((به نام خدا))

موضوع :

تعمیر و نگهداری نرم افزار

(Software Maintenance)

تهیه و تنظیم :

مهندس ایمان اشکاوند راد

Email : i81a@yahoo.com

Http://www.ashkavand.ir

دانلود مقاله ، PDF ، برنامه نویسی موبایل ، دلفی ، وب و ...

چکیده

تعمیر نرم افزار پرهزینه ترین قسمت توسعه سیستم اطلاعاتی است که ۶۰ تا ۸۰ درصد از منابع را مصرف می کند. بزرگترین عامل برای این هزینه ، درک نادرست برنامه است. وقتی که یادگیری تعمیر و نگهداری برنامه ها سخت باشد،کسانی مثل توسعه دهندگان زمان بیشتری را برای خواندن کد برنامه صرف می کنند و تغییرات نادرستی را در آن ایجاد می کنند. روشهای زیادی مثل (ابزارهای CASE,IDE) مکانیزمهای نمایشی به زبانهای برنامه نویسی موجود اضافه می کنند بجای اینکه تعییرات اساسی در روش کد نویسی ارائه دهند. بنابراین یادگیری کد در ادامه استفاده از ابزار CASE یا IDE است. این عامل می تواند مشکل ساز هم باشد چون که این ابزار اغلب در طی مراحل اولیه توسعه یک برنامه بکار می روند و اما بکار گیریشان در طی مراحل تعمیر و نگهداری مشکل است. یک بخشی که هزینه بالایی را در تعمیر نرم افزار دارد ، سیستم های موروثی است. سیستم هایی وجود دارد که از توسعه آنها بیش از ۲۰ تا ۳۰ سال (یا حتی بیشتر) به دلیل درخواست های توسعه و رشد برای تولیدات و سرویس های جدید ، می گذرد. یک مشکل عادی سازمان های تعمیر و نگهداری بی تجربگی پرسنل و کارکنان آنها است اکثر افرادی که کار تعمیر و نگهداری سیستم را انجام می دهند دانشجویان و افراد جدید کارمزد هستند.

کلید واژه

تعمیر و نگهداری نرم افزار ، مهندسی مجدد ، مهندسی معکوس ، **Software Maintenance** ، سیستم های موروثی

۱ – مقدمه

واژه "Maintenance" هنگامی که به همراه واژه نرم افزار می آید معنی متفاوتتری از معنی آن در دیگر نظام های مهندسی می دهد. در حقیقت تعداد زیادی از نظام های مهندسی قصد دارند که Maintenance را به عنوان یک فرآیند ی که از نظم اجزای در حال کار سیستم برای عمل تعمیر نگه داری می کند، معرفی کنند. در واقع مفهومش این است که روند رو به نزول یک محصول نرم‌افزاری به دلیل استفاده و گذشت زمان موجب تعمیر و نگهداری آن می شود که برای این منظور باید توابع و وظایف محصول به همراه تعاریف و ثبت آن در زمان انتشار نگه داری شود. البته این دید از Maintenance برای نرم افزار به کار گرفته نمی شود زیرا نرم افزار با استفاده وگذشت زمان از بین نمی رود. با این وجود نرم افزار بعد از تحویل به مشتری و بعد از مدتی که مقدار زیادی محاسبات الکترونیکی را انجام داده باشد نیاز به اصلاح قطعاتش است. قوانین تکامل لمان [50,51] توضیح می دهد که یک سیستم نرم افزاری موفق بعد از به پایان رسیدن زمانش، نیاز به تغییر دارد. سهم عمده ای از تغییرات مربوط به تغییراتی است که کاربر نیاز دارد. این مسئله به وسیله اولین قانون لمان بیان می شود.[50,51] برنامه ای که در محیط واقعی استفاده می شود لزوماً باید تغییر کند یا در حال اندکی پیشرفت در جهت سودمند شدن سیستم در آن محیط باشد. همچنین مهمترین تغییرات از نیاز به تطبیق نرم افزار برای تعامل با موجودیت های خارجی مانند افراد ، سازمان ها و سیستم های مصنوعی ناشی می گردد. در واقع به دلیل این که نرم افزار خیلی انعطاف پذیر است اغلب به عنوان آسانترین قسمت برای تغییرات در سیستم دیده می شود[21]

در این مقاله یک بازنگری بر روی فرآیند تعمیر و نگهداری نرم افزار[1] از قبیل ارتباطات ، مشکلات و راه حلهای در دسترس ، صورت پذیرفته است. ادامه مقاله به شرح زیر است: بخش دوم به تعریف تعمیر و نگهداری نرم افزار می پردازد و بخش سوم به دسته بندی های آن اشاره دارد. هزینه ها و رقابت های تعمیر و نگهداری نرم افزار در بخش چهارم بررسی می گردد. بخش پنجم تا هفتم به ساختار فعالیت تعمیر و نگهداری اختصاص دارد. به ویژه در بخش پنجم مدل های اصلی فرآیند تعمیر و نگهداری ، وظایف ، استفاده مجدد و ابزارها معرفی می گردند حال آن که در فصل ششم بر روی استانداردهای موجود بحث خواهد شد. مدیریت تعمیر نرم افزار در فصل هفتم مطرح خواهد شد.

---

[1] Software Maintenance

فصل های هشتم و نهم با دو موضوع مرتبط که در تعمیر و نگهداری نرم افزار پشتیبانی می گردد سروکار دارد، به نام های مهندسی معکوس و مهندسی مجدد. فصل دهم نیز به سیستم های موروثی[2] به عنوان یک پیامد مرتبط با رشد اقتصادی در دهه اخیر اشاره دارد. در آخر هم نکات و حوزه هایی برای بررسی در آینده در فصل یازدهم معرفی خواهد شد و فصل دوازدهم هم به معرفی منابع می‌پردازد.

۲- تعاریف

تعمیر نرم افزار فعالیت خیلی بزرگی است که اغلب به عنوان تمام اعمالی که روی سیستم نرم‌افزاری بعد از آن که سیستم قابل استفاده شد، انجام می گیرد تعریف می گردد.[56] این فعالیت تمام اعمال اصلاح خطاها ، افزایش[3] ، حذف و اضافه کردن قابلیت های جدید ، انطباق با تغییرات در داده‌های مورد نیاز و محیط های عمل ، بهبود کارایی ، قابلیت استفاده یا هر صفت کیفیت دیگر را پوشش می دهد.

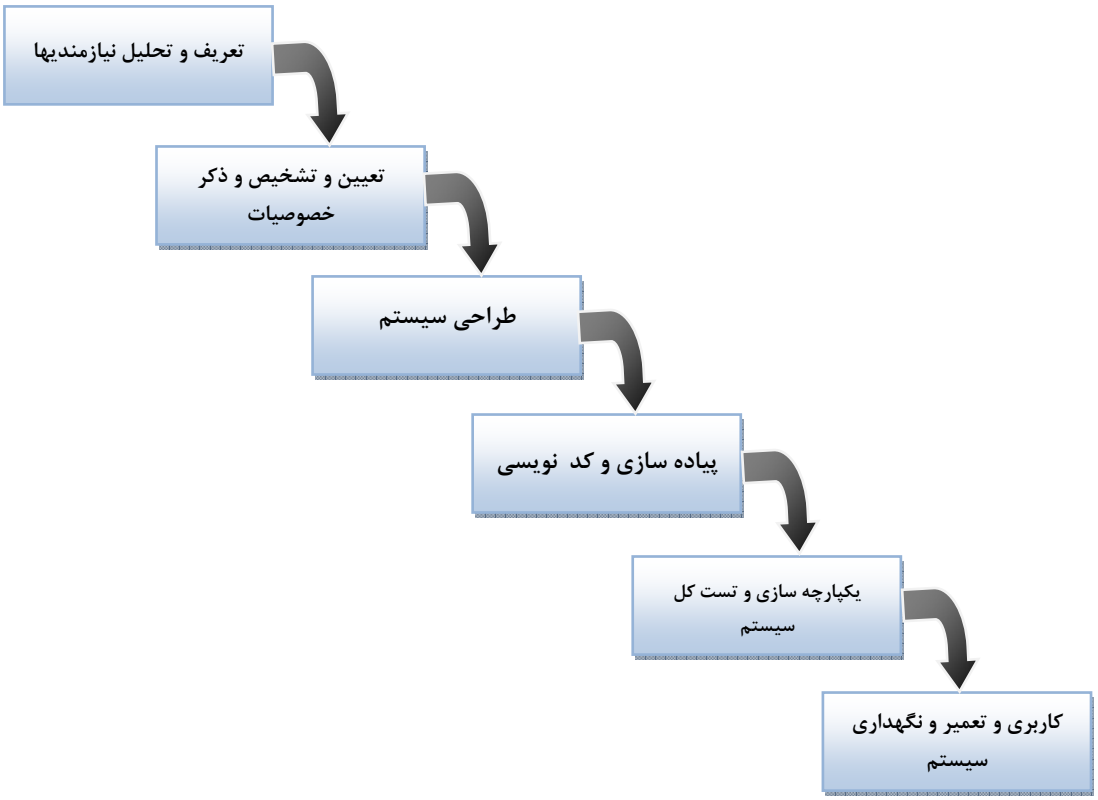
IEEE تعمیر نرم افزار را این گونه تعریف می نماید :

"تعمیر نرم افزار عبارت است از فرآیند اصلاح سیستم نرم افزاری یا اجزای آن بعد از تحویل سیستم ، برای رفع خطاها، بهبود کارایی یا دیگر خصوصیات یا تطبیق با تغییرات محیط"

این تعریف یک دید معمولی از تعمیر نرم افزار را که یک فعالیت بعد از تحویل[4] است را معرفی می‌کند. تعمیر نرم افزار هنگامی که یک سیستم به مشتری یا کاربر تحویل داده می شود شروع می‌گردد و همه فعالیت هایی که از عملیات سیستمی نگهداری می کند و با نیازهای کاربر مواجه است را در بر می گیرد. این دید به خوبی به وسیله چرخه حیات مدل آبشاری کلاسیک ، که عموماً شامل یک فاز نهایی از عملیاتی شدن سیستم و تعمیر آن است ، نشان داده می شود. شکل ۱ را مشاهده فرمایید.

---

[2] Legacy System
[3] Enhancement
[4] Post-Delivery



شکل ۱ – مدل آبشاری

تعدادی نویسنده با این دیدگاه موافق نبودند و اظهار کردند که تعمیر نرم افزار باید قبل از این که سیستم عملیاتی گردد شروع شود. Osborne,Chikofski [59] بیان داشتند که تعمیر نرم افزار را می توان به عنوان یک بخش اصلی از دوره حیات سیستم برای مدیریت تغییرات سیستم نرم افزاری پذیرفت. Pigoski [62] نیازهای لازم برای تعمیر نرم افزار هنگامی که توسعه سیستم شروع می گردد را می گیرد و آن را در قالب تعریف جدید بیان می کند:

"تعمیر نرم افزار مجموع فعالیت هایی است که برای فراهم کردن هزینه موثر پشتیبانی از یک سیستم نرم افزاری مورد نیاز است. فعالیت ها در طول مراحل قبل از تحویل[5] و بعد از تحویل اجرا می گردند. فعالیت های قبل از تحویل سیستم شامل برنامه ریزی برای عملیات بعد از تحویل سیستم است که قابل پشتیبانی و تصمیم گیری است. فعالیت بعد از تحویل شامل اصلاح نرم افزار، آموزش و یک عامل کمکی و راهنما می باشد"

[5] Pre-Delivery

این تعریف برای تعمیر نرم افزار با هدف استاندارد ISO بر روی فرآیندهای دوره حیات نرم افزار سازگار است.[44] این تعریف به طور قطع این تصویر از تعمیر نرم افزار که فقط به رفع خطا یا اشتباهات می پردازد را برطرف می کند.

۳- دسته بندی های تعمیر و نگهداری نرم افزار

در طول دو دهه ۱۹۷۰ و ۱۹۸۰ چندین محقق بر روی پدیده تعمیر و نگهداری با هدف تشخیص دلایلی که موجب شناسایی نیازهای لازم برای تغییرات ، نوسان ها و هزینه های مربوط به آنها می گردید، مطالعه می کردند. به عنوان نتیجه این مطالعات چندین دسته بندی از فعالیت تعمیر و نگهداری تعریف شده است. این دسته بندی ها به فهم بهتر اهمیت تعمیر و تاثیر آن روی هزینه و کیفیت سیستم برای استفاده کمک می کند.

Lienz and Swanson [54] تعمیر را به سه جز تقسیم کردند: اصلاح[۶] ، تطبیق[۷] و تکامل[۸].

نگهداری اصلاحی[۹] شامل همه تغییراتی است که موجب حذف فعالیت های معیوب در نرم افزار می گردد. عملیات نگهداری تطبیقی[۱۰] شامل تغییرات مورد نیاز برای تطبیق سیستم با جهشها در محیطی است که سیستم باید در آن فعالیت داشته باشد. برای مثال تغییرات یک سیستم برای کار آن بر روی پلتفرم سخت افزار جدید، سیستم عامل جدید ، DBMS ، Tp Monitor یا شبکه.

