عنوان مقاله: مدرن سازی معماری نرم افزار

نویسنده مقاله: علیرضا ارومند

تاریخ انتشار: ۲۴ اردیبهشت ۱۴۰۲

منبع: https://nikamooz.com/modernization-of-software-architecture

مدرن سازی معماری نرم افزار برای معماری‌های قدیمی و بیات، که ریسک تجاری و نقطه ضعفی برای مزایای رقابتی کسب‌وکارها محسوب می‌شوند، امری ضروری به‌حساب می‌آید؛ چراکه تغییر در آن‌ها بسیار سخت و کند بوده و احتمال این وجود دارد که در شرایطی، عدم اطمینان به کسب‌وکار ایجاد شود. در این صورت، فرصتی در اختیار رقبا قرار می‌گیرد تا مزایای زیادی نسبت به کسب‌وکار پیدا کرده و ممکن است که آن، آسیب ببیند. خطوط هوایی Southwest این مشکل را با گوشت و پوست خود لمس کرد.

در سال ۲۰۲۲، یک سیستمِ برنامه‌ریزیِ قدیمی، بحرانی بزرگ ایجاد کرد. در یک هفته، ۱۴۵۰۰ پرواز لغو شد و برند آن‌ها، آسیب‌های غیرقابل جبرانی دید. آن‌ها به‌دلایلی اشتباه، تیتر یک اخبار بین‌المللی شدند.

در مقابل، مدرن سازی معماری نرم افزار و طراحی و پیاده‌سازی یک معماری مدرن و دقیق، مزیت رقابتی بزرگی برای کسب‌وکار ایجاد می‌کند. در انگلیس، استارت‌آپ Cazoo تنها در ۹۰ روز کسب‌وکاری آنلاین در حوزه خودرو راه‌اندازی کرد و سریع‌ترین یونیکورن بریتانیا شد. رسیدن به ارزش یک میلیارد دلاری در کمتر از ۱۸ ماه، چیزی است که بسیاری از کسب‌وکارها در خواب هم نمی‌بینند. اما مسئله اصلی اینجاست که چطور در این زمان کوتاه، به این ارزش رسیده‌اند؟ یکی از عوامل کلیدی موفقیت Cazoo، توانایی آن‌ها در بهره‌گیری از شیوه‌ها و فناوری‌های معماری مدرن، مانند Serverless بود. استفاده از مدرن سازی معماری نرم افزار به آن‌ها امکان ساخت محصولی را داد تا در شرایطی که درحال ارائه خدمات به استفاده‌کننده‌ها بود، بتواند تغییر مقیاس داده یا نسخه جدید برای آن، نصب و راه‌اندازی کند.

شاید به نظر برسد با افزایش سن، شرکت‌ها و کسب‌وکارشان از یک استارت‌آپ چابک به شرکتی قدیمی، سنگین و سخت با معماری قدیمی تبدیل می‌شوند و این تغییر شرایط، اجتناب‌ناپذیر است؛ اما Netflix ثابت کرد که تغییر این شرایط امکان‌پذیر است. آن‌ها در سال ۲۰۰۹ تغییرات عمده‌ای در کار خود ایجاد کردند. آن‌ها نرم‌افزار خود را از یک نرم‌افزار با معماری Monolith به چندین نرم‌افزار با معماری Microservice مهاجرت دادند تا قابلیت‌های رقابتی خود را در بازار Streaming حفظ کنند. آقای Adrian Cockroft که آن زمان به‌عنوان CTO در شرکت Netflix مشغول به کار بود، دلیل خود برای این تغییر و مدرن سازی معماری نرم افزار را اینگونه بیان کرد:

«تهدیدی در ماهیت کسب‌وکارها وجود دارد. اگر شما نسخه‌های سه ماهه را انجام می‌دهید و رقیب شما درحال انتشار روزانه و تحویل مداوم است، از تجربه کاربری بسیار عقب خواهید افتاد و فقط از آن رنج خواهید برد.»

به عقیده‌ی من، هر رهبری باید دائماً از خود سؤالاتی بپرسد که در آن زمان، در Netflix نیز پرسیده شد:

- آیا ممکن است از رقبا عقب بیافتیم؟
- آیا ممکن است یک استارت‌آپ، خیلی سریع وارد شود و کسب‌وکار ما را دگرگون کند؟
- آیا بخش‌های حیاتی کسب‌وکار ما، درحال استفاده از سیستم‌های تاریخ مصرف‌دار هستند که هر لحظه ممکن است باعث از دست دادن جدی درآمد یا آسیب به شهرت ما شود؟

سازمان‌های زیادی مانند نتفلیکس وجود دارند که معماری و شیوه‌های خود را با موفقیت، مدرن‌سازی کرده و آن را از یک بدهی بزرگ به مزیتی رقابتی تبدیل کرده‌اند. می‌خواهیم طی چندین مطلب، بر مدرن سازی معماری نرم افزار برای دستیابی به مدل‌های عملیاتیِ مدرن و باکیفیت، متمرکز شویم. می‌خواهیم تیم‌هایِ محصولِ توانمندی داشته باشیم که جریان تغییراتِ سریعی در محصولات خود دارند و محصولاتی باکیفیت که درحال پیشرفت هستند را روزانه در محیط عملیاتی و مقیاس بالا، روانه بازار می‌کنند. در این مطلب، جنبه‌های کلیدی در مسیر مدرن سازی معماری نرم افزار و نحوه‌ی تطبیق آن‌ها با یکدیگر را بررسی می‌کنیم؛ سپس در قسمت‌های آینده، به موضوعات مختلف به‌صورت اختصاصی‌تر خواهیم پرداخت.

