فرایند نرم افزار شخصی

Personal Software Process

مهدی عیوضی
MahdiEivazi.40021160028@gmail.com
دانشکده برق و کامپیوتر
دانشگاه کاشان

پاییز ۱۴۰۲

چکیده

فرآیند نرم‌افزار شخصی (PSP) به عنوان یک روش اساسی در حوزه شیوه‌های توسعه نرم افـزار فـردی اسـت. ایـن مقاله کاوش عمیقی از اصول PSP، کاربرد آن در توسعه نرم‌افزار، مزایای مرتبط، و چالش‌های بالقوه ارائه می‌دهد.
اهداف این سند دو مورد است: اول، توضیح اجزای اصلی فرآیند نرم‌افزار شخصـی، شـامل مراحـل، تکنیـک هـا و ابزارهای مورد استفاده توسعه دهندگان. دوم، تجزیه و تحلیل مفاهیم عملی آن از طریق مطالعات موردی و مقایسه با سایر روش‌ها. این مقاله با معرفی ماهیت فرآیند نرم‌افزار شخصی و اهمیت آن در تقویت کیفیت و کارایی در توسعه نرم‌افزار در سطح شخصی آغاز می شود. این مقاله به تشریح مراحل مختلف فرآیند نرم‌افزار شخصی ادامه می دهد و بر اهمیت برنامه ریزی، ردیابی و اندازه گیری در افزایش بهره وری و کیفیت کد تاکید می کند. علاوه بر ایـن، ایـن سند مطالعات موردی در دنیای واقعی را ارائـه می‌کنـد کـه اجـرای موفقیـت‌آمیز فرآینـد نـرم‌افـزار شخصـی را در محیط‌های توسعه نرم‌افزار متنوع نشان می‌دهد. علاوه بر این، بینش هایی را در مورد چالش هایی که هنگام پـذیرش فرآیند نرم‌افزار شخصی با آن مواجه می شوند، همراه با استراتژی هایی برای کاهش این موانـع ارائـه مـی دهـد. در نتیجه، این مقاله بر اهمیت اساسی درک و ادغام فرآیند نرم‌افزار شخصی در شیوه‌های توسعه نرم‌افـزار فـردی تأکیـد می‌کند، و پتانسیل آن را برای افزایش قابل توجه کیفیت کد، بهره‌وری توسعه‌دهنده و قابلیت اطمینان کلـی نرم‌افـزار برجسته می‌کند.

کلیدواژه‌ها:فرایند توسعه شخصی (PSP)–فرایند توسعه تیمی (tsp)–فاز پس از مرگ (postmortem)
خط کد(LOC)– **تخمین مبتنی بر پروکسی**(PROBE)

فهرست مطالب

۱. مقدمه

مهندسی نرم افزار یکی از بزرگترین و تاثیرگذارترین صنایع در جامعه مدرن است. از برنامه‌های محاسباتی اولیه که فقط توسط سازمان‌های دولتی و اتاق‌های فکر دانشگاه استفاده می‌شوند، به برنامه‌های پیچیده‌ای که در هر جنبه‌ای از زندگی مدرن نفوذ می‌کنند، تکامل یافته است. صنایع بانکداری، مخابرات، مسافرت، پزشکی، سرگرمی و حتی کشاورزی برای کار به شدت به نرم افزار متکی هستند. نرم افزار حتی پیش پا افتاده ترین جنبه های زندگی ما را تحت تأثیر قرار می دهد، از خرید مواد غذایی گرفته تا شستن بار یا پر کردن مخازن سوخت خودروهایمان با بنزین و ... . با این حال، علی رغم نفوذ فراگیر آن، مهندسی نرم افزار یک رشته نسبتاً جوان است. اصطلاح «مهندسی نرم‌افزار» تنها از اواخر دهه ۱۹۶۰، پس از معرفی آن در عنوان کنفرانس کمیته علمی ناتو در گارمیش، آلمان، مورد استفاده عمومی قرار گرفت. یکی از انتقادات مکرر به حرفه نرم افزار، کیفیت پایین محصولاتی است که تولید می کند. این مشکل به دلایل زیادی نسبت داده شده است، از نحوه آموزش متخصصان نرم افزار گرفته تا مشکلات کلی ذاتی یک حرفه جوان. مقاله‌ای در دایره‌المعارف آنلاین ویکی‌پدیا، نقد توسعه نرم‌افزار را به شرح زیر خلاصه می‌کند:
(در مهندسی سنتی، اجماع واضحی وجود دارد که چگونه چیزها باید ساخته شوند، کدام استانداردها باید رعایت شوند، و کدام خطرات باید مراقبت شود. اگر یک مهندس این شیوه ها را رعایت نکند و چیزی شکست بخورد، از او شکایت می شود. چنین اتفاق نظری در مهندسی نرم‌افزار وجود ندارد: هرکسی روش‌های خود را تبلیغ می‌کند و ادعا می‌کند که مزایای زیادی در بهره‌وری دارد، که معمولاً توسط هیچ مدرک علمی و بی‌طرف پشتیبانی نمی‌شود.)[4]

یک مقابله قدرتمند با این انتقاد، پذیرش گسترده روش‌شناسی فرآیند نرم‌افزار شخصی (PSP) است. فرآیند نرم‌افزار شخصی که در سال ۱۹۹۳ توسط Watts S. Humphrey توسعه یافت، یک رویکرد منظم و ساختار یافته برای توسعه نرم افزار است. با استفاده از مفاهیم و روش‌های PSP در کار خود، افراد تقریباً در هر زمینه فنی می‌توانند مهارت‌های برآورد و برنامه‌ریزی خود را بهبود بخشند، تعهداتی را که می‌توانند انجام دهند، مدیریت کیفیت کار خود و کاهش تعداد ایرادات در محصولات خود داشته باشند.[4]

اهداف اولیه فرآیند نرم‌افزار شخصی حول محور پرورش یک رویکرد منضبط به شیوه های توسعه نـرم افـزار فـردی اسـت. هدف آن تجهیز توسعه‌دهندگان به ابزارها و تکنیک‌هایی برای برنامـه‌ریزی دقیـق کـار، بـرآورد دقیـق وظـایف، پیگیـری پیشرفت، شناسایی و اصلاح خطاها در مراحل اولیه توسعه، و بهبود مستمر مهارت‌های کدنویسی و طراحی است.[4]

اثربخشی متدولوژی فرآیند نرم‌افزار شخصی (و فناوری همراه آن، Team Software ProcessSM یا TSPSM) در هر دو محیط دانشگاهی و صنعتی در گزارش‌های فنی متعدد و مقالات مجلات بررسی شده مستند شده است. از آنجـایی کـه PSP بـه شدت بر جمع آوری و تجزیه و تحلیل داده های شخصی به عنوان اثبات اجـرای مـوثر فرآینـد متکـی اسـت، ادعاهـای مطرح شده در این گزارش ها و مقالات با شواهد عینی و داده های سخت پشـتیبانی مـی شـود. مفـاهیم و روش‌شناسـی فن‌آوری‌های PSP و TSP به سطحی از بلوغ رسیده‌اند که تضمین می‌کند که اصلاحات بیشـتر توسـط جامعـه حرفـه‌ای، دانشگاه‌ها و نهادهای صدور گواهینامه انجام شود. برای حمایت از این تلاش، گسترش بیشـتر آموزشـی فرآینـد نـرم‌افـزار شخصی باید توسط جامعه انجام شود و در آن پذیرفته شود. تحقیقات انجام شده توسـط فـورد و گیبـز نشـان داد کـه بـا پیشرفت یک حرفه، باید راه‌هایی برای ارزیابی و اطمینان از کفایت برنامـه‌های آموزشـی و آموزشـی و شایسـتگی افـراد حرفه‌ای برای پیشبرد این حرفه داشته باشد.[4]

معیارهای صلاحیت فرآیند نرم‌افزار شخصی حرفه ای برای ارزیابی هـم سـطح کسـب دانـش و هـم سـطح مهـارت در بـه کارگیری آن دانش مورد نیاز است. صدور گواهینامه یکی از پرکاربردترین مکانیسم‌هایی است که یک حرفه برای آشـکار ساختن مجموعه اصلی دانش و مهارت‌هایی که از یک حرفه‌ای انتظار می‌رود تسلط داشته باشد، برای ایجاد ارزیابی‌هـای عینی از آن شایستگی‌های اصلی، و ایجاد پایه‌ای برای ادامه صلاحیت افراد حرفه‌ای استفاده می‌کند.
اهمیت PSP فراتر از توسعه دهندگان فردی است و بر اکوسیستم توسعه نرم افزار گسترده تر تأثیر می گذارد. اجـرای آن مزایای متعددی را به همراه دارد، از جمله کاهش خطا، بهبود دقت تخمین، مدیریت زمان کارآمد، افزایش کیفیت کـد، و پرورش ذهنیت منظم در میان توسعه دهندگان.[1]

