



ویرایش دوم

# مهندسی و ساخت نرم افزار به صورت یک سرویس

رویکردی چابک با استفاده از رایانش ابری

آرماندو فاکس

دیوید پترسون

آرش پورحبیبی

صبا جمالیان



مهندسی و ساخت نرم افزار به صورت یک سرویس:  
رویکردی چابک با استفاده از رایانش ابری  
ویرایش دوم، ۲/۰ ب ۸

آرماندو فاکس و دیوید پترسون

ترجمه توسط آرش پورحبیبی و صبا جمالیان

ویرایش ۰/۰۲

۱۴۰۴ مرداد ۱۹

حق نشر این اثر برای نویسنده‌گان آن آرماندو فاکس و دیوید پترسون و همچنین مترجمان آرش پورحبيبي و صبا جماليان محفوظ است.  
شما آزاد هستيد که از اين مطالب کپي ديجيتال يا چاپي برای استفاده شخصي خود تهيه کنيد.  
شما نمي توانيد بدون اجازه صريح صاحبان حق نشر، اين مطالب را بهصورت ديجيتال يا چاپي، خواه برای نفع مالي يا نه، توزيع مجدد کنيد.

**ویرایش ترجمه: ۰/۰۲**  
**ویرایش متن اصلی: ۲/۰ ب ۸**

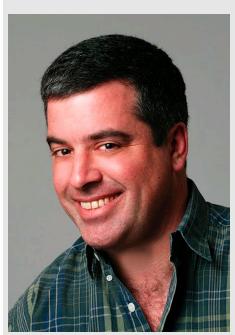
نمای پس زمینه روی جلد عکسی از آبگذر سگوبیا در کشور اسپانیا است. ما آن را به عنوان نمونه‌ای از طراحی زیبا، بادوام و ماندگار انتخاب کردیم. طول کامل این آبگذر حدود ۲۰ مایل (۳۲ کیلومتر) است و توسط رومیان در قرن اول یا دوم پس از میلاد ساخته شده است. این عکس مربوط به قطعه‌ای به طول ۸/۰ کیلومتری و ارتفاع ۲۸ متری است که با استفاده از بلوک‌های گرانیتی و بدون ملات ساخته شده است. طراحان رومی اصول معماری مجموعه ده‌جلدی *De Architectura* (در باب معماری) را که توسط مارکوس ویترویوس پولیو در ۱۵ سال پیش از میلاد نوشته است، دنبال کردند. این بنا تا حوالی سال‌های ۱۵۰۰ پس از میلاد و زمانی که شاه فردیناند و ملکه ایزابلا اولین بازسازی را بر روی طاق‌ها انجام دادند، دست‌نخورده مانده بود. این آبگذر تا همین اوخر مورد استفاده و آبرسانی بود.

عکس روی جلد برگرفته از عکس اصلی متعلق به برنارد گانیون، تحت مجوز CC-BY-SA 3.0:  
[https://commons.wikimedia.org/wiki/Aqueduct#/media/File:Aqueduct\\_of\\_Segovia\\_02.jpg](https://commons.wikimedia.org/wiki/Aqueduct#/media/File:Aqueduct_of_Segovia_02.jpg)

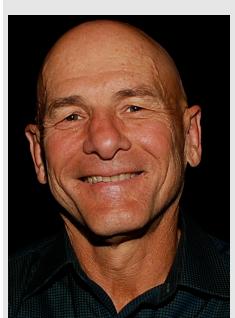
این کتاب با استفاده از LATEX و مجموعه‌ای از بسته‌های رویی تهیه شده است که همگی به صورت آزاد و رایگان در <http://github.com/armandofox/latex2ebook> در دسترس هستند.

آرتور کلپچوکوف طراحی روی جلد و گرافیک تمامی نسخه‌های اصلی کتاب را انجام داده است.

## درباره نویسنده‌گان



**آرماندو فاکس** استاد علوم رایانه دانشگاه کالیفرنیا، برکلی است. وی همچنین مسئولیت معاونت «گوناگونی و انصاف» را هم در سطح بخش مهندسی برق و علوم رایانه و هم سطح دانشگاه را بر عهده دارد. او به عنوان مشاور دانشگاه در زمینه استراتژی یادگیری دیجیتال نیز فعالیت می‌کند. او عنوان «دانشمند برجسته ACM» را دارد و در سال ۲۰۱۵ جایزه مدرس برجسته ACM (که با نام Karl V. Karlstrom شناخته می‌شود) را برای کارش در آموزش مهندسی نرم‌افزار دریافت کرد. پیش از این و در زمانی که در دانشگاه استنفورد مشغول به کار بود، جوازی تدریس و هدایتگری متعددی را از انجمن دانشجویان دانشگاه استنفورد، انجمن زنان مهندس و انجمن افتخاری مهندسی تاو بتا پی دریافت کرد. در سال ۲۰۱۶، او دیگر نویسنده‌این کتاب، دیوید پترسون، جایزه نویبدیخش ترین کتاب درسی جدید ("Texty") را از انجمن نویسنده‌گان کتاب‌های درسی و دانشگاهی برای ویرایش اول این کتاب دریافت کردند. پیش از این، او در طراحی ریزپردازندۀ پنتیوم پرو اینتل نقش داشته، یک شرکت نوپای موفق را برای تجاری‌سازی تحقیقات مربوط به پایان‌نامه خود در دانشگاه کالیفرنیا، برکلی در زمینه رایانش بر روی تلفن‌های همراه که شامل اولین مروگر وب گرافیکی دستگاه‌های همراه جهان Top Gun Wingman (Palm Pilot) بوده است، تأسیس کرده و هم‌بنیان‌گذار چندین شرکت نوپای موفق دیگر نیز بوده است. او مدرک کارشناسی خود را در رشته مهندسی برق و علوم رایانه از مؤسسه فناوری ماساچوست (MIT) و کارشناسی ارشد خود را از دانشگاه ایلینوی اربانا-شمپین دریافت کرد. او همچنین یک موسیقی‌دان آموزش‌دیده کلاسیک، کارگردان موسیقی و دوزبانه/دوفرهنگی (کوبایی-آمریکایی) اهل نیویورک است که به سانفرانسیسکو نقل مکان کرده است.



**دیوید پترسون** اخیراً پس از یک دوره ۴۰ ساله به عنوان استاد علوم رایانه در دانشگاه کالیفرنیا، برکلی بازنشسته شد. او در گذشته به عنوان رئیس بخش علوم رایانه دانشگاه کالیفرنیا، برکلی، رئیس انجمن تحقیقات رایانه‌ای و رئیس انجمن ماشین‌های رایانشی (ACM) خدمت کرده است. شناخته‌شده‌ترین پژوهه‌های تحقیقاتی او عبارت‌اند از: رایانه‌های کم‌دستور (RISC)، آرایه‌های افزونه دیسک‌های ارزان‌قیمت (RAID) و شبکه‌های ایستگاه‌های کاری (NOW). این تحقیقات منجر به تعداد کثیری مقالات علمی، ۶ کتاب و بیش از ۳۵ کسب افتخار شده است. چند جمله از این افتخارها عبارت‌اند از عضویت در آمادن در آکادمی ملی مهندسی، آکادمی ملی علوم و تالار مشاهیر مهندسی سیلیکون‌ولی؛ برگزیده شدن به عنوان قلوب موزه تاریخ رایانه، IEEE، ACM و هر دو سازمان AAAS. وی اخیراً به همراه جان هنسی، استاد دانشگاه استنفورد، به دلیل کارشان بر روی RISC و Mulligan از IEEE و همچنین جایزه کارشناسی از IEEE. پیش از برندۀ شدن جایزه Mulligan کتاب درسی ممتاز ("Texty") برای این کتاب، او همین جایزه را برای کتاب درسی پیش‌گام خود در زمینه معماری رایانه دریافت کرد. او تمام مدارک خود را از دانشگاه کالیفرنیا، لس آنجلس دریافت کرد و همچنین او جایزه فارغ‌التحصیلان دانشگاهی مهندسی برجسته همین دانشگاه را نیز دریافت کرده است. او در کالیفرنیا بزرگ شده است و برای تفریح با دو پسر بزرگ‌سالش وارد رویدادهای ورزشی می‌شود، از جمله بازی‌های هفتگی فوتbal و مسابقات دوچرخه‌سواری خیریه.



## درباره مترجمان



**آرش پورحبیبی** محقق و توسعه‌دهنده حوزه سامانه‌های رایانه‌ای است. وی در حال حاضر در شرکت اوراکل به عنوان مهندس ارشد بر روی سرویس هیت‌ویو (HeatWave) کار می‌کند که سرویس یکپارچه‌ای بری پایگاه داده مای‌اس‌کیوال (MySQL) برای پردازش تراکنش‌ها، تجزیه و تحلیل بلادرنگ داده در انبارهای داده و دریاچه‌های داده و همچنین یادگیری ماشین و هوش مصنوعی مولد است. او دکتراخود را در رشته علوم رایانه و ارتباطات از مؤسسه فناوری فدرال لوزان سوئیس (EPFL) در سال ۱۴۰۰ دریافت کرد. پایان‌نامه دکتراخود را در گریز از پرداخت هزینه و سریار فراخوانی‌های رویه‌ای دوردست (RPC) در مراکز داده از طریق طراحی مشترک ساخت‌افزار و نرم‌افزار متتمرکز بود. وی پیش از این، مدرک کارشناسی ارشد و کارشناسی خود را به ترتیب در سال‌های ۱۳۹۴ و ۱۳۹۲ از دانشگاه شیراز دریافت کرده است. او به تدریس و به‌طور کلی مقوله آموزش علاقه بسیاری دارد و به همین دلیل شروع به ترجمه این کتاب کرده است. او جدای از اینکه یک خود رایانه است، از دویدن لذت می‌برد، عاشق غذا و آشپزی است و همچنین دوست دارد جهان‌گردی کند.



**صبا جمالیان** مهندس سیستم و مدرس علوم رایانه است. در حال حاضر یک مهندس قابلیت اطمینان سایت در شرکت بِریز (Braze) است و به توسعه زیرساخت‌های بری آن‌ها کمک می‌کند. قبل از بِریز، او برای ارائه نرم‌افزار رلیتوپیتی‌وان (RelativityOne) به صورت سرویس یکپارچه‌ای نقش داشت. فرای حضور خود در صنعت، صبا به عنوان یک عضو هیئت علمی کمکی در دانشگاه روزولت در شیکاگو مشغول به تدریس مهندسی نرم‌افزار و طراحی سیستم است. او که فارغ‌التحصیل دانشگاه ایالتی بولینگ گرین (BGSU) در سال ۱۳۹۴ با مدرک کارشناسی ارشد و تخصص روش رایانش بری است، مدرک لیسانس خود را نیز در سال ۱۳۹۲ از دانشگاه شیراز گرفت. او که اعتقاد راسخ به فراهم ساختن دسترسی آزاد به اطلاعات دارد، در ارائه ترجمه فارسی کتاب «یک دست صدا ندارد» نوشته بن کالینز ساسمن و برایان فیتزپاتریک نقش داشته و از مشارکت در ترجمه کتابی که در دستان خود دارید خرسند است.



# فهرست کوتاه

۱	مقدمه‌ای بر نرم‌افزار به صورت یک سرویس، توسعه چابک و رایانش ابری ..	۲
بخش اول: چارچوب‌ها و زبان‌های برنامه‌نویسی SaaS		
۲	چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم .....	۵۶
۳	معماری نرم‌افزار کاربردی SaaS: ریزسروریس‌ها، API‌ها، و REST .....	۹۲
۴	ریلز، یک چارچوب مدل-نما-کنترل‌کر .....	۱۲۲
۵	تجربه‌های پیشرفته برنامه‌نویسی برای SaaS .....	۱۲۴
۶	مقدمه‌ای بر جاوا اسکریپت .....	۱۲۶
بخش دوم: توسعه نرم‌افزار با رویکردی چابک		
۷	توسعه رفتارمحور و روابط‌های کاربری .....	۱۲۸
۸	توسعه آزمون محور .....	۱۳۰
۹	میراث‌کد، بازسازی، و روش‌های چابک .....	۱۳۲
۱۰	تیمهای چابک .....	۱۳۴
۱۱	الگوهای طراحی برای SaaS .....	۱۳۶
۱۲	توسعه/عملیات: به کاراندازی، کارایی، قابلیت اطمینان و امنیت کاربردی ...	۱۳۸
	پس‌گفتار .....	۱۴۰

# فهرست مطالب

ح

پیش‌گفتار مترجمان

ر

پیش‌گفتار نویسندهان

۲

## ۱ مقدمه‌ای بر نرم‌افزار به صورت یک سرویس، توسعهٔ چابک و رایانش ابری

۵	مقدمه .....	۱-۱
۶	فرایندهای توسعهٔ نرم‌افزار: طرح-و-تبث .....	۲-۱
۱۳	فرایندهای توسعهٔ نرم‌افزار: منشور چابک .....	۳-۱
۱۸	تصمیم‌گیری کیفیت نرم‌افزار: آزمون .....	۴-۱
۲۱	بهره‌وری: اختصار، سنتز، بازکاربرد و ابزارها .....	۵-۱
۲۵	نرم‌افزار به صورت یک سرویس و معماری سرویس‌گرا .....	۶-۱
۲۹	به‌کاراندازی SaaS: رایانش ابری .....	۷-۱
۳۲	به‌کاراندازی SaaS: مرورگرها و تلفن همراه .....	۸-۱
۳۸	زیباکد در مقابل میراث‌کد .....	۹-۱
۴۰	گذرنامه بر این کتاب و نحوه استفاده از آن .....	۱۰-۱
۴۵	باورهای نادرست و خطرهای پنهان .....	۱۱-۱
۴۷	نکات پایانی: مهندسی نرم‌افزار چیزی فراتر از برنامه‌نویسی است .....	۱۲-۱

۵۵

## ۱۰۱ نرم‌افزار به صورت یک سرویس: چارچوب‌ها و زبان‌ها

۵۶

## ۱۰۲ چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم

۵۹	پیش‌درآمد: یادگیری چگونگی یادگیری چارچوب‌ها و زبان‌های برنامه‌نویسی .....	۱-۲
۶۳	برنامه‌نویسی دونفره .....	۲-۲
۶۵	معرفی روبی، یک زبان شبیه‌گرا .....	۳-۲
۷۵	اصطلاحات برنامه‌نویسی روبی: حالت شاعرانه، بلوكها، نوع‌دهی اردکی .....	۴-۲
۸۲	پُفک: آشنایی با روبی .....	۵-۲
۸۲	چِهم‌ها و باندلر: مدیریت کتابخانه‌ها در روبی .....	۶-۲
۸۶	باورهای نادرست و خطرهای پنهان .....	۷-۲
۸۸	نکات پایانی: چگونه می‌توان یک زبان را با جست‌وجو در گوگل یاد (ن)گرفت .....	۸-۲

<b>۹۲</b>	<b>۳ معماری نرم افزار کاربردی SaaS: ریز سرویس‌ها، API‌ها، و REST</b>
<b>۹۴</b>	۱-۳ معماری کلاینت-서ور در وب . . . . .
<b>۹۷</b>	۲-۳ ارتباطات در از خطوط سبیری HTTP استفاده می‌کند . . . . .
<b>۱۰۲</b>	۳-۳ پُفک: و HTTP . . . . .
<b>۱۰۲</b>	۴-۳ از وبگاه‌ها تا ریز سرویس‌ها: معماری سرویس‌گرا . . . . .
<b>۱۰۷</b>	۵-۳ واسطه‌های برنامه‌نویسی مبتنی بر REST: همه چیز نوعی منبع است . . . . .
<b>۱۱۳</b>	۶-۳ URI‌های مبتنی بر REST، فراخوانی واسطه‌های برنامه‌نویسی و JSON . . . . .
<b>۱۱۷</b>	۷-۳ پُفک: ساخت و به کاراندازی یک آپ SaaS ساده . . . . .
<b>۱۱۷</b>	۸-۳ باورهای نادرست و خطرهای پنهان . . . . .
<b>۱۱۹</b>	۹-۳ نکات پایانی: سیر تحول از واسطه دروازه مشترک تا معماری سرویس‌گرا . . . . .
<b>۱۲۲</b>	<b>۴ ریلز، یک چارچوب مدل-نما-کنترل‌گر</b>
<b>۱۲۴</b>	<b>۵ تجربیدهای پیشرفته برنامه‌نویسی برای SaaS</b>
<b>۱۲۶</b>	<b>۶ مقدمه‌ای بر جاوا اسکریپت</b>
<b>۱۲۷</b>	<b>دوم توسعه نرم افزار با رویکردی چابک</b>
<b>۱۲۸</b>	۷ توسعه رفتارمحور و روابط‌های کاربری
<b>۱۳۰</b>	۸ توسعه آزمون محور
<b>۱۳۲</b>	۹ میراث‌کد، بازسازی، و روش‌های چابک
<b>۱۳۴</b>	۱۰ تیم‌های چابک
<b>۱۳۶</b>	۱۱ الگوهای طراحی برای SaaS
<b>۱۳۸</b>	۱۲ توسعه/عملیات: به کاراندازی، کارایی، قابلیت اطمینان و امنیت کاربردی
<b>۱۴۰</b>	۱۳ پس‌گفتار
<b>۱۴۱</b>	واژه‌نامه انگلیسی به فارسی
<b>۱۵۹</b>	واژه‌نامه فارسی به انگلیسی
<b>۱۸۱</b>	فهرست اختصارات

# پیش‌گفتار مترجمان

زبان، نقشهٔ راه یک فرهنگ است. به شما می‌گوید مردم آن از کجا آمده‌اند و به کجا می‌روند.

—ریتا مای براون<sup>۱</sup>

زبان، نه فقط ابزار ارتباط، بلکه آینه‌ای از فرهنگ است. واژگان و ساختارهای زبانی، به ما نشان می‌دهند که مردم چگونه فکر می‌کنند، چه چیزهایی را مهم می‌دانند و چگونه با دنیای پیرامون تعامل می‌کنند. ترجمهٔ تخصصی در حوزه‌هایی چون رایانه<sup>۲</sup>، نه فقط به انتقال دانش، بلکه به گسترش زبان و فرهنگ علمی نیز کمک می‌کند.

دایرۀ واژگان تخصصی دنیای رایانه<sup>۳</sup> در زبان فارسی، به واسطۀ ضعف فرهنگی، تخصصی و تاریخی، ناتوان از پاسخ‌گویی کامل به نیاز کاربران و دانش‌پژوهان فارسی‌زبان شده است. این دشواری تنها به کمبود واژگان بسندۀ نمی‌کند، بلکه به نبود نهادهای معیارگذار پویا، نبود منابع آموزشی معتبر و ناهمانگی در معادل‌سازی‌ها نیز برمی‌گردد. در نتیجهٔ این آشفتگی و عدم انسجام، بسیاری از دانش‌پژوهان فارسی‌زبان برای یادگیری این حوزه باید به منابع انگلیسی متولّ شوند. باور ابتدایی ما این بود که زبان انگلیسی پیش‌زمینه‌ای مهم برای یادگیری اصولی علوم رایانه<sup>۴</sup> و یا مهندسی رایانه<sup>۵</sup> است. گرچه متوجه شدیم که زبان انگلیسی و علوم رایانه با هم ارتباط دارند، ولی لازم و ملزم یکدیگر نیستند. عدم مساوات در امکان دسترسی به اطلاعات و آموزش، نه تنها مهم‌ترین عامل ایجاد ناعدالتی است، بلکه از بزرگ‌ترین چالش‌های صنعت فناوری، نرم‌افزار<sup>۶</sup> و رایانه است.

خوشبختانه، تلاش‌های جامعه‌محور می‌توانند این شکاف را پر کنند. برای نمونه، ویکی‌پدیای فارسی این ادعا را ثابت می‌کند که دسترسی همگانی به اطلاعات و آموزش می‌تواند رشد و پیشرفت چشمگیری در این زمینه ایجاد کند. ویکی‌پدیای فارسی یک منبع ارزشمند برای گرفتن اطلاعات غیرتخصصی و تخصصی در زمینه‌های مختلف است که به لطف همکاری داوطلبانه جامعهٔ فارسی‌زبان، امروزه بزرگ‌ترین دانشنامهٔ فارسی محسوب می‌شود. دید ما به ترجمهٔ این کتاب هم دنباله‌رو همین تفکر است و باور داریم ترجمۀ کتاب‌های تخصصی دسترسی به این اطلاعات را برای افراد آسان‌تر می‌کند. امید داریم تا با ترجمهٔ این کتاب، گامی در این راستا برداشته باشیم و چه بسا خوانندگان این کتاب خود به جامعهٔ داوطلبان ویکی‌پدیا<sup>۷</sup> بپیوندند و مدخل‌های جدیدی را با استناد به این کتاب در ویکی‌پدیای فارسی ایجاد کنند.

## علوم رایانه یا مهندسی رایانه؟

در بسیاری از کشورها، رشتهٔ تحصیلی علوم رایانه، رشتهٔ اصلی در زمینهٔ رایانه است و رشتهٔ مهندسی رایانه، کمی مرتبط با مهندسی برق است. در ایران اما، علوم رایانه بیشتر به قسمت نظری این حوزه اشاره دارد و مهندسی رایانه در واقع همان علوم رایانه‌ای متدالول در سایر کشورهاست.

<sup>1</sup>Rita Mae Brown

<sup>2</sup>Computer

<sup>3</sup> با وجود اینکه واژۀ کامپیوتر در زبان فارسی واژه‌ای حافظت‌دار است، ما تصمیم گرفتیم که از «رایانه» به عنوان معادل فارسی استفاده کنیم. یکی از دلایل اصلی این انتخاب، ما، هماهنگی و انسجام بهتر با سایر اصطلاحات وابسته مثل Computer (رایانش) Computing («رایانش») بوده است.

<sup>4</sup>Computer Science

<sup>5</sup>Computer Engineering

<sup>6</sup>Software

<sup>7</sup>Wikipedia

هدف ما از ترجمهٔ این کتاب، ارائهٔ نسخه‌ای فارسی از یکی از مهم‌ترین منابع آموزش مهندسی نرم‌افزار<sup>۸</sup> مدرن است. در این کتاب، روش‌های اصولی برای طراحی و مهندسی نرم‌افزار مدرن با بهره‌گیری از ابزارها و تکنیک‌های پهروز دنیای حرفه‌ای ارائه می‌شود. برای آموزش و انتقال بهتر این مفاهیم، به یک زبان برنامه‌نویسی<sup>۹</sup> به عنوان ابزار اصلی نیاز است که کتاب زبان برنامه‌نویسی روی<sup>۱۰</sup> و به همراه آن چارچوب<sup>۱۱</sup> روی آن ییلز<sup>۱۲</sup> را انتخاب کرده است. همان‌طور که بارها در متن کتاب اشاره خواهد شد، آموزش این زبان و سایر فناوری‌های مربوط به آن موضوع اصلی کتاب نیست. بلکه صرفاً ابزاری است تا بتوان مفاهیم کتاب را به‌طوری عملی به‌کمک آن تجربه کرد. کتاب با یک اکوسیستم جامع از پروژه‌ها، تمرین‌ها و ابزارهایی همراه است که روی یک مخزن<sup>۱۳</sup> عمومی در دسترس هستند. ترجمهٔ این کتاب، تلاشی است برای همراهی با همین اکوسیستم و در دسترس قرار دادن مفاهیم بنیادین این حوزه برای فارسی‌زبانان، بهویژه دانشجویان، برنامه‌نویسان<sup>۱۴</sup> و علاقه‌مندان به طراحی مهندسی نرم‌افزار مدرن. کوشیده‌ایم تا این اثر را با بالاترین دقت، حفظ انسجام فنی و رعایت خوانایی عمومی ترجمه کنیم تا هم برای دانشجویان تازه‌کار و هم برای مدرسان دانشگاهی و متخصصان قابل استفاده باشد. سعی بر این داریم تا کلیهٔ مجموعهٔ پژوهه‌ها و تمرین‌ها را نیز در آیندهٔ نزدیک به فارسی تهیه و ارائه دهیم.

یکی از بزرگترین چالش‌های جامعهٔ فنی فارسی‌زبان، نبود یک زبان تخصصی استاندارد و یکپارچه است. بارها مشاهده شده که متخصصان در صحبت‌ها و نوشته‌های خود از واژگان عمومی یا تخصصی زیادی به زبان انگلیسی استفاده می‌کنند. این واژگان یا اصطلاحات عموماً با تنفس نادرست و گاهی با معنای اشتباه یا خارج از موضوع استفاده می‌شوند. تکرار این عمل منجر به شکل‌گیری یک «زبان میانی» شده است که اصول تعریف‌شدهٔ درستی ندارد، نه انگلیسی است و نه فارسی اصولی. این کتاب، تلاشی است برای مقابله با این چالش و ارائهٔ یک مرجع دقیق و منسجم در حوزهٔ مهندسی نرم‌افزار به زبان فارسی. در طول ترجمه، سعی کردیم با زبانی ساده و در عین حال فنی، مفاهیم را منتقل کنیم، اما با چالش‌های متعددی در انتخاب واژگان فارسی برای مفاهیم فنی روبرو بودیم. بسیاری از واژه‌های تخصصی علوم رایانه در فارسی، یا معادل رایج ندارند یا معادلهای موجود ناهمانگ و گاه متناقض‌اند. از این‌رو تلاش کردیم معادلهایی دقیق، هماهنگ و قابل درک برگزینیم؛ چه با بهره‌گیری از مصوبات فرهنگستان و روایت مجامعت تخصصی، چه با خلق واژه‌های جدید در صورت نیاز (که امیدواریم مورد استقبال خوانندگان و متخصصین این حوزه قرار گیرد).