معماری مدرن مبتنی بر جنبه های فنی و اجتماعی

یک معماری اگر به‌خوبی طراحی شده باشد، پیوستگی خوبی داشته و با جدیدترین فناوری‌ها ساخته شده باشد، بازهم به‌تنهایی، توانایی نوآوری در کسب‌وکار با سرعت بالا را تضمین نمی‌کند. در کنار معماریِ نرم‌افزارِ خوب، سازماندهی مؤثر تیم‌ها در محیطی که به آن‌ها قدرت نوآوری دهد نیز ضروری است. هر دو جنبه فنی و سازمانی معماری باید باهم هم‌خوانی داشته باشند تا راهکاری که ارائه می‌کنیم، بهینه باشد. این معماری، که همزمان به جنبه‌های فنی و اجتماعی توجه می‌کند، در اصلاح به معماریِ اجتماعی – تکنیکال (Socio-Technical Architecture) معروف شده است و برخی از معماران، خود را معماران فنی – اجتماعی می‌نامند.

شاید مدرن‌سازی Netflix ظاهراً یک تلاش کاملاً فنی به‌نظر برسد. به هر حال، آن‌ها با مدرن سازی معماری نرم افزار و با شکستن نرم‌افزار خود از یک معماری یکپارچه به تعداد زیادی نرم‌افزار کوچک با معماری میکروسرویس، یک الگوی معماری جدید را به کار بردند. اما اگر به دقت نگاه کنید، درمی‌یابید که میکروسرویس‌ها، یک الگوی معماریِ اجتماعی – فنی هستند .مزایای این الگوی معماری، به همان اندازه که فنی بوده، سازمانی نیز است. Sam Newman، نویسنده‌ی Building Microservices (O’Reilly)، این مورد را به‌خوبی توضیح می‌دهد.

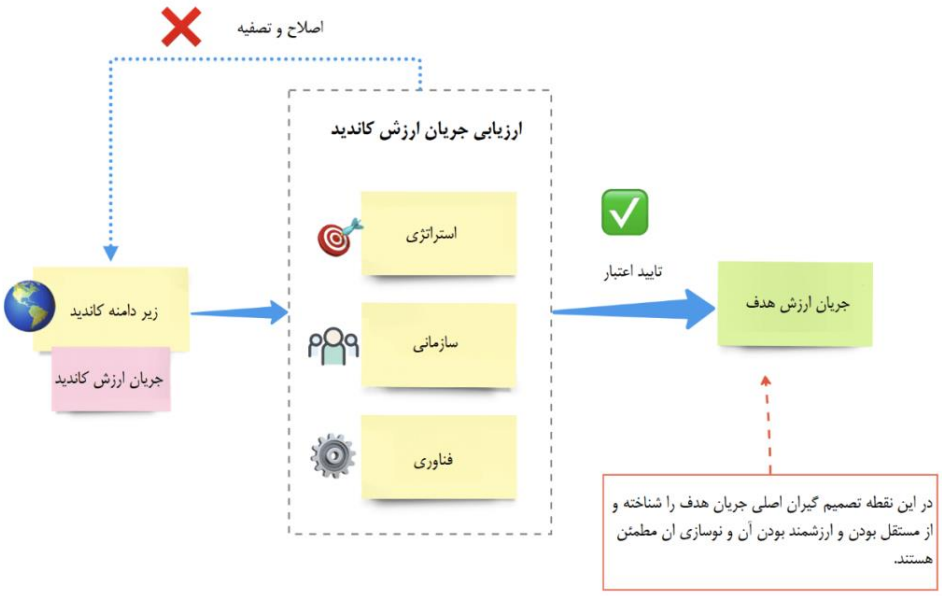
دلیل سوم (برای پذیرش میکروسرویس‌ها) و مدرن سازی معماری نرم افزار، واقعاً جایی است که شما به‌دنبال ایجاد درجه‌ی بالاتری از استقلال سازمانی هستید. شما به‌دنبال توزیع مسئولیت بین تیم‌ها هستید؛ می‌خواهید آن‌ها بتوانند تصمیم‌گیری کنند، نرم‌افزار را توسعه دهند و میزان هماهنگی موردنیاز تیم‌ها با سایر بخش‌های سازمان شما را کاهش دهند. میکروسرویس‌ها تنها سبکِ معتبرِ معماری برای ما نیستند؛ صرف‌نظر از الگوها و فناوری‌هایی که استفاده می‌کنیم، یک رویکرد و معماری اجتماعی – فنی برای دستیابی کامل به پتانسیلِ سرعت رسیدن به بازار (speed-to-market) برای سازمان‌ها ضروری است. همان‌طور که Sam Newman تأکید کرد، تیم‌ها برای تغییر و استقرار کدِ خود، بدون وابستگی و نیاز به هماهنگی با دیگران، نیاز به استقلال دارند .این جریان در سال‌های اخیر، به‌ویژه با انتشار کتاب (IT Revolution) Team Topologies در سال ۲۰۱۹، بسیار موردتوجه قرار گرفته، به یکی از مفاهیم داغ برای گفتگو در محافل مختلف بدل شده و توسط بسیاری از سازمان‌ها، از Docker گرفته تا دولت نروژ، پذیرفته شده است.