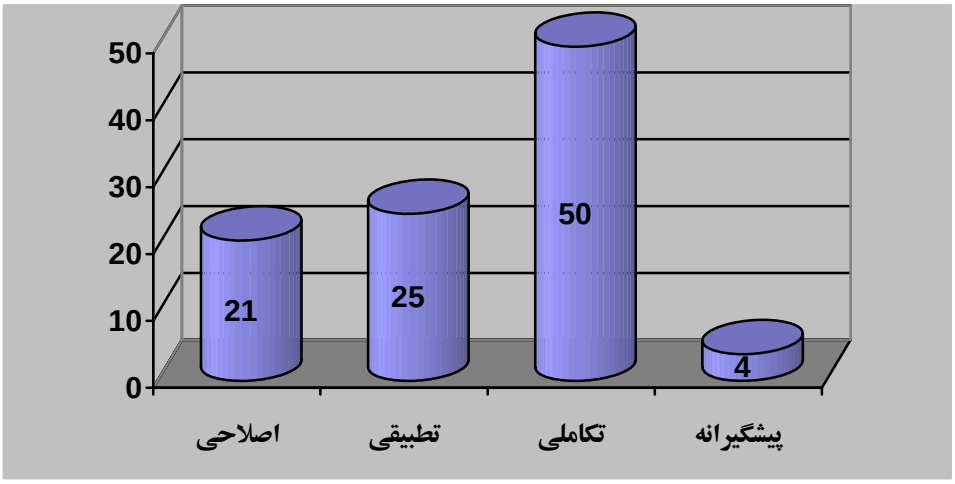
نگهداری تکاملی[۱۱] شامل تغییراتی است که از جانب درخواست های کاربر سازمان دهی می شود برای مثال می توان به مواردی چون اضافه کردن ، حذف و اصلاح توابع ، بازنویسی مستندات ، بهبود کارایی یا بهبود سهولت استفاده اشاره کرد. Pigoski[62] پیشنهاد می کند که دو دسته تطبیق و تکامل را به هم ملحق کرده و آنها را افزایش[۱۲] بنامیم. این دو نوع تغییر ، ماهیت را عوض نمی کنند بلکه آن را بهبود می بخشد. حقیقت امر این است که بعضی از سازمانها واژه تعمیر و نگهداری را برای رجوع به پیاده سازی تغییرات کوچک استفاده می کنند. در حالی که توسعه نرم افزار برای رجوع به همه نوع تغییرات استفاده می شود.

حالت ایده آل این است که عملیات تعمیر نباید قابلیت اطمینان را کاهش دهد و ساختار اجزای سیستم را به هم زند. هیچ یک از آنها نباید قابلیت تعمیر[۱۳] را کاهش دهند. در غیر این صورت تغییرات آینده تدریجاً مشکلتر و هزینه پیاده سازی آنها بیشتر و گرانتر خواهد شد. متاسفانه این

---

[6] Corrective
[7] Adaptive
[8] Perfective
[9] Corrective Maintenance
[10] Adaptive Maintenance
[11] Perfective Maintenance
[12] Enhancement
[13] Maintainability

حالت در عمل این گونه نیست و اغلب باعث یک پدیده کهنگی اشیاء سیستم می‌گردد.[60] این مطلب طبق قانون دوم لمان بیان می شود [50,51] به عنوان استنتاج از تغییرات برنامه مشاهده می‌شود که ساختار برنامه تمایل دارد که پیچیده شود. منابع زیادی باید برای حفظ منطق و ساده سازی ساختار برنامه اختصاص پیدا کند. بنابراین تعدادی از محققان گروه چهارمی را برای تعمیر در نظر گرفتند که آن را نگهداری پیشگیرانه[۱۴] نامیدند که شامل همه اصلاحاتی بود که سبب می‌شد که یک قطعه نرم افزار قابل تعمیر شود. [63] در شکل (۲) درصد هر یک از انواع عملیات های تعمیر و نگهداری را مشاهده می نمایید.



شکل ۲– درصد انواع فرآیند های تعمیر و نگهداری

ISO [43] سه دسته تعمیر نرم افزار را معرفی کرده بود : (**مشکل تفکیک پذیری**[۱۵]) که شامل شناسایی ، تحلیل و اصلاح نرم افزار است. عدم تشابه این دسته ها موجب ایجاد مشکلات موثری است. **اصلاح رابط**[۱۶] ، زمانی نیاز می شود که اضافه کردن یا تغییر موجب شود که سیستم سخت‌افزاری به وسیله نرم افزار کنترل شود. **گسترش تابعی یا بهبود کارایی**[۱۷]، ممکن است به وسیله خریدار در مرحله تعمیر لازم شود.

تعریف Maintainability از دید IEEE: " Maintainability بازتابی از تعریف تعمیر و نگهداری است. سادگی سیستم نرم افزاری یا مولفه های آن ، زمانی که می خواهد برای اصلاح خطاها، بهبود کارایی یا دیگر مشخصه ها یا تطبیق با تغییرات محیط ، تغییر نماید. [40] ISO

[14] Preventive Maintenance
[15] Problem Resolution
[16] interface modifications
[17] functional expansion or performance improvement

قابلیت تعمیر را به عنوان یکی از شش مولفه اصلی تعریف کیفیت نرم افزار فرض کرده و پیشنهاد نموده است که آن بر پایه چهار زیر مشخصه قابل تجزیه و تحلیل ، قابل تغییر ، پایداری و قابل تست باشد. [42] نسخه جدید استاندارد، قابلیت اضافه کردن را به عنوان پنجمین زیر مشخصه تحت توسعه پیشنهاد کرده است."

یک توصیه این است که تا آنجا که بتوان برای توسعه نرم افزار همه تغییرات باید بر طبق همان روال ها باشد. برای این که کمترین زمان خواب سیستم را داشته باشیم این حالت امکان پذیر است که از رفع مشکلات به صورت موقت استفاده کنیم و بعداً پیاده سازی تغییرات را دائمی کنیم.

IEEE [41] دوباره دسته بندی Lientz , Swanson [54] را تعریف می کند: نگاهداری اصلاحی ، تطبیق و تکامل و گروه چهارمی را به عنوان نگهداری اضطراری[18] اضافه می کند. تعریف IEEE را در زیر مشاهده می نمایید:

"Corrective Maintenance" : تغییراتی که بر روی تولیدات نرم افزاری بعد از تحویل سیستم برای رفع خطاهای کشف شده انجام می گیرد.

"Adaptive Maintenance": تغییراتی که روی تولیدات نرم افزاری بعد از تحویل ، برای حفظ قابل استفاده بودن برنامه در محیط های تغییر یافته یا در حال تغییر صورت می گیرد.

"Perfective Maintenace": تغییراتی که روی تولیدات نرم افزاری بعد از تحویل برای بهبود کارایی یا قابل تعمیر بودن صورت می گیرد.

"Emergency Maintenance" : نگهداری اصلاحی زمان بندی نشده که برای حفظ قابل استفاده بودن سیستم اجرا می شود.

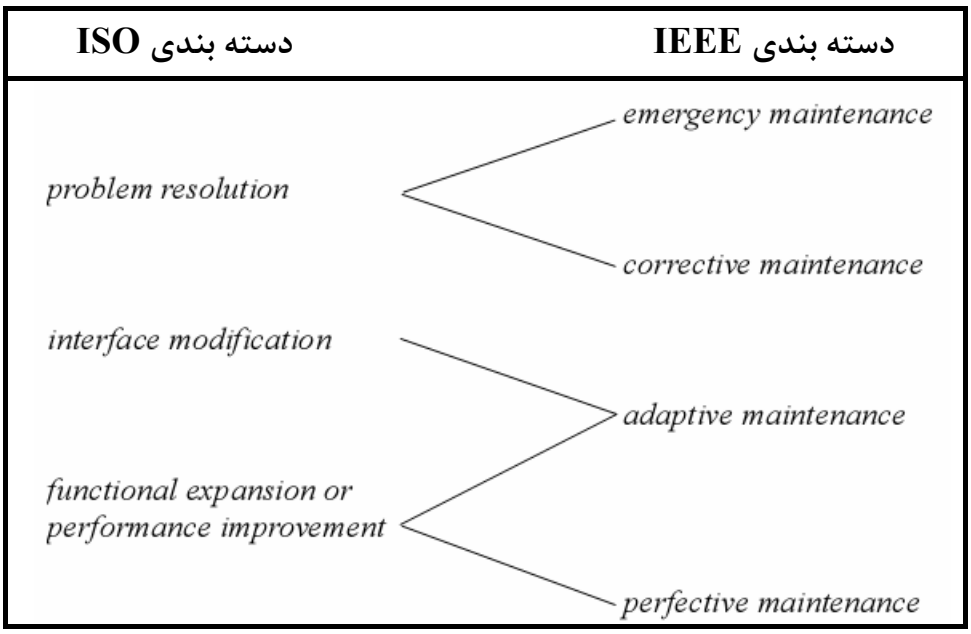
این تعاریف مقدمه ای بود بر نظریه تعمیر نرم افزار که می تواند هر یک از حالتهای زمانبندی شده یا زمان بندی نشده ، reactive یا Proactive باشد. که این حالتها را در شکل ۳ مشاهده می‌نمایید.

	زمان بندی نشده	زمان بندی شده
Reactive	**فوری و ضروری**	**اصلاحی** **تطبیقی**
Proactive		**تکاملی**

شکل ۳ – دسته بندی Software Maintenance از دید IEEE

[18] Emergency Maintenance

شکل ۴ تناظر ی که بین دسته بندی های IEEE و ISO وجود دارد را نشان می دهد.



شکل ۴ - تناظر بین دسته بندی های IEEE و ISO

۴- هزینه ها و رقابت ها

اگرچه یکی از تصمیمات این بود که تلاشهای تعمیر سیستم را دسته بندی کنند، ولی هزینه و بودجه تعمیر و نگهداری یک سیستم بزرگ هنوز برای اطلاعات سازمان آن سیستم واضح نیست. اشکال گوناگون چندین بررسی تعمیر نرم افزار [1,2,7,10,34,46,54,56,58] نشان داده است که تعمیر و نگهداری سیستم حدود ۶۰ تا ۸۰ درصد از کل هزینه های دوره حیات سیستم را به خود اختصاص می دهد. این بررسی ها همچنین گزارش داده اند که هزینه های تعمیر به دلیل این که هزینه های افزایش سیستم (اغلب ۷۵-۸۰٪) بیش از اصلاح آن است، بسیارگران تمام می‌شود.

چندین تکنیک و مدیریت مشکلات در هزینه های تعمیر نرم افزار شرکت دارند. از این میان بزرگترین مشکلات تعمیر نرم افزار عبارتند از : **درک برنامه**[۱۹] **، آنالیز ضربه ای**[۲۰] **و تست برگشت**[۲۱]

هنگامی که یک تغییر روی قطعات نرم افزاری اتفاق می افتد این مسئله خیلی مهم است که نگهدارنده[۲۲] سیستم درک کاملی از ساختار ، رفتار و توابع سیستم در شروع عملیات اصلاح به دست

[19] Program Comprehension
[20] Impact Analysis
[21] Regression Testing
[22] Maintainer

بیاورد. یکی از اساسی ترین این شناخت ها می تواند تولید یک طرح پیشنهادی اصلاح برای انجام هدف تعمیر (قابل تعمیر بودن سیستم) باشد. به عنوان نتیجه ، نگهدارندگان سیستم مقدار زیادی از وقتشان را صرف خواندن کد و مستندات زمینه آن برای درک منطق، هدف و ساختار سیستم می‌کنند. برآوردهای موجود نشان می دهد زمانی که برای درک برنامه برای عمل تعمیر صرف می‌شود بین ۵۰ تا۹۰ درصد متغیر است.[32,55,73] درک برنامه خیلی از اوقات ترکیبی از دو مسئله است زیرا اولاً نگهدارنده سیستم خیلی به ندرت پیش می آید که صاحب (نویسنده)کد برنامه باشد (هنگامی که یک بازه زمانی مهمی که بین توسعه و تعمیر و نگهداری سپری می شود) و ثانیاً مستندات کامل و به روز به ندرت در دسترس است. [26]

یکی از چالشهای مهم در تعمیر نرم افزار تصمیم گیری برای تاثیر اصلاحات پیشنهادی روی سیستم موجود است. **آنالیز ضربه ای** [6,64,35,81] عبارت است از عمل برآورد تاثیر نیروی بالقوه برای تغییر با هدف کمینه کردن تاثیرات غیر منتظره. وظیفه آنالیز ضربه ای شامل برآورد تغییرات پیشنهادی و ارزیابی ریسک پذیری آن پیاده سازی است و همچنین شامل برآورد تاثیر آن روی منابع ، تلاش و زمانبندی می باشد. همچنین آن به عنوان یک نتیجه از تغییرات پیشنهادی درگیر قسمتهایی از سیستم است که نیاز به اصلاح دارند. نکته ای که وجود دارد این است که اگرچه آنالیز ضربه ای یک وظیفه مرکزی را در فرآیند تعمیر و نگهداری سیستم بر عهده دارد اما توافقی در مورد تعریف آن در واژه نامه فنی مهندسی نرم افزار IEEE وجود ندارد. [40] و تعریفی برای آن ارائه نشده است.

بعد از این که یک تغییر روی سیستم پیاده شد ، سیستم نرم افزاری برای اطمینان از این که اجرای آن بر طبق اصلاحات امکان پذیر است دوباره تست می شود. فرآیند تست سیستم بعد از اصلاح آن تست برگشت[۲۳] نامیده می شود.[52] هدف از تست برگشت دو چیز است: ۱- کسب اطمینان از این که تغییرات درست است ۲- مطمئن شدن از این که قسمتهای تغییر نیافته سیستم تحت تاثیر قرار نگرفته اند. تست برگشت از تستی که در زمان توسعه سیستم اجرا می گردد متفاوت است زیرا یک مجموعه از موارد تست ممکن است برای استفاده مجدد در دسترس باشد. حقیقتاً تغییراتی که در طول فرآیند تعمیر به وجود می آید، معمولاً کوچک است (بازنویسی‌های بزرگ نسبتاً رویداد نادری در تاریخچه سیستم هستند) و بنابراین اجرای معمولی همه موارد تست بعد از هر تغییر ممکن است خیلی هزینه بر باشد. از طرف دیگر چندین استراتژی برای انتخاب تست برگشت موجود است که تلاش می کند که زیر مجموعه ای از نمونه های تست موجود را بدون تاثیر بر کارایی تست انتخاب کنند.[39,66]

---

[23] Regression Test

مشکل اصلی این است که تعمیر نرم افزار می تواند بر روی کاستی های فرآیند توسعه نرم افزار تاثیر بگذارد. Sneidewind [67] اظهار کرده است که مشکل اصلی در انجام تعمیر و نگهداری این است که ما نمی توانیم عمل تعمیر و نگهداری را روی سیستمی که برای این منظور طراحی نشده است انجام دهیم. بنابراین مشکلات عمده دیگری از قبیل مشخصات ذاتی نرم افزار و فرآیند تولید آن در عملیات تعمیر و نگهداری سیستم شرکت می کنند. Brooks [21] پیچیدگی ، انطباق، قابل تغییر بودن و نامرئی بودن را به عنوان چهار مشکل اصلی نرم افزار شناسایی کرده و Rajlich[65] ناپیوستگی را به این لیست اضافه می کند.

