QUALITY CHEKD

ویژه کنترل کیفیت و بررسی نرم افزار

روش‌های تولید نرم‌افزار

مقدمه

برای حل مسائل کنونی در صنعت نرم‌افزاری موجود، یک مهندس نرم‌افزار یا مهندسی باید راهبردهای توسعه را که شامل تعیین فرایند، روش‌ها و انواع ابزارهای مورد نیاز می‌باشد، به صورت کارآمد با یکدیگر ترکیب و آمیخته نماید. این راهبرد، بیشتر به‌عنوان یک شیوه فرایند(۱) مطرح می‌شود و روش پردازش برای مهندسی نرم‌افزار، بر پایه پروژه‌های واقعی و برنامه‌های کاربردی انتخاب می‌شود. روش‌ها و ابزارها باید به صورت صحیح مورد استفاده قرار گرفته و کنترل شوند و در زمان مورد نیاز، تحویل داده شوند. روش‌های مختلف تولید نرم‌افزار، الگوهای مختلفی را به‌منظور آسان‌سازی ایجاد و به‌کارگیری روش فرایند پیشنهاد می‌نمایند که در دنیای نرم‌افزار با اصطلاح «متدولوژی‌ها/ روش‌های نرم‌افزاری» شناخته شده‌اند. روش‌های نرم‌افزاری، مدل‌های متفاوت مجتمع‌سازی پروسه تولید نرم‌افزار هستند که هریک، از الگوها، مراحل و نشانه‌های متفاوتی استفاده می‌کنند و بر اساس کاربردهای متفاوت، روش‌های گوناگونی مورد نیاز است که بسته به امکان‌سنجی نیازهای متعدد پروژه، ترجیهات مشتری یا دیدگاه‌های مدیریتی مختلف، می‌توان یکی از این روش‌ها و یا ترکیبی از این الگوها را در فرایند تولید نرم‌افزار به‌کارگرفت. در این نوشتار، به شرح ویژگی‌های برتر و مهم‌ترین متدولوژی یا روش‌های نرم‌افزاریِ مطرح می‌پردازیم.

کلیدواژگان: تولید نرم‌افزار، تحلیل نرم‌افزار، توسعه نرم‌افزار.

روش‌های نرم‌افزاری

به منظور مجتمع‌سازی مراحل و گام‌های تولید محصول نرم‌افزاری در هر سازمانی، الگوها از آغاز صنعت نرم‌افزار تا امروز تحت تنوع عملکرد و بر اساس نیازمندی‌ها و سبک تفکر صاحب‌نظران مختلف این عرصه بوده است. روش‌های نرم‌افزاری، شامل موارد زیر است:

۱. ترتیبی خطی یا آبشاری؛

۲. پیش‌نمونه‌سازی؛

۳. تکامل‌یافته؛

۴. افزایشی؛

۵. تولید سریع کاربردها؛

۶. الگوی توسعه تجزیه تحلیل سیستم‌های ساخت‌یافته؛

۷. حلزونی؛

۸. مارپیچی برنده برنده؛

۹. مونتاژمؤلفه‌ها؛

۱۰. توسعه هم‌روند؛

۱۱. روش‌های رسمی ؛

۱۲. فنون نسل چهارم؛

۱۳. روش PX؛

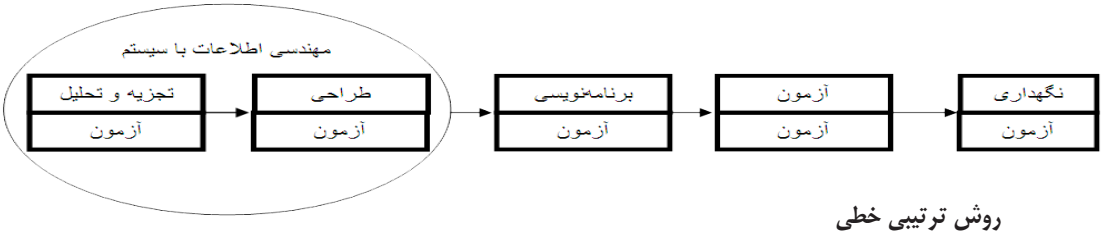
۱۴. روش MMC.

در ادامه، به شرح گام‌ها و مراحل هریک از این روش‌ها و نیز کاربردها و تفاوت‌های الگوهای ساخت محصول نرم‌افزاری می‌پردازیم.

۱. ترتیبی خطی یا آبشاری(۲)

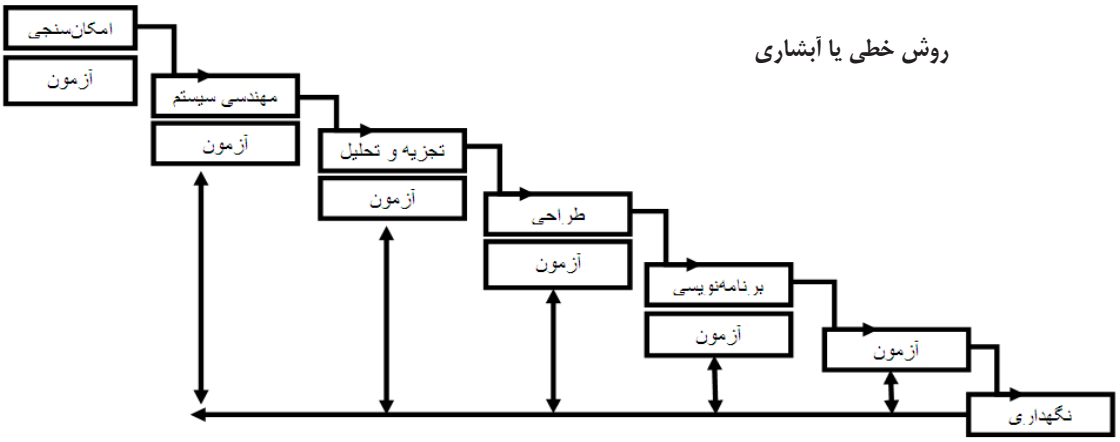
این روش با نام‌های دیگری چون چرخه حیات کلاسیک و یا روش آبشاری نیز مطرح گردیده، یکی از قدیمی‌ترین و پرکاربدترین الگوها برای مهندسی نرم‌افزار است. از مهم‌ترین محاسن این روش، سادگی و شفافیت مراحل انجام کار و فازها می‌باشد و در واقع، این الگو مدل مرجع و پایه‌ای برای روش‌های بعدی بود؛ چراکه روش‌های بعدی غالباً این الگو را مدنظر قرار داده و سعی در برطرف‌نمودن معایب این روش پایه داشته‌اند. از جمله معایب این روش، آن است که بنا به ذات خطی و سریال این الگو، تغییرات را نمی‌توان به صورت مستقیم به آن اعمال نمود. بنابراین، برای خواسته‌های متغیر کاربران، الگوی مناسبی نمی‌باشد. همچنین، در این الگو هیچ نمونه کاری شبیه‌سازی‌شده‌ای تا انتهای کار برای کاربر یا مشتری ایجاد نمی‌گردد. پس، ممکن است محصول خروجی، با خواسته‌های مشتری مغایر باشد و به بازگشت به عقب(۳) و دوباره کاری نیاز باشد.

دیاگرام و مراحل کار، بسته به اجرای خطی یا آبشاری این الگو، به دو صورت ذیل ممکن می‌باشد:



روش ترتیبی خطی

روش خطی یا آبشاری



این روش، شیوه‌ای سازمان‌یافته ترتیبی برای ایجاد نرم‌افزار پیشنهاد می‌کند که از سطح سیستم شروع می‌کند و به سطح طراحی، کد، تست و نگهداری حرکت می‌نماید. مراحل و فازهای مختلف این الگو، به شرح ذیل است:

– مهندسی اطلاعات یا سیستم(٤):

به‌دلیل اینکه نرم‌افزار، بخشی از یک سیستم بزرگ‌تر (یک تجارت) است، همیشه کار با ایجاد پیش‌نیاز برای تمام المان‌های سیستم و سپس نسبت‌دادن زیرمجموعه‌ای از این نیازها به نرم‌افزار شروع می‌شود. چنین دیدی از سیستم، لازم است؛ چراکه نرم‌افزار باید با المان‌هایی مثل سخت‌افزار، مردم و پایگاه داده ارتباط داشته باشد.

مهندسی سیستم و تجزیه و تحلیل، شامل جمع‌آوری نیازها در سطح سیستم – به همراه تجزیه و تحلیل – و طراحی در سطح بالا می‌باشد. مهندسی اطلاعات، شامل جمع‌آوری نیازها در سطح تجارت راهبردی و محیط تجاری می‌باشد.

– تجزیه تحلیل نیازمندی‌های نرم‌افزار(٥):

پروسه جمع‌آوری نیازها، روی جمع‌آوری و امکان‌سنجی خواسته‌ها و نیازهای نرم‌افزار تأکید می‌کند. برای اینکه ماهیت مسئله مشخص شود، مهندس نرم‌افزار یا تحلیلگر باید محدوده اطلاعات پردازشی نرم‌افزار، توابع مورد نیاز، کارکردها، کارآیی و واسطها را به‌خوبی بشناسد. در این فاز، نیازها برای سیستم و نرم‌افزار، مستندسازی شده، با کمک مشتری بازبینی می‌شود.