توپولوژیِ تیم‌ها، ابزاری برای سازمان‌هایی است که می‌خواهند بتوانند تغییرات سریع و مکرر را در برنامه‌های خود ایجاد و جریان سریعی از ارزش‌ها را در کسب‌وکار خود اعمال کنند. در عمل، این جریان سریع، اغلب به این معنی است که تیم‌ها کد را به‌صورت روزانه یا چندبار در طول روز، در محیط عملیاتی استقرار می‌دهند. توپولوژی تیم‌ها عمدتاً بر جنبه‌های اجتماعیِ معماریِ اجتماعی – فنی تمرکز دارد که از آن به‌عنوان تفکر اول تیم (Team-First Thinking) و مرزهای اول تیم (Team-First Boundaries) یاد می‌شود. معماری نرم‌افزار نیز باید توسط الزامات سازمانی برای دستیابی به جریان سریع، به‌سمت این دیدگاه هدایت شود.

یکی از مفاهیم اساسیِ معماری در توپولوژی‌های تیمی، جریان ارزش (Value Stream) است. جریانِ ارزش، دنباله‌ای از مراحل است که تیم برای کشف نیازهای برآورده نشده کاربران، طراحی راه‌حل‌ها برای آن نیازها، پیاده‌سازی آن راه حل‌ها در قالب نرم‌افزار و ارائه آن ارزش به مشتریان، طی می‌کند. تیمی که مسئول یک جریان ارزش است، تیم همسو با جریان (Stream-aligned Team) نامیده می‌شود. شکل زیر، نمای کلی و سطح بالا از یک جریان ارزش را ارائه می‌دهد:

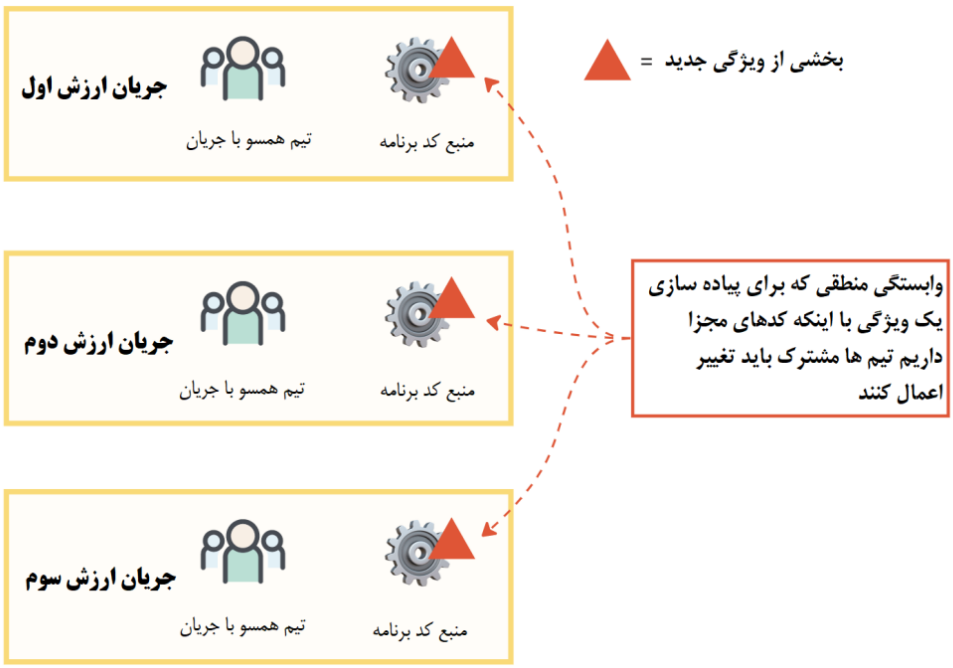
دستیابی به جریانی سریع، نیازمند این است که معماری و مدرن سازی معماری نرم افزار به گونه‌ای باشد که جریان‌های ارزش، از هم مستقل باشند. این بدان معناست که تیم، در تصمیم‌گیری درمورد جریان ارزش خود، بیشترین اختیار را دارد. مثلاً تیم می‌تواند تصمیم بگیرد چه چیزی بسازد، چگونه آن را ساخته و سپس به کار گیرد. این نکته اشاره به این مسئله دارد که شما آن را می‌سازید و شما آن را اجرا می‌کنید. (You Build It, You Run It) برای اینکه این مدل به‌درستی کار کند، تیم‌ها باید مالک نرم‌افزار برای جریان ارزش خود باشند، برروی انتشار نرم‌افزار خود نیز باید کنترل داشته و برای این کار، به سایرین وابسته نباشند. در این روش، به جای اینکه تیم‌ها را با ویژگی‌ها و الزامات مربوط به پیاده‌سازی بسنجیم، آن‌ها را براساس نتایج تجاری، مانند OKRها سنجش خواهیم کرد. یک ضدالگوی معروف در این کار، که بعضاً تیم‌ها به‌سمت آن میل پیدا می‌کنند، Feature Factory است.