در این لیست بعضی از ارقام قابل توجه بر روی هزینه های تعمیر و نگهداری نسبی و مطلق و نسبت آن با انواع وظایف اصلی را نشان می دهد. این ارقام و اعداد بر پایه داده های تجربی هستند. اگرچه تحقیقات تجربی زیادی روی این ناحیه خاص صورت نگرفته ولی بزرگی هزینه های تعمیر و نگهداری به صورت واضح مشخص است.

۴-۱-نسبت هزینه های تعمیر و نگهداری نرم افزار

در این قسمت به بررسی برخی از هزینه های تعمیر ونگهداری می پردازیم . هزینه های تعمیر نرم افزار و مدیریت تکامل آن حوزه نشان می دهد که بیش از ۹۰٪ از کل هزینه های نرم افزار را به خود اختصاص می دهد. این مسئله به بحران موروثی[۲۴] که به وسیله seacord et al (2003) مطرح شده است مربوط می گردد. مطالعات گوناگون روی این بحث در جدول زیر آمده است.

Year	Proportion of software maintenance costs	Definition	Reference
2000	>90%	Software cost devoted to system maintenance & evolution / total software costs	Erlikh (2000)
1993	75%	Software maintenance / information system budget (in Fortune 1000 companies)	Eastwood (1993)
1990	>90%	Software cost devoted to system maintenance & evolution / total software costs	Moad (1990)
1990	60-70%	Software maintenance / total management information systems (MIS) operating budgets	Huff (1990)
1988	60-70%	Software maintenance / total management information systems (MIS) operating budgets	Port (1988)
1984	65-75%	Effort spent on software	McKee (1984)

[24] legacy crisis

		maintenance / total available software engineering effort.	
1981	>50%	Staff time spent on maintenance / total time (in 487 organizations)	Lientz & Swanson (1981)
1979	67%	Maintenance costs / total software costs	Zelkowitz *et al.* (1979)

جدول ۱ – درصد هزینه های تعمیر و نگهداری صرف شده

۴-۲- هزینه های تعمیر مطلق نرم افزار

- هزینه های تعمیر نرم افزار سالیانه در USA بیش از ۷۰ بیلیون دلار برآورد شده است. (Sutherland, 1995; Edelstein, 1993)
- به عنوان مثال در USA دولت فدرال به تنهایی حدود ۸,۳۸ بیلیون دلار را در طول یک دوره پنج ساله برای تصحیح Y2K-Bug هزینه کرده است.
- در سطح شرکتها به طور مثال Nokia inc حدود ۹۰ میلیون دلار برای پیشگیری از تصحیح Y2k-Bug هزینه کرده است.

۴-۳- انواع وظایف تعمیر

- حدود ۶۵ درصد از فرآیند تعمیر صرف عملیات تکاملی می گردد. **(Lientz & Swanson 1981)**
- حدود ۷۵ درصد از هزینه های تعمیر صرف فراهم کردن افزایش (در شکلهای نگهداری تطابقی و تکاملی) می گردد. **(Martin, 1983; Nosek & Palvia, 1990; van Vliet, 2000)**
- مطالعات تعمیر نشان داده است که تقریباً حدود ۵۰ درصد از زمان صرف شده در فرآیند تعمیر صرف فهمیدن کد می گردد.**(Fjeldstad & Hamlen, 1983; Standish, 1984)**

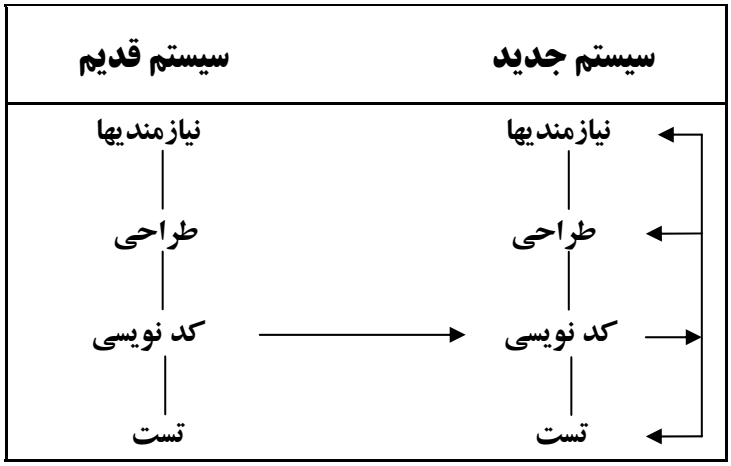
۴-۴- مقادیر کدهای موروثی

- در سال ۱۹۹۰ حدود ۱۲۰ بیلیون خط کد منبع برای شروع عملیات تعمیر تخمین زده شد.
- در سال ۲۰۰۰ در حدود ۲۵۰ بیلیون خط کد منبع برای شروع عملیات تعمیر تخمین زده شد و تعداد آن در حال افزایش است.**(Sommerville, 2000)**
- به عنوان یک میانگین از دارایی ۱۰۰ شرکت ، آنها از حدود ۳۵ میلیون خط کد نگهداری می کردند.**(Müller et al., 1994)**

- این مقدار سالیانه حدود ۱۰ درصد به دلیل فرآیند افزایش اضافه مـی گـردد. **(Müller et al., 1994)**
- به عنوان نتیجه مقادیر کدهای تعمیر و نگهداری در هر ۷ سال دو برابر می شود. **(Müller et al., 1994)**
- زبانهای قبلی منسوخ نشده اند. به طور مثال حـدود ۷۰ درصـد یـا بیـشتر ، از برنامـه هـای کاربردی تجاری فعـال بـه زبـان COBOL نوشـته شـده انـد. (Giga Information Group)
- دست کم حـدود ۲۰۰ بیلیـون خـط کـد COBOL بـه تنهـایی هنـوز در کامپیوترهـای mainframe موجود می باشد. (Giga Information Group)

۵ – مدل ها ، وظایف ، استفاده مجدد و ابزارها

یک مدل از تعمیر و نگهداری نرم افزار این گونه است که ابتدا بر روی کـد کـار کـرده و سـپس تغییرات لازم را برای مستندات ضمیمه فراهم می کند، البته اگر مستندات داشته باشد. ایـن حالـت از مدل quick-Fix گرفته شده است. در شکل ۵ این مدل را مشاهده می نمایید که جریان تغییرات را از نسخه قدیم به نسخه جدید سیستم نشان می دهد.[8]



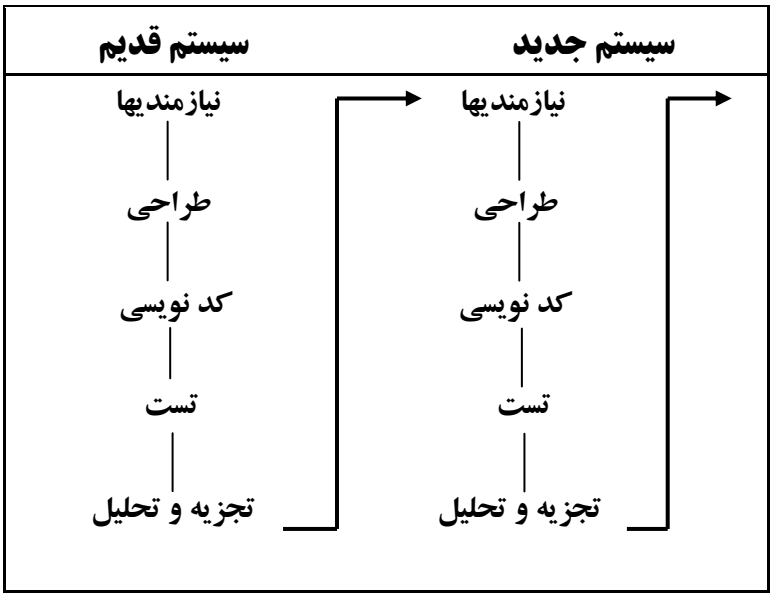
شکل ۵ – مدل Quick-Fix

حالت ایده آل این است که پس از این که کد تغییر کرد نیازها ، طراحی ، تـست و دیگـر اشـکال مستندات به هم پیوسته موجود ، به وسیله اصلاح به روز رسانی شود. بنابراین به دلیـل درک قابلیـت انعطاف پذیری نرم افزار کاربران انتظار دارند که نرم افزار به سرعت و بـا هزینـه معقـول[25] اصـلاح

[25] Cost-effectively

شود. تغییرات معمولاً در جنبش هستند بدون این که طرح و نقشه ، طراحی ، آنالیز ضربه ای و تست برگشت مناسبی داشته باشند. مستندات ممکن است هنگامی که کد اصلاح شد به روزرسانی شود یا نشود. زمان و بودجه اغلب در مضیقه هستند و موجب می شوند که تغییرات برنامه مستندسازی نشود و این به شدت مستند سازی را کاهش می دهد. به علاوه تکرار تغییرات ممکن است طراحی اولیه را خراب کند بنابراین انجام اصلاحات آینده تدریجاً گرانتر می شود.

مدل چرخه حیات تکاملی یک رویکرد تناوبی را برای تعمیر و نگهداری سیستم پیشنهاد می کند. این مدل ها ایده ای را مطرح می کنند که نیازهای سیستم را نمی توان در ابتدا به طور کامل جمع‌آوری کرد و فهمید. بنابراین سیستم در ساختارهایی توسعه پیدا خواهد کرد که هر کدام کامل و درست هستند و نیازهای ساختار قبلی را به وسیله یک فیدبک به کاربران رفع خواهد کرد.[36] شکل ۶ مدل افزایشی تکراری را نشان می دهد.



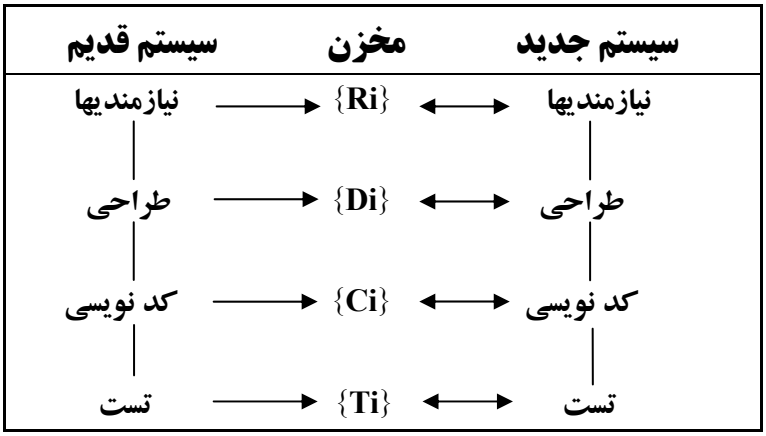
شکل ۶ – مدل افزایشی تکراری

ساختار جدید سیستم (تعمیر آن) با تجزیه و تحلیل نیازمندیهای سیستم موجود ، طراحی ، کد و تست مستندات شروع می گردد و با اصلاح بالاترین سطح اسناد که به وسیله تغییرات تحت تاثیر قرار می‌گیرد ، متاثر می شود و تغییرات به سمت یک مجموعه مستندات کامل گسترش می یابد.

به طور خلاصه در هر مرحله فرآیند تکاملی سیستم براساس یک تجزیه و تحلیل بر روی سیستم موجود طراحی مجدد می گردد. یک نکته مهم مدل افزایشی تکراری این است که مستندات با تغییر کد به روز رسانی می گردد. Visaggio [76] گزارش داده است که اطلاعاتی که از تکرار

آزمایشات کنترل شده به دست می آید ما را به مقایسه دو مدل تصحیح سریع[۲۶] و افزایشی تکـراری هدایت می کند و نشان می دهد که قابل تعمیر بودن سیستم به وسیله مدل تصحیح سریع ، زودتـر از بین می رود. آزمایشات همچنین نشان می دهند که سازمان ها خودشان را با مدل افزایـشی تکـراری تطبیق می دهند برای این که اصلاح تغییرات سریعتر از مدل تصحیح سریع اعمال می شود.

Basili [8] مدلی را پیشنهاد می کند که یک مدل قابل استفاده مجددکامل[۲۷] است که در شـکل ۷ مشاهده می نمایید.



شکل ۷ – مدل Full-Reuse

شکل ۷ تعمیر و نگهداری را به عنوان یک نمونه ویژه از توسعه نرم افزار متکی بر استفاده مجـدد را نشان می دهد. مدل استفاده مجدد کامل با تجزیه و تحلیل نیازمندیها آغاز می گردد و یـک سیـستم جدید را طراحی می کند و نیازمندیهای مناسب ، طراحی ، کد و تست را از نسخه های موجود قبلـی استفاده می نماید. این مهمترین تفاوت این مدل با مدل افزایشی تکراری است که با تجزیه و تحلیـل سیستم موجود شروع می گردد. حالت مرکزی مدل استفاده مجدد کامل این نظریه است که مخـزن مستندات و کامپوننتهای تعریف شده در نسخه های قبل از سیستم جاری ، در دیگر سیستمها نیـز در همان حوزه کاربرد قرار دارد. این مسئله استفاده مجدد را واضح و شفاف و مستند می سازد.

مدل افزایشی تکراری برای سیستم هایی که دارای دوره حیات طولانی هستند و زمان تولید آنها بیش از حد مجاز بوده است دنباله بسیار مناسبی است و آن تکامل سیستم را در چنین مسیری برای آسانتر شدن اصلاحات در آینده پیشنهاد می کند.

