرح زیر دانست:

- مدل فرآیند نرم‌افزار امکان سازمان‌دهی، برنامه‌ریزی، بودجه‌بندی، زمان‌بندی و مدیریت پروژه‌های نرم‌افزاری را فراهم می‌آورد.
- مدل فرآیند نرم‌افزار محصولاتی را که در هر مقطع از پروژه باید تولید شوند، مشخص می‌نماید.
- براساس مدل فرآیند نرم‌افزار می‌توان مشخص نمود که برای پشتیبانی فعالیتهای متفاوت ساخت نرم‌افزار از چه ابزارها و روشهای مهندسی باید استفاده کرد.
- مدل فرآیند چارچوبی برای تحلیل یا تخمین الگوهای تخصیص منابع و هزینه منابع در طی چرخهٔ تولید نرم‌افزار است.
- مدل فرآیند پایه‌ای تجربی برای تعیین عوامل موثر بر بهره‌وری، هزینه و کیفیت کلی نرم‌افزار است.

[1] Software process
[2] life cycle model

همانگونه که در شکل ۳-۱ دیده می‌شود، Boehm مدلهای فرآیند نرم‌افزار را براساس طیفی تقسیم‌بندی می‌کند که در یک انتهای آن هیچ برنامه‌ریزی و فرآیندی وجود ندارد و در انتهای دیگر آن قراردادهای تغییر ناپذیر که در آنها تمام جزییات مشخص شده اند، قرار دارد [۸] .



شکل ۳-۱. طیف میزان برنامه‌ریزی در مدلها

مدیران پروژه باید با توجه به پارامترهایی مانند طبیعت پروژه، اندازهٔ پروژه، نیازهای مشتری بهترین مدل فرآیند را که پاسخگوی نیاز آنها می‌باشد، انتخاب کنند. بنابراین، نمی‌توان گفت که مدل خاصی برای تمام پروژه‌ها و شرایط به شکل موفقیت آمیزی قابل استفاده است. به خاطر اهمیت فراوانی که مدلهای تولید نرم‌افزار در ساخت سیستمهای نرم‌افزاری با کیفیت و مقرون به صرفه دارند، در این بخش، مدلهای زیر مورد بررسی قرار می‌گیرد:

- مدل آبشاری
- مدل ساخت سریع برنامه کاربردی
- مدل افزایشی
- مدل پیچشی
- مدلهای سریع الانتقال[1] ساخت نرم‌افزار
  - Extreme Programming (XP)
  - Rational Unified Process (RUP)

[1] agile

در رابطه با هر مدل، مراحل پیشنهادی در هر مدل، طبیعت مدل، مزایا و معایب مدل بررسی می‌گردد. در مراجع [۲۸]، [۳۱] و [۳۳] بررسی جامعی برروی این مدلهای فرآیند صورت گرفته است.

۳-۱ فرآیند نرم‌افزار

فرآیند، روش انجام کار یا تولید محصول است. با گسترش این مفهوم به نرم‌افزار، *فرآیند نرم‌افزار*[1] را می‌توان روشی برای ساخت یا تولید نرم‌افزار دانست. *فرآیند نرم‌افزار* چارچوبی برای مجموعه‌ای از نواحی کلیدی فرآیند تعریف می‌کند. این نواحی کلیدی فرآیند پایه کنترل مدیریتی پروژه‌های نرم‌افزاری را فراهم و زمینه‌ای را برای به کارگیری روشهای مهندسی نرم‌افزار، تولید محصولات کاری (مدلها، مستندات، داده‌ها، گزارشها، فرمها و غیره)، دستیابی به مقاطع، تضمین کیفیت و مدیریت تغییرات ایجاد می‌کنند. فرآیند نرم‌افزار باید مناسب محصولاتی که ساخته می‌شود و تقاضای بازار نرم‌افزار باشد.

۳-۲ مدلهای فرآیند نرم‌افزار

مدل فرآیند نرم‌افزار یا *مدل* چرخهٔ عمر[2]، استراتژی ساخت نرم‌افزار است که در برگیرندهٔ فرآیند، روشها و ابزارهای مهندسی نرم‌افزار می‌باشد. در هر مدل فرآیند نرم‌افزار، مراحل عمومی فعالیت مهندسی نرم‌افزار یعنی تعریف، ساخت و پشتیبانی وجود دارد. مدل فرآیند ترتیب فعالیتهای پروژه را تعریف می‌کند. بدین ترتیب کل چرخهٔ عمر پروژه تعریف می‌شود.

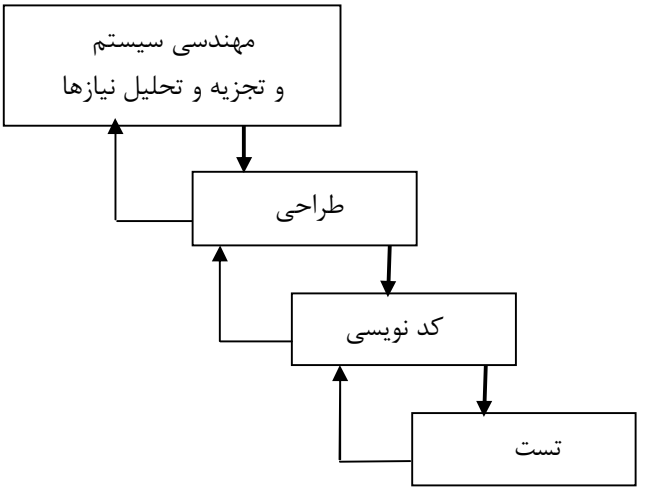
هم اکنون، سازمانها تنوعی از مدلهای فرآیند را برای ساخت سیستمهای نرم‌افزاری به کار می‌گیرند. مدل فرآیند نرم‌افزار براساس طبیعت پروژه و برنامهٔ کاربردی، روشها و ابزارهای مورداستفاده، تیم توسعه، و کنترلهای پروژه، خروجیهای موردنیاز و طبیعت بازار انتخاب می‌گردد. در ادامه، تنوعی از مدلهای متفاوت فرآیند نرم‌افزار مورد بحث قرار می‌گیرند.

۳-۲-۱ مدل آبشاری

مدل آبشاری[3] یا خطی- متوالی اولین بار توسط Winston Royce در سال ۱۹۷۰ معرفی گردید[۱۵] . این مدل، رهیافتی سیستماتیک و متوالی برای ساخت نرم‌افزار است که از تحلیل سیستمی آغاز می‌گردد و با مراحل تجزیه و تحلیل، طراحی و تست پیش می‌رود. این مدل قدیمی‌ترین و

[1] Software process
[2] life cycle model
[3] linear sequential

پراستفاده‌ترین مدل مهندسی نرم‌افزار است که درهنگامی که نیازها در ابتدای پروژه به خوبی شناخته شده هستند، رهیافتی معقول برای ساخت نرم‌افزار است. بسیاری از مدلهای دیگر نیز به نوعی تغییر شکل یافته یا بهبود یافتهٔ این مدل هستند. مدل ارائه شده توسط Royce در شکل ۳-۱ نشان داده شده است.



شکل ۳-۲. مدل آبشاری

همانگونه که در شکل ۳-۲ دیده می‌شود، در این مدل مراحل زیر به شکل متوالی انجام می‌گیرد:

۱- مهندسی سیستم و تجزیه و تحلیل نیازها
۲- طراحی
۳- کدنویسی
۴- تست

مهندسی سیستم و تجزیه و تحلیل نیازها. چون نرم‌افزار همیشه بخشی از سیستم بزرگتری است، کار با به دست آوردن نیازهای تمام عناصر سیستم و سپس تخصیص زیرمجموعه‌ای از این نیازها به نرم‌افزار آغاز می‌گردد. این دیدگاه سیستمی در هنگامی که نرم‌افزار باید با سایر عناصر سیستم مانند

سخت افزار، کاربران و پایگاههای داده‌ای تعامل داشته باشد، مهم است. مهندسی و تجزیه و تحلیل سیستمی دربرگیرندهٔ جمع آوری نیازها در سطح سیستمی و انجام تجزیه و تحلیل و طراحی در سطح سیستم می‌باشد. در بعضی از موارد نیازهایی نیز در سطح استراتژیک و تجاری جمع آوری می‌شود.

سپس فعالیت به دست آوردن نیازهای نرم‌افزاری با هدف درک طبیعت نرم‌افزاری که ساخته می‌شود، انجام می‌گیرد. بدین منظور مهندس نرم‌افزار باید حوزهٔ اطلاعاتی نرم‌افزار و همچنین کارکرد، رفتار و واسطهای نرم‌افزار را شناسایی و مدل کند. نیازهای سیستمی و نرم‌افزار در قالب مشخصات نیازهای سیستمی و مشخصات نیازهای نرم‌افزار مستند می‌گردد. مستندات نیازها بیان می‌کنند که محصول چه کاری باید انجام دهد. این مستندات توسط گروه تضمین کیفیت نرم‌افزار بررسی می‌شود و سپس توسط مشتری بازبینی می‌گردد. بعد از تایید مستندات توسط مشتری، برنامهٔ مدیریت پروژه شامل جدول زمان‌بندی و تخمین هزینه‌های پروژه ایجاد می‌گردد. بعد از تایید توسط گروه تضمین کیفیت نرم‌افزار و مشتری، مرحلهٔ طراحی آغاز می‌گردد.

طراحی. در مرحلهٔ طراحی، چهار فعالیت اصلی طراحی معماری، طراحی واسطهای نرم‌افزار، طراحی پایگاه داده‌ها و طراحی الگوریتمها انجام می‌پذیرد. فرآیند طراحی مشخصات نیازها را به طراحی تبدیل می‌کند. خروجی این مرحله در قالب مستندات طراحی مستند می‌شود. مستندات طراحی مشخص می‌کنند که محصول چگونه کار تعریف شده را انجام می‌دهد.

کد نویسی. در این مرحله، طراحیهای انجام شده به کد برنامه نویسی تبدیل می‌گردند. اگر طراحی مشروح انجام شده باشد، بخشی زیادی از فرآیند تولید کد به شکل خودکار قابل انجام خواهد بود.

تست. بعد از تولید کد، تست نرم‌افزار آغاز می‌گردد. در فرآیند تست منطق داخلی نرم‌افزار و کارکرد خارجی آن تست می‌شود.

هنگامی که سازندگان محصول احساس کنند که محصول با موفقیت کامل شده است، محصول به منظور تست پذیرش نهایی به مشتری ارائه می‌شود. خروجیهای این مرحله، راهنمای کاربر، راهنمای نصب و سایر مستندات ذکر شده در قرارداد می‌باشد. بعد از تایید محصول توسط مشتری، هر تغییر درخواستی در مرحلهٔ پشتیبانی انجام می‌گیرد.

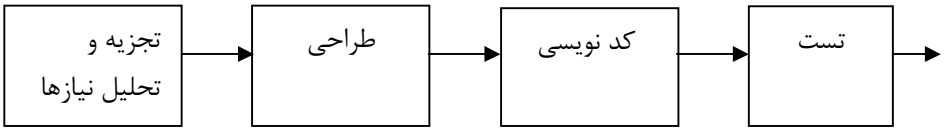
پشتیبانی. بعد از تولید نرم‌افزار و تحویل آن به مشتری، تغییراتی در نرم‌افزار پیش خواهد آمد. این تغییرات به خاطر خطاهای پیش آمده، تطبیق نرم‌افزار با محیط خارجی (تغییرات موردنیاز برای تطبیق با سیستم عامل یا ابزار جانبی عرضه شده به بازار)، درخواست کارکردهای جدید یا کارآیی بهتر از جانب مشتری می‌باشد. در پشتیبانی یا نگهداری نرم‌افزار هرکدام از مراحل ذکر شده در قبل، برای نرم‌افزار موجود دوباره تکرار می‌شوند.

در هر مرحله محصولاتی (شامل مستندات، مدلها یا کد) تولید می‌شود، درستی و کامل بودن این محصولات در انتهای مرحله توسط گروه تضمین کیفیت نرم‌افزار تایید می‌گردد و سپس به عنوان ورودی مرحلهٔ بعدی استفاده می‌شود. این محصولات کامل درنظر گرفته می‌شوند و تقریبا تغییر یا بازبینی در آنها صورت نمی‌گیرد مگر این که مشکل عمده‌ای وجود داشته باشد. در این مدل، بازخوردی نیز از یک مرحله به مرحلهٔ دیگر درنظر گرفته شده است. بعضی اوقات، در مرحلهٔ طراحی مشخص می‌گردد که مشکلی در مستندات مشخصات نیازها وجود دارد. مشخصات نیازها ممکن است که ناقص باشد (بعضی از ویژگیهای نرم‌افزار درنظر گرفته نشده باشند)، تضاد در آن وجود داشته باشد (دو یا چند نیاز در مشخصات بیان شده با یکدیگر در تضاد باشند) یا ابهام داشته باشد (بعضی از نیازها بیش از یک تفسیر داشته باشند). در صورت وجود چنین مشکلاتی در مشخصات نیازها، باید پیش از ادامهٔ فرآیند نرم‌افزار، آن را بازبینی نمود. با توجه به شکل ۱، اگر سازندگان سیستم مجبور شوند که مشخصات نیازها را در مرحلهٔ طراحی بازبینی کنند، فرآیند ساخت نرم‌افزار حلقهٔ بازخوردی نشان داده شده را دنبال خواهد کرد. با تایید مشتری، تغییرات لازم در مستندات داده می‌شود و برنامه‌ریزی و فعالیتهای طراحی به منظور درنظر گرفتن این تغییرات، انجام می‌شوند.

در مرحلهٔ پیاده‌سازی نیز ممکن است مشکلاتی در طراحی دیده شود. به عنوان مثال، ممکن است در رابطه با یک سیستم بلادرنگ در حال ساخت، اثبات شود که در صورت پیاده‌سازی طراحی انجام شده، سیستم نهایی بسیار کند خواهد بود. مشکلات طراحی باید پیش از ساخت نرم‌افزار اصلاح گردند. بازخورد درنظر گرفته شده از مرحلهٔ پیاده‌سازی به مرحلهٔ طراحی امکان انجام تغییرات لازم در مستندات طراحی و حتی مستندات نیازها را فراهم می‌آورد.

بسیاری از سازمانها در عمل، وجود این حلقهٔ بازخورد را نادیده گرفته اند و از آن به شکل خطی و متوالی استفاده کرده اند. این شیوهٔ به کارگیری مدل که در شکل ۲ نشان داده شده است، با ایدهٔ اولیهٔ خود Royce تفاوت دارد. او که مدافع ساخت تکراری و افزایشی نرم‌افزار بوده است[۲۴]، توصیه کرده بود که این مدل دوبار باید انجام شود [۱۵] :

"اگر برنامهٔ کامپیوتری موردنظر برای اولین بار تولید می‌شود، فعالیتها را طوری ترتیب دهید که نگارش عملیاتی که در نهایت به مشتری تحویل داده می‌شود، عملاً دومین نگارش باشد."



شکل ۳-۳. به کارگیری مدل آبشاری بدون حلقه‌های بازخوردی

در مدل آبشاری تا زمانی که مستندات هر مرحله کامل نشود و محصولات آن مرحله توسط گروه تضمین کیفیت نرم‌افزار به تایید نرسد، آن مرحله کامل نمی‌گردد. اگر درنتیجهٔ دنبال نمودن حلقهٔ بازخوردی، نیاز به انجام تغییراتی در محصولات مراحل قبلی باشد، تنها در صورتی که در مستندات آن مرحله تغییرات لازم انجام گیرد و تغییرات توسط گروه تضمین کیفیت نرم‌افزار تایید شوند، آن مرحله کامل محسوب می‌گردد.