– طراحی(٦):

طراحی نرم‌افزار، پروسه‌ای چند مرحله‌ای است که بر چهار ویژگی مجزای مسئله تأکید دارد:

* ساختمان داده‌ها؛
* معماری نرم‌افزار(٧)؛
* نمایش واسطها؛
* جزئیات رویه‌ای.

– تولید و توسعه کد(٨):

طراحی باید به فرم قابل پذیرش و خواندن توسط ماشین تبدیل شود. این فاز، در واقع، این‌کار را انجام می‌دهد. اگر طراحی با جزئیات کامل باشد، تولید کد به‌راحتی توسط ابزارهای تولید کد که بنا به نوع کاربردها تنوع بسیاری دارند، انجام می‌شود.

– آزمایش(۹):

هنگامی‌که کد تولید شد، تست برنامه آغاز می‌شود. فرایند تست، بر منطق درون نرم‌افزار تأکید دارد؛ البته با توجه به اینکه تمام دستورها تست شده‌اند. در این مرحله، با دادن ورودی تعریف‌شده به سیستم، نتایج واقعی حاصل می‌گردد که باید با نتایج مورد نظر تطابق داشته باشند.

– نگهداری(۱۰):

بی‌شک، بیشتر نرم‌افزارها، بعد از اینکه تحویل مشتری داده می‌شوند، دست‌خوش تغییرات مختلف قرار می‌گیرند و تغییرات نیز معمولاً به دلیل خطاها به‌وجود می‌آیند؛ زیرا نرم‌افزار باید با تغییرات موجود در محیط سازگار شود و مشتری نیز به یک کارکرد دیگر و یا کارایی بیشتر نیاز داشته باشد. فاز نگهداری نرم‌افزار، به جای شروع از اوّل، هریک از فازهای قبلی را روی برنامه موجود اعمال می‌کند.

روش آبشاری، با وجود اینکه جزء مهم‌ترین الگوهای زیربنایی ساخت محصول نرم‌افزاری می‌باشد، دارای مشکلات و نقاط ضعف زیاد است. مهم‌ترین اشکالات روش خطی یا آبشاری، شامل موارد ذیل می‌باشد:

۱. پروژه‌های واقعی، به دلیل ماهیت متغیر و پویای بستر پروژه نرم‌افزاری، معمولاً جریان ترتیبی را که این روش پیشنهاد می‌نماید، دنبال نمی‌کنند.

۲. برای مشتری مشکل است که به طور دقیق و یکجا تمام نیازهای خود را در ابتدای کار و آغاز پروژه بگوید.

۳. طبق الگوی آبشاری، مشتری باید برای دیدن نتایج نهایی، تا انتهای کار صبر داشته باشد؛ زیرا نسخه‌ای از برنامه که کار می‌کند، اواخر پروژه در دسترس خواهد بود و اگر یک اشکال تا زمان تولید برنامه نهایی دیده نشود و یا نظر مشتری اعمال نگردد، می‌تواند مصیبت‌بار باشد.

۴. ماهیت خطی این روش، باعث به‌وجود آمدن blocking states می‌شود که در این حالت، بعضی از اعضای تیم باید منتظر کامل‌شدن کار دیگران شوند. در واقع، زمان انتظار در این روش، بیشتر از زمان کار است. زمان انتظار، به‌خصوص در ابتدا و انتهای این روش، زیاد می‌شود.

۵. در این روش، زمان بسیاری صرف کار می‌شود که خود باعث تغییر نیاز مشتری می‌شود.

۶. در روش آبشاری یا خطی، برگشت به عقب و بازبینی و تصحیح عملکردها در مراحل مختلف، زمـــان‌بر و هزینه‌بر است.

به هر حال، این روش جایگاه مهم و مشخصی در مهندسی نرم‌افزار دارد و در واقع، به‌عنوان یک الگو و زیربنا برای روش‌های دیگر مطرح است. با وجود نقطه ضعف‌های این روش، بهتر از روش‌های بدون برآورد و برنامه‌ریزی و تصادفی است. مهم‌ترین مستندات تولیدشده در این روش، موارد ذیل است:

– مستند مشخصات(۱۱)؛

– مستند طراحی(۱۲)؛

– مستند کدنویسی(۱۳)؛

– مستند کاربر(۱۴)؛

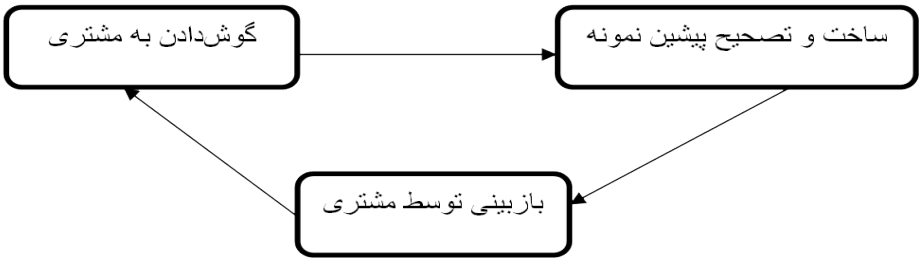
– مستند عملکرد برنامه(۱۵)؛

– نمونه‌های آزمایش(۱۶).

۲. پیش‌نمونه‌سازی(۱۷)

در پروژه‌های واقعی، معمولاً مشتری مجموعه‌ای از اهداف کلی خود را برای یک نرم‌افزار تعریف

می‌کند؛ ولی جزئیات ورودی‌ها، پردازش‌ها و خروجی‌های مورد نیاز را مشخص نمی‌کند و یا غالباً نیازمندی‌های مشتری متغیر است. از طرف دیگر، تولیدکننده نرم‌افزار نیز از کارایی الگوریتم، سازگاری با سیستم عامل یا شکلی که باید تعامل بین ماشین و انسان را اعمال کند، اطمینان ندارد. در این حالت، روش پیش‌نمونه‌سازی، بهترین روش است. بنابراین، این روش در مواردی که ابتدای کار از جزئیات خواسته‌های مشتری و یا جزئیات ریز طراحی اطلاعی نداریم، مناسب است. شکل ذیل، روند کار را در روش نمونه‌سازی نشان می‌دهد:



روش پیش‌نمونه‌سازی

این روش، با مرحله جمع‌آوری نیازمندی‌های مشتری آغاز می‌شود. تولیدکننده نرم‌افزار مشتری، در جلسات خود اهداف کلی برای نرم‌افزار را تعریف می‌کند، نیازمندی‌های شناخته‌شده را مشخص می‌نماید و مواردی را که نیاز به تعریف بیشتر دارند، معیّن می‌کند. سپس، یک طراحی سریع اتفاق می‌افتد. طرح تولیدشده بر شیوه ارائه موضوعات از نرم‌افزار که توسط مشتری/ کاربر قابل مشاهده است، مثل روش‌های ورود اطلاع و فرمت خروجی، تأکید دارد.

طرح سریع تولیدشده، منجر به تولید یک نمونه می‌شود. نمونه تولیدشده توسط مشتری ارزیابی می‌شود تا نیازهای نرم‌افزار برای گسترش آن، بهبود پیدا کند. حلقه تولیدشده تا زمانی‌که نیازهای مشتری را برآورده کند، ادامه می‌یابد و در هر دور، تولیدکننده نرم‌افزار از آنچه باید انجام دهد، نگرش بهتری پیدا می‌کند.

به طور ایده آل، این روش به‌عنوان مکانیزم شناخت نیازمندی‌های نرم‌افزار عمل می‌کند. اگر نمونه قابل استفاده تولید شود، تولیدکننده نرم‌افزار سعی در استفاده از بخش‌های برنامه موجود می‌کند و یا از ابزاری مثل تولیدکننده گزارش(۱۸) و مدیر پنجره‌ها(۱۹) که به تولید سریع نرم‌افزار کمک می‌نماید، استفاده می‌کند.

حال، سؤال مهم این است که از نمونه ساخته‌شده چه استفاده‌ای کنیم؟ در بسیاری پروژه‌ها، اوّلین سیستم تولیدی به‌سختی قابل استفاده است و ممکن است، خیلی کند، بزرگ و بدقواره باشد. ازاین‌رو، راهی جز شروع دیگر و ساختن نسخه دوباره طراحی‌شده که در آن این مشکلات برطرف گردیده، نیست. وقتی یک مفهوم سیستمی جدید، با فناوری جدید آمد، باید سیستم تولیدشده را دور بریزیم. پیش‌نمونه می‌تواند به‌عنوان سیستم اوّلیه کار کند. مشخص است که این روش، هم برای مشتری و هم برای تولیدکننده، خوش‌آیند است؛ زیرا کاربران، سیستم واقعی را حس می‌کنند و تولیدکنندگان سریع چیزی می‌سازند؛ اما این روش نیز دارای مشکلات خاصی می‌باشد. مهم‌ترین مشکلات روش پیش‌نمونه‌سازی، شامل موارد زیر است:

۱. مشتری آنچه را می‌بیند، به‌عنوان نسخه‌ای از سیستم که کار می‌کند، در نظر می‌گیرد؛ غافل از اینکه در این روند سریع، کیفیت و قابلیت نگهداری و برخی دیگر از مؤلفه‌های کلیدی، دیده نشده است و

مشتری می‌خواهد با تغییراتی کوچک، نمونه ساخته‌شده، به محصول کارا تبدیل شود. در بیشتر موارد، تولیدکننده از انجام کار پشیمان می‌شود.