حال که دریافتیم باید جریان‌های ارزشِ مستقلی داشته باشیم که مالک آن‌ها، تیم‌های مستقل است، مهم است که بتوانیم این جریان‌های ارزش را شناسایی کنیم. تصویر بعد، نحوه شناسایی جریان‌های ارزش مستقل را نشان می‌دهد. ابتدا کار خود را با شناسایی زیردامنه‌های تجاری (Business Subdomains، معروف به زیردامنه‌های کسب‌وکار، مناطقی از کسب‌وکار که به اندازه کافی کوچک هستند تا یک تیم واحد داشته باشد.) که قرار است در آن، سازمان جریان ارزش ایجاد کند، شروع می‌کنیم. مواردی که شناسایی کردیم را جریان‌های ارزش کاندید (Candidate Value Streams) می‌نامیم. سپس جریان‌های ارزش کاندید از سه دیدگاه مختلف یعنی استراتژیک، سازمانی و فناوری ارزیابی می‌شوند. اگر این دیدگاه‌ها با موفقیت تأیید شوند، جریان ارزش به‌عنوان یک جریان ارزش هدف درنظر گرفته می‌شود؛ به این معنی که اطمینان کافی وجود دارد که یک جریان ارزش مستقل خواهد بود و نوسازی آن می‌تواند آغاز شود. در طول این فرآیند، احتمالاً اصلاحات زیادی وجود دارد. اغلب، مسائل و مشکلات عمیقی ممکن است ظاهر شوند. مسائلی مانند عدم همسوییِ استراتژیک بین سهامداران یا مسائل و مدل‌های تأمین مالی. این مسائل باید قبل از ایجاد تغییرات ساختاری موردتوجه قرار گیرند. این کار، یک فرآیند یا چک‌لیست ساده نخواهد بود که خیلی راحت و بدون دردسر از بالا تا پایین تیک بزنیم و به پایان آن برسیم.



معماری، همسو با دامنه تجاری

یک راه مدرن سازی معماری نرم افزار، معماری همسو با دامنه تجاری است. ممکن است در سطح کد، معماری نرم‌افزار جدا شده باشد، اما به‌دلیل وابستگی در ایجاد تغییرات، وابستگی‌هایی بین جریان‌های ارزش وجود داشته باشد. این اتفاق در شرایطی رخ می‌دهد که دو بخش از سیستم مجبور باشند به‌طور هم‌زمان تغییر کنند، در حالیکه بخشی از یک منبع کد نیستند. این نوع وابستگی‌ها را وابستگی‌های منطقی می‌نامیم. همان‌طور که در تصویر بعد مشاهده می‌کنید، وابستگی‌های منطقی، مشکل‌ساز هستند؛ زیرا استقلال یک جریان ارزش را کاهش می‌دهند. در وابستگی منطقی، تیم‌های مختلف که مسئول جریان‌های ارزش متفاوت هستند، باید فرآیند کار و استقرار خود باهم هماهنگ کنند. این نیاز به هماهنگی به‌طور بالقوه، حتی منجر به گلوگاه‌هایی می‌شود که در آن، یک یا چند تیم برای انجام کارهای خود به‌خاطر تیمی دیگر متوقف می‌شوند.



برای به حداقل رساندن تغییراتِ وابسته، مهم است که دامنه کسب‌وکار را ترسیم کرده و به دقت تجزیه و تحلیل کنید تا دریابید با مدرن سازی معماری نرم افزار، وقتی ویژگی‌های جدیدی به سیستم اضافه می‌شود، کدام مفاهیم دامنه باهم تغییر می‌کنند. مفاهیمی که باهم تغییر می‌کنند را می‌توان در زیردامنه‌های یکسان سازماندهی کرد و جریان‌های ارزشی را می‌توان برای هر زیردامنه ایجاد کرد که حاوی یک منبع کد هماهنگ شده با زیردامنه باشد. در نتیجه، جریان‌های ارزش، مستقل خواهند بود و تیم‌ها می‌توانند هر مرحله از جریان ارزش خود را، از کشف تا استقرار، بدون نیاز به هماهنگی با سایر تیم‌ها طی کنند.

یکی از مؤثرترین راه‌ها برای ترسیمِ دامنه‌ی تجاری و شناساییِ زیردامنه‌های آن، تکنیکی به نام Event Storming است. Event Storming یک تکنیک است که در آن، افرادی که درگیر ساختِ یک محصول هستند، از گروه‌های متنوعی مانند مهندسان نرم‌افزار، مدیران محصول، طراحان UX، متخصصان QA و ...گرد هم می‌آیند و به‌طور گروهی، نقشه‌برداریِ بخشی از کسب‌وکار را با استفاده از رویدادهای دامنه (Domain Event) در طول یک بازه زمانی انجام می‌دهند. پس از اینکه گروه، بازه زمانی را با رویدادهای دامنه ترسیم کرد، رویدادها را می‌توان به زیردامنه‌ها تقسیم کرد.

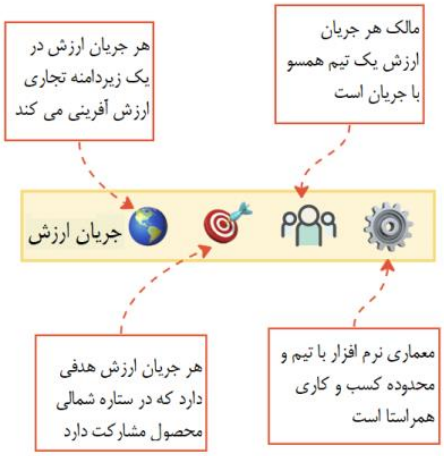
در مدرن سازی معماری نرم افزار، باید اذعان کنیم که هرگز نمی‌توان تغییرات وابسته را کاملاً حذف کرد. برخی از پیوندهای منطقی بین جریان‌های ارزش، همیشه وجود خواهند داشت. سرمایه‌گذاری در زیردامنه‌هایی که به خوبی طراحی شده‌اند، کلیدی برای جلوگیری از وابستگی غیرضروری در تغییرات است. باید دقت کنیم که اگر هنگام تشخیص زیردامنه‌ها، کم‌دقتی کنیم، ممکن است به‌راحتی مرزهای دامنه‌ای ایجاد شود که به‌خوبی تعریف نشده‌اند.