مدل آبشاری موفقیتهای زیادی در تولید سیستمهای نرم‌افزاری داشته است. البته، در مواردی نیز استفاده از آن منجر به شکست پروژه‌ها شده است. به منظور تصمیم‌گیری در رابطه با استفاده از هر مدلی در ساخت نرم‌افزار، لازم است که نقاط قوت و ضعف آن شناسایی گردد به همین منظور در ادامه، تحلیلی در رابطه با نقاط قوت و ضعف این مدل ارائه می‌گردد. نقاط قوت این مدل را می‌توان به شرح زیر دانست:

- نیاز به ارائهٔ مستندات در انتهای هر مرحله و بررسی درستی و کامل بودن تمام محصولات هر مرحله توسط گروه تضمین کیفیت نرم‌افزار، رهیافت پیشنهادی توسط این مدل را منظم و با قاعده می‌سازد. یکی از اصول مهم برای خاتمهٔ هر مقطع تایید محصولات آن مرحله توسط گروه تضمین کیفیت نرم‌افزار است.

- مستندات مشخصات نیازها، مستندات طراحی، مستندات پیاده‌سازی، و سایر مستنداتی مانند راهنمای پایگاه داده‌ها، راهنمای کاربر، راهنمای نصب و راهنمای عملیات نرم‌افزار، ابزارهای اساسی برای نگهداری محصول هستند. حدود ۶۷ درصد بودجهٔ نرم‌افزار اختصاص به پشتیبانی دارد و پیروی از مدل آبشاری و مستندات حاصل از این فرآیند، پشتیبانی سیستم را آسان‌تر می‌سازد. بسیاری از موفقیتهای مدل آبشاری به خاطر مبتنی بودن آن بر مستندات[1] می‌باشد.

نقاط ضعف این مدل نیز عبارتند از:

- پروژه‌های واقعی به ندرت از جریان متوالی پیشنهادی در مدل پیروی می‌کنند. البته مدل اولیه پیشنهادی توسط Royce دارای تکرار و حلقهٔ بازخوردی نیز می‌تواند باشد، اما این بازخوردها مستقیم و صریح نمی‌باشند. در نتیجهٔ ایجاد نقاط کنترلی برای پروژه مشکل می‌شود و تغییرات در روند انجام پروژه می‌تواند باعث به وجود آمدن سردرگمی در تیم پروژه گردد.

[1] Document-driven

- در ابتدای اغلب پروژه‌ها، عدم اطمینانی در رابطه با سیستمی که می‌خواهد ساخته شود، وجود دارد و به خاطر طبیعت سیستمی که در حال ساخته شدن است برای مشتری مشکل است که تمام نیازها را در ابتدای پروژه و به شکل صریح بیان نماید. اما در مدل آبشاری تا وقتی که تمام نیازها به طور دقیق و کامل در مرحله تحلیل نیازها مشخص نشود، نمی‌توان مرحله بعدی را آغاز کرد. بنابراین، مدل آبشاری در رابطه با پروژه‌هایی که در آنها نمی‌توان تمام نیازها را به طور کامل در ابتدا بیان نمود، دچار مشکل می‌شود.

- در مدل آبشاری نسخه کاری سیستمها تا اواخر پروژه قابل دسترسی نمی‌باشد. اگر مشکل عمده‌ای تا زمانی که سیستم عملیاتی بازبینی می‌شود، کشف نشود، هزینه زیادی به پروژه تحمیل می‌گردد.

- در مدل آبشاری نسخهٔ کاری سیستمها تا اواخر پروژه قابل دسترسی نمی‌باشد و در نتیجه کاربر نمی‌تواند تا اواخر پروژه نسخه‌ای از سیستم را در اختیار داشته باشد. درنتیجه میزان درگیری مشتری در پروژه و گرفتن نظریات مشتری کاهش می‌یابد.

- طبیعت خطی- متوالی مدل آبشاری باعث به وجود آمدن وضعیتهای توقف می‌شود که در آن بعضی از اعضای تیم پروژه باید منتظر سایر اعضای تیم باشند تا وظائف وابسته به وظائف آنها را به اتمام برسانند. این وضعیتهای توقف در ابتدا و انتهای مدل آبشاری بیشتر مشاهده می‌شود.

نقاط ضعف و قوت بیان شده برای مدل آبشاری، به انتخاب این مدل برای پروژه کمک می‌کنند. مدل آبشاری برای سالها در پروژه‌های مختلف به کار گرفته شده است و جایگاه مهمی در مهندسی نرم‌افزار دارد. این مدل، قالبی را فراهم می‌آورد که روشهای تجزیه و تحلیل، طراحی، برنامه نویسی و پشتیبانی در آن جای می‌گیرند و به کارگرفته می‌شوند. در طی سالها تجربه ساخت سیستمهای نرم‌افزاری اشکال جدید و تغییر یافته‌ای از این مدل در تولید سیستمهای کامپیوتری به کارگرفته شده است.

در مواردی می‌توان مدلهای مدرن ساخت نرم‌افزار مانند RUP را در چارچوب مدل آبشاری به کار گرفت[۲۳] . دلائل بسیاری برای این کار وجود دارد:

- تطابق رهیافت ساخت نرم‌افزار با رهیافت ساخت سیستمی بزرگتر
- رعایت استانداردهای موردنیاز به خصوص در شرایط مناقصه و قراردادی که در آنها اتمام مقاطع میانی پروژه مستلزم تحویل محصولات کار به شکلی متوالی می‌باشد.

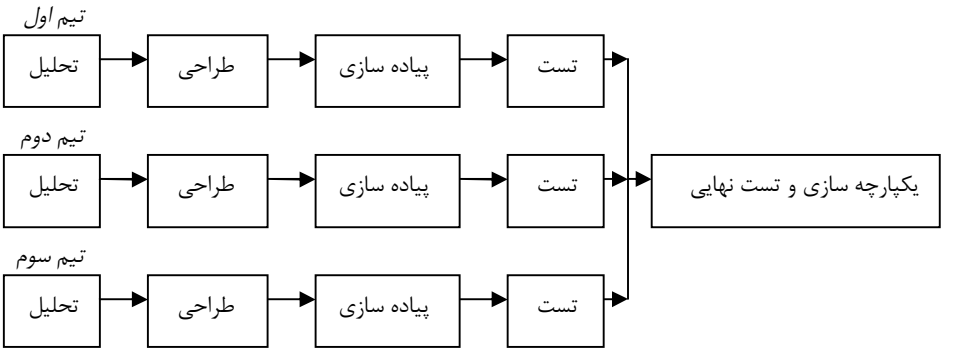
- در انجام چرخه‌های عمر ساده مانند پشتیبانی که در آنها نیازی به تکرار نمی‌باشد.
- در هنگام معرفی مدلهای جدید در سازمانهایی که در آنها مدل آبشاری ساخت نرم‌افزار وجود دارد به منظور پرهیز از انجام تغییرات زیاد به یکباره، ابتدا بعضی از جنبه‌های مدل جدید در چارچوب مدل آبشاری معرفی می‌شود و سپس گذر تدریجی به مدل جدید صورت می‌گیرد.

به منظور برطرف نمودن مشکلات مشاهده شده در مدل خطی در سالهای بعد گونه‌های دیگری از این مدل ارائه گردیدند که در ادامه به آنها اشاره می‌گردد.

۳-۲-۳ مدل ساخت سریع برنامه کاربردی

مدل *ساخت سریع برنامه کاربردی* (Rapid Application Development) یا RAD در سال ۱۹۹۱توسط James Martin ارائه گردید[۲۵]. Martinکیفیت بالای سیستمها، ساخت و تحویل سریع و هزینه‌های پایین را به عنوان اهداف کلیدی RAD تعریف می‌کند.

مدل RAD فرآیندی افزایشی برای ساخت نرم‌افزار است که سعی دارد چرخه عمر ساخت نرم‌افزار را کوتاه نماید. مدل RAD به کارگیری مدل آبشاری با سرعت بالا می‌باشد که در آن ساخت سریع نرم‌افزار با به کارگیری اجزای نرم‌افزاری[1] و ابزارهای مهندسی نرم‌افزار به کمک کامپیوتر (CASE) انجام می‌گیرد. اگر نیازها به خوبی شناخته شده باشند و محدوده پروژه کوچک باشد، فرآیند RAD تیم ساخت نرم‌افزار را قادر می‌سازد تا در دوره‌های زمانی خیلی کوتاه (مثلا، ۶۰ تا ۹۰ روز) سیستمی کاملا عملیاتی تولید کند. گونه‌ای از این مدل در شکل ۳-۴ دیده می‌شود.



شکل ۳-۴ گونه‌ای از مدل RAD

[1] software components

RAD به شکل بنیادین تکنیکهای ساخت جدیدی ارائه نمی‌کند، بلکه سعی دارد نشان دهد که چگونه روشها و ابزارهای موجود و استاندارد را می‌توان در چارچوب ساخت سریع سیستمها به کار گرفت. در RAD این اجزاء وجود دارد:

- طراحی مشترک برنامه کاربردی[1]. RAD معمولا توسط تیمهای ساخت کوچک ۴ تا ۸ نفره متشکل از سازندگان و کاربرانی که باید تصمیمهای طراحی بگیرند، انجام می‌شود. کاربران باید دانش لازم از حوزه کاربردی را داشته باشند و سازندگان نیز باید در استفاده از ابزارهای ساخت نرم‌افزار مهارت کافی داشته باشند. در اکثر رهیافتهای RAD از کارگاههای JAD در نقاط مختلف فرآیند ساخت که در محیطهای خارج از محل کار کاربران و سازندگان تشکیل می‌شود، استفاده می‌گردد.
- سرعت ساخت. پروژه‌های RAD معمولا در بازه‌های زمانی دو تا شش ماه انجام می‌شوند. پیشنهاد می‌شود که بیشتر از شش نفر-سال نیرو به پروژه معمول RAD نباید اختصاص داده شود. یکی از دلائل این سرعت استفاده از ابزارهای ساخت سریع و اجزاء نرم‌افزاری می‌باشد که باعث کاهش مدت زمان فعالیتهای پیاده‌سازی و تست واحدهای سیستم می‌گردد.
- محدودسازی زمانی. کنترل پروژه در RAD ساخت سیستم را اولویت‌بندی و برای پروژه مقاطع غیرقابل تغییر تعریف می‌کند. اگر پروژه بخواهد طولانی‌تر شود و به این مقاطع دست نیابد، در پروژه‌های RAD نیازها کاهش داده می‌شوند تا مقاطع پروژه رعایت گردد و به مدت زمان آنها افزوده نگردد.
- نمونه سازی. در RAD در مقاطع مختلف ساخت سیستم از نمونه سازی استفاده می‌شود. چرخه نمونه سازی به خصوص در مرحله به دست آوردن نیازها حداقل سه بار تکرار می‌شود.
- ابزارهای ساخت سریع. فرآیند ساخت در RAD توسط ترکیبی از زبانهای نسل چهارم، سازنده‌های واسط گرافیکی کاربر، سیستمهای مدیریت پایگاه داده‌ها و ابزارهای CASE پشتیبانی می‌شود.

همانند سایر مدلهای فرآیند، RAD برای تمام انواع پروژه‌ها مناسب نمی‌باشد. در انتخاب این مدل باید به فاکتورهای زیر توجه داشت:

- نوع سیستم. اگر بتوان برنامه کاربردی را طوری به پیمانه‌های مستقلی تقسیم نمود که این پیمانه‌ها در بازه زمانی سه ماه کامل شوند، این برنامه کاربردی توسط RAD قابل پیاده‌سازی

[1] Joint Application Design

می‌باشد. همچنین سیستم نباید از نظر محاسباتی پیچیده باشد و نیاز به کارآیی بالا وجود داشته باشد.

- اندازه پروژه. RAD برای پروژه‌هایی با مقیاس کوچک تا متوسط استفاده می‌شود. اگر پروژه‌های بزرگ را بتوان به بخشهای کوچکتری تقسیم نمود که هرکدام به شکل مستقل قابل تحویل باشند، می‌توان RAD را برای این پروژه‌ها نیز به کاربرد.
- اندازه تیم. تیم ساخت باید به منظور کاهش سربارهای مدیریتی و ارتباطی و افزایش تعهد به ساخت سیستم کوچک نگه داشته شود.
- درگیری مدیریت و کاربران نهایی. در پروژه RAD، مدیریت ارشد باید متعهد شود که کاربر نهایی را در پروژه درگیر نماید. سازندگان سیستم باید به آسانی به کاربران نهایی دسترسی داشته باشند. به شکل ایده آل کاربرنهایی و سازنده سیستم باید با هم کار کنند.
- اختیارات لازم. تیم RAD باید دارای اختیارات لازم برای انجام تصمیم‌گیریهای روزانه طراحی بدون نیاز به تایید و مشورت مدیریت سطح بالاتر باشد.

RAD دارای فواید زیر می‌باشد:

- ساخت و تحویل سریع تر. با استفاده از این مدل، نسخه عملیاتی سیستم سریع‌تر ساخته و تحویل داده می‌شود. مقیاس زمانی ساخت سیستم در محدوده سه تا شش ماه می‌باشد.
- تمرکز برروی نیازهای تجاری. ایده اصلی در RAD این است که ۸۰٪ سیستم را می‌توان در ۲۰٪ زمان تحویل داد. در RAD جنبه‌های فنی یا مهندسی دارای اهمیت می‌باشد، اما بعد از جنبه‌های تجاری مورد توجه می‌باشد.
- هزینه‌های پایین تر. با توجه به این که RAD سیستمها را بسیار سریع‌تر و با تمرکز برروی نیازهای تجاری تولید می‌کند، هزینه‌های تولید پایین می‌یابد.
- تعهد بیشتر ذینفعان سیستم. در پروژه‌های RAD کاربر احساس مالکیت بیشتری نسبت به سیستم پیدا می‌کنند. سازندگان سیستم نیز از ساخت سیستمهای موفق در بازه‌های زمانی کوتاه رضایت بیشتری پیدا می‌کنند.

مانند تمام مدلهای فرآیند، رهیافت RAD دارای معایبی نیز می‌باشد:

- در پروژه‌های بزرگ و مقیاس پذیر، RAD نیاز به منابع انسانی کافی برای ایجاد تعداد تیمهای لازم RAD دارد.
- به منظور انجام پروژه براساس RAD، مشتری و سازندگان سیستم باید خود را متعهد به انجام فعالیتهای سریع موردنیاز برای تحویل سیستم در بازه زمانی کوتاه‌تر بدانند. اگر این تعهد وجود نداشته باشد، پروژه شکست خواهد خورد.
- RAD در هنگامی که ریسکهای فنی بالا هستند، مناسب نمی‌باشد. ریسک فنی در هنگامی وجود دارد که برنامه کاربردی جدید از به میزان زیادی از تکنولوژی جدید استفاده می‌کند یا هنگامی که نرم‌افزار جدید باید به میزان زیادی با سیستمهای موجود کامپیوتری کار کند.

۳-۲-۴ مدلهای تکرارشونده و افزایشی

مدلهای تکراری و افزایشی تاکید بر تحویل و تکمیل تدریجی نرم‌افزار دارند. سوال اینجا است که چرا مدلهای تکراری و افزایشی مطرح می‌باشند. در اوائل دهۀ ۹۰ با گسترده شدن به کارگیری سیستمهای نرم‌افزاری در بازار و رشد شرکتها و سازمانهای تولید کنندۀ نرم‌افزار، رقابت نیز افزایش یافت. در بازار رقابتی عرضۀ به موقع و سریع محصول با کیفیت به بازار، یکی از عوامل موثر در بقای تولیدکنندۀ محصول است. به همین خاطر، تحویل محصول به بازار تجاری در انتهای یک چرخۀ یک یا چندساله دیگر ممکن نبود و نیاز به مدلهای فرآیندی بود که امکان ارائۀ محصولاتی حتی با امکانات کمتر اما در بازه‌های زمانی کوتاهتر را به بازار ممکن سازند.

یکی دیگر از فاکتورهای رقابتی مهم، کیفیت محصول است. تطابق با نیازهای مشتری از جمله عوامل موثر در کیفیت محصول است. با ارائۀ افزایشی محصول عملیاتی به مشتری و گرفتن مستمر بازخورد از وی می‌توان احتمال تطابق محصول ارائه شده با نیازهای مشتری را افزایش و هزینه‌های انجام تغییرات درخواستی از سوی مشتری را کاهش داد.

