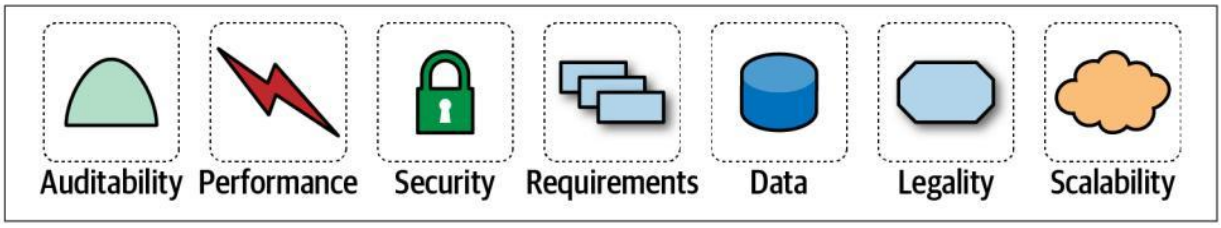
فصل ۴: تعریف ویژگی‌های معماری

وقتی شرکتی تصمیم می‌گیرد مسئله‌ای را با نرم‌افزار حل کند، فهرستی از نیازمندی‌ها را برای آن سیستم گردآوری می‌کند. تکنیک‌های بسیار متنوعی برای فرآیند گردآوری نیازمندی‌ها وجود دارد که عموماً توسط فرآیند توسعه نرم‌افزار مورد استفاده تیم تعریف می‌شوند. اما همان‌طور که در شکل ۴-۱ نشان داده شده است، معمار باید عوامل بسیار دیگری را نیز در طراحی یک راه‌حل نرم‌افزاری در نظر بگیرد.



شکل ۴-۱. یک راه‌حل نرم‌افزاری هم از نیازمندی‌های دامنه و هم از ویژگی‌های معماری تشکیل شده است

معماران ممکن است در تعریف نیازمندی‌های دامنه با کسب‌وکار همکاری کنند، اما یکی از مسئولیت‌های کلیدی آن‌ها شامل تعریف، کشف و تحلیل تمام کارهایی است که نرم‌افزار باید انجام دهد و مستقیماً به عملکرد دامنه مربوط نمی‌شود: *ویژگی‌های معماری* (architectural characteristics).

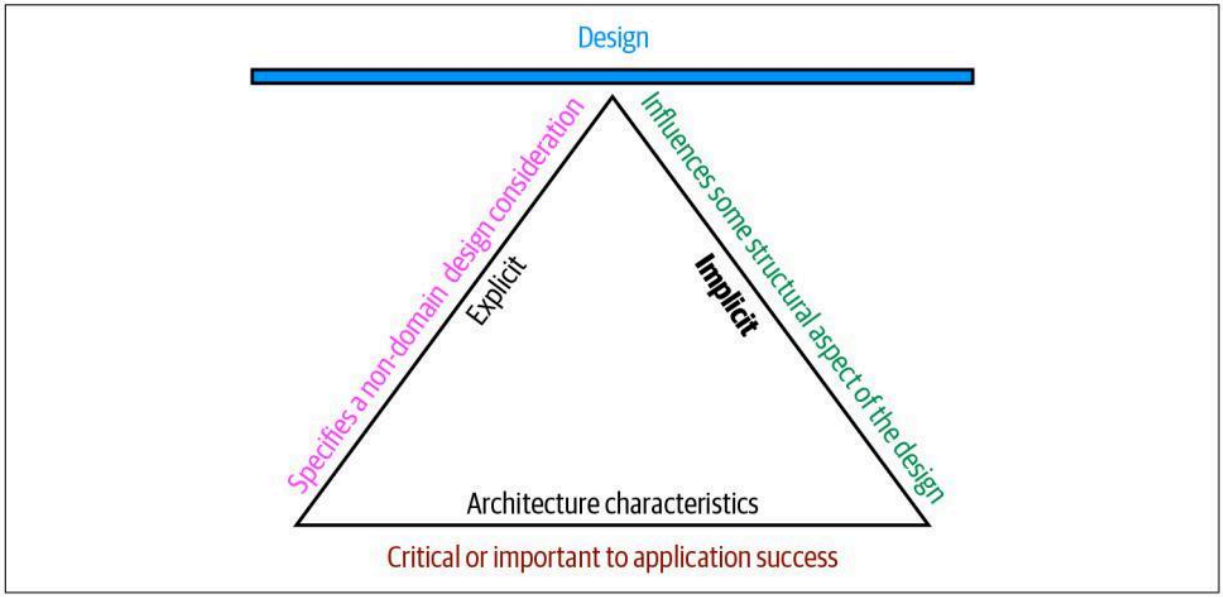
چه چیزی معماری نرم‌افزار را از کدنویسی و طراحی متمایز می‌کند؟ موارد زیادی، از جمله نقشی که معماران در تعریف ویژگی‌های معماری دارند؛ یعنی جنبه‌های مهم سیستم که مستقل از دامنه مسئله هستند. بسیاری از سازمان‌ها این ویژگی‌های نرم‌افزار را با اصطلاحات مختلفی توصیف می‌کنند، از جمله *نیازمندی‌های غیرکارکردی* (nonfunctional requirements)، اما ما این اصطلاح را نمی‌پسندیم زیرا خود-تحقیرآمیز است. معماران این اصطلاح را برای تمایز ویژگی‌های معماری از *نیازمندی‌های کارکردی* (functional requirements) ابداع کردند، اما نامیدن چیزی به عنوان *غیرکارکردی* از دیدگاه زبانی تأثیر منفی دارد: چگونه می‌توان تیم‌ها را متقاعد کرد که به چیزی «غیرکارکردی» توجه کافی داشته باشند؟ اصطلاح محبوب دیگر *صفات کیفیت* (quality attributes) است که آن را نیز نمی‌پسندیم زیرا به جای طراحی، ارزیابی کیفیت پس از واقعیت را القا می‌کند. ما *ویژگی‌های معماری* را ترجیح می‌دهیم زیرا دغدغه‌هایی را توصیف می‌کند که برای موفقیت معماری، و در نتیجه کل سیستم، حیاتی هستند، بدون آنکه از اهمیتشان بکاهد.

