

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده مهندسی کامپیوتر

عنوان گزارش فارسی

زیرعنوان گزارش

نویسندگان:

محمد شمس الدینی	محمد اکبرپورجنت	محمدسینا الهکرم
نام نویسنده ششم	نام نویسنده پنجم	نام نویسنده چهارم
نام نویسنده نهم	نام نویسنده هشتم	نام نویسنده هفتم
نام نویسنده دوازدهم	نام نویسنده یازدهم	نام نویسنده دهم
نام نویسنده پانزدهم	نام نویسنده چهاردهم	نام نویسنده سیزدهم

استاد راهنما: دکتر محمدهادی علائیان

چکیده

این قسمت شامل چکیده گزارش است. چکیده باید خلاصه‌ای جامع از محتوای گزارش را ارائه دهد و شامل موارد زیر باشد:

- هدف از انجام پروژه یا تحقیق
- روش‌های استفاده شده
- نتایج اصلی به دست آمده
- نتیجه‌گیری کلی

کلیدواژه‌ها: لاتک، فارسی، xepersian، گزارش، قالب

فهرست مطالب

۱	چکیده
۷	۱ فرایندهای مهندسی نرم افزار و چرخه های تکامل تا پیدایش
۷	۱.۱ پیش زمینه
۷	۱.۱.۱ تاریخچه
۸	۲.۱.۱ اهمیت موضوع
۸	۲.۱ اهداف پروژه
۸	۱.۲.۱ اهداف کوتاه مدت
۹	۲.۲.۱ اهداف بلندمدت
۹	۳.۱ ساختار گزارش
۱۰	۲ مشکلات مطرح در چرخه های توسعه و تکامل نرم افزار
۱۰	۱.۲ مفاهیم پایه
۱۰	۱.۱.۲ تعاریف اولیه
۱۰	۲.۱.۲ قضایای اساسی
۱۱	۲.۲ مرور ادبیات
۱۱	۱.۲.۲ کارهای پیشین
۱۱	۲.۲.۲ مقایسه روش ها
۱۱	۳.۲ روش های موجود

۱۲	روش اول	۱.۳.۲
۱۲	روش دوم	۲.۳.۲
۱۳	مزایا و معایب	۳.۳.۲
۱۴		۳ DevOps و نقش آن در فرایند تکامل نرم افزار	
۱۴	مقدمه و تعریف DevOps	۱.۳
۱۴	فلسفه DevOps و ارتباط آن با Agile	۲.۳
۱۵	چرخه عمر DevOps	۳.۳
۱۷		۴ چرایی نیاز به بازطراحی در پیاده سازی نرم افزار	
۱۷	مقدمه	۱.۴
۱۸	تعریف بازطراحی (Redesign) / (Reengineering)	۲.۴
۱۸	بازمهندسی در برابر مهندسی رو به جلو	۱.۲.۴
۱۸	مهندسی معکوس (بازیابی طراحی)	۲.۲.۴
۱۸	دلایل اصلی نیاز به بازطراحی	۳.۴
۱۸	تغییر نیازمندی ها	۱.۳.۴
۱۹	فناوری های جدید	۲.۳.۴
۱۹	ضعف معماری اولیه	۳.۳.۴
۱۹	انباشت بدهی فنی	۴.۳.۴
۱۹	مراحل بازطراحی نرم افزار	۴.۴
۲۰	تحلیل سیستم فعلی	۱.۴.۴
۲۰	شناسایی نقاط ضعف	۲.۴.۴
۲۰	طراحی مجدد معماری و پیاده سازی	۳.۴.۴
۲۰	استراتژی های مهاجرت	۴.۴.۴
۲۱	ابزارها و تکنیک های بازطراحی	۵.۴
۲۱	بازآرایی (Refactoring)	۱.۵.۴

۲۱	مهندسی معکوس (Reverse Engineering)	۲.۵.۴
۲۱	مهاجرت (Migration)	۳.۵.۴
۲۲	معیارهای تصمیم‌گیری برای بازطراحی	۶.۴
۲۲	هزینه (Cost)	۱.۶.۴
۲۳	زمان (Time)	۲.۶.۴
۲۳	ریسک (Risk)	۳.۶.۴
۲۳	اثر بر کیفیت (Effect) on Quality	۴.۶.۴
۲۳	مطالعه موردی	۷.۴
۲۳	بازطراحی اپلیکیشن PayPal	۱.۷.۴
۲۴	بازطراحی بانکداری برای کسب‌وکارهای کوچک (فوربیکس)	۲.۷.۴
۲۵	نتیجه‌گیری نهایی و توصیه‌ها برای تیم‌های توسعه	۸.۴
۲۵	ارزیابی واقع‌بینانه و مبتنی بر داده	۱.۸.۴
۲۶	اولویت‌بندی و رویکرد تدریجی	۲.۸.۴
۲۶	سرمایه‌گذاری بر روی اتوماسیون	۳.۸.۴
۲۶	مستندسازی همگام با توسعه	۴.۸.۴
۲۶	در نظر گرفتن پیامدهای فرهنگی	۵.۸.۴
۲۷	۵ چرایی نیاز به مهندسی معکوس در تکامل نرم‌افزار	
۲۷	مقدمه و تعریف مهندسی معکوس	۱.۵
۲۷	تمایز آن با Reengineering و Refactoring	۱.۱.۵
۲۸	دلایل نیاز به مهندسی معکوس	۲.۵
۲۸	فقدان مستندات یا مستندات ناقص	۱.۲.۵
۲۸	تحلیل سیستم‌های قدیمی (Legacy Systems)	۲.۲.۵
۲۹	درک ساختار و منطق سیستم‌های موجود	۳.۲.۵
۲۹	تسهیل مهاجرت به فناوری‌های جدید	۴.۲.۵

۲۹	چالش‌ها و محدودیت‌ها	۳.۵
۳۱	مطالعه موردی (Case Study)	۴.۵

فهرست تصاویر

۱۱	۱.۲ یک شکل نمونه برای نمایش
----	---------------------------------------

فصل ۱

فرایندهای مهندسی نرم افزار و چرخه های تکامل تا پیدایش

این فصل شامل مقدمه ای بر موضوع گزارش است.

در این فصل به مقدمه ای بر موضوع گزارش می پردازیم.

- هدف از انجام پروژه
- روش های استفاده شده
- نتایج اصلی به دست آمده
- نتیجه گیری کلی

۱.۱ پیش زمینه

در این بخش پیش زمینه و انگیزه انجام پروژه توضیح داده می شود.

۱.۱.۱ تاریخچه

تاریخچه موضوع مورد بررسی در این قسمت آورده می شود. می توان به کارهای گذشته و پیشرفت های صورت گرفته اشاره کرد.

۲.۱.۱ اهمیت موضوع

توضیح اهمیت و کاربردهای موضوع در این قسمت قرار می گیرد. برای مثال:

- کاربرد در صنعت
- کاربرد در تحقیقات علمی
- کاربرد در زندگی روزمره

می توان از [۱۸] برای ارجاع به منابع استفاده کرد.