با آمدن تکنولوژیهای جدید و اغلب ناشناخته، ریسکهای فنی پروژه‌ها نیز افزایش می‌یابد. عدم آشنایی با جوانب مختلف یک تکنولوژی و تواناییهای آن در حل مسئلۀ موردنظر یکی از ریسکهای فنی پروژه می‌باشد که ممکن است منجر به شکست پروژه‌ها گردد.

در رابطه با بسیاری از مسائل نیز نمی‌توان نیازها را از ابتدا به درستی مشخص نمود و تا زمانی که محصول ساخته نشود و عملیاتی نگردد، سایر نیازهای مرتبط با محصول به طور دقیق و روشن مشخص نخواهد شد.

به منظور حل این مشکلات، ایدهٔ ساخت تکاملی، تکرارشونده و افزایشی سیستمهای نرم‌افزاری مطرح گردید. البته، به کارگیری واژهٔ "تکاملی" و "تکامل" در فرهنگ فرآیند نرم‌افزار اولین بار توسط Tom Gilb در سال ۱۹۷۶ در کتاب Software Metrics انجام پذیرفت[۱۳] . وی در کتاب خود می‌نویسد:

> "سیستم پیچیده هنگامی به بهترین شکل تولید می‌گردد که در مراحل کوچک پیاده‌سازی شود و در هر مرحله میزان موفقیت اندازه گیری گردد و امکان اصلاح خطاها وجود داشته باشد. بدین ترتیب، پیش از اختصاص همه منابع به سیستم موردنظر فرصت دریافت بازخورد از دنیای واقعی و اصلاح خطاهای طراحی به وجود می‌آید..."

ایده‌های اولیه و تکمیلی در رابطه با توسعهٔ تکاملی نرم‌افزار را می‌توانید در [۱۴] مطالعه کنید. مدلهای تکادر مدلهای تکرارشونده و افرایشی ایدهٔ اصلی، ساخت افزایشی سیستم نرم‌افزاری است. در چنین مدلهایی سازندهٔ نرم‌افزار این فرصت را می‌یابد تا از آنچه در طی ساخت نسخه‌های اولیه و افزایشی سیستم آموخته است، درس بگیرد. در این مدلها ساخت سیستم با زیرمجموعه‌ای از نیازهای سیستم که معمولا نیازهای کلیدی هستند و برای مشتری ارزش بالاتری دارند، آغاز می‌گردد و سیستم به شکل تکراری در نسخه‌های متوالی کامل می‌گردد تا درنهایت سیستم کامل پیاده‌سازی شود. در هر تکرار، براساس بازخوردهای حاصل، تغییرات لازم در نسخهٔ قبلی داده می‌شود و کارکردهای جدید نیز اضافه می‌گردند. در طی سالهای اخیر مدلهای زیادی براساس فلسفهٔ ساخت تکرارشونده و تکراری ارائه شده اند که در ادامه چند نمونه از این مدلها بررسی می‌شوند. Larman و Basili در [۲۴] تحلیلی تاریخی و محتوایی از توسعهٔ تکرارشونده و افزایشی ارائه می‌کنند.

۳-۲-۴-۱ مدل افزایشی

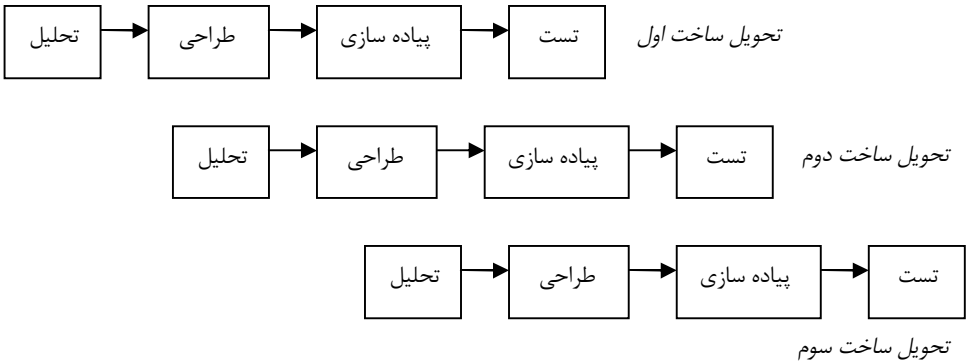
مدل افزایشی[1] یکی از مدلهای تکاملی نرم‌افزار است که امکان می‌دهد تا نرم‌افزار به تدریج کامل گردد. این مدل عناصر مدل خطی-متوالی را با فلسفهٔ نمونه سازی ترکیب می‌کند[۲۸] . نمونه‌ای از این مدل در شکل دیده می‌شود. همانگونه که در شکل دیده می‌شود، در این مدل، ابتدا مهندسی سیستم و برنامه‌ریزی برای ساختهای[2] مختلف نرم‌افزار انجام می‌پذیرد. هر ساخت قابلیت کارکردی به خصوصی را فراهم می‌کند. به عنوان مثال، اگر محصول موردنظر کنترل کنندهٔ زیردریایی هسته‌ای باشد، سیستم راهبری را می‌توان به عنوان یک ساخت درنظر گرفت. در یک سیستم عامل، زمان‌بند وظائف را می‌توان

[1] incremental model
[2] builds

یک ساخت درنظر گرفت. در هر تکرار مدل خطی-متوالی، نسخه‌ای اجرایی از نرم‌افزار تولید و در ساختار کلی نرم‌افزار یکپارچه و تست می‌گردد. فرآیند هنگامی متوقف می‌شود که محصول به کارکرد نهایی و مطلوب خودش دست پیدا کند. به عنوان مثال، به منظور تولید یک واژه‌پرداز براساس مدل افزایشی می‌توان آن را به چهار ساخت زیر تقسیم نمود:

ساخت اول: نیازهای پایه و قابلیتهای اولیه مدیریت فایل، ویرایش، تولید مستندات
ساخت دوم: قابلیتهای پیچیده‌تر ویرایش و تولید مستندات
ساخت سوم: غلط یاب املایی و کنترل گرامر
ساخت چهارم: قابلیتهای پیشرفتهٔ صفحه‌آرایی

سازندهٔ سیستم می‌تواند، محصول نهایی را به هر تعداد ساختی که فکر می‌کند، تقسیم نماید. اما، این محدودیت نیز وجود دارد که بعد از یکپارچه سازی هر ساخت در نرم‌افزار موجود، محصول حاصل باید قابل تست باشد. اگر محصول به ساختهای اندکی تقسیم شود، مدل تبدیل به روش ساخت و اصلاح می‌گردد. اگر تعداد ساختها نیز زیاد باشد، در هر مرحله، زمان زیادی صرف تست یکپارچهٔ کارکرد اندکی که افزوده شده است، می‌گردد. تقسیم بهینهٔ محصول به ساختها وابسته به نوع محصول و میزان تجربهٔ سازندهٔ سیستم می‌باشد. در شکل ۳-۵ نمونه‌ای از این مدل دیده می‌شود. براساس این مدل، بعد از تجزیه و تحلیل نیازهای سیستم، تیم نیازها برروی تجزیه و تحلیل ساخت دوم کار خود را آغاز می‌کند و تیم طراحی به طراحی اولین ساخت می‌پردازد. بنابراین، ساختهای مختلف به شکل موازی ساخته می‌شوند و هر تیم از اطلاعات به دست آمده در تمام ساختهای قبلی استفاده می‌نماید. در این گونه از مدل افزایشی این ریسک وجود دارد که ساختهای حاصل در یکدیگر یکپارچه نشوند. در گونهٔ دیگری از مدل افزایشی ابتدا تجزیه و تحلیل و طراحی معماری سیستم انجام می‌پذیرد و برای هر ساخت فرآیند طراحی مشروح، پیاده‌سازی، یکپارچه سازی، تست و تحویل به مشتری انجام می‌گیرد.
در مدل افزایشی، اولین ساخت محصول پایه است. در این محصول نیازهای اولیه برآورده می‌شوند، اما ویژگیهای مکمل تحویل داده نمی‌شوند. محصول پایه توسط مشتری استفاده می‌شود یا مورد ارزیابی مشروح قرار می‌گیرد.



شکل ۳-۵ مدل افزایشی

این مدل دارای فواید و کاربردهای زیر است:

- مدل افزایشی در هنگامی که نیروی انسانی کافی برای پیاده‌سازی کامل سیستم در زمان تعیین شده وجود ندارد، مفید است. می‌توان ساختهای اولیه را به تعداد نفرات کمتری انجام داد. اگر محصول پایه موردتوجه قرار گرفت، در صورت نیاز می‌توان نیروی انسانی بیشتری برای پیاده‌سازی ساخت بعدی اضافه نمود.
- ساختها را طوری می‌توان تنظیم نمود که ریسکهای فنی مدیریت شود. به عنوان مثال، اگر سیستمی نیاز به سخت افزار جدیدی داشته باشد و این سخت افزار در حال ساخت باشد و زمان تحویل آن نیز معین نباشد، می‌توان ساختهای اولیه را طوری برنامه‌ریزی نمود که از این سخت افزار استفاده نکند ولی بخشی از کارکرد به مشتری ارائه شود.
- در هنگامی که موعد تحویل را نمی‌توان تغییر داد از این مدل استفاده می‌شود.
- در این مدل، گرفتن نظرات وارزیابی مشتری زودتر انجام می‌گیرد.
- معرفی تدریجی محصول به مشتری، زمان لازم برای تطبیق محصول به وجود می‌آورد.

مشکلی که در رابطه با این مدل وجود این است که هر ساخت باید بدون ایجاد مشکل، به شکلی در ساختار موجود جای بگیرد. ساختار موجود باید این امکان گسترش را داشته باشد و افزودن ساخت بعدی به ساختار موجود باید آسان و راحت باشد. به منظور فراهم آوردن این امکان، طراحی انجام شده

باید انعطاف پذیر باشد و طراحی انجام شده برای کل سیستم باید با توجه به ساخت افزایشی سیستم صورت گیرد. بدین منظور باید معماری سیستم را به شکل باز طراحی شود.

۳-۲-۴-۲ مدل پیچشی

این مدل در ابتدا توسط Barry Boehm در سال ۱۹۸۸ ارائه گردید[۶] . با به کارگیری این مدل، امکان ساخت سریع نسخه‌های افزایشی از نرم‌افزار به وجود می‌آید. در طی نسخه‌های اولیه ممکن است که نمونه‌ای از نرم‌افزار ایجاد شود و در مراحل بعدی نسخه‌هایی عملیاتی از نرم‌افزار ارائه گردد.
پیشنهاد اولیهٔ Boehm در شکل ۳-۶ دیده می‌شود. مشخصهٔ اصلی این مدل ارزیابی ریسک در مراحل مشخصی از پروژه و انجام فعالیتهایی برای رفع این ریسکها می‌باشد. قبل از هر چرخه، فعالیت تحلیل ریسک انجام می‌شود و در انتهای هر چرخه رویه‌ای برای بازبینی وجود دارد که در رابطه با رفتن به چرخهٔ بعد تصمیم گیری می‌نماید. ریسک در این مدل عدم اطمینان در رابطه با وقوع رویدادی می‌باشد. به عنوان مثال، اگر هدف استفاده از زبان برنامه نویسی جدیدی باشد، ریسک موجود نبود کامپایلر مناسب آن در بازار است.

شکل ۳-۶ مدل پیچشی

همانگونه که در شکل دیده می‌شود، چرخهٔ مارپیچ با به دست آوردن اهدافی مانند کارآیی، کارکرد و غیره آغاز می‌گردد. راههای مختلف دستیابی به این اهداف و محدودیتهای موجود در رابطه با هر راه بررسی می‌گردد. هر راه در رابطه با اهداف ارزیابی می‌گردد. این کار معمولا منجر به شناسایی منابع ریسک در پروژه می‌گردد. مرحلهٔ بعدی ارزیابی ریسکهای شناسایی شده با انجام فعالیتهایی مانند تحلیل مشروح، نمونه سازی، شبیه سازی و غیره می‌باشد.

بعد از ارزیابی ریسک، مدل ساخت سیستم انتخاب می‌گردد. به عنوان مثال، اگر ریسکها واسط کاربر حاکم باشند، مدل مناسب استفاده از نمونه سازی است. اگر ریسکهای ایمنی مسئلهٔ اصلی باشند، استفاده از مدلهای صوری مناسب خواهد بود. اگر مهمترین ریسک شناسایی شده یکپارچه سازی زیرسیستمها باشد، مدل آبشاری مناسب‌ترین مدل ساخت خواهد بود.

نیازی به به کارگیری یک مدل در هر چرخهٔ مارپیچ نیست. مدل پیچشی می‌تواند مدلهای فرآیند دیگر را نیز در خود جای دهد. ممکن است نمونه سازی در یک چرخه برای برطرف نمودن ریسکهای مرتبط با نیازها استفاده شود و در چرخهٔ بعدی مدل آبشاری استفاده شود. روشهای صوری نیز می‌توانند برای بخشهایی از سیستم که نیازهای ایمنی و امنیتی بالا دارند، استفاده شود. Boehm برای استفاده از مدل پیچشی فرمهای استانداردی را پیشنهاد می‌کند که به طور تکاملی در هر دور از مارپیچ مدل پیچشی پر می‌شوند. در شکل ۳-۷ نمونه‌ای از فرم پرشده برای ساخت سیستم کاتالوگ اجزاء نرم‌افزاری دیده می‌شود.

نوع دیگری از این مدل به نام مدل پیچشی WINWIN در سال ۱۹۹۸ توسط Boehm ارائه شده است که مجموعه‌ای از فعالیتهای مذاکره را در شروع هر چرخه از مدل پیچشی معرفی می‌کند[۷] . این فعالیتهای عبارتند از:

۱- شناسایی ذی نفعان کلیدی سیستم یا زیرسیستم
۲- تعیین شرایط برد ذی نفعان شناسایی شده
۳- مذاکره با ذی نفعان به منظور دستیابی به مجموعه‌ای از شرایط برد-برد که در آن هر دو گروه سازندهٔ سیستم و ذی نفعان سیستم با یکدیگر به توافقی قابل قبول برای همه دست پیدا کنند.

انجام موفقیت آمیز این مراحل منجر به دستیابی به نتایج برد-برد می‌گردد که معیاری حیاتی برای ادامهٔ ساخت نرم‌افزار و انجام فعالیت تعریف سیستم می‌باشد.

اهداف	ایجاد سیستم مدیریت کاتالوگ اجزاء نرم‌افزاری
محدودیتها	یک سال زمان موجود است با اجزای موجود را پشنیبانی کند هزینه نباید بیشتر از ۱۰۰۰۰۰ دلار شود
راههای ممکن	خرید نرم‌افزار بازیابی اطلاعات خرید سیستم مدیریت پایگاه داده‌ها و ایجاد کاتالوگ با استفاده از زبان پرس و جو ایجاد سیستم خاص منظوره از پایه
ریسکها	تولید سیستم با توجه به محدودیتهای تعیین شده ممکن نباشد. کارکرد کاتالوگ نامناسب باشد
حل ریسک	ایجاد نمونه برای روشن شدن نیازها انعقاد قرارداد مشاوره برای بررسی سیستمهای بازیابی اطلاعات موجود و پیشنهاد سیستم مناسب افزایش زمان ساخت نرم‌افزار
نتایج	سیستمهای بازیابی اطلاعات انعطاف پذیر نمی‌باشند و نیازهای شناسایی شده برآورده نمی‌شوند ایجاد نمونه با استفاده از سیستم مدیریت پایگاه داده‌های موجود و بهبود آن برای دستیابی به سیستم کامل ساخت سیستم مدیریت کاتالوگ خاص منظوره مناسب نمی‌باشد.
برنامه‌ها	ایجاد کاتالوگ با استفاده از سیستم مدیریت پایگاه داده‌های موجود از طریق نمونه سازی تکاملی
تصمیمها	اختصاص بودجه برای یکسال

شکل ۳-۷ کاتالوگ پیشنهادی برای مدل پیچشی

مدل پیچشی برای ساخت سیستمهای بزرگ با ریسکهای فنی بالا مانند پروژه‌های نظامی، فضایی و سایر پروژه‌های پیچیده رهیافتی واقعی به نظر می‌آید. زیرا نرم‌افزار در طی پیشرفت فرآیند کامل می‌گردد، سازنده و مشتری ریسکها را بهتر شناسایی می‌کنند و در هر سطح از تکامل نرم‌افزار به آنها واکنش نشان می‌دهند. مدل پیچشی ریسکهای فنی را در تمام مراحل پروژه درنظر می‌گیرد و سعی دارد پیش از ایجاد مشکل، آنها را حل کند.