[26] quick - fix
[27] Full - reuse

مدل CMM : توابع تعمیر نرم افزار از کمیابی مدلهایی که مدیریت توسعه و بهبود آن را تسهیل نماید ، رنج می برند. مدل CMM ، یک مدل توسعه کیفیت فرآیند تعمیر نرم افزار است که بر پایه مدل CMM-SEI[28] که توسط Carnegie Mellon University بنا شده بود ، ارائه شده است که برای تکامل و بهبود فرآیند توسعه نرم افزار از آن استفاده می گردد. معماری مدل CMM محتوای نرم افزار را در فرآیند توسعه ، خصوصیات تغییر کرده یا گسترش یافته برای توابع تعمیر تقریباً حفظ می نماید. این ویژگیهای خاص برای تعمیر نرم افزار بر پایه تجربه و کار آزمودگی بر روی فرآیند تعمیر و نگهداری به دست آمده است. این ویژگیها فرآیند را به نواحی CMM که یک مسیر بهبود تکاملی برای بهبود تدریجی توابع نرم افزار است ، با توجه به نوع ساختار ارتقا می‌دهد.

مدل SMMMM : ساختار این مدل شبیه به مدل CMM طراحی شده است و در واقع متمم این مدل است.

مدل READABLE[29] : این مدل نمایش جدولی و اجرایی برنامه را که هم برای یادگیری و هم برای اداره کردن برنامه های کاربردی به کار می رود، را تولید می کند. هدف از ارائه این مدل ارائه یک روش برای ایجاد و تألیف برنامه های کاربردی ، به خصوص برنامه های وب است تا در فاز تعمیر و نگهداری یادگیریشان بهبود بخشیده شود. در این مدل راه حلی به عنوان یک روش اندیشیده شده است که به برنامه نویس کمک می کند تا با توجه به اطلاعات تعمیرکار درباره یک کلید یادگیری برنامه ، کمک نماید.

مدل Osborne : به واقعیت محیط تعمیر مربوط می شود. در نظریه Osborne مشکلات تکنیکی در طول تعمیر رخ می دهد و سبب ارتباطات ضعیف و کنترل بین مدیریت می گردد. مدل Osborne در چهار استراتژی زیر معرفی می گردد:

۱- نیازها و احتیاجات تعمیر ، نیاز دارند که شامل مشخصات تغییرات باشند.
۲- کیفیت یک برنامه نیاز دارد بر پایه کیفیت درخواست ها و احتیاجات بنا شود.
۳- استانداردهای متریک نیاز دارند به منظور شناسایی اهداف تعمیر با آن مواجه شوند.

---

[28] Capability Maturity Model of the Software Engineering Institute
[29] Readable , Executable , Augmentable Database-Linked Environment

۴- مدیران نیاز دارند که یک فیدبک برای بازنگری کارایی مدل تکراری افزایشی را فراهم نمایند تا بتوانند تغییراتی که سیستم در طول چرخه حیات نرم افزار در یک فرآیند تکراری دارد را مشاهده نمایند. این مدل از توسعه با تعمیر تطبیق پیدا می کند.

همه این مدلها توانایی و ضعف خاص خود را دارند. بنابراین معمولاً باید بیش از یک مدل را برای انجام تمام فعالیتهای تعمیر و نگهداری استفاده نمود. بهترین حالت ترکیب این مدل هاست البته زمانی که لازم باشد.

۵-۱- استفاده مجدد

اهداف استفاده مجدد در طول تعمیر نرم افزار باعث افزایش باروری سیستم ، کیفیت ، آسان سازی انتقال کدها ، کاهش زمان و تلاش برای عمل تعمیر و بهبود قابلیت تعمیر می‌گردد. (Biigerstaff et al [1989]) استفاده مجدد را این گونه تعریف می کند: **" استفاده مجدد ، از انواع مختلف اطلاعات یک سیستم برای سیستم مشابه دیگر به منظور تلاش برای توسعه یا تعمیر آن سیستم استفاده می کند."**

استفاده مجدد نرم افزار از فرآیند ، کارکنان و محصول گرفته می شود. **فرآیند** یک فعالیت یا عملی است که به وسیله ماشین یا شخص انجام می گیرد. یک متدلوژی می تواند مجدداً بر روی مشکلات کاربردهای متفاوت استفاده شود. طراحی شی گراء یک مثال کامل از استفاده مجدد در توسعه است. مدل COCOMO بوهم یک مثال دیگری از فرآیند استفاده مجدد است. COCOMO یک مدلی است که اجازه می دهد که عملیات تخمین هزینه ، تلاش و زمانبندی هنگامی که یک فعالیت توسعه نرم افزار برنامه ریزی می شود انجام گیرد. استفاده مجدد **کارکنان** شامل استفاده مجدد از اطلاعات افرادی است که با آن مسئله مواجه شده اند و بر پروژه های قبلی غلبه کردند و این اطلاعات را برای پروژه جدید به کار می بندیم. به عنوان مثال برای " یادگیری درس" از پروژه‌های قبلی استفاده می شود. برای بیشینه کردن سودمندی و تاثیر استفاده مجدد ، اطلاعات باید از نیازهایی به دست آید که برای استفاده مجدد از تحلیل حوزه ی آن حاصل شده باشد.

محصول استفاده مجدد شامل استفاده از پروژه هایی است که قبلاً ایجاد شده است. داده ، طراحی و برنامه ها همه محصولاتی هستند که می توانند دوباره استفاده شوند. قالبهای داده از قبیل XML می تواند به آسانی بین برنامه های کاربردی استفاده گردد. معماری و جزییات طراحی می تواند برای نقل و انتقال محصولات مشابه استفاده شود که می تواند باعث افزایش باروری سیستم و بهبود

کیفیت محصول گردد. برنامه استفاده مجدد از اجزای کد از قبیل ماژولها ، پکیج ها ، روالها ، توابع و روتین ها استفاده می نماید. اجزای برنامه می تواند به آسانی با هم ترکیب شده و یکپارچه گردند.

۵-۲- وظایف

وظایف تعمیر می تواند در ۵ گروه دسته بندی گردد. تجزیه و تحلیل/ جداسازی ، طراحی ، پیاده‌سازی ، تست و مستند سازی (Basili et al [1996]) وظیفه تجزیه و تحلیل/ جداسازی شامل آنالیز ضربه ای ، تجزیه و تحلیل هزینه موثر و جداسازی می باشد. آنالیز ضربه ای و تجزیه و تحلیل هزینه موثر شامل تحلیل پیاده سازی های متفاوت تکراری و مقایسه تاثیر آن روی زمانبندی ، هزینه و عملیات می باشد. طراحی شامل طراحی مجدد سیستم بر پایه درک تغییرات لازم می باشد. آن همچنین شامل مستندات نیمه رسمی شبیه به مرور مستندات پخش شده از سیستم می باشد. پیاده‌سازی شامل کد نویسی و تست واحد ها می باشد. کد نویسی و تست واحد مربوط به زمان صرف شده برای کد نویسی و تست تغییرات می باشد. این مرحله بیشتر به برنامه نویس مربوط می گردد. تست واحد معمولاً به صورت محلی بر روی کامپیوتر کاربر صورت می گیرد. عملیات تست شامل تستهای مجتمع سازی ، تست پذیرش و تست برگشت می باشد. تست مجتمع سازی به زمان صرف شده برای یکپارچه کردن اجزاء نرم افزار مربوط می گردد. در صورتی که تست پذیرش شامل بازبینی تغییرات سیستم برای بررسی توافق با نیازهای کاربر می باشد. تست پذیرش به وسیله کاربر نهایی برای اطمینان از این که تغییرات خواسته شده با موفقیت پیاده سازی شده است یا نه انجام می‌گیرد. مستند سازی شامل مستندات سیستم ، کاربر و دیگر مستندات می باشد. مستند سازی، فرآیند بسیار مهمی است به این دلیل که تغییرات انجام شده به اسناد تغییرات یا اصلاحات گذشته متکی است.

ابزارهای تعمیر نرم افزار ، یک محصول است که نگهدارنده نرم افزار در اجرای یک وظیفه از آنها استفاده می کند. (Takeng and Grubb [1996]) استفاده از ابزارها برای ساده سازی وظایف و افزایش کارایی و سودمندی تعمیر نرم افزار می باشد. چندین معیار برای انتخاب درست ابزار برای وظایف وجود دارد. این معیارها شامل توانایی ، ویژگی ها ، هزینه / سود ، پلتفرم ، زبانهای برنامه‌نویسی ، سهولت برای استفاده ، باز بودن معماری ، پایداری تولید کنندگان و فرهنگ سازمان می‌باشد. معیار توانایی ، تصمیم می گیرد که آیا ابزار قادر به انجام وظیفه است یا نه. همچنین آن

تصمیم می گیرد که یک متد می تواند از آغاز مکانیزه شود یا خیر. سپس ابزار مورد نیاز برای رسیدگی به آن کار را فراهم می نماید.

هر ابزار باید برای مزایایی که دارد در مقابل هزینه هایی که به جا می گذارد تحلیل گردد. مزیتهای یک ابزار نماینده کیفیت ، باروری ، حساسیت و کاهش هزینه می باشد. محیطی که ابزار روی آن اجرا می گردد پلتفرم نام دارد. این مسئله مهم است که ابزاری که انتخاب می کنیم از یک زبان پشتیبانی کند که یک استاندارد صنعتی است. ابزار باید شبیه حالتی باشد که کاربران با آن آشنا میگردند. ابزار باید توانایی این را داشته باشد که با ابزارهای فروشندگان دیگر بتوانند یکپارچه گردد. این مسئله هنگامی که یک ابزار نیاز دارد که با ابزارهای دیگر اجرا گردد نیاز می گردد. بازبودن معماری هنگامی که مشکلات تعمیر پیچیده می‌گردد یک وظیفه مهم را بر عهده دارد. بنابراین همیشه استفاده از یک ابزار کافی نیست و ممکن است نیاز باشد که چندین ابزار باهم کار کنند. این مسئله هم مهم است که اعتبار تولید کننده ابزار هم بررسی گردد. تولید کننده باید قادر به پشتیبانی از ابزار در آینده باشد. فاکتور مهم دیگر فرهنگ سازمان است. هر فرهنگ ، الگوی کاری خودش را دارد و باید در انتخاب ابزار به این مسئله توجه کرد.

ابزارهای انتخاب شده باید از درک برنامه و مهندسی معکوس ، عملیات تست ، مدیریت پیکربندی و مستند سازی پشتیبانی کنند. (Takang and Grubb [1996]) انتخاب یک ابزار مناسب می‌تواند باعث بهبود درک موارد پر اهمیت در پیاده سازی تغییرات گردد هنگامی که زمان زیادی برای مطالعه و درک سیستم صرف می شود. ابزارها برای مهندسی معکوس نیز اهداف مشابهی را انجام می‌دهند. تقسیم برنامه به برنامه نویس کمک می‌کند که فقط قسمتهایی از برنامه که به وسیله تغییرات تاثیر پیدا کرده است را انتخاب کند یا ببیند.

عملیات تست زمان زیادی در فرآیند تعمیر را به خود اختصاص می دهد و وظیفه طاقت فرسایی است. بنابراین آن می تواند بیشتر از ابزارهای دیگر مفید باشد. یک ابزار شبیه ساز تست به نگهدارنده سیستم کمک می کند که تاثیر تغییرات را در محیط شبیه سازی شده قبل از پیاده سازی تغییرات روی محیط واقعی مشاهده نماید. یک تولید کننده تستهای نمونه ، داده های تستی که برای تست توابع اصلاحی سیستم استفاده می‌گردد را تولید می نماید. مدیریت پیکربندی از ابزار مکانیزه شده بهره می برد. مدیریت پیکربندی و ابزار کنترل نسخه به نگهداری اشیاء سیستم نرم افزاری کمک می کند. یک سیستم کنترل منبع می تواند برای نگهداری تاریخچه نسخه فایل‌هایی که پشت سر هم هستند استفاده گردد و برنامه نویس می تواند از دنباله تغییرات فایل استفاده نماید.

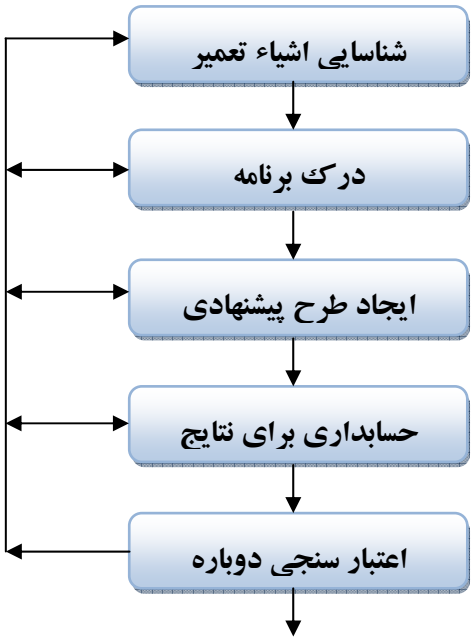
۵-۳- ابزارها و محصولات در دسترس از لحاظ تجاری

محصولات زیادی در فروشگاهها برای تعمیر نرم افزار در دسترس می باشد. یکی از این محصولات ابزارهای پیگیری خطا است که نقش مهمی را در فرآیند تعمیر بازی می کند. مثلاً Bugzilla که به وسیله موزیلا ساخته شده است. از دیگر محصولات پیگیری خطا می توان به هدایت کننده تست به وسیله Mercury Interactive ، رادار ابریشم به وسیله نرم افزار ، مدیریت SQA به وسیله نرم افزار Rational و مدیریت QA به وسیله Compuware ، ProTeus III Expert CMMS به وسیله تکنولوژی عقاب و غیره اشاره نمود.

محصولات خاصی برای زبان های برنامه نویسی java طراحی شده اند از قبیل CCFinder ، JAAT.(Kamjya etal [2001]) CCFinder کدهای Clone را در زبان جاوا شناسایی می‌کند و JAAT یک تحلیل مستعار را برای برنامه های جاوا اجرا می کند. برای زبان های C++ ابزاری وجود دارد که OCL نام دارد که یک دیباگ برنامه بر پایه گزارش است که ابزاری برای دیباگ برنامه های C++ با استفاده از تنظیم گزارشات در Object Constraint Language – OCL می باشد. (Hobatr and Malloy [2001]) خلاصه اینکه ابزارها باعث افزایش کارایی و راندمان می شوند و تعداد زیادی ابزار برای فرآیند تعمیر در بازار وجود دارد.