در این ترجمه، تلاش کردیم که به خواننده کمک کنیم تا در حین یادگیری مطالب، با کلمات کلیدی و تخصصی دنیای رایانه، با زبان انگلیسی نیز آشنایی پیدا کند. این آشنایی می‌تواند به ارتباط ساده‌تر بین فارسی‌زبانان و متون و متابع انگلیسی منجر شود. از این‌رو، هرگاه که یک کلمهٔ کلیدی یا نام خاصی را برای اولین بار به فارسی ترجمه کرده‌ایم، معادل انگلیسی آن را در پانویس همان صفحه قرار داده‌ایم (مگر در مواردی خاص). همچنین در پاره‌ای موارد، توضیحات تکمیلی برای تصریح برخی اصطلاحات در پانویس کتاب آورده‌ایم. علاوه بر این، در انتهای کتاب، یک واژنامه شامل تمامی این کلمات کلیدی و ترجمهٔ فارسی آن‌ها به‌طور دوطرفه قرار داده‌ایم. تلاش کردیم که با سواس و دقت در ترجمهٔ چنین کلماتی، نه تنها به انسجام متن فارسی کمک کنیم، بلکه به خواننده نیز کمک کنیم تا با معادل زبان انگلیسی واژگان نیز آشنا شود.

برای انسجام سبک و صحت زبان، در تمامی متن از زبان فارسی معیار و نگارشی یکدست استفاده کردیم. سعی کردیم از آخرین اصول نگارشی زبان فارسی پیروی کنیم. در این راستا تلاش کردیم تا از نکته‌های نگارش ترجمهٔ متون تخصصی رایانه که توسط دکتر محمد قدسی و دکتر حمید ضرابی‌زاده تدوین شده است<sup>۱۵</sup> نیز پیروی کنیم. از لطف و همکاری خانم مونا اصفهانی برای ویراستاری این اثر نیز سپاسگزاریم. همچنین لازم می‌دانیم از پدیدآورندگان بستهٔ زیرشین<sup>۱۶</sup> (X Persian) و جامعهٔ

<sup>8</sup>Software Engineering

<sup>9</sup>Programming Language

<sup>10</sup>Ruby (Programming Language)

<sup>11</sup>Framework

<sup>12</sup>Ruby On Rails

<sup>13</sup>Repository (Version Control)

<sup>14</sup>Programmer

## پیوندها

توسعه‌دهنگان حوزهٔ **TeX** فارسی، مخصوصاً آقای وفا خلیقی، که حروف‌چینی به زبان فارسی و با کیفیت بالا را ممکن کرده‌اند قدردانی کنیم. سپاسگزار خدمات آقای صابر راستی‌کردار نیز برای تهیهٔ مجموعهٔ قلم<sup>۱۵</sup> وزیر ( شامل وزیرمتن<sup>۱۶</sup> و وزیرکد<sup>۱۷</sup> ) و در اختیار گذاشتن آن به صورت رایگان هستیم؛ پادشاهان گرامی باد.

نهایتاً از آرماندو فاکس<sup>۱۸</sup> و دیوید پترسون<sup>۱۹</sup> که سخاوتمندانه به ما اجازه دادند تا این کتاب را به فارسی ترجمه کنیم و ترجمهٔ فارسی آن را به رایگان در اختیار فارسی‌زبانان بگذاریم صمیمانه سپاسگزاریم. این ترجمه بر اساس ویرایش ۲/۰ ب ۸ نسخهٔ اصلی کتاب است. بی‌تردید این ترجمه، با وجود تلاش فراوان، خالی از نقص نیست. خوشحال می‌شویم اگر پیشنهادها و موارد اصلاحی خود را، بهخصوص در این نسخه‌های ابتدایی، با ما در میان بگذارید. امید داریم که این ترجمه گامی کوچک در مسیر ترویج آموزش علوم رایانه به زبان فارسی باشد. گسترش منابع آموزشی استاندارد به زبان فارسی، ارتقاء کیفیت مطالب در وبکی‌پدیای فارسی و ایجاد انسجام واژگانی در آموزش دانشگاهی از اهداف بلندمدت ما در این مسیر هستند. باور داریم که فراهم کردن دسترسی رایگان، ساده و علمی به متون مرجع، می‌تواند به پرورش نسل جدیدی از دانش‌آموختگان و پژوهشگران فارسی‌زبان در حوزهٔ فناوری<sup>۲۰</sup> کمک کند. این ترجمه نه فقط کوششی برای فارسی‌سازی یک متن انگلیسی، بلکه تلاشی برای پیوند زدن زبان فارسی به بدنۀ دانش فنی روز جهان است. با امید به اینکه این ترجمه به درک بهتر مفاهیم مهندسی نرم‌افزار برای فارسی‌زبانان بینجامد.

آرش و صبا  
۱۴۰۲ پاییز

## پیوندها

<https://sharif.edu/~ghodsi/typing-rules.pdf><sup>۱۵</sup>

<https://ctan.org/pkg/xepersian><sup>۱۶</sup>

<https://rastikerdar.github.io/vazirmatn/><sup>۱۷</sup>

<https://rastikerdar.github.io/vazir-code-font/><sup>۱۸</sup>

<sup>۱۵</sup>Font (Computer)

<sup>۱۶</sup>Armando Fox

<sup>۱۷</sup>David Patterson

<sup>۱۸</sup>Technology



# پیشگفتاری بر ویرایش دوم

اگر می‌خواهید کشتی بسازید، مردان را برای جمجمه آوری چوب، تقسیم کار و صدور دستورات گرد هم نیاورید. بلکه در آن‌ها اشتیاقی برای دریاهای وسیع و بی‌پایان برانگیزید.

—آنتوان دو سنت‌اگزوپری<sup>۱</sup>، قلعه<sup>۲</sup>، ۱۹۴۸

اگر دلتان برای «خوشامدگویی» ویرایش اول این کتاب تنگ شده است، می‌توانید آن را از طریق وبگاه <http://www.saasbook.info/welcome-1st-edition> مطالعه کنید.  
ما ویرایش اول این کتاب را در سال ۲۰۱۴ با هدف کمک به سایر مدرسان و دانشجویان مهندسی نرم‌افزار منتشر کردیم تا آن‌ها نیز آنچه را که ما کشف کرده بودیم امتحان کنند: متداول‌ترین نرم‌افزار نرم‌افزار به صورت یک سرویس نه تنها یک روش عالی برای توسعه و به کاراندازی نرم‌افزار است، بلکه راهکار بسیار مناسبی برای آموزش مهندسی نرم‌افزار نیز محسوب می‌شود. ما از موقوفیت این کتاب، محتوای آموزشی همراه آن (برای اطلاعات بیشتر به [saasbook.info](http://saasbook.info)<sup>۳</sup> مراجعه کنید) و دوره‌های آموزشی آزاد گستردۀ<sup>۴</sup> در [edX.org](http://edX.org)<sup>۵</sup> که باعث گسترش بیشتر این دانش و ترویج این دیدگاه شده‌اند، شگفت‌زده و مفتخر شده‌ایم!

چه چیزهایی عوض شده‌اند: Codio و CHIPS و COD

ویرایش دوم این کتاب دارای ساختاری بهبودیافته (به نظر ما) و همچنین بازنگری‌های اساسی در تقریباً نیمی از مطالب است.

**محتوای آموزشی محتوامحور و پُفک.** بیشتر قسمت‌های این کتاب بر اساس روش‌های آموزشی محتوامحور<sup>۶</sup> (COD) طراحی شده‌اند: مجموعه‌ای از مفاهیم کلیدی که به یادگیرنده چگونه اندیشیدن دربارهٔ یک ایدهٔ مهم را نشان می‌دهند. در میان این قسمت‌ها، پروژه‌ها و فعالیت‌های عملی/کدنویسی<sup>۷</sup> (پُفک) قرار گرفته‌اند که در آن دانشجویان از طریق انجام تمرینات عملی، مفاهیم قسمت‌های آموزشی محتوامحور را به کار می‌گیرند. هر پُفک<sup>۷</sup> دارای امتیازی بین یک تا سه است که با نشان «آب‌گگر» نشانه‌گذاری شده است و میزان زمان و تلاش موردنیاز برای تکمیل آن را نشان می‌دهد.

چرا این همه نقل قول؟ ما معتقدیم که نقل قول‌ها خواندن کتاب را جذاب‌تر می‌کنند، اما علاوه بر آن، ابزاری مؤثر برای انتقال خرد پیشینیان و همچنین برای ایجاد استانداردهای فرهنگی برای مهندسی نرم‌افزار خوب هستند. علاوه بر این می‌خواهیم خوانندگان با بخشی از تاریخ این حوزه آشنا شوند به همین دلیل در ابتدای هر فصل و در سراسر متن این کتاب از نقل قول‌های برندگان جایزهٔ توریگ استفاده کرده‌ایم.

<sup>1</sup>Antoine de Saint-Exupéry

<sup>2</sup>Citadelle (Book)

<sup>3</sup>Massive Open Online Course (MOOC)

<sup>4</sup>edX

<sup>5</sup>Content-Oriented Didactic (COD)

<sup>6</sup>Coding/Hands-On Integrated Projects (CHIPS)

<sup>7</sup>ما در این کتاب عبارت Coding/Hands-On Integrated Projects را به صورت «پروژه‌ها و فعالیت‌های عملی/کدنویسی» ترجمه کردیم و شرname «پُفک» را به عنوان معادل فارسی کوتاه‌نوشت CHIPS که در نسخه اصلی کتاب مورد استفاده قرار گرفته است، در نظر گرفتیم.

**دوره آموزشی یکپارچه با استفاده از گدیو.** ما با گدیو<sup>۸</sup> همکاری نزدیکی داشته‌ایم تا محتوای آموزشی محتوامحور، پُفکها و سامانه ارزیابی خودکار را در بستر و محیط توسعه یکپارچه آموزشی گدیو ادغام کنیم. ما بهشدت توصیه می‌کنیم که اساتید و دانشجویان از گدیو برای شروع سریع تر استفاده کنند، زیرا این سکو شامل سامانه ارزیابی خودکار برای بیشتر پُفکها و محیطی از پیش تنظیم شده با نسخه‌های صحیح و مناسب همه ابزارهای موردنیاز است. برای شروع، به codio.com/esaas مراجعه کنید. اگر از گدیو استفاده نمی‌کنید، کدهای آغازین و مستندات موردنیاز دانشجویان برای هر پُفک در یک مخزن عمومی در گیت‌هاب<sup>۹</sup> در دسترس است که نام آن در هر پُفک مشخص شده است. همچنین اساتید می‌توانند برای دریافت راه حل‌های مرجع و سامانه ارزیابی خودکار سازگار با گریداسکوب<sup>۱۰</sup> به saasbook.info مراجعه کنند.

## چه چیزهایی عوض شده‌اند: تغییرات اساسی در محتوا

**ارائه SaaS با رویکرد تلفن‌همراه و API-محور.** از زمان انتشار ویرایش اول این کتاب، مدل ترکیبی رایانش ابری در کنار کلایینتها همچنان روش غالب در توسعه نرم‌افزار بوده است. اما نرم‌افزار به صورت یک سرویس (SaaS) از ارائه صرف یک نمای مبتنی بر HTML فراتر رفته و به ارائه داده به کلایینتها تلفن‌همراه از طریق واسطه‌های برنامه‌نویسی (API) تغییر مسیر داده است. همچنین، توجه بیشتری به طراحی برای افراد دارای معلولیت صورت گرفته است. به همین دلیل، ما از همان ابتدا بر رویکرد API-محور و مبتنی بر منابع در طراحی سمت سرور تأکید می‌کنیم. همچنین، در سمت کلایین، استفاده از چارچوب‌های مبتنی بر استانداردهای باز<sup>۱۱</sup> و مطابق با تلفن‌همراه (مانند بوت استرپ) را توصیه می‌کنیم. این رویکرد جدید ارائه API-محور به یادگیرندگان کمک می‌کند تا طراحی آپ‌های خود را با تمرکز بر منابع در نظر بگیرند. همچنین می‌آموزند که چگونه یک واسط برنامه‌نویسی مبتنی بر REST این منابع را در اختیار یک کلایین قرار می‌دهد. پس از درک این مفاهیم، می‌توانند همین طرز فکر را به توسعه آپ‌های تلفن‌همراه منتقل کنند.

**از «یادگیری روبی و ریلز» به «یادگیری یک زبان برنامه‌نویسی و چارچوب».** با توجه به اینکه زبان‌های برنامه‌نویسی و چارچوب‌های توسعه نرم‌افزار دائمًا در حال تغییر و تحول هستند، توضیحات مربوط به روبی، ریلز و جاوا‌اسکریپت در این ویرایش جدید به یک استراتژی کلی‌تر برای یادگیری سریع زبان‌ها و چارچوب‌های جدید گسترش یافته است. همچنین، تأکید بیشتری بر درک رابطه بین یک چارچوب و ویژگی‌های زبانی که باعث سازگاری و عملکرد بهتر آن می‌شود صورت گرفته است.

## چه چیزی تغییر نکرده است؟

دانشجویان ما در دانشگاه کالیفرنیا، برکلی همچنان می‌پرسند: «آیا این دوره آموزشی به من فلان چیز را آموزش می‌دهد؟» که در سال ۱۹۲۱، جایگزین‌های رایج برای فلان چیز شامل ریاکت، سرویس لامبدای آمازون، مانگودی‌بی<sup>۱۲</sup> و نود چی‌اس بودند. پاسخ ما از ویرایش اول تاکنون تغییری نکرده است. اکوسیستم حوزه نرم‌افزار آن‌قدر سریع تکامل می‌یابد که در هر مقطع زمانی، چارچوبها و ابزارهای متعددی برای انتخاب وجود دارد. از آنجا که انتخاب‌های ما نمی‌توانند همه را راضی کنند و به احتمال زیاد در چند سال آینده قدیمی خواهند شد، همچنان ابزارهایی را برمی‌گزینیم که به بهترین شکل از هدف آموزشی ما در آموزش یک روش خاص برای توسعه نرم‌افزارهای عالی پشتیبانی می‌کنند. امید ما این است که یادگیرندگان بتوانند با استفاده از روش‌ها و اصول پیشنهادی ما، زبان‌ها و چارچوب‌های جدید را به سرعت بیاموزند.

<sup>8</sup>Codio

<sup>9</sup>GitHub

<sup>10</sup>Gradescope (Software)

<sup>11</sup>Open Standard

<sup>12</sup>MongoDB (Database)

## سیپاس‌گزاری

لازم می‌دانیم که از چندین نفر قدردانی و سیپاس‌گزاری کنیم، چراکه واقعاً ایجاد و نگهداری یک مجموعه خوب از مطالب درسی نیاز به همکاری گسترده دارد. علاوه بر افرادی که در ویرایش اول از آن‌ها تشکر کرده‌ایم، همکاران زیر به‌طور ویژه در بازبینی فنی تغییرات ویرایش دوم کمک شایانی کرده‌اند: کریستین استیونز-مارتینز، استاد دانشگاه دوک (فصل ۱)؛ مارک اسماکر، استاد دانشگاه واترلو (فصل ۲)؛ بلاگوستا کوستو، مؤسسه فناوری فدرال لوزان سوئیس (فصل ۳)؛ مایکل وردیکیو، استاد دانشگاه نظامی سیتادل کارولینای جنوبی (فصل ۴)؛ ماتیاس ماسکاتزینی، توسعه‌دهنده مستقل روبی آن ریلز (فصل ۵)؛ تام هستینگز، استاد دانشگاه کلرادو در کلرادو اسپرینگز (فصل ۶)؛ سام ریچارد، شرکت گوگل، برای مطالب فصول ۲ و ۶ درباره برنامه‌های وب پیشرو (PWA)؛ هنک واکر، استاد دانشگاه تکراس ای‌اندام (فصلو ۷ و ۸)؛ پرابهات وایشن، استاد مؤسسه فناوری نیوجرسی (فصل ۹)؛ آنانستازیا کوردیا، استاد دانشگاه تولین (فصل ۱۰)؛ اد گرینگر، استاد دانشگاه ایالتی کارولینای شمالی (فصل ۱۱)؛ دانیل کوردیرو، استاد دانشگاه سائوپائولو (فصل ۱۲) در نهایت، از پیتر ژانگ، متخصص فناوری حقوقی در ملبورن، استرالیا، به‌خاطر بازبینی دقیق و جامع کل ویرایش دوم صمیمانه قدردانی می‌کنیم.

همچنین باید یادی کنیم از اعضای اصلی و تأثیرگذار گروه آزمایشی ما که در روزهای نخستین انتشار ویرایش اول شکل گرفت و همچنان با ما همراه بوده‌اند و به صورت فعال با پشتیبانی و مشارکت خود نقش بزرگی در بهبود این مجموعه آموزشی داشته‌اند: مایکل وردیکیو و هنک واکر از فهرست بالا، به‌علاوه روز ویلیامز، از دانشگاه بین‌گهمنتون و کریستن والکوت-جاستیس، از دانشگاه کلرادو در کلرادو اسپرینگز.

همکاران ما در شرکت گیت‌هاب، به‌ویژه ونسا چناری (که با نام کاربری @mozzadrella نیز شناخته می‌شود)، مدیر بخش آموزش توسعه‌دهنگان، همچنان بیش از حد انتظار ما از تلاش‌های آموزشی‌مان حمایت کرده‌اند.

تیم گددیو، به‌ویژه الیز دیتریک، مکس کرافی و مدیرعامل شرکت فیلیپ اسنالون، کار فوق العاده‌ای در یکپارچه‌سازی مطالب کتاب و تمرین‌ها در سکوی گددیو انجام داده‌اند که تجربه‌ای روان و بی‌دردرس برای مدرسان و دانشجویان فراهم می‌کند. امیدواریم این تجربه یکپارچه، یادگیری مطالب این کتاب را برای تعداد بیشتری از دانشجویان جذاب کند.

در نهایت، همچون همیشه، از هزاران دانشجوی دانشگاه کالیفرنیا، برکلی و دستیاران آموزشی و همچنین صدها هزار دانشجو که از طریق دوره‌های آموزشی آزاد گسترده ما را دنبال می‌کردند، به‌خاطر کمک در رفع اشکالات و علاقهً مداومشان به این مطالب سیپاس‌گزاریم!

آرماندو فاکس  
ژانویه ۲۰۲۱  
دانشجوی فرانسیسکو، کالیفرنیا

## پیوندها

<http://www.saasbook.info/welcome-1st-edition>  
<http://www.saasbook.info>  
<https://codio.com/esaas>

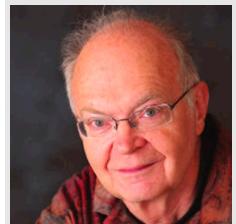


# مقدمه‌ای بر نرم‌افزار به صورت یک سرویس، توسعهٔ چاپک و رایانش ابری

اجازه دهید نگرش سنتی خود را نسبت به ساخت برنامه‌ها تغییر دهیم: به جای اینکه فکر کنیم مهمترین وظیفه‌مان این است که به یک رایانه دستور دهیم که چه کاری انجام دهد، بیشتر بر این تمرکز کنیم که به انسان‌ها توضیح دهیم می‌خواهیم رایانه چه عملی انجام دهد.

—دانلود گنوث، برنامه‌نویسی ادبیانه، ۱۹۸۴

**دانلود گنوث (۱۹۳۸–)**، یکی از برجهسته‌ترین دانشمندان علوم رایانه است. او جایزهٔ تورینگ را در سال ۱۹۷۴ به خاطر مشارکت‌های عمده در تجزیه و تحلیل الگوریتم‌ها و طراحی زبان‌های برنامه‌نویسی و به‌ویژه به‌خاطر تهیه کتاب چندجلدی خود به نام هنر برنامه‌نویسی رایانه‌ای، دریافت کرد. این کتاب به باور سبیاری مرجع قاطع برای تجزیه و تحلیل الگوریتم‌ها است. گنوث همچنین سامانه حروفچینی  $\text{\TeX}$  را ابداع کرد که این کتاب با استفاده از آن تهیه شده است.



۵	.....	۱-۱	مقدمه
۶	.....	۲-۱	فرایندهای توسعهٔ نرم‌افزار: طرح-و-ثبت
۱۳	.....	۳-۱	فرایندهای توسعهٔ نرم‌افزار: منشور چاپک
۱۸	.....	۴-۱	تضمين کیفیت نرم‌افزار: آزمون
۲۱	.....	۵-۱	بهره‌وری: اختصار، سنتز، بازکاربرد و ابزارها
۲۵	.....	۶-۱	نرم‌افزار به صورت یک سرویس و معماری سرویس‌گرا
۲۹	.....	۷-۱	به‌کاراندازی SaaS: رایانش ابری
۳۲	.....	۸-۱	به‌کاراندازی SaaS: مرورگرها و تلفن همراه
۳۸	.....	۹-۱	زیباگد در مقابل میراث کد
۴۰	.....	۱۰-۱	گذری بر این کتاب و نحوهٔ استفاده از آن
۴۵	.....	۱۱-۱	باورهای نادرست و خطرهای پنهان
۴۷	.....	۱۲-۱	نکات پایانی: مهندسی نرم‌افزار چیزی فراتر از برنامه‌نویسی است

## پیش‌نیازها و مفاهیم

هر فصل با خلاصه‌ای کوتاه از پیش‌نیازها و مفاهیم مهم فصل شروع می‌شود. پیش‌نیازها، مهارت‌ها یا دانشی هستند که می‌بایست از پیش داشته باشید تا از مطالب آن فصل بیشترین بهره را ببرید. مفاهیم، ایده‌های اصلی هستند که می‌خواهیم پس از پایان فصل، فارغ از جزئیات مطرح شده، آنها را به خاطر بسپارید.

### پیش‌نیازها:

اولین پیش‌نیاز این است که مشخص کنید آیا این کتاب برای شما مناسب است یا خیر!

این کتاب برای شما مناسب است اگر می‌خواهید...
فقط چارچوب فلان را بیاموزید، بدون اینکه اصول طراحی آن را بدانید
یک آموزش گام به گام مشابه یک «دستور پخت» را برای یک چارچوب یا زبان برنامه‌نویسی خاص دنبال کنید
فقط با خواندن و تماشای فیلم یاد بگیرید
به اصول طراحی نرم‌افزار اجازه دهید تا ارزیابی و ایجاد فناوری‌های جدید را به شما اطلاع دهد
چگونگی یادگیری چارچوب‌ها و زبان‌های برنامه‌نویسی جدید را یاد بگیرید و به سرعت از آنها استفاده کنید
با انجام دادن و تمرین کردن یاد بگیرید

پیش‌نیاز فنی این فصل داشتن آگاهی ابتدایی در مورد **HTML**، زبان نشانه‌گذاری ابیمتنی است که زبان اصلی یا میانجی وب است. ما مطالعهً مقدمه‌ای بر **HTML** از شبکه توسعه‌دهندگان موزیلا را که به صورت رایگان در اختیار عموم قرار دارد، توصیه می‌کنیم. پیشنهاد می‌کنیم تمام بخش‌های تحت عنوان **راهنمای راهنمایها و همچنین دو ارزشیابی را بررسی کنید.**

### مفاهیم:

مفاهیم مهم این فصل، تضاد بین توسعه نرم‌افزاری طرح-و-ثبت و توسعه نرم‌افزاری چاپک و همچنین هم‌افزایی میان توسعه چاپک، نرم‌افزار به صورت یک سرویس و رایانش ابری است.

• **فرابیندهای طرح-و-ثبت و یا چرخه‌های حیات** محصول متنکی به طرح و برنامه‌ریزی دقیق و اولیه است، درحالی‌که **توسعه نرم‌افزار چاپک** به اصلاح تدریجی یک پیش‌نمونه با بازخورد مستمر از مشتری در طی چندین تکرار ۱-۴ هفته‌ای متنکی است. از میان این دو، توسعه چاپک سابقه درخشنان‌تری در مدیریت تغییرات، اجرای پروژه‌های فشرده با تیمهای کوچک و تحويل به موقع نرم‌افزار با کیفیت در محدوده بودجه تعیین شده را دارد.

• **کیفیت نرم‌افزار** به عنوان ارائه ارزش تجاری برای مشتریان و توسعه‌دهندگان تعریف شده است و شامل انواع مختلفی از آزمون‌ها می‌شود. در توسعه چاپک، خود توسعه‌دهندگان، به جای یک تیم مستقل تضمین کیفیت، مسئولیت اصلی کیفیت نرم‌افزار را بر عهده دارند.

• **وضوح از طریق اختصار، سنتز، بازکاربردپذیری و خودکارسازی** به کمک ابزارها چهار روش برای بهبود بهره‌وری توسعه‌دهندگان هستند. چارچوب برنامه‌سازی **روبی آن ریلز** هر چهار روش را به کار می‌گیرد.

• **نرم‌افزار به صورت یک سرویس (SaaS)** نرم‌افزار است که بر روی سرورهای اینترنتی مستقر شده است و از طریق آن‌ها در دسترس میلیون‌ها کاربر قرار دارد. در مقایسه با **نرم‌افزار به صورت یک محصول (SaaS)** که کاربران بر روی دستگاه‌های خود نصب می‌کنند، ارتفا و تکامل SaaS آسان‌تر است زیرا در واقع تنها یک نسخه از آن نصب و به کاراندازی شده است. **رایانش ابری** با استفاده از رایانه‌های انبارگون حاوی ۱۰۰,۰۰۰ سرور، محاسبات و ذخیره‌سازی

## فصل ۱. مقدمه‌ای بر توسعهٔ چاپک و SaaS

قابل اطمینان و مقیاس‌پذیر را برای SaaS فراهم می‌کند. «صرفه به مقیاس» این امکان را می‌دهد تا رایانش ابری به عنوان یکی از خدمات رفاهی (همانند آب و برق)، ارائه شود به‌طوری که شما فقط برای همان مقداری که استفاده کردید، هزینه می‌پردازید.

- دستگاه‌های تلفن همراه اکنون بیشترین سهم بازدید از وبگاه‌ها را تشکیل می‌دهند. برای توسعه، آزمون و به‌کاراندازی آپ‌های «تلفن همراه‌محور» که به‌طور خاص برای این دستگاه‌ها ساخته می‌شوند، می‌توان از همان ابزارهایی بهره برد که برای SaaS تحت مروگرهای دسکتاپ استفاده می‌شود. در این راه می‌توان از **طراحی وب واکنش‌گرا** برای انطباق خودکار با اندازه‌های مختلف صفحه نمایش و در عین سازگاری با کاربران دارای معلولیت استفاده کرد.
- تکامل و توسعه **میراث‌کد** در دنیای واقعی حیاتی است، اما اغلب در کتاب‌ها و دوره‌های مهندسی نرم‌افزار نادیده گرفته می‌شود. در روش‌های چاپک، توسعه‌دهندگان در هر دوره کد را مرتبا بهبود می‌بخشند، بنابراین مهارت‌های به‌دست‌آمده برای کار با میراث‌کد نیز به کار می‌آیند.