مدل های عملیاتیِ محصول محور

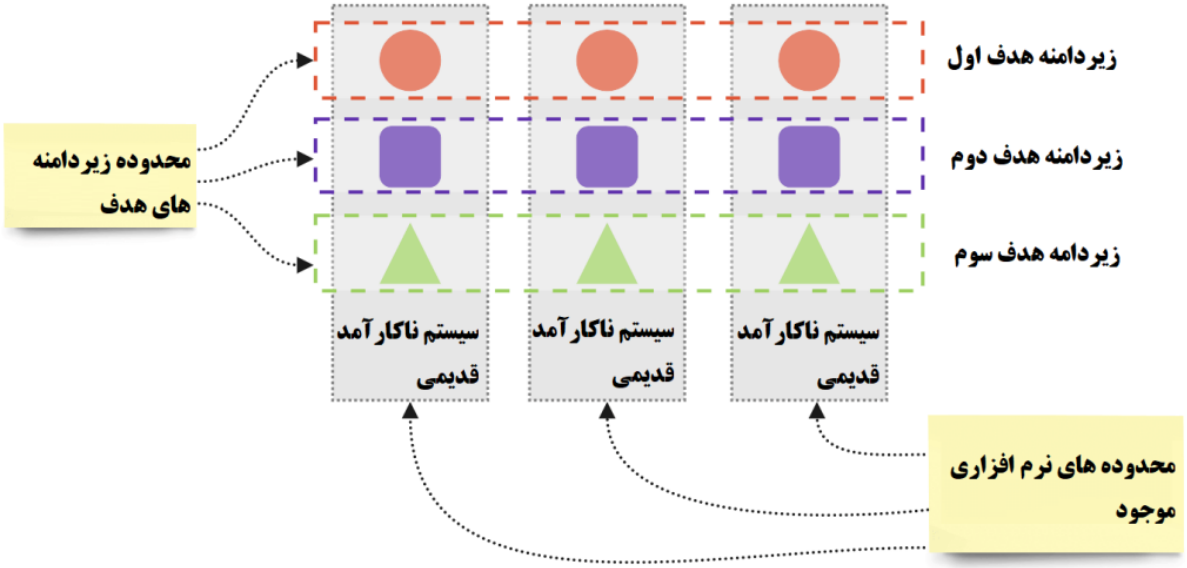
مدرن سازی معماری نرم افزار برای توانمندسازی مدل‌های عملیاتی مدرن ضروری است. در گذشته، نرم‌افزار به‌صورت پروژه‌محور ساخته می‌شد. در آن روش، تیم، محدوده‌ای از پیش تعیین‌شده در بازه زمانی معلوم شده از قبل و در چارچوب بودجه محدود ارائه می‌کرد. امروزه، سازمان‌ها به‌سمت مدل‌های عملیاتی و محصول‌محور حرکت می‌کنند که در آن، تیم‌هایِ توانمندِ محصول، حولِ نتایج کسب‌وکار متمرکز می‌شوند، به مشتریان نزدیک‌تر می‌شوند و تصمیمات خود را برای محصول می‌گیرند. کارشناسان مدیریت محصول مانند Teresa Torres، Melissa Perri و Marty Cagan بر این عقیده هستند که تیم‌ها باید به‌صورت منظم، مثلاً هر هفته، با مشتریان خود صحبت کنند.

تیم‌های مدرن، عمری طولانی دارند و به‌طور مستمر در جستجوی نیازهای برآورده نشده‌ی کاربران خود هستند، نه اینکه در پایان پروژه، ازهم جدا شوند. با این روش، نه‌تنها تیم‌ها برای اکتشافات خود و همکاریِ نزدیک با مشتریان، توانمند می‌شوند، بلکه آن‌ها به کار پایدار نیز تشویق می‌شوند. آن‌ها به‌طور نامحدود، از کدهای خود پشتیبانی می‌کنند و بنابراین می‌خواهند آن را سالم نگه دارند تا به‌راحتی تکامل یافته و پشتیبانیِ ویژگی‌های موجود و تولید ویژگی‌های جدید در آن آسان باشد.

حرکت به‌سمت مدل عملیاتیِ محصول‌محور، ابتدا با تعریفِ واضح محصولات سازمان آغاز می‌شود و سپس با نحوه‌ی مواجهه با مشتری و تیم‌های داخلی، همراه با نتایجِ کسب‌وکار و مشتری‌ای که هر محصول به آن کمک می‌کند، ادامه می‌یابد. این کار به‌عنوان طبقه‌بندی محصول (Product Taxonomy) شناخته می‌شود و پایه و اساس ساختار سازمانی و معماری نرم‌افزار است که مدرن سازی معماری نرم افزار می‌تواند کمک کننده باشد. تمام جریان‌های ارزشی که به یک محصول معین کمک می‌کند، با ستاره شمالی (North Star) محصول همسو خواهند شد و همان‌طور که در تصویر زیر نشان داده شده است، به رهبران محصول و فناوری، اطلاعاتی ارزشمند می‌دهد.