۲.۱ اهداف پروژه

اهداف اصلی این پروژه عبارتند از:

۱. بررسی و مطالعه موضوع اصلی
۲. طراحی و پیاده سازی راه حل پیشنهادی
۳. ارزیابی و مقایسه نتایج
۴. ارائه پیشنهادات برای کارهای آینده

۱.۲.۱ اهداف کوتاه مدت

اهداف کوتاه مدت شامل موارد زیر است:

- مطالعه ادبیات موضوع
- آشنایی با ابزارها و تکنیک های مورد نیاز

۲.۲.۱ اهداف بلندمدت

اهداف بلندمدت شامل:

- توسعه یک سیستم کامل
- انتشار نتایج در مجلات معتبر

۳.۱ ساختار گزارش

این گزارش در چندین فصل سازماندهی شده است:

فصل ۱ شامل مقدمه و اهداف پروژه است

فصل ۲ به بررسی مبانی نظری و کارهای مرتبط می پردازد

فصل ۳ نتایج و پیشنهادات را ارائه می دهد

در پایان نیز منابع و مراجع استفاده شده آورده شده است.

فصل ۲

مشکلات مطرح در چرخه‌های توسعه و تکامل نرم‌افزار

در این فصل به بررسی مبانی نظری و کارهای انجام شده قبلی می‌پردازیم.

۱.۲ مفاهیم پایه

در این بخش مفاهیم و تعاریف پایه‌ای ارائه می‌شود.

۱.۱.۲ تعاریف اولیه

تعریف ۱.۲. یک تعریف نمونه عبارت است از...

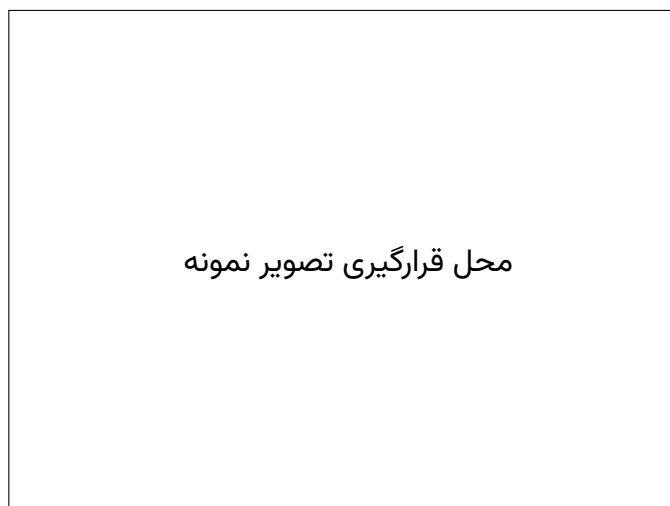
مثال ۱.۲. مثالی برای توضیح بهتر مفهوم: می‌توان فرمول ریاضی نوشت: $f(x) = x^2 + 2x + 1$

۲.۱.۲ قضایای اساسی

قضیه ۱.۲. اگر a و b دو عدد حقیقی باشند، آنگاه:

$$(a + b)^2 = a^2 + 2ab + b^2$$

می‌توان به شکل‌ها نیز اشاره کرد، مانند شکل ۱.۲.



شکل ۱.۲: یک شکل نمونه برای نمایش

۲.۲ مرور ادبیات

مرور کارهای انجام شده در این حوزه:

۱.۲.۲ کارهای پیشین

محققان مختلفی در این زمینه فعالیت کرده‌اند. برای مثال:

- [۱۸]: ارائه روش نوین برای حل مسئله

- کارهای دیگر در این زمینه

۲.۲.۲ مقایسه روش‌ها

جدول زیر مقایسه‌ای بین روش‌های مختلف ارائه می‌دهد:

همانطور که در جدول ۱.۲ مشاهده می‌شود، هر روش مزایا و معایب خود را دارد.

۳.۲ روش‌های موجود

در این بخش روش‌های موجود برای حل مسئله بررسی می‌شود.

جدول ۱.۲: مقایسه روش‌های مختلف

روش	دقت	سرعت	پیچیدگی
روش الف	بالا	متوسط	کم
روش ب	متوسط	بالا	متوسط
روش ج	بالا	کم	زیاد

۱.۳.۲ روش اول

توضیحات مربوط به روش اول:

۱. گام اول: تعریف مسئله

۲. گام دوم: جمع‌آوری داده‌ها

۳. گام سوم: پردازش و تحلیل

۴. گام چهارم: ارائه نتایج

۲.۳.۲ روش دوم

روش دوم رویکرد متفاوتی دارد:

الگوریتم پیشنهادی به شکل زیر است:

```

1 def algorithm(data):
2     result = []
3     for item in data:
4         if item > threshold:
5             result.append(item)
6     return result

```

Listing 2.1:

۳.۳.۲ مزایا و معایب

هر یک از روش‌های ذکر شده مزایا و معایب خاص خود را دارند که باید در انتخاب روش مناسب مدنظر قرار گیرند.

فصل ۳

DevOps و نقش آن در فرایند تکامل نرم افزار

۱.۳ مقدمه و تعریف DevOps

DevOps یک فرهنگ، فلسفه و مجموعه‌ای از روش‌ها و ابزارها است که هدف اصلی آن، یکپارچه‌سازی و خودکارسازی فرآیندهای بین تیم‌های توسعه نرم افزار (Development) و عملیات فناوری اطلاعات (Operations) است. در مدل سنتی، این دو تیم جدا از هم عمل می‌کردند که منجر به کندی، خطاهای بیشتر و هماهنگی دشوار می‌شد. ظهور DevOps پاسخی به این چالش‌ها بود تا با ایجاد همکاری و مسئولیت مشترک، شکاف بین ساخت نرم افزار و اجرای پایدار آن را از بین ببرد. در نهایت، DevOps به سازمان‌ها این توانایی را می‌دهد که نرم افزارها را سریع‌تر، قابل اطمینان‌تر و با کیفیت بالاتر در اختیار کاربران قرار دهند.

۲.۳ فلسفه DevOps و ارتباط آن با Agile

فلسفه DevOps بر پایه اصولی استوار است که فرهنگ همکاری، خودکارسازی و بهبود مستمر را ترویج می‌دهد. این فلسفه را می‌توان در "حلقه بی‌پایان" عملیات DevOps (که شامل مراحل برنامه‌ریزی، توسعه، استقرار و نظارت است) و همچنین در "سه راهی" معروف آن (جریان، Flow، بازخورد - Feed back) و یادگیری مستمر (Continuous Learning)) خلاصه کرد.

