Arman Process Journal 3(1), Spring 2022

Arman Process Journal
(APJ)

Homepage: www.armanprocessjournal.ir

Achieve the Alterability in Design of Software System Architecture

***A. Mohammadi*[*,1]**

[1] *School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran.*

Received: 3 March 2022
Accepted: 22 May 2022

KEYWORDS:

Software Systems
Alterability
Software Architecture
Modifications
Extensibility

ABSTRACT

Today, the increasing use of software systems in organizations, companies and small and large industries has left software developers confused about how to develop software with old development methods. With the increasing use of architecture-based process models, software architecture design has become particularly important challenge. Software architecture is one of the key parts of software production, especially its commercial type, which, of course, has been developed in recent years by creating classic models of software production to larger software. Software architecture is a structural system or structures of an operating system that includes software elements, properties visible from outside those elements, and the relationships between them. A good architectural design is a design that meets the quality needs expected by the customer. Achieving the alterability ability is one of the most important quality features in the design of modern software systems. In this article, first the various methods of software architecture design will be examined and then the qualitative feature of alterability and changeability will be introduced in detail. Finally, software architecture design based on achieving the alterability will be discussed.

[1] Corresponding author
a.mohammadi66@gmail.com

NUMBER OF REFERENCES
20

NUMBER OF FIGURES
0

NUMBER OF TABLES
0

نشریه تخصصی آرمان پردازش، دوره ۳، شماره ۱، بهار ۱۴۰۱

فصلنامه تخصصی آرمان پردازش

(APJ)

Homepage: www.armanprocessjournal.ir

دستیابی به قابلیت تغییر در طراحی معماری سیستم های نرم افزاری

علیرضا محمدی [*،۱]

[۱] دانشکده مهندسی کامپیوتر، دانشگاه علم و صنعت، تهران، ایران

چکیده

امروزه موج رو به افزایش استفاده از نرم‌افزار در سازمان‌ها، شرکت‌ها و صنایع کوچک و بزرگ توسعه‌دهندگان نرم‌افزار را دچار سردرگمی در چگونگی توسعه نرم‌افزار با روش‌های توسعه قدیمی نموده است. با گسترش روز افزون استفاده از مدل های فرایند مبتنی بر معماری در توسعه نرم افزار، بهبود طراحی معماری نرم افزار اهمیت ویژه ای یافته است. معماری نرم‌افزار از کلیدی‌ترین بخش‌های تولید نرم‌افزار مخصوصاً نوع تجاری آن است که البته در سالهای اخیر با ایجاد مدل‌های کلاسیک تولید نرم‌افزار به نرم‌افزارهای عظیم تر توسعه یافته است. معماری نرم‌افزار یک سیستم ساختاری یا ساختارهایی از یک سیستم عملیاتی است که عناصر نرم‌افزاری، خصوصیات قابل مشاهده از بیرون آن عناصر، و ارتباطات بین آنها را شامل می‌شود. یک طراحی معماری کارامد، طراحی است که نیازهای کیفی مورد انتظار مشتری را برآورده نماید. دستیابی به قابلیت تغییر در مشخصه های عملیاتی از مهم ترین ویژگی های کیفی در طراحی معماری سیستم های نرم افزاری مدرن می باشد. در این مقاله ابتدا روش های گوناگون طراحی معماری نرم افزار مورد بررسی قرار خواهد گرفت و سپس ویژگی کیفی قابلیت تغییر به طور دقیق معرفی خواهد شد. در نهایت درخصوص طراحی معماری نرم افزار با تکیه بر دستیابی به قابلیت تغییر بحث خواهد شد.

واژگان کلیدی:

سیستم های نرم افزاری

تغییر پذیری

معماری نرم افزار

اصلاحات

توسعه پذیری

[*،۱] نویسنده مسئول

a.mohammadi66@gmail.com

تعداد جداول	تعداد شکل ها	تعداد مراجع
۰	۰	۲۰

مقدمه

دنیای فناوری اطلاعات مملو از تخصص‌ها و حوزه‌هایی است که هریک از آن‌ها جایگاه خود را دارند و ترکیب این نقش‌ها با یکدیگر است که باعث می‌شود محصولات یکپارچه و دقیق آماده شود. یکی از مهم‌ترین مشاغل دنیای فناوری اطلاعات طراحی نرم‌افزار است. در ساده‌ترین تعریف طراحی نرم‌افزار فرآیند حل مسئله و برنامه‌ریزی در راستای ساخت یک نرم‌افزار است. به بیان دقیق‌تر، طراحی نرم‌افزار فرایندی است که توسط آن یک برنامه‌نویس یا تیمی از برنامه‌نویسان، مشخصه‌ای از نرم‌افزار را طراحی می‌کنند که هدف آن به انجام رساندن اهداف از پیش تعیین شده با استفاده از مجموعه‌ای از مولفه‌های اولیه با توجه به محدودیت‌ها است. امروزه یکی از مهمترین ویژگیهای هر سیستم نرم افزاری، کیفیت می باشد. با گسترش ابزارهای گوناگون برای توسعه نرم افزار، توسعه نرم افزارهایی که کارکردهای مورد نظر مشتریان را برآورده سازند، امری آسان و سریع گشته است. در حال حاضر، تفاوت بین دو نرم افزار را توانایی نرم افزارها در برآورده ساختن ویژگی های کمی و کیفی مورد انتظار تعیین می نماید [۱].

معماری نرم افزار یک برنامه یا سیستم کامپیوتری، ساختار یا ساختاره ایی از سیستم می باشد، که در برگیرنده اجزاء، صفات قابل مشاهده آن اجزا و ارتباط بین آنها باشد. معماری نرم افزار شامل اولین تصمیمات طراحی سیستم میباشد و این تصمیمات زیربنای فعالیتهای طراحی، پیاده سازی، استقرار و نگهداری سیستم میباشد. همچنین معماری نرم افزار، اولین عنصر قابل ارزیابی در فرایند توسعه یک سیستم نرم افزار میباشد. بنابراین برای طراحی سیستمی که نیازهای کیفی مورد نظر را برآورده سازد، تولید معماری نرم افزار کارامد اولین گام در دستیابی به کیفیت در نرم افزار و همچنین ارزیابی ویژگیهای کیفی است [۴-۲]. درمدل های فرایند توسعه نرم افزار مبتنی بر معماری معمولا ابتدا نیازهای کیفی سیستم تعیین شده و سپس معماری نرم افزار مربوطه طراحی می گردد. پس از طراحی معماری، میتوان به ارزیابی آن پرداخت و تغییرات لازم را در طراحی مورد نظر ایجاد داد. بنابراین دو بخش اساسی در مدل های فرایند توسعه نرم افزار مبتنی بر معماری، بخش های طراحی و ارزیابی معماری نرم افزار می باشند. این دو بخش در ارتباط مستقیم با یکدیگر می باشند و هر یک مکمل دیگری می باشد. بنابراین فرا یند طراحی معماری را میتوان شامل ساخت معماری نرم افزار، ارزیابی آن و اصلاح معماری پیشنهادی دانست [۷-۵]. در این مقاله، هدف بررسی روش های موجود در طراحی معماری بر اساس ویژگی های کیفی و بررسی نحوه دستیابی به قابلیت تغییر می باشد.

رویکردهای طراحی معماری سیستم های نرم افزاری

در مراجع مختلف مانند [۱۱-۸] معماری نرم افزار یک سیستم، ساختاره ایی از سیستم می باشد، که در برگیرنده اجزاء، صفات قابل مشاهده آن اجزا و ارتباط بین آنها باشد. درواقع معماری نرم افزار، سازمان زیربنایی سیستم میباشد، که در قالب اجزا و روابط بین آنها و همچنین روابط آنها با محیط، بیان شده است و برای طراحی و تکامل آن اصولی وجود دارد. در این نوع تعریف، فرایند تولید معماری، عضوی از معماری در نظر گرفته شده است. زیرا اصول طراحی و تکامل نیز عضوی از معماری در نظر گرفته شده اند. معماری نرم افزار شامل مجموعه ای از تصمیم گیری ها بر اساس فاکتورهای متعددی است که تمامی این تصمیم گیری ها می‌توانند بر روی قابلیت هایی از قبیل کیفیت، کارایی، نگهداری و موفقیت سراسری نرم افزار تأثیرگذار باشند. از تعاریف فوق می توان به نتایج زیر دست یافت [۱۳-۱۲]:

- معماری، اجزای نرم افزار را تعریف می نماید. همچنین در این تعریف، از جزئیاتی از اجزا، که در نحوه استفاده و ارتباط با اجزای دیگر کاربردی ندارند؛ صرف نظر می گردد.
- هر سیستم نرم افزار شامل چندین ساختار می باشد و هیچ یک از این ساختارها، به تنهایی معماری نرم افزار نمی باشد. بلکه این ساختارها در کنار یکدیگر معماری نرم افزار را تشکیل می دهند.
- اساسا هر سیستم نرم افزاری دارای یک معماری پایه می باشد. زیرا هر سیستم نرم افزاری دارای ماژول ها و اجزایی است که این اجزا با یکدیگر دارای رابطه می‌باشند. بمنظور اعمال شرایط و ویژگی های جدید در یک سیستم نرم افزاری، معماری پایه یک سیستم نرم افزاری باید پویا و قابل تغییر[1] باشد.
- رفتار هریک از اجزاء، بخشی از ویژگی های عملکردی معماری نرم افزار می باشد. زیرا این رفتار در نحوه ارتباط بین اجزاء، تعاملات و نهایتا بازخوردهای معماری تاثیرگذار است.
- معماری نرم افزار باید قابل ارزیابی و قابل تغییر باشد.

در راستای دستیابی به اهداف فوق سبک ها و الگوهای مختلفی در توسعه معماری سیستم های نرم افزاری بکارگرفته شده اند. الگوهای معماری یا سبک های معماری شامل شرحی از اجزاء و نوع روابط بین آنها می باشد به نحوی که تعدادی قانون برای معرفی اجزاء و

[1] Alterable

نحوه ارتباط بین آنها، مشخص گردد. پرکاربرد رویکردهای طراحی معماری سیستم های نرم افزاری عبارتند از [۱۷-۱۴]:

الگوی لایه‌ای: معماری لایه‌ای رویکردی سلسله مراتبی دارد. این الگو برای ساخت برنامه‌هایی استفاده می‌شود که در آن‌ها یک برنامه را می‌توان در گروه‌هایی از زیروظایف تقسیم کرد که هر کدام از این وظایف در لایه‌ها و سطوح خاص انتزاعی قرار می‌گیرند. هر لایه خدماتی را برای لایه‌های بالاتر از خود ارائه می‌کند. در هر لایه وظایف خاصی انجام می‌شود و هر چه لایه‌ها به سمت بالاتر حرکت می‌کنند، این الگو به سمت کدهای زبان ماشین متمایل می‌شود.

الگوی کلاینت ـ سرور: این الگوی معماری از دو بخش یک سرور و چند کلاینت تشکیل‌شده است. مولفه سرور سرویس‌هایی را برای چند مولفه کلاینت ارائه می‌کند. کلاینت‌ها به‌منظور دسترسی به سرویس موردنیاز خود، درخواستی برای سرور ارسال کرده، سرور درخواست را دریافت و پردازش کرده و در ادامه سرویس مدنظر کلاینت را ارائه می‌کند. علاوه بر پاسخ‌گویی به درخواست کلاینت‌ها، سرور سیستم به‌طور پیوسته آماده است تا درخواست‌ها را از کلاینت‌ها دریافت کرده و سرویس‌دهی کند.

الگوی Pipe-filter : این الگو بیشتر در ارتباط با سامانه‌ها و برنامه‌های ساخت‌یافته که وظیفه آن‌ها تولید و پردازش داده‌های جریانی است، کاربرد دارد. به عبارت دقیق‌تر، از این الگو در موازی‌سازی و ساخت خط لوله پردازشی استفاده می‌شود. این الگو بر مبنای فیلترهای متعددی که درون یک برنامه قرار می‌گیرند و به شکل هم‌زمان با یکدیگر در حال اجرا بوده و از طریق کانال‌هایی با یکدیگر در ارتباط هستند کار می‌کند. در این سامانه‌ها/برنامه‌ها در هر مرحله از پردازش درون یک مولفه فیلتر انجام می‌شود. داده‌هایی که باید پردازشی روی آن‌ها انجام شود، از طریق مسیرهایی از یک مولفه به مولفه دیگر می‌رسند. این مسیرها می‌توانند به‌منظور بافر کردن یا هماهنگ‌کردن سایر مولفه‌ها با یکدیگر استفاده شوند.