۲. تولیدکننده نرم‌افزار گاهی برای رسیدن سریع به یک نمونه قابل استفاده، از برخی روش‌های پیاده‌سازی استفاده می‌کند که در آینده مشکل‌آفرین است؛ مثل سیستم عامل نامناسب، زبان برنامه‌نویسی ساده که در دسترس و شناخته شده است و یا الگوریتم غیرکارا که بتواند فقط قابلیت ارائه یک نمونه را داشته باشد. بعد از گذشت زمان، تولیدکننده نرم‌افزار شناخت بیشتری نسبت به این موارد پیدا می‌کند و دلایل نامناسب‌بودن موارد فوق را از یاد می‌برد و این اجزا، به صورت اجزای بنیادی سیستم باقی می‌مانند.

۳. به دلیل دیده‌نشدن نکات کیفیتی و قابلیت‌های افزوده در ابتدای کار، در این روش همواره دچار دوباره‌کاری و اتلاف زمان خواهیم بود.

باوجود مشکلات فوق، این روش به‌عنوان یک الگوی مؤثر در مهندسی نرم‌افزار است. نکته کلیدی، این است که در ابتدا باید قوانین بازی را تعریف کنیم. مشتری و تولیدکننده باید بر سر اینکه پیش‌نمونه‌ها، تنها مکانیزمی برای جمع‌آوری نیازمندی‌ها است و بعد دور ریخته می‌شود، توافق کنند.

۳. تکامل‌یافته(۲۰)

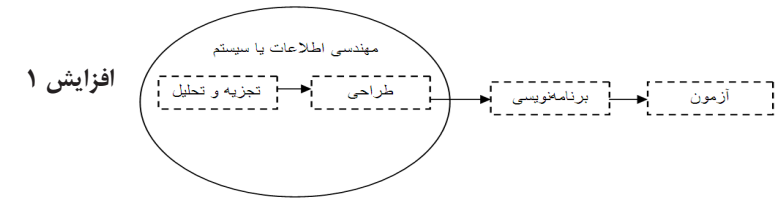
نرم‌افزار، مانند سیستم‌های پیچیده، با گذشت زمان تکامل می‌یابد. تجارت و نیازمندی‌های محصول، غالباً تغییر می‌کند. موعدهای تحویل به‌هم‌فشرده، تکمیل یک نرم‌افزار جامع را غیرممکن می‌کند؛ اما یک نسخه محدود از محصول برای ارائه در مقابل فشارهای تجاری نیاز است. در این حالت، مجموعه‌ای از نیازهای اصلی و ریشه‌ای محصول به‌خوبی شناخته می‌شوند و جزئیات محصول و یا توسعه سیستم در آینده تعریف می‌شود. در چنین حالاتی، مهندسان نرم‌افزار نیاز به روش‌هایی دارند که بتواند با محصولاتی که با گذشت زمان متحول می‌شوند، سازگار باشند.

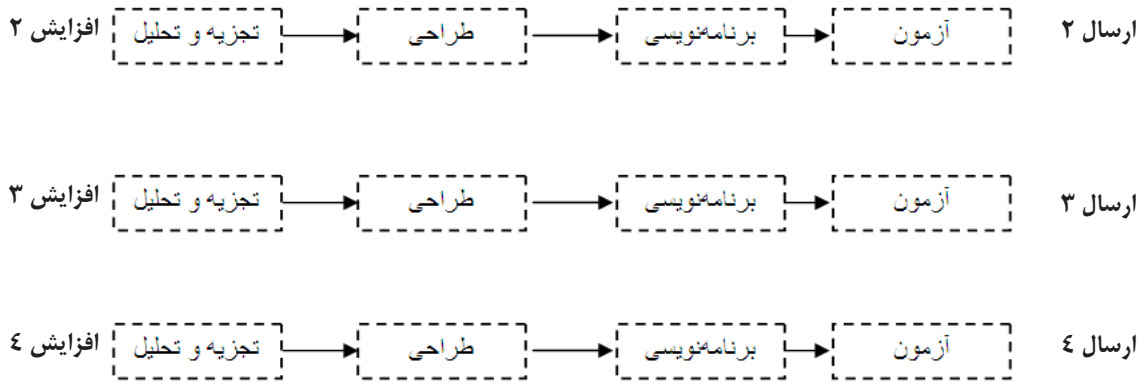
روش ترتیبی خطی، برای تولید خطی طراحی شده است. در این روش، فرض بر این است که محصول نهایی پس از اتمام ترتیب خطی، ارائه می‌شود. روش پیش‌نمونه‌سازی به‌عنوان یک دستیار برای مشتری و تولیدکننده است تا نیازها به‌خوبی شناخته شوند. در کل، این روش برای تحویل یک محصول تولیدی طراحی نشده است. ذات تکامل‌پذیر نرم‌افزار، در این گونه روش‌های مهندسی نرم‌افزار کلاسیک، در نظر گرفته نشده است.

روش‌های تکامل‌پذیر و مهندس (Iterative) روش‌های تکامل‌پذیر، نرم‌افزار را قادر می‌سازد که به طور روزافزون نسخه‌های کامل‌تری را توسعه بدهند. روش‌هایی که در ادامه ذکر می‌شوند، از جمله شیوه‌های تکامل‌پذیر می‌باشند؛ چراکه نرم‌افزار و محصول نهایی را در قالب نسخه‌های قابل ارتقا و بهینه‌سازی ارائه می‌کنند.

٤. افزایشی(۲۱)

این روش، عناصر و المان‌های روش ترتیبی خطی را با فلسفه تکراری روش پیش‌نمونه‌سازی ترکیب می‌کند. شکل ذیل روند کار این روش را نمایش می‌دهد:





نمای کلی روش افزایشی

هر ترتیب خطی، یک رشد قابل تحویل از نرم‌افزار را تولید می‌کند؛ برای نمونه، نرم‌افزار پردازش کلمه در مرحله اوّل مدیریت فایل پایه، ویرایش و توابع تولید مستندات را تولید می‌کند. ویرایش پیشرفته‌تر و قابلیت تولید مستندات در مرحله دوم، چک‌کردن گرامر و املا در مرحله آخر، و قابلیت لایه صفحات پیشرفته در مرحله چهارم، به‌عنوان روال‌های کلیدی به محصول افزوده می‌گردد.

این نکته قابل توجه است که جریان پردازش برای هر مرحله می‌تواند به روش پیش‌نمونه‌سازی ملحق شود. وقتی روش افزایشی استفاده می‌شود، مرحله اوّل تحویل، معمولاً هسته محصول(۲۲) است. در این مرحله، نیازهای پایه مطرح می‌شوند؛ اما بسیاری از ویژگی‌های (شناخته شده یا نشده) برای آینده باقی می‌ماند و حتی محصول توسط مشتری استفاده می‌شود. در حین اینکه هسته محصول توسط مشتری استفاده و ارزیابی می‌شود، طرح و برنامه مرحله بعد ریخته می‌شود. طرح بعدی، تغییرات هسته محصول را برای رسیدن به نیازهای مشتری به همراه کارآیی و ویژگی‌های بیشتر دربردارد. این روند، در هر مرحله تکرار می‌شود تا محصول نهایی حاصل گردد. این روش، شبیه پیش‌نمونه‌سازی و روش‌های دیگر تکامل‌یافته، ذاتاً تکرارپذیر است؛ ولی برخلاف پیش‌نمونه‌سازی، این روش تأکید بر تحویل محصولات عملیاتی در هر مرحله دارد.

بنابراین، نسخه‌های قبلی دور ریخته نمی‌شوند. مراحل اوّلیه، نسخه‌های عریان محصول نهایی هستند؛ اما آنها قابلیت ارائه چارچوبی برای ارزیابی توسط کاربر را دارند. این روش، برای حالتی که پرسنل کافی برای اتمام پروژه در زمان تعیین‌شده کافی نباشد، مناسب است. مراحل اوّلیه می‌تواند با پرسنل کمتری صورت بگیرد. اگر هسته محصول به صورت کامل و خوب دریافت شده باشد، می‌توان پرسنل اضافه‌ای برای انجام مراحلِ بعدی به خدمت گرفت. علاوه بر این، مراحل این روش می‌تواند به صورتی طراحی شود که بتوان خطرهای فنی(۲۳) را در نظر گرفت؛ به‌عنوان مثال، یک سیستم کلان ممکن است نیاز به سخت‌افزاری جدید داشته باشد که در حال ساخت است و زمان عرضه آن، مشخص نیست. بنابراین، باید بتوان مراحل اوّلیه را طوری طراحی کرد که از استفاده این سخت‌افزار، به دور باشند و در نتیجه، می‌توان بدون تأخیر نامنظم، خروجی با کارکرد جزئی برای ارائه به کاربر داشته باشیم. باید توجه داشت که در این روش، مجبور به انجام دوباره بعضی کارها هستیم که از نقاط ضعف آن به شمار می‌رود؛ ولی به هر حال، این روش برای بازار رقابتی گزینه مناسبی می‌باشد.