۶ – فرآیندها

تعدادی نویسنده مدلهای فرآیند را برای تعمیر نرم افزار پیشنهاد کردند. این مدلها تعمیر و نگهداری را به صورت ترتیبی از فعالیت های وابسته به هم یا فازها سازماندهی می کنند و ترتیبی که این فازها باید اجرا شوند را مشخص می نمایند. گاهی اوقات آنها همچنین پیشنهاد می‌کردند که قسمتهای قابل تحویل هر فاز باید برای فازهای بعدی فراهم باشد. به عنوان مثال یک فرآیند در شکل ۸ نشان داده شده است.[82]

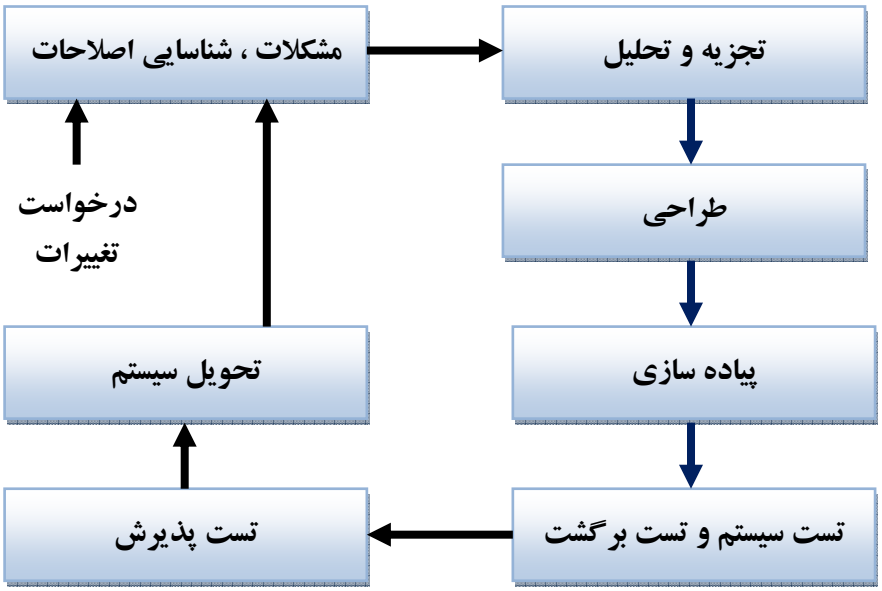


شکل ۸– یک مثال از فرآیند تعمیر و نگهداری نرم افزار

IEEE و ISO هر دو تعمیر نرم افزار را مورد خطاب قرار دادند. ابتدا با یک استاندارد خاص[41] و سپس به عنوان قسمتی از آن استاندارد روی چرخه حیات فرآیند ها .[44] دو قسمت بعد فرآیند تعمیر و نگهداری را به وسیله این دو سند شرح می دهند.

IEEE – 1219 – ۶,۱

استاندارد IEEE فرآیند تعمیر را در هفت فاز همان طور که در شکل ۹ مشاهده می کنید سازماندهی می نماید. به علاوه برای شناسایی فازها و ترتیب اجرای آنها برای هر فاز استاندارد قابل تحویل ، فعالیت های گروه بندی شده ، فرآیندهای پشتیبانی شده و مرتبط به هم و کنترل آنها نشان داده شده است.



شکل ۹ – فرآیند تعمیر و نگهداری از دیدگاه IEEE

۱- **مشکلات، شناسایی اصلاحات ، دسته بندی و اولویت :** این فازی است که درخواست تغییرات (MR[30]) به وسیله یک کاربر ، یک مشتری ، یک برنامه نویس یا مدیر که دسته‌بندی تعمیر را بر عهده دارد ناشی می گردد. (بخش ۳ را برای تعریف دسته بندی تعمیر و نگهداری مشاهده می نمایید) یک اولویت و هویت منحصر به فرد یک فاز همچنین شامل فعالیتهایی است که تعیین می کند که چه هنگامی یک درخواست پذیرفته یا رد می‌گردد و آن را به یک دسته اصلاحات زمانبندی شده برای پیاده سازی واگذار می کند.

۲- **تجزیه و تحلیل :** این فاز یک طرح مقدماتی را برای طراحی ، پیاده سازی ، تست و تحویل ایجاد می کند. تجزیه و تحلیل در دو سطح ارائه می گردد: امکان سنجی تجزیه و تحلیل و تجزیه و تحلیل جزییات. امکان سنجی تجزیه و تحلیل راه حل متناوبی است و تاثیرات و هزینه ها را تعیین می کند در حالی که تجزیه و تحلیل جزییات ، نیازمندیهای عمل اصلاح را مشخص می نماید. تجزیه و تحلیل یک استراتژی تست را ایجاد می نماید و طرح پیاده‌سازی شده را توسعه می دهد.

۳- **طراحی :** اصلاح سیستم در حقیقت در این فاز طراحی می گردد. این شامل استفاده از همه سیستم های جاری ، مستندات پروژه ، نرم افزار موجود ، پایگاه داده و خروجی تجزیه و تحلیل در این فاز می باشد. فعالیت ها شامل شناسایی تاثیر ماژول های نرم افزاری ، اصلاح مستندات ماژول های نرم افزاری ، ایجاد تست نمونه برای طراحی جدید و تست برگشت می باشد.

۴- **پیاده سازی :** این فاز شامل فعالیتهای کد نویسی و تست واحد ، مجتمع سازی کدهای اصلاح شده و تست برگشت ، تجزیه و تحلیل ریسک و بازنگری است. این فاز همچنین شامل یک تست آمادگی برای مرور سازماندهی یک سیستم قوی و تست برگشت می‌باشد.

۵- **تست سیستم/ برگشت :** این فازی است که تمام سیستم برای اطمینان از برآورده شدن نیازهای اصلی برنامه به اضافه اصلاحات ، تست می گردد. به علاوه برای تست توابع و رابطه ها ، این فاز نیز شامل تست برگشت برای تایید اعتبار این که هیچ خطای جدیدی به سیستم اضافه نشده است نیز می باشد. در نهایت این فاز عهده دار بازبینی آمادگی سیستم برای تست پذیرش است.

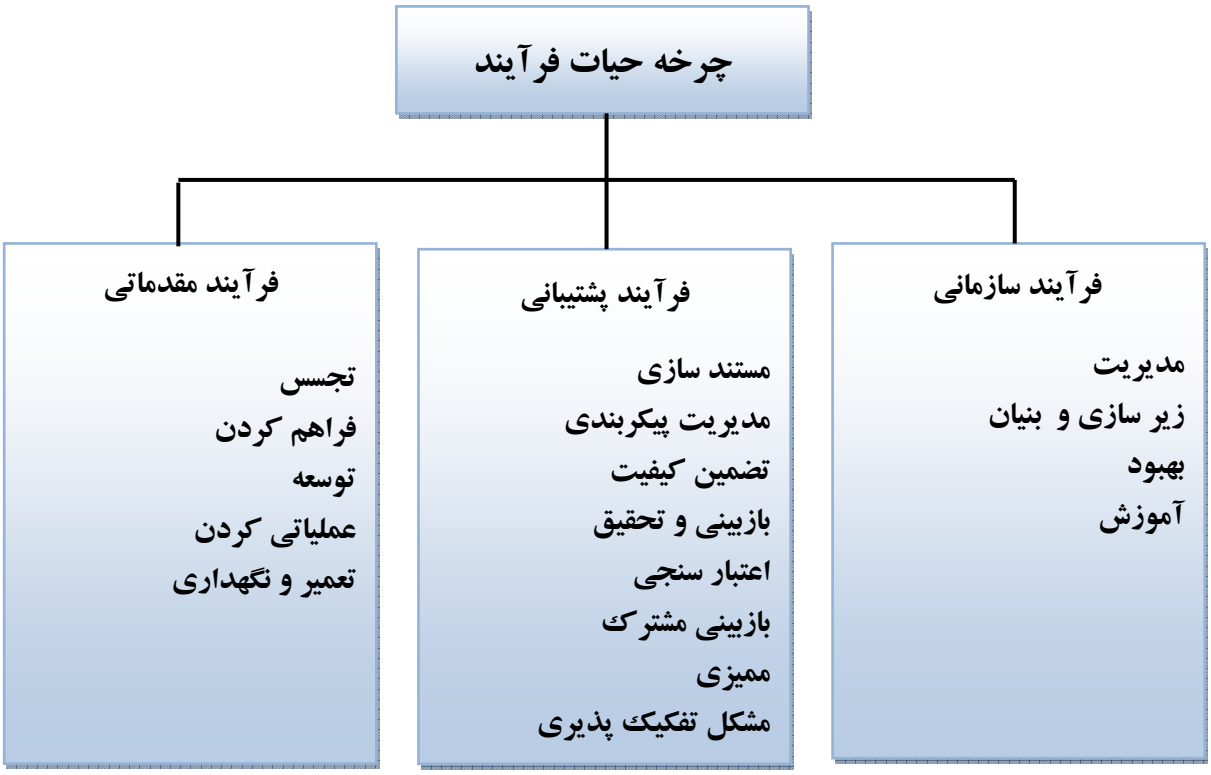
[30] Modification Request

۶- **تست پذیرش :** این مرحله به تست سیستم مجتمع شده کامل مربوط می گردد و درگیر با کاربران ، مشتریان یا یک شخص ثالثی که به وسیله مشتری معین شده است می باشد. تست پذیرش شامل تست توابع ، تست Interpoerability و تست برگشت می باشد.

۷- **تحویل :** این فازی است که اصلاح سیستم برای عملیات نصب و اجرا رها می گردد و آن شامل اعمال آگاهی دادن به کاربران ، اجرای عملیات نصب و آموزش سیستم و فراهم کردن بایگانی این نسخه برای پشتیبانی می باشد.

۶,۲ ISO – 12207 :

هنگامی که استاندارد IEEE – 1219 [4] تطابق خاصی با تعمیر نرم افزار داشت استاندارد ISO-12207 [44] بر پایه کل فرآیندهایی بود که شامل چرخه حیات نرم افزار است. این استاندارد ۱۷ فرآیند را در سه دسته مقدماتی ، پشتیبانی و سازماندهی گروه بندی می کند. فرآیندها به اجزاء اصلی فعالیت هایی تقسیم می شوند که هر کدام باعث پیشروی سازماندهی در وظایف می‌گردد. شکل ۱۰ فرآیندها و دسته بندیهای آن را در قالب سه گروه نشان می دهد. تعمیر و نگهداری یکی از ۵ فرآیند گروه مقدماتی است. به عبارت دیگر یکی از فرآیندهایی است که برای هدایت توابع اصلی در طول چرخه حیات سیستم و راه اندازی و بهره برداری از فرآیند های سازماندهی شده مهیا می شود. شکل ۱۱ فعالیت های تعمیر و نگهداری را نشان می دهد.



شکل ۱۰ – فرآیند چرخه حیات سیستم از دید **ISO**

شکل ۱۱ – فعالیت های **فرآیند** تعمیر و نگهداری

فرآیند پیاده سازی : این فعالیت شامل وظایف برنامه ریزی توسعه و روال های تعمیر و نگهداری نرم افزار، ایجاد روالها برای دریافت ، ضبط و پیگیری درخواست های تعمیر و بناکردن ساختاری از

رابطه ها به همراه فرآیند مدیریت پیکربندی است. فرآیند پیاده سازی در ابتدای چرخه حیات سیستم شروع می گردد. Pigoski [62] اظهار کرده است که برنامه ریزی تعمیر باید به صورت موازی با برنامه ریزی توسعه فراهم شود. این فعالیت شامل تعریف حوزه هایی برای تعمیر ، شناسایی و تجزیه و تحلیل متناوب است. آن همچنین شامل سازمان کارمندان سیستم ، تعمیر و نگهداری و تعیین مسئول و منابع است.

مشکلات و تجزیه و تحلیل اصلاحات : نخستین وظیفه این فعالیت این است که با تجزیه و تحلیل درخواست های تعمیر هماهنگ شود. وظایف دیگر آن عبارتند از : گزارش مشکلات یا درخواست اصلاح ، دسته بندی ، تعیین حوزه های آن در موقعیتهای اندازه ، هزینه ، زمان مورد نیاز و تعیین نقاط بحرانی آن. برای سازماندهی تعمیر و نگهداری این مسئله که مشکلات و مسائل تکرار شود و یا درخواست ها بازبینی گردد توصیه شده است. وظایف دیگر به توسعه و مستند سازی سیستم مربوط می گردد که به طور متناوب برای تغییرات پیاده سازی و انتخاب تنظیمات به عنوان یک مشخصه در قرارداد ، تکرار می گردد.

اصلاحات پیاده سازی : این فعالیت شامل شناسایی آیتم های مورد نیاز برای عمل اصلاح و فرآیند توسعه برای پیاده سازی واقعی تغییرات می باشد. نیازهای اضافی در فرآیند توسعه با روالهای تست برای اطمینان از این که نیازهای جدید یا اصلاح شده ،کامل و درست پیاده سازی شده اند هماهنگ می گردد و برای اینکه مطمئن شویم که موارد اصلاح نشده تحت تاثیر تغییرات قرار نگرفته اند.

پذیرش یا بازنگری تعمیر و نگهداری : وظیفه این فعالیت تعیین تمامیت سیستم اصلاح شده و اعلام پایان آن هنگامی که سازمان تعمیر و نگهداری مجوز لازم را برای اتمام رضایت بخش بودن درخواست تعمیر را به دست آورد، می باشد. چندین فرآیند پشتیبانی ممکن است با هم مشارکت داشته باشند به انضمام فرآیند های تضمین کیفیت ، تایید ، اعتبارسنجی و فرآیند بازنگری موارد مشترک.

کوچ و مهاجرت [۳۱]: این فعالیت هنگامی اتفاق می افتد که سیستم از یک محیط به محیط دیگر منتقل گردد. برای این منظور نیاز است که طرح و نقشه مهاجرت توسعه پیدا کند و کاربران یا مشتریان سیستم دلایلی که محیط قبلی نمی توانسته از آنها پشتیبانی کند را ببینند. وظایف دیگر با عملیات موازی محیطهای قدیم و جدید و عملیات های بعدی برای تشخیص تاثیر حرکت به محیط جدید مطابقت پیدا می کند.