معیار	آمازون	اکتبر - ACA	نومبر - ACA	دسامبر - ACA
مشتری در روز (هدف)	-	۵۰,۰۰۰	۵۰,۰۰۰	۳۰,۰۰۰
مشتری در روز (محقق شده)	۱۰,۰۰۰,۰۰۰ <	۸۰۰	۳,۷۰۰	۳۴,۳۰۰
میانگین زمان پاسخ (ثانیه)	۰,۲	۸	۱	۱
مدت خرایی در ماه (ساعت)	۰,۰۷	۴۴۶	۱۰۷	۳۶
زمان در دسترس بودن (درصد)	۹۹/۳۹٪	۴۰٪	۸۵٪	۹۵٪
نرخ خطأ	-	۱۰٪	۱۰٪	-
امن	بله	خیر	خیر	خیر

شکل ۱-۱: مقایسه عملکرد وبگاه آمازون با Healthcare.gov یا همان، وبگاه ACA، در سه ماه اول شروع به کارش (توبر ۲۰۱۳). بعد از یک شروع نامطمئن، ضربالاجل از ۱۵ دسامبر ۲۰۱۳ تا ۳۱ مارچ ۲۰۱۴ جایه‌جا شد که همین مسئله، کاهش «مشتری در روز» مورد انتظار در ماه دسامبر را توضیح می‌دهد. توجه داشته باشد که در محاسبه زمان دسترس پذیری و بگاه ACA، زمان‌های صرف شده برای «فعالیت‌های نگهداری از پیش برنامه‌بازی شده» در نظر گرفته شدند، در صورتی که در مورد وبگاه آمازون این زمان‌ها محاسبه شده‌اند (راپتیشن ۲). نرخ خطأ برای خطاهای اساسی در فرم‌هایی که به شرکت‌های بیمه فروخته شده محسوب شده (هورسلی ۲۰۱۳). بسیاری از کارشناسان حوزهٔ امنیت، وبگاه را نامن خوانند، احتمالاً به این دلیل که توسعه دهنده‌ان تحت فشار زیادی برای ایجاد قابلیت‌های مورد نظر بودند و توجه کمی به امنیت داشتند (هرینگتون ۲۰۱۳).

## ۱-۱ مقدمه

خیلی ساده است. این وبگاهی است که می‌توانید در آن بیمه‌های درمانی مقررین به صرفه تهیه کنید، همانطوری که از (وبگاه) کایاک<sup>۱</sup> بلیت هوایی‌ها تهیه می‌کنید یا از (وبگاه) آمازون<sup>۲</sup> یک تلویزیون می‌خرید... از همین سه‌شنبه، هر آمریکایی می‌تواند به HealthCare.gov برود تا با چیزی به نام بازارچه بیمه آشنا شود... پس به دوستان و خانواده خود بگویید... مطمئن شوید که در وبگاه ثبت‌نام می‌کنند. بیایید به هموطنانمان کمک کنیم تحت پوشش بیمه قرار گیرند. (صداي تشويق).

- رئیس جمهور باراک اوباما، سخنرانی در مورد لایحهٔ مراقبت مقررین به صرفه<sup>۳</sup>، دانشگاه دولتی پرنス جرج، مریلند، ۲۶ سپتامبر ۲۰۱۳

...اکنون شش ماه از آغاز به کار بازارچه بیمه مرتبط به لایحهٔ مراقبت مقررین به صرفه می‌گذرد. می‌توانم بگویم این حرکت تا اینجای کار با پستی و بلندی‌های همراه بوده و فکر می‌کنم همه متوجه باشند که من هم از اینکه این حرکت، خوب می‌دونید، با مشکلات مختلفی روبرو بوده خوشحال نیستم و این مسائل فکر من را مشغول کرده.

- رئیس جمهور باراک اوباما، سخنرانی در مورد لایحهٔ مراقبت مقررین به صرفه، ملاقات با خبرنگاران در کاخ سفید، ۱۴ نوامبر ۲۰۱۳

وقتی لایحهٔ مراقبت مقررین به صرفه (ACA) در سال ۲۰۱۰ تصویب شد، به عنوان بلندپروازه‌ترین برنامه اجتماعی آمریکا طی چند دهه شناخته می‌شد، و می‌توان گفت گل سرسید فعالیت‌های دولت، اوباما بود. همانطور که می‌لیون‌ها نفر از طریق وبگاه<sup>۴</sup> آمازون اجناس مختلف خریداری می‌کنند، HealthCare.gov -که به همان وبگاه لایحهٔ مراقبت مقررین به صرفه معروف است- نیز قرار بود به میلیون‌ها آمریکایی که تحت پوشش بیمه نیستند امکان دهد به خرید بیمه اقدام کنند. با وجود اینکه سه سال صرف ساخت آن شد، وقتی در یکم اکتبر ۲۰۱۳ آغاز به کار کرد با سر به زمین خورد.

شکل ۱-۱ وبگاه آمازون را با HealthCare.gov در سه ماه نخست کارش مقایسه می‌کند و نشان می‌دهد نه تنها گند، پرخطا و نامن بود بلکه بیشتر موقع خراب و از دسترس خارج بوده است. چرا شرکت‌هایی مثل آمازون می‌توانند نرم‌افزارهایی بسازند که می‌توانند به مشتریان بسیار بیشتری خدمات بسیار بهتری ارائه کنند؟ در حالی‌که رسانه‌ها بعدها از تصمیم‌های پرسش‌برانگیز زیادی پرده برداشتند، تعداد شگفت‌آوری از موارد اتهام به روش‌های به کار گرفته شده در تولید و

<sup>1</sup>Kayak (Travel Search Engine)

<sup>2</sup>Amazon.com

<sup>3</sup>Affordable Care Act

<sup>4</sup>Website

توسعهٔ نرم‌افزار<sup>۵</sup> مربوط بودند (جانسون و رید ۲۰۱۳). با توجه به رویکرد آن‌ها، همانطور که یک تحلیل‌گر گفت، «اگر به نتیجه می‌رسیدند جای تعجب بود.» (جانسون ۲۰۱۳<sup>۶</sup>) ما مفتخریم که این شانس را داریم تا نشان دهیم چگونه شرکت‌های اینترنتی و سایرین، خدمات نرم‌افزاری موفق ایجاد می‌کنند. همانطور که در مقدمه توضیح داده شد، این یک مسئلهٔ کمالت‌آور دانشگاهی نبیست که نتیجه‌اش فقط برای گروه محدودی اهمیت داشته باشد. پروژه‌های شکست‌خوردهٔ نرم‌افزاری می‌توانند رسوابی به بار بیاورند و حتی می‌توانند رئیس‌جمهورها را از میدان به در کنند. از طرف دیگر، پروژه‌های نرم‌افزاری موفق می‌توانند خدماتی ایجاد کنند که میلیون‌ها نفر همه‌روزه از آن‌ها بهره می‌برند و حتی سازندگان شان به نام‌هایی آشنا در ذهن همهٔ ما تبدیل شوند. همهٔ کسانی که در ایجاد این خدمات دست داشته‌اند به کارشان افتخار می‌کنند، درست بر خلاف ACA.

در ادامهٔ این فصل خواهیم دید فجایعی مانند ACA چطور اتفاق می‌افتد و چطور می‌توان از تکرار تاریخ جلوگیری کرد. سفر خود را با خاستگاه مهندسی نرم‌افزار شروع می‌کنیم که با روش‌های توسعهٔ نرم‌افزاری که تاکید زیادی بر طراحی و مستندسازی داشتند آغاز شد، زیرا که این رویکرد در سایر پروژه‌های «بزرگ» مهندسی مانند مهندسی عمران عملکرد خوبی از خود به جا گذاشته است. سپس مروری بر آمارهایی از میزان موفقیت روش‌های طرح-و-ثبت<sup>۷</sup> خواهیم داشت، و متاسفانه خواهیم دید که پروژه‌هایی که به سرنوشت و نتایجی مشابه ACA می‌رسند اتفاقاً بسیار معمول هستند، هرچند که به خوبی شناخته شده نباشند. نتایج غالباً نامیدکننده پیروی از خرد متعارف و روش‌های سنتی رایج در مهندسی نرم‌افزار، در نهایت الهام‌بخش تعدادی از توسعه‌دهندگان<sup>۸</sup> نرم‌افزار بود تا به نوعی شورش کنند. درحالی‌که منشور چاپک<sup>۹</sup> هنگام معرفی اش بسیار بحث‌برانگیز بود، اما با گذشت زمان توسعهٔ نرم‌افزاری چاپک<sup>۱۰</sup> بر منتقدان خود غلبه کرد. روش چاپک به تیم‌های کوچک امکان می‌دهد، از غول‌های صنعت، به خصوص در پروژه‌های کوچک، پیشی بگیرند. گام بعدی ما در این سفر به بررسی این مطلب اختصاص دارد که چگونه معماری سرویس‌گرا<sup>۱۱</sup> این امکان را فراهم می‌کند تا با ترکیب تعداد زیادی سرویس نرم‌افزاری کوچک، که هر یک توسط گروه‌های کوچک چاپک توسعه یافته و اداره می‌شوند، خدمات نرم‌افزاری بزرگ و موفقی همانند آمازون ارائه داد.

نکتهٔ آخر اما حیاتی اینکه، در عمل بسیار کم اتفاق می‌افتد که توسعه‌دهندگان نرم‌افزاری را از صفر تولید کنند. به‌طور معمول، ارتقا و بهبود نرم‌افزارها و مجموعه کدهای عظیم فعلی در اولویت قرار دارد. در گام بعدی سفرمان خواهیم دید که برخلاف روش طرح-و-ثبت، که هدفش طراحی کامل از قبل و سپس پیاده‌سازی است، روش چاپک تقریباً همهٔ زمانش را صرف بهبود و ارتقا نرم‌افزارها و کدهای نوشته‌شده موجود می‌کند. بنابراین می‌توان گفت با یادگیری بیشتر و بهتر شدن در روش چاپک، شما مهارت‌های لازم برای بهبود نرم‌افزارهای موجود را هم تمرين می‌کنید. برای شروع این سفر، روش تولید نرم‌افزار مورد استفاده در ساخت HealthCare.gov را مورد بررسی قرار می‌دهیم.

## ۲-۱ فرایندهای توسعهٔ نرم‌افزار: طرح-و-ثبت

اگر بناها می‌خواستند همانطور که برنامه‌نویسان برنامه می‌نویسند خانه بسازند، اولین دارکوبی که از راه می‌رسید تمدن بشر را نابود می‌کرد.

<sup>5</sup>Software Development

<sup>6</sup>Affordable Care Act

<sup>7</sup>Plan-and-Document

<sup>8</sup>Developer (Software)

<sup>9</sup>Agile Manifesto

<sup>10</sup>Agile

<sup>11</sup>Service-oriented Architecture

## قانون دوم واینبرگ، ۱۹۸۷، منسوب به جرالد واینبرگ<sup>۱۲</sup>، دانشمند رایانه در دانشگاه نیراسکا<sup>۱۳</sup>

غیرقابل پیش‌بینی بودن توسعه نرم‌افزار در اواخر دهه ۱۹۶۰ به همراه فحایعی مانند ACA تحقیقاتی در مورد توسعه نرم‌افزارهای باکیفیت در زمان‌بندی و بودجه قابل پیش‌بینی انجامید. همانند سایر رشته‌های مهندسی، عبارت **مهندسی نرم‌افزار** خلق شد (ناور و رنل ۱۹۶۹). هدف یافتن روش‌هایی برای ساخت نرم‌افزارهایی بود که همانند ساختن پل‌ها در مهندسی عمران، از نظر کیفیت، هزینه و زمان قابل پیش‌بینی باشدند.

یکی از تلاش‌های مهندسی نرم‌افزار اعمال یک روش مهندسی منظم بر روند توسعه<sup>۱۴</sup> معمولاً بی برنامه نرم‌افزار بود. بر همین اساس، تاکید بر آن بود که برای انجام پروژه قبل از شروع به برنامه‌نویسی<sup>۱۵</sup>، طرحی شامل مستندات<sup>۱۶</sup> کامل و با جزئیات وجود داشته باشد که پیشرفت پروژه را بتوان با آن بررسی و اندازه‌گیری کرد. در ادامه، هر تغییری در پروژه باید در مستندات و طرح نیز اعمال شود.

هدف همه این فرایندهای توسعه نرم‌افزار «طرح-و-ثبت» این است که با استفاده از مستندسازی گسترشده، پیش‌بینی‌پذیری را افزایش دهنده. بدیهی است هر بار اهداف پروژه تغییر کند، مستندات نیز باید متعاقباً عوض شوند. برخی نویسندهای کتاب‌های دانشگاهی این مطلب را این‌طور بیان کردند (لیتبریج و لaganier ۲۰۰۲؛ براد ۲۰۰۱):

می‌باشد از تمامی مراحل توسعه نرم‌افزار مستندات تهیه کرد، که این شامل نیازمندی‌ها، طراحی‌ها، راهنمای کاربران<sup>۱۷</sup>، دستورالعمل‌های آزمون‌گرایان<sup>۱۸</sup> و طرح‌های پروژه می‌شود.

تیموتی لیتبریج<sup>۱۹</sup> و رابت لaganier<sup>۲۰</sup>، ۲۰۰۲

مستندسازی خوبی است که در رگ‌های مهندسی نرم‌افزار جریان دارد.

براد ۲۰۰۱، اریک<sup>۲۱</sup>

(حاشیه‌های این جنبه در این کتاب با هدف ارائه توضیحات اضافی در مورد زمینه تاریخی و یا چشم‌اندازی در مورد مطلب آورده شده هستند. مطالعه این قسمت‌ها اختیاری است اما همانطور که جorges سانتایانا گفت، «کسانی که تاریخ نمی‌دانند، محاکوم به تکرار آن هستند.») شرکت فناوری گروه سی‌جی‌آی مناقصه ساخت قسمت پسین ویگاه ACA را ۹۶ برنده شد. تخمین اولیه میلیون‌دلاری آن‌ها در ادامه به ۲۹۲ میلیون دلار افزایش پیدا کرد (یکی ۲۰۱۳). همین شرکت در ساخت یک سامانه ثبت اسلحه در کانادا هم درگیر بود که در آن نیز هزینه از تخمین اولیه دو میلیون دلار به دو میلیارد دلار جهش داشت. وقتی سازمان MITRE تحقیقاتی در مورد مشکلات ویگاه ACA برای ایالت ماساچوست انجام داد، به این نتیجه رسید که شرکت فناوری گروه سی‌جی‌آی از تحریه لایم برای ساختن این ویگاه برخوردار نبوده، داده‌های را از دست داده، در آزمودن ویگاه به خوبی عمل نکرده و بهطور کلی مدیرت ضعیفی بر پروژه داشته است (بیدگود ۲۰۱۴).

این فرایند<sup>۲۲</sup> حتی به عنوان یک استاندارد رسمی مستندسازی پذیرفته شده است: استاندارد شماره IEEE/ANSI ۸۳۰/۱۹۹۳.

دولت‌هایی مانند آمریکا قوانین جامعی برای جلوگیری از فساد در خرید لوازم دارند، که منجر به نوشتن مشخصات فنی و قراردادهای تفصیلی می‌شوند. از آنجایی که هدف مهندسی نرم‌افزار این بود که توسعه نرم‌افزار را به اندازه ساختن پل‌ها قابل پیش‌بینی کند و مشخصات فنی کامل نیز داشته باشد، قراردادهای دولتی و روش توسعه نرم‌افزار طرح-و-ثبت به صورت طبیعی با هم جوش می‌خورند. بنابراین مانند بیشتر کشورها، مقررات خرید آمریکا انتخابی به جز دنبال کردن چرخه حیات<sup>۲۳</sup> طرح-و-ثبت برای توسعه دهندگان ACA باقی نگذاشتند.

قطععاً مانند سایر رشته‌های مهندسی، دولت بندهایی در قرارداد می‌گنجاند که حتی در صورت تأخیر بتواند به محصول دسترسی پیدا کند. نکته مضحک این است که هرچقدر پیمان کار در توسعه نرم‌افزار تأخیر بیشتری داشته باشد، در نهایت درآمد بیشتری هم خواهد داشت. بنابرین هنر واقعی در چانه‌زنی در مفاد قرارداد و بندهای جريمیه است. همانطور که یکی از مفسران در مورد ACA گفت (هاوارد ۲۰۱۳): «شرکت‌هایی که موفق می‌شوند این‌گونه پروژه‌ها را بگیرند، شرکت‌هایی هستند که در

12Gerald Weinberg

13University of Nebraska

14Develop (Software)

15Programming

16Documentation (Set of Documents)

17User (Computing)

18Tester

19Timothy Lethbridge

20Robert Laganiere

21Eric Braude

22Process

23Life Cycle

## فصل ۱. مقدمه‌ای بر توسعهٔ چاپک و SAAS

گرفتن پروژهٔ مهارت دارند و معمولاً<sup>۲۴</sup> نه آن‌هایی که در اجرای آن مهارت دارند.» مفسر دیگری معتقد بود روش طرح-و-ثبت برای نیازها و شیوه‌های امروزی مناسب نیست، به خصوص در شرایطی که پیمان‌کاران دولتی تمرکز خود را روی پیشینه کردن سود می‌گذارند (چونگ ۲۰۱۳). نسخه اولیه این روش توسعهٔ نرم‌افزار طرح-و-ثبت در سال ۱۹۷۰ تهیه شد (رویس ۱۹۷۰) که شامل فازهای زیر می‌شود:

۱- تحلیل و ثبت نیازمندی‌ها

۲- طراحی معماری

۳- پیاده‌سازی و یکپارچه‌سازی<sup>۲۵</sup>

۴- درستی‌سننجی

۵- بهره‌برداری و نگهداری<sup>۲۶</sup>

با این فرض که هرچه اشکالات زودتر پیدا شوند، درست کردن شان ارزان‌تر خواهد بود، فلسفهٔ این فرایند تکمیل یک فاز قبل از وارد شدن به این بعد است تا به این ترتیب بتوان اشکالات را تا جایی که امکان داشته باشد سریع‌تر از بین برد. همچنین اگر فازهای اولیه درست انجام شوند از کارهای بی‌مورد در فازهای بعد پیش‌گیری می‌شود. از آنجایی که ممکن است این فرایند سال‌ها به طول بینجامد، مستندسازی<sup>۲۷</sup> گستردگی باعث این اطمینان خاطر می‌شود که اگر یک نفر از پروژهٔ جدا شود اطلاعات مهم از بین نخواهد رفت و افراد جدید می‌توانند بعد از ملحق شدن به پروژهٔ به سرعت به روز شوند.

از آنجایی که این فرایند از بالا به پایین کامل می‌شود، فرایند توسعهٔ نرم‌افزار **مدل آبشاری**<sup>۲۸</sup> یا **چرخهٔ حیات** توسعهٔ نرم‌افزار مدل آبشاری نام گرفته است. واضح و قابل درک است که با توجه به پیچیدگی هر مرحله از چرخهٔ حیات مدل آبشاری، هر انتشار محصول یک واقعهٔ بزرگ است که مهندسان برایش عرق ریخته‌اند و با هیاهوی زیاد همراه است. در چرخهٔ حیات مدل آبشاری عمر بلند مدت نرم‌افزار توسط فاز نگهداری تایید و تصدیق می‌شود، که در آن اشکالات به محض اینکه پیدا می‌شوند، مورد اصلاح قرار می‌گیرند. نسخه‌های آلتی و جدیدتر نرم‌افزارهایی که با مدل آبشاری تهیه شده‌اند نیز باید از همین چندین فاز عبور کنند که معمولاً بین ۶ تا ۱۸ ماه زمان می‌برند.

مدل آبشاری می‌تواند در کارهای مانند پروژه‌های فضایی ناسا که از قبیل به‌طور کامل و دقیق قابل تعریف هستند موثر باشد. اما در کارهایی که نظر مشتری دربارهٔ آن چیزی که می‌خواهد ثابت نیست، به مشکل برمی‌خورد. این نقل قول از یک برندهٔ جایزهٔ تورینگ<sup>۲۹</sup> به خوبی بیانگر این نگرش است:

برای دور انداختن یکی از پیاده‌سازی‌ها آماده باشید چون بالاخره این کار را خواهید کرد.

—فرد بروکس جونیور<sup>۳۰</sup>

به عبارت دیگر برای مشتری درک اینکه واقعاً چه می‌خواهد، زمانی که نسخهٔ اولیه را دیده باشد ساده‌تر است. همچنین برای مهندسان نیز پس از اینکه برای بار اول انجامش داده باشند، فهمیدن اینکه چطور آن را بهتر بسازند آسان‌تر می‌شود.

این مشاهدات به ایجاد یک چرخهٔ حیات توسعهٔ نرم‌افزار جدید در دههٔ ۱۹۸۰ انجامید که تهیه پیش‌نمونه‌ها<sup>۳۱</sup> را با مدل آبشاری ترکیب می‌کرد (بوهم ۱۹۸۶). ایدهٔ این روش تکرار متناوب چهار فاز است که هر تکرار منجر به پیش‌نمونه‌ای خواهد شد که نسخهٔ اصلاح‌شدهٔ تکرار پیشین است.

ویندوز ۹۵ با یک جشن ۳۰۰ میلیون دلاری<sup>۳۲</sup> به صورت عمومی معرفی شد. برای این برنامه مایکروسافت کمین معرفو چی لنو را استخدام و ساختمان امپایر استیت نیویورک را با رنگ‌های نشانواره ویندوز نورپردازی کرد. مجوز استفاده از آهنگ «من را راه بیندار» گروه موسیقی رولینگ استونز نیز برای استفاده به عنوان موسیقی اصلی جشن خریداری شد.

<sup>24</sup>Integration

<sup>25</sup>Maintenance

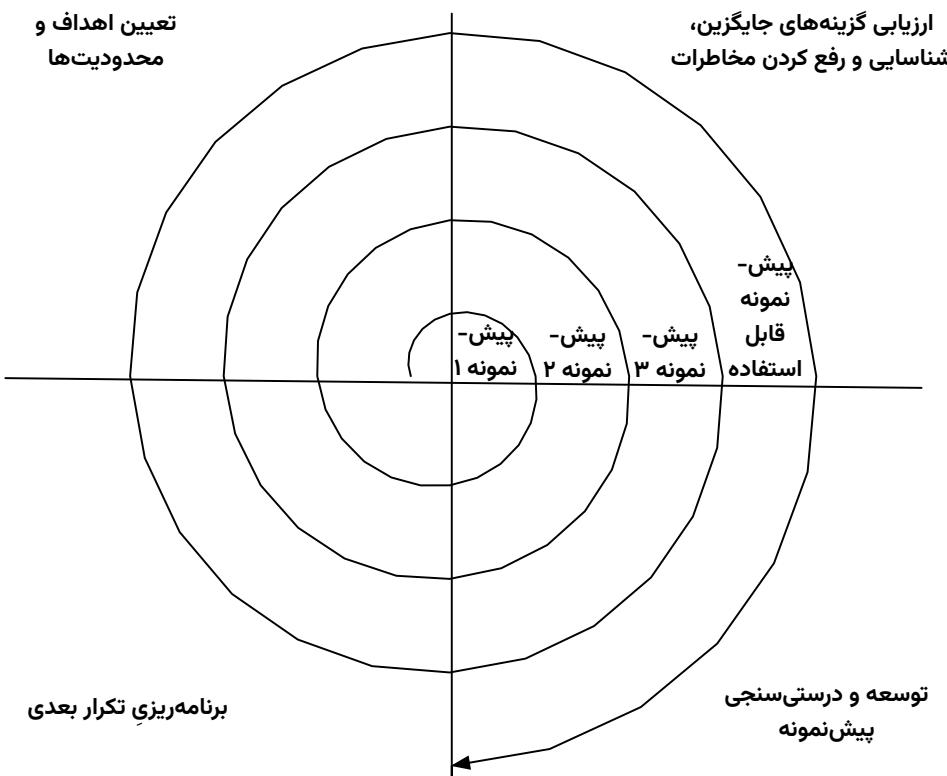
<sup>26</sup>Documentation (Act of Documenting)

<sup>27</sup>Waterfall Model

<sup>28</sup>Turing Award

<sup>29</sup>Fred Brooks, Jr.

<sup>30</sup>Prototype



شکل ۱-۲: چرخه حیات مدل مارپیچی، چرخه حیات مدل آبشاری را با تهییه پیش‌نمونه‌ها ترکیب می‌کند. از مرکز شروع و با هر تکرار دور مارپیچ از چهار فاز می‌گذرد و یک پیش‌نمونه بهبود یافته را تولید کرده و ادامه می‌دهد تا زمانی که محصول آماده انتشار شود.