هدف این مقاله ارائه یک کاوش گسترده از فرآیند نرم‌افزار شخصی اسـت. در مؤلفـه‌های اساسـی خـود، از جملـه مراحـل، تکنیک‌ها و ابزارهای به‌کاررفته بررسی می‌شود. علاوه بر این، مطالعات موردی را ارائه می‌کند که کاربردهای موفق فرآینـد نرم‌افزار شخصی را نشان می‌دهد، چالش‌های بالقوه را مورد بحث قرار می‌دهد و بینش‌هایی را درباره مفـاهیم عملـی آن در شیوه‌های توسعه نرم‌افزار مدرن ارائه می‌دهد.[1]

۲. ساختار متودولوژی فرآیند نرم افزار شخصی

متدولوژی فرآیند نرم‌افزار شخصی به عنوان یک چارچوب سیستماتیک طراحی شده برای بهبود شیوه های توسعه نـرم افـزار فـردی است. فرآیند نرم‌افزار شخصی که در مراحل و تکنیک های متمایز ساختار یافته اسـت، توسـعه دهنـدگان را از طریـق یـک رویکـرد منضبط برای برنامه ریزی، طراحی، کدگذاری، آزمایش و بازتاب کارشان راهنمایی مـی کنـد. ایـن متـدولوژی سـاختاریافته شـامل مراحل متوالی است که هر کدام با دقت به جنبه های حیاتی توسعه نرم افزار می پردازند. ساختار فرآیند نرم‌افزار شخصی یک نقشـه راه جامع برای توسعه دهندگان فراهم می کند که برنامه ریزی دقیق، اجرای دقیق و بهبود مستمر را در طول چرخه عمر توسعه نرم افزار تقویت می کند.[1]

ساختار فرآیند فرآیند نرم‌افزار شخصی با شروع بیانیه الزامات، اولین مرحله در فرآیند فرآیند نرم‌افزار شخصی برنامه ریزی است. یک اسکریپت برنامه ریزی که این کار را راهنمایی می کند و یک خلاصه طرح برای ثبت داده های برنامه ریزی وجود دارد. در حالی که مهندسان اسکریپت را دنبال می کنند تا کار را انجام دهند، زمان و داده های نقص خود را روی گزارش های زمان و نقص ثبت می کنند. در پایان کار، در مرحله پس از مرگ فرآیند نرم‌افزار شخصی، داده های زمان و نقص را از لاگ ها خلاصه می کنند، اندازه برنامه را اندازه گیری می کنند و این داده ها را در فرم خلاصه طرح وارد می کنند. پس از اتمام، آنها محصول نهایی را به همراه فرم خلاصه طرح تکمیل شده تحویل می دهند[1].

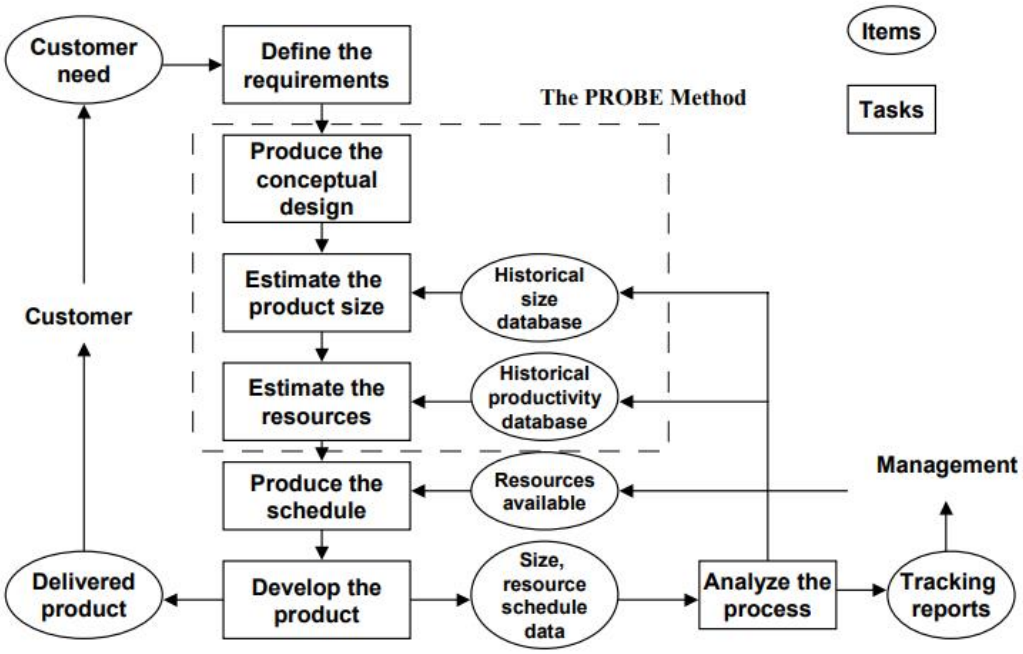
مراحل متودولوژی فرآیند نرم‌افزار شخصی به شرح زیر است:

۱. فاز برنامه ریزی
۲. فاز طراحی
۳. فاز کد نویسی
۴. فاز تدوین
۵. فاز تست
۶. فاز پس از مرگ

مراحل متودولوژی :

فاز برنامه ریزی :

فرآیند نرم‌افزار شخصی با مرحله برنامه ریزی آغاز می شود، مرحله ای حیاتی که در آن توسعه دهندگان اندازه پروژه را تخمین می زنند، وظایف را برنامه ریزی می کنند و منابع را تخصیص می دهند. این مرحله شامل تقسیم پروژه به واحدهای قابل مدیریت، تخمین اندازه هر واحد و تخصیص زمان و منابع بر این اساس است. توسعه‌دهندگان برنامه‌های دقیقی را ایجاد می‌کنند که وظایف و زمان‌بندی‌ها را مشخص می‌کند تا از یک رویکرد ساختاریافته به پروژه اطمینان حاصل کنند.فرایند برنامه ریزی در فرآیند نرم‌افزار شخصی به شکل زیر است:[3]



نیازمندی ها(Requirements): مهندسان برنامه ریزی را با تعریف کاری که باید با جزئیات هر چه بیشتر انجام شود شروع می کنند. اگر تنها چیزی که آنها دارند یک بیانیه الزامات یک جمله ای است، آن گزاره باید مبنایی برای طرح باشد. البته، دقت برآورد و نقشه به شدت تحت تأثیر میزان آگاهی مهندسان از کار مورد نظر است.[3]

طراحی مفهومی(Conceptual Design): برای برآورد و برنامه ریزی، مهندسان ابتدا نحوه طراحی و ساخت محصول را تعریف می کنند. با این حال، از آنجایی که فاز برنامه ریزی برای تولید یک طرح کامل محصول خیلی زود است، مهندسان چیزی را تولید می کنند که طرح مفهومی نامیده می شود. این اولین است، اگر مهندسان مجبور به ساختن آن بر اساس آنچه در حال حاضر می‌دانند، محصول چه شکلی خواهد بود. بعداً، در مرحله طراحی، مهندسان جایگزین های طراحی را بررسی می کنند و یک طرح کامل محصول را تولید می کنند.[2]

تخمین اندازه محصول و منابع مورد نیاز(Estimate Product size and resources) :همبستگی اندازه برنامه با زمان توسعه فقط برای تیم‌ها و سازمان‌های مهندسی خوب است. با این حال، برای مهندسان فردی، همبستگی به طور کلی بسیار بالا است. بنابراین، فرآیند نرم‌افزار شخصی با تخمین اندازه‌های محصولاتی که شخصاً توسعه خواهند داد توسط مهندسان شروع می‌شود. سپس، بر اساس اندازه شخصی و داده‌های بهره‌وری، مهندسان زمان مورد نیاز برای انجام کار را تخمین می‌زنند. در فرآیند نرم‌افزار شخصی ، این برآورد اندازه و منابع با روش تخمین مبتنی بر پروکسی انجام می شود.[3]