اما، در وضعیتهایی مانند نرم‌افزارهای قراردادی متقاعدکردن مشتری نسبت به قابل کنترل بودن پروژه مشکل است. این مدل نیاز به خبرگی زیادی در رابطه با مدیریت ریسک دارد و موفقیت این مدل وابسته به این خبرگی است. تجزیه و تحلیل ریسک دارای هزینه است و در پروژه‌های کوچک که ریسکهای فنی کمی دارند، این مدل مناسب به نظر نمی‌رسد.

۳-۲-۴-۳ توسعهٔ سریع الانتقال نرم‌افزار

در اوائل سال ۲۰۰۱، گروهی از خبرگان صنعت نرم‌افزار در نشستی ارزشها و اصولی را معرفی کردند که به سازندگان نرم‌افزار اجازه می‌دهد که با سرعت بیشتری سیستمهای نرم‌افزاری را تولید کنند و به تغییرات پاسخ دهند. آنها نام "[1]اتحاد سریع الانتقال" را برخود نهادند[۴]. در طی این فعالیت، بیانیهٔ "اتحاد سریع الانتقال" به شرح زیر صادر گردید:

ما راههای بهتری را برای ساخت نرم‌افزار پیشنهاد می‌کنیم.
در طی تجارب خود به ارزشهای زیر دست یافته ایم:
- ارزش افراد و تعاملات نسبت به فرآیندها و ابزارها
- ارزش نرم‌افزار عملیاتی نسبت به مستندات کامل
- همکاری مشتری به توافق در رابطه با قرارداد
- پاسخ به تغییرات نسبت به دنبال نمودن طرح و برنامه
یعنی، در حالی که موارد سمت چپ دارای ارزش هستند،
ما به موارد سمت راست ارزش می‌دهیم.

ارزش افراد و تعاملات نسبت به فرآیندها و ابزارها

افراد مهمترین عنصر موفقیت هستند. اگر تیم دارای افرادی قوی نباشد، حتی فرآیندی خوب هم نمی‌تواند پروژه را از شکست نجات دهد. اما فرآیند بد می‌تواند قوی‌ترین افراد را بدون تاثیر سازد. حتی

[1] Agile Alliance

گروهی از قوی‌ترین افراد نیز اگر نتوانند به شکل تیمی کار کنند، شکست می‌خورند. عضوی قوی در تیم نرم‌افزاری فردی است که دارای مهارتهای مناسبی است و می‌تواند به خوبی با دیگران کار کند. کار خوب با دیگران، ارتباط و تعامل از داشتن ذکاوت برنامه نویسی سطح بالا مهمتر است.
انتخاب ابزار مناسب و قابل استفاده نیز در موفقیت سهم مهمی دارد. قبل از خرید ابزارهای گران قیمت با ویژگیهای بسیار زیاد، ابتدا بررسی کنید که آیا به این ابزار نیاز دارید. فرض نکنید که ابزارهای بزرگتر و بهتر به طور خودکار باعث می‌شوند که بهتر کار کنید. ساختن تیم از ساختن محیط مهمتر است. ابتدا تیم خوبی تشکیل دهید و سپس اجازه دهید که تیم محیط را براساس نیاز پیکربندی نماید.

ارزش نرم‌افزار عملیاتی نسبت به مستندات کامل

نرم‌افزار بدون مستندات بسیار دردسر آفرین می‌شود، زیرا کد رسانهٔ ایده آلی برای به ارتباط گذاشتن اهداف و ساختار سیستم نمی‌باشد. بنابراین، تیم نیاز به تولید مستندات قابل استفاده خواندنی دارد که ساختار سیستم و منطق موجود در بسیاری از تصمیم‌گیریها را توصیف کند. اما، این مستندات نباید زیاد بزرگ باشد زیرا زمان زیادی برای تولید و همزمان سازی آن با کد گرفته خواهد شد. بنابراین تنها باید مستندات لازم را تولید نمود.

ارزش همکاری مشتری نسبت به رعایت قرارداد

نرم‌افزار را نمی‌توان مانند یک کالا سفارش داد. نمی‌توانید توصیفی از نرم‌افزاری که می‌خواهید بنویسید و آن را به فردی بدهید که آن را با هزینه و زمان ثابتی تولید کند. این سبک از کار منجر به تولید محصولی با کیفیت پایین و درنتیجه شکست پروژه می‌شود. پروژه‌های موفق به شکل منظم و متناوب از نظرات مشتری استفاده می‌کنند. به جای این که همه چیز به قرارداد یا بیان مشخصات کار وابسته باشد، مشتری نرم‌افزار به شکل نزدیکی با تیم ساخت نرم‌افزار کار خواهد کرد و نظرات خود را به آنها منتقل خواهد نمود. البته، اهمیت قراردادها با اندازهٔ پروژه افزایش می‌یابد.

ارزش پاسخ به تغییرات نسبت به دنبال نمودن طرح و برنامه

اغلب توانایی پاسخ به تغییرات است که موفقیت یا شکست پروژه‌ها را تعیین می‌کند. هنگامی که برنامه‌ریزی می‌کنیم باید مطمئن شویم که برنامه‌ها انعطاف پذیری لازم برای تطبیق پذیری با شرایط تجاری و تکنولوژی را دارند. استراتژی مناسب برنامه‌ریزی این است که برنامه‌ریزی مشروح برای وظائفی که در چند هفتهٔ آینده درپیش است صورت گیرد و چون معمولا به شکل تخمینی نیازهای چندماه آینده را می‌دانیم، برنامه‌ای کلی‌تر برای چندماه آینده ایجاد شود. شرکت کنندگان در ساخت نرم‌افزار

برای انجام تغییرات آماده می‌شوند و قراردادها نیز طوری تنظیم می‌گردند که امکان بهبودها و تغییرات آتی را بدهند.

جنبه‌های اصلی مدلهای سریع الانتقال سادگی و سرعت آنها می‌باشد. در فرآیند ساخت، گروه ساخت فقط برروی اعمال کلیدی وموردنیاز در وهلهٔ اول متمرکز می‌شوند، آنها را به سرعت ارائه می‌کنند، بازخورد را جمع آوری می‌کنند و نسبت به اطلاعات جمع آوری شده واکنش نشان می‌دهند. به طورکلی، می‌توان گفت که مدل فرآیند aglie دارای ویژگیهای زیر می‌باشد:

- افزایشی باشد. ساختهای کوچک نرم‌افزار در چرخه‌های کوتاه
- مبتنی بر همکاری باشد.. مشتری و سازندگان سیستم به طورپیوسته با ارتباطی نزدیک با یکدیگر کار کنند. تطبیق پذیر با تغییرات باشد.
- یادگیری و تغییر آنها آسان باشد
- به خوبی مستند شده باشد.

مدلهای فرآیند نرم‌افزار که می‌توان آنها را در گروه مدلهای سریع الانتقال به حساب آورد، به شرح زیر می‌باشند:

- Scrum [۳۰]
- Crystal [۱۱]
- Feature Driven Development [۲۷]
- Adaptive Siftware Development [۱۷]
- Exterme Programming [۳]
- Dynamic System Development Method [۱۲,۳۵]
- Open Source Software development [۲۶]
- Rational Unified Process [۲۲]

Extreme Programming

Extreme Programming یا XP به منظور برطرف نمودن چرخه‌های عمر طولانی در مدلهای قدیمی با به کارگیری فعالیتهایی که در فرآیندهای ساخت نرم‌افزار درستی و خوبی آنها به اثبات رسیده اند، ارائه گردید[۳] . فعالیتهای موجود در XP برای خودشان جدید نمی‌باشند، اما در XP این فعالیتها و اصول طوری جمع آوری و سازمان دهی شده اند که در مجموع مدل فرآیندی جدید برای ساخت نرم‌افزار

ارائه می‌شود. واژهٔ "extreme" هم از اینجا آمده است که این اصول و فعالیتهای بدیهی و روزمره در XP به سطوح نهایی خود رسیده اند.XP دارای چهار پنج مرحله می‌باشد: Exploration، Planning، Iterations to Release، Productionizing، Maintenance و Death.

در مرحلهٔ Exploration مشتریان آن ویژگیهایی را تمایل دارند در نسخهٔ اول وجود داشته باشند برروی کارتهای داستان[1] می‌نویسند. در همین زمان، تیم ساخت، ابزارها، تکنولوژیها و تکنیکهای مورداستفاده در پروژه را شناسایی می‌کنند. تکنولوژی مورداستفاده در پروژه تست می‌شود و معماریهای ممکن برای سیستم با ساختن نمونه‌هایی ارزیابی می‌گردند. بسته به اندازهٔ پروژه و میزان آشنایی برنامه نویسان به تکنولوژی مورداستفاده، این مرحله چند هفته تا چند ماه طول می‌کشد.

مرحلهٔ Planning اولویت داستانها را تعیین می‌کند و توافقی نسبت به محتوی ساخت کوچک اول حاصل می‌گردد. سازندگان سیستم میزان تلاش موردنیاز برای هر داستان را تخمین می‌زنند و توافقی برروی زمان‌بندی حاصل می‌گردد. مدت زمان ساخت اول معمولا از دوماه بیشتر نمی‌شود. خود مرحلهٔ زمان‌بندی دو یا سه روز بیشتر طول نمی‌کشد.

مرحلهٔ Iterations to Release شامل چندین تکرار پیش از تولید ساخت اول است. زمان‌بندی ایجاد شده به چندین تکرار شکسته می‌شود که پیاده‌سازی هر تکرار نیز یک تا چهار هفته به طول می‌انجامد. اولین تکرار سیستمی با معماری کل سیستم ایجاد می‌کند. این معماری با انتخاب داستانهایی که منجر به ایجاد ساختار کل سیستم می‌شوند، انجام می‌پذیرد. مشتری در رابطه با ویژگیهایی که باید در هر ساخت درنظر گرفته شود، تصمیم می‌گیرد. تستهای کارکردی که توسط مشتری ایجاد شده اند در انتهای هر تکرار اجرا می‌شوند. در انتهای آخرین تکرار سیستم آمادهٔ تولید است. پیش از ارائه سیستم به مشتری لازم است تا تستها و کنترلهایی در رابطه با کارآیی سیستم انجام پذیرد.

شیوه‌های کار در XP

XP با به کارگیری مجموعه‌ای از ایده‌ها و شیوه‌های کار برای ساخت نرم‌افزار تیم را شکل می‌دهد و با فراهم آوردن بازخوردهای کافی تیم را قادر می‌سازد تا شیوه‌های کار خود را براساس وضعیت فعلی تنظیم کنند[۳] . شیوه‌های کار XP عبارتند از:

بازی برنامه‌ریزی. برنامه‌ریزی در XP به منظور پیش بینی خروجیها در تاریخ تحویل و تعیین کارهای بعدی انجام می‌گیرد. این برنامه‌ریزی به دو شکل انجام می‌شود:

[1] story cards

در برنامه‌ریزی ساخت مشتری ویژگیهای مطلوب را به سازندگان سیستم ارائه می‌کند و سازندگان سیستم پیچیدگی هر ویژگی را تخمین می‌زنند. با داشتن تخمین هزینه‌ها و آگاهی از اهمیت ویژگیها، برنامه‌ای تخمینی برای پروژه ایجاد می‌گردد. با توجه به این که این برنامه لزوما دقیق نمی‌باشد، تیمهای XP به طور منظم این برنامه را بهنگام می‌سازند.

در طی برنامه‌ریزی برای تکرارها مشتری ویژگیهای مطلوب برای دو هفتهٔ بعد را مشخص می‌نماید. سازندگان سیستم وظائف را تقسیم می‌کنند و هزینهٔ انجام آنها را تخمین می‌زنند. براساس میزان کار انجام شده در تکرار قبلی، میزان کار تکرار جاری تعیین می‌گردد.

تستهای مشتری. مشتری به همراه ارائهٔ ویژگیهای موردنیاز، یک یا چند تست پذیرش خودکار نیز برای تست وجود ویژگی مزبور در سیستم ارائه می‌کند. تیم این تستها را می‌سازد و از آنها برای اثبات این که ویژگی به درستی پیاده‌سازی شده است، استفاده می‌کند. خودکارسازی تست اهمیت دارد زیرا در هنگام فشرده بودن زمان انجام پروژه، از تستهای دستی صرف نظر می‌شود.

ساختهای کوچک. سیستم در ساختهای کوچک تست و تحویل می‌گردد. بدین ترتیب، در انتهای هر تکرار، نرم‌افزار قابل رویت می‌باشد و برای کار به مشتری تحویل داده می‌شود.

طراحی ساده. تیمهای XP بر روی ساده‌ترین راه حل ممکن را که در وضعیت جاری قابل پیاده‌سازی باشد، تاکید دارند. پیچیدگی غیرلازم و کد اضافی فورا حذف می‌گردد. تیم XP طراحی دقیقا مناسب کارکرد فعلی سیستم انجام می‌دهند. طراحی در XP فعالیتی است که در تمام مراحل انجام می‌شود. در برنامه‌ریزی برای ساخت و تکرار مراحلی برای طراحی وجود دارد. علاوه براین، در طی فاکتورگیری مجدد و در طی انجام کل پروژه، تیمهای XP در جلسات طراحی سریع و بازبینی طراحیها شرکت می‌کنند.

بهبود طراحی. در XP از تکنیک بهبود طراحی به نام فاکتورگیری مجدد[1] استفاده می‌شود. فرآیند فاکتورگیری مجدد به حذف تکرارها (نشانه‌ای از طراحی ضعیف) می‌پردازد و سعی دارد تا چسبندگی[2] کد را افزایش و coupling کد را کاهش دهد. نتیجه این است که تیم XP با طراحی ساده آغاز می‌کند و همیشه دارای طراحی خوب و ساده برای نرم‌افزار است. بدین ترتیب، سرعت ساخت افزایش می‌یابد.

فاکتورگیری مجدد با تست کامل نرم‌افزار همراه می‌گردد تا این تضمین به وجود آید که با تکامل طراحی مشکلی در نرم‌افزار به وجود نمی‌آید. این تستها، تستهاس مشتری و تستهای برنامه نویس هستند که هر دو برروی نرم‌افزار انجام می‌شوند.

---

[1] refactoring
[2] cohesion

ساخت مبتنی بر تست. در XP گرفتن نظریات مشتری و بازخورد اهمیت زیادی دارد و در ساخت نرم‌افزار گرفتن بازخورد خوب نیاز به تست و ارزیابی دارد. اعضای تیم XP در چرخه‌های زمانی خیلی کوتاه کد با پوشش تست ۱۰۰ درصد تولید می‌کنند. تستهای برنامه نویس یا تستهای واحدی در هر زمان که برنامه نویس کدی را تولید می‌کند، اجرا می‌شوند. بدین ترتیب، برنامه نویسها در رابطه با کاری که انجام می‌دهند، بازخوردی فوری دریافت می‌کنند.

برنامه نویسی زوجی[1]. در XP گروههای دونفره‌ای از سازندگان سیستم که درکنار هم می‌نشینند و از یک ماشین استفاده می‌کنند، فعالیتهای ساخت سیستم را انجام می‌دهند. این شیوهٔ کار تضمین می‌کند که تمام محصولات کاری توسط حداقل یک سازندهٔ دیگر بازبینی می‌گردد و درنتیجه طراحی، تست و کدنویسی بهتر انجام می‌پذیرد. دو مغز از یک مغز واقعا بهتر هستند! این شیوهٔ کار به شکل زوج باعث به ارتباط گذاشتن دانش در سراسر تیم نیز می‌گردد.

مالکیت جمعی کد. در هر پروژهٔ XP هر زوج برنامه نویس می‌توانند در هر زمان هر کدی را بهبود دهند. یعنی تمام کدها موردتوجه همهٔ افراد هستند که این امر باعث افزایش کیفیت کد و کاهش خطاها می‌شود.