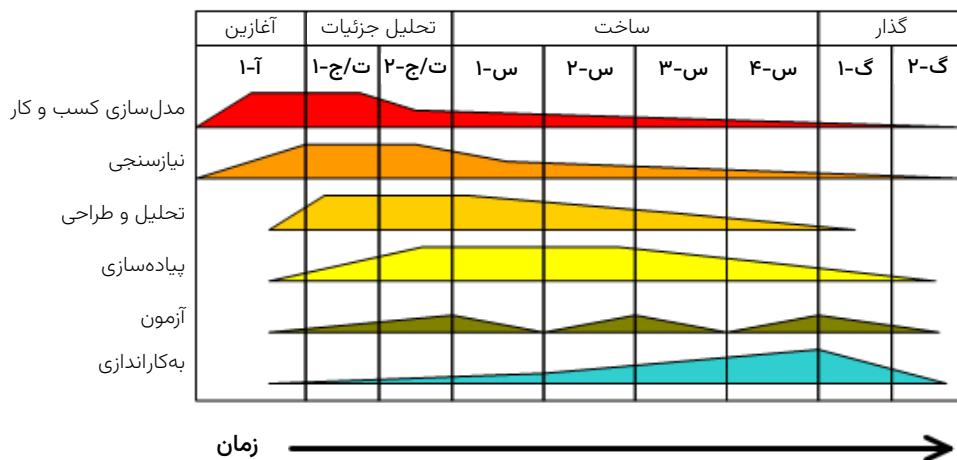
شکل ۱-۲ چهار فاز این مدل از توسعه نرم‌افزار را نمایش می‌دهد که در آن علت نام‌گذاری این روش به **مدل مارپیچی<sup>۳۱</sup>** نیز مشخص می‌شود. این فازها عبارت‌اند از:

- ۱- تعیین اهداف و محدودیت‌های این تکرار<sup>۳۲</sup>
- ۲- ارزیابی گزینه‌های جایگزین و شناسایی و رفع کردن مخاطرات
- ۳- توسعه و درستی‌سنجی پیش‌نمونه تکرار فعلی
- ۴- برنامه‌ریزی تکرار بعدی

در این مدل بهجای مستندسازی همهٔ نیازمندی‌ها در ابتدای کار، مانند مدل آبشاری، اسناد نیازمندی‌ها بر اساس نیاز در حین انجام تکرار تولید می‌شوند و همراه با پیشرفت پروژه تکامل می‌یابند. مشتری حتی قبل از تکمیل محصول در تکرارها حضور فعال دارد و این امر احتمال سوءتفاهم‌ها را می‌کاهد. با این حال، همانطور که طراحان روش از ابتدا در نظر داشتند، این تکرارها ۶ تا ۲۴ ماه طول می‌کشند و بنابراین در حین تکرار، زمان زیادی برای تغییر نظر مشتریان وجود دارد!

<sup>31</sup>Spiral Model

<sup>32</sup>Iteration



شکل ۱-۳: چرخهٔ حیات فرایند یکپارچهٔ رشنال به پروژه این امکان را می‌دهد که در هر فاز چندین تکرار داشته باشد و مشخص می‌کند که چه مهارت‌هایی در تیم اجرایی پروژه در بازه‌ها مختلف زمانی بیشتر یا کمتر مورد نیاز است. RUP سه «ضابطهٔ مکمل» هم دارد که در این شکل نشان داده نشده‌اند: مدیریت پیکربندی و تعیینات، مدیریت پروژه و محیط. (عکس از وبگاه توسط Dutchgilder).<sup>33</sup>

پس مدل مارپیچی هم بر طراحی، برنامه‌ریزی و مستندسازی گسترش دنکیه دارد اما انتظار دارد طرح با انجام هر تکرار تکامل یابد.

با توجه به اهمیت توسعهٔ نرم‌افزار، علاوه بر دو روش ذکر شدهٔ فوق، انواع متعدد دیگری از متداول‌ترین طرح-و-ثبت پیشنهاد شده‌اند. یک مورد جدید، فرایند یکپارچهٔ رشنال<sup>34</sup> (RUP) (کراچن<sup>35</sup> ۲۰۰۵)<sup>36</sup> است که در دهه ۱۹۹۰ توسعه یافته و ویژگی‌های چرخه‌های حیات مدل آبشاری و مدل مارپیچی را در هم می‌آمیزد و همچنین برای تهییهٔ مستندات و نمودارها استانداردهایی را در بر می‌گیرد. ما RUP را به عنوان نمایندهٔ جدیدترین تفکر در چرخه‌های حیات طرح-و-ثبت در نظر می‌گیریم. این روش برخلاف چرخه‌های حیات مدل آبشاری و مدل مارپیچی به مسائل کسب و کار نزدیکتر است تا مسائل فنی.

همانند مدل آبشاری و مدل مارپیچی، RUP نیز شامل فازهایی می‌شود:

۱- آغازین<sup>37</sup>: این فاز توجیه اقتصادی نرم‌افزار را تعریف می‌کند و با ارزیابی میزان گستردگی پروژه، زمان‌بندی و بودجه را تعیین می‌کند. این دو سپس برای قضاوت در مورد پیشرفت کار و توجیه هزینه‌ها مورد استفاده قرار می‌گیرند. در این فاز همچنین ارزیابی اولیه‌ای در مورد تهدیدهای واردشده نسبت به زمان‌بندی و بودجه صورت می‌گیرد.

۲- تحلیل جزئیات<sup>38</sup>: همکاری با ذی‌نفعان<sup>39</sup> به منظور تعیین موارد استفاده<sup>۴۰</sup> طراحی معماري نرم‌افزار، تعیین برنامهٔ توسعه و در نهایت ساخت پیش‌نمونهٔ اولیه.

**طراحی عظیم از ابتدا**. یا به اختصار **BDUF**، نامی است که برخی افراد برای فرایندهای مانند مدل آبشاری، مدل مارپیچی و RUP که بیان به برنامه‌ریزی، طراحی و مستندسازی گسترش دارند، به کار می‌برند. نام‌های گوناگون دیگری نیز به آن‌ها داده شده: فرایندهای سنگین وزن، طراحی محور، منظم، یا ساخت یافته.

<sup>33</sup>Rational Unified Process

<sup>34</sup>Inception (RUP phase)

<sup>35</sup>Elaboration (RUP phase)

<sup>36</sup>Stakeholder

<sup>37</sup>Use case

-۳- ساخت<sup>۳۸</sup>: نوشتن و پیاده‌سازی محصول و همچنین مورد آزمون<sup>۳۹</sup> قرار دادن آن که در نهایت به انتشار اولین نسخه خارجی می‌انجامد.

-۴- گذار<sup>۴۰</sup>: محصول را از محیط توسعه به محیط واقعی منتقل می‌کند، که شامل آزمون پذیرش<sup>۴۱</sup> توسط مشتری و همچنین آموزش کاربران می‌شود.

برخلاف مدل آبشاری هر فاز شامل تکرار می‌شود. برای مثال، ممکن است یک پروژه یک تکرار در فاز آغازین، دو تکرار در فاز تحلیل جزئیات، چهار تکرار در فاز ساخت و دو تکرار در فاز گذار داشته باشد. همچنین ممکن است مانند مدل مارپیچی، یک پروژه به طور مکرر همهٔ چهار فاز را تکرار کند. علاوه بر فازهای پروژه که به طور پویا تغییر می‌کنند، RUP<sup>۴۲</sup> شش «ضابطهٔ مهندسی» (که گردش کار<sup>۴۳</sup> هم نامیده می‌شوند) را تعریف می‌کند که روی پروژه کار می‌کنند می‌باشد آنها را به طور جمعی بوضیح دهنده:

- ۱- مدل‌سازی کسب‌وکار<sup>۴۴</sup>
- ۲- نیازسنجی
- ۳- تحلیل و طراحی
- ۴- پیاده‌سازی
- ۵- آزمون
- ۶- به‌کاراندازی<sup>۴۵</sup>

این ضوابط از این نظر که به صورت اسمی در همهٔ عمر پروژه حضور دارند، از فازها ایستاتر هستند. با این حال بعضی ضوابط بیشتر در فازهای اولیه مورد استفاده قرار می‌گیرند (مانند مدل‌سازی کسب‌وکار)، درحالی‌که برخی ضوابط متناظراً در طول پروژه (مانند آزمون) و برخی دیگر نزدیک به پایان پروژه (به‌کاراندازی) بیشتر مورد استفاده قرار می‌گیرند. شکل ۱-۳ رابطهٔ میان فازها و ضوابط را نشان می‌دهد و مساحت زیر نمودار، در آن نشان‌دهندهٔ میزان فعالیت در هر ضابطه در طول زمان است. متأسفانه یکی از معایب تدریس روش‌های طرح-و-ثبت این است که شاید دانشجویان حسن کنند توسعهٔ نرم‌افزار خسته‌کننده است (ناروکی و دیگران ۲۰۰۲؛ استلر و دیگران ۲۰۱۲). البته این عیب به اندازه‌ای نیست که باعث شود از تدریس آن چشم بپوشیم. خبر خوب اینکه روش‌های جایگزینی وجود دارند که برای بسیاری از پروژه‌ها به اندازهٔ کافی کارآمد هستند و برای تدریس در کلاس نیز مناسب‌تر هستند. در قسمت بعدی به توضیح این روش‌ها خواهیم پرداخت.

<sup>38</sup>Construction (RUP phase)

<sup>39</sup>Testing

<sup>40</sup>Transition (RUP phase)

<sup>41</sup>Acceptance Testing

<sup>42</sup>Rational Unified Process

<sup>43</sup>Workflow

<sup>44</sup>Business Modeling

<sup>45</sup>Deployment (Software)

**چکیده:** در همهٔ فرایندها یا چرخه‌های حیات توسعهٔ نرم‌افزار، **فعالیت‌های اساسی** مهندسی نرم‌افزار مشابه هستند، اما برهمنگش آن‌ها در طول زمان نسبت به انتشار محصول در بین مدل‌های مختلف متفاوت است. مشخصهٔ چرخهٔ حیات مدل آشنازی این است که بسیاری از طرح‌ریزی‌ها قبل از شروع برنامه‌نویسی انجام می‌شود، و اینکه هر فاز تنها پس از اتمام فاز قبل شروع می‌شود. چرخهٔ حیات مدل مارپیچی برای ساخت پیش‌نمونه‌ها مرتبهٔ فازهای توسعه را تکرار می‌کند، اما مانند مدل آشنازی مشتریان تنها هر ۶ تا ۲۴ ماه یکبار با پروژه درگیر می‌شوند. چرخهٔ حیات فرایند یکپارچهٔ رشنال که جدیدتر از بقیه است شامل فازها، تکرارها و پیش‌نمونه‌ها می‌شود و در عین حال مهارت‌های انسانی مورد نیاز در پروژه را تعیین می‌کند. همهٔ این چرخه‌های حیات بر برنامه‌ریزی و طراحی دقیق و مستندسازی جامع و کامل تکیه دارند و همگی پیشرفت پروژه را بر اساس طرح و برنامهٔ اولیه می‌سنجدند.

### ■ بیشتر بدانیم: مدل بلوغ قابلیت (CMM)

(برای خوانندگان کنجکاوی که می‌خواهند بیشتر در مورد آنچه پشت پرده می‌گذرد بدانند، توضیحات اضافه تحت عنوان «بیشتر بدانیم» گنجانده شده است. خوانندگان مبتدی می‌توانند در خوانش اول با خیال راحت از آن‌ها صرف‌نظر کنند، اما امیدواریم با کسب تجربه بیشتر، این قسمت‌ها برای آن‌ها جذابیت بیشتری پیدا کند!)

مؤسسهٔ نرم‌افزار در دانشگاه کارنگی ملون برای ارزیابی فرایندهای توسعهٔ نرم‌افزار که بر مبنای روش‌های طرح-و-ثبت انجام می‌شوند، **مدل بلوغ قابلیت (CMM)** (پالک و دیگران ۱۹۹۵) را ارائه کرد. با مدل‌سازی فرایندهای توسعهٔ نرم‌افزار، یک سازمان می‌تواند آن‌ها را در خود بهبود بخشد. از دید تحقیقات این مؤسسه، پنج سطح بلوغ در فرایند تولید نرم‌افزار وجود دارد:

- ۱- ابتدایی یا پر هرج و مرح: توسعهٔ نرم‌افزار بدون مستندسازی، بی‌ثبات و بدون اصول و موردي.
- ۲- تکرارپذیر: بدون دنبال کردن یک انضباط سفت و سخت اما بعضی فرایندها قابل تکرار با نتایج ثابت هستند.
- ۳- تعریف‌شده: فرایندهای استاندارد تعریف شده و همراه با مستندسازی که با گذر زمان بهبود می‌یابند.
- ۴- مدیریت‌شده: مدیریت قادر است توسعهٔ نرم‌افزار را با استفاده از معیارهای مربوط به فرایند کنترل کند، و فرایند مورد استفاده را با پروژه‌های مختلف با موفقیت تطبیق دهد.
- ۵- بهینه‌سازی: اعمال بهینه‌سازی‌های کمی و سنجیده بر فرایندها به عنوان بخشی از فرایند مدیریت.

مدل CMM به طور ضمیم سازمان‌ها را به بالا رفتن در سطح‌های توسعی می‌کند. بسیاری این مدل را به عنوان یک متداول‌وزی توسعهٔ نرم‌افزار می‌شناسند، با وجودی که به این عنوان ارائه نشده است. برای مثال، (narouki و دیگران ۲۰۰۲) سطح دوم CMM را با متداول‌وزی چاپک مقایسه می‌کنند (رجوع شود به قسمت بعدی).

### ۱-۲-۱- خودآزمایی

**یک وجه تشابه و یک تفاوت اصلی فرایندهای مانند مدل مارپیچی و RUP نسبت به مدل آشنازی چیست؟**

- ◊ همگی بر برنامه‌ریزی، طراحی و مستندسازی تکیه دارند اما فرایندهای مدل مارپیچی و RUP به جای پیمودن یک مسیر واحد طولانی تا محصول نهایی، از تکرار و پیش‌نمونه‌ها برای بهبود تدریجی محصول در طول زمان استفاده می‌کنند. ■

در پایان هر قسمت، یک یا چند پرسش آورده شده است تا از آن‌ها برای خودآزمایی استفاده کنید. ایرادی ندارد که گاهی اوقات نیاز به بازخوانی یک قسمت داشته باشید تا جواب درست به پرسش‌ها بدهید.

### ۲-۲-۱- خودآزمایی

**تفاوت‌های میان فازهای سه فرایند طرح-و-ثبت بحث شده کدام‌اند؟**

- ◊ فازهای مدل آشنازی برنامه‌ریزی (نیازمندی‌ها و طراحی معماری) را از پیاده‌سازی مجزا می‌کنند.

بعد از آن آزمون محصول صورت می‌گیرد و سپس محصول منتشر می‌شود. و در نهایت یک فاز عملیاتی مجزا قرار دارد. در مقابل، فازهای مدل ماربیچی، یک تکرار را مورد هدف قرار می‌دهند: تعیین اهداف تکرار، بررسی همه راه حل‌ها، توسعه و درستی‌سنگی این تکرار و برنامه‌ریزی برای تکرار بعدی. فازهای RUP بیشتر به اهداف کسب و کار گره خورده‌اند: فاز آغازین توجیه اقتصادی، زمان‌بندی و بودجه را تهیه می‌کند، فاز تحلیل جزئیات همکاری با مشتریان برای ساخت، پیش‌نمونه اولیه را به همراه دارد، فاز ساخت، نسخه اولیه محصول را می‌سازد و مورد آزمون قرار می‌دهد، و فاز گذار محصول را نصب و به‌کاراندازی می‌کند. ■

### ۳-۱ فرایندهای توسعه نرم‌افزار: منشور چاپ

اگر مسئله‌ای راه حل نداشته باشد، ممکن است نه یک مسئله بلکه یک واقعیت باشد، که نیاز به حل کردن ندارد، بلکه فقط لازم است با گذشت زمان با آن کنار آمد.

—شیمون پرز<sup>۴۶</sup>

#### پروژه شماره ۵۰۱ موشك

آریان<sup>۵</sup>. در چهارم زوئن ۱۹۹۶<sup>۳۷</sup> تاریخ پس از برخاستن، زمانی که یک عدد ممیز شناور به یک ناشی از تشبعات ماشین پرتو درمانی تراک-۲۵<sup>۴۸</sup>، فروپاشی مدارگرد اقیمی مریخ<sup>۴۹</sup>، و نیمه کاره رها شدن پروژه نرم‌افزار پرونده مجازی افی‌آی<sup>۵۰</sup> را شنیده‌اند که دیگر کلیشهای شده‌اند. هیچ مهندس نرم‌افزاری چنین پروژه‌هایی را در سابقه کاری خود نمی‌خواهد. حتی در مقاله‌ای لیستی با عنوان «لیست نرم‌افزارهای ننگین» ارائه شده بود که در آن از دهها پروژه نرم‌افزاری مطرح نام برده‌اند که در مجموع ۱۷ میلیارد دلار را هدر داده‌اند، درحالی‌که اکثر آن‌ها نیمه کاره رها شده بودند (چارت ۲۰۰۵).<sup>۵۱</sup>

شکل ۴-۱ چهار مطالعه بر روی پروژه‌های نرم‌افزاری را خلاصه می‌کند. تنها ۱۶٪ تا ۱۰٪ از پروژه‌ها طبق بودجه و زمان‌بندی تعیین شده پیش‌رفتند و مابقی که در اکثرب قرار دارند، متوقف یا نیمه کاره رها شده‌اند. با نگاهی دقیق‌تر به ۱۱٪ از پروژه‌های موفق‌آمیز در مطالعه (ب) همه چیز حتی نگران‌کننده‌تر به نظر می‌رسد، زیرا کمتر از ۱٪ پروژه‌های جدید طبق زمان‌بندی و بودجه تعیین شده خود پیش‌رفته‌اند. اگرچه سه مطالعه اول مربوط به بازه زمانی ۱۰ تا ۲۵ سال قبل هستند، مطالعه (ت) مربوط به سال ۲۰۱۳ است. تقریباً ۴۰٪ از این پروژه‌های بزرگ متوقف یا نیمه کاره رها شدند، و ۵۰٪ از آن‌ها با تأخیر، با صرف هزینه بیش از بودجه تعیین شده و بدون برخی از قابلیت‌های لازم به سرانجام رسیده بودند. اگر تاریخ را ملاک قرار دهیم، شانس آقای او باما برای آغاز به کار موفق رفتند. HealthCare.gov تنها یک به ده بود.

#### روش چاپ با عنوانین دیگری

مانند فرایند سبک‌وزن یا  
بی‌قاعده نیز شناخته می‌شود.

«ما با توسعه نرم‌افزار و کمک به دیگران در انجام آن، در حال کشف روش‌های بهتری برای توسعه نرم‌افزار هستیم. در حین این کار به ارزش‌های زیر رسیدیم:

• افراد و برهمنشی‌های بین آن‌ها به جای فرایندها و ابزارها

<sup>46</sup>Shimon Peres

<sup>47</sup>Ariane 5 Rocket

<sup>48</sup>Therac-25 (radiation therapy machine)

<sup>49</sup>Mars Climate Orbiter

<sup>50</sup>Virtual Case File (FBI Software)

<sup>51</sup>Agile Alliance

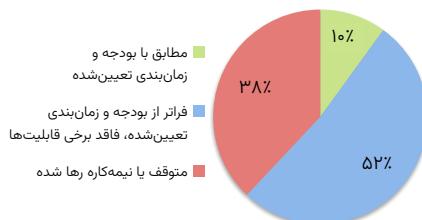
**(الف) پروژه‌های نرم‌افزاری (جانسون ۱۹۹۵)**



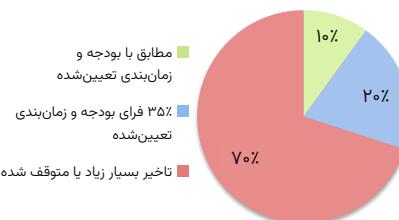
**(ب) پروژه‌های نرم‌افزاری (تیلور ۲۰۰۰)**



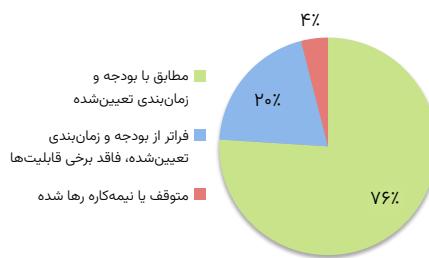
**(ت) پروژه‌های نرم‌افزاری (جانسون ۲۰۱۳)**



**(پ) پروژه‌های نرم‌افزاری (جونز ۲۰۰۴)**



**(ث) پروژه‌های نرم‌افزاری (جانسون ۲۰۱۳)**



شکل ۱-۴: (الف) مطالعه‌ای در سال ۱۹۹۵ روی پروژه‌های نرم‌افزاری نشان داد که ۵۳٪ از پروژه‌ها و زمان‌بندی تعیین شده‌شان تا سه برابر فراتر رفته‌اند و ۳۱٪ دیگر پروژه‌های نرم‌افزاری قبل از تکمیل، لغو و رها می‌شوند (جانسون ۱۹۹۵). در آمریکا هزینهٔ این پروژه‌ها ۱۰۰ میلیارد دلار تخمین زده شده است. (ب) مطالعهٔ دیگری توسط انجمن رایانهٔ بریتانیا در سال ۲۰۰۰ نشان داد که فقط ۱۳٪ عدد از ۱۰۷ پروژه طبق بودجه و زمان‌بندی شان پیش رفتند. نیمی از پروژه‌ها مربوط به تکراری و تبدیل اطلاعات گذشته بوده و نیمه دیگر مربوط به پروژه‌های ساخت سامانه‌های جدید بوده است، اما از ۱۳۰ پروژه موفق ۱۱۷ عدد از نوع اول و فقط سه عدد از نوع دوم بودند (تیلور ۲۰۰۰). (پ) مطالعهٔ دیگری بر روی ۲۵۰ پروژهٔ بزرگ، هر کدام با حجمی معادل بیش از یک میلیون خط به زبان سی، نیز نتایج نامیدکننده مشاهده داشت (جونز ۲۰۰۴). (ت) نتایج ملاتات‌آنگیز پروژه‌های «بزرگ» (دست کم ۱۰ میلیون دلار) که در این مطالعه (جانسون ۲۰۱۳) بر روی ۵۰,۰۰۰ پروژه به دست آمده نشان می‌دهد که HealthCare.gov تنها ۱٪ شناس موفقیت داشته است. (ث) اما خیر خوب اینکه پروژه‌های کوچک (با هزینهٔ کمتر از ۱ میلیون دلار) در همان مطالعه تا حد زیادی به موقع و در محدوده بودجه تعیین شده تکمیل شده‌اند، که انگیزه استفاده از روش‌های چاپک را بیشتر می‌کند.

## ۰ نرم افزاری که کار می کند به جای مستندسازی جامع

### ۰ همکاری با مشتری به جای چانه زنی در قرارداد

### ۰ واکنش به تغییرات به جای دنبال کردن برنامه

یعنی با اینکه موارد سمت چپ ارزشمند هستند، ما برای موارد سمت راست ارزش بیشتری قائلیم.»

اساس این مدل دیگر توسعه نرم افزار، پذیرفتن تغییرات به عنوان یک واقعیت زندگی است:

توسعه دهنگان باید به طور مستمر پیش نمونه های را که نافصل است اما کار می کند، بهبود بخشنند تا زمانی که مشتری از نتیجه آن راضی شود، در حالی که مشتری برای هر تکرار باز خورد می دهد. مدل چاپ برای کاهش استیاهات بر توسعه آزمون محور (TDD)<sup>۵۲</sup> تاکید دارد که در آن قبیل از شروع به نوشتن برنامه، آزمون ها<sup>۵۳</sup> نوشته می شوند. این مدل همچنین بر استفاده از روایت های کاربری<sup>۵۴</sup> برای رسیدن به توافق و تایید نیازمندی های مشتری، و معیار سرعت<sup>۵۵</sup> برای سنجش پیشرفت پروژه تاکید دارد. همه این موارد را در فصول بعدی با جزئیات برسی خواهیم کرد.

از نظر طول عمر نرم افزاری، چرخه حیات نرم افزاری چاپک آنقدر سریع است که نسخه های جدید هر هفته یا دو هفته یک بار ارائه می شوند (حتی ممکن است برقی هر روز منتشر شوند). بنابراین بر خلاف روند مدل های طرح-و-ثبت، انتشار نسخه های جدید اتفاق ویژه ای محسوب نمی شود. در این روش فرض بر بهبود مستمر در تمام طول عمر نرم افزار است.

در قسمت قبل اشاره کردیم که ممکن است فرایندهای طرح-و-ثبت برای تازه وارد ها خسته کننده و طاقت فرسا باشد، اما در مورد روش های چاپک چنین نیست. این دیدگاه در یکی از اولین نقد های روش چاپک توسط یک مدرس مهندسی نرم افزار به خوبی دیده می شود:

زمانی که برنامه نویسی لذت بخش بود را به خاطر می آورید؟ آیا به همین دلیل بود که به رایانه و بعد هم به علوم رایانه علاقه مند شدید؟ آیا دانشجویان کارشناسی ارشد ما به همین علت به این رشته می آمدند - چون دوست دارند برای رایانه برنامه نویسی کنند؟ خب، ممکن است روش های توسعه نرم افزار قابل احترام و امیدوار کننده ای وجود داشته باشند که برای این دسته از افاده آآل هستند. ... [روش چاپک] لذت بخش و موثر است، چون در آن نه تنها فرایند را در باتلاقی از انبوه مستندات گیر نمی اندازیم، بلکه توسعه دهنگان در طول فرایند توسعه با مشتریان به صورت رو در رو کار می کنند و از همان اوایل شروع کار نرم افزاری که کار می کند را تولید می کنند.

- ری مک کاول<sup>۵۶</sup>، «روش های توسعه چاپک، آماده برای بر هم زدن وضع موجود»، SIGCSE Bulletin، ۲۰۰۱،

منشور چاپک با تاکید زدایی و کاستن از اهمیت برنامه ریزی، طراحی، مستندسازی و مشخصات الزام آور قراردادی، در تضاد با خرد متعارف و نظر بزرگان حوزه مهندسی نرم افزار حرکت کرد و به همین خاطر با آگوش باز از آن استقبال نشد (کورمیک ۲۰۰۵):

[منشور چاپک] تلاش دیگری برای کم ارزش جلوه دادن رشتہ مهندسی نرم افزار است... در حرفه مهندسی نرم افزار دو گروه وجود دارند، مهندسان و هکرها<sup>۵۷</sup>... به نظر من این چیز بیشتر از تلاش برای مشروعیت بخشیدن به رفتارهای هکری نیست... حرفه مهندسی نرم افزار تنها زمانی بهبود می باید که مشتریان از پرداخت پول برای نرم افزاری که طبق مفاد قرارداد کار نمی کند، سر باز زندن... تغییر فرهنگی از فرهنگی که ذهنیت هکری را تشویق می کند به فرهنگی که بر روش های قابل پیش بینی مهندسی نرم افزار استوار شده راه تبدیل مهندسی نرم افزار به یک رشتہ آبرومند مهندسی است.