ارتباط DevOps با متدولوژی Agile بسیار عمیق است. Agile بر انعطاف‌پذیری، تحویل تدریجی و پاسخگویی به تغییرات در طول فرآیند توسعه تأکید دارد. DevOps این فلسفه را گسترش می‌دهد و

آن را به فرآیند استقرار و عملیات پس از توسعه تسری می‌بخشد. در حقیقت، DevOps مکمل Agile است؛ در حالی که Agile سرعت و کیفیت توسعه را افزایش می‌دهد، DevOps تضمین می‌کند که این تغییرات سریع می‌توانند به صورت ایمن و پایدار در محیط تولید مستقر شوند. بنابراین، می‌توان DevOps را به عنوان ادامه طبیعی و ضروری جنبش Agile در نظر گرفت که تمرکز آن بر روی کل چرخه عمر نرم افزار است.

۳.۳ چرخه عمر DevOps

چرخه عمر DevOps یک فرآیند تکراری و مستمر است که مراحل مختلفی از ایده تا تحویل نرم افزار و نظارت بر آن را در بر می‌گیرد. این چرخه با استفاده از ابزارهای خودکار به هم پیوسته، جریان ارزش را سریع و کارآمد می‌کند.

• برنامه‌ریزی (Plan)

در این فاز اولیه، اهداف پروژه تعریف، وظایف زمان‌بندی و پیشرفت کار رهگیری می‌شود. این مرحله تضمین می‌کند که همه اعضای تیم از اهداف کسب‌وکار و برنامه‌های فنی آگاه هستند. **ابزارها:** از ابزارهایی مانند Jira برای ردیابی Issues و مدیریت پروژه و Confluence برای مستندسازی و همکاری استفاده می‌شود.

• توسعه (Code)

توسعه‌دهندگان در این مرحله نرم افزار را می‌نویسند. برای اطمینان از سازگاری و قابلیت تکرار محیط‌های توسعه، از ابزارهای خاصی استفاده می‌شود. **ابزارها:** Docker برای بسته‌بندی نرم افزار در کانتینرهای سبک و قابل حمل، Kubernetes برای مدیریت و خودکارسازی این کانتینرها، و Puppet & Ansible برای مدیریت پیکربندی و خودکارسازی زیرساخت به کار می‌روند.

• یکپارچه‌سازی مستمر (Continuous Integration)

این تمرین شامل ادغام مکرر کد نوشته‌شده توسط تمام توسعه‌دهندگان به یک ریپازیتوری مشترک است. پس از هر ادغام، فرآیندهای ساخت و تست به طور خودکار اجرا می‌شوند تا خطاها در اسرع وقت شناسایی شوند. CI تضمین می‌کند که کدها به طور مداوم با یکدیگر یکپارچه شده و از بروز تعارضات بزرگ در آینده جلوگیری می‌کند.

• تحویل مستمر (Continuous Delivery)

CD گام بعدی پس از CI است. این تمرین تضمین می‌کند که پس از هر ادغام موفقیت‌آمیز کد،

می‌توان نرم‌افزار را در هر لحظه و با کمترین تلاش به صورت دستی در محیط تولید منتشر کرد. در تحویل مستمر، فرآیند استقرار تا مرحله نهایی خودکار است، اما انتشار نهایی در محیط تولید به صورت دستی و با تأیید یک انسان انجام می‌شود.

• استقرار مستمر (Continuous Deployment)

این پیشرفته‌ترین مرحله است که در آن، هر تغییری که از تست‌ها در مراحل CI/CD موفقیت‌آمیز عبور کند، به طور خودکار در محیط تولید مستقر می‌شود. در این مدل، هیچ مداخله دستی در فرآیند استقرار وجود ندارد و انتشار نرم‌افزار به یک رویداد عادی و روزمره تبدیل می‌شود. این امر سرعت ارائه ارزش به کاربر نهایی را به حداکثر می‌رساند.

ابزارهای CI/CD: از ابزارهایی مانند Jenkins، GitHub Actions/GitLab CI و CircleCI برای خودکارسازی کامل خط لوله از یکپارچه‌سازی تا استقرار استفاده می‌شود.

• نظارت و بازخورد (Monitoring & Feedback)

پس از استقرار نرم‌افزار در محیط تولید، عملکرد آن تحت نظارت دقیق قرار می‌گیرد تا از پایداری و سلامت سرویس اطمینان حاصل شود. داده‌های مربوط به عملکرد برنامه، زیرساخت و تجربه کاربر جمع‌آوری و تجزیه و تحلیل می‌شوند. این داده‌ها به صورت یک حلقه بازخورد (Feed-back Loop) به تیم‌های توسعه و برنامه‌ریزی بازمی‌گردند تا برای بهبود مستمر محصول و رفع مشکلات در چرخه‌های بعدی مورد استفاده قرار گیرند.

ابزارها: Grafana برای تجسم متریک‌ها، Stack Elastic برای مدیریت و تحلیل لاگ‌ها، Prometheus برای مانیتورینگ و هشدار و Sentry برای ردیابی خطاها در لحظه استفاده می‌شوند.

فصل ۴

چرایی نیاز به بازطراحی در پیاده‌سازی نرم‌افزار

۱.۴ مقدمه

سیستم‌های نرم‌افزاری، برخلاف دارایی‌های فیزیکی که دچار فرسایش مکانیکی می‌شوند، به مرور زمان کارایی خود را در انطباق با واقعیت‌های تجاری و بستر فناورانه از دست می‌دهند. این پدیده، که اغلب به آن «کهنگی نرم‌افزاری» گفته می‌شود، منجر به افزایش فزاینده در هزینه‌های عملیاتی و نگهداری می‌گردد. برآوردها نشان می‌دهد که نگهداری نرم‌افزار به‌عنوان پرهزینه‌ترین فاز چرخه حیات نرم‌افزار، تقریباً ۶۰ درصد از کل تلاش‌های صورت گرفته در این چرخه را به خود اختصاص می‌دهد.

سازمان‌ها در محیط‌های رقابتی و نظارتی امروز، تحت فشار مستمر برای افزایش چابکی و پاسخگویی به تغییرات بازار، مقررات جدید، و نیازهای در حال تحول کاربران قرار دارند. زمانی که سیستم‌های قدیمی (Legacy Systems) به مانعی برای نوآوری تبدیل می‌شوند و بخش نامتناسبی از بودجه را مصرف می‌کنند، بازمهندسی (Reengineering) به یک ضرورت استراتژیک تبدیل می‌گردد. هدف از بازمهندسی، نه صرفاً تولید ویژگی‌های جدید، بلکه بازیابی و طولانی کردن عمر سیستم‌های حیاتی است، ضمن کاهش هزینه‌های بالای نگهداری.

بازمهندسی، اساساً یک سرمایه‌گذاری در مدیریت ریسک و بهینه‌سازی مالی محسوب می‌شود. زمانی که هزینه‌های نگهداری بیش از نیمی از بودجه توسعه را می‌بلعد، این هزینه عملاً منابعی را که می‌توانست صرف نوآوری شود، از بین می‌برد (هزینه فرصت). بنابراین، بازمهندسی به عنوان راهکاری برای تثبیت سازمانی، کاهش ریسک‌های شکست سیستمی، و تضمین انطباق با قوانین، بر تحویل ویژگی‌های فوری اولویت می‌یابد.