استاندارد کدنویسی. تیمهای XP از استاندارد کدنویسی مشترکی استفاده می‌کنند، به طوری که به نظر می‌آید تمام کدهای سیستم توسط یک فرد خبره نوشته شده اند. رعایت استاندارد به خصوصی مهم نیست بلکه اهمیت دارد که تمام کدها برای همه آشنا باشند. این شیوهٔ کار امکان مالکیت جمعی کد را افزایش می‌دهد.

تشبیه. تیمهای XP دارای آرمان و دیدگاه مشترکی نسبت به چگونگی کار برنامه هستند که به آن استعاره می‌گویند. تشبیه توصیفی از چگونگی کار برنامه می‌باشد. به عنوان مثال، این برنامهٔ سیستم بازیابی اطلاعات مبتنی برعامل مانند کندوی عسلی کار می‌کند که در آن زنبورها برای جمع آوری گرده به بیرون می‌روند و گرده‌ها را به کندو می‌آورند. بعضی اوقات، تشبیه مناسبی یافت نمی‌شود. در این مواقع، تیمهای XP از سیستم مشترکی از اسامی استفاده می‌کنند تا مطمئن شوند که همه چگونگی کار سیستم را درک می‌کنند.

حداکثر کار ۴۰ ساعت در هفته. هر سازندهٔ سیستم نباید بیشتر از ۴۰ ساعت در هفته کار کند. در صورت به وجود آمدن اضافه کار، این پدیده به عنوان مشکل به حساب می‌آید.

مشتری در سایت. مشتری باید به عنوان عضوی از تیم به شکل تمام وقت حاضر و در دسترس باشد.

---

[1] pair programming

متدولوژی XP نیز در همهٔ پروژه‌ها مناسب نمی‌باشد. XP برای پروژه‌های کوچک و متوسط درنظر گرفته شده است. Beck پیشنهاد می‌کند که اندازهٔ تیم بین ۳ تا حداکثر ۲۰ نفر محدود گردد. ارتباط و هماهنگی بین اعضای تیم در تمام مواقع باید ممکن باشد. به عنوان مثال، Beck پیشنهاد می‌کند که پراکنده شدن اعضای تیم در دو طبقه یا حتی در یک طبقه برای XP غیرقابل تحمل است. اما، توزیع جغرافیایی تیمها درخارج از محدودهٔ XP نمی‌باشد. حالتی می‌تواند وجود داشته باشد که در آن دو تیم با تعامل محدود برروی پروژه‌های مرتبط به هم کار کنند.

فرهنگ تجاری موجود در تیم ساخت یکی دیگر از مسائل مهم در XP می‌باشد. هر نوع مقاومتی در برابر شیوه‌های کاری و اصول XP از سوی اعضای تیم، مدیریت و مشتری ممکن است باعث شکست فرآیند گردد.

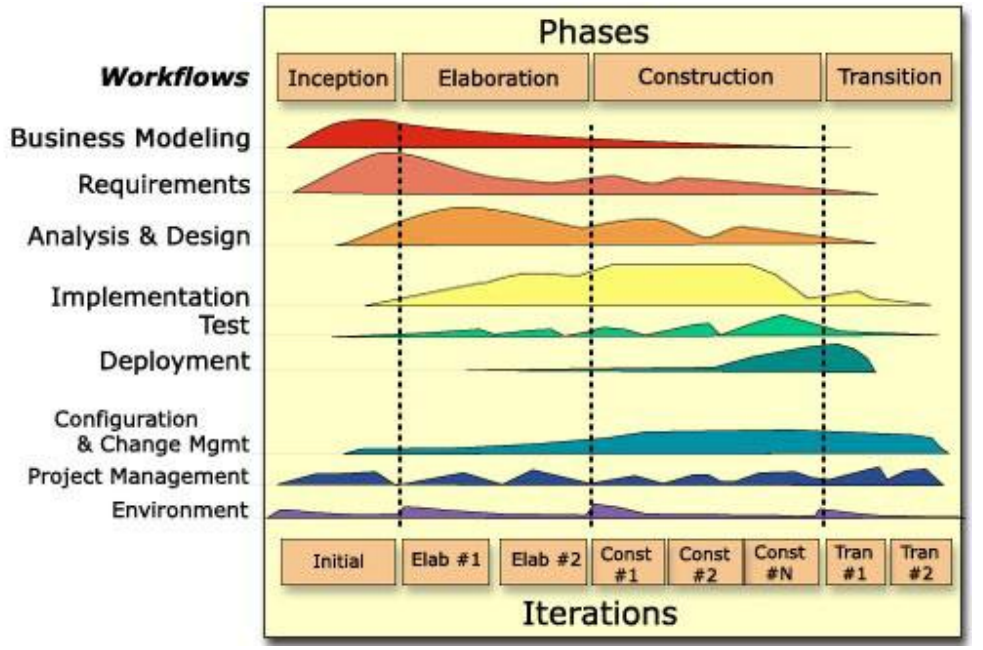
همچنین تکنولوژی مورداستفاده نیز می‌تواند مانعی در برابر موفقیت پروژه‌های XP باشد. به عنوان مثال، اگر تکنولوژی مورداستفاده تغییرات تدریجی را پشتیبانی نکند یا نیاز به زمان بازخورد بالا داشته باشد، این تکنولوژی برای XP مناسب نمی‌باشد.

Rational Unified Process

RUP یک فرآیند مهندسی نرم‌افزار است که توسط Philippe Krutchen، Ivar Jacobson و سایرین در شرکت Rational به منظور تکمیل زبان مدل سازی UML ایجاد گردید[۲۲] . RUP رهیافتی تکرار شونده[1] و مبتنی بر UML است که در آن موارد کاربرد[2] پایه‌ای برای ساخت سیستم هستند[3]. همانگونه که در شکل ۳-۹ دیده می‌شود. فرآیند RUP دارای دو ساختار یا بعد می‌باشد:

- بعد افقی که نشان دهندهٔ زمان است و جنبه‌های چرخهٔ حیات فرآیند را نشان می‌دهد.
- بعد عمودی که نشان دهندهٔ جریانهای کاری کلیدی است که فعالیتهای مهندسی نرم‌افزار را به شکل منطقی براساس طبیعت آنها گروه‌بندی می‌کنند.

iterative [1]
use case [2]
use case driven [3]



شکل ۳-۸ دو بعد فرآیند RUP

بعد افقی جنبهٔ پویای فرآیند را بازنمایی می‌کند و برحسب چرخه‌ها، مراحل، تکرارها و مقاطع بیان می‌گردد. در RUP محصول نرم‌افزاری برحسب توالی تکرارهای افزایشی ساخته می‌شود. این ساخت افزایشی امکان تست و اعتبارسنجی ایده‌های طراحی و همچنین کنترل ریسکهایی را که در طی چرخه حیات نرم‌افزار رخ می‌دهد، به وجود می‌آورد. براساس این تقسیم‌بندی هر پروژهٔ RUP دارای چهار مرحله به نامهای Inception، Elaboration، Construction و Transition می‌باشد. این مراحل به تکرارهایی تقسیم می‌شوند که هرکدام هدف به خصوصی را در تولید بخش قابل نمایشی از نرم‌افزار ایفا می‌کنند.

در مرحلهٔ Inception اهداف پروژه مشخص می‌گردند به طوری که نیازهای تمام ذی نفعان (مانند کاربر نهایی، خریدار، یا طرف قرارداد) درنظر گرفته شود. این فعالیت شامل مشخص نمودن محدودهٔ پروژه، مرزهای پروژه و معیارهای پذیرش پروژه می‌باشد. مورد کاربردهایی که در کارکرد سیستم نقش حیاتی دارند، شناسایی می‌گردند. معماریهای ممکن برای سیستم پیشنهاد می‌شود و زمان‌بندی و هزینه برای کل پروژه و برای مرحلهٔ بعدی تخمین زده می‌شود.

در مرحلهٔ Elaboration پایه‌های معماری نرم‌افزار بنا نهاده می‌شود. حوزهٔ مسئله با درنظر گرفتن کل سیستمی که قرار است ساخته شود، تحلیل می‌گردد. اگر تصمیم به ادامهٔ پروژه باشد، برنامهٔ پروژه ایجاد می‌گردد. در RUP فرض می‌شود که در این مرحله معماری معتبری به همراه نیازها و برنامهٔ پروژه به دست می‌آید. بعد از این مرحله، اغلب مورد کاربردها و تمام بازیگرها شناسایی می‌شوند، معماری نرم‌افزار توصیف می‌گردد، نمونهٔ اجرایی از معماری ایجاد و اجرا می‌گردد. در انتهای مرحلهٔ Elaboration تحلیلهای لازم برای تعیین ریسکها، میزان ایستایی vision سیستم نرم‌افزاری، ایستایی معماری و میزان استفاده از منابع در مقایسه با آنچه که در ابتدا برنامه‌ریزی شده مشخص می‌شود.

در مرحلهٔ Construction تمام اجزای باقیمانده و ویژگیهای برنامهٔ کاربردی ساخته شده و در محصول یکپارچه شده و تست می‌شوند. نتایج مرحلهٔ ساخت (نسخه‌های آلفا، بتا و تست) با سرعت ممکن ساخته می‌شوند. پیش از رفتن به مرحلهٔ Transitionدر طی مرحلهٔ Construction یک یا چند نسخه ساخته می‌شود.

هنگامی که مشخص شود که محصول برای تحویل به جامعهٔ کاربری به تکامل لازم رسیده است (این تکامل به عنوان مثال، با نظارت بر نرخ درخواست برای تغییر) مرحلهٔ Transtition آغاز می‌گردد. براساس پاسخ کاربر، نسخه‌های بعدی برای اصلاح مشکلات موجود یا به اتمام رسانیدن ویژگیهایی که تا بدین مرحله مدنظر نبوده اند، ساخته می‌شوند. مرحلهٔ Transition شامل تست بتا، ایجاد پایلوت، آموزش کاربران و گروه نگهداری سیستم، ارائهٔ محصول به بازار، شبکهٔ توزیع و تیمهای فروش می‌باشد. در این مرحله ممکن است چندین تکرار صورت گیرد. مستندات کاربر (راهنماها، مواد درسی) نیز در این مرحله تولید می‌گردند.

بعد عمودی جنبهٔ ایستای فرآیند را براساس اجزای فرآیند بازنمایی می‌کند. این اجزا عبارتند از فعالیتها، جریانهای کاری (discinplines) محصولات و نقشها. در طی این چهار مرحلهٔ اصلی، ۹ گردش کاری[1] مطابق شکل ۳-۸ انجام می‌پذیرد. تمام این جریانهای کاری در هر تکرار و با توجه هدف درنظر گرفته شده در هر مرحله انجام می‌پذیرند. این جریانهای کاری عبارتند از Business Modeling، Requirements، Analysis and Design، Implementation، Test، Configuration & Change Management، Project Management و Environment.

[1] workflow

RUP به شکل چارچوب فرآیند استفاده می‌شود و سازمانها می‌توانند با توجه به نیازها، مشخصه‌ها، محدودیتها، طبیعت سازمان، فرهنگ سازمانی و حوزهٔ کاربردی که در آن عمل می‌کنند، این چارچوب را تغییر دهند، تنظیم کنند و گسترش دهند. باید توجه داشت که نباید فرآیندی را به شکل کورکورانه دنبال نمود و محصولات بدون استفاده و بی ارزش را براساس آن تولید کرد. در عوض، باید فرآیند در ضمن این که سبک درنظر گرفته می‌شود، محصولات لازم را به منظور تولید سریع نرم‌افزاری با کیفیت تولید کند. علاوه بر فرآیند باید به شیوه‌های کاری سازمان موردنظر، رویه‌ها و قوانین مشخص به منظور تکمیل فرآیند نیز توجه شود. عناصری که احتمال تغییر آنها وجود دارد، محصولات نرم‌افزاری، فعالیتها، نقشها، جریانهای کاری، راهنماییهای انجام کار و قالب مستندات می‌باشد. در RUP ابزارهایی به منظور طراحی و مهندسی فرآیند و تولید گونه‌های خاص شرکت یا گونه‌های خاص پروژه از RUP به نام مورد ساخت[1] وجود دارد. خود RUP نیز شامل گونه‌ها یا موردهای ساخت از قبل آماده شده‌ای برای سازمانهای تولیدکنندهٔ نرم‌افزار می‌باشد.

شیوه‌های کار در RUP

در RUP شیوه‌های کار به شرح زیر می‌باشد[۲۲] :

- ساخت افزایشی نرم‌افزار
- مدیریت نیازها
- استفاده از معماریهای مبتنی بر اجزاء
- بررسی پیوستهٔ کیفیت نرم‌افزار
- کنترل تغییرات در نرم‌افزار

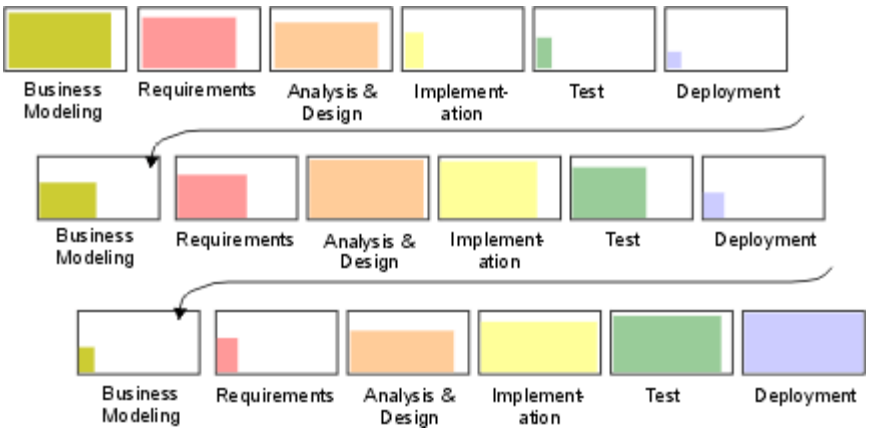
در ادامه، در رابطه با هر یک از این شیوه‌های کار توضیح مختصری داده می‌شود.

ساخت افزایشی نرم‌افزار

در RUP با معرفی مفهوم تکرار[2] فلسفهٔ ساخت تکراری نرم‌افزار دنبال می‌گردد. هر تکرار دربرگیرندهٔ مجموعه‌ای از فعالیتهای ساخت می‌باشد و منجر به تولید یک نسخه[3] از نرم‌افزار می‌گردد. این نسخه از نرم‌افزار که زیرمجموعه‌ای از محصو نهایی است، نگارشی ایستا و اجرایی از محصول به هر عنصر جانبی

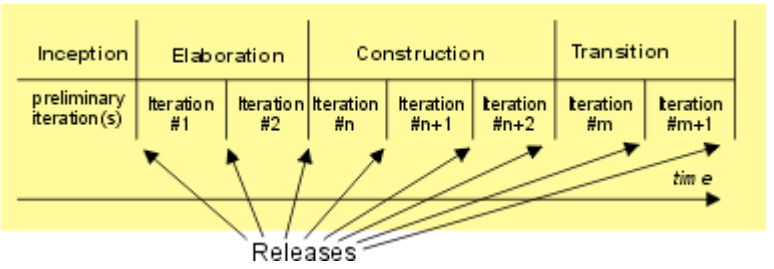
development case [1]
iteration [2]
release [3]

لازم برای استفاده از این نسخه می‌باشد. بنابراین، هر تکرار در فرآیند ساخت گذر کاملی است برروی حداقل جریانهای کاری Requirements، Analysis & Design، Implementation و Test. قابلیتهای درنظرگرفته شده در هر نسخه باید قابل نمایش باشد. در هر تکرار محصول کاملتر می‌گردد تا این که محصول نهایی به دست آید. مفهوم تکرار در شکل دیده می‌شود.