- استیون راتکین<sup>۵۸</sup>، «منشوری که بدینی برمی انگیزد»، IEEE Computer ۲۰۰۱،

<sup>52</sup> Test-Driven Development

<sup>53</sup> Test

<sup>54</sup> User Story

<sup>55</sup> Velocity (Software Development Metric)

<sup>56</sup> Renee McCauley

<sup>57</sup> Hacker

<sup>58</sup> Steven Ratkin

پرسش: پاسخ منفی روش چاپک و پاسخ مثبت استفاده از روش طرح-و-ثبت را توصیه می‌کند	
۱	آیا نیاز به تعیین مشخصات سامانه وجود دارد؟
۲	آیا مشتریان در دسترس نیستند؟
۳	آیا سامانه‌ای که ساخته می‌شود بزرگ است؟
۴	آیا سامانه‌ای که ساخته می‌شود بیچیده است (مثلاً سامانه‌ای بلادرنگ)؟
۵	آیا محصول عمر طولانی خواهد داشت؟
۶	آیا از ابزارهای نرم‌افزاری ضعیف استفاده می‌کنید؟
۷	آیا تیم پروژه از نظر جغرافیایی پراکنده‌اند؟
۸	آیا تیم در یک محیط با فرهنگ متمایل به مستندسازی قرار دارد؟
۹	آیا تیم از مهارت برنامه‌نویسی پایینی برخوردار است؟
۱۰	آیا سامانه‌ای که ساخته می‌شود مشمول قوانین و مقررات رسمی می‌شود؟

شکل ۱-۵: ده پرسش برای کمک به انتخاب استفاده از یک چرخهٔ حیات چاپک (اگر پاسخ منفی باشد) یا یک چرخهٔ حیات طرح-و-ثبت (اگر پاسخ مثبت باشد) (سامرویل ۲۰۱۰). این نکته برای ما قابل توجه است که هر وقت این پرسش‌ها را برای پروژه‌هایی که توسط تیم‌های داشتجویی انجام می‌شوند مطرح می‌کیم، تقریباً همهٔ پاسخها به سمت روش چاپک است. چنان‌که این کتاب هم نشان می‌دهد، نرم‌افزارهای مترباز ابزارهای عالی هستند که در اختیار داشتجویان قرار دارند (پرسش ۶). پرسش‌های ما از صنعت (رجوع شود به پیش‌گفتار) نشان داد که داشتجویان فارغ‌التحصیل واقع‌اگاه مهارت‌های برنامه‌نویسی خوبی دارند (پرسش ۹). پاسخ سایر هشت پرسش بهطور واضح برای پروژه‌های داشتجویی منفی است.

حتی دو نفر از منتقدین مواضع مخالفشان نسبت به روش چاپک را در قالب یک کتاب ۴۳۲ صفحه‌ای چاپ کردند! (استفنز و روزنبرگ ۲۰۰۳)

«خطوط نبرد ترسیم شده‌اند. خصومت‌ها بین اردوگاه‌های مسلح جامعه توسعهٔ نرم‌افزار شدت گرفته است. این بار بانگ صفا آرایی این است: «برنامه‌نویسی مفرط (XP)! ... چیزی که XP (دوباره) کشف کرد همان دعوای قدیمی بین جامعهٔ نرم‌افزاری است: برنامه‌نویسی در مقابل مهندسی نرم‌افزار (یا به عبارتی هکرها ژولیده در مقابل دانشمندان وارسته رایانه)».

حقوقین حوزه مهندسی نرم‌افزار به مقایسهٔ چرخه‌های حیات طرح-و-ثبت و چاپک پرداختند و در میان شگفتی افراد بدین به این نتیجه رسیدند که اتفاقاً روش چاپک بسته به شرایط می‌تواند مفید باشد. شکل ۱-۵ ده پرسشن از یک کتاب محبوب مهندسی نرم‌افزار (سامرویل ۲۰۱۰) را نشان می‌دهد که با پاسخ دادن به آن‌ها می‌توان فهمید که چه زمانی از روش‌های طرح-و-ثبت و چه زمانی از روش‌های چاپک باید استفاده کرد.

در حالی‌که شکل ۱-۴(ت) نتایج نامیدکننده پروژه‌های نرم‌افزاری بزرگ را نشان می‌دهد که از روش چاپک استفاده نمی‌کنند، شکل ۱-۴(ث) موقفيت پروژه‌های نرم‌افزاری کوچک (با هزینهٔ کمتر از ۱ میلیون دلار) را نشان می‌دهد که معمولاً از روش چاپک استفاده می‌کنند. با توجه به اینکه سه چهارم این پروژه‌ها طبق برنامهٔ زمان‌بندی و بودجه و با پیاده‌سازی همهٔ قابلیت‌ها پیش‌رفته‌اند، این نتایج با مابقی نمودارهای شکل ۱-۴ در تضاد کامل است. موقفيت پروژه‌ها تنور محبوبیت روش چاپک را داغ‌تر کرده و مطالعات و نظرسنجی‌های اخیر روش چاپک را به عنوان روش اصلی توسعه در ۶۰٪ تا ۸۰٪ تیم‌های برنامه‌نویسی در سال ۲۰۱۳ نشان می‌دهند (دفتر ایکانا میک تایمز ۲۰۱۲؛ مؤسسه مدیریت پروژه ۲۰۱۲). حتی یک مقاله به این نتیجه رسید که روش چاپک توسط اکثر تیم‌های برنامه‌نویسی که از نظر جغرافیایی پراکنده‌اند نیز مورد استفاده قرار می‌گیرد، که البته در این موارد اجرای آن چنان‌هم آسان نیست (استنلر و دیگران ۲۰۱۲).

بنابراین ما در این کتاب، در شش فصل توسعهٔ نرم‌افزار بخش دوم کتاب بر روش چاپک تمرکز خواهیم کرد. اما در هر فصل نگاهی هم به روش‌های طرح-و-ثبت در موضوعاتی مانند نیازمندی‌ها، آزمون، مدیریت پروژه و نگهداری خواهیم داشت. این تضاد و مقایسه به خوانندگان این امکان را می‌دهد تا خودشان تصمیم بگیرند که کدام روش در چه زمانی مناسب است. بخش اول این کتاب به معرفی نرم‌افزار به صورت یک سرویس<sup>۵۹</sup> و محیط‌های برنامه‌نویسی آن از جمله زبان‌های

<sup>۵۹</sup>Software as a Service

برنامه‌نویسی روی و جاوا‌اسکریپت<sup>۶۰</sup> و چارچوب روی آن ریلز می‌پردازد.

چرخهٔ حیات چابک در واقع خانواده‌ای از روش‌های مختلف است و نه یک روش واحد. ما برنامه‌نویسی مفطر<sup>۶۱</sup> (XP) را دنبال خواهیم کرد که شامل این موارد است: تکرارهای یک تا دو هفتاهای، توسعهٔ رفتارمحور<sup>۶۲</sup> (رجوع شود به فصل ۷)، توسعهٔ آزمون محور (رجوع شود به فصل ۸)، و برنامه‌نویسی دونفره<sup>۶۳</sup> (قسمت ۲-۲). **اسکرام**<sup>۶۴</sup> یک گونهٔ محبوب دیگر است (قسمت ۱۰-۱)، که در آن تیم‌های خودسازمانده<sup>۶۵</sup> از تکرارهای دو تا چهار هفته‌ای که **اسپرینت**<sup>۶۶</sup> خوانده می‌شوند، استفاده می‌کنند و سپس برای برنامه‌ریزی برای اسپرینت بعدی دوباره به دور هم جمع می‌شوند.<sup>۶۷</sup> یک ویژگی کلیدی این روش جلسه‌های سرپاپی روزانه برای برای تشخیص و برطرف کردن مواضع و مشکلات است. در حالی که چندین نقش در تیم اسکرام وجود دارد، مرسوم است که این نقش‌ها به مرور زمان به طور چرخشی میان افراد جابه‌جا شوند. روشی دیگر **کان‌بان**<sup>۶۸</sup> است که نشئت گرفته از فرایند تولید بهنگام<sup>۶۹</sup> شرکت تویوتا<sup>۷۰</sup> است که نگاهی همچون یک خط لوله<sup>۷۱</sup> یا خط تولید به توسعهٔ نرم‌افزار دارد. در این روش نفرات تیم نقش‌های ثابتی دارند و هدف این است که تعداد نفرات تیم طوری متوازن شود که هیچ گلوگاهی با کارهای روی‌هم‌ابداشت و منتظر پردازش وجود نداشته باشد. یک ویژگی معمول این روش، دیواری از کارت‌ها برای نشان دادن وضعیت همهٔ کارهای موجود در خط لوله است. چرخه‌های حیات ترکیبی نیز وجود دارند که سعی می‌کنند از هر روش نکات مثبت‌شان را بگیرند. برای نمونه **اسکرام‌بان**<sup>۷۲</sup> از جلسه‌های روزانه و اسپرینت‌های اسکرام استفاده می‌کند، اما فاز طراحی و برنامه‌ریزی را با روش پویاتر کان‌بان یعنی کنترل خط لوله‌ای دیوار کارت‌ها جایگزین می‌کند. با اینکه اکنون می‌فهمیم و یاد می‌گیریم چطور بعضی از نرم‌افزارها را با موفقیت بسازیم، اما همیشه همهٔ پروژه‌ها کوچک نیستند. در ادامه نشان خواهیم داد که چطور نرم‌افزار را طراحی کنیم که این امکان را به ما بدهد تا با ترکیب کردن قطعات کوچک‌تر سرویس‌های بزرگی همچون ویگاه آمازون ساخت.

**چکیده:** برخلاف چرخه‌های حیات طرح-و-ثبت، چرخهٔ حیات چابک با مشتریان همکاری می‌کند تا به طور مستمر امکاناتی را به پیش‌نمونه‌ای که کار می‌کند بیفزاید تا زمانی که مشتری از کار راضی شود. این کار به مشتری امکان می‌دهد در حین تکمیل پروژه هر چیزی را که می‌خواهد تغییر دهد. مستندسازی عمدتاً از طریق روایت‌های کاربری و آزمایه‌ها تولید می‌شود و ملاک پیشرفت در چارچوب یک طرح و برنامه از پیش تعریف شده نیست. بهجای آن از **سرعت** برای اندازه‌گیری پیشرفت استفاده می‌شود که در واقع برابر با آهنگ تکمیل و اضافه شدن امکانات جدید به پروژه است.

<sup>60</sup>JavaScript

<sup>61</sup>Extreme Programming

<sup>62</sup>Behavior-Driven Development

<sup>63</sup>Pair Programming

<sup>64</sup>Scrum

<sup>65</sup>Self-Organizing (Team Management)

<sup>66</sup>Sprint

<sup>67</sup>ما از لفظ «خودسازمانده» در ترجمة Self-Organizing استفاده کردیم که توصیف دقیق‌تری نسبت به لفظ متدائل‌تر «خدمنختار» است. تیم‌های خودسازمانده در چگونگی انجام کار خودمنختار هستند، اما در اینکه چه کاری انجام دهند این اختیار را ندارند. در مقابل تیم‌های خدمخنث (Autonomous) معمولاً استقلال و اختیار کاملی دارند تا تصمیم بگیرند چه کاری را انجام بدهند.

<sup>68</sup>Kanban

<sup>69</sup>Just-in-time

<sup>70</sup>Toyota (Company)

<sup>71</sup>Pipeline

<sup>72</sup>ScrumBan

### ■ بیشتر بدانیم: اصلاح مقررات خرید و تملک دولت

رئیس جمهور اوباما با تأخیر به مشکلات مربوط به خرید و تملک نرم‌افزار پی برد. او در ۱۴ نوامبر ۲۰۱۳ در یک سخنرانی گفت: «...وقتی به بازنگری در کارهای خودم می‌پردازم یکی از چیزهایی که به آن می‌رسم و می‌دانم این است که روند خرید فناوری در دولت فدرال پیچیده، دست و پاگیر و منسوخ شده است... این یکی از دلایلی است که برنامه‌های فناوری اطلاعات دولتی دائمًا از بودجه فراتر می‌روند و از برنامه زمان‌بندی عقب هستند... اکنون که می‌دانم دولت در گذشته در این مسائل خوب عمل نکرده، دو سال پیش زمانی که داشتم در مورد این مسئله فکر می‌کردیم... می‌توانستیم بیشتر تلاش و بهتر عمل کنیم تا به نوعی این سنت را بر هم بزنیم.»

در واقع، مدت‌ها قبیل از ویگاه ACA، زمزمه‌ها و درخواست‌هایی برای اصلاحات در خرید و تملک نرم‌افزار به گوش می‌رسید، از جمله در این مطالعه منتشر شده توسط آکادمی‌های ملی آمریکا در مورد وزارت دفاع این کشور:

«وزارت دفاع به مانعی یا در واقع فرهنگ و مقرراتی برخورد کرده است که برنامه‌های بزرگ، نظارت سطح بالا و یک روش توسعه و آزمون سنجیده و تربیتی (مدل آبشاری) را ترجیح می‌دهد. برنامه‌هایی که از آن‌ها انتظار ارائه راه حل‌های کامل و تقریباً بی‌نقص وجود دارد و توسعه آن‌ها سال‌ها زمان می‌برد، در وزارت دفاع بسیار معمول هستند... این رویکردها در تضاد با شیوه‌های خرید و تملک چابک هستند که در آن‌ها محصول مرکز اصلی توجه است. در رویکرد چابک، کاربران نهایی محصول از همان ابتدا و به طور مکرر در روند تکمیل برنامه درگیر می‌شوند. همچنین نظرات بر توسعهٔ تدریجی محصول به پایین‌ترین سطح عملی واگذار می‌شود و تیم مدیریت برنامه این انعطاف‌پذیری را دارد که محتوای هر مرحله از توسعهٔ تدریجی را تغییر دهد تا با زمان‌بندی تحويل کار هماهنگ باشد... رویکردهای چابک به پذیرندگان خود این امکان را داده است که از غول‌های صنعتی جافتاده و شناخته شده که با ساختارهای مدیریتی پیچیده، فرایندمحور و متعلق به دوران انقلاب صنعتی احاطه شده بودند، پیش بگیرند. رویکردهای چابک به این خاطر موفق شده‌اند که استفاده‌کنندگان آن‌ها مسائل موثر بر مخاطرات در حیطهٔ برنامه‌های فناوری اطلاعات را تشخیص داده‌اند و ساختارها و فرایندهای مدیریتی خود را برای کاهش این مخاطرات تغییر داده‌اند.» (شورای پژوهش ملی آمریکا ۲۰۱۰)

### ۱-۳-۱ خودآزمایی

درست یا نادرست: یک تفاوت بزرگ بین توسعهٔ نرم‌افزار مدل مارپیچی و چابک ساخت پیش‌نمونه‌ها و ارتباط و تعامل با مشتریان در حین انجام فرایند است.

◊ نادرست: در هر دو روش، پیش‌نمونه‌هایی که کار می‌کنند اما کامل نیستند ساخته می‌شوند و مشتری در ارزیابی آن‌ها کمک می‌کند. تفاوت آن‌ها در این است که در روش چابک مشتری حداقل هر دو هفته یک بار با کار درگیر می‌شود درحالی‌که در مدل مارپیچی ممکن است حتی تا دو سال این اتفاق نیفتد. ■

### ۲-۳-۱ خودآزمایی

درست یا نادرست: یک تفاوت بزرگ بین مدل آبشاری و روش چابک این است که در روش چابک از جمع‌آوری نیازمندی‌ها استفاده نمی‌شود.

◊ نادرست: درحالی‌که روش چابک برخلاف مدل آبشاری به مستندسازی گستردۀ نیازمندی‌ها نمی‌پردازد، ارتباط و تعامل با مشتریان منجر به ایجاد نیازمندی‌ها تحت عنوان روایت‌های کاربری می‌شود، که در فصل ۷ به آن خواهیم پرداخت. ■

### ۴-۱ تضمین کیفیت نرم‌افزار: آزمون

و کاربران پا خنده و طعنه گفتند:

«این دقیقاً همان چیزیست که ما درخواست کرده بودیم، اما نه آن چیزی که ما می‌خواهیم.»

– ناشناس

تعریفی متعارف از **کیفیت** برای هر محصولی «در خور استفاده بودن» آن است، که باید هم برای تولیدکننده و هم برای مصرف‌کننده ارزش تجاری ایجاد کند (جورن و گرینا ۱۹۹۸). در مورد نرم‌افزار، کیفیت هم به معنی برآورده کردن نیازهای مشتری (راحتی در استفاده، دریافت جواب‌های صحیح، عدم خروج ناگهانی و غیره) و **همچنین ساده و راحت بودن بهبود و اشکال‌زدایی**<sup>۷۳</sup> آن برای توسعه‌دهندگان است. ریشهٔ **تصمیم‌گیری کیفیت** (QA)<sup>۷۴</sup> نیز به صنعت تولید برمی‌گردد، و به فرایندهای معیارهایی که منجر به تولید محصولات با کیفیت بالا می‌شوند، و همچنین به ایجاد فرایندهای تولیدی که کیفیت را بالا می‌برند، اشاره دارد. پس تصمیم‌گیری نرم‌افزار، به معنی حصول اطمینان از کیفیت بالای محصول درحال تولید و همچنین ایجاد فرایندها و استانداردهایی در یک سازمان است که به تولید محصولات با کیفیت بالا منجر می‌شود. همانطور که در ادامه خواهیم دید، در برخی از فرایندهای نرم‌افزاری طرح-و-ثبت حتی از یک تیم جداگانه QA برای بررسی و آزمودن کیفیت نرم‌افزار استفاده می‌کنند (قسمت ۱۰-۸).

فرایندهای تعیین کیفیت نرم‌افزار شامل دو اصطلاح می‌شود که معمولاً به جای یکدیگر به کار می‌روند: اما تفاوت ظرفی دارند (بوهم ۱۹۷۹):

- **درستی‌سنجد**<sup>۷۵</sup>: آیا محصول را درست ساخته‌اید؟ (آیا محصول، طبق مشخصات<sup>۷۶</sup> از قبیل تعریف شده ساخته شده؟)

- **اعتبارسنجی**<sup>۷۷</sup>: آیا محصول درست را ساخته‌اید؟ (آیا این همان چیزی است که مشتری می‌خواهد؟ یا به عبارت دیگر، آیا مشخصات تعریف شده با خواستهٔ مشتری تطابق دارد؟)

پیش‌نمونه‌های نرم‌افزار که جوهرهٔ روش چاپک هستند، عموماً بیشتر به اعتبارسنجی کمک می‌کنند تا به درستی‌سنجد، چون بسیاری از مشتریان با دیدن کارکرد محصول در مورد چیزی که می‌خواهند تغییر عقیده می‌دهند.

طریقهٔ اصلی درستی‌سنجد و اعتبارسنجی، **آزمون** است. انگیزهٔ اصلی انجام آزمون یافتن هرچه زودتر اشتباهات توسط توسعه‌دهندگان است تا هزینهٔ اصلاح کردن آن‌ها کاهش یابد. با توجه به فراوانی ترکیب‌های مختلف ورودی، انجام آزمون به صورت جامع ممکن نیست. یک راه برای کاهش فضای ممکن ورودی، انجام آزمون‌های مختلف در فازهای مختلف توسعهٔ نرم‌افزار است. بهترتب از سطح پایین به بالا، ابتدا **آزمون واحد**<sup>۷۸</sup> قرار می‌گیرد. این نوع آزمون به ما این اطمینان را می‌دهد که یک زویه<sup>۷۹</sup> یا تابع<sup>۸۰</sup> واحد، کاری را انجام می‌دهد که از آن انتظار می‌رود. سطح بعد **آزمون پودمان**<sup>۸۱</sup> است که تک‌تک واحدها را به صورت سراسری مورد آزمون قرار می‌دهد. به طور مثال، آزمون واحد در محدودهٔ یک کلاس<sup>۸۲</sup> واحد کار می‌کند، اما آزمون پودمان تمام کلاس‌ها را بررسی می‌کند. بر روی این سطح، **آزمون یکپارچگی**<sup>۸۳</sup> قرار دارد، که تصمیم می‌کند واسطه‌های بین واحدها دارای فرضیات ثابت و سازگار با یکدیگر هستند و آن‌ها به درستی باهم ارتباط برقرار می‌کنند. این سطح، به بررسی عملکرد واحدها نمی‌پردازد. در بالاترین سطح، **آزمون سامانه**<sup>۸۴</sup> یا **آزمون پذیرش** قرار دارد که وظیفهٔ بررسی برنامه کامل و یکپارچه (که از ترکیب واحدهای مختلف تشکیل شده است) را از جهت دارا بودن مشخصات تعیین شده بر عهده دارد. در فصل ۸، جایگزینی را برای آزمون معرفی می‌کنیم که **روش‌های صوری**<sup>۸۵</sup> نام دارد.

<sup>73</sup> Debugging

<sup>74</sup> Quality Assurance (QA)

<sup>75</sup> Verification

<sup>76</sup> Specification

<sup>77</sup> Validation

<sup>78</sup> Unit Testing

<sup>79</sup> Procedure (Programming)

<sup>80</sup> Function (Programming)

<sup>81</sup> Module Testing

<sup>82</sup> Class (Computer Programming)

<sup>83</sup> Integration Testing

<sup>84</sup> System Testing

<sup>85</sup> Formal Methods

**غیر عملی بودن آزمون جامع.**  
فرض کنید که آزمودن یک برنامه یک ناونوایی طول می‌کشد و این برنامه فقط یک ورودی<sup>۸۶</sup> دارد که ما می‌خواهیم آن را به طور کامل و جامع مورد آزمون قرار دهیم. (بدهی است که اکثر برنامه‌ها زمان بیشتری برای اجرا نیاز دارند و ورودی‌های آن‌ها نیز بیشتر است.) فقط همین مورد ساده<sup>۸۷</sup> ناونوایی یا ۵۰۰ سال طول خواهد کشید.

همانطور که به طور خلاصه در قسمت ۱-۳ ذکر شد، انجام آزمون در گونهٔ XP<sup>۸۶</sup> از روش چاپک با نوشتمن آزمون‌ها قبل از نوشتمن کد انجام می‌پذیرد. پس از این کار باید با نوشتمن کمترین میزان کد، آزمون را با موفقیت پشت سر گذاشت. این کار تضمین می‌کند که زین‌پس کد شما همواره مورد آزمون قرار گیرد، و احتمال نوشتمن کدی را که بعداً دور ریخته می‌شود، کاهش می‌دهد. روش XP فلسفهٔ انجام آزمون از همان ابتدا را بسته به سطح آزمون به دو بخش تقسیم می‌کند. در روش XP برای آزمون سامانه، آزمون پذیرش و آزمون یکپارچگی، از توسعهٔ رفتارمحور (BDD<sup>۸۷</sup>) استفاده می‌شود، که موضوع فصل ۷ این کتاب است. این روش همچنین برای آزمون واحد و آزمون پودمان از توسعهٔ آزمون محور (TDD<sup>۸۸</sup>) استفاده می‌کند، که موضوع فصل ۸ است.

**چکیده:** انجام آزمون مخاطرات ناشی از اشتباهات در طراحی را کاهش می‌دهد.

- انجام آزمون با گونه‌های مختلفش به درستی‌سنجدی یک نرم‌افزار در داشتن مشخصات تعیین‌شده و همچنین اعتبارسنجدی طراحی آن مطابق با خواسته‌های مشتری کمک می‌کند.
- با تقسیم آزمون به آزمون واحد، آزمون پودمان، آزمون یکپارچگی و آزمون سامانه یا آزمون پذیرش در جهت غلبه بر غیرعملی بودن انجام یک آزمون جامع تلاش می‌شود. هر آزمون سطح بالاتر، انجام آزمون‌هایی دقیق‌تر و جزئی‌تر را به سطوح پایین‌تر واگذار می‌کند.
- روش توسعهٔ چاپک با استفاده از نوشتمن آزمون‌ها پیش از نوشتمن کد و انجام توسعهٔ رفتارمحور و توسعهٔ آزمون محور بسته به سطح آزمون، بر سختی‌های آزمون غلبه می‌کند.

### ■ بیشتر بدانیم: انجام آزمون: روش‌های طرح-و-ثبت در مقابل چاپک

در فرایند توسعه مدل آبشاری، انجام آزمون پس از پایان هر فاز و همچنین در یک فاز پایانی درستی‌سنجدی که شامل آزمون‌های پذیرش می‌شود، صورت می‌گیرد. در مدل ماربیچی، این اتفاق پس از هر تکرار می‌افتد که می‌تواند یک تا دو سال به طول بینجامد. در گونهٔ XP از روش چاپک، ضمانت از طریق توسعهٔ آزمون محور انجام می‌گیرد که در آن آزمون‌ها قبل از نوشتمن کد نوشتمنه می‌شوند. این دیدن معنی است که حتی اگر کدی از قبل وجود داشته باشد و بنا بر ارتقا و بهبود کد موجود باشد، نوشتمن آزمون‌ها باید پیش از نوشتمن کد مربوط به ارتقا صورت پذیرد. حجم و میزان گستردگی آزمون به این بستگی دارد که آیا شما در حال ارتقای کدی تمیز (زیباگد) هستید و یا کدی که در واقع یک میراث کد است. مسلمًاً یک میراث کد به آزمون بیشتری نیاز دارد.

### ۱-۴-۱ خودآزمایی

می‌دانیم که تمامی روش‌های آزمون زیر در درستی‌سنجدی به ما کمک می‌کنند؛ اما کدام مورد بیش از دیگران به اعتبارسنجدی کمک می‌کند: آزمون واحد، آزمون پودمان، آزمون یکپارچگی و یا آزمون پذیرش؟

◦ اعتبارسنجدی به آنچه مشتری واقعاً می‌خواهد می‌پردازد تا اینکه در مورد درست بودن کد و مشخصات آن به بحث پردازد؛ بنابراین آزمون پذیرش بیش از همهٔ موارد می‌تواند تفاوت بین «کار را درست انجام دادن» و «کار درست را انجام دادن» را تعیین کند. ■

<sup>86</sup>Extreme Programming

<sup>87</sup>Behavior-Driven Development

<sup>88</sup>Test-Driven Development

## ۵-۱ بهره‌وری: اختصار، سنتز، بازکاربرد و ابزارها

بر اساس قانون مور<sup>۸۹</sup> منابع سخت‌افزاری نزدیک به ۵۰ سال است که در هر ۱۸ ماه دو برابر می‌شوند. این یعنی رایانه‌هایی سریع‌تر با حافظه‌های بسیار بزرگ‌تر که قادر به اجرای برنامه‌های بسیار بزرگ‌تری هستند. برای ساخت نرم‌افزارهای کاربردی<sup>۹۰</sup> بزرگ‌تری که بتوانند از این رایانه‌های قدرتمندتر حداقل بهره را ببرند، مهندسان نرم‌افزار می‌باشند بهره‌وری خود را افزایش دهند.