۵. روش توسعه کاربردها(۲٤)

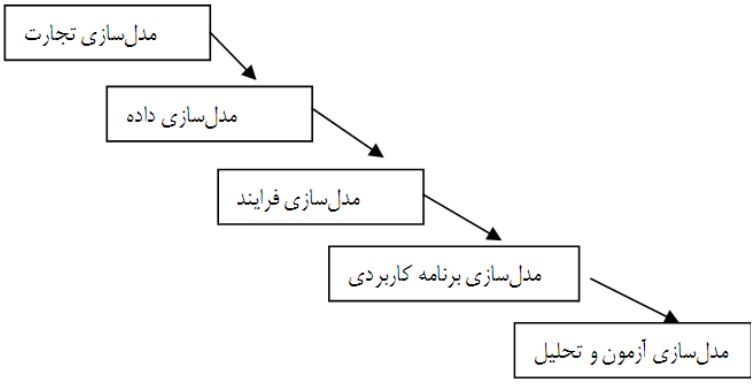
یک روش پردازش ترتیبی خطی است که تأکید بر چرخه تولید کوتاه دارد. این روش، یک انطباق از

روش ترتیبی خطی است که با شیوه مبتنی بر مؤلفه(۲۵)، به تولید سریع می‌رسد. اگر نیازمندی‌ها خوب شناخته شود و محدوده کار درست فرض شود، پروسه RAD، تیم توسعه را به ساختن یک سیستم کامل با کارایی بالا و زمان بسیار کم تولید (معادل ۶۰ ـ ۹۰ روز کاری) قادر می‌سازد. در درجه اوّل، این شیوه در سیستم‌های اطلاعاتی استفاده می‌شود. از جمله محاسن کلیدی این الگو، تفکیک انواع فازها و رده‌های مدل‌سازی در گروه‌های مستقل است.

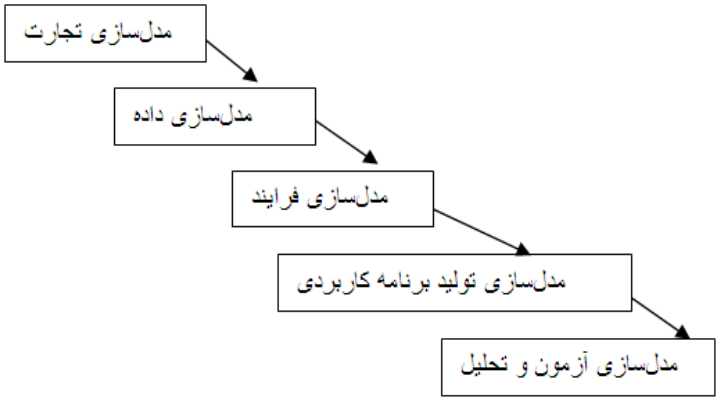
بنابراین، تیم کاری در هر گروه می‌تواند به صورت مستقل عمل کند و در نهایت، می‌توان انواع فرآوردها و محصولات تیم‌های کاری مختلف را مجتمع‌سازی و یکپارچه نمود. در واقع، همین استقلال عملکرد، باعث افزایش سرعت چشمگیر توسعه و نام‌گذاری این روش به الگوی تولید سریع کاربردها گردیده است.

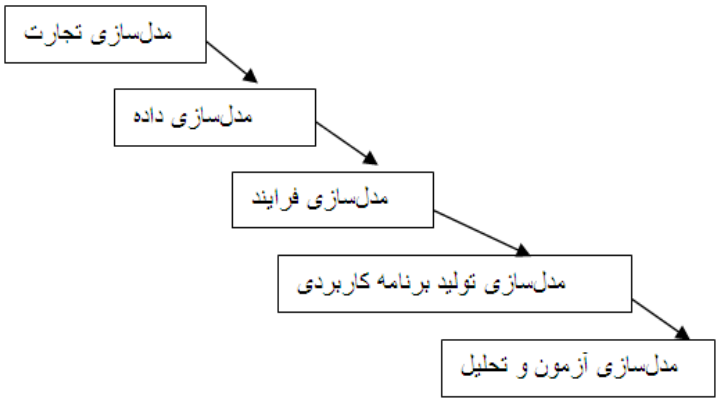
انواع طبقات مدل‌سازی و فازهای این روش در شکل ذیل آمده است:

تیم ۳



تیم ۲





روش توسعه سریع کاربردها

شرح مراحل کار روش توسعه سریع کاربردها، مطابق موارد زیر می‌باشد:

- مدل‌سازی تجاری(۲۶):

در این الگو، فرآیند مدل‌سازی تجارت یا کسب‌وکار مورد نظر، با جواب‌دادن به سؤالات زیر، جریان داده‌ها در عملکرد تجاری خاصی را مدل‌سازی می‌کند:

* چه اطلاعاتی فرآیندهای تجاری را راه‌اندازی می‌کند؟
* چه اطلاعاتی تولید می‌شود؟
* چه کسی این اطلاعات را تولید می‌کند؟
* این اطلاعات کجا می‌روند و چه خروجی ایجاد می‌کنند؟
* چه کسی این اطلاعات را پردازش می‌کند؟

- مدل‌سازی داده‌ای(۲۷)

جریان داده که به‌عنوان بخشی از فاز قبلی تعریف شد، در مجموعه‌ای از اشیای داده‌ای که برای پشتیبانی از تجارت مورد نیاز هستند، جای می‌گیرد. در این مرحله، خصیصه‌ها و صفات هر شیء مشخص می‌شود و رابطه بین این اشیا تعریف می‌شود.

- مدل‌سازی فرآیند(۲۸):

در این فاز، پردازش‌ها و روال‌های کاری برای اضافه‌کردن، تغییر، حذف و یا بازیابی اشیای داده، تحت کارکردهای خاصی تعریف می‌شوند.

- تولید برنامه کاربردی(۲۹):

در این روش، تولید برنامه‌های کاربردی با استفاده از تکنیک‌های نسل چهارم (GT4) انجام می‌شود. به‌جای ساختن نرم‌افزار با استفاده از زبان‌های برنامه‌نویسی سنّتی، اگر ممکن باشد، RAD از اجزا و مؤلفه‌ها(۳۰) و بخش‌های موجود استفاده می‌کند و یا اگر لازم باشد اجزای قابل استفاده مجدد تولید می‌کند. در تمام حالات، ابزار خودکار برای سرعت‌بخشیدن به تولید نرم‌افزار کاربردی استفاده می‌شود.

- دوره آزمون(۳۱):

به‌دلیل اینکه روش توسعه سریع کاربردها بر اجزای قابل استفاده مجددِ تست شده تأکید دارد، بنابراین، زمان تست نرم‌افزارهای تولیدی در این روش کاهش می‌یابد. البته اجزای جدید نیاز به تست دارند و در نهایت، به تست جامع کل نرم‌افزار به صورت یکپارچه نیاز داریم.

اگر یک محصول تجاری را بتوان به روش فوق به گونه‌ای مهیا نمود که ساختن عملیات و کارکردهای اصلی محصول در عرض کمتر از سه ماه امکان‌پذیر باشد، این محصول، کاندید خوبی برای این مدل است.

بنابراین، می‌توان هر عمل اصلی را به یک تیم RAD سپرد و در نهایت، نتایج کار را به هم متصل و مجتمع‌سازی نمود.

با وجود ویژگی‌های مثبت الگوی توسعه سریع کاربردها، در این روش نیز تولیدکنندگان همواره با مشکلات و نقاط ضعفی مواجه می‌باشند. مهم‌ترین مشکلات الگوی توسعه سریع کاربردها، شامل موارد زیر است:

– برای پروژه‌های بزرگ، ولی قابل گسترش(۳۲) ، این روش نیاز به منابع انسانی کافی دارد تا تیم‌های RAD را به تعداد ایجاد کند و نیز منابع انسانی باید دارای تعهد کاری باشند. این روش، تولیدکننده و مشتری را در یک چارچوب زمانی کوتاه، محدود می‌کند. اگر پروژه در این فاصله تمام نشود، پروژه RAD از بین می‌رود.

– تمام انواع پروژه‌ها، برای RAD مناسب نیستند. اگر یک سیستم را نتوان به صورت مناسب به پیمانه‌هایی مستقل تقسیم کرد، ساختن اجزای ضروری برای RAD مشکل‌ساز خواهد بود.

– اگر کارایی بالا مدنظر باشد و بخواهیم این کارایی را از طریق تنظیم واسطه‌های اجزای سیستم(۳۳) به دست آوریم، روش به صورت مناسب کار نخواهد کرد.

– RAD هنگامی که احتمال خطرات فنی بالا باشد، مناسب نیست. این امر زمانی اتفاق می‌افتد که یک برنامه به یک فناوری جدید وابستگی داشته باشد و یا یک نرم‌افزار جدید نیاز به درجه بالایی از کنش‌پذیری(۳۴) برنامه‌های رایانه‌ای موجود داشته باشد. در چنین مواردی، به‌کارگیری الگوی مدل‌سازی باعث کاهش کنش‌پذیری متقابل بین اجزای نرم‌افزاری و ماژول‌ها و بسته نرم‌افزاری می‌گردد و تعاملات را دشوار می‌سازد. در این گونه موارد، غالباً به‌کارگیری روش‌ها شیئ‌گرا و مبتنی بر مؤلفه، احتمال موفقیت را افزایش می‌دهد.