شکل ۳-۹. مفهوم تکرار در RUP

زمان انجام هر تکرار بستگی به اندازه و طبیعت هر پروژه دارد. هر تکرار با مقطع جزیی[1] جمع‌بندی می‌گردد. در این مقطع، نتایج هر تکرار بسته به معیارهای موفقیت آن تکرار که در برنامهٔ تکرار تدوین شده است، ارزیابی می‌گردد. هر مرحله در RUP به تکرارهایی تقسیم می‌گردد. رابطهٔ بین تکرار و مرحله در RUP در شکل ۳-۱۰ دیده می‌شود.



شکل ۳-۱۰. رابطهٔ تکرار با مراحل در RUP

[1] minor milestone

مدیریت نیازها

مدیریت نیازها، روشی سیستماتیک برای به دست آوردن، سازمان دهی، به ارتباط گذاشتن و مدیریت نیازهای درحال تغییر سیستم نرم‌افزاری است. RUP رهیافتی مبتنی بر موردکاربرد[1] است. یعنی، موردکاربردهایی که ایجاد می‌شوند، پایه‌ای برای سایر فعالیـتهای ساخت نرم‌افزار هستند. موردکاربردها برای به دست آوردن و مستندسازی نیازها استفاده می‌شوند و نقش عمده‌ای در طراحی، تست، طراحی رابط کاربر و مدیریت پروژه دارند.

استفاده از معماری مبتنی بر اجزاء

در RUP تمرکز اصلی فعالیتهای طراحی برروی معماری می‌باشد. در تکرار اول، تمرکز اصلی برروی تولید و تعیین اعتبار معماری می‌باشد. در چرخه‌های اولیهٔ ساخت، نمونهٔ اجرایی از معماری ایجاد می‌شود و به تدریج رشد می‌کند تا سیستم نهایی ایجاد گردد. در RUP قالبی برای توصیف معماری برمبنای دیدهای چندگانهٔ معماری ارائه می‌شود. در طی فرآیند طراحی فعالیتهای خاصی برای شناسایی محدودیتهای معماری و عناصر مهم برای معماری انجام می‌شود و راهنماییهایی برای انتخاب معماری ارائه می‌شود.

یکی دیگر از مفاهیم پایه در RUP ساخت برمبنای اجزاء می‌باشد. RUP به چند طریق ساخت برمبنای اجزاء را پشتیبانی می‌کند:

- رهیافت تکرارشونده به سازندگان سیستم امکان می‌دهد که به طورتکاملی اجزاء را شناسایی کنند و تصمیم بگیرند که چه اجزایی را بسازند، چه اجزایی را استفادهٔ مجدد کنند و چه اجزایی را خریداری نمایند.

- تمرکز برروی معماری نرم‌افزار باعث تعیین اجزاء، رابطهٔ بین آنها و الگوهای تعامل این اجزاء با یکدیگر می‌گردد.

- مفاهیمی مانند بسته‌ها، زیرسیستمها و لایه‌ها در طی تجزیه و تحلیل و طراحی به منظور سازمان دهی اجزاء و مشخص نمودن واسط آنها استفاده می‌شود.

[1] use case

- تست در RUP ابتدا برروی اجزاء صورت می‌گیرد و سپس به منظور تست مجموعه‌های بزرگتری از اجزاء گسترش می‌یابد.

مدل سازی بصری نرم‌افزار

در RUP از زبان مدل سازی UML برای به تصویرکشیدن، مشخص نمودن، ساخت و مستندسازی محصولات مرتبط با سیستم نرم‌افزاری در حال تولید استفاده می‌شود.

ارزیابی پیوستهٔ کیفیت

در رابطه با تضمین کیفیت نرم‌افزار، ایده‌ای که در RUP دنبال می‌شود این است که حفظ کیفیت نرم‌افزار مسئولیت تمام اعضای سازمان تولیدکنندهٔ نرم‌افزار است. در ساخت نرم‌افزار، توجه به کیفیت محصول و کیفیت فرآیند می‌باشد. در RUP مدیریت کیفیت در هر discipline به شکل زیر انجام می‌پذیرد:

Requirements : مدیریت کیفیت در این discipline شامل تجزیه و تحلیل مجموعهٔ مستندات نیازها به منظور بررسی سازگاری، وضوح و دقت نیازها می‌باشد.

Analysis & Design : مدیریت کیفیت شامل ارزیابی محصولات طراحی دربرگیرندهٔ بررسی سازگاری مدل طراحی با مدل نیازها و تبدیل آن به محصولات پیاده‌سازی می‌باشد.

Implementation : مدیریت کیفیت شامل ارزیابی محصولات پیاده‌سازی و ارزیابی کد برنامه‌ها و محصولات اجرایی و بررسی سازگاری محصولات پیاده‌سازی شده با محصولات نیازها، طراحی و تست می‌باشد.

Test: این dicipline کاملا مرتبط با مدیریت کیفیت می‌باشد و در آن سعی می‌شود تطابق محصول با نیازهای کارکردی و غیرکارکردی و استانداردها از طریق انجام آزمایشاتی بررسی گردد.

Environment : در این discipline راهنماییهای لازم به منظور دستیابی به مناسب‌ترین پیکربندی فرآیند برای برآورده شدن نیازها ارائه می‌گردد.

Deployment : در این discipline مدیریت کیفیت عبارت است از ارزیابی محصولات deployment و پیاده‌سازی و ارزیابی محصولات قابل اجرا براساس محصولات موردنیاز نیازها، طراحی و تست برای تحویل محصولات به مشتری می‌باشد.

Project Management: این discipline شامل بازبینی تمام فعالیتهای لازم برای مدیریت کیفیت شامل انجام بازبینیها و ممیزی‌های لازم برای ارزیابی پیاده‌سازی، دنبال نمودن و پیشرفت فعالیت ساخت می‌باشد.

کنترل تغییرات در نرم‌افزار

در طی هر فرآیند تکرارشونده، بسیاری از محصولات کاری اغلب تغییر می‌کنند. در RUP جزیی از فرآیند به نام Configuration & Change Management شامل مدیریت محصولات پروژه‌های نرم‌افزاری و تغییرات در جریان آنها می‌باشد. با ارائهٔ اولین نمونه به کاربران، تغییرات درخواست خواهد شد. به منظور کنترل این تغییرات و مدیریت موثر محدودهٔ پروژه و انتظارات ذی نفعان پروژه، لازم است که هر تغییری در هر کدام از محصولات ساخت از طریق فرآیند درخواست تغییرات پیشنهاد و توسط فرآیندی مدیریت گردد. درخواستهای تغییر برای مستندسازی و پیگیری مشکلات و درخواستهای اصلاح یا هر نوع درخواست دیگری برای تغییر در محصول استفاده می‌شود. این درخواستها برای مدیریت محدودهٔ پروژه و بررسی اثر تغییرات پیشنهاد شده در پروژه استفاده می‌شوند.

۳-۳ فرآیند نرم‌افزار و مدیریت پروژه

مدیریت موثر پروژه نرم‌افزاری متمرکز بر چهار P می‌باشد: People (افراد)، Product (محصول)، Process (فرآیند) و Project (پروژه). فرآیند نرم‌افزار چارچوبی را فراهم می‌کند که براساس آن برنامهٔ جامعی برای ساخت نرم‌افزار ایجاد می‌گردد. تعداد اندکی از فعالیتهای این چارچوب برای تمام پروژه‌های نرم‌افزاری بدون درنظر گرفتن اندازه و پیچیدگی آنها قابل استفاده می‌باشند. با درنظر گرفتن مشخصات پروژهٔ نرم‌افزاری و نیازهای تیم پروژه، مجموعه‌ای از وظائف – وظائف، مقاطع پروژه، محصولات کاری و نقاط تضمین کیفیت – به عنوان فعالیتهای چارچوب انتخاب می‌گردند. فعالیتهای فراگیری مانند تضمین کیفیت نرم‌افزار، مدیریت پیکربندی نرم‌افزار و اندازه‌گیری نرم‌افزار در طی فرآیند ساخت نرم‌افزار انجام می‌پذیرند و مستقل از فعالیتهای فرآیند نرم‌افزار هستند.

یکی از مسائلی که مدیر پروژه با آن رو به رو می‌باشد، انتخاب مناسب‌ترین مدل فرآیند نرم‌افزار برای ساخت سیستم نرم‌افزاری می‌باشد. در این بخش، در رابطه با انواع مدلهای فرآیند نرم‌افزار بحث

گردید. مدیر پروژه باید براساس معیارهای زیر در رابطه با مناسب‌ترین مدل فرآیند نرم‌افزار تصمیم بگیرد:

- مشتریانی که محصول نرم‌افزار را تقاضا کرده اند
- افراد و تیمی که محصول را تولید می‌کنند
- مشخصات خود محصول
- میزان ریسک
- محیطی که تیم نرم‌افزاری در آن کار می‌کنند

پس از انتخاب مدل فرآیند، تیم مدیریت پروژه براساس مجموعهٔ متداول فعالیتهای چارچوب، برنامهٔ اولیه‌ای را برای پروژه تعریف می‌کند. بعد از تعریف برنامهٔ اولیه، برنامهٔ کاملی که شامل وظائف کاری موردنیاز چارچوب می‌باشد، ایجاد می‌گردد.

۳-۴ بهبود فرآیند نرم‌افزار

بهبود فرآیند نرم‌افزار روشی برنامه‌ریزی شده با پیروی از فعالیتهای استاندارد به منظور مستندسازی و تجزیه و تحلیل فعالیتها، روشها، تکنیکها و ابزارهایی است که در سازمان نرم‌افزاری به منظور ساخت و نگهداری نرم‌افزار و محصولات مرتبط با آن استفاده می‌شود. CMM یکی از استانداردهای بالقوهٔ اصلی در راستای بهبود فرآیند نرم‌افزار است. سایر فعالیتهای مرتبط با بهبود فرآیند نرم‌افزار استانداردهای سری ISO-9000 و SPICE یا ISO/IEC 15504 می‌باشند. علاوه براین، با گردآوری، تحلیل و مهندسی تجارب حاصل از پروژه‌های نرم‌افزاری نیز می‌توان فرآیند بهبود نرم‌افزار را هدایت نمود[۳۲] .

مشکلاتی مانند طولانی شدن پروژه‌های نرم‌افزاری، هزینه‌های بالا، کیفیت پایین محصولات تولید شده، عدم توانایی در مدیریت فرآیند نرم‌افزار و سایر مسائل مرتبط با تولید نرم‌افزار، باعث شد تا دپارتمان دفاع (DoD) ایالات متحده موسسهٔ مهندسی نرم‌افزار (Software Engineering Institute) را در دانشگاه کارنگی ملون تاسیس کند. یکی از موفقیتهای این موسسه، ایجاد مدلهای CMM به منظور بهبود فرآیند نرم‌افزار (Software Process Improvement) بود. یکی از اهداف اولیه این فعالیت، افزایش کیفیت نرم‌افزارهای دفاعی با ارزیابی فرآیندهای نرم‌افزار شرکتهایی که می‌خواستند برای دپارتمان دفاع نرم‌افزار تولید کنند و انعقاد قرارداد با شرکتهای مطلوب بود.

مدلهای CMM گروه مرتبطی از استراتژیها برای بهبود نرم‌افزار بدون توجه به مدل فرآیند نرم‌افزار مورداستفاده می‌باشد. در CMM واژهٔ maturity در به معنای اندازه‌ای از خوب بودن خود فرآیند است. مدلهای ارائه شده عبارتند از: SW-CMM برای نرم‌افزار، P-CMM برای مدیریت منابع انسانی، SE-CMM برای مهندسی سیستم، IPD-CMM برای ساخت یکپارچهٔ محصول و SA-CMM برای کسب نرم‌افزار. علاوه براین، فعالیت CMM Integration یا CMMI سعی دارد این مدلها را که با یکدیگر همپوشانی نیز دارند، یکپارچه نماید[۲۹].

ایده‌ای که در SW-CMM دنبال می‌شود این است که استفاده از تکنیکهای جدید نرم‌افزاری به تنهایی باعث افزایش بهره‌وری و سوددهی نمی‌شود زیرا عمدهٔ مشکلات در چگونگی مدیریت فرآیند نرم‌افزار است. استراتژی SW-CMM بهبود مدیریت فرآیند نرم‌افزار است با این باور که بهبود در تکنیکها و روشها نتیجهٔ طبیعی این فعالیت خواهد بود[۳۴]. بهبود در فرآیند باعث کیفیت بهتر نرم‌افزار و اتمام پروژه‌های نرم‌افزاری با توجه به بودجه و زمان درنظرگرفته شده می‌باشد. تغییرات پیشنهادی و بهبودهای حاصل به تدریج و به شکل افزایشی انجام خواهند گرفت. SW-CMM پنج سطح تکامل برای ارزیابی سازمانها تعریف می‌کند:

سطح ۱: اولیه

در این پایین‌ترین سطح، فرآیند نرم‌افزار تعریف شده‌ای وجود ندارد و فرآیند نرم‌افزار به شکل اتفاقی می‌باشد. هیچ فعالیت مدیریت مهندسی نرم‌افزار معتبری در سازمان وجود ندارد. اگر پروژه‌ای مدیر کارآمد و تیم ساخت خوبی داشته باشد، ممکن است که این پروژه موفق شود. اما، اغلب به خاطر نبود مدیریت معتبر و برنامه‌ریزی پروژه‌ها با توجه به هزینه‌ها و زمان درنظر گرفته شده به پایان نمی‌رسند. در سازمانهای سطح ۱، فرآیند غیرقابل پیش بینی می‌باشد و همه چیز وابسته به افراد می‌باشد. با تغییر افراد، فرآیند نیز تغییر می‌کند.

سطح۲: قابل تکرار

در این سطح، فعالیتهای پایه‌ای مدیریت پروژه در سازمان وجود دارد. تکنیکهای برنامه‌ریزی و مدیریت مبتنی بر تجربهٔ حاصل از محصولات مشابه هستند. به همین خاطر به این سطح قابل تکرار گفته می‌شود. در سطح ۲، اندازه‌گیریهایی مانند اندازه‌گیریهای لازم برای پیگیری هزینه‌ها و زمان‌بندی صورت می‌گیرد. این اندازه‌گیریها ابزاری برای یافتن مشکلات پیش از وقوع آنها و رفع مشکلات می‌باشد. اندازه‌گیریهای انجام شده برای بدست آوردن تخمینهای واقعی هزینه و زمان در رابطه با پروژه‌های مشابه آینده قابل استفاده می‌باشند. در این سطح فرآیند لازم برای تکرار موفقیتهای قبلی در پروژه‌های مشابه وجود دارد.

سطح ۳: تعریف شده

در این سطح، فرآیند تولید نرم‌افزار به شکل کاملی مستند می‌شود. جنبه‌های مدیریتی و فنی فرآیند به شکل روشن تعریف می‌شوند و هر زمان که ممکن باشد، کارهای لازم برای بهبود فرآیند انجام می‌گیرد. در این سطح، بازبینها به منظور دستیابی به کیفیت انجام می‌پذیرند. در این سطح معرفی تکنولوژیهای جدید مانند محیطهای CASE به منظور افزایش بهره‌وری و کیفیت معقول به نظر می‌رسد. تکنولوژیهای جدید در سطح ۱ می‌توانند باعث آشوبگونه شدن تولید نرم‌افزار گردند. این سطح شامل تمام مشخصه‌های سطح ۲ می‌باشد.

سطح۴: مدیریت شده

سازمان سطح ۴ برای هر پروژه اهداف کیفیتی و بهره‌وری تعریف می‌کند. این دو کمیت به طور پیوسته اندازه گیری می‌شوند و هر زمان که انحرافی از این اهداف حاصل گردد، اعمال اصلاحی صورت می‌گیرد. کنترلهای آماری کیفیت مانند تعداد خطاهای کشف شده در هر ۱۰۰۰ خط برنامه، برای تمایز انحراف تصادفی یا معنی دار از استانداردهای کیفی یا بهره‌وری صورت می‌گیرد. با استفاده از سنجه‌ها، فرآیند نرم‌افزار و محصول از نظر کمی بررسی و کنترل می‌شوند. این سطح شامل تمام مشخصه‌های سطح ۳ می‌باشد.