مهندسان چهار سازوکار اساسی را برای پهبود بهره‌وری خود ایجاد کردند:

۱- وضوح به کمک اختصار

۲- سنتز

۳- بازکاربرد

۴- خودکارسازی به کمک ابزارها

**وضوح<sup>۹۱</sup> به کمک اختصار**<sup>۹۲</sup> منعکس کننده یکی از پیش‌فرض‌های محرك و اصلی بهبود بهره‌وری برنامه‌نویسان است: هر چه کد برنامه راحت‌تر فهمیده شود و خواناتر باشد، اشکالات<sup>۹۳</sup> کمتری خواهد داشت و نگهداری و تکامل آن آسان‌تر است. در راستای همین اصل، می‌توان به این نتیجه هم رسید که هرچه برنامه کوچک‌تر باشد، معمولاً فهم آن نیز آسان‌تر است. ما این مفهوم را با شعار «وضوح به کمک اختصار» بیان می‌کنیم.

زبان‌های برنامه‌نویسی برای رسیدن به این هدف، دو روش را در پیش می‌گیرند. در روش اول، دستورات و ساختارهایی را در اختیار برنامه‌نویس قرار می‌دهند تا بتواند ایده‌هایش را به صورت<sup>۹۴</sup> طبیعی و با حروف یا کاراکترهای کمتری بنویسد. برای مثال هر دو خط زیر یک مفهوم باشند<sup>۹۵</sup>:

- assert\_greater\_than\_or\_equal\_to(a, b)
- expect(a).to be >= b

در خط اول نه تنها ممکن است برای لحظه‌ای ترتیب آرگومان‌های ورودی باعث سردرگمی شود، بلکه خواندن عبارتی طولانی‌تر که تعداد حروفش دو برابر است، بار ادراکی مضاعفی را برای خواننده ایجاد می‌کند. خط دوم اما (که اتفاقاً به زبان رویی است) کوتاه‌تر، خواناتر و قابل فهم‌تر است، که این ویژگی‌ها نگهداری آن را نیز آسان‌تر می‌کند.

روش دیگر برای افزایش وضوح، بالا بردن سطح تجرید<sup>۹۶</sup> است. این روش در ابتدا به تولید زبان‌های برنامه‌نویسی سطح بالاتری مانند فورترن<sup>۹۷</sup> و کوبال<sup>۹۸</sup> منجر شد. ظهور این زبان‌ها باعث ارتقای سطح تولید و مهندسی نرم‌افزار از زبان اس‌میلی<sup>۹۹</sup> برای یک رایانه خاص، به زبان‌های سطح بالایی شد، که می‌توانستند تنها با عوض کردن کامپایلر<sup>۹۹</sup> بر روی رایانه‌های مختلف اجرا شوند. با ادامه روند رو به رشد عملکرد ساخت‌افزار، تعداد بیشتری از برنامه‌نویسان مشتاق بودند تا کارهایی را که قبل از خودشان انجام می‌دادند به کامپایلرها و سامانه‌های زمان اجرا<sup>۱۰۰</sup> واگذار کنند.

<sup>89</sup>Moore's Law

<sup>90</sup>Application (Software/Program)

<sup>91</sup>Clarity

<sup>92</sup>Conciseness

<sup>93</sup>Bug (Computer)

<sup>94</sup>Assertion (Computer Programming)

<sup>95</sup>Abstraction (Computer Science)

<sup>96</sup>Fortran (Programming Language)

<sup>97</sup>COBOL (Programming Language)

<sup>98</sup>Assembly Language

<sup>99</sup>Compiler

<sup>100</sup>Runtime System

به‌طور مثال جاوا<sup>۱۰۱</sup> و زبان‌های برنامه‌نویسی مشابه دیگری، مدیریت حافظه را خود به‌دست گرفتند؛ ویژگی‌ای که در زبان‌های قدیمی‌تری مانند سی<sup>۱۰۲</sup> و سی‌پلاس‌پلاس<sup>۱۰۳</sup> بر عهدهٔ برنامه‌نویسی گذاشته شده‌بود. زبان‌های اسکریپتنویسی<sup>۱۰۴</sup> مانند پایتون<sup>۱۰۵</sup> و روئی سطح تجربید را از این هم بالاتر بردند. از نمونه‌هایی از این سطح تجربید بالاتر می‌توان به مواردی مانند **بازتاب**<sup>۱۰۶</sup>، که به برنامه اجازه می‌دهد تا خودش را تحت نظر بگیرد و یا **توابع مرتبهٔ بالاتر**<sup>۱۰۷</sup> نام برد، که امکان بازکاربرد رفتارهای سطح بالاتر را با ارسال توابع به عنوان آرگومان<sup>۱۰۸</sup> به توابع دیگر فراهم می‌کند. این سطح بالاتر از تجربید برنامه‌ها را مختص‌تر و در نتیجه (به‌طور معمول) خواندن، درک و نگهداری آن‌ها را آسان‌تر کرد. ما برای نشان دادن و برجسته کردن مثال‌هایی که بهره‌وری را به کمک اختصار بالا می‌برند از این نماد «اختصار» استفاده می‌کنیم.



### سنتر<sup>۱۰۹</sup> به دسته راهکارهایی برای افزایش بهره‌وری اشاره دارد که در آن‌ها کد به‌جای اینکه

به صورت دستی نوشته و به صورت خودکار تولید شود. سنتر مدارهای منطقی برای مهندسین سخت‌افزار به این معناست که آن‌ها بتوانند سخت‌افزار را به صورت توابع بولی توصیف کنند و در ازای آن ترانزیستورهایی بهینه شده که توابع مورد نظر را پیاده‌سازی کرده‌اند، تحويل بگیرند. مثال کلاسیک در سنتر نرم‌افزاری عملیات **بیت‌بلیت**<sup>۱۱۰</sup> است که یک دستور گرافیکی برای ترکیب دو بیت‌مپ<sup>۱۱۱</sup> با استفاده از یک ماسک به شمار می‌رود. روش ابتدایی حل این مسئله، یک عبارت شرطی برای انتخاب نوع ماسک در درونی ترین حلقه داشت، که البته روش کندی بود. اما راه حل بهتر، نوشتن برنامه‌ای بود که می‌توانست کد خاص‌منظورهای را بدون داشتن عبارت شرطی داخل حلقه سنتر کند. ما برای مشخص کردن مثال‌هایی که با تولید خودکار کد باعث افزایش بهره‌وری می‌شوند، از این نماد چرخ‌نده‌های «تولید کد» استفاده می‌کنیم. چارچوب رویی آن ریلز از امکانات زبان برنامه‌نویسی رویی برای **فرابرنامه‌نویسی**<sup>۱۱۲</sup> به صورت گسترده استفاده می‌کند، که این امکان را به برنامه‌های رویی می‌دهد تا به‌طور خودکار در زمان اجرای کد سنتر کند.



**بازکاربرد**<sup>۱۱۳</sup> و یا استفادهٔ مجدد از قسمت‌هایی که پیش‌تر طراحی شده‌اند به‌جای نوشتن همه‌چیز از صفر، سومین راهکار افزایش بهره‌وری است. از آنجایی که اعمال تغییرات کوچک در نرم‌افزار آسان‌تر از سخت‌افزار است، امکان استفادهٔ مجدد از قسمت‌هایی پیشین که شاید حتی کاملاً هم مناسب کار فعلی نباشند، در نرم‌افزار بیشتر از سخت‌افزار است. ما مثال‌هایی را که برای افزایش بهره‌وری از بازکاربرد استفاده می‌کنند، با این نماد «بازکاربرد» که شبیه نماد بازیافت است، مشخص می‌کنیم.



رویه‌ها و توابع برنامه‌نویسی در اولین روزهای تولید نرم‌افزار با این هدف ابداع شدند تا بخش‌های مختلف یک برنامه بتوانند از یک کد یکسان با پارامترهای متفاوت استفادهٔ مجدد کنند. به دنبال آن، کتابخانه‌های استانداردی برای ورودی/خروجی<sup>۱۱۴</sup> و توابع ریاضی به وجود آمدند تا برنامه‌نویسان بتوانند کدهایی را که افراد دیگر توسعه داده‌اند، مورد بازکاربرد قرار دهند.

رویه‌هایی که در کتابخانه‌ها<sup>۱۱۵</sup> وجود دارند، امکان استفادهٔ مجدد از پیاده‌سازی کارهای مشخصی را به وجود می‌آورند. ولی بیشتر برنامه‌نویسان می‌خواهند مجموعه‌ای **گرددآوری** شده از کارها را مدیریت و مورد استفادهٔ مجدد قرار دهند. از این رو، قدم بعدی در بازکاربرد پذیری نرم‌افزار، **برنامه‌نویسی**

<sup>101</sup>Java (Programming Language)

<sup>102</sup>C (Programming Language)

<sup>103</sup>C++ (Programming Language)

<sup>104</sup>Scripting Language

<sup>105</sup>Python (Programming Language)

<sup>106</sup>Reflection (Computer Programming)

<sup>107</sup>Higher-order Function

<sup>108</sup>Argument (Programming)

<sup>109</sup>Synthesis

<sup>110</sup>Bit Blit

<sup>111</sup>Bitmap

<sup>112</sup>Metaprogramming

<sup>113</sup>Reuse

<sup>114</sup>Input/Output (I/O)

<sup>115</sup>Library (Computing)

**شی‌عگرا<sup>۱۱۶</sup>** بود. با استفاده از امکان وراثت<sup>۱۱۷</sup> در زبان‌های مانند سی‌پلاس‌پلاس و جاوا می‌توان کارهای یکسانی را بر روی اشیای مختلف انجام داد و از پیاده‌سازی آن کارها استفاده مجدد کرد. اگرچه وراثت امکان استفاده مجدد از پیاده‌سازی‌ها را فراهم می‌کرد، وجود یک روش و یا الگوی کلی برای انجام کارهایی که حتی پیاده‌سازی‌شان متفاوت بود نیز فرصتی مناسب برای بازکاربرد ایجاد می‌کرد. این‌جا بود که **الگوهای طراحی<sup>۱۱۸</sup>**، با الهام از معماری ساختمانی (الکساندر و دیگران ۱۹۷۷)، سر برآوردند تا پاسخگوی این نیاز باشند. زبان‌های برنامه‌نویسی برای پشتیبانی بهتر از بازکاربرد الگوهای طراحی امکاناتی را به خود افزودند. **نوع‌دهی پویا<sup>۱۱۹</sup>** یکی از این امکانات بود که ترکیب و درهم آمیختن تجربیدها را آسان می‌ساخت. **میکس‌این‌ها<sup>۱۲۰</sup>** دیگر امکانی بودند که استفاده از کارکرد چندین مُنْد<sup>۱۲۱</sup> را بدون نیاز به وراثت چندگانه و دردسراهیش، فراهم می‌ساختند. پایتون و روبي از نمونه زبان‌هایی هستند که امکانات را بازکاربرد الگوهای طراحی فراهم می‌کنند.

توجه داشته باشید که بازکاربرد به این معنی نیست که شما تکه‌کدی را کپی کنید و آن را عیناً در چندین محل قرار دهید. مشکل کپی کردن این است که در هنگام رفع یک اشکال و یا افزودن یک ویژگی جدید ممکن است همهٔ نسخه‌های کپی شده را بهروز نکنید. یک رهنمود نامآشنا در مهندسی نرم‌افزار که از این‌گونه تکرارها جلوگیری می‌کند این است که:

برای هر قسمی از اطلاعات موجود در یک سامانه باید تنها یک نمایش واحد، بدون ابهام و معتبر وجود داشته باشد.

—آندي هانت<sup>۱۲۲</sup> و ديو توماس<sup>۱۲۳</sup>، ۱۹۹۹



این رهنمود در این شعار خلاصه شده است: **خودت را تکرار مکن<sup>۱۲۴</sup>**. در این کتاب ما از لفظ DRY و نماد حوله<sup>۱۲۵</sup> برای نشان دادن مثال‌هایی که از این قاعده پیروی می‌کنند استفاده می‌کنیم. روبي و جاوا‌اسکریپت که در این کتاب ما به آن‌ها خواهیم پرداخت و از آن‌ها استفاده می‌کنیم، از زبان‌های اسکریپتنویسی مدرن معمول هستند که از مدیریت حافظه خودکار، نوع‌دهی پویا، پشتیبانی از توابع مرتبهٔ بالاتر و سایر مکانیسم‌های مختلف برای افزایش استفاده مجدد از کد بهره می‌برند. روبي با دربرگرفتن پیشترفت‌های مهم در زبان‌های برنامه‌نویسی پا را فراتر از زبان‌هایی مانند پرل<sup>۱۲۶</sup> در پشتیبانی از پارادایم‌های برنامه‌نویسی<sup>۱۲۷</sup> متعدد مانند برنامه‌نویسی شی‌عگرا و برنامه‌نویسی تابعی<sup>۱۲۸</sup> می‌گذارد.

**خودکارسازی<sup>۱۲۹</sup>**، چهارمین و آخرین راهکارمان در بهبود بهره‌وری است که نمایانگر یکی از کارکردها و ارزش‌های اساسی مهندسی نرم‌افزار است: یافتن راههایی برای جایگزین کردن کارهای خسته‌کننده دستی با ابزارها است. این روند به صرفه‌جویی در زمان، افزایش دقت و یا هردوی این‌ها کمک می‌کند. بدیهی‌ترین نمونه‌های این‌گونه ابزارها برای توسعهٔ نرم‌افزار، کامپایلرها و مفسرها<sup>۱۳۰</sup> هستند که همانطور که پیش‌تر اشاره شد، سطح تجرید را بالا می‌برند و کد نیز تولید می‌کنند. اما ابزارهای

<sup>116</sup> Object-oriented (Programming)

<sup>117</sup> Inheritance (OOP)

<sup>118</sup> Design Pattern

<sup>119</sup> Dynamic Typing (Type Checking)

<sup>120</sup> Mix-in (OOP)

<sup>121</sup> Method (OOP)

<sup>122</sup> Andy Hunt

<sup>123</sup> Dave Thomas

<sup>124</sup> Don't Repeat Yourself (DRY)

<sup>125</sup> از آنجایی که کلمه DRY کوتاه‌نوشت معادل انگلیسی عبارت «خودت را تکرار مکن» است و از نظر لغوی به معنای خشک می‌باشد، نماد حوله برای آن انتخاب شده است.

<sup>126</sup> Perl (Programming Language)

<sup>127</sup> Programming Paradigm

<sup>128</sup> Functional Programming

<sup>129</sup> Automation

<sup>130</sup> Interpreter

## فصل ۱. مقدمه‌ای بر توسعه چاپک و SAAS

بهره‌وری ظرفیتی مانند میکفایل‌ها<sup>۱۳۱</sup> و سامانه‌های کنترل نسخه<sup>۱۳۲</sup> (قسمت ۲-۱۰) نیز وجود دارند که کارهای خسته‌کننده را به صورت خودکار انجام می‌دهند. ما مثال‌های مربوط به ابزارها را با این نماد چکش مشخص می‌کنیم.

یکی از نکات مهم، زمان مورد نیاز برای یاد گرفتن طریقه استفاده از یک ابزار جدید در مقابل میزان صرفه‌جویی در وقت استفاده از آن است. قابل اعتماد بودن ابزار، کیفیت تجربه کاربری<sup>۱۳۳</sup> و نحوه تصمیم‌گیری در انتخاب یک ابزار از میان چندین گزینه، از دیگر موارد حائز اهمیت هستند که می‌توان به آن‌ها اشاره کرد. با وجود این، یکی از ابزارهای بنیادی مهندسی نرم‌افزار این است که یک ابزار جدید می‌تواند زندگی ما را بهتر کند.

نویسنده‌گان این کتاب نیز به ارزش خودکارسازی و ابزارها واقف هستند و از آن‌ها استقبال می‌کنند. به همین دلیل در طول این کتاب شما را با چندین ابزار آشنا خواهیم کرد تا بهره‌وری‌تان را افزایش دهیم. خبر خوب اینکه هر ابزاری که در این کتاب به شما معرفی می‌شود، به‌دقت مورد بررسی قرار گرفته است تا قابل اطمینان باشد. همچنین زمانی را که صرف آموختن آن می‌کنید، در ادامه با صرفه‌جویی‌ای که در زمان توسعه ایجاد می‌شود و همچنین افزایش کیفیتی که به همراه خواهد داشت، چندین برابر جبران خواهد شد. برای مثال در فصل ۷، نشان می‌دهیم که چطور **کیوکامبر**<sup>۱۳۴</sup> به صورت خودکار روایت‌های کاربری را به آزمون‌های پذیرش تبدیل می‌کند و همچنین اینکه چطور **پیووتال ترکر**<sup>۱۳۵</sup> به طور خودکار «سرعت» را که مقیاسی برای نرخ افزوده شدن قابلیت‌ها به نرم‌افزار است، محاسبه می‌کند. در فصل ۸، به معرفی آر-اسپ<sup>۱۳۶</sup> می‌پردازیم که به خودکارسازی فرایند آزمون واحد کمک می‌کند. و اما خبر بد اینکه شما باید کار با چندین ابزار جدید را یاد بگیرید، که البته ما فکر می‌کنیم توانایی فرآگیری سریع ابزارها یک نیاز برای موقوفیت در مهندسی نرم‌افزار است؛ بنابراین این مهارت خوبی برای سرمایه‌گذاری و پرورش دادن است.

پس چهارمین راهکار برای افزایش بهره‌وری، خودکارسازی کارها به کمک ابزارهای می‌کنند. ما مثال‌هایی را که از خودکارسازی استفاده می‌کنند با نماد ریات نمایش می‌دهیم؛ اگرچه که بیشتر این مثال‌ها به ابزارها نیز مرتبط‌اند.



### یادگیری ابزارهای جدید

آیه ۴ از فصل ۱۴ کتاب مقدس نسخه شاه جیمز درباره افزایش بهره‌وری با صرف زمان برای یادگیری و استفاده از ابزارها صحبت می‌کند: «جالی که گاوی نیاشد، آخر تمیز است؛ اما محصول فراوان به نیروی گاو بدست می‌آید.»



**چکیده:** قانون مور به مهندسین نرم‌افزار الهام بخشید تا بهره‌وری‌شان را به کمک موارد زیر افزایش دهند:

- میل به اختصار، با استفاده از دستورات خلاصه‌تر و با بالا بردن سطح تجرید به کمک زبان‌های برنامه‌نویسی سطح بالاتر. مثال‌هایی در این زمینه شامل **باتاب** می‌شوند که به برنامه اجازه می‌دهد تا خودش را تحت نظر بگیرد، و **توابع مرتبه بالاتر**، که امکان بازکاربرد رفتارهای سطح بالاتر را با ارسال توابع به عنوان آرگومان به توابع دیگر فراهم می‌کند.
- سنتز و تولید خودکار کد پیاده‌سازی.
- بازکاربرد و استفاده مجدد طراحی‌ها با پیروی از اصل **خودت را تکرار مکن (DRY)** و با تکیه بر نوآوری‌هایی مانند روش‌های کتابخانه‌ها، برنامه‌نویسی شیءگرا و الگوهای طراحی که بازکاربرد را آسان‌تر می‌کنند.
- استفاده (و ابداع) ابزارهایی برای خودکارسازی کارهای خسته‌کننده.

<sup>131</sup>Makefile

<sup>132</sup>Version Control

<sup>133</sup>User Experience (UX)

<sup>134</sup>Cucumber (Software tool)

<sup>135</sup>Pivotal Tracker

<sup>136</sup>RSpec (Testing tool)

### ■ بیشتر بدانیم: بهرهوری: چرخه حیات طرح-و-ثبت در مقابل چاپ

بهرهوری با شاخص مهندس-ساعت برای پیاده‌سازی یک قابلیت جدید سنجیده می‌شود. تفاوت در این است که چرخه‌ها در مدل آبشاری و مدل مارپیچی به نسبت روش چاپک بسیار طولانی‌تر هستند (۶ تا ۲۴ ماه در مقابل نیمی از ماه). در نتیجه در روش‌های طرح-و-ثبت، در فاصله‌ی بین ارائه نتایج به مشتری کارهای بیشتری انجام می‌شود، بنابراین احتمال ردد شدن قسمت بیشتری از کارها از سمت مشتری افزایش می‌یابد.

### خودآزمایی ۱-۵

کدام راهکار ضعیفترین استدلال برای مزایای بهرهوری کامپایلرها برای زبان‌های برنامه‌نویسی سطح بالا است: وضوح به کمک اختصار، سنتر، بازکاربرد، و یا خودکارسازی و استفاده از ابزارها؟ ◊ کامپایلرها استفاده از زبان‌های سطح بالا را ممکن می‌سازند و به برنامه‌نویسان این امکان را می‌دهند تا با استفاده از ساختارهای مختصerte در زبان‌های سطح بالا، بهرهوری خود را افزایش دهند. کامپایلرها همچنین کد سطح بالا را به عنوان ورودی گرفته و کد سطح پایین منتظر با آن را تولید یا سنتز می‌کنند. کامپایلرها قطعاً ابزار نیز هستند. با اینکه شما می‌توانید ادعا کنید که بازکاربرد به کمک زبان‌های سطح بالا آسان‌تر است، اما بازکاربرد ضعیفترین استدلال از بین چهار گزینه برای توضیح مزیت‌های استفاده از کامپایلرها است. ■

### ۶- نرم افزار به صورت یک سرویس و معماری سرویس‌گرا

در اواسط دهه ۱۹۹۰، با گسترش وب<sup>۱۳۷</sup> و افزایش تعداد مخاطبان آن، ایده جدید شروع به ظهور کرد: به جای اتکا به کاربران برای نصب نرم‌افزار بر روی رایانه‌های خود، چرا نرم‌افزار را به صورت مرکزی بر روی سرویهایی مبتنی بر اینترنت اجرا نکنیم و به کاربران امکان دسترسی به آن را از طریق مرورگرها یابشان بدھیم؟ سیلزفوس<sup>۱۳۸</sup> شاید اولین شرکت بزرگی بود که به طور کامل از این مدل جدید که نرم‌افزار به صورت یک سرویس (SaaS) نام گرفت، استقبال کرد. نمونه‌هایی از SaaS<sup>۱۳۹</sup> که امروزه بسیاری از ما هر روز از آن استفاده می‌کنیم عبارت‌اند از موتورهای جستجو<sup>۱۴۰</sup>، شبکه‌های اجتماعی و تماسای ویدیو. اما حتی آپ‌هایی<sup>۱۴۱</sup> مانند واژه‌پرداز<sup>۱۴۲</sup>، مدیریت دفترچه تلفن و تقویم‌ها که قبلًا مدل غالب ارائه آن‌ها برای کاربران از طریق نصب نرم‌افزار بر روی دستگاه‌های ایشان بود، تا حد زیادی به SaaS مهاجرت کرده‌اند. مزایایی که SaaS هم برای کاربران و هم برای توسعه‌دهندگان دارد، محبوبیت آن را توجیه می‌کند:

- از آنجایی که کاربران نیازی به نصب نرم‌افزار کاربردی ندارند، دیگر لازم نیست نگران مناسب و سریع بودن سخت‌افزارشان و یا مناسب بودن نسخهٔ سیستم عامل<sup>۱۴۳</sup> دستگاه‌شان باشند.

<sup>137</sup> Web (World Wide Web)

<sup>138</sup> Salesforce

<sup>139</sup> Software as a Service

<sup>140</sup> Search Engine

عبارت Application Software در زبان انگلیسی که معمولاً به صورت Application یا شکل کوتاه‌تر آن یعنی App به کار می‌رود، به دستهٔ خاصی از نرم‌افزارها اشاره دارد. ما در این ترجمه، برای از عبارت «نرم‌افزار کاربردی» استفاده کردہ‌ایم که برابر رسمی‌تر و دقیق‌تری در زبان فارسی است. اما برای App، از واژهٔ رایج و بدیرفته‌شده «آپ» استفاده کردیم که شکل کوتاه‌تر، عمومی‌تر و آشناتری است. معادل ساده، کوتاه و پذیرفته‌شده دیگری برای این واژه در زبان فارسی وجود ندارد و از آن جایی که این واژه در زبان فارسی نیز به خوبی جا افتاده است، از همین واژه وام‌گرفته بوده گرفتایم. همچنین، از به کار بردن واژه‌هایی مانند «برنامه» یا «نرم‌افزار» (به ترتیب معادل‌های Software و Program) به تنها یک برای اشاره به App پرهیز کردہ‌ایم، چرا که این واژه‌ها دقیق نیستند و ممکن است به مفاهیم گستردگری اشاره داشته باشند. برای حفظ دقیق و جلوگیری از ابهام، ترجیح داده‌ایم از واژگان متمایز و دقیق استفاده کنیم.

<sup>142</sup> Word Processor (Software)

<sup>143</sup> Operating System

**نرم افزار به صورت یک محصول (SaaS)** نامی متدائل است که بعدها و در حدود سال ۲۰۱۵ برای توصیف نرم‌افزاری ظاهر شد که باید هنگام انتشار با نصب شود، برخلاف که در آن کاربر همیشه از آخرین نسخهٔ یک نرم‌افزار تحت وب استفاده می‌کند.

سال معرفی	زبان برنامه‌نویسی	چارچوب برنامه‌نویسی SaaS
۱۹۹۶	VB.NET	صفحه‌های سرور فعل (ASP.NET)
۱۹۹۷	جاوا	بین‌های سازمانی جاوا (EJB)
۱۹۹۹	جاوا	صفحات سرور جاکارتا (JSP)
۲۰۰۲	جاوا	اسپرینگ
۲۰۰۴	روبی	روبی آن ریلز
۲۰۰۵	پایتون	جنگو
۲۰۰۶	پی‌اچ‌پی	زند
۲۰۰۷	روبی	سیناترا

شکل ۱-۶: نمونه‌هایی از چارچوب‌های برنامه‌نویسی SaaS و اینکه هر کدام به چه زبان برنامه‌نویسی نوشته شده‌اند.