۲.۴ تعریف بازطراحی (Redesign) / (Reengineering)

بازمهندسی نرم‌افزار، فرآیند بررسی و تغییر یک سیستم موجود با هدف پیاده‌سازی آن در یک فرم جدید یا تطبیق داده شده است. این فرآیند از نرم‌افزار و مستندات موجود استفاده می‌کند تا نیازمندی‌ها و طراحی سیستم هدف را تولید کند.

۱.۲.۴ بازمهندسی در برابر مهندسی رو به جلو

برخلاف مهندسی رو به جلو (Forward Engineering) که با یک سند مشخصات تعریف‌شده آغاز می‌شود، بازمهندسی با سیستم موجود به عنوان «مشخصات» خود آغاز شده و از طریق فرآیندهای درک و تبدیل، سیستم هدف را استخراج می‌کند.

۲.۲.۴ مهندسی معکوس (بازیابی طراحی)

مرحله حیاتی که بازمهندسی را تعریف می‌کند، بازیابی طراحی یا مهندسی معکوس (Reverse Engineering) است. این مرحله برای بازیابی منطق و چرایی تصمیمات معماری از دست‌رفته که در طول پیاده‌سازی اولیه اتخاذ شده‌اند، ضروری است. قبل از شروع هرگونه کار فنی، زمینه و هدف بازمهندسی باید در چارچوب اهداف کلان سازمانی تعریف شود.

۳.۴ دلایل اصلی نیاز به بازطراحی

۱.۳.۴ تغییر نیازمندی‌ها

تغییر در نیازمندی‌ها (ناشی از تکامل نیازهای کاربر یا تغییر بازار و مقررات) اجتناب‌ناپذیر است و ایجاد تغییرات دیر هنگام می‌تواند هزینه‌های توسعه را تا ۳۰ درصد افزایش دهد. استراتژی دفاعی، طراحی معماری بر اساس تجزیه مبتنی بر نوسان (Volatility-Based Decomposition) است. این اصل مستلزم کپسوله‌سازی اجزای مستعد تغییر برای جلوگیری از نشت تغییرات در سراسر سیستم و افزایش مقاومت در برابر انحراف ویژگی است. در غیر این صورت، سیستم با بدهی معماری روبه‌رو می‌شود.

۲.۳.۴ فناوری های جدید

پذیرش فناوری های جدید نیروی محرکه قوی برای بازمهندسی است و اغلب برای رفع محدودیت های عملکردی و مقیاس پذیری سیستم های قدیمی ضروری است. این شامل گذار به برنامه های ابرمحور (Cloud-Native) و معماری میکروسرویس ها می شود. زیرساخت های قدیمی IT اغلب نیازمند سفارشی سازی های گسترده و راه حل های میان افزار پیچیده هستند تا قابلیت همکاری با ابزارهای دیجیتال جدید تضمین شود.

۳.۳.۴ ضعف معماری اولیه

نیاز به بازطراحی اغلب ریشه در پذیرش الگوهای ضدطراحی (Anti-Patterns) دارد. بدنام ترین آن، الگوی «توپ گلی بزرگ» (Big Ball of Mud) - BBoM است که در آن ساختار سیستم فاقد تفکیک مسئولیت ها و سازماندهی مشخص است. این وضعیت باعث انباشت بدهی فنی شده و اصلاح سیستم را دشوار می کند.

۴.۳.۴ انباشت بدهی فنی

بدهی فنی، هزینه استعاری تصمیم های کوتاه مدت است. همانند بدهی مالی، بهره مند است و با گذشت زمان رشد می کند. بر اساس گزارش ها، حدود ۴۲٪ از زمان توسعه دهندگان صرف مقابله با بدهی فنی می شود. زمانی که بدهی فنی از ۵۰٪ ارزش فناوری یک سیستم فراتر رود، بازمهندسی کامل از نظر اقتصادی توجیه پذیر است.

۴.۴ مراحل بازطراحی نرم افزار

فرآیند بازمهندسی نرم افزار یک روش شناسی ساختاریافته است که از تحلیل سیستم موجود آغاز شده و تا پیاده سازی استراتژی های مهاجرت با کمترین ریسک ادامه می یابد.

۱.۴.۴ تحلیل سیستم فعلی

ارزیابی جامع سیستم فعلی گام اول است تا مشخص شود کدام بخش ها ارزش حفظ کردن دارند. این تحلیل شامل سنجش شاخص هایی مانند زمان پاسخ، درصد در دسترس بودن (uptime) و آزمون بار اوج (Peak Load) Testing است. همچنین ارزیابی هزینه کل مالکیت (TCO) برای تحلیل اقتصادی سیستم ضروری است.

۲.۴.۴ شناسایی نقاط ضعف

در غیاب مستندات کامل، مهندسی معکوس برای درک منطق و بازیابی طراحی به کار می رود. ابزارهای هوش مصنوعی مانند Copilot GitHub می توانند وابستگی ها، فراخوانی ها و ساختارهای منطقی را استخراج کرده و مستندات به روز تولید کنند.

۳.۴.۴ طراحی مجدد معماری و پیاده سازی

بازطراحی معماری معمولاً شامل گذار از ساختارهای یکپارچه به معماری های توزیع شده مانند میکروسرویس ها است. چالش های کلیدی این گذار عبارتند از:

- افزایش هزینه های زیرساختی و تست برای هر سرویس جدید.
- از دست رفتن تضمین های ACID و نیاز به مدیریت ثبات نهایی (Eventual Consistency).
- استفاده از الگوهایی مانند ساگا (Saga) و تضمین هم توانایی (Idempotency) برای هماهنگی تراکنش ها.

۴.۴.۴ استراتژی های مهاجرت

انتخاب روش مهاجرت بستگی به میزان تحمل ریسک سازمان دارد:

- **مهاجرت انفجار بزرگ (Big Bang Migration):** سوئیچ فوری از سیستم قدیمی به سیستم جدید؛ پرریسک و مناسب سیستم های غیر بحرانی.
- **مهاجرت افزایشی (Incremental Migration) یا الگوی انجیر خفه کننده:** جایگزینی تدریجی بخش ها و اجرای همزمان سیستم های قدیم و جدید برای کاهش ریسک.

۵.۴ ابزارها و تکنیک‌های بازطراحی

فرآیند بازطراحی و نگهداری سیستم‌های قدیمی معمولاً با به کارگیری مجموعه‌ای از ابزارها و تکنیک‌های تخصصی انجام می‌شود...

۱.۵.۴ بازآرایی (Refactoring)

بازآرایی فرآیند بازسازی (restructuring) ساختار داخلی کد بدون تغییر رفتار خارجی آن است... [۱].

۲.۵.۴ مهندسی معکوس (Reverse Engineering)

مهندسی معکوس فرآیند استخراج طراحی، معماری و مشخصات یک سیستم از کد منبع موجود است...

۳.۵.۴ مهاجرت (Migration)

مهاجرت به فرآیند انتقال یک سیستم نرم‌افزاری از یک محیط تکنولوژیکی قدیمی به یک محیط جدیدتر... [۲].