یک ویژگی معماری سه معیار را برآورده می‌کند:

- یک ملاحظه طراحی غیرمرتبط با دامنه را مشخص می‌کند.

- بر جنبه‌ای ساختاری از طراحی تأثیر می‌گذارد.
- برای موفقیت برنامه، حیاتی یا مهم است.

این بخش‌های درهم‌تنیده تعریف ما در شکل ۴-۲ نشان داده شده است.



شکل ۴-۲. ویژگی‌های متمایزکننده مشخصه‌های معماری

تعریف نشان داده شده در شکل ۴-۲، علاوه بر چند اصلاح‌کننده، از سه مؤلفه فهرست‌شده تشکیل شده است:

یک ملاحظه طراحی غیرمرتبط با دامنه را مشخص می‌کند

هنگام طراحی یک برنامه، نیازمندی‌ها مشخص می‌کنند که برنامه چه کاری باید انجام دهد؛ ویژگی‌های معماری معیارهای عملیاتی و طراحی را برای موفقیت مشخص می‌کنند که به چگونگی پیاده‌سازی نیازمندی‌ها و چرایی انتخاب‌های خاص مربوط می‌شوند. به عنوان مثال، یک ویژگی معماری مهم و رایج، سطح مشخصی از عملکرد را برای برنامه مشخص می‌کند که اغلب در سند نیازمندی‌ها ظاهر نمی‌شود. حتی مرتبط‌تر: هیچ سند نیازمندی بیان نمی‌کند «از بدهی فنی جلوگیری کنید»، اما این یک ملاحظه طراحی رایج برای معماران و توسعه‌دهندگان است. ما این تمایز بین ویژگی‌های صریح و ضمنی را به تفصیل در بخش «استخراج ویژگی‌های معماری از دغدغه‌های دامنه» در صفحه ۶۵ پوشش می‌دهیم.