۲- داده‌های مرتبط با سرویس معمولاً به همراه خود سرویس نگهداری می‌شود، بنابراین کاربران نیازی به نگرانی در مورد گرفتن نسخهٔ پشتیبان از داده‌ها، از دست دادن آن‌ها بر اثر مشکلات سخت‌افزاری و یا از دست دستگاه نظیر تلفن یا تبلت<sup>۱۴۴</sup> را ندارند.

۳- هنگامی که گروهی از کاربران بخواهند به صورت جمیع بر روی داده‌های یکسانی کار کنند و برهمنکش داشته باشند، SaaS بسیار مناسب است.

۴- هنگامی که حجم داده‌ها زیاد است و یا مرتباً به روزرسانی می‌شوند، منطقی‌تر است که داده‌ها به صورت مرکز نگهداری شوند و دسترسی به آن از راه دور و از طریق SaaS انجام گیرد.

۵- تنها یک نسخه از نرم‌افزار سرور<sup>۱۴۵</sup> در یک محیط یکسان و کنترل شده سخت‌افزاری و نرم‌افزاری (سیستم عامل) که توسط توسعه‌دهنده انتخاب شده اجرا می‌شود. این امر باعث اجتناب از دردسر توزیع فایل‌های اجرایی با قابلیت سازگاری و اجرا بر روی انواع مختلف رایانه‌ها می‌شود. هرچند که مروگرهای مختلف هنوز برخی رفتارهای ناسازگار دارند (موضوعی که در فصل ۶ به آن می‌پردازیم).

۶- از آنجایی که تنها نسخه از نرم‌افزار سرور در اختیار توسعه‌دهندگان است، آن‌ها می‌توانند نرم‌افزار و حتی سخت‌افزار مرتبط با آن را مرتباً ارتقا دهند، مادامی که واسطه‌های برنامه‌نویسی<sup>۱۴۶</sup> (API) خارجی را نقض نکنند. علاوه بر این، توسعه‌دهندگان می‌توانند نسخه‌های جدید نرم‌افزار کاربردی را بر روی تعداد کمی از کاربران به‌طور موقت آزمایش کنند، بدون اینکه برای کاربران با درخواست‌های مکرر برای ارتقا مزاحمتی ایجاد کنند.

۷- شرکت‌های ارائه‌دهنده SaaS دائماً در حال رقابت هستند تا امکانات جدیدی را برای کاربران فراهم کنند تا مطمئن شوند که کاربران آن‌ها را برای رقیب دیگری ترک نمی‌کنند که سرویس بهتری ارائه می‌دهد.

جای تعجب نیست که با محبوبیت SaaS، چارچوب‌های برنامه‌نویسی زیادی با هدف کمک به ساختن این‌گونه از نرم‌افزارهای کاربردی ایجاد شدند. تعدادی از این چارچوب‌ها در شکل ۱-۶ آورده شده‌اند. در این کتاب، ما از چارچوب روبی آن ریلز (یا به صورت سادهٔ ریلز) که به زبان روبی نوشته شده است، استفاده می‌کنیم. هرچند ایده‌هایی را که در این کتاب مطرح می‌شوند می‌توان با سایر چارچوب‌های برنامه‌نویسی نیز به کار گرفت. ما ریلز را به این دلیل انتخاب کردی‌ایم که گروهی آن را ساخته‌اند که چرخهٔ حیات چاپک را به خوبی پذیرفته‌اند، بنابراین ابزارهای مرتبط با آن به خوبی از

<sup>144</sup>Tablet

<sup>145</sup>Server

<sup>146</sup>Application Programming Interface

روش چابک پشتیبانی می‌کند. اگر از قبیل با رویی و یا ریلز آشنایی ندارید، این فرصت خوبی است تا مهارتی مهم در مهندسی نرم افزار را تمرين کنید: همواره سعی کنید که از ابزار مناسب برای کارها استفاده کنید، حتی اگر این به معنای یادگیری یک زبان و یا ابزار جدید باشد! اتفاقاً یکی از ویژگی‌های جذاب جامعه کاربران ریلز این است که مشارکت‌کنندگان آن به طور معمول با ابداع ابزارهای جدید برای خودکارسازی کارهایی که قبلاً به صورت دستی انجام می‌شدند، بهره‌وری را بهبود می‌بخشند.

توجه داشته باشید که ارتفاً مداوم SaaS، به دلیل اینکه تنها یک نسخه از نرم افزار وجود دارد، با چرخهٔ حیات توسعهٔ چابک کاملاً هم‌خوانی دارد. از این‌رو، شرکت‌هایی همچون آمازون، ای‌بی<sup>۱۴۷</sup>، فیسبوک<sup>۱۴۸</sup>، گوگل<sup>۱۴۹</sup> و دیگر سرویس‌دهندگان SaaS، همگی بر چرخهٔ حیات چابک متکی هستند. همچنین شرکت‌های نرم افزاری سنتی‌تر همچون مایکروسافت<sup>۱۵۰</sup> بیش از پیش و به صورت فرازینده‌ای از روش چابک در توسعهٔ محصولات‌شان استفاده می‌کنند. فرایند چابک با ماهیت تغییرات سریع و مداوم در نرم افزارهای کاربردی SaaS کاملاً منطبق است.

با وجود تمامی مزایایش، SaaS هنوز یک مزیت مهم را در زمینهٔ بازکاربرد نرم افزاری کم داشت. هنگام تولید SaaS<sup>۱۵۱</sup>، توسعه‌دهندگان می‌توانند از **کتابخانه‌های نرم افزاری**، حاوی کدهایی برای انجام وظایف مشترک بین بسیاری از نرم افزارهای کاربردی، استفاده گستردگی کنند. از آنجایی که این کتابخانه‌ها را غالب دیگران می‌نوشند (به اصطلاح کتابخانه‌های شخص ثالث)، مزیت بازکاربرد نرم افزاری را با خود به همراه داشتند. در اواسط دهه ۲۰۰۰، پدیده مشابهی در دنیای SaaS شروع به شکل‌گیری کرد: ظهرور **معماری سرویس‌گرا** (SOA<sup>۱۵۲</sup>)، که در آن یک سرویس Mi توانست از سرویس‌های دیگری که توسعه‌دهندگان دیگر ساخته و نگهداری می‌شوند، برای کارهای متداول و مشترک استفاده کند. سرویس‌هایی که به صورت اختصاصی برای طیف محدودی از وظایف ساخته شده بودند، ریزسرویس<sup>۱۵۳</sup> نامیده شدند. نمونه‌های رایج امروزی از این دسته از سرویس‌ها شامل پردازش کارت اعتباری، جست‌وجوی اینترنتی، مسیریابی برای رانندگی و بسیاری موارد دیگر هستند. با مستحکم شدن استانداردهایی برای ارائه، تعامل و برهمکنش<sup>۱۵۴</sup> با چنین سرویس‌های خارجی، سرانجام مزیت مهم بازکاربرد نرم افزاری برای SaaS از راه رسید. فصل ۳ به جزئیات بیشتری در مورد SOA و ریزسرویس‌ها می‌پردازد.

البته، ما هنوز باید به یک تفاوت عمدی بین SaaS و SaaP<sup>۱۵۵</sup> بپردازیم، و آن زیرساخت سخت‌افزاری است که آپ‌ها<sup>۱۵۶</sup> بر روی آن اجرا می‌شوند. در مورد SaaP، این سخت‌افزار از رایانه‌های شخصی<sup>۱۵۷</sup> میلیون‌ها کاربر تشکیل شده است. در قسمت بعدی، به زیرساخت سخت‌افزاری که SaaS را ممکن می‌سازد، خواهیم پرداخت.

**چکیده:** نرم افزار به صورت یک سرویس (SaaS) هم برای مشتریان و هم برای ارائه‌دهندگان سرویس چابک و خوش‌آیند است، زیرا داشتن کلاینت فرآگر و همگانی (مرورگر وب) استفاده از سرویس را برای کاربران راحت‌تر کرده است و همچنین وجود نسخه‌های واحد از نرم افزار در یک محیط متمرکز، ارائه و ارتقای سرویس را برای ارائه‌دهندگان آن آسان‌تر کرده است. با توجه به وجود توانایی و تمايل به ارتقای مکرر در مورد SaaS، فرایند توسعهٔ نرم افزار چابک برای توسعهٔ این‌گونه نرم افزار محبوب است. از همین‌رو چارچوب‌های بسیاری برای پشتیبانی هم‌زمان از SaaS و روش چابک وجود دارد. این کتاب از روی آن ریلز استفاده می‌کند.

<sup>147</sup> eBay

<sup>148</sup> Facebook

<sup>149</sup> Google

<sup>150</sup> Microsoft

<sup>151</sup> Software as a Product

<sup>152</sup> Service Oriented Architecture

<sup>153</sup> Microservice

<sup>154</sup> Interaction

<sup>155</sup> App (Application Software/Program)

<sup>156</sup> Personal Computer

**خودآزمایی ۱-۶**

برخی از محبوب‌ترین آپ‌های شرکت گوگل که به صورت SaaS ارائه می‌شوند عبارت‌اند از جست‌وجوگر گوگل، یوتیوب، گوگل میز، جی‌میل، تقویم گوگل و گوگل داکس. برای هر یک از این آپ‌ها، یک مزیت ارائه آن به عنوان SaaS به جای SaaP را نام ببرید.

◊ پاسخ‌های صحیح بسیاری برای این پرسش وجود دارد، در ادامه پاسخی که ما به این پرسش می‌دهیم آمده است:

۱- عدم نیاز به نصب توسط کاربر: گوگل داکس

- ۱ از دست نرفتن داده‌ها: جی‌میل، تقویم گوگل
- ۲ همکاری کاربران: گوگل داکس
- ۳ مجموعه داده‌های بزرگ و در حال تغییر: جستجوگر گوگل، گوگل مپز، گوگل نیوز و بوتیوب
- ۴ نرم افزار به صورت متمرکز در یک محیط واحد: جستجوگر گوگل
- ۵ عدم نیاز به حضور در محل به هنگام به روزرسانی آپ: گوگل داکس

## ۲-۶-۱ خودآزمایی

درست یا نادرست: اگر شما از فرایند توسعه چابک برای توسعه آپ‌های SaaS استفاده می‌کنید، می‌توانید به جای روبی و ریلز از پایتون و جنگو و یا از زبان‌های مبتنی بر چارچوب دات‌نت شرکت ماکروسافت و ASP.NET استفاده کنید.

درست. جنگو و ASP.NET نیز از چارچوب‌های برنامه‌نویسی برای SaaS و توسعه به روش چابک هستند. ■

## ۱-۱ به کاراندازی SaaS: رایانش ابری

### جان مک‌کارتی (۱۹۷۱-۲۰۱۱) در

سال ۱۹۷۱ جایزه نورینگ را دریافت کرد. اوی مخترع زبان برنامه‌نویسی لیسبی بود و از پیشگامان استفاده از تکنیک اشتراک زمانی بر روی رایانه‌های بزرگ بود. با آمدن خوشه‌هایی از سخت‌افزارهای معمولی و گسترش شبکه‌های سریع، روپایش برای ایجاد «رایانش همگانی» اشتراکی به حقیقت پیوست.



اگر رایانه‌هایی که من از آن‌ها صحبت کرده‌ام و طرفدارشان هستم در آینده حضور داشته باشند، آنگاه شاید روزی برسد که رایانش<sup>۱۵۷</sup> همچون سامانه تلفن بعنهوان یک ابزار و خدمات همگانی در بیاید... این رایانش همگانی می‌تواند پایه و بنیان یک صنعت جدید و مهم را بنا گزارد.

جان مک‌کارتی<sup>۱۵۸</sup>، در جشن صد سالگی مؤسسه فناوری ماساچوست<sup>۱۵۹</sup> (MIT) در سال ۱۹۶۱

SaaS سه خواسته را برای زیرساخت فناوری اطلاعات<sup>۱۶۰</sup> (IT) مطرح می‌کند:

- ۱ ارتباط، اینکه به هر مشتری امکان تعامل و کار با سرویس داده شود.
- ۲ مقیاس‌پذیری<sup>۱۶۱</sup>، اینکه مرکز سرویس‌دهنده باید بتواند با وجود نوسانات تقاضا در طول روز و پیوست.
- ۳ دسترسی‌پذیری<sup>۱۶۲</sup>، اینکه هم سرویس و هم حامل ارتباطات باید بی‌وقفه در دسترس باشد: همه روزه و در هر ۲۴ ساعت از شباهنگ روز (که با ۲۴×۷ «۲۴×۷ نشان داده می‌شود). استاندارد طلایی در مورد دسترسی‌پذیری که توسط سامانه تلفن همگانی ایالات متحده تعیین شده است، در دسترس بودن در ۹۹,۹۹٪ زمان (معروف به «پنج تا نه») یا چیزی در حدود ۵ دقیقه قطعی در سال است. وبگاه آمازون چهار تا نه را هدف قرار داده است، که حتی برای سرویسی که به خوبی اداره می‌شود دشوار است.

<sup>157</sup>Computing

<sup>158</sup>John McCarthy

<sup>159</sup>Massachusetts Institute of Technology

<sup>160</sup>Information Technology

<sup>161</sup>Scalability

<sup>162</sup>Availability

## فصل ۱. مقدمه‌ای بر توسعهٔ چاپک و SAAS

وجود اینترنت و البته دسترسی به اینترنت پسرعت در خانه به راحتی نیاز ارتباطی SaaS را برطرف می‌کند. اگرچه که تعدادی از اولین سرویس‌های مبتنی بر وب بر روی رایانه‌های بزرگ و گران قیمت به کاراندازی شده بودند – تا حدی به این علت که این رایانه‌ها قابل اطمینان‌تر بودند و تا حدی به این علت که اداره کردن تعداد کمی رایانهٔ بزرگ آسان‌تر بوده است – رویکردی برخلاف آن خیلی زود صنعت را تصاحب کرد. مجموعه‌هایی از تعدادی رایانهٔ کوچک و ارزان قیمت که به‌وسیلهٔ سوئیچ‌های اینترنت<sup>۱۶۳</sup> معمولی به‌هم متصل شده بودند، که بعداً به **خوشه**<sup>۱۶۴</sup> معروف شدند، چندین مزیت نسبت به رویکرد قبلی یا همان سخت‌افزار «آهن گنده»<sup>۱۶۵</sup> داشت:

- به علت اتکای آن‌ها به سوئیچ‌های اینترنت برای اتصال به یکدیگر، خوشه‌ها بسیار مقیاس‌پذیرتر از سروهای مرسوم سنتی هستند. در ابتدا خوشه‌ها شامل ۱۰۰۰ رایانه می‌شوند. مراکز داده<sup>۱۶۶</sup> امروزی بیش از ۱۰۰,۰۰۰ رایانه در خود جای دادند.

- انتخاب بادقت نوع سخت‌افزار مورد استفاده در مرکز داده از یک سو و کنترل بادقت وضعیت نرم‌افزار از سوی دیگر، این امکان را فراهم کرد تا تعداد بسیار کمی اپراتور بتوانند با موفقیت هزاران سرور را راهاندازی و اداره کنند. به طور مشخص، برخی از مراکز داده برای سهولت از **ماشین‌های مجازی**<sup>۱۶۷</sup> استفاده می‌کنند. یک ناظر ماشین مجازی<sup>۱۶۸</sup> نرم‌افزاری است که یک رایانهٔ واقعی را چنان با موقوفیت شبیه‌سازی و تقلید می‌کنند که شما می‌توانید حتی یک سیستم عامل را بر روی تجربید ماشین مجازی‌ای که ارائه می‌دهد اجرا کنید (پویک و گلبرگ ۱۹۷۴). هدف این است که این شبیه‌سازی با سریار<sup>۱۶۹</sup> کم صورت گیرد و یکی از کاربردهای معمول آن برای ساده‌سازی توزیع نرم‌افزار در یک خوشه است. به این ترتیب، چندین آپ می‌توانند سخت‌افزار را با یکدیگر به اشتراک بگذارند، به طوری که هر آپ بر این باور است که بر روی یک نسخهٔ مجزا و اختصاصی سیستم عامل اجرا می‌شود. اگر بنا بر این باشد که آپ‌ها سیستم عامل را نیز به اشتراک بگذارند، آن‌گاه راه کارآمدتر برای اشتراک‌گذاری سخت‌افزار، استفاده از **مجازی‌سازی**<sup>۱۷۰</sup> است. یک نمونه از این نوع از مجازی‌سازی، استفاده از ابزار محبوب داکر<sup>۱۷۱</sup> است که به هر آپ اجازه می‌دهد تا در کانتینر<sup>۱۷۲</sup> متعلق به خودش بر روی یک سیستم عامل مشترک اجرا شود.

دو تن از معمارهای ارشد شرکت گوگل نشان دادند که هزینهٔ معادل تعدادی پردازنده<sup>۱۷۳</sup>، حافظه و فضای ذخیره‌سازی برای خوشه‌ها بسیار کمتر از روش «آهن گنده» است، شاید چیزی در حدود بیست برابر کمتر (باروسو و هلسله ۲۰۰۹).

- اگرچه سروهای سامانه‌های ذخیره‌سازی سنتی نسبت به قطعات مورد استفاده در خوشه‌ها قابل اطمینان‌تر هستند، اما زیرساخت نرم‌افزاری خوشه با بهره‌گیری حداکثری از افزونگی نرم‌افزاری و سخت‌افزاری باعث قابل اطمینان شدن کل سامانه می‌شود. هزینهٔ کم سخت‌افزار باعث می‌شود تا افزونگی در سطح سخت‌افزار ارزان تمام شود. سرویس‌دهنده‌های امروزی همچنین از چندین مرکز داده که از نظر جغرافیایی توزیع شده‌اند استفاده می‌کنند تا بروز بلاهای طبیعی منجر به قطعی سرویس‌شان نشود.

<sup>۱۶۳</sup>Ethernet

<sup>۱۶۴</sup>Computer Cluster

<sup>۱۶۵</sup>اصطلاح آهن گنده که ریشه آن به دهه ۱۹۷۰ میلادی برمند است، به معنی یک رایانه بسیار بزرگ، گران قیمت و سریع است و معمولاً به رایانه‌های بزرگ مانند بزرگ‌رایانه شرکت آی‌بی‌ام و آبرایانه شرکت کری اشاره دارد.

<sup>۱۶۶</sup>Datacenter

<sup>۱۶۷</sup>Virtual Machine

<sup>۱۶۸</sup>Virtual Machine Monitor

<sup>۱۶۹</sup>Overhead

<sup>۱۷۰</sup>Virtualization

<sup>۱۷۱</sup>Docker

<sup>۱۷۲</sup>Container (Virtualization)

<sup>۱۷۳</sup>Processor (Computing)

<sup>۱۷۴</sup>Redundancy

با رشد مراکز داده اینترنتی، تعدادی از سرویس‌دهندگان به این نتیجه رسیدند که هزینهٔ سرانه آن‌ها به مراتب کمتر از هزینه‌ای است که دیگران برای راهاندازی مراکز داده کوچک‌ترشان می‌پردازند. این عمدتاً به علت صرفه به مقیاس <sup>۱۷۵</sup> است، زمانی است که شما تعداد ۱۰۰,۰۰۰ رایانه را در یک زمان خریداری و اداره می‌کنید. آن‌ها همچنین از بهره‌وری <sup>۱۷۶</sup> بالاتری سود می‌برند چراکه شرکت‌های بسیاری می‌توانند این مراکز داده عظیم را، که (باروسو و هلتسله ۲۰۰۹) <sup>۱۷۷</sup> نامیدند، به اشتراک بگذارند. این در حالی است که مراکز داده کوچک‌تر معمولاً بهره‌وری بین ۱۰٪ تا ۲۰٪ دارند. بنابراین این شرکت‌ها دریافتند که می‌توانند با در اختیار گذاشتن سخت‌افزار مرکز داده خود به صورت «پرداخت به میزان مصرف» <sup>۱۷۸</sup> کسب درآمد کنند.

<sup>۱۷۹</sup> نتیجهٔ آن با نامهای سرویس‌های ابری عمومی، رایانش همگانی و یا اغلب رایانش ابری شناخته می‌شود. این گونه سرویس‌ها رایانش، فضای ذخیره‌سازی و ارتباطات را در ازای مقداری اندرکی پول برای هر ساعت ارائه می‌دهند (آرمیراست و دیگران ۲۰۱۰). علاوه بر این، هیچ هزینهٔ اضافی برای افزایش مقیاس پرداخت نمی‌شود: استفاده از ۱۰۰۰ رایانه به مدت ۱ ساعت، هزینهٔ بیشتری نسبت به استفاده از ۱ رایانه به مدت ۱۰۰۰ ساعت ندارد. سرویس‌های وب آمازون <sup>۱۸۰</sup>، موتور اجرای برنامه گوگل <sup>۱۸۱</sup> و مایکروسافت آژور <sup>۱۸۲</sup>، نمونه‌هایی پیش رو از مدل رایانش به صورت پرداخت به میزان مصرف هستند که به نوعی «بینهایت مقیاس‌پذیر» <sup>۱۸۳</sup> به شمار می‌روند. سرویس ابری همگانی یعنی اینکه امروزه هر شخصی با داشتن یک کارت اعتباری و یک ایدهٔ خوب می‌تواند یک شرکت عرضه <sup>۱۸۴</sup> را راهاندازی کرده که می‌تواند حتی تا داشتن میلیون‌ها مشتری رشد کند، بدون اینکه به ساخت و اداره مرکز داده شخصی خودش نیاز داشته باشد.

از سال ۲۰۱۰ تا ۲۰۲۰، رایانش ابری و SaaS تحولی بزرگ را در صنعت رایانه آغاز کردند. برای مشخص شدن تاثیر کامل این انقلاب باید تا پایان این دههٔ پیش رو صبر کرد. اما چیزی که مشخص است این است که مهندسی و ساخت SaaS برای رایانش ابری، اساساً با مهندسی بسته‌های نرم‌افزاری برای رایانه‌های شخصی و سرورها متفاوت است، و به همین علت شما در حال خواندن این کتاب هستید.

<sup>175</sup>Economies of Scale

<sup>176</sup>Utilization

<sup>177</sup>Warehouse Scale Computer

<sup>178</sup>Pay-as-you-go

<sup>179</sup>Cloud Computing

<sup>180</sup>Amazon Web Services (AWS)

<sup>181</sup>Google App Engine

<sup>182</sup>Microsoft Azure

<sup>183</sup>Scalable

**لوئیز باروسو**، از مدیران ارشد شرکت گوگل و برندهٔ جایزه ACM/IEEE Eckert-Mauchly

در سال ۲۰۲۰، در ابتدای سخنرانی خود برای پذیرش این جایزه، تاریخچه مختصه رایانه‌های انبارگون را ارائه می‌دهد.

**چکیده**

- اینترنت راه ارتباطی را برای SaaS فراهم می‌کند.
- **رایانش ابری** سخت‌افزار رایانش و ذخیره‌سازی مقیاس‌پذیر و قابل اطمینان را برای SaaS فراهم می‌کند.
- رایانش ابری شامل **خوشه‌هایی** از سرورهای معمولی است که به‌وسیله سوئیچ‌های شبکه محلی به‌هم متصل هستند. و یک لایه نرم‌افزاری بر روی آن‌ها قرار دارد که افزونگی کافی برای قابل اطمینان ساختن این سخت‌افزارهای مقررین به صرفه را فراهم می‌کند.
- این خوشه‌های بزرگ یا رایانه‌های انبارگون با صرفه به مقیاس باعث کاهش هزینه‌ها می‌شوند.
- با بهره جستن از صرفه به مقیاس، تعدادی از ارائه‌دهندگان سرویس‌های رایانش ابری این زیرساخت سخت‌افزاری را به عنوان **سرویس رایانش همگانی** کم‌هزینه ارائه می‌دهند که هر کسی می‌تواند از آن به صورت پرداخت به میزان مصرف استفاده کند. به این شکل که در صورت نیاز با افزایش درخواست کاربران فوراً منابع بیشتری تهیه می‌شود و به محض اینکه حجم درخواست‌ها فروکش کرد، منابع اضافی بازگردانده می‌شوند.

**خودآزمایی ۱-۷-۱**

درست یا نادرست: مراکز داده داخلی با پذیرش SOA و خرید سخت‌افزارهای مشابه رایانه‌های انبارگون می‌توانستند به همان اندازه در هزینه‌ها صرفه‌جویی کنند.

◊ نادرست. اگرچه که تقلید از راهکارهای مورد استفاده در رایانه‌های انبارگون می‌توانست هزینه‌ها را تا حدی کاهش دهد، اما مزیت عمده رایانه‌های انبارگون حاصل صرفه به مقیاس آن است که امروزه به معنی ۱۰۰,۰۰۰ سرور است که بسیار بزرگ‌تر از اکثر مراکز داده داخلی است. ■

**۸-۱ به کاراندازی SaaS: مرورگرها و تلفن همراه**

از حدود سال ۱۹۹۴ موفقیت خیره‌کننده وب به سرعت منجر به حذف تدریجی بسیاری از آپ‌های SaaS دسکتاپ<sup>۱۸۴</sup> شد. رابطه‌های کاربری کلاینت<sup>۱۸۵</sup> اختصاصی سرویس‌های پولی مانند AOL<sup>۱۸۶</sup> ای اوال<sup>۱۸۷</sup> و کامپیوسر<sup>۱۸۸</sup> با درگاه‌های وب<sup>۱۸۹</sup> رایگانی مانند یاهو! جایگزین شدند. آپ‌های SaaS<sup>۱۹۰</sup> تخصصی برای دسترسی به خدمات مبتنی بر اینترنت، مانند یودارا<sup>۱۹۱</sup> برای رایانامه، با سرویس‌های رایانامه مبتنی بر مرورگر<sup>۱۹۲</sup> مانند هات‌میل جایگزین شد. حتی آپ‌هایی همچون مایکروسافت ورد نیز از سوی رقبای مبتنی بر مرورگر خود همچون گوگل داکس<sup>۱۹۳</sup> تحت فشار قرار گرفتند. بنابراین مرورگر به

<sup>184</sup>Desktop

<sup>185</sup>User Interface

<sup>186</sup>Client

<sup>187</sup>AOL

<sup>188</sup>CompuServe

<sup>189</sup>Web Portal

<sup>190</sup>Eudora (Email Client)

<sup>191</sup>Browser (Web)

<sup>192</sup>Google Docs (Online Word Processor)

یک کلاینت همگانی و فراگیر تبدیل شد: هر وبگاه می‌توانست تمامی اطلاعات لازم برای رندر<sup>۱۹۳</sup> و نمایش داده شدن رابط کاربری‌اش را در هنگام بازدید به مرورگر ارائه دهد. این کار با استفاده از HTML<sup>۱۹۴</sup>, یا همان زبان نشانه‌گذاری ابیمتنی، صورت می‌گرفت. همانطور که در ادامه خواهیم دید، جاوااسکریپت بعداً به عنوان راهی برای غنی‌تر کردن تجربه برهمنشی<sup>۱۹۵</sup> صفحات وب<sup>۱۹۶</sup> وارد صحنه شد، اما محتوای واقعی قابل مشاهده صفحه همواره از HTML تشکیل شده است.