روش تخمین مبتنی بر پروکسی: PROBE مخفف Proxy Based Estimating است و از پراکسی ها یا اشیاء به عنوان مبنایی برای تخمین اندازه احتمالی یک محصول استفاده می کند. با تخمین مبتنی بر پروکسی ، مهندسان ابتدا اشیاء مورد نیاز برای ساخت محصول توصیف شده توسط طرح مفهومی را تعیین می کنند. سپس نوع و تعداد متدهای احتمالی را برای هر شی تعیین می کنند. آنها به داده های تاریخی در مورد اندازه اشیاء مشابهی که قبلا توسعه داده اند اشاره می کنند و از رگرسیون خطی برای تعیین اندازه کلی احتمالی محصول نهایی استفاده می کنند. از آنجایی که اندازه شی تابعی از سبک برنامه نویسی است، روش تخمین مبتنی بر پروکسی به مهندسان نشان می دهد که چگونه از داده های برنامه هایی که شخصاً توسعه داده اند برای تولید محدوده اندازه برای استفاده شخصی خود استفاده کنند. هنگامی که آنها اندازه اشیاء را تخمین زدند، از رگرسیون خطی برای تخمین مقدار کل کدی که قصد توسعه آن را دارند استفاده کردند. برای استفاده از رگرسیون خطی، مهندسان باید داده های چرخشی خود را در مورد اندازه برنامه تخمینی در برابر واقعی حداقل سه برنامه قبلی داشته باشند.[3]
روش تخمین مبتنی بر پروکسی همچنین از رگرسیون خطی برای برآورد منابع توسعه استفاده می کند. باز هم، این تخمین بر اساس اندازه تخمین زده شده در مقابل داده های تلاش واقعی از حداقل سه پروژه قبلی است. داده ها باید یک همبستگی معقول بین اندازه برنامه و زمان توسعه را نشان دهند. فرایند نرم‌افزار شخصی نیاز دارد که r ۲ برای همبستگی حداقل ۰ باشد. زمانی که مهندسان کل زمان کار را تخمین زدند، از داده های تاریخی خود برای تخمین زمان مورد نیاز برای هر مرحله از کار استفاده می کنند. ٪ToDate یک آمار در حال اجرا از درصد کل زمان توسعه که یک مهندس در هر مرحله توسعه صرف کرده است را حفظ می کند. با استفاده از این درصدها به عنوان راهنما، مهندسان زمان کل توسعه تخمینی خود را به مراحل برنامه ریزی، طراحی، بررسی طراحی، کد، بررسی کد، کامپایل، آزمون واحد و مراحل پس از مرگ اختصاص می دهند. پس از اتمام، آنها تخمینی برای اندازه برنامه، کل زمان توسعه و زمان مورد نیاز برای هر مرحله توسعه دارند.[3]

تولید زمانبدی: هنگامی که مهندسان زمان مورد نیاز برای هر مرحله فرآیند را می دانند، زمانی را که هر روز یا هفته در کار صرف می کنند، تخمین می زنند. با این اطلاعات، آنها زمان کار را در ساعات برنامه ریزی شده در دسترس تقسیم می کنند تا زمان برنامه ریزی شده برای تکمیل هر کار را تولید کنند. برای پروژه های بزرگتر، فرآیند نرم‌افزار شخصی همچنین روش ارزش کسب شده را برای زمان بندی و ردیابی کار معرفی می کند.[3]

نمودار گانت برای متودولوژی فرآیند نرم‌افزار شخصی به این صورت است که زمانی که برای هر مرحله از ۶ مرحلـه گفتـه شـده را برحسب هفته نوشته میشود.

فاز طراحی:

مرحله طراحی در فرآیند نرم افزار شخصی مرحله محوری در چرخه عمر توسعه نرم افزار را تشکیل می دهد که بر ایجاد طرح ها و مشخصات دقیق برای نرم افزار آینده تمرکز دارد. این مرحله به عنوان پل مهمی بین برنامه‌ریزی اولیه و اجرای کد واقعـی اسـت و هدف آن ایجاد یک طرح جامع برای هدایت توسعه‌دهندگان در طول فرآیند توسعه است.[1]

ترتیب مرحله های طراحی :

۱. **طراحی معماری** : این فعالیت محوری بر تعریف ساختار کلی و سازماندهی نرم افزار متمرکز است. معمـاران و توسـعه دهندگان ارشد برای ایجاد طرح اولیه معماری سیستم، تعیین ماژول ها، اجزاء و روابط آنها با یکدیگر همکاری می کنند. این ممکن است شامل تصمیمات معماری ماننـد انتخـاب الگوهـای طراحـی، تعریـف رابـط هـای سیسـتم و شناسـایی عملکردهای کلیدی هر ماژول یا جزء باشد.[1]

۲. **برنامه ریزی با جزییات**: هدف در اینجا تجزیه نرم افزار به واحدهای قابل مدیریت با مشخصات دقیـق بـرای هـر یـک است. این شامل توضیح بیشتر در مورد عملکردها، ورودی ها، خروجی ها و رابط های مـاژول هـا یـا اجـزای جداگانـه است. توسعه دهندگان برنامه های دقیقی را برای هر ماژول ایجاد می کنند و هدف، رفتار مورد انتظار و تعامـل بـا سـایر بخش های سیستم را مشخص می کنند.[1]

۳. **طراحی رابط**: طراحی رابط کاربری (UI) و تجربه کاربر (UX) برای اطمینان از تعامـل بهینـه بـین کـاربران و نـرم افـزار. طراحان فریم‌های سیمی، ماکاپ‌ها یا نمونه‌های اولیه را ایجاد می‌کنند که طرح‌بندی UI نرم‌افزار، جریان ناوبری و عناصر تعامل را به نمایش می‌گذارد. این مرحله اغلب شامل تست قابلیت استفاده برای اصلاح طراحی رابط است.[1]

۴. **طراحی پایگاه داده**: هنگامی که نرم افزار شامل ذخیره سازی داده ها می شود، این فعالیت بر طراحی طـرح پایگـاه داده، تعریف جداول، روابط و مکانیسم های ذخیره سازی داده ها تمرکز دارد. معماران یا توسعه دهندگان پایگاه داده سـاختار پایگاه داده را طراحی می کنند، داده ها را عادی می کنند و روابط بین موجودیت های مختلف داده برقرار می کنند.[1]

۵. **مستندات** : مستندات جامع برای ضبط و انتقال تصمیمات طراحی، ساختارها و برنامه های انجـام شـده در ایـن مرحلـه ضروری است. ایجاد اسناد دقیق، از جمله نمودارهای معماری، نمودارهای جریان، مدل های داده و مدل های رابط. ایـن اسناد به عنوان مرجعی برای توسعه دهندگان در طول مراحل بعدی توسعه عمل می کنند.[1]

فاز کدنویسی :

هدف اصلی مرحله کدنویسی تبدیل طرح های طراحی تفصیلی ایجاد شده در مراحل قبلی به کد واقعی است. ایـن مرحلـه شـامل ترجمه مشخصات طراحی به کدهای کاربردی و کارآمد است که نیازهای نرم افزار را برآورده می کند.مراحل کدنویسـی بـه شـرح زیر است:[1]

۱. پیاده سازی بر اساس طراحی: توسعه دهندگان شروع به نوشتن کد بر اساس طرح های طراحی دقیق در مرحله طراحی می کنند. این شامل ترجمه مشخصات طراحی، الگوریتم ها و ساختارهای سیستم به کد برنامه نویسی است. نوشتن کدهای ماژولار، قابل استفاده مجدد و مستند شده بر اساس استانداردهای کدگذاری تعیین شده و بهترین شیوه ها.[1]
۲. کنترل نسخه و مستندات: مدیریت تغییرات کد، اطمینان از کنترل نسخه، و حفظ مستندات جامع در طول فرآیند کدنویسی. استفاده از سیستم‌های کنترل نسخه (مانند Git، SVN) برای مدیریت نسخه‌های کد، مستندسازی تغییرات کد، بازبینی‌ها، و اطمینان از به‌روزرسانی نظرات و اسناد کد در کنار تغییرات کد.[1]
۳. بازسازی و بهینه سازی : بهبود مستمر کیفیت، خوانایی و عملکرد کد. شناسایی فرصت‌ها برای بهینه‌سازی کد، بازآفرینی کد برای افزایش خوانایی، قابلیت نگهداری و کارایی بدون تغییر در عملکرد آن.[1]

فاز تدوین :

هدف اصلی مرحله تدوین، اطمینان از اینکه کد نوشته شده از نظر نحوی صحیح است و می تواند با موفقیت به نرم افزار اجرایی ترجمه شود، است. این مرحله شامل ترجمه کد منبع به دستورالعمل های قابل خواندن توسط ماشین و انجام بررسی های اولیه برای خطاها است[1].