طبقه‌بندیِ محصول، به‌عنوان چشم‌اندازی از هدفی که سازمان در تلاش است به آن برسد، عمل می‌کند. این یک راه عالی برای برجسته کردن بخش‌هایی است که بیشترین عدم تطابق بین ساختارهای فعلی و ایده‌آل وجود دارد. این کار آشکار می‌سازد که مدرن سازی معماری نرم افزار در کجا بیشترین سود را خواهد داشت. تصویر بعدی، سناریوی مشترکی را نشان می‌دهد که هر جریان مدرن‌سازی، احتمالاً با آن مواجه می‌شود: مرزهای دامنه برای جریان‌های ارزش هدف، کاملاً با سیستم‌های فناوری اطلاعات موجود ناهمسو هستند و برای همسویی مجدد، نیاز به سرمایه‌گذاری بزرگ برای نوسازی دارند.



مدرن سازی معماری نرم افزار، نه‌تنها فرصتی برای حرکت به‌سمت یک مدل عملیاتی محصول‌محور بوده، بلکه فرصتی هم برای بهبود محصولات است. مدرن و نوسازی فقط به‌معنای بازسازی سیستم قدیمی با فناوری‌ها و الگوهای جدید، در عین حفظ همان ویژگی‌ها نیست؛ بلکه مدرن سازی معماری نرم افزار، مدرن کردنِ محصول، دامنه، فرآیندهای کسب‌وکار و نحوه ایجاد محصولات توسط سازمان نیز است.

محدوده های معماری

پیش از مدرن سازی معماری نرم افزار، لازم است بدانیم که محدوده‌های معماری چیست؟ در سازمان‌های بزرگ با محصولات زیاد و هزاران کارمند، برای مقابله با سطوح بالاتر پیچیدگی، باید به معماری در حوزه‌های مختلف فکر کرد. یکی از ویژگی‌های مهم دامنه‌ها این است که برای وضوح هنگام تصمیم‌گیری ضروری هستند. همه باید محدوده‌ای که در آن کار می‌کنند را درک کنند. به‌عنوان مثال، تصمیمات معماری، مانند انتخاب فناوری، در چه سطحی باید اعمال شوند؟ آیا هر تیمی باید استقلال کامل برای استفاده از هر فناوری داشته باشد؟ آیا باید گروه‌هایی از تیم‌های مرتبط را ملزم به استفاده از فناوری‌های مشابهی کرد؟ یا باید انتخاب‌های فناوری در سطح سازمانی استاندارد شود؟

در این نوشته، از اصطلاحات محدوده ۱، ۲ و ۳ همان‌طور که در تصویر نشان داده شده است، استفاده می‌کنم. دامنه ۱ (معروف به زیردامنه) دامنه‌ای است که به اندازه کافی کوچک و در اختیار تیمی واحد است. بنابراین، توسط یک جریان ارزش واحد سرویس‌دهی می‌شود. دامنه ۲ با گروهی از حوزه‌های ۱ مرتبط است و دامنه ۳ با گروهی از دامنه‌های حوزه ۲ مرتبط هستند. سازمان‌های بزرگ‌تر ممکن است دامنه‌های بیشتری داشته باشند. دقت کنید که شما مجبور نیستید از این اصطلاحات در سازمان خود استفاده کنید و این فرآیند می‌تواند با هر اصطلاحی که در شرکت شما استفاده می‌شود، سازگار شود. اما این مهم است که مشخص کنید در هر زمان، در چه حوزه‌ای کار می‌کنید.

هنگام کار در محدوده ۲ و بالاتر، به‌دلیل پیچیدگیِ تیم‌هایِ متعدد، پاسخ به بسیاری از سؤالات معماری دشوار است:

- جریان‌های ارزش چگونه باید گروه‌بندی شوند؟
- هر گروه چقدر باید استقلال داشته باشد؟
- آیا تکرار زیاد است؟
- کجا باید خدمات مشترکی ساخته شود که توسط جریان‌های ارزشِ چندگانه استفاده شود؟

طبقه‌بندی محصول، همچنین می‌تواند با تعریف سطوح طبقه‌بندی بالاتر مانند گروه‌های محصول و سبد محصولات همسو با نتایج کسب‌وکار، که برای هر سطح بهینه شده‌اند، برای پاسخ به این سؤالات کمک کند. با این حال، شناسایی اهداف استراتژیک روشن در هر سطح، همیشه ساده نیست؛ به‌ویژه برای سازمان‌هایی که درحال گذار از یک مدل عملیاتی پروژه‌محور هستند. در قسمت‌های بعد نشان خواهیم داد که چگونه Wardley Mapping که یک تکنیک نقشه‌برداریِ استراتژی است، برای چنین سناریوهایی عالی عمل می‌کند. می‌توان از آن برای ترسیم چشم‌اندازِ کسب‌وکار در هر حوزه‌ای استفاده کرد. همچنین می‌توان به شناسایی حوزه‌هایی که بیشترین شانس را برای مزیت رقابتی محصولات ارائه می‌کنند و متعاقباً همسویی با نتایج کلیدی دارند، کمک کند.

پلتفرم های توسعه دهنده داخلی

مرزهای دامنه‌ی خوب، وابستگی در تغییر را کاهش می‌دهد و منبع کد جدا شده، تیم‌ها را قادر می‌سازد تا تغییرات کد را بدون وابستگی و اصطکاک انجام دهند. با این حال، یک جنبه‌ی حیاتی دیگر وجود دارد که برای استخراج ارزش از مدرن سازی معماری نرم افزار و فعال کردن جریان سریع تغییرات ضروری است: توانایی تیم‌ها برای تغییر آسان و سریع کدشان و استقرار و پشتیبانی از آن در محیط عملیاتی. اینجاست که پلتفرم‌های توسعه‌دهنده داخلی (Internal Developer Platforms یا IDP) وارد فرایند مدرن سازی معماری نرم افزار می‌شوند.