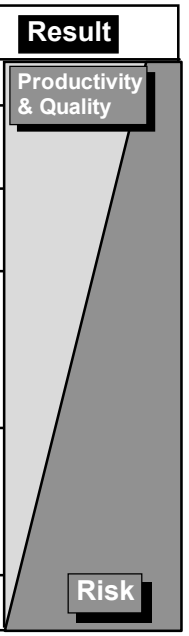
سطح ۵: درحال بهینه سازی

هدف سازمان سطح ۵ بهبود پیوستهٔ فرآیند می‌باشد. تکنیکهای آماری کنترل کیفیت و فرآیند به عنوان راهنمایی برای سازمان به منظور بهبود پیوستهٔ فرآیند استفاده می‌شوند. علت خرابیها و مشکلات شناسایی و مرتفع می‌گردد. تکنولوژیهای جدید شناسایی و برای استفاده از آنها برنامه‌ریزی می‌گردد. دانش بدست آمده در هر پروژه در پروژه‌های بعدی استفاده می‌شود. این سطح شامل تمام مشخصه‌های سطح ۴ می‌باشد.

به منظور بهبود فرآیند نرم‌افزار، هر سازمان در ابتدا با استفاده از روشها و پرسشنامه‌های ارزیابی موجود، درکی از وضعیت موجود فرآیند خودش به دست می‌آورد وسپس فرآیند مطلوب را فرموله می‌سازد. سپس، اعمال لازم برای دستیابی به سطح موردنظر تعیین و اولویت‌بندی می‌شوند. در نهایت، برای بهبود فرآیند برنامه‌ریزی و برنامه اجرا می‌شود.

SEI برای هر سطح، تعدادی نواحی کلیدی فرآیند (Key Process Areas) در نظر می‌گیرد. KPA اعمال مهندسی نرم‌افزار مانند مدیریت نیازها و مدیریت پروژه هستند که برای دستیابی به اهداف سطح موردنظر باید انجام شوند. هر KPA با مشخصه‌های اهداف، تعهدات برای دستیابی به اهداف، تواناییهای لازم برای برآوردن تعهدات، فعالیتها، روشهای نظارت بر پیاده‌سازی و روشهای تایید پیاده‌سازی شناسایی می‌گردد. در شکل ۳-۷ رابطهٔ نواحی کلیدی فرآیند با هر سطح تکامل CMM دیده می‌شود.

Level	Characteristic	Key Process Areas
Optimizing (5)	Continuous process capability improvement	Process change management Technology change management Defect prevention
Managed (4)	Product quality planning; tracking of measured software process	Software quality management Quantitative process management
Defined (3)	Software process defined and institutionalized to provide product quality control	Peer reviews Intergroup coordination Software product engineering Integrated software management Training program Organization process definition Organization process focus
Repeatable (2)	Management oversight and tracking project; stable planning and product baselines	Software configuration management Software quality assurance Software subcontract management Software project tracking& oversight Software project planning Requirements management
Initial (1)	Ad hoc (success depends on heroes)	"People"



شکل ۳-۱۲. رابطهٔ نواحی کلیدی فرآیند با هر سطح تکامل CMM

۳-۵ مطالعهٔ موردی: مدل فرآیند در ISMN

یکی از مهمترین تصمیم‌گیریها در مدیریت پروژهٔ ISMN انتخاب مدل فرآیندی بود که استراتژی ساخت سیستمهای ISMN را مشخص کند و در بستر آن روشها و ابزارهای موردنیاز به کارگرفته شود. در این بخش، تجربهٔ حاصل از ارائه و به کارگیری مدل فرآیند مناسب با نیازهای پروژه توضیح داده می‌شود. همچنین، سعی می‌گردد تا مهمترین شیوه‌های کاری که در پروژهٔ ISMN به انجام رسیدند یا در صورت به کارگیری این شیوه‌های کاری نتایج مطلوبتری حاصل می‌گردید، مورد اشاره قرار گیرند.

یکی از پرسشهای مهم در پروژهٔ ISMN این بود که از میان مدلهای موجود، مدل فرآیند برای انجام این پروژه چه باید باشد؟ مدل فرآیند نرم‌افزار براساس طبیعت پروژه و برنامهٔ کاربردی، روشها و ابزارهای مورداستفاده، و کنترلهای موردنیاز در پروژه و خروجیهای موردنیاز انتخاب می‌گردد. مدل فرآیند مورد استفاده در ISMN باید دارای ویژگیهای زیر می‌بود:

- سازگاری با استانداردهای ارائه شده برای تجزیه و تحلیل و طراحی سیستمهای مبتنی بر TMN. در [۱۹] روش UTRAD یا Unified TMN Requirements Analysis and Design به عنوان راهنمایی برای تحلیل و طراحی این گونه سیستمها ارائه شده است.
- درنظر گرفتن خبرگی تیم ساخت سیستمها. تیم ساخت سیستمها در ISMN در حوزهٔ کاربردی موردنظر تجربهٔ عملی چندانی نداشت.
- درنظر گرفتن حوزهٔ کاربردی. حوزهٔ کاربردی مدیریت شبکهٔ سوییچ برای تیم سازنده حوزه‌ای کاملا جدید بود. اما، در دنیا سیستمهای مشابه فراوانی با قابلیتهای مشابه وجود داشت که امکان بهره گیری از تجارب مشابه را فراهم می‌آورد.
- درنظر گرفتن طبیعت سیستمهایی که باید ساخته می‌شدند:
  - یکی از نکات قابل توجه در رابطه با ایجاد سیستمهای مبتنی بر TMN وجود استانداردهایی است که سبک معماری فیزیکی، معماری کارکردی و معماری اطلاعاتی سیستمها را مشخص می‌کنند[۱۸] و طراحان سیستم معمولا با درنظرگرفتن معماریهای پیشنهاد شده در این استانداردها و نیازهای کلان موجود، معماری سیستمی شبکهٔ مدیریت مخابرات را طراحی می‌کنند.
  - استانداردهای TMN در چارچوب مجموعه کارکردهای مدیریتی و سرویسهای مدیریتی قالبی را برای تعریف نیازهای کارکردی سیستمهای مدیریتی ارائه می‌کنند. بنابراین، می‌توان گفت که نیازهای این سیستمها براساس استانداردهای ارائه شده قابل شناسایی هستند و میزان تغییرات در آنها زیاد نمی‌باشد. از سوی دیگر، کارکردهای بیان شده بسیارکلی هستند و راهنمایی خاصی در رابطه با جزییات و چگونگی پیاده‌سازی ارائه نمی‌کنند. همچنین این استانداردها مستقل از شبکهٔ مدیریت شونده ارائه شده اند. بنابراین تنها می‌توان از آنها به عنوان یک راهنما استفاده نمود و برای به دست آوردن نیازهای مشروح باید فعالیت به دست آوردن نیازها با دقت انجام پذیرد.
- سادگی درک و قابل فهم بودن
- فراهم آوردن امکان انجام کنترلهای مدیریتی

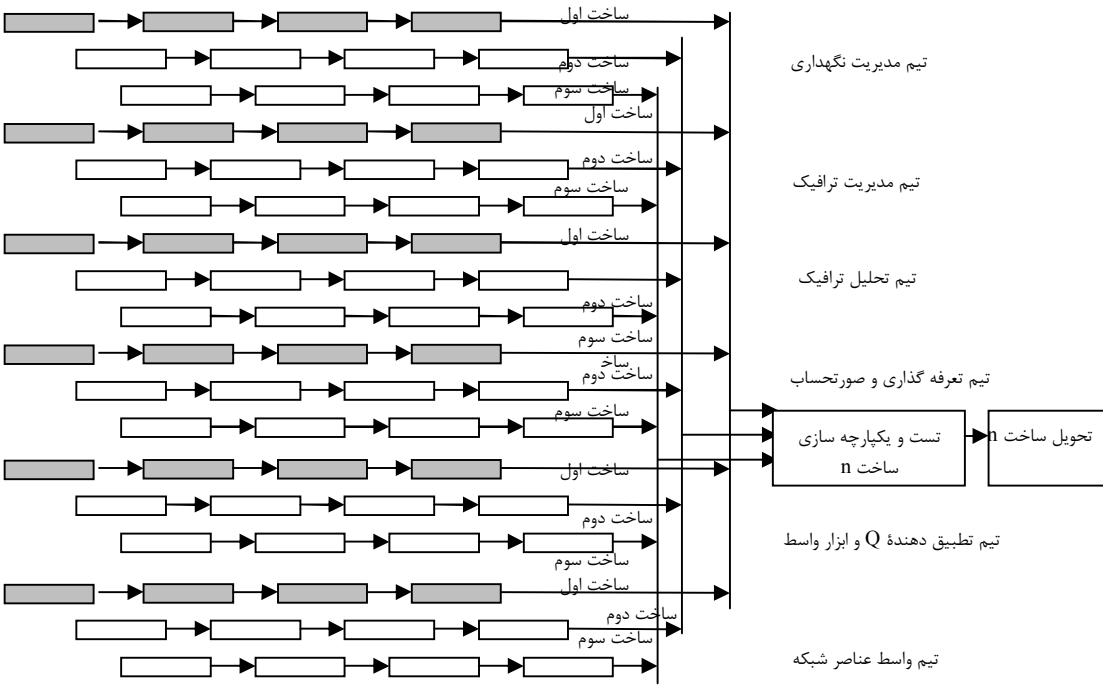
با توجه به نیازهای فوق در رابطه با مدل فرآیند درخواستی، در پروژهٔ ISMN، از فرآیندی به نام UTDP (Unified TMN Development Process) که برای تولید سیستمهای مبتنی بر TMN در گروه مدیریت شبکه مرکز تحقیقات مخابرات ایران ارائه شده است، استفاده شد. این مدل، عناصر مدل آبشاری، فلسفهٔ تولید تکراری و افزایشی در مدل نمونه سازی و فلسفهٔ تقسیم سیستم به کارکردهای مستقل و ساخت موازی در RAD (Rapid Application Development) ترکیب می‌کند. مراحل تحلیل و طراحی این مدل ایده گرفته از متدولوژی UTRAD [۱۹] می‌باشد و زبان مدل سازی UML در آن پشتیبانی می‌شود. UTDP مبتنی بر ایجاد افزایشی ساختهای اجرایی از سیستم است. مدل مورداستفاده در شکل ۳-۱۱ دیده می‌شود.

با توجه به استانداردهای موجود، طبیعت نرم‌افزار، اولویتهای مشتری و طبیعت بسیار پویای محیطی که سیستمها در آن تولید می‌شدند، تصمیم گرفته شد تا بعد از انجام مهندسی سیستم و تخصیص نیازها به اجزای سیستم، سیستم براساس کارکردهای مدیریتی به ساختهایی تقسیم شود. ساختهای درنظر گرفته شده برای زیرسیستمهای مختلف سیستم در جدول ۳-۱ دیده می‌شود.

همانگونه که در شکل ۲ دیده می‌شود، هر تیم براساس ساختهای تعریف شده، فرآیند ساخت مراحل تجزیه و تحلیل، طراحی، پیاده‌سازی و تست را دنبال می‌کند و در انتها توسط تیمها فرآیند یکپارچه سازی همهٔ ساختها با یکدیگر و تست و تحویل ساخت کلی سیستم انجام می‌پذیرد. در طی انجام فرآیند از کارشناسان مجموعهٔ شرکت مخابرات به عنوان "مشتری در سایت" برای به دست آوردن نیازها، رفع مشکلات مرتبط با حوزهٔ مخابراتی و ارزیابی سیستمها بهره گرفته شد. به منظور موفقیت این استراتژی ساخت، لازم است تا شیوه‌های کاری موفق در انجام پروژه‌های نرم‌افزاری نیز همراه مدل فرآیند ارائه شده گردد که در ادامه به بیان شیوه‌های کاری تجربه شده می‌پردازیم.

جدول ۳-۱. ساختهای درنظر گرفته شده برای ساخت افزایشی و تکراری سیستمهای پروژهٔ ISMN

سیستم مدیریت نگهداری	**سیستم تحلیل ترافیک**
ساخت اول: نظارت برآلارم و همبستگی آلارمها ساخت دوم: تعیین محل خرابی ساخت سوم: سرپرستی مشکل و رفع خرابی	ساخت اول: تحلیلهای ترافیکی در رابطه با سوییچ، ترانک گروپ ساخت دوم: تحلیلهای ترافیکی در رابطه با مقصد، گزارشهای مدیریتی ساخت سوم: تحلیلهای ترافیکی پیشرفته (ساعت شلوغی، روند ترافیکی، تحلیل کارآیی، زمان‌بندی تحلیلها)
سیستم مدیریت ترافیک	**سیستم صدورصورتحساب و تعرفه گذاری**
ساخت اول: پایش وضعیت ترافیک ساخت دوم: پایش عملکرد ترافیک و سرپرستی ترافیک ساخت سوم: کنترل ترافیک	ساخت اول: تعرفه گذاری و مدیریت حسابها ساخت دوم: جمع آوری برخط CDR و تولید صورتحساب ساخت سوم: تولید صورتحساب براساس فرمولها ، ترازها ، و تناقضهای مالی
سیستم ابزار واسط و تطبیق دهنده	**واسطه با عناصر سوییچ**
ساخت اول: کارکردهای لازم برای پشتیبانی ساخت اول سیستمهای مدیریتی	ساخت اول: کارکردهای لازم برای پشتیبانی ساخت اول سیستمهای مدیریتی
ساخت دوم:کارکردهای لازم برای پشتیبانی ساخت دوم سیستمهای مدیریتی	ساخت دوم: کارکردهای لازم برای پشتیبانی ساخت دوم سیستمهای مدیریتی
ساخت سوم: کارکردهای لازم برای پشتیبانی ساخت سوم سیستمهای مدیریتی	ساخت سوم: کارکردهای لازم برای پشتیبانی ساخت سوم سیستمهای مدیریتی



شکل ۳-۱۱. UTDP مدل فرآیند پیشنهادی برای ساخت سیستمهای ISMN

شیوه‌های کاری تجربه شده در پروژه ISMN

شیوه‌های کاری تجربه شده در فرآیند انجام پروژهٔ ISMN شیوه‌های کاری تجربه شده در اکثر پروژه‌های نرم‌افزاری هستند که درستی آنها در این پروژه هم به اثبات رسیده است:

توسعهٔ افزایشی و تکرارشوندهٔ نرم‌افزار

توسعه به روش افزایشی و تکرارشونده نوعی روش حل مسئله به سبک "تقسیم کن و غلبه کن" (divide and conquer) می‌باشد که در آن به منظور انجام یک پروژه بزرگ و پیچیده سعی می‌گردد تا این پروژه به اجزای کوچکتری که تعریف، طراحی و پیاده‌سازی آن آسان‌تر و دقیق‌تر انجام می‌گیرد، شکسته شود و این اجزا به تدریج ایجاد شوند تا محصول نهایی کامل گردد.

مدلهای فرآیند و متدولوژیهای سُبک اما کارا و گویا

پروژه‌های بزرگ که از نظر تکنولوژیکی و مدیریتی دارای پیچیدگی می‌باشند، نیاز به مدلهای فرآیندی و متدولوژیهایی دارند که با تولید محصولات کاری غیرلازم یا معرفی مراحل غیرضروری برای انجام پروژه، عامل دیگری را به عوامل پیچیدگی پروژه اضافه نکنند. فرآیندها و متدولوژیهای مورداستفاده در پروژه در ضمن این که باید سبک باشند باید محصولات لازم را به منظور تولید سریع نرم‌افزاری با کیفیت تولید کنند. علاوه بر فرآیند باید به شیوه‌های کاری سازمان موردنظر، رویه‌ها و قوانین مشخص به منظور تکمیل فرآیند نیز توجه شود.