فاز تست :

تست واحد : تست واحدهای کد منفرد (توابع، کلاس‌ها یا ماژول‌ها) برای اطمینان از صحت و عملکرد آنها به صورت مجزا. توسعه دهندگان موارد آزمایشی را اجرا می کنند که برای تأیید رفتار واحدهای جداگانه طراحی شده اند، با هدف شناسایی و اصلاح عیوب در مراحل اولیه.[2]

تست ادغام: بررسی تعاملات و رابط های بین ماژول ها یا اجزای مختلف هنگام یکپارچه سازی. توسعه‌دهندگان تعاملات بین ماژول‌ها یا زیرسیستم‌ها را آزمایش می‌کنند و از عملکرد صحیح آن‌ها هنگام ترکیب اطمینان حاصل می‌کنند.[2]

تست سیستم: آزمایش کل سیستم به عنوان یک کل برای تأیید عملکرد و عملکرد آن در برابر الزامات مشخص شده. تست جامع برای اطمینان از اینکه نرم افزار تمام الزامات کاربردی و غیر کاربردی را برآورده می کند. این شامل تست عملکرد، عملکرد، امنیت و قابلیت استفاده است.[2]

تست پسرفت: اطمینان از اینکه اصلاحات یا بهبودهای انجام شده در پایگاه کد تأثیر منفی بر عملکردهای موجود نمی گذارد. اجرای مجدد تست‌های قبلاً اجرا شده برای تأیید اینکه تغییرات جدید نقص‌های جدیدی ایجاد نکرده است یا باعث از کار افتادن عملکردهای موجود نشده است.[2]

تست پذیرش کاربر: مشارکت دادن کاربران نهایی یا ذینفعان برای اعتبارسنجی نرم افزار بر خلاف انتظارات آنها. کاربران یا ذینفعان آزمایش ها را در یک محیط واقعی یا شبیه سازی شده انجام می دهند و تأیید می کنند که نرم افزار نیازهای آنها را برآورده می کند و کاربر پسند است.[2]

فاز پس از مرگ:

هدف اولیه مرحله پس از مرگ، انعکاس پروژه تکمیل شده، تجزیه و تحلیل فرآیند دنبال شده و شناسایی زمینه های بهبود است. هدف این مرحله جمع آوری بینش ها و درس های آموخته شده برای افزایش توسعه نرم افزار در آینده است. این به

شناسایی درس های کلیدی، چه مثبت و چه منفی، از پروژه کمک می کند. درک موفقیت ها و شکست ها به بهبود پروژه های آینده کمک می کند. پس از مرگ نقش مهمی در شناسایی شکاف‌ها، ناکارآمدی‌ها یا حوزه‌هایی که نیاز به بهبود در فرآیند توسعه دارند، ایفا می‌کنند. مستندسازی جنبه های موفقیت آمیز پروژه برای قدردانی از دستاوردها و شناسایی شیوه های تکرار در پروژه های آینده ضروری است. تجزیه و تحلیل چالش‌ها و شکست‌ها بینش‌هایی در مورد اشتباهات پیش آمده ارائه می‌کند و تیم‌ها را قادر می‌سازد از اشتباهات درس گرفته و از تکرار آن‌ها جلوگیری کنند. خلاصه کردن نکات کلیدی و درس های آموخته شده از پروژه تضمین می کند که بینش های ارزشمند از بین نمی روند و می توان آنها را در تلاش های آینده به کار برد. ارزیابی اثربخشی روش‌ها، ابزارها و گردش‌های کاری مورد استفاده در طول پروژه به شناسایی مناطق برای اصلاح فرآیند کمک می‌کند.[1]

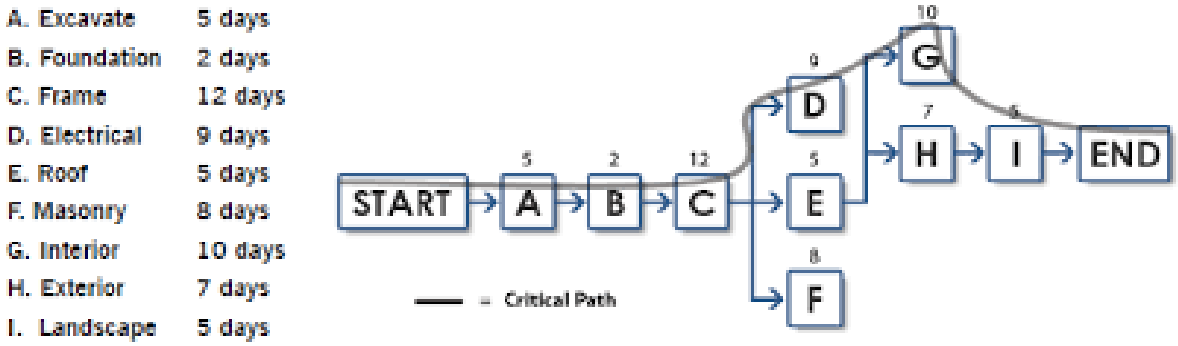
مراحل فاز پس از مرگ:

۱. جمع آوری داده ها: داده ها، معیارها و بازخوردهای مرتبط را از اعضای تیم درگیر در مراحل مختلف پروژه جمع آوری کنید.[1]
۲. تجزیه و تحلیل: تجزیه و تحلیل داده های جمع آوری شده، شناسایی الگوها، روندها، موفقیت ها، شکست ها و زمینه های بهبود.[1]
۳. موارد قابل عمل را شناسایی کنید: بر اساس تجزیه و تحلیل، موارد قابل اجرا را برای بهبود مشخص کنید، خواه به تغییرات فرآیند، ابزارها، ارتباطات یا جنبه های دیگر مربوط باشند.[1]
۴. یافته‌های سند: یافته‌های پس از مرگ شامل موفقیت‌ها، چالش‌ها، درس‌های آموخته‌شده و بهبودهای پیشنهادی را در یک گزارش رسمی خلاصه کنید.[1]
۵. پیاده سازی توصیه ها: با اعمال تغییرات یا بهبودهای توصیه شده در پروژه های آینده، بر روی بینش های به دست آمده از پس از مرگ عمل کنید.[1]

نمودارهای مورد استفاده در مستندات:

متدولوژی فرآیند نرم افزار شخصی از نمودارهای مختلفی برای مستندسازی فرآیند توسعه استفاده می کند. این نمودارها را می توان برای ردیابی پیشرفت، شناسایی و حل مسائل و ارتباط وضعیت پروژه با ذینفعان استفاده کرد.

نمودار شبکه فعالیت (AND): این نمودار توالی وظایف یک پروژه و وابستگی های بین آنها را نشان می دهد. **نمودار شبکه فعالیت** می تواند برای شناسایی مسیرهای بحرانی و تنگناها در برنامه استفاده شود. یک نمونه از نمودار شبکه فعالیت شبکه در شکل زیر مشاهده میشود.



نمودار شبکه فعالیت، همچنین به عنوان نمودار شبکه یا نمودار PERT (تکنیک ارزیابی و بررسی برنامه) شناخته می شود، یک نمایش گرافیکی از وظایف پروژه و وابستگی های متقابل آنها است. به طور گسترده ای در مدیریت پروژه برای تجسم توالی فعالیت ها و روابط آنها برای کمک به برنامه ریزی، زمان بندی و مدیریت پروژه های پیچیده استفاده می شود. نمودار شبکه فعالیت به ویژه برای پروژه هایی با وظایف متعددی که باید به ترتیب خاصی اجرا شوند مفید است.

گره ها: گره ها فعالیت ها یا وظایف فردی را در پروژه نشان می دهند. هر گره معمولاً با نام کار و مدت تخمینی آن برچسب گذاری می شود.
فلش ها (لبه ها): فلش ها گره ها را به هم متصل می کنند و روابط بین وظایف را نشان می دهند. جهت فلش ها نشان دهنده ترتیبی است که وظایف باید انجام شوند.
وابستگی ها: وابستگی ها روابط بین وظایف را نشان می دهد، که نشان می دهد کدام کارها باید قبل از شروع سایرین تکمیل شوند. دو نوع وابستگی وجود دارد:
Finish-to-Start (FS): تا زمانی که کار اول تمام نشده باشد، کار دوم نمی تواند شروع شود.
Start-to-Start (SS): تا زمانی که کار اول شروع نشده باشد، کار دوم نمی تواند شروع شود.
مسیر بحرانی: مسیر بحرانی طولانی ترین دنباله ای از وظایف وابسته است که باید به موقع تکمیل شود تا کل پروژه طبق برنامه ریزی تکمیل شود. هر گونه تاخیر در یک کار در مسیر بحرانی کل پروژه را به تاخیر می اندازد.
شروع زودهنگام (ES) و پایان زودهنگام (EF): شروع زودهنگام اولین نقطه زمانی است که یک کار می تواند شروع شود. شروع زود هنگام اولین نقطه زمانی است که می توان یک کار را تکمیل کرد. این مقادیر به تعیین زودترین زمان تکمیل پروژه کمک می کند.

شروع دیرهنگام (LS) و پایان دیرهنگام (LF): شروع دیرهنگام آخرین نقطه زمانی است که یک کار می تواند بدون تاخیر پروژه شروع شود. پایان دیرهنگام آخرین نقطه زمانی است که می توان یک کار را بدون تأخیر در پروژه تکمیل کرد. این مقادیر به شناسایی وظایفی که می‌توانند بدون تأثیر بر زمان اتمام پروژه به تعویق بیفتند، کمک می‌کنند.
شناور(Slack) : شناور مقدار زمانی است که یک کار می تواند بدون تأثیر بر زمان اتمام پروژه به تاخیر بیفتد. وظایف در مسیر بحرانی دارای شناور صفر هستند، در حالی که وظایف مسیرهای غیر بحرانی ممکن است شناور مثبت داشته باشند.
نقاط عطف: نقاط عطف نقاط مهمی در پروژه هستند که نشان دهنده تکمیل گروهی از وظایف یا پروژه به عنوان یک کل است.
نمودارهای شبکه فعالیت نمایشی بصری از جدول زمانی پروژه، وابستگی ها، و مسیرهای حیاتی را ارائه می دهند و مدیران پروژه را قادر می سازند تا زمان بندی پروژه را بهتر برنامه ریزی و کنترل کنند. آنها ابزار ارزشمندی برای شناسایی تاخیرهای احتمالی، بهینه سازی تخصیص منابع و اطمینان از اینکه پروژه در مسیر خود باقی می ماند، هستند.