کتابنامه

[۱] IBM. (۲۰۲۳). Code Refactoring. Retrieved from <https://www.ibm.com/think/topics/code-refactoring>

[۲] GeeksforGeeks. (۲۰۲۳). Best Practices for Data Migration. Retrieved from <https://www.geeksforgeeks.org/blogs/best-practices-for-data-migration>

۶.۴ معیارهای تصمیم‌گیری برای بازطراحی

تصمیم‌گیری برای انجام فرآیند بازطراحی یک سیستم نرم‌افزاری، نیازمند سنجش و ارزیابی دقیق چندین معیار حیاتی است تا بتوان توجیه فنی و اقتصادی آن را به درستی بررسی کرد. مهم‌ترین این معیارها عبارتند از:

۱.۶.۴ هزینه (Cost)

برآورد دقیق تمامی هزینه‌های مستقیم و غیرمستقیم پروژه بازطراحی امری ضروری است. این هزینه‌ها شامل دستمزد تیم توسعه، هزینه‌های مربوط به خرید یا اجاره ابزارها و زیرساخت‌های جدید، هزینه‌های آموزش پرسنل و همچنین هزینه‌های احتمالی توقف یا کاهش عملکرد سیستم در حین اجرای پروژه می‌شود. این معیار باید در مقابل هزینه‌های ادامه کار با سیستم قدیمی (مانند هزینه‌های بالای نگهداری و رفع نقص) سنجیده شود.

۲.۶.۴ زمان (Time)

تخمین مدت زمان مورد نیاز برای تکمیل فرآیند بازطراحی از اهمیت بالایی برخوردار است. یک برنامه‌ریزی واقع‌بینانه باید شامل مراحل تحلیل، طراحی، پیاده‌سازی، تست و استقرار باشد. زمان‌بندی طولانی می‌تواند منجر به منسوخ شدن فناوری‌های به کار رفته در طول اجرای پروژه شود، در حالی که زمان‌بندی بسیار فشرده نیز کیفیت نهایی را به خطر می‌اندازد.

۳.۶.۴ ریسک (Risk)

ارزیابی ریسک‌های بالقوه در موفقیت پروژه بازطراحی یک گام کلیدی است. این ریسک‌ها می‌توانند شامل پیچیدگی فنی بالای سیستم، legacy از دست دادن مهارت‌های تخصصی مورد نیاز، بروز مشکلات غیرمنتظره در حین مهاجرت داده‌ها، و مقاومت کاربران در برابر پذیرش سیستم جدید باشد. شناسایی این ریسک‌ها و برنامه‌ریزی برای مدیریت آن‌ها شانس موفقیت پروژه را افزایش می‌دهد.

۴.۶.۴ اثر بر کیفیت (Effect on Quality)

در نهایت، باید تأثیر مثبت بازطراحی بر کیفیت محصول نهایی به وضوح تعریف و اندازه‌گیری شود. این بهبود کیفیت می‌تواند به صورت افزایش کارایی، (Performance) افزایش قابلیت اطمینان (Reliability)، (Scalability) افزایش امنیت، بهبود قابلیت نگهداری (Maintainability) و افزایش قابلیت گسترش (ability) سیستم ظاهر شود. این معیار نهایی، توجیه اصلی برای سرمایه‌گذاری روی پروژه بازطراحی محسوب می‌شود.

۷.۴ مطالعه موردی

بر اساس نتایج جستجو، اطلاعات مربوط به بازطراحی پی‌پال بیشتر بر به‌روزرسانی تجربه کاربری... [۱].

۱.۷.۴ بازطراحی اپلیکیشن PayPal

هدف اصلی از بازطراحی پی‌پال، تبدیل آن از یک ابزار ساده برای انتقال پول به یک "راهنمای سلامتی مالی"...

۲.۷.۴ بازطراحی بانکداری برای کسب‌وکارهای کوچک (فوربیکس)

در ایران، نمونه بارز بازطراحی در سیستم بانکی، ظهور "نئوبانک‌های کسب‌وکاری" مانند فوربیکس است...

کتابنامه

[۱] Elastic Path. (۲۰۲۳). Re- Redesign. Checkout Radical PayPal's Deconstructing .
<https://www.elasticpath.com/blog/deconstructing-paypals-radical-checkout-redesign>

[۲] Think Company. (۲۰۲۳). Redesign Digital Successful a for Steps ۶ .
<https://www.thinkcompany.com/blog/۶-steps-for-a-successful-digital-redesign-project>

۸.۴ نتیجه‌گیری نهایی و توصیه‌ها برای تیم‌های توسعه

بازطراحی نرم‌افزار یک سرمایه‌گذاری استراتژیک برای حفظ سلامت، کارایی و طول عمر سیستم‌های نرم‌افزاری محسوب می‌شود. همان‌طور که در بخش‌های پیشین بررسی شد، این فرآیند با استفاده از تکنیک‌هایی مانند بازآرایی، مهندسی معکوس و مهاجرت، و با در نظرگیری معیارهای حیاتی چون هزینه، زمان، ریسک و اثر بر کیفیت انجام می‌پذیرد. بررسی سیستم‌هایی مانند پی‌پال نیز نشان می‌دهد که یک بازطراحی موفق می‌تواند منجر به افزایش رضایت کاربر، بهبود قابلیت نگهداری و کسب مزیت رقابتی پایدار شود. در پایان، موارد کلیدی زیر می‌تواند راهنمای تیم‌های توسعه در این مسیر باشد:

۱.۸.۴ ارزیابی واقع‌بینانه و مبتنی بر داده

پیش از هر اقدامی، با استفاده از معیارهای کمی (مانند اندازه پیچیدگی کد، تعداد باگ‌ها، هزینه نگهداری) و کیفی (رضایت کاربران و تیم توسعه) به ارزیابی دقیق نیازمندی‌های سیستم موجود بپردازید. این ارزیابی، مبنای علمی و متقاعدکننده‌ای برای تصمیم‌گیری در مورد لزوم و دامنه بازطراحی فراهم می‌کند.

۲.۸.۴ اولویت‌بندی و رویکرد تدریجی

بازطراحی کامل یک سیستم بزرگ در یک بازه زمانی کوتاه، ریسک بسیار بالایی دارد. توصیه می‌شود پروژه به بخش‌های کوچک‌تر و مستقل تقسیم شده و به صورت تدریجی و با اولویت‌بندی بر اساس مازول‌هایی که بیشترین مشکل را ایجاد می‌کنند، اجرا شود. این رویکرد، مدیریت پروژه را آسان‌تر کرده و امکان دریافت بازخورد سریع را فراهم می‌کند.

۳.۸.۴ سرمایه‌گذاری بر روی اتوماسیون

استقرار یک خط لوله قوی یکپارچه‌سازی و تحویل مستمر (CI/CD) و یک مجموعه جامع از آزمون‌های خودکار را در اولویت قرار دهید. این امر با اطمینان از اینکه تغییرات کد، عملکرد موجود را خراب نمی‌کند، ایمنی و سرعت فرآیند بازطراحی را به طور چشمگیری افزایش می‌دهد.