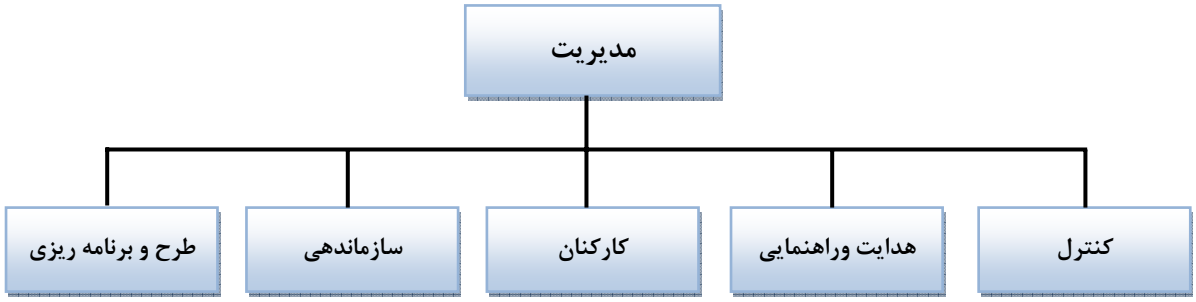
---

[31] Migration

از رده خارج کردن نرم افزار : آخرین فعالیت تعمیر و نگهداری شامل از رده خارج کردن سیستم نرم افزاری است و لازم است که طرح از رده خارج کردن سیستم توسعه پیدا کند و آن به اطلاع کاربران سیستم برسد.

۷ – مدیریت تعمیر و نگهداری

مدیریت **" فرآیند طراحی و نگهداری یک محیط در حالتهای انفرادی ، کار گروهی در گروهها و انجام موثر اهداف انتخابی"** می باشد.[79] تعمیر و نگهداری کلید این هدف است که یک هزینه موثر و کافی را برای پشتیبانی از سیستم نرم افزاری در تمام طول دوره حیات سیستم فراهم نماید. تعمیر و نگهداری با کیفیت و سودمندی هماهنگ است و سودمندی و بهره بری سیستم را ایجاب می نماید. تعدادی نویسنده [48,79,74] توافق کردند که تعمیر و نگهداری شامل پنج تابع مجزا از هم باشد که در شکل ۱۲ آنها را مشاهده می نمایید. این توابع عبارتند از : طرح و برنامه‌ریزی ، سازماندهی ، کارکنان ، هدایت و راهنمایی (گاهی اوقات هماهنگ کننده هم نامیده می شود) و کنترل.



شکل ۱۲ – توابع مدیریت پروژه

طرح و برنامه ریزی : شامل انتخاب ماموریت ها و اهداف از قبل تعیین شده و تعیین دنباله ای از فعالیتها برای انجام آنها می باشد. تعهد افراد و منابع و زمان بندی فعالیت ها در میان چندین فعالیت بحرانی در این تابع قرار دارد.

سازماندهی : عبارت است از مدیریت توابع برای بنا کردن یک ساختار مجتمع از وظایف افراد برای پر کردن سازمان. این همچنین موجب چیدن رابطه میان وظایف و اعطای مسئولیت ها و اختیارات لازم می گردد.

کارکنان : این تابع شامل پر کردن موقعیت ها در سازمان به وسیله انتخاب و آموزش افراد می باشد. دو فعالیت مهم این تابع ارزیابی و تخمین پروژه های کارکنان و مهیا کردن آنها برای توسعه اصلی است مانند بهبود آگاهی و دانش ، رفتار و روش و مهارت.

هدایت و راهنمایی : ایجاد یک محیط کاری و جوی که به شرکت افراد برای دست یابی به اهداف گروه و سازمان کمک خواهد کرد.

کنترل : اندازه گیری کارایی واقعی نمونه های منحرف شده از اهداف طرح و انجام فعالیت های اصلاحی روی آنها که این کار مستلزم پاداش و انضباط کارکنان پروژه می گردد.

استاندارد IEEE- 1219 [41] الگوی راهنمایی را برای تدارک یک طرح تعمیر و نگهداری نرم‌افزار بر پایه استاندارد خودش پیشنهاد می نماید. شکل ۱۳ طرح کلی این الگو را نمایش می دهد. Pigoski [62] بر روی مراقبت خاصی که باید برای گذر یک سیستم از تیم توسعه به سازمان تعمیر و نگهداری صورت گیرد تاکید می کند به این دلیل که این بخش یکی از بخشهای بسیار بحرانی چرخه حیات سیستم است.

1. Introduction
Describes the purpose, goals, and scope of the software maintenance effort; determines deviations from the standard.

2. References
Identifies the documents that pose constraints on the maintenance effort and any other supporting documents.

3. Definitions
Defines or references all terms required to understand the plan.

4. Software Maintenance Overview
Describes organization, scheduling priorities, resources, responsibilities, tools, techniques, and methods used in the maintenance process.

- 4.1 Organization
- 4.2 Scheduling Priorities
- 4.3 Resource Summary
- 4.4 Responsibilities
- 4.5 Tools, Techniques, and Methods

5. Software Maintenance Process
Identifies the actions to perform for each phase of the maintenance process; actions are to be defined in terms of input, output, process, and control.

- 5.1 Problem/modification identification/classification and prioritization
- 5.2 Analysis
- 5.3 Design
- 5.4 Implementation
- 5.5 System Testing
- 5.6 Acceptance Testing
- 5.7 Delivery

6. Software Maintenance Reporting Requirements
Describes how information will be collected and provided to members of the maintenance organization.

7. Software Maintenance Administrative Requirements
Describes the standards, practices and rules for anomaly resolution and reporting.

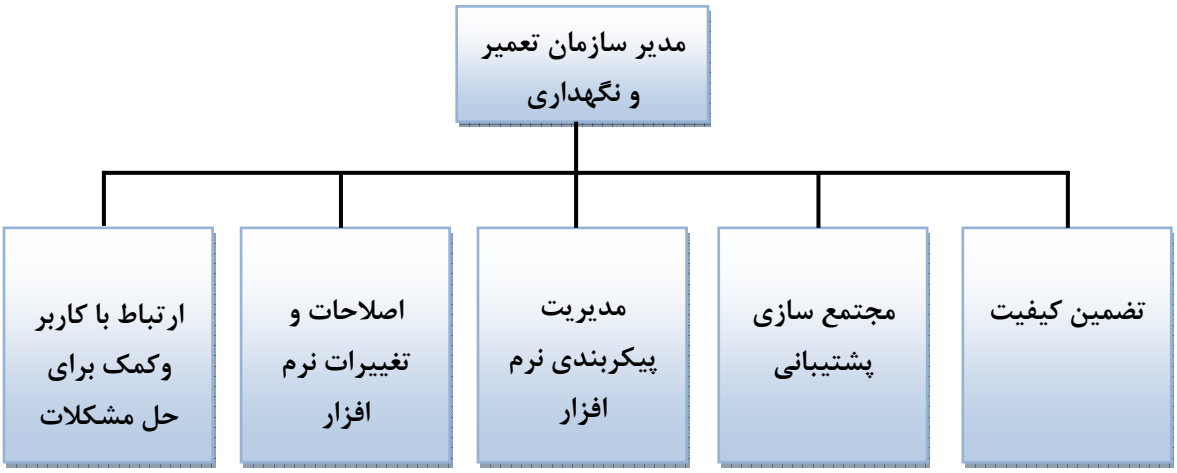
- 7.1 Anomaly Resolution and Reporting
- 7.2 Deviation Policy
- 7.3 Control Procedures
- 7.4 Standards, Practices, and Conventions
- 7.5 Performance Tracking
- 7.6 Quality Control of Plan

8. Software Maintenance Documentation Requirements
Describes the procedures to be followed in recording and presenting the outputs of the maintenance process.

شکل ۱۳- یک مثال از برنامه ریزی برای تعمیر و نگهداری نرم افزار

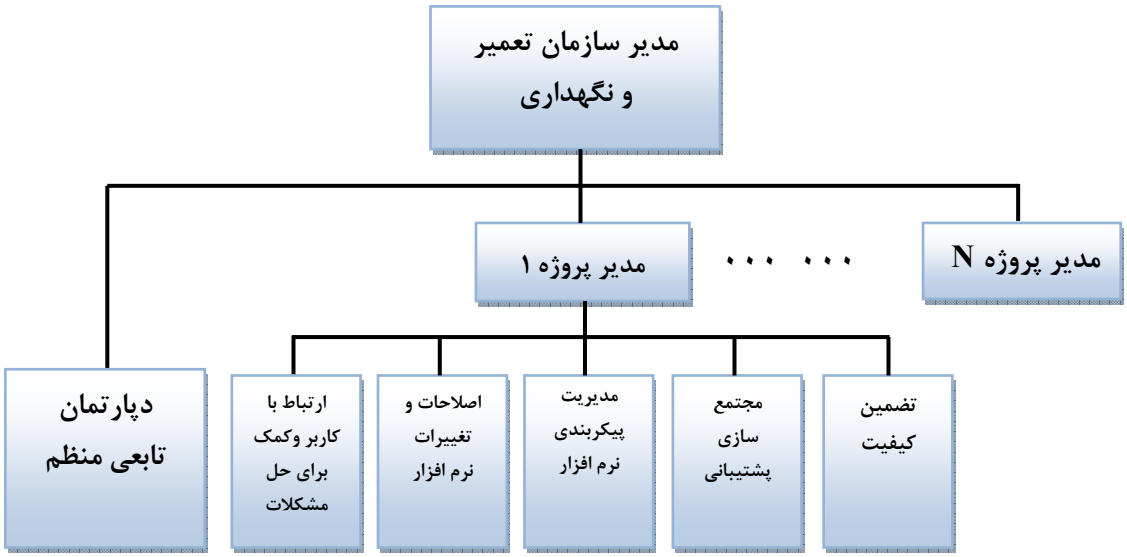
سازمان تعمیر نرم افزار می تواند بر پایه سه ساختار سازمانی متفاوت بنا شود که عبارتند از : تـابعی ، پروژه و ماتریس. [74,83]

سازمان تابعی: این سازمان دارای یک ساختار سلسله مراتبی است که در شـکل ۱۴ نـشان داده شـده است. سازمان تابعی مزایای سازمان دهی مرکزی را برای منابع خاص شـبیه هـم ارائـه مـی نمایـد. مهمترین ضعفی که مشکلات رابط دارند این است که ممکن است دارای راه حلهای سخت و مشکل باشند. هنگامی که در حوزه تابعی شامل بیش از یک برخورد پروژه باشیم ممکن اسـت موجـب رخ دادن اولویت های وابسته این پروژه ها در رقابت برای منابع شود. به علاوه فقدان یک نقطه مرکزی که مسئولیت کامل و توانایی و اختیار پروژه را در دست داشته باشد موجب یـک حـوزه تـابعی بـا اهمیت ، روی اهداف ویژه پروژه می گردد.



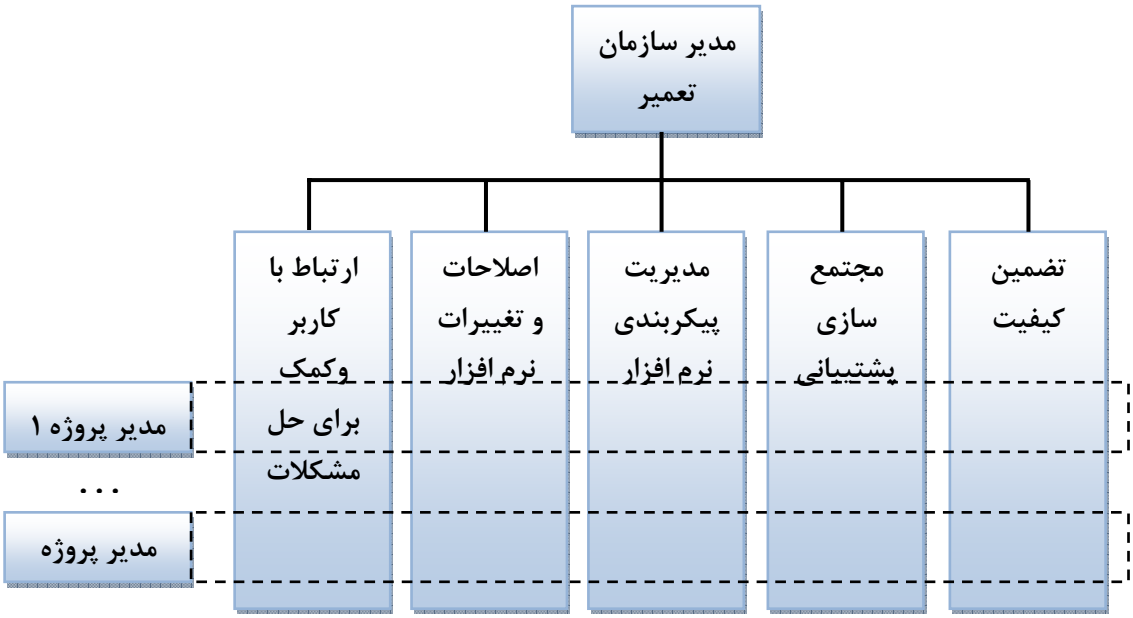
شکل ۱۴ – یک سازمان تابعی

سازمان پروژه : در واقع برعکس سازمان تابعی است (شکل ۱۵ را مـشاهده نماییـد) در ایـن نمونـه مدیر ، مسئولیت و اختیار کامل را برای هدایت پروژه در اختیار می گیرد. همه منابع مورد نیاز برای انجام اهداف پروژه به صورت مجزا از ساختار تابعی سازماندهی شده هستند. مدیر پروژه ممکن است منابع اضافی را از خارج از سازمان فراهم نماید. مزیت این نوع سازمان را مـی تـوان کنتـرل کامـل پروژه ، تصمیم گیری سریع و انگیزه بالای افراد پروژه دانست. ضعف آن نیز این اسـت کـه بـرای تشکیل تیم زمان شروعی وجود دارد و آن ممکن است باعث استفاده غیر موثر از منابع گردد.