نمودار هزینه کیفیت (COQ): نمودار هزینه کیفیت (COQ) یک نمایش بصری است که به سازمان ها کمک می کند تا هزینه های مرتبط با کیفیت محصولات یا خدمات خود را تجزیه و تحلیل و مدیریت کنند. نمودار COQ هزینه‌های مرتبط با کیفیت را در چهار گروه اصلی دسته‌بندی می‌کند و بینش‌هایی را در زمینه‌هایی ارائه می‌کند که می‌توان برای بهبود کیفیت کلی و در عین حال به حداقل رساندن هزینه‌های غیر ضروری، بهبودهایی انجام داد. چهار دسته از هزینه ها در نمودار COQ به شرح زیر است:

هزینه‌های پیشگیری: هزینه‌های پیشگیری هزینه‌هایی هستند که در وهله اول برای جلوگیری از بروز نقص یا خطا انجام می‌شوند. مثال‌ها: برنامه‌های آموزشی، بهبود فرآیند، برنامه‌ریزی کیفیت، مدیریت کیفیت تامین‌کننده و فعالیت‌های تضمین کیفیت تحت هزینه‌های پیشگیری قرار می‌گیرند

هزینه های ارزیابی: هزینه های ارزیابی مربوط به ارزیابی و بازرسی محصولات یا خدمات است تا اطمینان حاصل شود که استانداردهای کیفی مطابقت دارند. مثال‌ها: بازرسی، آزمایش، ممیزی کیفیت و هزینه‌های مربوط به حفظ سیستم‌های کنترل کیفیت، هزینه‌های ارزیابی محسوب می‌شوند..

هزینه های خرابی داخلی: هزینه های خرابی داخلی زمانی ایجاد می شود که عیوب یا خطاها قبل از رسیدن محصول یا خدمات به دست مشتری شناسایی شوند. مثال‌ها: دوباره کاری، مواد ضایع شده، آزمایش مجدد و خرابی به دلیل مشکلات کیفی هزینه‌های خرابی داخلی هستند.

هزینه های شکست خارجی: هزینه های خرابی خارجی زمانی متحمل می شوند که عیوب یا خطاها توسط مشتریان پس از تحویل محصول یا خدمات شناسایی شود. مثال‌ها: ادعاهای گارانتی، بازگشت مشتری، اقدامات قانونی و هزینه‌های مرتبط با نارضایتی مشتری، هزینه‌های شکست خارجی هستند.

هزینه اجزای نمودار کیفیت: نمودار COQ معمولاً این دسته بندی هزینه ها را در قالبی گرافیکی، اغلب با استفاده از نمودار دایره ای یا نمودار میله ای نشان می دهد. این نمودار به ذینفعان اجازه می دهد تا به صورت بصری توزیع هزینه های مرتبط با کیفیت و تأثیر آنها بر بودجه کلی را درک کنند. هر دسته به عنوان بخشی از نمودار نشان داده می شود که اندازه بخش مربوط به نسبت هزینه کل است.
مزایای COQ :

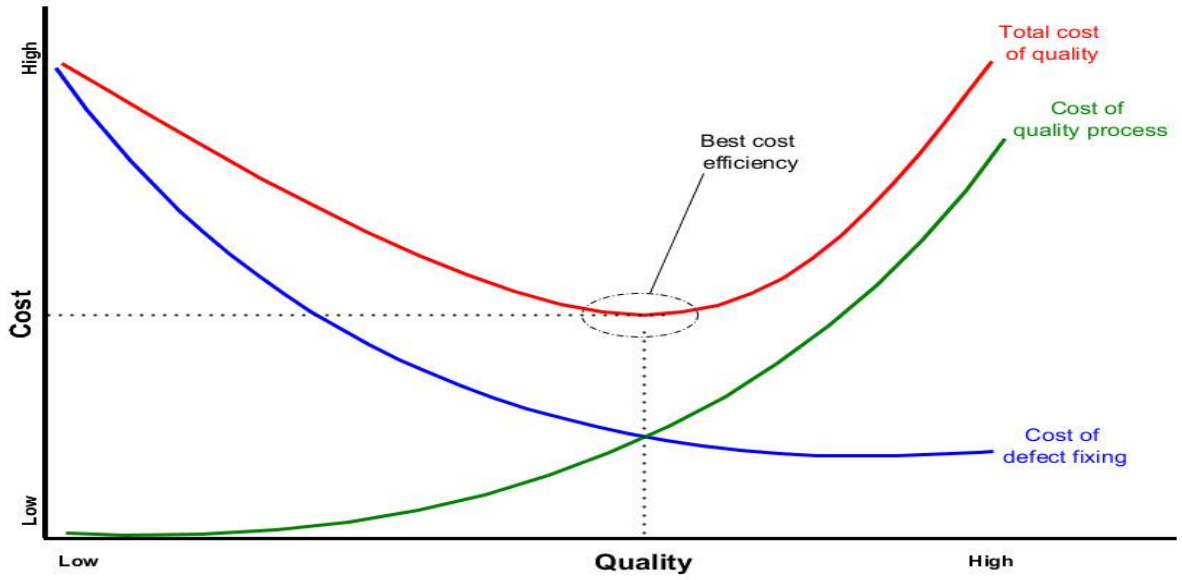
شناسایی مناطق پرهزینه: نمودار COQ به شناسایی مناطقی که منابع قابل توجهی در آنها تخصیص داده شده است کمک می کند و بینش هایی را در مورد فرصت هایی برای کاهش هزینه ارائه می دهد.

پشتیبانی تصمیم گیری: به تصمیم گیری آگاهانه در مورد محل سرمایه گذاری منابع برای حداکثر تأثیر بر کیفیت محصول یا خدمات کمک می کند.

بهبود مستمر: با به‌روزرسانی منظم نمودار COQ، سازمان‌ها می‌توانند تغییرات هزینه‌های مرتبط با کیفیت را در طول زمان پیگیری کنند و اثربخشی طرح‌های بهبود کیفیت را ارزیابی کنند.

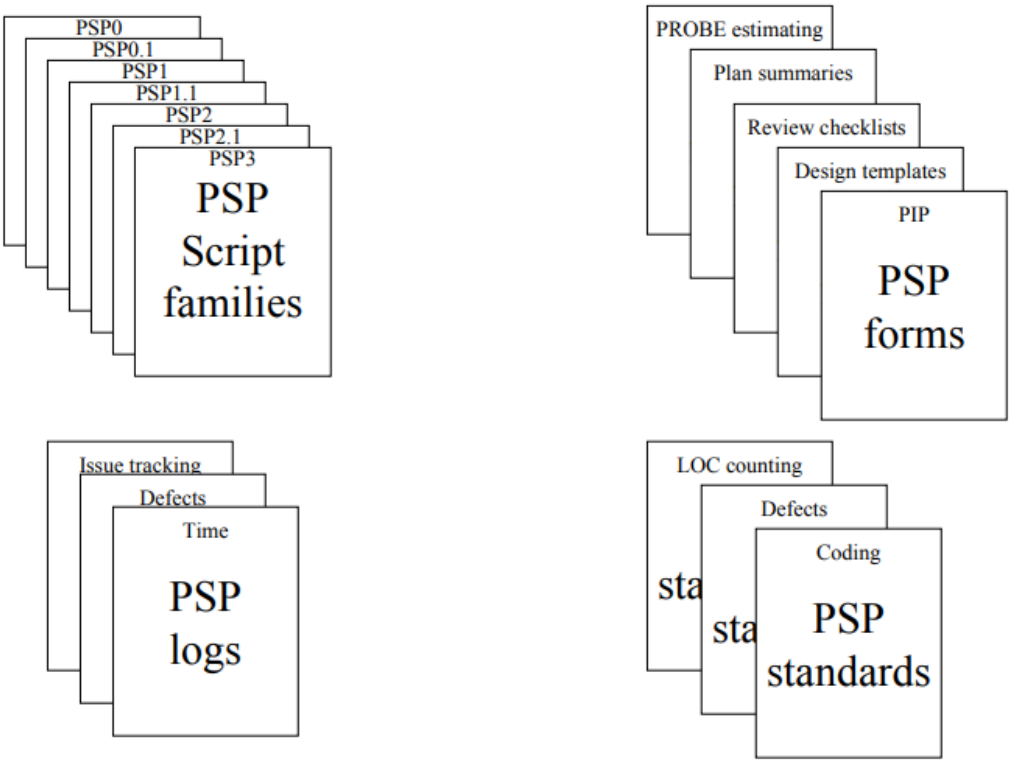
ابزار ارتباطی: نمودار به عنوان یک ابزار ارتباطی عمل می‌کند و به تیم‌ها و ذینفعان اجازه می‌دهد تفکیک هزینه و تأثیر فعالیت‌های مرتبط با کیفیت را درک کنند.