بر جنبه‌ای ساختاری از طراحی تأثیر می‌گذارد

دلیل اصلی که معماران سعی در توصیف ویژگی‌های معماری در پروژه‌ها دارند، به ملاحظات طراحی مربوط می‌شود: آیا این ویژگی معماری برای موفقیت نیازمند ملاحظات ساختاری خاصی است؟ به عنوان مثال، *امنیت* (security) تقریباً در هر پروژه‌ای یک دغدغه است و همه سیستم‌ها باید سطح پایه‌ای از اقدامات احتیاطی را در حین طراحی و کدنویسی رعایت کنند. با این حال، زمانی به سطح یک ویژگی معماری ارتقا می‌یابد که معمار نیاز به طراحی چیز خاصی داشته باشد. دو مورد پیرامون پرداخت در یک سیستم نمونه را در نظر بگیرید:

- **پردازشگر پرداخت شخص ثالث:** اگر یک نقطه یکپارچه‌سازی (integration point) جزئیات پرداخت را مدیریت کند، معماری نباید به ملاحظات ساختاری خاصی نیاز داشته باشد. طراحی باید بهداشت امنیتی استاندارد مانند رمزگذاری و هش کردن را در بر گیرد، اما به ساختار خاصی نیاز ندارد.
- **پردازش پرداخت درون‌برنامه‌ای:** اگر برنامه تحت طراحی باید پردازش پرداخت را مدیریت کند، معمار ممکن است یک ماژول، مؤلفه یا سرویس خاص را برای آن منظور طراحی کند تا دغدغه‌های امنیتی حیاتی را به صورت ساختاری جدا کند. اکنون، ویژگی معماری هم بر معماری و هم بر طراحی تأثیر می‌گذارد.

البته، حتی این دو معیار نیز در بسیاری از موارد برای این تصمیم‌گیری کافی نیستند. با این حال، این موارد برخی از ملاحظاتی را که معماران باید هنگام تعیین نحوه طراحی برای قابلیت‌های خاص در نظر بگیرند، نشان می‌دهد.

برای موفقیت برنامه، حیاتی یا مهم است

برنامه‌ها *می‌توانند* از تعداد زیادی ویژگی معماری پشتیبانی کنند... اما نباید. پشتیبانی از هر ویژگی معماری به پیچیدگی طراحی می‌افزاید. بنابراین، یک کار حیاتی برای معماران، انتخاب کمترین تعداد ویژگی‌های معماری به جای بیشترین تعداد ممکن است.

ما همچنین ویژگی‌های معماری را به دو دسته **ضمنی** و **صریح** تقسیم می‌کنیم. ویژگی‌های ضمنی به ندرت در نیازمندی‌ها ظاهر می‌شوند، با این حال برای موفقیت پروژه ضروری هستند. به عنوان مثال، دسترس‌پذیری، پایایی و امنیت تقریباً زیربنای همه برنامه‌ها هستند. معماران باید از دانش خود در مورد دامنه مسئله برای کشف این ویژگی‌های معماری در مرحله تحلیل استفاده کنند.

در شکل ۴-۲، انتخاب مثلث عمدی است: هر یک از عناصر تعریف، دیگری را پشتیبانی می‌کند و آن‌ها نیز به نوبه خود از طراحی کلی سیستم پشتیبانی می‌کنند. نقطه اتکایی که توسط مثلث ایجاد شده، این واقعیت را نشان می‌دهد که این ویژگی‌های معماری اغلب با یکدیگر تعامل دارند و این منجر به استفاده فراگیر از اصطلاح *بده‌بستان* (trade-off) در میان معماران می‌شود.

فهرست (جزئی) ویژگی‌های معماری

ویژگی‌های معماری در طیف گسترده‌ای از سیستم نرم‌افزاری وجود دارند. با وجود تلاش‌ها برای تدوین استانداردهای جهانی، هیچ استاندارد واقعی و جامعی وجود ندارد. در عوض، هر سازمانی تفسیر خود را از این اصطلاحات ایجاد می‌کند. با وجود حجم و مقیاس، معماران معمولاً ویژگی‌های معماری را به دسته‌های کلی تقسیم می‌کنند.

ویژگی‌های معماری عملیاتی

این ویژگی‌ها، قابلیت‌هایی مانند عملکرد، مقیاس‌پذیری، کشسانی، دسترس‌پذیری و پایایی را پوشش می‌دهند.