شکل ۱۵ – یک سازمان پروژه ای

سازمان ماتریسی : این سازمان ترکیبی از دو سازمان تابعی و پروژه ای با هدف بیشینه کردن مزایا و کمینه کردن ضعف های دو نوع سازمان می باشد. شکل ۱۶ یک سازمان ماتریسی را نشان می‌دهد. استاندارد سلسله مراتب عمودی یک سازمان با سازمان افقی برای هر پروژه ترکیب شده است. نکته مهم این سازمان این است که تعادل بین اهداف حوزه تابعی و آن پروژه ها برقرار گردد. مشکل اصلی این است که هر فرد باید به دو مدیر پاسخگو باشد و این می تواند باعث برخورد منابع گردد. یک راه حل این است که وظایف ، مسئولیت و اختیارات مدیران تابعی و پروژه ای برای هر نوع تصمیم گیری مشخص گردد. همان طور که در شکل ۱۷ مشاهده می نمایید.



شکل ۱۶- یک سازمان ماتریسی

	مدیر	
تصمیم	**پروژه**	**تابعی**
تغییرات بودجه	پیشنهاد کردن	موافقت کردن
شرکت دادن منابع	پیشنهاد کردن	موافقت کردن
نیازهای تغییرات	تصمیم گرفتن	توصیه کردن
تغییرات طرح منتشر شده	تصمیم گرفتن	توصیه کردن

شکل ۱۷ – یک مثال از برخورد تصمیمات دو مدیر

یک مشکل عادی سازمان های تعمیر و نگهداری بی تجربگی پرسنل و کارکنان آنها است. Beath و Swanson [10] گزارش داده اند که ۲۵ درصد از افرادی که کار تعمیر و نگهداری سیستم را انجام می دهند دانشجویان هستند و بیش از ۶۱ درصد از آنها افراد جدید کارمزد هستند.
Pigoski [62] تایید کرده است که بین ۶۰ تا ۸۰ درصد از کارکنان تعمیر و نگهداری پرسنل جدید کارمزد هستند. تعمیر و نگهداری هنوز به وسیله تعداد زیادی از سازمان ها به عنوان یک پیامد غیر استراتژیک درک نشده است و این مسئله دلیل استفاده از دانشجویان و کارکنان کارمزد برای تعمیر و نگهداری را روشن می سازد. دو مشکلی که وجود دارد این است که بیشتر دانشگاهها تعمیر و نگهداری را آموزش نمی دهند و همچنین این مبحث به ندرت در مباحث و برنامه های آموزشی آمده است. برای مثال تعمیر نرم افزار در بین ۲۲ درس مهندسی نرم افزار در برنامه آموزشی این رشته لیست نشده است.[61] فقدان ارزیابی پرسنل تعمیر و نگهداری مشکلات مدیریتی حادی را به وجود می آورد. اولاً تغییر و تبدیلهای زیاد و ثانیاً: روحیه و دلگرمی پایین کارکنان.

۸ – مهندسی معکوس

مهندسی معکوس این گونه تعریف می گردد : **" فرآیند تجزیه و تحلیل سیستم برای شناسایی اجزا و ارتباطات و روابط آنها با هم و ایجاد نمایش سیستم در دیگر اشکال یا در یک سطح انتزاعی بالاتر می باشد."** [29] بنابراین مهندسی معکوس یک فرآیند آزمایش است و فرآیندی نیست که سیستم را تغییر دهد و بنابراین درگیر تغییر سیستم نرم افزاری در هنگام آزمایش نمی گردد.
اگرچه مهندسی معکوس نرم افزار در مرحله تعمیر و نگهداری آن سازماندهی می گردد ولی آن برای تعداد زیادی از حوزه های مشکلات کاربرد دارد. Chikosfsky و Cross II [29] شش

هدف را برای مهندسی معکوس شناسایی کرده اند که عبارتند از : تهیه یک نسخه کپی سیستم به همراه پیچیدگیهای آن ، تولید دیدهای متناوب ، پوشاندن اطلاعات گم شده ، شناخت تاثیرات ، ترکیب انتزاع های سطوح بالاتر و آسان سازی استفاده مجدد.

استاندارد IEEE – 1219 [41] توصیه کرده است که مهندسی معکوس به عنوان کلید تکنولوژی پشتیبان متکی بر سیستم است که کد منبع را به عنوان تنها نمایش معتبر در اختیار دارد. به عنوان مثال حوزه های مشکلاتی که مهندسی معکوس ، با موفقیت آنها را به کار می گیرد عبارتند از : شناسایی قابل استفاده مجدد بودن دارایی ها [23] ، پیدا کردن اشیاء در برنامه های رویه ای [24,37] ، کشف معماری [49] ، محرک ادراکی مدل های داده [18] ، شناسایی نسخه برداری ها و تکرار ها [47] ، انتقال برنامه های دودویی در کد منبع [30] ، تجدید کردن رابطهای کاربردی [57] ، برنامه های ترتیب موازی [17] ، ترجمه [22] ، فرآیند انتقال برنامه از سیستم اصلی به سیستم های کوچکتر و ارزانتر مانند Client – Server (Downsizing) [74] ،مهاجرت [27] و بسته بندی کدهای موروثی[72].

مهندسی معکوس اصولی دارد که همچنین برای فرآیندهای تجاری مهندسی مجدد و برای ایجاد یک مدل از تشکیلات موجود نیز به کار گرفته می شود.[45] مهندسی معکوس به عنوان یک فرآیند مشکل برای تعریف کردن است زیرا آن یک رشته جدید و با رشد سریع است. نقل قول است که مهندسی معکوس به عنوان دو مرحله فرآیند دیده شده است. استخراج اطلاعات و تجرید (مجرد سازی). استخراج اطلاعات محصولات سیستم ، در ابتدا کد منبع را برای به دست آوردن سطرهای اطلاعات تجزیه و تحلیل می کند بنابراین تجرید اطلاعات ، اسناد و دیدهای مربوط به کاربر را ایجاد می کند.

۹ – مهندسی مجدد

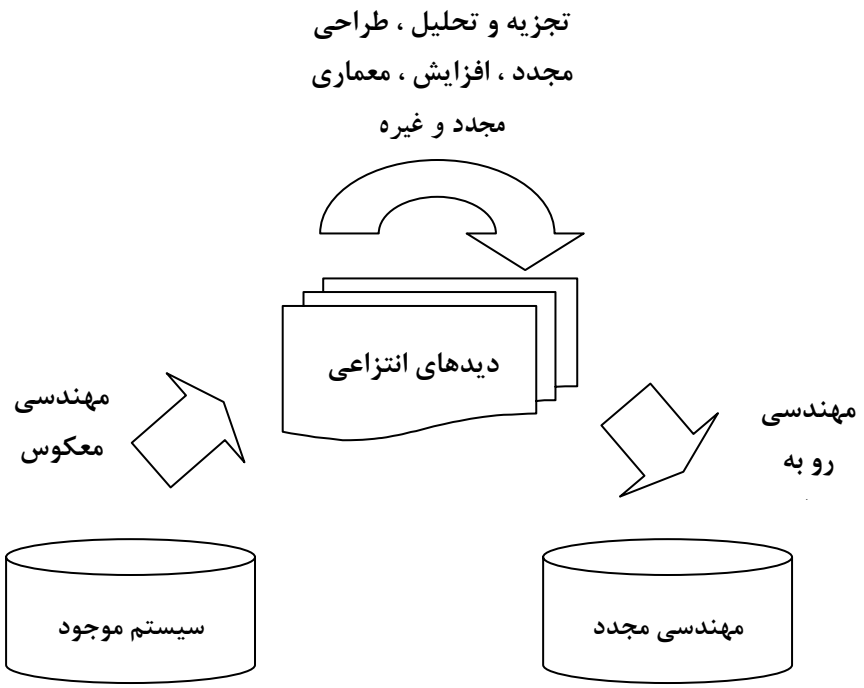
بکارگیری مهندسی مجدد بر روی یک سیستم نرم افزاری به منظور درک بهتر و نگهداری طولانی‌تر آن صورت می گیرد که به وسیله انجمن تعمیر پذیرفته می شود. Cross II و Chikofsky [29] در مقاله ای مهندسی مجدد را این گونه تعریف کردند : **" آزمایش و دگرگونی یک سیستم برای به هم پیوستن و پیاده سازی آن در فرم جدید ".** همان مقاله نشان می دهد که نوسازی[۳۲] و احیا[۳۳] به

[32] Renovation
[33] Reclamation

عنوان دو واژه مترادف هستند. تجدید[۳۴] واژه دیگری است که برای این منظور به کار می رود. [5]
Arnold تعریف بسیار جامع زیر را برای مهندسی مجدد ارائه می کند:

"مهندسی مجدد نرم افزار هر فعالیتی است که باعث درک نرم افزار گردد یا باعث فراهم کردن یا بهبود خود نرم افزار شود که معمولاً برای افزایش قابلیت تعمیر ، قابلیت استفاده مجدد یا قابلیت استنتاج کردن می باشد".

این بدیهی است که مهندسی مجدد مستلزم تعدادی از اشکال مهندسی معکوس برای ایجاد چندین دید مجرد از سیستم می گردد و نیز شامل احیاء این دیدهای انتزاعی است که به وسیله فعالیت های مهندسی رو به جلو برای درک سیستم در فرم جدید استفاده می گردد. این فرآیند در شکل ۱۸ توضیح داده شده است. وقوع مرحله مهندسی معکوس نقش مهندسی مجدد را از ساختار مجدد آن متمایز می نماید و همچنین باعث تبدیل یک محصول از یک شکل به شکل دیگر در همان سطح تجرید می گردد. [29]



شکل ۱۸ – مهندسی معکوس و مهندسی مجدد

Arnold [5] هفت دلیل اصلی را برای اثبات مهندسی مجدد بیان می کند:

۱- مهندسی مجدد می تواند به کاهش ریسک تکامل یک سازمان کمک کند.

[34] Renewal

۲- مهندسی مجدد می تواند به سازمان کمک کند تا مبلغ سرمایه گذاری شده در نرم افزار را دوباره به دست بیاورد.

۳- مهندسی مجدد می تواند به آسانتر شدن نرم افزار برای تغییر کمک کند.

۴- مهندسی مجدد یک تجارت بزرگ است.

۵- مهندسی مجدد قابلیت توسعه ابزار CASE را دارد.

۶- مهندسی مجدد یک عامل برای خودکار کردن تعمیر نرم افزار به شمار می رود.

۷- مهندسی مجدد به عنوان یک سازمان دهنده برای به کار بستن تکنیکهای هوش مصنوعی برای حل مشکلات مهندسی مجدد نرم افزار است.

استاندارد IEEE ۱۲۱۹ [41] این مسئله را یادآوری می کند که مهندسی مجدد به تنهایی نمی تواند باعث احیاء سیستم شود اما آن می تواند موارد قابل استفاده مجدد را برای توسعه های آینده فراهم کند که شامل فریم ورکی برای محیط شی گراء است.

مهندسی مجدد نرم افزار یک فرآیند پیچیده است که فقط توسط ابزار مهندسی مجدد پشتیبانی می‌گردد و به طور کامل خودکار نمی باشد. این دلیل خوبی برای مداخله انسان در پروژه های مهندسی مجدد است. ابزارهای مهندسی مجدد می تواند به فراهم کردن انتقال سیستم به محیط تعمیر جدید کمک کند برای مثال برای استقرار در یک منبع . اما آنها نمی توانند در چنین محیطهایی که مسیر بهینه در امتداد مهاجرت سیستم به سمت آن نیست تعریف شوند. فعالیت هایی وجود دارند که فقط انسان می تواند آنها را شروع کند. دیگر مشکلات ابزارهای مهندسی مجدد در حاشیه ابزارهای ایجاد تست کافی برای رسیدن به انتهای تولید مهندسی مجدد که کاملاً هم ارز با سیستم اصلی هستند خلاصه می شود. البته این مسئله شامل چندین بار چک کردن دستی جزیی است زیرا خیلی به ندرت پیش می آید که یک کاربرد مهندسی معکوس بدون توابع موجود تغییرات ، شروع گردد و توابع جدید آغازین اضافه شود. برای موفقیت در مهندسی مجدد نرم افزار نیاز به خرید یک یا چندین ابزار مهندسی مجدد است.