۴.۸.۴ مستندسازی همگام با توسعه

فرآیند بازطراحی را فرصتی برای جبران کمبود مستندات سیستم قدیمی بدانید. همگام با پیاده‌سازی کد جدید، مستندات طراحی، معماری و نحوه راه‌اندازی را به روز کنید. این کار نگهداری سیستم را در آینده بسیار ساده‌تر خواهد کرد.

۵.۸.۴ در نظر گرفتن پیامدهای فرهنگی

بازطراحی تنها یک چالش فنی نیست، بلکه یک تغییر سازمانی است. تیم را از مزایای بلندمدت این کار آگاه سازید و برای پذیرش این تغییر و یادگیری فناوری‌ها یا روش‌های جدید، فرهنگسازی و آموزش لازم را فراهم کنید. موفقیت نهایی در گرو همراهی و مهارت تیم توسعه است.

در نهایت، بازطراحی را نه به عنوان یک هزینه، بلکه به عنوان یک ضرورت برای بقا و رشد نرم‌افزار در نظر بگیرید. یک برنامه‌ریزی دقیق، اجرای گام‌به‌گام و تمرکز بر کیفیت، می‌تواند عمر سیستم شما را طولانی کرده و ارزش آن را در بلندمدت به میزان قابل توجهی افزایش دهد.

فصل ۵

چرایی نیاز به مهندسی معکوس در تکامل نرم افزار

۱.۵ مقدمه و تعریف مهندسی معکوس

مهندسی معکوس (Reverse Engineering) در مهندسی نرم افزار به فرایندی گفته می شود که در آن یک نرم افزار موجود مورد تحلیل دقیق قرار می گیرد تا ساختار درونی، اجزا، وابستگی ها و منطق عملکرد آن شناخته شود، بدون این که لزوماً تغییری در سیستم ایجاد گردد [۱].

هدف اصلی مهندسی معکوس، بازیابی دانش از دست رفته یا مستندسازی نشده درباره ی سیستم است. به کمک مهندسی معکوس می توان فهمید که نرم افزار چگونه طراحی شده، اطلاعات چگونه در آن جریان دارد و بخش های مختلف آن چه ارتباطی با هم دارند.

برای مثال، اگر نرم افزاری در دسترس باشد ولی مستندات طراحی آن موجود نباشد، با مهندسی معکوس می توان از روی کدها و فایل های اجرایی، مستندات و مدل های طراحی را بازسازی کرد. این کار در پروژه هایی که نرم افزارهای قدیمی (Legacy Systems) مورد استفاده قرار می گیرند، بسیار اهمیت دارد [۲].

۱.۱.۵ تمایز آن با Reengineering و Refactoring

مفاهیم مهندسی معکوس، بازمهندسی (Reengineering) و بازآرایی کد (Refactoring) هرچند مشابه اند، ولی اهداف متفاوتی دارند.

- **مهندسی معکوس (Reverse Engineering):** هدف آن درک سیستم موجود است؛ یعنی بررسی و تحلیل بدون تغییر کد منبع.
- **بازمهندسی (Reengineering):** پس از شناخت کامل سیستم، آن را بازطراحی یا بازنویسی می‌کنیم تا عملکرد بهتر یا نگهداری آسان‌تری داشته باشد.
- **بازآرایی کد (Refactoring):** تمرکز بر بهبود ساختار درونی کد منبع است، بدون اینکه رفتار کلی نرم افزار تغییر کند.

می‌توان گفت:

مهندسی معکوس شناخت سیستم
بازمهندسی شناخت + تغییر ساختار کلی
بازآرایی اصلاح درونی کد بدون تغییر عملکرد

در چرخه‌ی عمر نرم افزار، مهندسی معکوس نقش مهمی در مرحله‌ی نگهداری (Maintenance) دارد، زیرا در این مرحله معمولاً نیاز به درک مجدد از ساختار و منطق سیستم احساس می‌شود.

۲.۵ دلایل نیاز به مهندسی معکوس

۱.۲.۵ فقدان مستندات یا مستندات ناقص

بسیاری از سیستم‌های نرم افزاری بدون مستندات کافی توسعه یافته‌اند یا مستندات آن‌ها در طول زمان از بین رفته است [۴]. در این شرایط، مهندسی معکوس به تیم توسعه کمک می‌کند تا از روی نرم افزار، مستندات طراحی و نمودارهای سیستم را بازسازی کند.

۲.۲.۵ تحلیل سیستم‌های قدیمی (Legacy Systems)

در سازمان‌ها هنوز از سیستم‌هایی استفاده می‌شود که بر پایه فناوری‌های قدیمی ساخته شده‌اند. مهندسی معکوس به توسعه‌دهندگان کمک می‌کند تا ساختار کلی این سیستم‌ها را درک کنند و در صورت نیاز آن‌ها را به فناوری‌های جدید منتقل نمایند.

۳.۲.۵ درک ساختار و منطق سیستم‌های موجود

گاهی نرم‌افزار توسط تیم‌های مختلف توسعه یافته و در نتیجه کدها پیچیده و نامنظم شده‌اند. مهندسی معکوس ابزاری برای درک ارتباط بین ماژول‌ها، کلاس‌ها و داده‌ها فراهم می‌کند و درک درستی از منطق سیستم به تیم توسعه می‌دهد.

۴.۲.۵ تسهیل مهاجرت به فناوری‌های جدید

تغییر پلتفرم‌ها و ابزارها اجتناب‌ناپذیر است. برای مثال، ممکن است سازمانی بخواهد نرم‌افزار خود را از نسخه‌ی دسکتاپ به تحت وب منتقل کند. مهندسی معکوس امکان تحلیل دقیق سیستم فعلی را فراهم می‌کند تا مهاجرت بدون خطا و از دست دادن داده انجام گیرد [۴].

۳.۵ چالش‌ها و محدودیت‌ها

در این فصل، به بررسی چالش‌ها و محدودیت‌های موجود در به‌کارگیری رویکرد DevOps در کنار مهندسی معکوس نرم‌افزار پرداخته می‌شود. هدف از این بخش، شناسایی عواملی است که می‌توانند بر اثربخشی، اعتبار و قابلیت تعمیم نتایج حاصل از اجرای این رویکردها تاثیرگذار باشند. محدودیت‌های شناسایی‌شده در چهار محور اصلی شامل مسائل حقوقی و مالکیت فکری، هزینه و زمان‌بر بودن فرآیند، تفسیر نادرست منطق کد، و ریسک‌های امنیتی مورد تحلیل قرار گرفته‌اند. هر یک از این عوامل، به‌صورت مستقیم یا غیرمستقیم می‌توانند مانعی در مسیر پیاده‌سازی مؤثر DevOps و بازمهندسی سیستم‌های نرم‌افزاری در محیط‌های واقعی ایجاد کنند و ضرورت به‌کارگیری رویکردی میان‌رشته‌ای و تصمیم‌گیری دقیق در این زمینه را نشان می‌دهند.