اصطلاح	تعریف
دسترس‌پذیری (Availability)	مدت زمانی که سیستم باید در دسترس باشد (اگر ۲۴/۷ باشد، باید اقداماتی برای راه‌اندازی سریع سیستم در صورت بروز هرگونه خرابی در نظر گرفته شود).
تداوم (Continuity)	قابلیت بازیابی از فاجعه (Disaster recovery).
عملکرد (Performance)	شامل تست استرس، تحلیل بار حداکثر، تحلیل فرکانس استفاده از توابع، ظرفیت مورد نیاز و زمان‌های پاسخ است.
قابلیت بازیابی (Recoverability)	نیازمندی‌های تداوم کسب‌وکار (به عنوان مثال، در صورت بروز فاجعه، سیستم با چه سرعتی باید دوباره آنلاین شود؟).
پایایی/ایمنی (Reliability/safety)	ارزیابی اینکه آیا سیستم باید ضدخرابی (fail-safe) باشد، یا اینکه آیا به گونه‌ای مأموریت-بحرانی (mission critical) است که بر زندگی افراد تأثیر بگذارد.
استواری (Robustness)	توانایی مدیریت خطا و شرایط مرزی در حین اجرا، در صورت قطع شدن اتصال اینترنت یا قطع برق یا خرابی سخت‌افزار.
مقیاس‌پذیری (Scalability)	توانایی سیستم برای عملکرد و کارکردن با افزایش تعداد کاربران یا درخواست‌ها.

ویژگی‌های معماری ساختاری

معماران باید به ساختار کد توجه کنند و مسئولیت کیفیت کد مانند ماژولاریتی، وابستگی کنترل‌شده و کد خوانا را بر عهده دارند.

اصطلاح	تعریف
قابلیت پیکربندی (Configurability)	توانایی کاربران نهایی برای تغییر آسان جنبه‌های پیکربندی نرم‌افزار.
توسعه‌پذیری (Extensibility)	چقدر مهم است که بتوان قطعات جدیدی از عملکرد را به سیستم اضافه کرد.
قابلیت نصب (Installability)	سهولت نصب سیستم بر روی تمام پلتفرم‌های ضروری.
قابلیت اهرم‌سازی/استفاده مجدد (Leverageability/reuse)	توانایی استفاده از مؤلفه‌های مشترک در چندین محصول.
محلی‌سازی (Localization)	پشتیبانی از چندین زبان، کاراکترهای چندبایتی و واحدهای اندازه‌گیری یا ارزها.
قابلیت نگهداری (Maintainability)	اعمال تغییرات و بهبود سیستم چقدر آسان است؟
قابلیت حمل (Portability)	آیا سیستم نیاز به اجرا بر روی بیش از یک پلتفرم دارد؟
قابلیت پشتیبانی (Supportability)	برنامه به چه سطحی از پشتیبانی فنی، لاگ‌گیری و امکانات اشکال‌زدایی نیاز دارد؟
قابلیت ارتقا (Upgradeability)	توانایی ارتقای آسان/سریع از نسخه قبلی به نسخه جدیدتر.

ویژگی‌های معماری فراگیر (Cross-Cutting)

بسیاری از ویژگی‌ها خارج از دسته‌های معمول قرار می‌گیرند اما محدودیت‌های طراحی مهمی را تشکیل می‌دهند.

اصطلاح	تعریف
دسترس‌پذیری (Accessibility)	دسترسی برای همه کاربران، از جمله کسانی که دارای معلولیت هستند.
قابلیت بایگانی (Archivability)	آیا داده‌ها پس از مدتی نیاز به بایگانی یا حذف خواهند داشت؟
احراز هویت (Authentication)	نیازمندی‌های امنیتی برای اطمینان از اینکه کاربران همان کسی هستند که ادعا می‌کنند.
مجوزدهی (Authorization)	نیازمندی‌های امنیتی برای اطمینان از اینکه کاربران فقط به عملکردهای خاصی دسترسی دارند.
قانونی (Legal)	سیستم در چه محدودیت‌های قانونی‌ای فعالیت می‌کند (GDPR و غیره)؟
حریم خصوصی (Privacy)	توانایی پنهان کردن تراکنش‌ها از کارمندان داخلی شرکت.
امنیت (Security)	آیا داده‌ها باید رمزگذاری شوند؟ چه نوع احراز هویتی باید وجود داشته باشد؟
قابلیت استفاده/دستیابی (Usability/achievability)	سطح آموزش مورد نیاز برای کاربران برای رسیدن به اهدافشان با برنامه.