نمودار هزینه کیفیت ابزاری قدرتمند برای سازمان ها برای تجسم و مدیریت هزینه های مرتبط با ارائه محصولات یا خدمات با کیفیت است. تصمیم گیری آگاهانه را قادر می سازد، از تلاش های بهبود مستمر حمایت می کند و فرهنگ کیفیت را در سازمان پرورش می دهد.در شکل زیر یک نمونه از نمودار هزینه کیفیت مشاهده میشود.[1]



سطح‌های مختلف فرآیند نرم افزار شخصی:

از آنجایی که فرآیند فرآیند نرم‌افزار شخصی دارای تعدادی روش است که عموماً توسط مهندسان انجام نمی‌شود، نسخه‌های فرآیند نرم‌افزار شخصی در مجموعه‌ای از هفت نسخه فرآیندی معرفی شده‌اند. این نسخه ها از PSP0 تا PSP3 نامگذاری شده اند هر نسخه بر اساس نسخه قبلی ساخته شده و بهبودها و اصلاحات را معرفی می کند و هر نسخه دارای مجموعه ای مشابه از گزارش ها، فرم ها، اسکریپت ها و استانداردها است، همانطور که در شکل زیر نشان داده شده است. و فرم ها الگوهایی را برای ثبت و ذخیره داده ها ارائه می دهند و استانداردها مهندسان را در حین انجام کار راهنمایی می کنند.[3]



یک اسکریپت فرآیند نرم‌افزار شخصی همان چیزی است که دمینگ آن را فرآیند عملیاتی می نامد . به عبارت دیگر، فرآیندی است که برای استفاده طراحی شده است. این در قالبی ساده برای استفاده با دستورالعمل های کوتاه و دقیق ساخته شده است. در حالی که اسکریپت ها توصیف می کنند که چه باید کرد، آنها بیشتر شبیه چک لیست هستند تا آموزش. آنها شامل مواد آموزشی مورد نیاز کاربران آموزش ندیده نیستند. هدف این اسکریپت هدایت مهندسان در استفاده مداوم از فرآیندی است که آنها آن را درک می کنند. چندین بخش بعدی این گزارش روش‌های مختلفی را که فرآیند نرم‌افزار شخصی برای برنامه‌ریزی، تخمین جمع‌آوری داده‌ها، مدیریت کیفیت و طراحی استفاده می‌کند، شرح می‌دهد.[3]

PSP0 (PSP 0.1، PSP 0.2 و غیره): این نسخه های اولیه مفاهیم اساسی فرآیند نرم‌افزار شخصی را معرفی کردند. آنها بر روی شیوه‌های اساسی مانند مدیریت زمان، ردیابی نقص و تخمین اندازه تمرکز کردند و یک رویکرد ساختاریافته برای توسعه‌دهندگان به منظور ارتقای مهارت‌ها و فرآیندهای خود ارائه کردند.[1]

PSP1: این نسخه بر روی اصول اساسی فرآیند نرم‌افزار شخصی گسترش یافته و تکنیک‌ها و ابزارهای دقیق‌تری را برای تخمین، برنامه‌ریزی و ردیابی ارائه می دهد. بر اهمیت جمع آوری داده ها در مورد عملکرد شخصی برای تجزیه و تحلیل و بهبود تاکید کرد[1] .

.

PSP2: با تکیه بر PSP1، این تکرار روش‌شناسی را بیشتر اصلاح کرد و شیوه‌های اضافی برای بهبود فرآیند را ترکیب کرد و بر اهمیت معیارها در تصمیم‌گیری تأکید کرد.[1]

PSP3: این نسخه به اصلاح روش ادامه داد و تکنیک های پیشرفته تری را برای تخمین، ردیابی نقص و بهبود فرآیند معرفی کرد. این بر تصمیم گیری مبتنی بر داده تأکید داشت و هدف آن بهبود بیشتر بهره وری و کیفیت تلاش های توسعه نرم افزار فردی بود.
[1]

جمع آوری داده در فرآیند نرم افزار شخصی:

در فرآیند نرم‌افزار شخصی ، مهندسان از داده ها برای نظارت بر کار خود و کمک به آنها در برنامه ریزی بهتر استفاده می کنند. برای انجام این کار، آنها داده هایی را در مورد زمانی که در هر مرحله از فرآیند صرف می کنند، اندازه محصولاتی که تولید مـی کننـد و کیفیت این محصولات جمع آوری می کنند. [3]

اندازه‌گیری زمان: در فرآیند نرم‌افزار شخصـی ، مهندسان از گزارش ثبت زمان برای اندازه گیری زمان صرف شده در هر مرحلـه از فرآیند استفاده می کنند. در این گزارش، آنها زمان شروع کار بر روی یک کار، زمانی که کار را متوقف کردند و هر زمـان وقفـه را یادداشت می کنند. به عنوان مثال، یک وقفه یک تلفن خواهد بود تماس بگیرید، یک استراحت کوتاه، یا کسی کـه بـرای پرسـیدن سوالی صحبت را قطع می کند. مهندسین با ردیابی دقیق زمان، تلاش‌هایی را که واقعاً برای وظایف پروژه صرف می‌شود، پیگیـری می‌کنند. از آنجایی که زمان وقفه اساساً تصادفی است، نادیده گرفتن این زمان ها یک خطای تصادفی بزرگ را به داده های زمـانی اضافه می کند و دقت تخمین را کاهش می دهد.[3]

اندازه‌گیری مقدار: از آنجایی که زمان لازم برای توسعه یک محصول تا حد زیادی با اندازه آن محصول تعیین می شـود، مهندسـان هنگام استفاده از فرآیند نرم‌افزار شخصی ابتدا اندازه محصولاتی را که قصد توسعه آن را دارند تخمین می زنند. سپس، پس از اتمام، اندازه آن ها را اندازه گیری می کنند. محصولاتی که تولید کردند این داده‌های اندازه‌ای را که برای برآورد دقیق اندازه نیاز دارند در اختیار مهندسان قرار می‌دهد. با این حال، برای اینکه این داده ها مفید باشند، اندازه گیری انـدازه بایـد بـا زمـان توسـعه محصـول مرتبط باشد. در حالی که خطوط کد (LOC) معیار اصلی اندازه فرآیند نرم‌افزار شخصـی است، هر اندازه اندازه ای را می توان استفاده کرد که ارتباط منطقی بین زمان توسعه و اندازه محصول فراهم می کند. همچنین باید امکان انـدازه گیـری خودکـار انـدازه واقعـی محصول را فراهم کند.[3]

خطوط کد(LOC):
فرآیند نرم‌افزار شخصی از اصطلاح (LOC منطقی) برای اشاره به ساختار منطقی زبان برنامه نویسی استفاده می کند. از آنجایی که راه های زیادی برای تعریف خط کد منطقی وجود دارد، مهندسان باید دقیقاً نحوه اندازه گیری خط کد را تعریـف کننـد. هنگـامی کـه مهندسان در یک تیم یا در یک سازمان نرم افزاری بزرگتر، آنها باید از استاندارد خط کد تیم یا سـازمان اسـتفاده کننـد. اگـر چنـین استانداردی وجود نداشته باشد، فرآیند نرم‌افزار شخصی مهندسان را در تعریف خود راهنمـایی مـی کنـد. از زمـان فرآینـد نـرم‌افـزار شخصـی نیازمند آن است که مهندسان اندازه برنامه هایی را که تولید می کنند اندازه گیری کنند، و از آنجـایی کـه شـمارش دسـتی اندازه برنامه هم زمان بر و هم نادرست است، فرآیند نرم‌افزار شخصی همچنین مهندسان را در نوشتن دو شمارنده خـط کـد خودکـار برای استفاده در دوره فرآیند نرم‌افزار شخصی راهنمایی می کند.[3]

دسته بندی اندازه:

برای ردیابی اینکه چگونه اندازه یک برنامه در طول توسعه تغییر می کند، مهم است که دسته بندی های مختلف خط کد محصـول را در نظر بگیرید. این دسته بندی ها هستند.[3]