الگوی کارگزار: این الگو برای ساخت سامانه‌های توزیع‌شده با مولفه‌های جدا از هم کاربرد دارد. این مولفه‌ها می‌توانند از طریق سرویس‌های راه دور با یکدیگر ارتباط برقرار کنند. یک مولفه کارگزار وظیفه هماهنگ کردن ارتباط میان سایر مولفه‌ها را عهده‌دار است. در این الگوی معماری سرورها می‌توانند قابلیت‌ها، توانایی‌ها، سرویس‌ها و مشخصات را به کارگزار واگذار کنند.

الگوی نظیربه‌نظیر: در این الگوی معماری، هر یک از مولفه‌ها به‌عنوان یک نظیر/همتا در نظر گرفته‌شده و هر یک از آن‌ها ممکن است در قالب یک کلاینت رفتار کرده و سرویسی را از سایر همتایان درخواست کند. درعین‌حال یک همتا ممکن است نقش یک سرور را بازی کرده و به همتایان خود سرویسی را ارائه کند. همتا می تواند به شکل پویا و در مدت‌زمانی که ایفاگر یک نقش است، تغییر وضعیت دهد [۱۸].

الگوی معماری Event-Bus : این الگوی بر مبنای رخدادها بوده و از چهار مولفه اصلی منبع رویداد، شنوندگان رویداد، کانال و event bus تشکیل‌شده است. مولفه منبع پیام‌ها را برای یکسری کانال‌های خاص از طریق event bus منتشر می‌کند [۱۹].

الگوی تخته‌سیاه: این الگو برای حل مشکلاتی استفاده می‌شود که هیچ‌گونه استراتژی مستقیمی برای حل آن‌ها وجود ندارد. از این الگوی معماری عمدتا در ارتباط با ادغام‌سازی بخش‌های تخصصی و متنوعی استفاده می‌شود که نمی‌توان یک راهکار قطعی برای ادغام و توسعه آن‌ها پیدا کرد [۲۰].

دستیابی به قابلیت تغییر در طراحی معماری

معماران نرم افزار به طور معمول قابلیت تغییر را در معماری و اجزاء نرم افزار برنامه ریزی می کنند و مکانیسم هایی در ساختار آن جهت پشتیبانی از این تغییرات قرار می دهند. درک موقعیت هایی که تغییر برای آنها برنامه ریزی شده است و ثبت گزینه های ممکن در داخل شرایط خاص معمولا به صورت صریح و روشن انجام نمی شود. این در شرایطی مهم می شود که یک ساختار برای نسخه های متعدد محصول برای مدت طولانی در خط تولید محصول بکار رود و آن ساختار برای ایجاد تغییر در انواع محصولات مختلف مورد استفاده قرار گیرد. معرفی تغییرات به طور واضح و نشان دادن جاهایی از ساختار که در آنها تغییر مجاز است، اهمیت دارد. در این قسمت شرح خواهیم داد که چگونه فرایند مدیریت تغییرات در یک ساختار معماری می تواند به صورت صریح تر با انجام مجموعه ای از گامهای منطقی انجام شود و چگونه استفاده از تغییر پذیری در ارتباط با انتخاب مشخصه های محصول می تواند باعث هدایت به سمت مکان های مناسب در ساختار معماری سیستم گردد. در این راستا، مهم ترین گامهای پیشنهادی در دستیابی به قابلیت تغییر در معماری عبارتند از:

- شناسائی و ایجاد موارد کاری مشخص برای تغییرات موردنیاز در معماری و سیستم
- لاگ کردن، دسته بندی و اولویت بندی تغییرات
- برنامه ریزی و زمانبندی برای تولید و انتشار تغییرات
- امکان سنجی تغییرات، کنترل و مدیریت پیکربندی
- ایجاد یا انتخاب رویکرد معماری مناسب و تدوین چارچوب جامع در جهت اعمال تغییرات
- ایجاد پایگاه داده جامع برای ثبت و نگهداری تغییرات