ایتالیا-پذیری (Italy-ility)

یکی از همکاران نیل داستانی درباره ماهیت منحصر به فرد ویژگی‌های معماری تعریف می‌کند. او برای مشتری‌ای کار می‌کرد که دستور کارش نیازمند یک معماری متمرکز بود. با این حال، برای هر طرح پیشنهادی، اولین سؤال از سوی مشتری این بود: «اما اگر ارتباطمان با ایتالیا را از دست بدهیم چه اتفاقی می‌افتد؟» سال‌ها پیش، به دلیل یک قطعی ارتباطی نادر، دفتر مرکزی ارتباط خود را با شعب ایتالیا از دست داده بود و این از نظر سازمانی یک ضربه روحی بود. بنابراین، یک نیازمندی قطعی برای تمام معماری‌های آینده، چیزی بود که تیم در نهایت آن را *ایتالیا-پذیری* نامید،

که همه می‌دانستند به معنای ترکیبی منحصر به فرد از دسترس‌پذیری، قابلیت بازیابی و انعطاف‌پذیری است.

استاندارد ISO 25010 (نمونه)

سازمان ISO فهرستی از ویژگی‌ها منتشر کرده است که با موارد بالا همپوشانی دارد. در ادامه برخی از تعاریف آن آمده است:

- **بهره‌وری عملکرد:** عملکرد نسبت به منابع مصرفی (شامل رفتار زمانی، بهره‌برداری از منابع، ظرفیت).
- **سازگاری:** توانایی تبادل اطلاعات و عملکرد در یک محیط مشترک (شامل همزیستی، قابلیت همکاری).
- **قابلیت استفاده:** استفاده مؤثر، کارآمد و رضایت‌بخش (شامل یادگیری‌پذیری، محافظت از خطای کاربر، دسترس‌پذیری).
- **پایایی:** عملکرد صحیح تحت شرایط مشخص (شامل بلوغ، دسترس‌پذیری، تحمل خطا، قابلیت بازیابی).
- **امنیت:** محافظت از اطلاعات و داده‌ها (شامل محرمانگی، یکپارچگی، پاسخگویی، اصالت).
- **قابلیت نگهداری:** سهولت تغییر و بهبود نرم‌افزار (شامل ماژولاریتی، قابلیت استفاده مجدد، قابلیت تحلیل، آزمون‌پذیری).
- **قابلیت حمل:** سهولت انتقال به محیط دیگر (شامل سازگاری، قابلیت نصب، قابلیت جایگزینی).

ابهامات فراوان در معماری نرم‌افزار

یک ناامیدی مداوم در میان معماران، عدم وجود تعاریف واضح برای موارد حیاتی است. این باعث می‌شود شرکت‌ها اصطلاحات خود را تعریف کنند که منجر به سردرگمی در سطح صنعت می‌شود. از آنجا که نمی‌توان یک نام‌گذاری استاندارد را تحمیل کرد، بهترین راهکار پیروی از توصیه **طراحی دامنه-محور (DDD)** برای ایجاد و استفاده از یک **زبان فراگیر (Ubiquitous Language)** در میان همکاران است تا سوءتفاهم‌ها کاهش یابد.

بده‌بستان‌ها و معماری کمترین-بدی

برنامه‌ها به دلایل مختلفی فقط می‌توانند از تعداد کمی از ویژگی‌های معماری پشتیبانی کنند. دلیل اصلی این است که هر ویژگی معماری اغلب بر دیگری تأثیر منفی می‌گذارد. به عنوان مثال، بهبود *امنیت*، تقریباً همیشه بر *عملکرد* تأثیر منفی خواهد گذاشت.

این فرآیند مانند پرواز با هلیکوپتر است؛ یک تمرین تعادلی که به خوبی فرآیند بده‌بستان را توصیف می‌کند. بنابراین، معماران به ندرت می‌توانند سیستمی طراحی کنند که تک‌تک ویژگی‌ها را به حداکثر برساند.

هرگز به دنبال معماری *بهترین* نباشید، بلکه به دنبال معماری *کمترین-بدی* (least worst) باشید.

ویژگی‌های معماری بیش از حد منجر به راه‌حل‌های سنگین و ناکارآمد می‌شود. این نشان می‌دهد که معماران باید تلاش کنند تا معماری را تا حد امکان **تکرارشونده (iterative)** طراحی کنند. اگر بتوانید تغییرات را در معماری آسان‌تر انجام دهید، استرس کمتری برای کشف چیز دقیقاً درست در اولین تلاش خواهید داشت. این همان درس مهم **توسعه نرم‌افزار چابک (Agile)** است که در معماری نیز صادق است.