٦. الگوی توسعه تجزیه تحلیل سیستم‌های ساخت‌یافته(۳۵)

این الگو، مجموعه‌ای از استانداردها و آموزش‌ها برای طراحی سیستم‌های رایانه‌ای است. این استانداردها و آموزش‌ها، شامل استانداردهای ساختاری(۳۶) می‌شود که ساختار یک پروژه گسترش‌یافته را به شکل مجموعه‌ای صریح از کارهای تعریف‌شده با مجموعه آشکار از رابطه‌های بین آنها و محصول قابل لمس، مشخص می‌کند. همچنین، استانداردها وآموزش‌های فنی برای مجموعه پرسنل تکنیک‌های

گام‌ها و مراحل روش توسعه تجزیه تحلیل سیستم‌های ساخت‌یافته

قابل استفاده اثبات‌شده و ابزارها و قوانین مشخص و آموزش زمان و چگونگی استفاده از آنها را فراهم می‌کند. استانداردهای مستندسازی(۳۷) علاوه بر این، ذخیره محصولات حاصل از فعالیت‌های توسعه را با جزئیات آنها، مهیا می‌نماید. مهم‌ترین مراحل و گام‌های الگوی توسعه تجزیه تحلیل سیستم‌های ساخت‌یافته، در شکل صفحه قبل نمایش داده شده است.

در ادامه، به شرح جزئیات عملیات و کارکردها در هر گام از الگوی توسعه تجزیه تحلیل سیستم‌های ساخت‌یافته می‌پردازیم:

- امکان‌سنجی(۳۸):

امکان‌سنجی، نوعی ارزیابی کوتاه و مختصر از سیستم اطلاعاتی پیشنهادی است؛ برای مشخص‌کردن اینکه آیا سیستم می‌تواند نیازمندی‌های مشخص تجاری مربوط به ارگان را بپوشاند و آیا یک نمونه تجاری(۳۹) برای گسترش چنین سیستمی وجود دارد یا نه.

- بررسی محیط موجود (سیستم جاری)(۴۰):

این مرحله، شروع مطالعه کامل است و شامل: شروع برنامه‌ریزی پروژه، بررسی مستندات از مرحله امکان‌سنجی، مطالعه راهبرد سیستمی اطلاعات و شروع مستندسازی پروژه است. جزئیات محیط موجود، شامل: ناحیه تجاری(۴۱) و اهداف آن و کاربران درگیر، تحلیل می‌شوند و در مدل فعالیت تجاری(۴۲) ذخیره می‌گردند. همچنین، روش جریان داده برای تمام توابع به هنگام‌سازی و توابع مهم بازیابی اطلاعات، و روش منطقی داده برای تمام داده‌های جاری بررسی می‌شود.

هم‌زمان با این کارها، نیازمندی‌های سیستم پیشنهادی شامل: سرویس‌های جاری و نیازمندی‌های مورد نیاز برای حل مشکلات کنونی، و همچنین محدودیت‌های پروژه، تحلیل و مستندسازی می‌شوند.

- گزینه‌های سیستم تجاری(۴۳):

در خلال این فاز، گزینه‌های ممکن برای سیستم به همراه مجموعه متفاوتی از توابع پیشنهادی و واسطه‌های مختلف با سیستم‌های دیگر (دستی یا رایانه‌ای) مشخص می‌شوند. برای هر گزینه، آثار آن برای سازمان، هزینه، مزایا و دیگر مفروض‌ها مانند محدودیت‌های فنی، تحلیل می‌شود.

- تعریف نیازمندی‌ها:

بر اساس گزینه انتخاب‌شده، مشخصات نیازمندی‌ها تولید می‌شود. مدل‌های سیستم جاری مانند مدل جریان داده منطقی، تغییر داده می‌شوند تا با توابع و نیازمندی‌های داده‌ای سیستم پیشنهادی یکپارچه گردند. بعد از تأیید نیازمندی‌های سیستم مورد نیاز با کمک کاربران، توابع از مدل جریان داده و دیگر توابع بازیابی مورد نیاز با هم یکپارچه شده، در مستندات تعریف توابع با جزئیات تشریح می‌شوند. در خلال فاز سوم، می‌توان قسمت‌هایی از سیستم پیشنهادی را به صورت پیش‌نمونه‌هایی ساخت تا نیازمندی‌ها به صورت جزئی‌تر از این طریق مشخص گردند. همچنین، وقایع تجاری(۴۴) با جزئیات تشریح می‌شوند و همراه با توابع و موجودیت‌های مرتبط با آنها دسته‌بندی می‌شوند.

- تحلیل گزینه‌های فنی سیستم(۴۵):

تحلیل محیط فنی ممکن، می‌تواند هم‌زمان با فازهای ۳و۵ اتفاق بیفتد. وقتی گزینه‌های فنی سیستم را برای سیستم مورد نیاز تعریف می‌کنیم، مستندات شامل اطلاعات برنامه‌ریزی و استانداردهای داخل سازمان برای نصب و محیط فنی باید مطالعه شود. برای هر گزینه، روش فنی تعریف‌شده، اثر آن بر سازمان، هزینه و مزایا و دیگر مفروض‌های مرتبط، باید تحلیل شود تا کاربران را در انتخاب یک گزینه خوب کمک کند. سپس، گزینه فنی انتخاب‌شده برای مشخص‌کردن جزئیات محیط فنی، اهداف طراحی سیستم و برنامه‌ریزی برای پیاده‌سازی محیط هدف، گسترش می‌یابد.

– طراحی منطقی(٤٦):

مهم‌ترین اهداف این فاز، طراحی دیالوگ برای توابعی که مرتبط با فعل و انفعالات کاربر است، بازیابی توابع و به هنگام‌سازی آنها و تبدیل این توابع به مدل‌های پردازشی مرتبط می‌باشد.

هنگامی‌که مدل‌های پردازشی تعریف می‌شوند، اهداف طراحی مانند: ماژولاریتی، استفاده اشتراکی، حداکثر پیوستگی و حداقل هم‌بندی(۴۷) باید در نظر گرفته شود؛ به این معنا که پیمانه‌های مستقلی تعریف شود که کمترین میزان وابستگی و پیچیدگی واسطها را داشته و هریک از این ماژول‌ها به صورت مستقل عمل نموده، کمترین نیاز را به برقراری ارتباط با سایر ماژول‌ها داشته باشد. همچنین، شامل واسطهای ساده‌ای باشند که از لحاظ ابعاد داده‌ای، ساختاری، زمانی و ساختمان داده‌ای، کمترین ارجاع و وابستگی را به سایر واسطها و پیمانه‌ها داشته باشند.

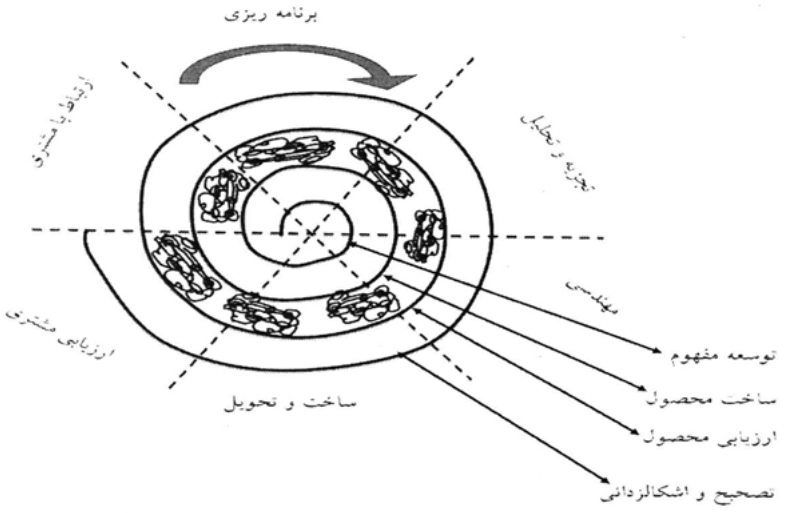
– طراحی فیزیکی(٤٨):

طراحی فیزاکی توسط یادگیری قوانین محیط پیاده‌سازی که باید در راهنمای مشخصه محصول توسط تهیه‌کننده قرار گیرد، صورت می‌پذیرد. همچنین، لازم است عملیات دیگری از قبیل تکمیل مشخصات توابع و بهینه‌سازی طراحی داده و پردازش و تست آنها توسط اهداف زمانی و اندازه‌ای که در اهداف طراحی وجود دارد نیز انجام شود.

۷. روش حلزونی(٤٩)

این روش کارا و رایج که توسط BOEHM پیشنهاد شده، شیوه‌ای از توسعه پردازش نرم‌افزار تکاملی است که ذات تکرارپذیر مدل پیش‌نمونه‌سازی را با چهره کنترل‌شده و منظم مدل ترتیبی خطی، تلفیق می‌کند. این روش توسعه سریع، برای نسخه‌های افزایشی نرم‌افزار را فراهم می‌کند. در این روش، نرم‌افزار به صورت مجموعه از کارکردهای تحویلی افزایشی(۵۰) توسعه می‌یابد. در خلال دوره‌های اوّلیه، نسخه‌ها به صورت یک مدل کاغذی یا پیش‌نمونه است. در دوره‌های بعدی، نسخه‌های کامل‌تری از سیستم مهندسی تولید می‌شود.

این روش، به مجموعه فعالیت‌هایی تقسیم می‌شود که به آنها «ناحیه‌های کاری»(۵۱) می‌گوییم و بین سه تا شش ناحیه است. شکل ذیل، مدل شش ناحیه‌ای حلزونی را نشان می‌دهد و شرح کامل نواحی موجود در شکل نیز در ادامه خواهد آمد:



روش حلزونی

– **ارتباط با مشتری(۵۲):** در این فاز، ارتباط و جلسات مؤثر بین توسعه‌دهنده و مشتری صورت می‌پذیرد که منجر به کشف چرایی و چیستی مسئله، تجزیه و تحلیل نیازها و بستن قراردادها می‌شود.