مشکلات حقوقی و مالکیت فکری

مهندسی معکوس نرم‌افزار معمولاً با محدودیت‌های قانونی و حقوقی گسترده‌ای همراه است. بسیاری از سیستم‌های نرم‌افزاری موجود در شرکت‌های بزرگ، تحت مجوزهای اختصاصی، قراردادهای توسعه یا توافق‌نامه‌های عدم افشا (NDA) طراحی و نگهداری می‌شوند. در چنین شرایطی، هرگونه تلاش برای تحلیل معکوس، استخراج کد منبع یا بازطراحی اجزای نرم‌افزار می‌تواند نقض حق مالکیت فکری محسوب شود [۲]. به‌ویژه در کشورهایی با نظام حقوقی سختگیرانه مانند ایالات متحده، قوانین

کپی‌رایت و پتنت می‌توانند مانع هرگونه مهندسی معکوس حتی با هدف بهبود سازگاری یا پایداری سیستم شوند [۱۶]. از سوی دیگر، محدودیت‌های بین‌المللی نیز موجب پیچیدگی بیشتر می‌شوند. در محیط‌های چندملیتی، تعریف و اجرای حقوق مالکیت نرم‌افزار می‌تواند میان کشورها متفاوت باشد و در نتیجه، دسترسی به کد یا داده‌های واقعی جهت تحلیل علمی محدود گردد [۱۴]. همچنین، تضاد میان نوآوری و حفاظت از مالکیت فکری چالشی فلسفی در این حوزه ایجاد کرده است. برخی محققان معتقدند که قوانین سختگیرانه‌ی IP، پیشرفت فناوری را کند می‌کنند زیرا مانع از یادگیری از سیستم‌های پیشین می‌شوند [۷]. در مقابل، گروهی دیگر بر این باورند که مهندسی معکوس بی‌رویه بدون رعایت مجوزها، ریسک سرقت فناوری و نقض حقوق مولف را افزایش می‌دهد. در این تحقیق نیز، به دلیل عدم دسترسی به داده‌های واقعی شرکت‌ها و محدودیت حقوقی تحلیل نمونه‌های صنعتی، بررسی‌های تجربی محدود به جنبه‌های نظری باقی مانده است.

هزینه و زمان بر بودن

از منظر اقتصادی و اجرایی، بازمهندسی نرم‌افزار فرآیندی پرهزینه و زمان‌بر است که نیازمند منابع انسانی، زیرساختی و مالی قابل‌توجهی است [۱۵]. در گزارش‌های صنعتی آمده است که شرکت‌ها سالانه میلیاردها دلار صرف نگهداری و بازطراحی سیستم‌های قدیمی خود می‌کنند و در بسیاری از موارد، هزینه‌ی بازمهندسی از هزینه‌ی توسعه‌ی مجدد سیستم جدید نیز بیشتر است [۱۷]. عوامل متعددی در افزایش این هزینه موثرند. برای نمونه، نبود مستندات کافی، نیاز به تحلیل وابستگی‌ها، بازسازی مدل داده‌ها، بازطراحی معماری و آزمون‌های مکرر همه باعث افزایش زمان اجرای پروژه می‌شوند. هرچه ساختار کد قدیمی‌تر و پیچیده‌تر باشد، زمان تحلیل و اصلاح آن به‌صورت تصاعدی افزایش می‌یابد [۱۱]. به‌علاوه، یکی از مشکلات رایج در پروژه‌های بازمهندسی، برآورد نادرست هزینه و زمان است. بسیاری از سازمان‌ها در آغاز پروژه تخمین دقیقی از میزان بدهی فنی و سطح ناسازگاری فناوری ندارند؛ در نتیجه، با افزایش غیرمنتظره‌ی هزینه‌ها، پروژه در میانه‌ی راه متوقف یا محدود می‌شود [۳].

تفسیر نادرست از منطق کد

در فرآیند بازمهندسی، درک صحیح از منطق درونی نرم‌افزار اهمیت حیاتی دارد. با این حال، بسیاری از سیستم‌های میراثی فاقد مستندات کامل هستند و مهندسان مجبورند رفتار سیستم را تنها از طریق تحلیل کد استنباط کنند. این امر منجر به تفسیر نادرست از منطق برنامه و روابط میان اجزای آن می‌شود [۹]. مطالعات نشان می‌دهند که درصد قابل‌توجهی از خطاهای به‌وجودآمده پس از بازمهندسی، ناشی از برداشت اشتباه از منطق کسب‌وکار و وابستگی‌های داخلی سیستم است [۱۰]. علاوه بر این، خروج

نیروهای کلیدی از سازمان و از بین رفتن دانش ضمنی باعث می شود که درک دقیق از چرایی و چگونگی تصمیمات گذشته از بین برود [۸]. ابزارهای خودکار تحلیل معنایی و مدل سازی معکوس هنوز در بسیاری از محیط های صنعتی به طور کامل توسعه نیافته اند، و این امر احتمال سوء برداشت از منطق کد را بیشتر می کند.

ریسک های امنیتی

ریسک های امنیتی از مهم ترین موانع در مسیر باز مهندسی و باز طراحی نرم افزار محسوب می شوند. بسیاری از سیستم های قدیمی بر پایه ی فناوری ها و چارچوب هایی بنا شده اند که دیگر به روزرسانی نمی شوند [۶]. این وضعیت باعث می شود که آسیب پذیری های شناخته شده برای مدت طولانی در سیستم باقی بمانند و مهاجمان بتوانند از آن ها سوء استفاده کنند [۵]. در فرآیند باز مهندسی، این خطر وجود دارد که مهاجرت داده ها، تقسیم مازول ها یا اتصال سیستم های جدید با سیستم های قدیمی، مسیرهای جدیدی برای نفوذ ایجاد کند. همچنین، اگر فرآیند DevOps به درستی با الزامات امنیتی یکپارچه نشود، اتوماسیون نادرست می تواند دروازه هایی برای نفوذ به محیط تولید باز کند [۱]. یکی دیگر از چالش های امنیتی، ضعف در مدیریت وصله های امنیتی است. پژوهش Dissanayake و همکاران [۴] نشان می دهد که کمتر از ۲۰ درصد از سازمان ها فرآیند وصله گذاری خود را به صورت نظام مند و خودکار انجام می دهند، که این امر احتمال بروز آسیب پذیری در چرخه ی باز مهندسی را افزایش می دهد.