۱۰ – سیستم های موروثی

یک بخشی که هزینه بالایی را در تعمیر نرم افزار دارد ، سیستم های موروثی است. سیستم هایی وجود دارد که از توسعه آنها بیش از ۲۰ تا ۳۰ سال (یا حتی بیشتر) به دلیل درخواست های توسعه و رشد برای تولیدات و سرویس های جدید ، می گذرد. آنها به طور نمونه در محیط Mainframe بدون استفاده از تکنیکهای توسعه استاندارد و زبانهای برنامه نویسی منسوخ شده قرار دارند. ساختار

آنها به دلیل تاریخچه طولانی تغییرات و تطابق ها و وجود مستندات متناقض و نبود دنباله ای از تست های در دسترس ، به هم ریخته شده است. با این وجود از این سیستم های با وضع وخامت بار استفاده می شود. بیشتر سیستم های موروثی از ترابایتها داده فعال و موثر نگهداری می کنند) و مقدار زیادی از دانش و تجربیات حوزه کاربردی خود را کپسوله می کنند. گاهی اوقات کدهای موروثی فقط در حوزه دانش و قوانین تجاری ضبط شده ، قرار دارند و این مسئله مستلزم این است که یک سیستم جایگزین جدید ممکن است بر دانشی که در سیستمهای قدیمی بسته بندی شده است تکیه کند. خلاصه اینکه سیستمهای موروثی به عنوان **" سیستمهای بزرگی هستند که ما نمی‌دانیم که چگونه از عهده آنها بربیاییم ولی وجود آن سیستم ها برای سازمان ما امری حیاتی هستند"** [14] همین طور Brodie و Stonebraker [20] سیستم موروثی را این گونه تعریف می کنند: **" سیستم اطلاعاتی مهمی که در برابر تغییرات ایستادگی می کند و ارزیابی آن برای مواجه شدن با تغییرات جدید و دائمی به دلیل نیازهای تجاری جدید."**

چندین حالت برای مدیریت سیستمهای موروثی در دسترس است که این راه حلها عبارتند از :

۱- دور انداختن سیستم موروثی و ساختن یک سیستم جایگزین ۲- ثابت کردن سیستم و استفاده از آن به عنوان یک مولفه از سیستم جدید بزرگتر ۳- ادامه دادن تعمیر سیستم برای یک دوره زمانی دیگر ۴- اصلاح سیستم برای دادن یک حیات دوباره به سیستم.[15]

تغییرات ممکن است در حوزه ساده سازی سیستم باشد. (کاهش اندازه و پیچیدگی) یا یک نگهداری پیشگیرانه باشد. (مستند سازی دوباره ، ایجاد ساختار مجدد و مهندسی مجدد) یا فرآیندهای غیرعادی برای نگهداری تطبیقی باشد (اصلاح رابط ، بسته بندی و مهاجرت). این موارد ممکن است برای دیگر موارد تکرار نشود و تصمیم گیری برای یک هدف یا ترکیب اهداف ، بیشتر شایسته سیستمهای موروثی خاص است که باید یک برآورد تکنیکی و ارزش تجاری سیستم برای آن صورت گیرد. چهار فاکتور برای موفقیت آمیز بودن استفاده از سیستم موروثی و مشخص کردن تصمیمات عبارتند از: کهنگی[۳۵] ، افت[۳۶] سیستم ، انحلال پذیری[۳۷] و ارزش تجاری.

کهنگی : این فاکتور سیستم را به دلیل پیشرفت نرم افزار مهندسی داده و تکامل پلتفرم های سخت‌افزار و نرم افزار اندازه گیری می نماید. همچنین کهنگی شامل هزینه ای است که از نتیجه عدم به کاری مزیت متدهای جدید و تکنولوژیهایی که هزینه نگهداری را کاهش می دهند نیز حاصل می گردد.

---

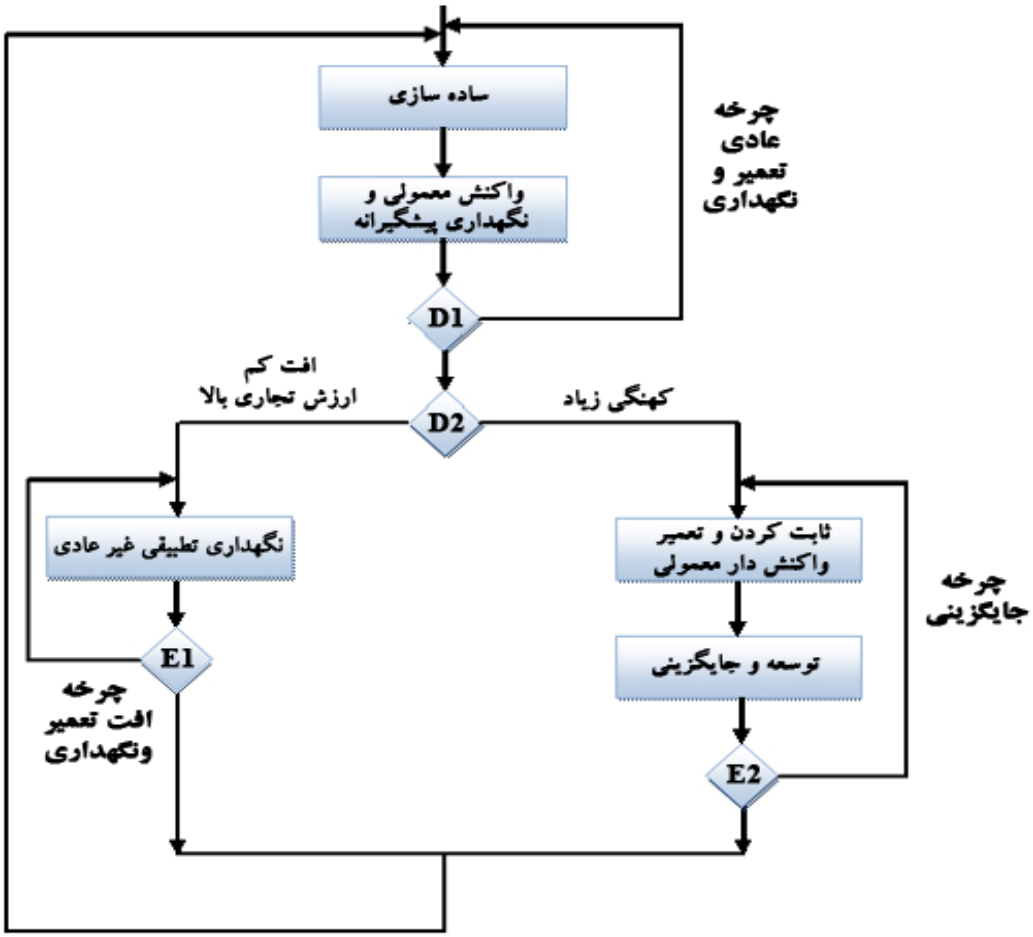
[35] Obsolescence
[36] Deterioration
[37] Decomposability

افت سیستم : اندازه گیری کاهش نگاهداشت پذیری سیستم (فقدان قابلیت تجزیه و تحلیل ، قابلیت اصلاح ، پایداری ، قابلیت تست و غیره) که به علت تعمیر سیستم در دوره حیات آن اتفاق می افتد. افت سیستم مستقیماً بر روی هزینه نگهداری آن تاثیر می گذارد.

انحلال پذیری : اندازه گیری قابلیت شناسایی و استقلال اجزای اصلی یک سیستم. همه سیستمها می‌توانند با داشتن سه مولفه مطرح شوند : رابطه ها ، حوزه توابع و سرویس مدیریت داده. انحلال پذیری سیستم نشان می دهد که مولفه های سیستم چگونه به خوبی معماری آن را منعکس می کنند.

ارزش تجاری : عبارت است از اندازه گیری پیچیدگی فرآیند تجاری و قوانین سیستم یا مولفه های آن و پیاده سازی و ارتباط آنها برای به دست آوردن تاثیر آن در عملیات تجاری.

منبع [25] توضیح می دهد که چگونه این فاکتورها می تواند اندازه گیری شود و در مدل چرخه حیات سیستم برای سیستم های موروثی معرفی گردد که از این فاکتورها برای انتقال تصمیمات استفاده می شود. شکل ۱۹ یک بازنگری بر چرخه حیات سیستم را نشان می دهد. این مدل اهمیت این امر که یک سیستم به وسیله چرخه تعمیر دائماً تحت تعمیر قرار دارد را نشان می دهد. در شکل همچنین سه تا از چرخه هایی که به تعمیر عادی و غیر عادی و جایگزین مربوط می شود را پررنگتر نشان می‌دهد. تعمیر غیر عادی با تعمیر عادی از لحاظ وسعت اصلاح و تاثیر آن در فرآیندهای تجاری متضمن آن دارای اختلاف می باشد. سیستمهای جایگزین و جدید وارد چرخه تعمیر عادی می شوند. به هر حال تصمیمات روی سیستم در حال اجرا نیاز به وارد شدن به چرخه تعمیر غیر عادی دارد یا یک چرخه جایگزین (به نقاط D1 و D2 دقت نمایید) نیاز دارد که سیستم بر پایه ۴ فاکتور بحث شده در بالا تعیین شود. بازگشت به چرخه حیات عادی (به نقاط E1 و E2 دقت نمایید) مستلزم ارزیابی پیشرفت / تکامل فرآیند برنامه ریزی شده است.



شکل ۱۹ – چرخه حیات سیستم برای سیستم موروثی

۱۱– نتیجه

این مقاله یک دید بالایی بر تعمیر نرم افزار داشت و آن مشکلات استراتژیک و راه حل های در دسترس برای آن بود. موضوع های این مقاله نشان داد که راه حلهای تکنیکی و مدیرانه ای وجود دارد که می تواند از کاربردهای استانداردهای سطح بالای مهندسی در تعمیر نرم افزار پشتیبانی شود. البته مشکلاتی هم وجود دارد و اساسی هم هستند و تحقیقاتی را برای به دست آوردن فهم بهتر تعمیر نرم افزار و پیدا کردن راه حل بهتر به کار می بندد.

امروزه سیستم های نرم افزاری که طراحی و ساخته می شوند خیلی زیاد تغییر می کنند و این به طور حتم یکی از مهمترین تاثیرات روی تعمیر نرم افزار در آینده است. تکنولوژی شی ، ابزارهای نرم افزاری و تکنولوژی هایی که از کار گروهی بر روی یک پروژه پشتیبانی می کنند ، پرداخت هزینه به شرکت های دیگر برای فراهم کردن سرویسهایی که توسط کارکنان آنها انجام می شود.

(OutSourcing) و سیستم های قابل افزایش کاربر ، یک مثال کوچکی از نواحی است که تحت تاثیر تعمیر نرم افزار قرار دارد.

تکنولوژی شی گرا در سالهای اخیر دارای محبوبیت زیادی شده است و رشد فزاینده ای داشته است و اکثر سیستم های جدید معمولاً توسعه خود را با یک هدف شی گرا شروع می کنند. در میان دلایل اصلی برای استفاده از تکنولوژی شی گرا ، افزایش قابلیت اصلاح مطرح می باشد و از این رو تعمیر آسانتر می گردد. این مسئله از میان مفاهیمی از قبیل کلاسها، مخفی کردن اطلاعات ، ارث‌بری ، چندریختی و مقید سازی پویا ناشی می گردد.ولی اطلاعات کافی که تاثیر تکنولوژی شی‌گرا را بر روی تعمیر نرم افزار نشان دهد وجود ندارد.[38,68]

Wilde و Huitt [80] بر روی تعدادی از مشکلاتی که ممکن است در تعمیر نرم افزارهایی که با استفاده از تکنولوژی شی گرا توسعه یابند و برای پشتیبانی از ابزارهای ممکن توصیه شده ، بحث می‌کنند. در میان مشکلات شناخته شده این امر که ارث بری ممکن است موجب مشکلتر شدن ، پیداکردن و تجزیه و تحلیل آن در میان کلاسهای به هم وابسته شود قابل توجه است. [9,33] و ممکن است موجب افزایش دوباره کاری شود.[53] همچنین تغییرات ممکن است باعث مشکلتر شدن پیاده سازی نرم افزارهای شی گرا در مقایسه با نرم افزارهای رویه ای شود. بنابراین توسعه شی گرا باعث بروز رسانی تغییرات کمتر می گردد.

یک سیستم نرم افزاری از لحاظ اندازه ، پیچیدگی و سن ، رشد می کند. تعمیر و نگهداری و تکامل آن به وسیله یک وظیفه مشخص برای نیاز به ترکیب تاثیرات گروهی از مهندسین نرم افزار متوقف می گردد. کار روزانه این گروههای مهندسی نرم افزار تولید بدنه ای برای اطلاعات مشترک ، تجربیات و تخصص هاست که نسبتاً پایه و اساس رشد سیستم درگذشت زمان است و برای کاهش تغییرات ، خطاها را در عملیات تعمیر و نگهداری معرفی می کنند. هر زمان که به این توافق میرسیم که درک مناسبی از سیستم در دسترس نیست مهندسین نرم افزار آن را به عنوان بخش (مقدماتی) از عملیات تعمیر و نگهداری که فعالیت هزینه بر و زمان بری است را توسعه می دهند. آمار موجود نشان می دهد که ۲/۳ زمان مهندسی نرم افزار در یک تیم تعمیر صرف جستجو در کدها و مستندات موجود برای کشف (مجدد) و فرآیند اطلاعاتی که شاید قبلاً چندین بار در طول دوره حیات سیستم از آن گرفته شده است می گردد. [13] با این وجود این مسئله مهم است که بدانید این اطلاعات به ندرت از طریق روش های منطقی ثبت می شوند. معمولاً آن در ذهن مهندسین نرم افزار قرار دارد و هنگامی که مهندسین کار (یا وظیفه) خود را تغییر دادند فراموش می شود. بنابراین این یکی از عوامل بهبود سودمندی تیم تعمیر و نگهداری است.

مسلماً دید متداول تعمیر نرم افزار یک فعالیت بعد از تحویل است که به ۷۰ سال قبل بر می گردد و شدیداً وابسته به چرخه مدل آبشاری است. با پدیدار شدن مدلهای چرخه حیات تکاملی و تکراری نشان دادیم که تعمیر یک فعالیت قبل و بعد از تحویل سیستم است. بنابراین هنوز یک نظریه گسترده برای تعمیر نرم افزار وجود دارد که همه آنها درباره رفع خطاها یا اشتباهات است. این مسئله نشان می دهد که چرا چندین نویسنده ترجیح دادند که از واژه تکامل نرم افزار به جای تعمیر غیر اصلاحی استفاده نمایند. اخیراً Bennett و Rajlich [16] یک مدل مرحله ای را در چندین مسیر مفید پیشنهاد کردند. آنها توسعه اولیه را حذف کردند و یک مرحله تکامل صریح را اضافه نمودند. در طول پیشرفت توابع و قابلیتهای آنها ملاقات با کاربران و دریافت نظرات آنها نیاز می‌گردد. تکامل به وسیله یک مرحله سرویس دهی به موضوعاتی از سیستم که آسیب دیده اند تا تعمیر شوند دنبال می‌گردد و سپس سیستم به سمت فاز خروجی حرکت می کند و در نهایت به مرحله Close - Down می رود.

منابع :

[A] Software Maintenance, Gerardo Canfora and Aniello Cimitile
[B] Software Maintenance As Part of the Software Life Cycle , Prepared by: Kagan Erdil ,Emily Finn, Kevin Keating, Jay Meattle, Sunyoung Park, Deborah Yoon , 2003
[C] Software Maintenance Costs , Ó Jussi Koskinen
[D] Software Maintenance Maturity Model (SMmm)