– **برنامه‌ریزی(۵۳):** فعالیتهایی است که برای تعریف منابع، زمان‌ها، و اطلاعات دیگر مربوط به پروژه نیاز است؛ به‌عنوان مثال، تشکیل تیم‌های کاری و تعیین وظایف آنها، جزء این فعالیت‌ها می‌باشد.

– **تحلیل ریسک(٤٥):** فعالیت‌های لازم برای برآورد خطرهای فنی و مدیریتی و بودجه و زمان را پوشش می‌دهد. این فاز حیاتی، در بیشتر الگوهای قبلی نادیده گرفته می‌شد که یکی از برتری‌های الگوی حلزونی نسبت به سایر الگوهای قبلی است. در این فاز، تحلیل ریسک در انواع ابعاد: ریسک پروژه، ریسک فناوری، ریسک تجاری و بازار فروش محصول، ریسک زمان پایان کار در پروژه، ریسک منابع مالی و امکانات، ریسک منابع پرسنلی و تیم‌های کاری انجام می‌شود. سپس، این ریسک‌ها بر اساس درجه اهمیت (خطرناک، بحرانی، کم اهمیت و قابل چشم‌پوشی) و هزینه و تأثیرشان در جدولی از بالاترین هزینه‌ها و خطرناک‌ترین ریسک‌ها تا کم‌اهمیت‌ترین آنها طبقه‌بندی شده و برای ریسک‌های پراهمیت‌تر خطوط تفکیک‌کننده ایجاد می‌شود و طرح مدیریت و کاهش ریسک برای ریسک‌های بالای خط اجرا می‌شود تا تأثیرشان در پروژه کاهش یابد

– **فاز مهندسی(٥٥):** فعالیت‌های لازم برای مهندسی و تولید و ساخت یک یا چند ارائه خاص از نرم‌افزار را پوشش می‌دهد.

– **ساخت و انتشار(٥٦):** مجموعه فعالیت‌های لازم برای تولید، تست، نصب و تهیه مستندات آموزشی را شامل می‌شود.

– **ارزیابی مشتری(٥٧):** فعالیت‌های لازم برای جمع‌آوری بازخوردهای مشتری بر اساس ارزیابی از نرم‌افزار در خلال مرحله مهندسی و پیاده‌سازی را پوشش می‌دهد.

مجموعه فعالیت‌هایی که در هریک از مراحل این روش باید انجام شود، بسته به بزرگی و کوچکی پروژه، تفاوت دارد. در تمام حالات، فعالیت‌هایی مثل مدیریت پیکربندی نرم‌افزار و کیفیت آن اعمال می‌شود. هنگامی که پروسه تکاملی نرم‌افزار شروع می‌شود، تیم مهندسی نرم‌افزار در جهت عقربه‌های ساعت این مسیر حلزونی را طی می‌کند. در مرحله اوّل، (دور اوّل) ممکن است یک پیش‌نمونه ساخته شود. هر بار که از ناحیه طراحی می‌گذریم، به طرح پروژه نزدیک‌تر می‌شویم. هزینه و زمان نیز طبق بازخوردی که از کاربر گرفته می‌شود، تنظیم می‌شود. علاوه بر آن، مدیر پروژه تعداد دورهای لازم برای تکمیل پروژه را تنظیم می‌کند. بر خلاف روش کلاسیک که پس از تولید نرم‌افزار پایان می‌یابد، این مدل می‌تواند از هر جهت خود را با حیات یک نرم‌افزار سازگار نماید.

این روش برای توسعه نرم‌افزارهای بزرگ، یک روش واقع‌گرا و بهینه است؛ زیرا حین اینکه پردازش پیشرفت می‌کند، نرم‌افزار نیز گسترش می‌یابد و در نتیجه، توسعه‌دهنده و مشتری بهتر با خطرات در هر سطح تکامل، برخورد می‌کنند. این روش، مدل پیش‌نمونه‌سازی را به‌عنوان مکانیزم کاهش خطر استفاده می‌کند و مهم‌تر اینکه، توسعه‌دهنده را قادر می‌سازد روش پیش‌نمونه را در هر مرحله از تکامل محصول، اعمال نماید. این مدل در هر مرحله، خطرات را در نظر می‌گیرد و این باعث کاهش آسیب‌پذیری سیستم می‌شود.

۸. مارپیچی بَرنده بَرنده(۵۸)

در روش حلزونی، یکی از اعمال چارچوبی، به برقراری ارتباط با مشتری مربوط می‌شد. هدف این عمل، روشن‌کردن خواسته‌ها از سوی مشتری است. درحالت ایده‌آل، سازنده صرفاً از مشتری می‌پرسد که چه چیزی مورد نیاز است و مشتری جزئیات لازم برای پیشرفت کار را فراهم می‌آورد. متأسفانه، چنین چیزی به‌ندرت رخ می‌دهد.

تولیدکننده نرم‌افزار مشتری، در جلسات خود اهداف کلی برای نرم‌افزار را تعریف می‌کند، نیازمندی‌های شناخته‌شده را مشخص می‌نماید و مواردی را که نیاز به تعریف بیشتر دارند، معیّن می‌کند. سپس، یک طراحی سریع اتفاق می‌افتد. طرح تولیدشده بر شیوه ارائه موضوعات از نرم‌افزار که توسط مشتری/ کاربر قابل مشاهده است، مثل روش‌های ورود اطلاع و فرمت خروجی، تأکید دارد

در حالت واقعی، مشتری و سازنده وارد فرایند گفت‌وگو و بحث می‌شوند و در این فرایند ممکن است، از مشتری خواسته شود تا میان عملکرد، کارایی و ویژگی‌های دیگر سیستم یا محصول، در مقابل هزینه و زمان بازیابی، موازنه برقرار کند. در بهترین مباحثات، سعی می‌شود تا یک نتیجه «بُرد بُرد» حاصل شود؛ یعنی مشتری با دستیابی به محصول یا سیستمی که واجد بیشتر نیازهای اوست، برنده می‌شود و سازنده با کارکردن در چارچوب مهلت و بودجه‌ای واقعی و قابل حصول، برنده خواهد شد.

روش مارپیچی win win بوهم، مجموعه‌ای از اعمال مباحثاتی را در آغاز هر دور جدید از الگوی مارپیچی تعریف می‌کند. به جای یک عمل ارتباط با مشتری، اعمال ذیل انجام می‌شود:

ـ شناسایی واگذارنده‌های کلیدی سیستم یا زیرسیستم‌ها؛

ـ تعیین «شرایط بُرد» واگذارنده‌ها؛

ـ بحث و گفت‌وگو درباره شرایط بُرد واگذارنده، برای مصالحه و توافق آنها در یک مجموعه شرایط بُرد بُرد، برای همه موارد مربوط (از جمله تیم پروژه نرم‌افزار).

با به پایان بردن موفقیت‌آمیز این مراحل اوّلیه، یک نتیجه بُرد بُرد حاصل می‌گردد که ملاک کلیدی برای پیشروی به سوی تعریف نرم‌افزار و سیستم می‌شود.

علاوه بر تأکیدهایی که بر زودهنگام‌بودن مباحث می‌شود، مدل win win سه مرحله فرایند دارد که نقاط لنگرگاه نام دارند. این نقاط، به تعیین زمان تکمیل یک چرخه حول مارپیچ کمک کرده، نقاط عطفی برای اتخاذ تصمیم، قبل از ادامه پروژه هستند. در اصل، نقاط لنگرگاه سه دید متفاوت از پیشرفت پروژه را در راستای طی‌کردن مارپیچ به دست می‌دهند.

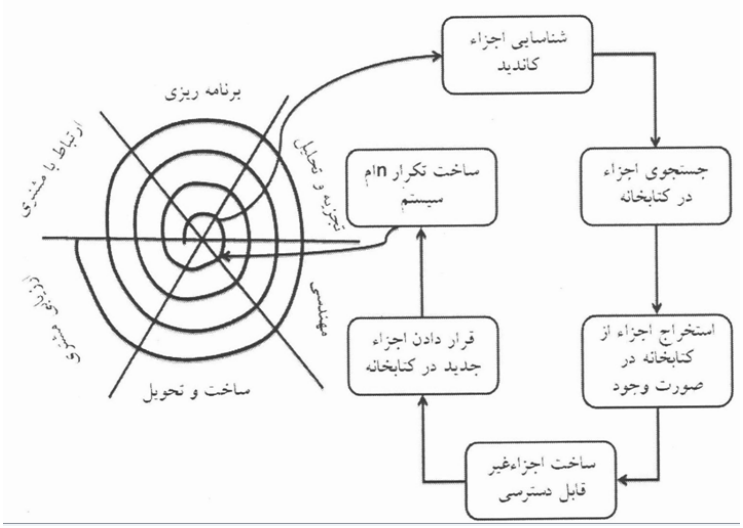
نخستین نقطه لنگرگاهی که «اهداف چرخه حیات» نام دارد، برای فعالیت عمده در مهندسی نرم‌افزار، مجموعه‌ای از اهداف را تعیین می‌کند؛ برای مثال، به‌عنوان بخشی از اهداف چرخه حیات، یک مجموعه اهداف با تعریف خواسته‌های محصول/ سیستم سطح بالا همراه می‌شود. دومین نقطه لنگرگاهی که «معماری چرخه حیات» نام دارد، اهدافی را تعیین می‌کند که باید به موازات تعریف معماری سیستم و نرم‌افزار برآورده شوند.