آماده سازی محیط توسعهٔ نرم‌افزار

یکی از فعالیتهایی که در پروژه‌های نرم‌افزاری جدید، بزرگ، پیچیده نیاز به انجام آن وجود دارد، آماده‌سازی محیط پروژه است. آماده سازی محیط که در Rational Unified Process نیز به عنوان یک "گردش کار" تکمیلی به آن اشاره شده است، فعالیتی است که لازم است در مرحلهٔ شناخت پروژه‌های بزرگ تیمی را به انجام آن اختصاص داد. آماده سازی محیط توسعهٔ نرم‌افزار می‌تواند شامل موارد زیر باشد:

- تعریف محصولاتی که باید در طی انجام فرآیند تولید شوند و زمان تولید آنها براساس مدل فرآیند انتخاب شده
- ایجاد راهنماییهایی برای انجام فعالیتهایی مانند مدل سازی مورد کاربرد، مدل سازی کلاس و غیره.
- تعریف قالبهایی برای خروجیهای پروژه در شکلهای قابل استفاده برای اعضای پروژه

- بررسی، ارزیابی و انتخاب ابزارهایی که باید در طی مراحل مختلف پروژه استفاده شوند. این ابزارها شامل ابزارهای مدیریت نیازها، تحلیل و طراحی، پیاده‌سازی، تست، مدیریت پیکربندی و غیره می‌باشد. لازم است که این ابزارها انتخاب شوند، برای کاربردهای خاص پروژه پیکربندی شوند، چگونگی استفاده از آنها به افراد پروژه آموزش داده شود و در طی استفاده از این ابزارها پشتیبانی لازم برای آنها ارائه گردد.
- تحقیق و بررسی در رابطه با تکنولوژی موردنیاز و قابل استفاده در پروژه. انتخابهایی مانند زبان برنامه نویسی یا سرویس دهنده برنامه کاربردی (application server) تاثیر به سزایی در موفقیت پیاده‌سازی پروژه دارد. بدین منظور لازم است تا تیمی در رابطه با ارزیابی و انتخاب تکنولوژی در پروژه فعالیت کند. اگر نیازها مشخص باشد و تجربهٔ مشابه وجود داشته باشد، این انتخاب به آسانی انجام می‌پذیرد، اما به هر حال باید در همان ابتدای پروژه انجام شود. ممکن است نیازهای سیستم نرم‌افزاری در ابتدا مشخص نباشد، در این حالت لازم است تا از همان ابتدای پروژه تیم ارزیابی فعالیت خود را آغاز نماید تا در هنگامی که نیازها به میزان کافی مشخص گردید، با اطلاعات به دست آمده بدون صرف زمان و هزینهٔ زیاد انتخاب درستی در این رابطه انجام شود.

نکتهٔ مهم این است که بخشی از این فعالیتها مانند تعریف محصولات یا تعریف قالبهای خروجی محصول حتما باید در مراحل اولیهٔ پروژه صورت گیرند.

تیم کنترل و مدیریت پروژه

در پروژه‌های بزرگ، نیاز به تیمی می‌باشد که با استفاده از ابزارهای مدیریت پروژه و پیگیری به طور دقیق برنامه‌ریزی و پیشرفت فعالیتهای پروژه را تحت کنترل داشته باشند و مدیر پروژه و مدیران تیمها را از چگونگی پیشرفت پروژه آگاه و مشکلات احتمالی را گزارش کنند.

آموزش کارشناسان پروژه

لازم است که کارشناسان درگیر در پروژه در رابطه با فرآیند انجام پروژه، روشهای تحلیل، طراحی، تست و پیاده‌سازی و ابزارهایی که در پروژه مورد استفاده قرار خواهد گرفت، آموزش ببینند. این آموزش از راههای مختلف قابل انجام است:

- برگزاری دوره‌های آموزشی
- برگزاری کارگاههای آموزشی فشرده

- برگزاری کارگاههایی که در طی آن افراد با ابزارها و تکنولوژیهای جدید مورد استفاده در پروژه آشنا شوند و با آنها به شکل عملی کار کنند.
- وجود مربی برای بازبینی نتایج کار، راهنمایی افراد در حین انجام کار و پاسخ به پرسشهای افراد. این روش بسیار کارا و موثر است.

مشتری در سایت و درگیری مشتری در انجام پروژه

در رابطه با پروژه‌های بزرگ و حوزهٔ کاربردی آنها جدید یا ناشناخته می‌باشد، حضور مشتری یا نمایندگان مشتری در قالب تحلیل گر سیستم کاری یا کاربران سیستم در کنار تیمهای تحلیل، طراحی و پیاده‌سازی امری حیاتی به نظر می‌رسد. وجود مشتری امکان انجام بازبینهای مکرر و گرفتن نظرات مشتری در رابطه با محصولات مختلف تولید شده را به وجود می‌آورد. بدین ترتیب، مشکلات با سرعت بیشتری مرتفع می‌گردد و امکان انتقال تجارب و دانش حوزهٔ کاربردی به اعضای تیم توسعه فراهم می‌شود.

ایجاد استانداردهای یکنواخت در بین تیمهای پروژه

تعریف استانداردهای مستند سازی، مدل سازی، کد نویسی و تست در پروژه یکی از دیگر نکاتی است که می‌تواند سربار ارتباطی بین تیمهای مختلفی را که در پروژه فعالیت می‌کنند، کاهش دهد و باعث یکپارچه شدن تیم توسعهٔ پروژه گردد. رعایت استاندارد به خصوصی مهم نیست بلکه اهمیت دارد که تمام محصولات تولید شده در پروژه برای همه آشنا باشند. این شیوهٔ کار امکان مالکیت جمعی محصولات پروژه را افزایش می‌دهد.

مدیریت پیکربندی

به خاطر حجم بالای فعالیتهای تجزیه و تحلیل، طراحی و پیاده‌سازی که در پروژه‌های بزرگ انجام می‌گیرند و محصولات کاری متفاوتی که ایجاد می‌شوند، استفاده از روشهای مدیریت پیکربندی و کنترل تغییرات برروی این محصولات کاری نیز اهمیت زیادی می‌یابد. بدین منظور پیشنهاد می‌شود که تیم مجزایی به منظور انجام فعالیت مدیریت پیکربندی، شامل ایجاد رویه‌های لازم، انتخاب، تهیه، نصب و آموزش ابزارهای موردنیاز و پیگیری تغییرات در محصولات کاری پروژه در کنار سایر تیمهای اجرایی پروژه ایجاد گردد.

نتیجه گیری

فرآیند نرم‌افزار مجموعه‌ای از گامهای قابل پیش بینی به منظور ساخت نرم‌افزاری مقرون به صرفه و با کیفیت است. مدل نرم‌افزار با توجه فاکتورهایی مانند طبیعت سیستم نرم‌افزاری، اندازهٔ سیستم، طبیعت بازار و مشتری مشخص می‌کند که چگونه باید از فرآیند نرم‌افزار، روشها و ابزارهای مهندسی نرم‌افزار استفاده نمود. بدین ترتیب، انتخاب و به کارگیری مدل فرآیند نرم‌افزار در پروژه‌های نرم‌افزاری، امکان سازمان دهی، برنامه‌ریزی، بودجه‌بندی، زمان‌بندی و مدیریت پروژه‌های نرم‌افزاری را فراهم می‌آورد. مدل فرآیند با کیفیت یکی از عوامل دستیابی به محصول با کیفیت است. به همین منظور چارچوبهایی مانند CMM به منظور بهبود فرآیند نرم‌افزار در شرکتهای نرم‌افزار ارائه گردیده اند.

همراه با رشد و تکامل سیستمهای نرم‌افزاری و با توجه به مدل بازار و مشتری، مدلهای فرآیند نیز تکامل یافته اند. در این بین مدلهای توسعهٔ سریع الانتقال مانند XP، FDD یا Scrum نیز موردتوجه قرار گرفته اند. RUP نیز یکی از مدلهایی است که به شکل محصول ارائه می‌گردد. در این چارچوب مراحل، استراتژی توسعهٔ نرم‌افزار، محصولاتی که در هر مرحله باید تولید شوند، قالبهای مستندسازی محصولات هر مرحله، ابزارهایی که باید برای تولید این محصولات استفاده شوند، نقشهایی که مسئول انجام فعالیتهای مختلف هستند، نقاط کنترلی برای کنترل کیفیت محصول و راهنماییهای لازم برای انجام فعالیتهای مختلف ارائه می‌شود. بنابراین، می‌توان از آن به عنوان مدل مناسبی برای توسعهٔ سیستمهای نرم‌افزاری بهره گرفت.

در هر صورت، باید توجه داشت که نباید مدلهای فرآیندی را به شکل کورکورانه دنبال نمود و محصولات بدون استفاده و بی ارزش را براساس آنها تولید کرد. در عوض، باید فرآیند در ضمن این که سبک درنظر گرفته می‌شود، محصولات لازم را به منظور تولید سریع نرم‌افزاری با کیفیت تولید کند. علاوه بر فرآیند باید به شیوه‌های کاری سازمان موردنظر، رویه‌ها و قوانین مشخص به منظور تکمیل فرآیند نیز توجه شود.

مراجع

1- Ambler, S., *Agile Modeling: Effective Practices For Extreme Programming and Unified Process*, New York, John Wiley, 2002.
2- Abmbler, S., Lessons in Agility from Internet-based development, *IEEE Software*, 19(2): 66-73, 2002.
3- Beck, K. *Extreme progamming explained: Embrace change*, Addison-Wesley, 1999.

4- Beck, K., Beedle, M. van Bennekum, A., Cockburn, A., Cunningham, W., Fowler,M., et al., *Manifesto for Agile Software Development* (22.3.2002), agilemanifesto.org, 2002.
5- Beck K., B. Boehm, Agility through Discipline: A Debate, *IEEE Computer* 36:6:47-56, June 2003.
6- Boehm B., A Spiral Model of Software Development and Enhancement, *Proc. Int'l Workshop Software Process and Software Environments*, ACM Press, 1985; also in *ACM Software Eng. Notes*, Aug. 1986, pp. 22-42.
7- Boehm, B. Using the WINWIN Spiral Model: A Case Study, *Computer*, vol. 31, no. 7, July 1998, pp. 33-44.
8- Boehm, B. Get Ready for the Agile Methods, With Care, Computer, 35(1): 64-69, 2002.
9- Butler, J. Rapid Application Development in Action, *Managing System Development*, Applied Computer Research, vol. 14, no. 5, May 1994, pp. 6-8.
10- Coad P. et al., Feature-Driven Development, *Java Modeling in Color with UML*, Prentice Hall,1999.
11- Cockburn A., *Agile Software Development*, Addison- Wesley, 2002.
12- DSDM Consortium, *Dynamic Systems Development Method*, Ashford, 2000.
13- Gilb T., *Software Metrics*, Little, Brown, and Co.,1976.
14- Gilb, T. *Principles of Software Engineering Management*, Addison-Wesley, 1988.
15- Royce W., Managing the Development of Large Software Systems, *Proceedings of Westcon*, IEEE CS Press,1970, pp. 328-339.
16- Gilb T., Evolutionary Development, *ACM Software Eng. Notes*, Apr. 1981, p. 17.
17- Highsmith, J. A. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, New York, Dorest House Publishing, 2000.
18- ITU-T Recommendation M.3010, *Principles for a Telecommunications management network*, 1998
19- ITU-T Recommendation M.3020, *TMN Interface Specification Methodology*,1998.
20- Jacobson, I., G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.
21- Konrad M. M. B. Chrissis, J. Fergusen, Capability Maturity Modeling at the SEI, *Software Process-Improvement and Practice*, March 1996.
22- Krutchen, P. *The Rational Unified Process: An Introduction*, Addison-Wesley, 2000.
23- Krutchen, P. Going Over the Waterfall with the RUP, Rational Edge, e-zine for the rational community, Spetember 2001.
24- Larman, Craig and Basili, Victor R. Iterative and Incremental Development: A Brief History. *IEEE Computer* 36:6:47-56, June 2003.
25- Martin, J. *Rapid Application Development*, Prentice-Hall, 1991.
26- O'Really, T. Lessons from Open Source Software Development, *Communications of the ACM*, vol. 42, No. 4: 32-37, 1999.
27- Palmer, S. R. and Fesling, J. M. *A Practical Guide to Feature-Driven Development*, Prentice-Hall, 2002.

28- Pressman, R. *Software Engineering: A Practitioner's Approach*, Fifth Edition, McGraw-Hill, 2001.
29- Paulk et al., *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, 1995.
30- Schwaber, K. and Beedle M., *Agile Software Development with Scrum*, Prentice-Hall, 2002.
31- Schach, R. S. *Classical and Object Oriented Software Engineering*, McGraw-Hill, Fifth Edition, 2002.
32- Schneider K. J. Hunnius, V. Basili, Experience in Implementing a Learning Software Organization, *IEEE Software*, June 2002.
33- Sommerville, Ian, *Software Engineering*, Sixth Edition, Addison-Wesley, 2000.
34- Software Engineering Institute, *Capability Maturity Model for Software*, www.sei.cmu.edu/cmm.
35- Stapleton, J. *Dynamic Systems Development Method: The method in practice*, Addison Wesley, 1997.

پرسشها

۱- مدل فرآیند نرم‌افزار را تعریف کنید.
۲- نقش مدل فرآیند نرم‌افزار در ساخت سیستمهای نرم‌افزار چیست؟
۳- مزایا و معایب مدل فرآیند آبشاری چیست؟
۴- مدلهای توسعهٔ سریع الانتقال را با تعیین معیارهایی با یکدیگر مقایسه کنید.
۵- مدلهای آبشاری، افزایشی، پیچشی و RAD را با یکدیگر مقایسه کنید.
۶- سیستمی را مثال بزنید که مدل فرآیند RAD برای ساخت آن مناسب باشد. مراحل ساخت سیستم مثال زده شده را بیان کنید.
۷- سیستمی را مثال بزنید که مدل فرآیند افزایشی برای ساخت آن مناسب باشد. مراحل ساخت سیستم مثال زده شده را بیان کنید.
۸- سیستمی را مثال بزنید که مدل فرآیند پیچشی برای ساخت آن مناسب باشد. مراحل ساخت سیستم مثال زده شده را بیان کنید.
۹- (تحقیق) با مطالعهٔ مجلات و مقالات مرتبط با مهندسی نرم‌افزار، گزارشی از سیر تکاملی مدلهای نرم‌افزاری، مسائل روزهای و کارهای آتی در رابطه با این مدلها تهیه کنید.

مطالعه بیشتر

در دائره المعارف مهندسی نرم‌افزار در فصلی با عنوان مدلهای فرآیند در مهندسی نرم‌افزار، از نگاهی دیگر، تعدادی از مدلهای فرآیند در مهندسی نرم‌افزار گروه‌بندی و توصیف می‌شود. در این فصل از

دائره‌المعارف، سعی می‌شود تا بازبینی جامعی برروی بعضی از مدلها و شیوهٔ توسعهٔ سیستم توسط آنها ارائه گردد.

> J.J. Marciniak (ed.), Encyclopedia of Software Engineering, 2nd, Edition, John Wiley and Sons, Inc, New York, December 2001.Walt Scacchi, Process Models in Software Engineering, Institute for Software Research, University of California, Irvine, February 2001,

پیشرفتهای مرتبط با مهندسی نرم‌افزار را می‌توان در مجلاتی مانند IEEE Software و IEEE Computer مطالعه نمود. در واژه شناسی IEEE نشریاتی با عنوان Magazine بیشتر به انتقال حاصل تجارب اهل فن و افراد حرفه‌ای در حوزهٔ مهندسی نرم‌افزار اختصاص داد. در این مجلات گزارش تجارب صورت گرفته برروی فرآیندها، ابزارها، روشها و فعالیتهای مهندسی نرم‌افزار ارائه می‌گردد. در نشریاتی با عنوان Journal بیشتر حاصل تحقیقات پژوهشگران در حوزهٔ مهندسی نرم‌افزار به چاپ می‌رسد. شمارهٔ ذیل از نشریهٔ IEEE Computer اختصاص به بحث در رابطه با روشهای سریع الانتقال دارد:
IEEE Computer 36:6:47-56, June 2003

کتاب زیر در رابطه با مدل سازی سریع الانتقال بحث می‌کند:

Ambler, S., *Agile Modeling: Effective Practices For Extreme Programming and Unified Process*, New York, John Wiley, 2002.

کتاب شناسی در رابطه با مدلهای توسعهٔ سریع الانتقال در نشانی زیر یافت می‌گردد:
http://collaboration.csc.ncsu.edu/agile/Bibliography.htm