- پایه: هنگامی که یک محصول موجود بهبود می یابد، خط کد پایه به اندازه نسخه اصلی محصول قبل از انجام هرگونه تغییر است.
- اضافه شده: کد اضافه شده، کدی است که برای یک برنامه جدید نوشته شده یا به یک برنامه پایه موجود اضافه شده است.
- اصلاح شده. خط کد اصلاح شده همان کد پایه در یک برنامه موجود است که تغییر کرده است.[3]

• حذف شده: خط کد حذف شده همان کد پایه در یک برنامه موجود است که حذف شده است.[3]

• جدید و تغییر یافته: زمانی که مهندسان نرم افزاری را توسعه می دهند، اضافه کردن یا اصلاح یک خط کد نسبت به حذف یا استفاده مجدد از آن زمان بسیار بیشتری طول می کشد. بنابراین، در فرآیند نرم افزار شخصی ، مهندسان فقط از کد اضافه یا اصلاح شده برای تخمین اندازه و منابع استفاده می کنند. این کد خط کد جدید و تغییر یافته نامیده می شود.[3]

• استفاده مجدد: در فرایند نرم‌افزار شخصی، خط کد استفاده مجدد، کدی است که از یک کتابخانه استفاده مجدد گرفته شده و بدون تغییر در یک برنامه یا نسخه برنامه جدید استفاده می شود. استفاده مجدد کد پایه اصلاح نشده حفظ شده از یک نسخه برنامه قبلی را محاسبه نمی کند و هیچ کدی را که با اصلاحات مجدداً استفاده شده است محاسبه نمی کند.[3]

• استفاده مجدد جدید: معیار استفاده مجدد جدید خط کد را که یک مهندس توسعه می دهد و به کتابخانه استفاده مجدد کمک می کند، حساب می کند.[3]

• مجموع: مجموع خط کد اندازه کل یک برنامه است، صرف نظر از منبع کد[3]

محاسبه اندازه:

هنگام اصلاح برنامه ها، اغلب لازم است تغییرات ایجاد شده در برنامه اصلی را ردیابی کنید. این داده ها برای تعیین حجم محصول توسعه یافته، بهره وری مهندس و کیفیت محصول استفاده می شود. برای ارائه این داده ها، فرآیند نرم افزار شخصی از روش حسابداری اندازه برای ردیابی تمام اضافات، حذف ها و تغییرات ایجاد شده در یک برنامه استفاده می کند.[1]

برای استفاده از حسابداری اندازه، مهندسان به داده هایی در مورد مقدار کد در هر دسته اندازه نیاز دارند. به عنوان مثال، اگر از محصولی با ۱۰۰۰۰۰ خط کد برای توسعه نسخه جدید استفاده شود و ۱۲۰۰۰ خط کد کد حذف شده، ۲۳۰۰۰ خط کد اضافه شده، ۵۰۰۰ خط کد کد اصلاح شده و ۳۰۰۰ خط کد استفاده مجدد وجود داشته باشد، خط کد جدید و تغییر یافته وجود داشته باشد. خواهد بود[3] :

مقدار خط کد جدید =مقدار قبلی +مقدار اضافه شده

هنگام اندازه گیری اندازه کل یک محصول، محاسبات به شرح زیر است:

کل خط کد = پایه – حذف + اضافه + استفاده مجدد

نه خط کد اصلاح شده و نه «استفاده مجدد جدید» در کل گنجانده شده است. این به این دلیل است که یک خط کد اصلاح شده را می توان با یک خط کد حذف شده و یک خط کد اضافه شده نشان داد و خط کد "استفاده مجدد جدید" قبلاً در خط کد اضافه شده به حساب می آید.

پیش تر گفته شد که در نسخه PSP2 بررسی کیفیت به این متد اضافه شد، اکنون به بررسی چگونگی آن میپردازیم:

کیفیت نرم افزار به طور فزاینده ای اهمیت پیدا می کند و نرم افزار معیوب یک مشکل فزاینده است. هر گونه نقص در بخش کوچکی از یک برنامه بزرگ به طور بالقوه می تواند مشکلات جدی ایجاد کند. همانطور که سیستم‌ها سریع‌تر، پیچیده‌تر و خودکارتر می‌شوند، خرابی‌های فاجعه‌بار به طور فزاینده‌ای محتمل‌تر و بالقوه آسیب‌رسان‌تر می‌شوند. مشکل این است که کیفیت برنامه های بزرگ به کیفیت قطعات کوچکتری که از آن ساخته شده اند بستگی دارد. بنابراین، برای تولید برنامه های بزرگ با کیفیت بالا، هر مهندس نرم افزاری که یک یا چند قطعه از سیستم را توسعه می دهد، باید کار با کیفیت بالا انجام دهد. این بدان

معناست که همه مهندسان باید کیفیت کار شخصی خود را مدیریت کنند. برای کمک به آنها در انجام این کار، فرآیند نرم‌افزار شخصی مهندسان را در ردیابی و مدیریت هر نقص راهنمایی می کند[3] .

عیوب و کیفیت: محصولات نرم افزاری با کیفیت باید نیازهای عملکردی کاربران را برآورده کنند و به طور قابل اعتماد و پیوسته عمل کنند. در حالی که عملکرد نرم افزار برای کاربران برنامه بسیار مهم است، عملکرد آن قابل استفاده نیست مگر اینکه نرم افزار اجرا شود. با این حال، برای اجرای نرم افزار، مهندسان باید تقریباً تمام نقص های آن را برطرف کنند. بنابراین، در حالی که جنبه های زیادی برای کیفیت نرم افزار وجود دارد، اولین نگرانی کیفیت مهندس لزوماً باید در یافتن و رفع نقص باشد. داده های فرآیند نرم‌افزار شخصی نشان می دهد که حتی برنامه نویسان باتجربه در هر هفت تا ده خط کد، نقصی را وارد می کنند. اگرچه مهندسان معمولاً اکثر این عیوب را هنگام کامپایل و آزمایش واحد پیدا می‌کنند، روش‌های نرم‌افزاری سنتی نقص‌های زیادی را در محصول نهایی به جا می‌گذارند.

اشتباهات ساده کدنویسی می تواند نقص های بسیار مخرب یا سخت پیدا کند. برعکس، بسیاری از عیوب طراحی پیچیده اغلب به راحتی قابل یافتن هستند. اشتباه و پیامدهای آن تا حد زیادی مستقل است. حتی خطاهای پیش پا افتاده پیاده سازی می تواند باعث مشکلات جدی سیستم شود. این امر به ویژه مهم است زیرا منشأ اکثر نقص‌های نرم‌افزار، نادیده‌گیری‌ها و اشتباهات ساده حرفه‌ای است. در حالی که مسائل طراحی همیشه مهم هستند، برنامه‌های تازه توسعه‌یافته معمولاً در مقایسه با تعداد زیاد اشتباهات ساده، نقص‌های طراحی کمی دارند. برای بهبود کیفیت برنامه، آموزش فرآیند نرم‌افزار شخصی به مهندسان نشان می‌دهد که چگونه تمام نقص‌هایی را که در برنامه‌های خود پیدا می‌کنند پیگیری و مدیریت کنند.[3]

مسئولیت مهندس: اولین اصل کیفیت فرآیند نرم‌افزار شخصی این است که مهندسان شخصاً مسئول کیفیت برنامه هایی هستند که تولید می کنند. از آنجایی که مهندس نرم افزاری که یک برنامه را می نویسد با آن آشنا است، آن مهندس می تواند به بهترین نحو و موثرترین نقص را پیدا کند، برطرف کند و از آن جلوگیری کند. فرآیند نرم‌افزار شخصی مجموعه‌ای از اقدامات و اقدامات را برای کمک به مهندسان در ارزیابی کیفیت برنامه‌هایی که تولید می‌کنند و راهنمایی آنها در یافتن و رفع تمام نقص‌های برنامه در سریع‌ترین زمان ممکن ارائه می‌کند. علاوه بر اندازه‌گیری و ردیابی کیفیت، روش‌های مدیریت کیفیت فرآیند نرم افزار شخصی عبارتند از حذف زودهنگام نقص و پیشگیری از نقص.[1]

رفع نقص اولیه: هدف اصلی کیفیت فرآیند نرم‌افزار شخصی یافتن و رفع نقص قبل از اولین کامپایل یا تست واحد است. فرآیند فرآیند نرم‌افزار شخصی شامل مراحل طراحی و بررسی کد است که در آن مهندسان شخصاً محصولات کاری خود را قبل از بازرسی، کامپایل یا آزمایش بررسی می کنند. اصل پشت فرآیند بررسی فرآیند نرم‌افزار شخصی این است که افراد معمولاً اشتباهات مشابهی را تکرار می کنند. بنابراین، مهندسان با تجزیه و تحلیل داده‌ها در مورد نقص‌هایی که مرتکب شده‌اند و ایجاد چک لیستی از اقدامات مورد نیاز برای یافتن آن اشتباهات، می‌توانند عیوب را به بهترین نحو پیدا و رفع کنند. مهندسان آموزش دیده فرآیند نرم‌افزار شخصی به طور متوسط ۶.۵۲ نقص در ساعت در بررسی کد شخصی و ۲.۹۶ نقص در ساعت در بررسی طراحی شخصی پیدا می کنند. این با ۲.۲۱ نقص در هر ساعت در تست واحد مقایسه می شود. با استفاده از داده های فرآیند نرم‌افزار شخصی ، مهندسان می توانند هم در زمان صرفه جویی کنند و هم کیفیت محصول را بهبود بخشند. برای مثال، از میانگین داده‌های فرآیند نرم‌افزار شخصی ، زمان حذف ۱۰۰ نقص در تست واحد ۴۵ ساعت است در حالی که زمان یافتن این تعداد نقص در بررسی کد تنها ۱۵ ساعت است.[1]