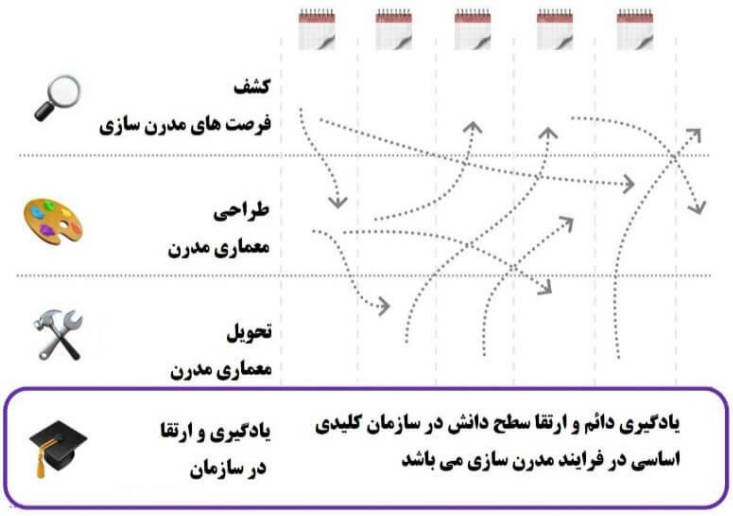
در اصطلاحات توپولوژی تیم، هدف یک پلتفرم، کاهش بار شناختی، ظرفیت شناختی جمعی و تیم‌های همسو با جریان است. یک IDP ابزارهایی را در اختیار تیم قرار می‌دهد که تیم‌ها برای ایجاد، توسعه و پشتیبانی از برنامه‌های خود، به آن‌ها نیاز دارند. یک IDP خوب این قابلیت‌ها را ازطریق یک تجربه توسعه‌دهنده خاص (Developer Experience یا DX) به نمایش می‌گذارد؛ به این معنی که استفاده از پلتفرم، به‌دلیل مستند بودن و سلف‌سرویس بودن آن، آسان است. وقتی تیم‌ها می‌خواهند برنامه‌های جدید ایجاد کنند یا کدی را مستقر کنند، فرآیند باید یکپارچه و ساده باشد و نیازی به برقراری ارتباط با تیم پلتفرم یا هر تیم دیگری وجود نداشته باشد. این فرآیند ارائه‌ی قابلیت‌های موردنیاز تیم‌ها و یک DX خوش‌تعریف و خوب، اغلب به‌عنوان جاده سنگ‌فرش (Paved Road) یا مسیر طلایی (Golden Path) شناخته می‌شود.

امروزه معیارهای ارزیابی برای IDP بسیار بالا تعیین شده است. IDPهای استاندارد طلایی، تیم‌ها را قادر می‌سازند تا برنامه‌های جدید ایجاد کنند و آن‌ها را در یک محیط تولیدی، تنها در چند ساعت یا کمتر، مستقر کنند .علاوه‌بر این، پلتفرم‌ها، معیارها، نظارت، ثبت گزارش، خطوط لوله استقرار و سایر ابزارهای موردنیاز برای این برنامه‌ها را فراهم می‌کنند تا تیم‌ها بتوانند مدل عملیاتی You Build It, You Run It را بدون غرق شدن در زیرساخت‌هایی که جریان ارزش در محصول اصلی آن‌ها را کاهش می‌دهد، اتخاذ کنند.

مدرن سازی معماری نرم افزار سفری در یک مسیر است نه مقصد

برای بسیاری از سازمان‌ها، مدرن سازی معماری نرم افزار و تغییر از معماری قدیمی به مدرن، چندین سال طول می‌کشد. هیچ راه حل سریعی برای سیستم‌هایی که طی سال‌ها یا دهه‌ها دچار افت شده‌اند، وجود ندارد. اما این بدان معنا نیست که باید سال‌ها طول بکشد تا مدرنیزاسیون شروع به ارائه ارزش کند. به‌جای تلقی مدرن سازی معماری نرم افزار به‌عنوان پروژه‌ای که در آن، معماری مثل یک هدف، از ابتدا تعریف شده، سپس یک طرح دقیق برای انتقال از وضعیت فعلی به حالت هدف ایجاد می‌شود، استعاره سفر مناسب‌تر است. در این استعاره نیز اهداف و چشم‌انداز ایجاد می‌شود، اما وقتی گامی به جلو برمی‌داریم، با ظهور بینش‌های جدید، امکان تصحیح فرآیند وجود دارد. با این رویکرد، معمولاً در عرض ۳ تا ۶ ماه، ارزش‌هایی تحویل می‌شوند. ارزش، یک کلمه کلیدی است؛ زیرا هر مرحله در سفر مدرن سازی معماری نرم افزار باید همیشه براساس ارزش تجاری و مشتری باشد. ارتباط برقرار کردن بین تصمیمات معماری و کسب‌وکار و استراتژی محصول، یک ضرورت است. همان‌طور که گفته شد، تکنیک‌هایی مانند Wardley Mapping، به این امر کمک می‌کند.