برای مثال، به‌عنوان بخشی از معماری چرخه حیات، تیم پروژه نرم‌افزاری باید تشریح کند که موجودیت مؤلفه‌های آماده و قابل استفاده مجدد را ارزیابی کرده، تأثیر آنها را بر تصمیم‌گیری‌های معماری مد نظر قرار داده است. «قابلیت عملیاتی اوّلیه»، سومین نقطه لنگرگاهی است و مجموعه‌ای از اهداف است که عبارت‌اند از: آماده‌سازی نرم‌افزار جهت نصب/توزیع، آماده‌سازی پایگاه پیش از نصب، و کمک به تمام کسانی که نرم‌افزار را استفاده یا پشتیبانی می‌کنند.

۹. مونتاژ مؤلفه‌ها(۵۹)

فناوری شئ‌گرا، یک چارچوب فنی جامع برای پردازش‌های مبتنی بر مؤلفه فراهم کرده است. کلاس‌های شیءگرا با ایجاد سطح انتزاع مناسب و سهولت طبقه‌بندی و دسترسی مؤلفه‌ها یک الگوی تکرارپذیر است که برنامه‌های کاربردی را بر اساس مؤلفه‌های نرم‌افزاری بسته‌بندی‌شده‌ای به نام «کلاس» ایجاد نموده و پس از ساخت کلاس‌ها، آنها درون مخزن اشیا(۶۰) نگهداری می‌شوند.

بنابراین، می‌توان در مراحل بعدی، از این دسته از کلاس‌ها در صورت نیاز استفاده مجدد نمود و کارایی توسعه و طراحی را افزایش داد. شکل ذیل، جزئیات این الگو را نشان می‌دهد:



روش مونتاژ مؤلفه‌ها

این روش، بسیاری از مشخصات مدل حلزونی را دارد؛ البته می‌توان از هر روش دیگری داخل این مدل استفاده کرد. در مرحله مهندسی، پیش از هر کاری به دنبال کلاسی می‌گردیم که بتواند بر روی داده مورد نظر، الگوریتم مشخص را پیاده کند. کلاس‌های موجود، از پروژه‌های دیگر در یک کتابخانه کلاس یا همان مخزن اشیا، ذخیره شده‌اند.

اگر کلاس مورد نظر وجود داشته باشد، استفاده مجدد می‌شود؛ وگرنه از نو ساخته می‌شود و طراحی و توسعه در تکرارهای مختلف بالغ‌تر شده و تکمیل می‌گردد. این روش، باعث می‌شود نرم‌افزارهای با قابلیت استفاده مجدد را با سرعت بالاتری تولید کنیم. برآوردها نشان می‌دهد که این مدل ۷۰٪ زمان گسترش سیستم را کاهش می‌دهد. البته این نتایج به طور دقیق به استحکام مؤلفه‌ها وابستگی دارد. نرم‌افزارهای تولیدشده از طریق مدل، غالباً دارای کیفیت بالا می‌باشد و در محیط‌های کاربردی و عملی، قدرت پاسخگویی خوب و کارایی مناسبی دارند.

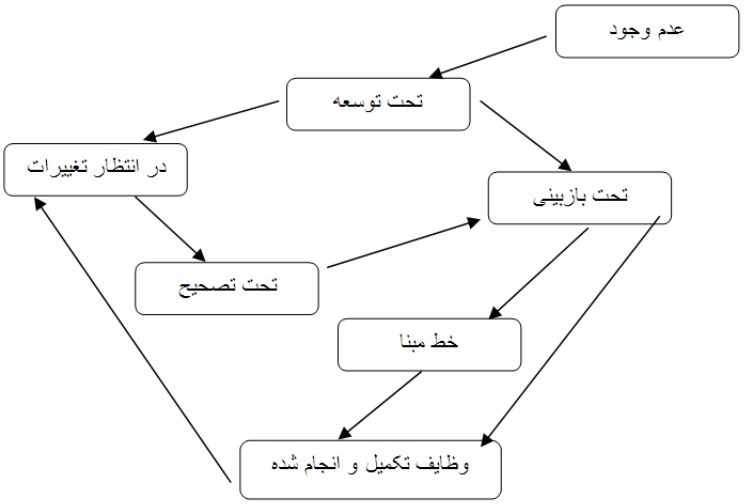
۱۰. توسعه هم‌روند(۶۱)

این روش با نام مهندسی هم‌روند(۶۲) نیز شناخته شده است. مدیران پروژه که وضعیت پروژه را از دیدگاه فازهای اصلی ردیابی می‌کنند، غالباً ایده‌ای در مورد وضعیت‌هایشان ندارند. در چنین مواردی که مدیران درگیر ردیابی وظایف متفاوت و پیچیده از فازهای مختلف توسعه محصول هستند، عمدتاً ترجیح می‌دهند از روش‌ها و الگوهایی که توسعه هم‌روند محصول را پشتیبانی کنند، استفاده نمایند.

یک فعالیت، مثل فعالیت تجزیه و تحلیل، می‌تواند در هریک از وضعیت‌های ذکرشده باشد. به طور

مشابه، فعالیت‌های دیگر مثل طراحی و ارتباط با مشتری، می‌توانند در حالتی مشابه ارائه شوند. تمام فعالیت‌ها به طور هم‌روند وجود دارند؛ ولی در حالت‌های مختلف قرار می‌گیرند؛ برای مثال، وقتی فعالیت ارتباط با مشتری، اوّلین دور خود را تمام می‌کند و در وضعیت انتظار تغییرات(۶۳) می‌باشد، فعالیت تجزیه و تحلیل که در وضعیت enon قرار دارد، به حالت تحت توسعه(۶۴) می‌رود. اگر مشتری تغییری در نیازهای خود انجام دهد، فعالیت تجزیه و تحلیل به وضعیت انتظار تغییرات می‌رود. این روش، یکسری اتفاقات را که باعث تغییر وضعیت فعالیت‌ها از یک وضعیت به وضعیت دیگر می‌شوند، تعریف می نماید.

برای مثال، اگر در مراحل اوّلیه طراحی، یک عدم سازگاری در مدل تحلیل‌شده کشف شود، اتفاق تصحیح مدل تحلیل‌شده، تولید می‌گردد که فعالیت تجزیه و تحلیل را از حالت انجام و تکمیل‌شده(۶۵) به حالت انتظار تغییرات می‌برد. شکل ذیل، یک فعالیت نوعی و انواع وضعیت‌های آن را، از روش توسعه هم‌روند نشان می‌دهد.



روش توسعه هم‌روند

این روش، بیشتر برای تولید برنامه‌های مشتری/خدمتگزار(۶۶) استفاده می‌شود. در واقع، این مدل بر تمام انواع روش‌های توسعه نرم‌افزار قابل اعمال است و تصویر دقیق از وضعیت جاری یک پروژه را می‌دهد. علاوه بر محدودکردن فعالیت‌های مهندسی نرم‌افزار، این مدل به ترتیب رویدادها، شبکه‌ای از فعالیت‌ها را تعریف می‌کند. هر فعالیت در شبکه به طور هم‌زمان با دیگر فعالیت‌ها وجود دارد و رویدادها در داخل یک فعالیت داده‌شده تولید می‌شود. در کل، می‌توان گفت این روش زمان اجرای پروژه را کم می‌کند و این امر با شناسایی فعالیت‌هایی که می‌توان به طور موازی و هم‌زمان، تحقق می‌یابد؛ ولی باید توجه داشت که در این روش، احتمال خطر بالا می‌رود؛ زیرا هماهنگی بین تیم‌های کاری مختلف، مشکل می‌شود و اگر تیمی کارش را انجام ندهد، کل پروژه زیر سؤال می‌رود.

۱۱. روش‌های رسمی(۶۷)

این روش، شامل مجموعه‌ای از فعالیت‌هاست که منجر به ایجاد مشخصات وابسته به ریاضی(۶۸) از نرم‌افزار می‌شود و در واقع، روش نیازمندی‌های محصول، نرم‌افزاری را به شبه‌کدهای مبتنی بر اثبات توسط فرمول‌های ریاضی تبدیل می کند.

بنابراین، هر مؤلفه قبل از اجرا و به‌کارگیری اعتبارسنجی فنی رسمی شده است. این مدل، مهندس

نرم‌افزار را قادر می‌سازد که سیستم‌های مبتنی بر رایانه را توسط اعمال یک علامت دقیق ریاضی، مشخص کرده و آن را گسترش دهد. استفاده از این روش، باعث حذف بسیاری از مشکلاتی می‌شود که غلبه بر آنها توسط روش‌های مهندسی نرم‌افزار دشوار است. ابهام، کم‌وکاست و عدم سازگاری را می‌توان به‌راحتی توسط آنالیزهای مبتنی بر ریاضی، کشف و اصلاح کرد. باید توجه داشت که این تشخیص و اصلاح، از طریق بازدید موردی انجام می‌گیرد.

وقتی روش‌های صوری در خلال طراحی استفاده می‌شود، به‌عنوان اساس و پایه‌ای برای تأیید برنامه مطرح است و بنابراین، مهندس نرم‌افزار را قادر به کشف و اصلاح خطاهایی می‌سازد که ممکن است در غیر این صورت کشف نشود.