- بررسی تعاملات تغییرات با با اجزاء قبلی معماری
- بررسی پیکربندی، تحلیل یا ارزیابی معماری تغییریافته
- پیاده سازی تغییرات براساس شاخص های عملکردی و کیفی در معماری های قبلی و تغییریافته
- مقایسه شاخص های کمی و کیفی در معماری های قدیم و جدید سیستم نرم افزاری
- گزارش وضعیت تغییرات در معماری سیستم و اطمینان از انطباق مشخصه های عملکردی در محصول
- ایجاد خط مبنا در براساس بروندادهای فرایند مدیریت تغییر در پروژه

لازم به توضیح است، اساسا چرخه عمر توسعه سیستم نرم افزاری نقطه شروعی برای دستیابی به قابلیت تغییر در معماری نرم افزار است . تغییرات از تصمیمات اتخاذ شده در طول هرمرحله از چرخه عمر توسعه سیستم پیروی می کنند. این تغییرات ممکن است از آغاز کار شناخته شوند، یا می توانند در ابتدا بدون ساختار و مبهم باشد. اگر تغییرات مبهم باشند ، اولین اقدام ضروری درگیر شدن همه بخش های مرتبط در یک فرایند ایجاد شناخت درباره شرایط نامطمئن و تبیین تغییرات پیش از مدیریت نمودن آنهاست . به علاوه در مراحل اولیه فرایند تغییر باید روش هایی که می توانند برای حذف مقاومت در مقابل تغییرات به کار روند ، مورد توجه قرارداده شوند . در گام شناسائی هر شی داده ویژگی‌های مشخصی دارد که این اشیاء و ویژگی ها در طرح شناسایی اشیاء نرم‌افزار در طول فرایند نرم‌افزاری به وجود می‌آیند و ممکن است چندین بار تغییر کنند، می‌توان برای هر شی یک نمودار تکاملی ایجاد نمود. کنترل پیکربندی روش‌ها و ابزارها را ترکیب می‌نماید تا نسخه‌های گوناگونی از اشیاء پیکربندی را که طی فرایند توسعه ایجاد شده‌اند را مهار نماید. عموما در پروژه‌های مهندسی نرم‌افزار بزرگ تغییرات کنترل نشده به سرعت سبب به وجود آمدن اختلال و آشوب می‌شوند. اساسا در فعالیت کنترل تغییرات پروژه، رکنی بنام کنترل دست یابی، تعیین می‌کند کدامیک از مهندسین نرم‌افزار اختیار ارزیابی دارد و اصلاح یک شی پیکربندی خاص را انجام می‌دهد. همچنین رکن دیگری بنام کنترل همزمانی، ما را مطمئن می‌سازد تغییرات موازی که توسط دو فرد مجزا انجام می‌شوند نتایج یکدیگر را از بین نمی‌برند. مدل های مناسب اعمال تغییرات دیدگاه های مختلفی هستند که بخش ها نسبت به سیستم پیشرفته یا جدید دارند . تجزیه و تحلیل سیستم جدید به نحوی که توسط بازیگران نقش های مختلف درک می‌شود می تواند تعیین نماید که چگونه معماری استفاده می شود، چگونه بر کاری که کاربران مسئول اجرای آن هستند اثر دارد و چگونه بر ویژگی های سازمانی تاثیر می‌گذارد. ارزیابی و مقایسه بین معماری قدیم و جدید ، برآیند و بازخورد تغییراتی را که ناشی از سیستم اطلاعاتی معماری جدید هستند را نشان دهد و بر اساس آن می توان تاثرات تغییرات بر معماری و قابلیت های عملکردی نرم افزار را مدیریت کرد .

بررسی پیکر بندی اطمینان ایجاد می کند که کیفیت مدنظر در ضمن انجام تغییرات به دست می‌آید. گزارش وضعیت تغییرات در معماری اطلاعاتی را در مورد تغییر فراهم می‌نماید. اساسا در این فرایند مجموعه ای از اشیاء مربوط به هم و هر محصولی که درطول پروژه تهیه میشود اقلام پیکربندی نرم افزار نامیده می شوند مانند مجموعه ای از مستندات، نرم افزارها یا سخت افزارها که توسط خط مبنا تثبیت می شوند. خط مبنا تصویری از معماری تغییر یافته است است که بعد از اتمام فرایند مدیریت تغییرات از پروژه تهیه می‌شود تا بعد از به روزرسانی های جدید پروژه، خط مبنای جدید با خط مبنای اولیه مقایسه شود و همواره ابزاری برای ارزیابی معماری نرم افزار وجود داشته باشد. در مواقعی که پروژه نرم افزاری از چند فاز طولانی تشکیل شده است، استفاده از خطوط مبناهای مختلف می‌تواند شما را در تجزیه و تحلیل پروژه کمک کند. شما با ساختن خطوط مبنای متعدد برای دروه‌های مختلف پروژه می‌توانید تغییرات دوره‌های مختلف را نسبت به یکدیگر بررسی کنید.