پیشگیری از نقص: موثرترین راه برای مدیریت عیوب، جلوگیری از معرفی اولیه آنهاست. در فرآیند نرم‌افزار شخصی سه راه متفاوت اما متقابل برای جلوگیری از نقص وجود دارد. اولین مورد این است که مهندسان داده های مربوط به هر نقصی را که پیدا کرده و برطرف می کنند، ثبت کنند. سپس آنها این داده ها را بررسی می کنند تا مشخص کنند چه چیزی باعث نقص شده است و تغییراتی در فرآیند برای از بین بردن این علل ایجاد می کند. مهندسان با اندازه‌گیری عیوب خود، از اشتباهات خود آگاه‌تر می‌شوند، نسبت به عواقب آن حساس‌تر هستند و داده‌های لازم برای جلوگیری از اشتباهات مشابه در آینده را دارند. کاهش سریع اولیه در کل نقایص در طول چند برنامه دوره اول فرآیند نرم‌افزار شخصی نشان دهنده اثربخشی این روش پیشگیری است. دومین

رویکرد پیشگیری، استفاده از روش طراحی موثر و نمادگذاری برای تولید طرح های کامل است. برای ثبت کامل یک طرح، مهندسان باید آن را کاملاً درک کنند. این نه تنها طرح های بهتری را تولید می کند. منجر به اشتباهات طراحی کمتر می شود. سومین روش پیشگیری از نقص نتیجه مستقیم روش دوم است: با طراحی دقیق تر، زمان کدگذاری کاهش می یابد و در نتیجه تزریق نقص کاهش می یابد. داده های فرآیند نرم‌افزار شخصی برای ۲۹۸ مهندس باتجربه تاثیر بالقوه کیفیت یک طراحی خوب را نشان می دهد. این داده ها نشان می دهد که در طول طراحی، مهندسان به طور متوسط ۱.۷۶ نقص در ساعت تزریق می کنند، در حالی که در هنگام کدگذاری ۴.۲۰ نقص در ساعت تزریق می کنند. از آنجایی که کدنویسی یک طرح کاملاً مستند زمان کمتری می‌برد، با تولید یک طرح کامل، مهندسان نیز زمان کدگذاری خود را کاهش می‌دهند. بنابراین، مهندسان با تولید طرح‌های کامل، نقص‌های کدگذاری کمتری را تزریق می‌کنند.[3]

چالش‌های متودولوژی فرآیند نرم افزار شخصی:

زمانبر بودن: پذیرش فرآیند نرم‌افزار شخصی مستلزم زمان و تلاش برای یادگیری و تمرین است، که ممکن است در ابتدا تا زمانی که توسعه دهندگان در متدولوژی مهارت پیدا کنند، توسعه را کند کند.

مقاومت در برابر تغییر : تشویق توسعه‌دهندگان به اتخاذ شیوه‌های جدید و پیروی از فرآیندهای سخت‌گیرانه ممکن است با مقاومت مواجه شود، به‌ویژه در تیم‌هایی که به رویکردهای انعطاف‌پذیرتر یا غیررسمی عادت دارند.

پیچیدگی: فرآیند نرم‌افزار شخصی نیاز به نظم و انضباط در مستندسازی هر مرحله از فرآیند توسعه دارد، که می تواند به عنوان دست و پا گیر و وقت گیر تلقی شود.

جمع آوری و پردازش اطلاعات: جمع‌آوری و تجزیه و تحلیل دقیق داده‌های شخصی برای بهبود فرآیند ممکن است یک چالش باشد، به خصوص اگر توسعه‌دهندگان آن را چالش برانگیز بدانند که به طور مداوم کار خود را پیگیری کنند.

انطباق با پویایی تیمی: فرآیند نرم‌افزار شخصی ، که در درجه اول بر روی شیوه های فردی متمرکز است، ممکن است زمانی که در یک محیط تیمی اعمال می شود، نیاز به انطباق داشته باشد و فرآیندهای فردی را در جریان کار تیمی ادغام می کند.

پذیرش موفقیت آمیز فرآیند نرم‌افزار شخصی اغلب نیازمند تعهد توسعه دهندگان و مدیریت برای غلبه بر این چالش ها است. آموزش، پذیرش تدریجی، فرهنگ سازمانی حمایتی و تقویت مستمر مزایای آن می تواند به کاهش این چالش ها کمک کند و اجرای موفقیت آمیز فرآیند نرم‌افزار شخصی را در پروژه های توسعه نرم افزار امکان پذیر کند.[1]

۳. خلاصه و نتیجه‌گیری

در این مطالعه، اجرای فرآیند نرم‌افزار شخصی به عنوان یک روش ساختار یافته برای توسعه نرم‌افزار فردی مورد بررسی قرار گرفت. روش فرآیند نرم‌افزار شخصی ، شامل مراحل از برنامه ریزی و طراحی تا آزمایش و تجزیه و تحلیل پس از مرگ، برای اثربخشی آن در افزایش کیفیت نرم افزار، توسعه مهارت های فردی و بهبود فرآیند مورد بررسی قرار گرفت.

در طول تحقیق، مزایای فرآیند نرم‌افزار شخصی آشکار شد. رویکرد منضبط، فرآیند توسعه سیستماتیک را تسهیل می‌کند، که منجر به بهبود ردیابی نقص، مدیریت زمان بهتر و بهبود کیفیت کد می‌شود. با تأکید بر برنامه ریزی دقیق، برآورد دقیق و تصمیم گیری مبتنی بر داده، فرآیند نرم‌افزار شخصی به نتایج قابل پیش بینی پروژه و افزایش مهارت های فردی کمک کرد.

این مطالعه هم اجرای موفق پروژه و هم چالش‌هایی را که در طول پذیرش فرآیند نرم‌افزار شخصی بـا آن مواجـه می‌شـوند، تحلیـل کرد. در حالی که پروژه‌های موفق کیفیت نرم‌افزار بالاتر، پیش‌بینی‌پذیری بهبود یافته و افزایش مهارت‌های فردی را نشان می‌دهنـد، چالش‌هایی مانند سرمایه‌گذاری اولیه، مقاومت در برابر تغییر، و پیچیدگی‌ها در جمـع‌آوری و تجزیـه و تحلیـل داده‌هـا بـه عنـوان موانعی در پیاده‌سازی مؤثر فرآیند نرم‌افزار شخصی شناسایی شدند.

در نتیجه، پیاده‌سازی فرآیند نرم‌افزار شخصی مزایای قابل‌توجهی را از نظر کیفیت نرم‌افزار، قابلیت پیش‌بینی و توسـعه مهارت‌هـای فردی ارائه می‌دهد. با وجود چالش‌هایی که در پذیرش وجود دارد، رویکرد ساختاریافته فرآینـد نـرم‌افـزار شخصـی بـه طـور قابـل توجهی به بهبود فرآیندها و نتایج در توسعه نرم‌افزار کمک می‌کند. برای ادغام موفقیت آمیز فرآینـد نـرم‌افـزار شخصـی در اقـدامات سازمانی، پرداختن به چالش ها از طریق آموزش، پذیرش تدریجی و فرهنگ سازمانی حمایتی ضـروری اسـت. تعهـد مسـتمر بـه پذیرش اصول فرآیند نرم‌افزار شخصی و استفاده از مزایای آن می تواند به بهبود شیوه های توسعه نرم افزار و در نهایت محصـولات نرم افزاری بهتر منجر شود.
این مطالعه بر اهمیت روش‌های منضبط مانند فرآیند نرم‌افزار شخصی در بهبود فرآیندهای توسعه نرم‌افزار و قابلیت‌های توسعه‌دهنده فردی تاکید می‌کند و راه را برای تولید نرم‌افزار کارآمدتر و باکیفیت‌تر هموار می‌کند.

مراجع

[1] Introduction to the Personal Software Process by Watts Humphrey
[2] A Discipline for Software Engineering by Watts Humphrey
[3] The Personal Software Process by CarnegieMellon Software Engineering Institute
[4] The personal Software Process Body of Knowledge by CarnegieMellon Software Engineering Institute