۴.۵ مطالعه موردی (Case Study)

مثال از تحلیل معکوس یک سیستم قدیمی یا نرم افزار متن باز

یکی از مطالعات شاخص در این حوزه، پژوهش Reverse engineering a legacy software in a complex system: A systems engineering approach و Maximiliano Moraga است که توسط Yang-Yang Zhao در سال ۲۰۱۸ منتشر شده است؛ در این تحقیق یک نرم افزار میراثی که در قالب بخشی از سیستم پیچیده ای قرار داشت، مورد تحلیل معکوس قرار گرفت تا دلیل شکل گیری ساختار، منطق عملکرد، و جایگاه آن در بستر کلی سیستم بازشناسی شود [۱۲]. در این مطالعه، تیم محققان با استفاده از مدل CAFCR (Customer Objectives – Application – Function – Component – Resources) و ابزارهای مهندسی معکوس، توانستند نقشه راه مرحله ای برای ارتقای تدریجی و همزمان نگهداری و توسعه نرم افزار میراثی تدوین کنند [۱۲]. به عنوان مثال، ابتدا نمودارهای رابطه ای بین

مؤلفه‌ها، وابستگی‌های زمان‌بر و حرکت از معماری مونوپولی به معماری ماژولار استخراج شد، سپس بر اساس آن تصمیماتی برای بهبود کارایی، ارتقای قابلیت نگهداری و افزون‌کردن کارکردهای جدید اتخاذ گردید [۱۲]. مطالعه دیگری تحت عنوان Case Studies in Model-Driven Reverse Engineering توسط A. Pascal و همکاران در سال ۲۰۱۹ ارائه شده است که در آن بازمهندسی سه نرم‌افزار عملیاتی با استفاده از رویکرد مدل‌محور بررسی شده است؛ در این نمونه، استخراج مدل‌های سطح بالا، تحلیل ماژول‌ها و طراحی مجدد با هدف کاهش فرسایش معماری انجام شده است [۱۳]. این مثال‌ها نشان می‌دهند که تحلیل معکوس در نرم‌افزارهای میراثی صرفاً جهت استخراج کد نیست، بلکه درک منطق کسب‌وکار، وابستگی‌های پنهان، و ساختار معماری را نیز امکان‌پذیر می‌کند؛ ولی همزمان باید توجه داشت که هر پروژه پژوهشی یا صنعتی با محدودیت‌ها و پیچیدگی‌های خاص خود مواجه است.

بررسی خروجی‌های حاصل از فرآیند مهندسی معکوس

در پروژه Moraga Zhao، خروجی‌های مهمی حاصل شده است: از جمله بازسازی نمودار زمینه (con-text diagram) برای نرم‌افزار مورد بررسی، استخراج اهداف مشتریان، مشخص شدن معیارهای کیفیت در رابطه با بازار هدف و ترکیب آن با وابستگی فنی مؤلفه‌ها، که منجر به تدوین نقشه‌راه برای بازمهندسی تدریجی نرم‌افزار شد [۱۲]. این نقشه راه به شرکت امکان داد تا ضمن ادامه نگهداری سیستم قدیمی، به تدریج عملکردهای جدید را نیز افزوده و قابلیت نگهداری را ارتقا دهد. در مطالعه Pascal، یکی دیگر از خروجی‌های کلیدی، استخراج مدل‌های معماری (architecture models) و فرم‌های بصری وابستگی‌ها میان ماژول‌ها بود؛ این مدل‌ها کمک کردند تا مسیرهای پرتکرار تغییرات، نقاط بحرانی در سیستم و اجزایی که بیشترین پیچیدگی را داشتند شناسایی شوند [۱۳]. همچنین گزارش شده که این مدل‌سازی موجب کاهش هزینه نگهداری و کاهش فرسایش معماری شده است. علاوه بر این، خروجی‌های عملی دیگری نیز شامل مستندسازی بازگشتی (redocumentation) سیستم، بازیابی دانش ضمنی کارکنان قدیمی، انتقال آن به اعضای تیم جدید، کاهش وابستگی به افراد خاص، و آماده‌سازی سیستم برای استقرار روش‌های نوین مانند DevOps بود. به عنوان مثال، مکانیسم‌های اتوماسیون استقرار (CI/CD) و کانتینری‌سازی پس از مهندسی معکوس بهتر قابل پیاده‌سازی شدند زیرا ساختار ماژولار بهتر درک شده بود. با این حال، باید به این نکته نیز توجه شود که خروجی‌های مهندسی معکوس معمولاً به صورت کامل قابل تعمیم نیستند؛ یعنی مدل‌ها، نقشه‌ها و تصمیماتی که در یک سازمان حاصل شده، ممکن است در سازمان دیگر با زیرساختی متفاوت، قابل اجرا یا مؤثر نباشند. همچنین کیفیت خروجی‌ها وابسته به میزان دسترسی به کد، مستندسازی قبلی، همکاری تیم‌های پیشین، و ابزارهای تحلیل مورد استفاده است؛ در محیط‌هایی که مستندات کم است، خطای استخراج منطق می‌تواند زیاد شود.

کتاب نامه

- 2025 mitigations, and risks Key software: legacy using in Vulnerabilities Atiba. [1]
National Software. in Issues Property Intellectual Council. Research National [2]
.1991 D.C., Washington, Press, Academies
2025 them, avoid to how & systems legacy maintaining of costs V DevSquad. [3]
security Software Babar. Ali M. and Zahedi, M. Jayatilaka, A. Dissanayake, N. [4]
approaches, challenges, of review literature systematic a – management patch
.2020 preprint, arXiv practices. and tools
cyber modern fueling are software legacy and systems outdated How HeroDevs. [5]
.2024 attacks,
risk security cyber ticking a is hardware and software legacy How Integrity360. [6]
.2023 time-bomb,
minimize and restrictions Understanding law: engineering Reverse Watchdog. IP [7]
.2021 risks,
ap- and models concepts, re-engineering software of Overview Journal. JoIV [8]
.2023 proaches,
and debt technical in Lessons product: the becomes code legacy When Medium. [9]
.2023 opportunities, missed
during logic business misinterpreting of risks biggest The Blog. AI ModelCode [10]
.2024 modernization, legacy

- [۱۱] Legacy ModLogix. software re-engineering: Risks and mitigations. ۲۰۲۲.
- [۱۲] Zhao. Y. and Moraga M. Reverse engineering a legacy software in a complex system: an approach. International INCOSE Engineering Systems Symposium, A system: ۱۲۵۰-۱۲۶۴, ۲۸ volume, ۲۰۱۸.
- [۱۳] Pascal A. et al. Case studies in model-driven reverse engineering. In: Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Management (IC³KEM), ۲۰۱۹, ۷۳۱-۷۴۰, ۲ volume (IC³KEM), ۲۰۱۹.
- [۱۴] Quarkslab. Reverse engineering: A threat to intellectual property? innovations, ۲۰۲۳.
- [۱۵] RecordPoint. The hidden costs of maintaining legacy systems. ۲۰۲۴.
- [۱۶] TTC Consultants. Reverse engineering and intellectual property rights: Balancing the scales. ۲۰۲۴ considerations, legal and innovation.
- [۱۷] vFunction. How much does it cost to maintain legacy software? ۲۰۲۲.
- [۱۸] نویسندگان اول and نویسندگان دوم. عنوان مقاله نمونه. نام مجله، ۱۰(۲): ۱۲۳-۱۴۵، ۱۴۰۲.