مزایای دستیابی به قابلیت تغییر در معماری نرم فزار

اگرچه هیچ چارچوب یا متدولوژی نمی‌تواند موفقیت ۱۰۰٪ را تضمین کند، مدیریت موثر تغییرات در معماری نرم افزار می‌تواند به مدیریت ریسک و حفاظت از خدمات مبتنی بر معماری که ارائه می‌دهید، کمک کند. حفظ و تکامل سیستم‌های نرم افزاری برای بقای هر سازمانی در فضای رقابتی بازار امروز ضروری است. تنظیمات هر عنصر در زیرساخت فناوری اطلاعات می‌تواند ارزش خدمات را مختل کرده و بر بهره‌وری تأثیر منفی بگذارد. تغییر ساختاریافته و برنامه ریزی شده به حداقل رساندن خطر احتمالی ناشی از تغییرات زیرساختی کمک می‌کند. در عین حال، فرایند مدیریت تغییر و برنامه‌ریزی مدون، دارای مزایای قابل توجهی در پروژه های نرم افزاری است. برخی از مزایای ناشی از مدیریت موثر تغییرات عبارتند از:

- بهبود کیفیت خدمات سیستم نرم افزاری
- کاهش تأثیر سوء بر عملکردهای بروزرسانی شده نرم افزار
- بهبود دید در تغییر ماژول ها و اجزاء نرم افزار
- اولویت پاسخگویی در اعمال تغییرات

- پایبندی به معماری و سایر مقررات انطباق در محصول
- مدیریت ریسک بهبودیافته
- کاهش اختلالات خدمات و خرابی سیستم
- افزایش بهره‌وری بخش های نرم افزار و کاهش هزینه های جانبی توسعه نرم افزار
- اجرای سریع‌تر و موثرتر تغییرات در محصول نرم افزاری

نهایتا لازم به ذکر است، طراحی یک روند مدیریت تغییر استاندارد که توسط طراح معماری نرم افزار تصویب شود، به مدیریت سریع، اقتصادی و موثر تغییرات در هنگام وقوع آن‌ها کمک می‌کند. سپس این فرآیند می‌تواند توسط نرم‌افزار پشتیبانی مدیریت خدمات به صورت خودکار انجام و اجرا شود. کنترل تغییر یک عنصر تابع فرایند مدیریت تغییر کلی است که برای اطمینان از کنترل، ثبت، تحلیل و تأیید تغییرات طراحی شده است. هرچند توصیه‌های کلی ارائه شده می‌تواند تا حد زیادی سیستم های نرم افزاری را در دستیابی و مدیریت موثر تغییرات درست راهنمایی کند اما بیشترین تفاوت بین محصولات نرم افزاری در مشخصه ای قابل انتظار از سوی مشتریان است. بنابراین در هر سیستم نرم افزاری با اتکا به تحلیل ریشه‌ای نیازمندی ها و تغییرات موردنیاز در پروژه می توان راهکارهای متناسبی برای دستیابی به قابلیت تغییر در معماری منتخب آن سیستم ارائه نمود.