از مهم‌ترین مشکلات، این است که گسترش مدل‌های صوری در حال حاضر، بسیار وقت‌گیر و هزینه‌بر است. همچنین، به دلیل فقدان پیش‌زمینه در مورد نیاز برای توسعه‌دهندگان نرم‌افزار در زمینه اعمال روش‌ها صوری، نیاز به آموزش گسترش وجود دارد. علاوه بر این، استفاده از این روش‌ها به‌عنوان مکانیزم ارتباطی با مشتریان با سطح دانش معمولی، بسیار مشکل است. پس، این روش، تنها برای نرم‌افزارهای امن و بحرانی مثل: برنامه‌های مربوط به هواپیمایی، دستگاه‌های پزشکی و سیستم‌های بلادرنگ، مفید و به‌صرفه است.

۱۲. فنون نسل چهارم(۶۹)

فنون نسل چهارم، شامل محدوده وسیعی از ابزارهای نرم‌افزاری است که یک ویژگی مشترک دارند و آن این است که مهندسان نرم‌افزار را قادر می‌سازند که برخی از مشخصات نرم‌افزار را در سطح بالا(۷۰) تعریف کنند. سپس، ابزار بر اساس مشخصات توسعه‌دهنده، کدهای مورد نظر را تولید می‌کند. به طور قطع، هر چه مشخصات نرم‌افزار در سطح بالاتری توسط ابزار تعریف شود، برنامه سریع‌تر ساخته می‌گردد.

این روش، بر توانایی مشخص‌کردن نرم‌افزار با استفاده از فرم‌های خاص و یا یک نشان گرافیکی تأکید دارد؛ به‌طوری که مشتری آن را بفهمد. در حال حاضر، یک محیط توسعه نرم‌افزار که فنون نسل چهارم را پشتیبانی کند، شامل بخش‌ها و ابزارهای ذیل است:

- زبان‌های غیر رویه‌ای برای پرس‌وجوهای پایگاه داده؛
- تولید گزارش؛
- دست کاری داده؛
- تولید کد؛

از مهم‌ترین مشکلات، این است که گسترش مدل‌های صوری در حال حاضر، بسیار وقت‌گیر و هزینه‌بر است. همچنین، به دلیل فقدان پیش‌زمینه در مورد نیاز برای توسعه‌دهندگان نرم‌افزار در زمینه اعمال روش‌ها صوری، نیاز به آموزش گسترش وجود دارد. علاوه بر این، استفاده از این روش‌ها به‌عنوان مکانیزم ارتباطی با مشتریان با سطح دانش معمولی، بسیار مشکل است

– قابلیت گرافیک سطح بالا؛
– قابلیت صفحه گسترده.

در ابتدا، بسیاری از ابزارهای گفته‌شده برای دامنه‌ای از برنامه‌های خیلی خاص در دسترس بود؛ اما امروزه محیط فنون نسل چهارم برای بیشتر برنامه‌ها گسترش یافته است. شبیه بسیاری از روش‌ها، GT4 با مرحله جمع‌آوری نیازمندی‌ها آغاز می‌شود. به طور مطلوب، مشتری نیازمندی‌های خود را تشریح می‌کند و این نیازمندی‌ها به طور مستقیم به داخل یک پیش‌نمونه عملیاتی ترجمه می‌شود. برای برنامه‌های کوچک، می‌توان مستقیماً از مرحله جمع‌آوری اطلاعات با استفاده از زبان‌های غیر رویه‌ای نسل چهارم به مرحله پیاده‌سازی رفت. به هر حال، برای نمونه‌های بزرگ‌تر، نیاز داریم که یک راهبرد طراحی برای سیستم داشته باشیم؛ هرچند از زبان‌های غیر رویه‌ای نسل چهارم استفاده شود.

شبیه تمامی روش‌های مهندسی نرم‌افزار، این مدل نیز مزایا و معایبی دارد. استفاده از زبان‌های غیر رویه‌ای نسل چهارم برای پروژه‌های بزرگ بدون طراحی جامع، باعث به‌وجود آمدن کیفیت پایین، نگهداری ضعیف و عدم پذیرش مشتری خواهد شد. در واقع، به همان مشکلاتی می‌رسیم که برای توسعه نرم‌افزار با روش‌های خاص داشتیم. پیاده‌سازی به‌وسیله فنون نسل چهارم، توسعه‌دهندگان نرم‌افزار را قادر می‌سازد که خروجی مورد نظر را با بهترین شکل و طرح ارائه دهند. به طور واضح، یک ساختار داده با داده‌های مرتبط باید موجود باشد و به طور آماده برای GT4 در دسترس باشد. برای تبدیل فنون نسل چهارم پیاده‌سازی‌شده به یک محصول، توسعه‌دهنده باید مراحل تست، تولید مستندات با مفهوم و تمامی فعالیت‌های یکپارچه‌سازی جواب را که در دیگر روش‌های مهندسی نرم‌افزار نیاز است، انجام دهد. به علاوه، نرم‌افزار گسترش‌یافته توسط فنون نسل چهارم، باید در حالتی ساخته شود که نگهداری آن به صورت فوری صورت گیرد.

۱۳. روش XP

روش XP(۷۱)، از رده روش‌های مدل‌سازی چابک(۷۲) است. این رده از الگوهای مدل‌سازی در مقابل رده مدل‌سازی روش‌های سنگین‌وزن(۷۳) قرار می‌گیرند و با یکدیگر قابل مقایسه می‌باشند. مهم‌ترین خصوصیات الگوهای مدل‌سازی چابک، این است که این رده از الگوها که عموماً برای پروژه‌های کوچک به‌کار گرفته می‌شوند، بسیار انعطاف‌پذیر، تطابقی، مقرون به‌صرفه، اکتشافی و غیرقابل پیش‌بینی هستند.

شناخت این روش‌ها، در بررسی و کنترل کیفیت نرم‌افزارها تأثیر بسیاری دارد. نظم، سرعت بالا، چالاکی بررسی و دقت، از جمله مواردی هستند که با شناخت روش‌ها به دست می‌آیند. امیدوارانه، می‌توان گفت در سال‌های اخیر محصولات برون‌خط و درون‌خط مرکز نور با کمترین اشکال و در زمان کم، بررسی شده و به تولید رسیده‌اند. با توجه به شرایط مرکز پیشنهاد می‌گردد که با مطالعه بیشتر در هریک از این روش‌ها، یکی از روش‌ها انتخاب و بومی‌سازی شود و به صورت جدی پیاده‌سازی گردد. همه این فعالیت‌ها، به کیفیت بالای محصول و در نهایت، رضایت مشتریان می‌انجامد.

۱۴. روش CMM

روش تکامل قابلیت (Capability Maturity Model) که به‌اختصار CMM نامیده می‌شود، شیوه‌ای است که برای توسعه و تصحیح فرایند توسعه نرم‌افزار استفاده می‌گردد. این مدل، توسط مؤسسه مهندسی نرم‌افزار (SEI) که مؤسسه توسعه و تحقیق مورد حمایت ایالات متحده است، ایجاد شد. CMM، مشابه ایزو ۹۰۰۱ می‌باشد. ایزو ۹۰۰۱، به طور خاص با توسعه و نگهداری نرم‌افزار سروکار دارد. تفاوت اصلی میان این دو روش، در اهداف آنها نهفته است: ایزو ۹۰۰۱، سطح قابل پذیرشی از کیفیت را برای فرایند نرم‌افزار تعیین می‌کند؛ درحالی‌که سی.ام.ام چارچوبی را برای بهبود پیوسته فرایند فراهم می‌آورد و در تعریف و رسیدن به راهکارهای لازم، صریح‌تر می‌باشد.

پی‌نوشت‌ها:

1. Process model.
2. Linear Sequential Mode.
3. Back Track.
4. System /Information Engineering.
5. Software Requirements Analysis.
6. Design.
7. Software Architecture.
8. Code Generation.
9. Testing.
10. Maintenance.
11. Specification Document.
12. Design Document.
13. Code.
14. User Manual.
15. Operatios Manual.
16. Test Cases.
17. The prototyping Model.
18. Report generator.
19. windows Manager.
20. Evolutionary Software Process Models.
21. The Incremental Model.
22. Core Product.
23. Technical Risks.
24. Rapid Application Development.
25. Component Based.
26. Business Modeling.
27. data modeling.
28. process model.
29. application generation.
30. components.
31. testing turnover.
32. scalable.
33. interfaces to system components.
34. interoperability.

structure systems analysis development method (

35. SSADM).
36. structural standards.
37. documentation standards.
38. Feasibility.
39. Business Case.
40. Investigation of Current Environment.
41. Business Area.
42. Business Activity Model.
43. Business System Options.
44. Business events.
45. Technical System Options.
46. Logical Design.
47. Maximal Cohesion & Minimal Coupling.
48. Physical Design.
49. The Spiral Model.
50. Ineremental Release.
51. Task Regions.
52. Customer Communication.
53. Planning.
54. Risk Analysis.
55. Engineering.
56. Construction & Release.
57. Customer Evaluation.
58. Win Win Model.
59. The Component Assembly Model.
60. Object Pool.
61. The Concurrent Model Development.
62. Concurrent Engineering.
63. Awaiting Changes.
64. Under Development State.
65. Done.
66. Client / Server.
67. The formal Methods Model.
68. Mathematical Specification.
69. Fourth Generation Techniques.
70. High Level.
71. Extreme Programming.
72. Agile Modeling.
73. Heavy Weight Modeling. ■