با استفاده از استعاره سفر، نوسازی و مدرن سازی معماری نرم افزار، جریان‌های متعددی از فعالیت‌ها است. ازجمله کشف فرصت‌های نوسازی جدید، طراحی معماری مدرن، ارائه نوسازی و یادگیری و ارتقا در سازمان. همان‌طور که در تصویر نشان داده شده است، فعالیت می‌تواند در هر جریان، به‌طور همزمان در حوزه‌های مختلف اتفاق بیافتد. بعضی از قسمت‌های یک معماری را می‌توان قبل از شروع کشف در قسمتی دیگر مدرن کرد و به حلقه‌های بازخورد، اجازه ظهور داد.



از آنجایی که مدرنیزاسیون، بسیاری از جنبه‌های مدل کسب‌وکار و عملیات را تحت تأثیر قرار می‌دهد، یادگیری و ارتقاء مهارت‌ها، دو جنبه کلیدی هستند که از نظر اهمیت، با سایر مسیرهای موجود در تصویر، برابری می‌کنند. اینکه از کارکنان انتظار داشته باشیم در مدرن سازی معماری نرم افزار، سیستمی را با موفقیت مدرن‌سازی کنند، در حالی که سال‌هاست در یک طرز فکر معمولی کار می‌کنند، کاملاً غیرواقع‌گرایانه است. علاوه‌بر این، معماری هرگز ثابت نیست و به‌طور مداوم با تغییر استراتژی و زمینه‌های کاری، تکامل خواهد یافت. یادگیری و ارتقاء مهارت به این معنی است که همه تیم‌ها، تکنیک‌هایی مانند EventStorming و مفاهیمی مانند Team Topologies را درک می‌کنند تا بتوانند به‌طور مداوم، محل‌های پیشرفت را شناسایی کرده و ارائه دهند.

یکی از خطرات اصلی در هر سفر مدرنیزه شدن، تمام شدن انگیزه و بازگشت به کارهای معمولی و عادت‌های قدیمی است. برای حفظ شتاب بالا در مدرن سازی معماری نرم افزار، باید یک تیم توانمندسازیِ مدرنیزاسیونِ معماری (Architecture Modernization Enabling Team یا AMET) ایجاد شود. این تیم هدفش این است که با همکاری و رهبری اولویت‌های استراتژیک را تعیین کرده و از طرق مختلف، از تیم‌ها در این سفر حمایت کنند. مثلاً اگر تیمی با Event storming آشنا نیست، مدتی به‌عنوان تسهیل‌گر در کارگاه‌های Event Storming حضور یابند. اگر تیمی مشکلی با تکنولوژی‌ها و معماری‌های مدرن دارد، مدتی به‌عنوان مربی توسعه نرم‌افزار در این تیم‌ها حضور یابد. در مجموع، با حمایت‌های خود، شتاب سفر را بالا نگه دارد. AMET همچنین بر ایجاد شیوه‌های معماری پایدار و پرورش تغییرات پایدار متمرکز است که حتی پس از پایان مدرن سازی معماری نرم افزار نیز ادامه می‌یابد.

جمع بندی

در این نوشته‌ها، اصول و تکنیک‌های زیادی برای پیمودن یک سفر مدرن سازی معماری نرم افزار ارائه خواهد شد اما یک راهنمای گام‌به‌گام یا یک چارچوب سفت و سخت ارائه نخواهیم کرد. شما همیشه باید به انواع سؤالات زیر فکر کنید و در نهایت، تصمیم نهایی را خودتان بگیرید:

- آیا می‌خواهید در سراسر کسب‌وکار گسترده‌تر شوید یا در یک منطقه‌ای خاص، عمیق‌تر شوید؟
- آیا می‌خواهید از هم دور شوید و بسیاری از احتمالات را بررسی کنید یا روی یک تصمیم یا راه حل خاص، همگرا شوید؟
- آیا می‌خواهید یک قدم کوچک‌تر و ایمن‌تر بردارید که ارزش کمتری ارائه می‌کند یا یک گام بزرگ‌تر و پرخطرتر که ارزش بیشتری ارائه می‌کند؟
- چقدر زمان می‌خواهید برروی مدرن سازی معماری نرم افزار در مقابل ادامه توسعه ویژگی‌های معمولی و رفع اشکال، سرمایه‌گذاری کنید؟
- استراتژی شما چیست؟ کجا باید تمرکز کنید، چقدر باید سرمایه‌گذاری کنید و چه چیزی را باید برون‌سپاری کنید؟

مطمئن باشید که تکنیک‌ها و مفاهیم این نوشته، مانند Event Storming، Wardley Mapping و Team Topologies امتحان و آزمایش شده است. این تکنیک‌ها شما را در شرایط بسیار خوبی قرار می‌دهند تا به این سؤالات پاسخ دهید و در هر مرحله از سفر، تصمیم درستی بگیرید.

در طی چندین مطلب، با مفهوم مدرن سازی معماری نرم افزار و تکنیک‌های مرتبط با آن آشنا خواهیم شد. بخش اعظمی از مطالب این مقاله، از نوشته‌ها و سخنرانی‌های آقای Nick Tune گرفته شده و بعضاً تجارب شخصی و مطالعات جانبی نیز به آن اضافه می‌شود که در این صورت، منابع دیگر نیز معرفی خواهند شد.

سال ۱۴۰۱ در روزهای ابتدایی سال نیت کردم که بیشتر بنویسم و این نیت را به‌صورت عمومی بیان کردم. نتیجه این شد که سال گذشته، هیچ مطلبی ننوشتم که البته دلایل زیادی داشت و عبور می‌کنم. به هر حال، امیدوارم امسال بتوانم این مجموعه نوشته‌ها را تکمیل کنم و سال خوبی هم پیش روی همگی ما باشد.