نتیجه گیری

حوزه طراحی معماری نرم افزار از مهمترین حوزه ها در علم مهندسی نرم افزار است. معماری یک سیستم نرم افزاری تقریبا هیچ گاه محدود به یک سبک معماری تک محور و تک بعدی نیست. بلکه اغلب ترکیبی از سبک های معماری مختلف است که تمامی مشخصه ای سیستم را تشکیل می دهند و قابلیت تغییر یک مولفه اساسی آن می باشد. با وجود پیشرفت های چشم گیر، مهندسی تغییرات معماری نرم افزار جزء ضعیف ترین حلقه ها در زنجیره فعالیت های مهندسی نرم افزار است. امروزه تحلیل نیازمندی‌های جدید و دستیابی به قابلیت تغییر در معماری نرم افزار از ملزومات موفقیت در یک پروژه نرم افزاری محسوب می‌گردد. دستیابی به قابلیت تغییر در مشخصه های عملیاتی از مهم ترین ویژگی های کیفی در طراحی معماری سیستم های نرم افزاری مدرن می باشد. در این مقاله سعی شد مهم ترین مفاهیم و گامها درخصوص دستیابی به قابلیت تغییر در طراحی سیستم های نرم افزاری را تحلیل، بررسی و تبیین گردد.

منابع

[1] Ahmad, M.O., Markkula, J., Oivo, M.: Kanban in software development: a systematic literature review. In: 2013 39th Euromicro Conference on Software Engineering and Advanced Applications, pp. 9–16. IEEE (2013)

[2] Angelov, S., & de Beer, P. (2017). Designing and applying an approach to software architecting in agile projects in education. Journal of Systems and Software, 127, 78–90.

[3] Babar, M., Brown, A., & Mistrik, I. (2013). Making software architecture and agile approaches work together: Foundations and approaches. Agile software architecture: Aligning agile processes and software architectures.

[4] Kruchten, P. B. (1995). The 4 + 1 view model of architecture. IEEE Software, 12(6), 42–50.

[5] Abrahamsson, P., Babar, M. A., & Kruchten, P. (2010). Agility and architecture: Can they coexist? IEEE Software, 27(2).

[6] Buschmann, F., Henney, K., & Schmidt, D. C. (2007). Pattern-oriented software architecture, on patterns and pattern languages (Vol. 5). Wiley.

[7] Chauhan, M. A., Babar, M. A., & Benatallah, B. (2016). Architecting cloud-enabled systems: A systematic survey of challenges and solutions. Software: Practice and Experience.

[8] Matinlassi, M., Niemelä, E., & Dobrica, L. (2002). Quality-driven architecture design and quality analysis method. A revolutionary initiation approach to a product line architecture. Espoo: VTT Technical Research Centre of Finland.

[9] Kazman, R., Bass, L., Webb, M., & Abowd, G. (1994). SAAM: A method for analyzing the properties of software architectures. In Proceedings of the 16th International Conference on Software Engineering (pp. 81–90). IEEE Computer Society Press.

[10] Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., & Little, R. (2002). Documenting software architectures: Views and beyond. Pearson Education.

[11] Coplien, J. O. & Bjørnvig, G. (2011). Lean architecture: For agile software development. Wiley.

[12] Chauhan, M. A., Babar, M. A., & Probst, C. W. (2016). A process framework for designing software reference architectures for providing tools as a service. In Product-

Focused Software Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, 22–24 November 2016, Proceedings 17 (pp. 111–126). Springer.

[13] Gorton, I. (2006). Essential software architecture. Springer Science & Business Media.

[14] Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., & Carriere, J. (1998). The architecture tradeoff analysis method. In Fourth IEEE International Conference on Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings (pp. 68–78). IEEE.

[15] Kiv, S., Heng, S., Wautelet, Y., Poelmans, S., Kolp, M.: Using an ontology for systematic practice adoption in agile methods: expert system and practitioners-based validation. Expert Syst. Appl. 195, 116520 (2022)

[16] Wirsing, M., et al.: Software Engineering for Collective Autonomic Systems, 537 p. Springer, Heidelberg (2015)

[17] Silva, E., Batista, T., Oquendo, F.: A mission-oriented approach for designing system-of-systems. In: Proceedings of the 10th IEEE System-of-Systems Engineering Conference (SoSE), pp. 346–351, May 2015

[18] SAE Standard AS5506-2012: Architecture Analysis & Design Language (AADL), 398 p., September 2012

[19] Quilbeuf, J., Cavalcante, E., Traonouez, L.-M., Oquendo, F., Batista, T., Legay, A.: A logic for the statistical model checking of dynamic software architectures. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 806–820. Springer, Heidelberg (2016).

[20] Oquendo, F., et al.: Proceedings of the 1st ACM International Workshop on Software Engineering for Systems-of-Systems (SESoS), Montpellier, France, July 2013