همانطور که از نام آن پیداست، HTML نمونه‌ای از یک زبان نشانه‌گذاری<sup>۱۹۷</sup> است: متن را با نشانه‌گذاری (حاشیه‌نویسی‌های در مورد ساختار متن) به صورت ترکیب می‌کند که تشخیص نحوی و قواعدی این دو از هم آسان باشد. از نظر فنی، HTML (نسخه پرکاربرد کنونی) در واقع فقط یک نوع سند است که می‌تواند به زبان XML<sup>۱۹۸</sup> بیان شود. XML یک زبان نشانه‌گذاری گسترش‌پذیر است که می‌تواند هم برای نمایش داده‌ها و هم برای توصیف زبان‌های نشانه‌گذاری دیگر استفاده شود.

جداسازی ساختار منطقی یک سند HTML از ظاهر آن، مزایای زیادی را به همراه دارد. ساختار به نوع محتوایی اشاره دارد که هر جزء منطقی یک صفحه نشان می‌دهد، مانند عنوان اصلی یا فرعی، فهرست گلوله‌ای، یک پاراگراف متن و غیره. برخی از اجزای صفحه ساده هستند، مانند عنوان صفحه یا منوی کشویی<sup>۱۹۹</sup> انتخاب‌ها، و مطابق با یک عنصر HTML بدون فرزند هستند. اغلب، یک جزء از صفحه شامل یک عنصر div<sup>۲۰۰</sup> است که عناصر دیگری را نیز در داخل خود جای داده است. عناصر div<sup>۲۰۱</sup> اغلب عناصری که از نظر منطقی با هم مرتبط هستند (و ممکن است تودرتو باشند) را باهم گروه‌بندی می‌کند. ظاهر نه تنها به تایپوگرافی<sup>۲۰۰</sup> اساسی مانند فونت‌ها و رنگ‌ها، بلکه به چیدمان عناصر در یک صفحه نیز اشاره دارد. به عنوان مثال، یک منوی پیمایش که در حالت عادی و در یک صفحه نمایش کامل بهتر است به صورت مجموعه‌ای از زبانه‌های افقی نمایش داده شود، اگر روی صفحه تلفن همراه نشان داده شود، بهتر است که به صورت یک منوی کشویی مشخص شود، هر چند که انتخاب‌های داخل منو و معانی آن‌ها یکسان هستند.

کلید جداسازی ساختار و ظاهر، استفاده از **شیوه‌نامه آبشاری** یا به اختصار CSS<sup>۲۰۱</sup> است که در سال ۱۹۹۶ به عنوان راهی برای مرتبط کردن اطلاعات مربوط به رندر بصیری با عناصر HTML معرفی شد. مفهوم کلیدی CSS عبارت است از **انتخابگر**<sup>۲۰۲</sup>—عبارتی که با یک یا بیشتر از عناصر HTML در یک سند مطابقت دارد. حتی اگر شما توسعه‌دهنده‌ای نیستید که مسئول ظاهر بصیری خواهد بود، درک انتخابگرهای CSS مهم است، زیرا همانطور که در فصل ۶ خواهیم دید، انتخابگرها مکانیزم کلیدی هستند که توسط چارچوب‌های جاوااسکریپت مانند جی‌کوئری<sup>۲۰۳</sup> استفاده می‌شود و به شما امکان می‌دهد صفحات وب برهمکنشی و غنی بسازید. در حالی‌که چندین روش وجود دارد که یک انتخابگر می‌تواند با یک عنصر<sup>۲۰۴</sup> تطبیق داده شود، اما معمول‌ترین روش مرتبط کردن انتخابگر با خصوصیت class<sup>۲۰۵</sup> مربوط به عناصر است، زیرا در یک صفحه، چندین عنصر از انواع مشابه یا متفاوت می‌توانند خصوصیات کلاس یکسانی را به اشتراک بگذارند. شکل ۷-۱ یک صفحه خیلی ساده HTML را نشان می‌دهد. مکانیسم اصلی استفاده از CSS برای «استایل دادن» به HTML و یا تعریف سبک نگارش آن به شرح زیر است:

۱- وقتی مرورگر صفحه را بارگیری می‌کند، به دنبال یک یا چند عنصر link می‌گردد، که باید

<sup>193</sup> Render (Computing)

<sup>194</sup> HyperText Markup Language

<sup>195</sup> Interactive

<sup>196</sup> Webpage

<sup>197</sup> Markup Language

<sup>198</sup> Extensible Markup Language

<sup>199</sup> Dropdown Menu

<sup>200</sup> Typography

<sup>201</sup> Cascading Style Sheets

<sup>202</sup> Selector (CSS)

<sup>203</sup> jQuery

<sup>204</sup> Element

<sup>205</sup> Attribute (Computing)

به عنوان یادآوری، در ابتدای فصل و در صفحه پیش‌نیازها و مفاهیم، مطالعی را برای بارگیری خودآموز مبانی اولیه HTML و CSS پیشنهاد کرده‌ایم.

<https://gist.github.com/edb6a189f7892a17b0bc06f0ec2e6d34>

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <link rel="stylesheet"
5       href="https://getbootstrap.com/docs/4.0/dist/css/bootstrap.min.css">
6     <title>Dietary Preferences of Penguins</title>
7   </head>
8   <body>
9     <div class="container">
10       <h1>Introduction</h1>
11       <p class="lead">
12         This article is a review of the book
13         <i>Dietary Preferences of Penguins</i>,
14         by Alice Jones and Bill Smith. Jones and Smith's controversial work
15         makes three hard-to-swallow claims about penguins:
16       </p>
17       <ul class="list-group">
18         <li class="list-group-item">
19           First, that penguins actually prefer eating tropical foods to fish
20         </li>
21         <li class="list-group-item">
22           Second, that eating tropical foods makes them smell unattractive to predators
23         </li>
24       </ul>
25     </div>
26   </body>
27 </html>
```

شکل ۱-۱: یک سند ۵ HTML در ساده‌ترین حالت یک فایل متی است که با دستور «اعلان نوع سند» تعریف شده در XML شروع می‌شود و به دنبال آن یک عنصر `html` می‌آید که عناصر فرزندش نشانگر اجزای موجود در صفحه هستند. استفاده از نماد قلاب‌های زاویه‌ای برای تعریف تگ‌های HTML برگرفته است از SGML (زبان نشانه‌گذاری تعمیم‌بافته استاندارد)، که در واقع استانداردسازی مذوونی است از زبان نشانه‌گذاری تعمیم‌بافته آی‌بی‌ام. این زبان در دهه ۱۹۶۰ میلادی برای کدگذاری سندهای پروژه‌ها توسعه یافت، با این هدف که این سندها (به راحتی) توسط رایانه قابل خوانده شدن باشند.

فرزندان عنصر `HTML` از سند `head` باشند. این عناصر شیوه‌نامه‌ها<sup>۲۰۶</sup> را مشخص می‌کنند که باید از آن‌ها در کنار این سند `HTML` استفاده شوند. در این مثال، خصوصیت `href` از عنصر پیوند<sup>۲۰۷</sup> (یا همان مقصد و هدفش) به شیوه‌نامه اصلی چارچوب بوت‌استرپ<sup>۲۰۸</sup> اشاره دارد که در ادامه به آن می‌پردازیم.

- مرورگر، هر شیوه‌نامه‌ای را که مورد ارجاع قرار گرفته باشد بارگیری می‌کند. یک شیوه‌نامه شامل مجموعه‌ای از انتخابگرها است، و برای هر انتخابگر، مجموعه‌ای از قوانین برای نمایش عناصری که با آن انتخابگر مطابقت دارند نیز گنجانده شده. این قوانین می‌توانند تایپوگرافی، طرح‌بندی صفحه، رنگ‌ها و موارد دیگر را مشخص کنند.

- هنگام نمایش صفحه، مرورگر قوانین تعریف شده<sup>۲۰۹</sup> CSS را با عناصر منطبق در هر صفحه نمایش داده شده مطابقت می‌دهد. در این مثال، چارچوب بوت‌استرپ قوانین پایه‌ای برای شیوه نمایش هر نوع عنصر<sup>۲۱۰</sup> (`h1`, `p`, `ul`, و غیره) را ارائه می‌کند، و خصوصیات `class` در عناصر مختلف برای مطابقت با انتخابگرهای خاص CSS در چارچوب بوت‌استرپ آورده شده‌اند تا با استفاده از آن‌ها قالب‌بندی خاصی به عناصر صفحه داده شود.

**وبگاه باع ذهن CSS نشان**  
در حالی‌که قواعد CSS ساده است، ساخت شیوه‌نامه‌هایی که از نظر بصری جذاب باشند، نیازمند مهارت‌های طراحی و تایپوگرافی است. آن دسته از ما که فاقد آن مهارت‌ها هستند، بهتر است که از شیوه‌نامه‌هایی موجود طراحی شده توسط حرffe‌ای‌ها استفاده کنند. مجموعه‌هایی از این شیوه‌نامه‌هایی از پیش طراحی شده را تحت عنوان چارچوب‌های CSS می‌شناسند. این مجموعه‌ها گاهی شامل کدهایی به زبان جاوا‌اسکریپت نیز هستند تا جلوه‌های بصری بیشتری (مثل انیمیشن و یا محو کردن) را که نمی‌توان به تنهایی با CSS پوشش داد، به نمایش بگذارند. این نوع از مجموعه‌ها تحت عنوان چارچوب‌های پیشین<sup>۲۱۱</sup> شناخته می‌شوند. یکی از این چارچوب‌های پیشین پرکاربرد که ما نیز در این کتاب بارها به آن اشاره می‌کنیم چارچوب بوت‌استرپ است، که یک پروژه متعدد باز<sup>۲۱۲</sup> ارائه شده توسط شرکت توییتر<sup>۲۱۳</sup> است. یک چارچوب CSS خوب، حداقل چهار مزیت اصلی زیر ارائه می‌دهد:

۱- مجموعه‌ای از مؤلفه‌های سطح بالا که چندین عنصر HTML سطح پایین را در یک واحد منطقی ترکیب و ارائه می‌دهد. به عنوان مثال، یک منوی پیمایش با فهرست‌های کشویی می‌تواند به عنوان یک مؤلفه واحد مدیریت شود، هرچند که شامل چندین عنصر HTML است.

۲- ارائه یک تعریف استعاری شtronجی از ابعاد صفحه برای مشخص کردن چیدمان اجزا در آن. مثلاً در مورد بوت‌استرپ، صفحه به ۱۲ ستون تقسیم می‌شود، و هر مؤلفه را می‌توان به گونه‌ای مشخص کرد که تعدادی از ستون‌ها را در بر بگیرد، به علاوه اینکه همان مؤلفه، دستورالعمل‌های طرح‌بندی متفاوتی برای صفحه‌های کوچک‌تر و بزرگ‌تر دارد.

۳- پشتیبانی از طراحی وب واکنش‌گرا<sup>۲۱۴</sup> و نمایش عناصر صفحه با توجه به اندازه صفحه نمایش. برای مثال، یک منوی پیمایش که معمولاً به عنوان مجموعه‌ای از زبانه‌های افقی نمایش داده می‌شود، زمانی که نمایشگر خیلی کوچک است، به طور خودکار به عنوان گزینه‌های عمودی که روی هم سوار شده‌اند نمایش داده می‌شود، حتی اگر به صراحت چنین دستورالعمل‌هایی را ارائه نکرده باشد. ویگاه<sup>۲۱۵</sup> و بعدها<sup>۲۱۶</sup> نموونه‌هایی از نحوه رفتار نوارهای پیمایش در بوت‌استرپ را با تغییر اندازه صفحه نمایش نشان می‌دهد.

<sup>206</sup> Stylesheet (Web Development)

<sup>207</sup> Link (Web Hyperlink)

<sup>208</sup> Bootstrap (Front-end Framework)

<sup>209</sup> Frontend

<sup>210</sup> Open Source (Software)

<sup>211</sup> Twitter

<sup>212</sup> Responsive Web Design

۴- پشتیبانی از دسترسی‌پذیری<sup>۲۱۳</sup> برای کاربران دارای معلولیت. به طور مثال، محتوای را که باید از نظر بصری پنهان باشد، می‌توان طوری ارائه داد که همچنان برای فناوری‌های کمکی مانند صفحه‌خوان‌ها قابل دسترسی باشد.

چارچوب‌های CSS/HTML با سلطه دستگاه‌های تلفن همراه اهمیت ویژه‌ای پیدا کرده‌اند. شرکت اپل<sup>۲۱۴</sup> در سال ۲۰۰۷ آیفون<sup>۲۱۵</sup> را معرفی کرد. اگرچه قطعاً این اولین گوشی هوشمندی نبود که امکان نصب آپ‌های جانبی یا مرور وب را فراهم می‌کرد، اما اولین گوشی بود که در این کار بسیار موفق شد و متعاقباً به طور گستردگی مورد کمی‌برداری قرار گرفت. تا سال ۲۰۱۷، تنها ده سال بعد، حدود یک سوم جمعیت جهان گوشی‌های هوشمند<sup>۲۱۶</sup> داشتند، که در مجموع بیشتر از دسکتاپ‌ها با لپ‌تاپ‌ها<sup>۲۱۷</sup> از وبگاه‌ها بازدید می‌کردند (انگ ۲۰۱۸). تا حد خوبی، سبک‌ها و تنظیمات تعریف شده توسط CSS که با دقیق طراحی شده‌اند می‌توانند محتواهای HTML یکسانی را در طیف وسیعی از اندازه‌های مختلف صفحه قابل استفاده کنند. به همین دلیل، در حالی‌که شکل ۱-۱ نشان می‌دهد که چندین رویکرد برای توسعهٔ برنامه‌های کلاینت وجود دارد، ما در این کتاب توصیه به ساخت آپ‌های «تلفن‌همراه‌محور» می‌کیم که با استفاده از HTML، CSS و جاوااسکریپت ایجاد شده‌اند و شاید در ادامه آن‌ها را به برنامه‌های وب پیشرو<sup>۲۱۸</sup> ارتقا دهید (قسمت ۶-۶). این رویکرد از ابزارهای گستردگی موجود در آن اکوسیستم، به ویژه چارچوب‌های نمایش مانند یوتاسترپ، کتابخانه‌های دستکاری و انجام تغییرات بر روی DOM<sup>۲۱۹</sup> مانند چی‌کوئری (قسمت ۶-۶) و ابزارهای آزمونی مانند چرزمین<sup>۲۲۰</sup> (قسمت ۶-۶) بهره می‌برد.

على رغم تفاوت‌های موجود در روش‌های ساخت SaaS موبایل یا دسکتاپ، همهٔ این آپ‌ها از نظر ساختاری مشابه هستند: آن‌ها یک رابط کاربری محلی ارائه می‌دهند که احتمالاً شامل فضای ذخیره‌سازی محلی بر روی دستگاه کاربر نیز هست. آن‌ها همچنین از استانداردها و پروتکل‌های آزاد و باز مربوط به SaaS برای برقراری ارتباط با یک یا چند سرور دوردست<sup>۲۲۱</sup> استفاده می‌کنند.

**بارگشت به SaaS؛ گسترش آپ‌های قابل نصب توسط کاربر**  
یک مزیت عمده SaaS را از بین برده است: کاربران باید یک بار دیگر به صورت دستی آپ‌های خود را در صورت یافتن اشکال امنیتی یا زمانی که دستگاه‌های خود را ارتقا می‌دهند، بهروزرسانی کنند. و البته این بهروزرسانی‌ها در مورد آپ‌های تلفن همراه بسیار بیشتر از نرم‌افزارهای گشته است.  
به طور مثال، در سال ۲۰۱۴، آپ تلفن همراه توبیت به طور متوسط هر ۲۰ روز یک بار بهروزرسانی می‌شد.<sup>۶</sup>

<sup>213</sup>Accessibility

<sup>214</sup>Apple

<sup>215</sup>iPhone

<sup>216</sup>Smartphone

<sup>217</sup>Laptop

<sup>218</sup>Progressive Web Application

<sup>219</sup>Document Object Model

<sup>220</sup>Jasmine (JavaScript Testing Tool)

<sup>221</sup>Remote (Networking)

مزایا	معایب
<b>وبگاه واکنش‌گرا و یا «اول برای موبایل» («آپ چندصفحه‌ای»)</b>	<p>استفاده از همان ابزارها، چارچوب‌ها و زبان‌های برنامه‌نویسی مشابه برای آب‌های دسکتاپ قابل استفاده بر روی دستگاه‌های مختلف، پس نیازی به توسعه و نگهداری چند نسخه مجرأ نیست</p> <p>کاربر هرگز نیاز به نصب بروزرسانی‌ها ندارد با کمی تلاش، می‌توان آن را جوړی ساخت که حتی در صورت قطع اتصال به اینترنت بتوان از آن استفاده کرد<sup>۱</sup></p> <p>قرار دادن آیکن مربوط به آن بر روی صفحه اصلی دستگاه کاربر به عنوان یک نشانک وب</p>
<b>(PWA)</b>	<p>همان مزایا و معایب وبگاه واکنش‌گرا، اما حتی در صورت قطع ارتباط با اینترنت همچنان می‌تواند به خوبی عمل کند</p> <p><b>آپ اختصاصی سکوی اندروید (جاوا) یا آی اواس (آبجکتیو-سی)</b></p>

<p>نیاز به نصب و یادگیری یک سکو جدید به همراه محیط توسعه، چارچوب آزمون، و خط لوله به کاراندازی مربوط به آن برای بارگیری و نصب به موقع بروزرسانی‌ها باید به خود کاربران متکی بود</p> <p>باید از نسخه‌های قدیمی پشتیبانی کرد تا زمانی که اکثر کاربران نسخه نصب شده خود را بروزرسانی کرده باشند</p> <p>پشتیبانی از چندین سکو سخت افزاری ممکن است نیاز به حفظ و نگهداری چندین مجموعه کد داشته باشد</p>	<p>بهترین کارایی امکان فهرست شدن بر روی فروشگاه‌های نرم افزار کاربردی دسترسی تضمین شده به تمام امکانات سخت افزار آن سکو ممکن است تحت بارزی بدافزار قرار گیرد (سیاست‌های هر عرضه‌کننده و سکو متفاوت است)</p>
--	---

شکل ۱-۸: سه رویکرد برای پیاده‌سازی کلاینت تلفن همراه. امروزه، اکثرب قریب به اتفاق ویژگی‌های سکو تلفن همراه، از جمله فعالیت‌های بدون نیاز به ارتباط خارجی و احراز هویت مبتنی بر بیومتریک، از طریق استانداردهای باز و آزاد دنیای وب و همچنین از طریق محیط‌های برنامه‌نویسی اختصاصی سکوها در دسترس هستند.

### چکیده

- یک سند **HTML** (زبان نشانه‌گذاری ابرمنتی) از مجموعه‌ای از عناصر به صورت سلسله‌مراتبی تشکیل شده است. هر عنصر با یک **تگ** در <قلاوهای زاویه‌ای> شروع می‌شود که ممکن است دارای **خصوصیات اختیاری** باشد. برخی از عناصر در بین‌رینده محتوا هستند. به طور کلی، عناصر ساختار منطقی بخش‌های سند را توصیف می‌کنند، اما نه اینکه چگونه سند هنگام ریندر و نمایش روی صفحه نشان داده می‌شود.
- **شیوه‌نامه آبشاری (CSS)** یک زبان شیوه‌نامه است که ویژگی‌های بصری عناصر را در یک صفحهٔ وب توصیف می‌کند. یک شیوه‌نامه مجموعه‌ای از ویژگی‌های بصری را با انتخابگرهایی که با یک یا چند عنصر صفحه مطابقت دارند، پیوند می‌زند. راههای زیادی برای بیان انتخابگرهایی وجود دارد که با عناصر مختلف مطابقت دارند، اما رایج‌ترین آن‌ها این است که یک یا چند کلاس CSS را با عنصر مورد نظر پیوند بزنیم و انتخابگرهایی بنویسیم که عناصر را بر اساس کلاس مطابقت دهند.
- شیوه‌نامه‌های CSS از سندهای HTML جدا هستند و عناصر **link** در درون عنصر **head** یک سند HTML، یک یا چند شیوه‌نامه را با آن سند پیوند می‌زنند.
- دستگاه‌های تلفن همراه در حال حاضر بیشترین بازدید از آپهای **SaaS** را به خود اختصاص می‌دهند. یکی از راههای ساخت آپهای کلاینت «تلفن‌همراه‌محور» که به خوبی روی گوشی‌های هوشمند کار می‌کنند، استفاده از چارچوب‌های CSS مانند بوت‌استرپ است. این چارچوب‌ها بسته به نوع دستگاهی که HTML در آن مشاهده می‌شود، مجموعه‌های مختلفی از قوانین قالب‌بندی CSS را برای عناصر HTML یکسان ارائه می‌کنند.
- یکی دیگر از راههای ساخت آپهای کلاینت «تلفن‌همراه‌محور» ساخت آپهای قابل نصب و اختصاصی برای سکوهای مختلف است. این‌گونه آپ‌ها ممکن است امکان دسترسی به برخی از امکانات خاص دستگاه را بدنهند که از طریق HTML قابل دسترسی نیستند، اما مزایای مهم SaaS مانند حذف نیاز به نصب به روزرسانی‌ها و حفظ و نگهداری چندین مجموعه کد را نیز نفی می‌کنند.

### خودآزمایی ۱-۸-۱

- چگونه می‌توانید اطمینان حاصل کنید که شیوه‌نامه‌های CSS یکسانی برای همه صفحات و بگاه یا آپ شما استفاده می‌شود؟
- ◊ هر یک از سندهای HTML باید شامل پیوندهای مربوط به شیوه‌نامه‌های خود باشد، بنابراین شما می‌باشید مطمئن شوید که عناصر **<link>** یکسانی در عنصر **<head>** همه صفحات و بگاه یا آپ شما وجود دارد. ■

### ۹-۱ زیبائک در مقابل میراث کد

برای من برنامه‌نویسی فراتر از یک هنر عملی پراهمیت است. بلکه این یک مبادرت و سفری عظیم در شالودهٔ دانش است.

— گریس موری هاپر —<sup>۲۲۲</sup>

<sup>222</sup>Grace Murray Hopper

گریس موری هاپر (۱۹۰۶-۱۹۹۲) یکی از اولین برنامه‌نویسان بود. او اولین کامپیویلر را ساخت و با نام «گریس شگفت‌انگیز» شناخته می‌شد. او در نیروی دریایی آمریکا به درجه دریاداری رسید و در سال ۱۹۹۷ یک ناو جنگی به نام نام‌گذاری شد: یواس‌اس هاپر



برخلاف سخت افزار، از نرم افزار انتظار می رود تا در طول زمان دچار رشد و تکامل شود. درحالی که طراحی های سخت افزار پیش از ساخته شدن و ورود به بازار باید کامل شده باشند، طراحی اولیه نرم افزار می تواند به راحتی وارد بازار شده و در طول زمان ارتقا باید. اساساً، هزینه ارتقا بعد از ورود به بازار برای سخت افزار نجومی و برای نرم افزار قابل قبول و مقرر و مقرن به صرفه است.

بنابراین نرم افزار می تواند به یک جاودانگی فناورانه دست یابد، درحالی که سخت افزارها نسل به نسل کهنه شده و از بین می روند، یک نرم افزار ویژگی های جدید بنا به درخواست کاربران، تطبیق اشکالات باعث **تکامل نرم افزار** نیست، بلکه افزودن ویژگی های جدید بنا به درخواست کاربران، تطبیق با نیازهای در حال تغییر کسب و کار، بهبود کارایی<sup>223</sup> و سازگار شدن با محیط های جدید همگی از محركه های تکامل نرم افزار هستند. مشتریان نرم افزار انتظار دارند تا در طول زمانی که از یک نرم افزار استفاده می کنند، در مورد نسخه های جدید آن اطلاع پیدا کرده و بتوانند آن ها را نصب کنند؛ و یا حتی مشکلات نسخه های جدید را گزارش دهند تا توسعه دهنده این ها را تصحیح کنند. آن ها حتی گاهی مبلغ اضافی به صورت سالیانه تحت عنوان هزینه نگهداری پرداخت می کنند تا این امکان بهره مند شوند.

همانطور که نویسنده کان همواره امیدوارند تا آنچه خلق کرده اند و زاده تخلیشان است تا مدت ها خوانده شود که لقب کتاب کلاسیک را به خود بگیرد (چیزی حدود ۱۰۰ سال برای یک کتاب)، مهندسین نرم افزار هم باید امیدوار باشند که عمر چیزی که خلق کردند تا مدت ها ادامه یابد. البته نرم افزار نسبت به کتاب این برتری را دارد که در طول زمان می تواند بهبود یابد. در حقیقت، یک نرم افزار بادوام را معمولاً دیگران نگهداری می کنند و بهبود می بخشنده و مسئولیت را از دوش خالقین اصلی اثر بر می داردند.

**میراث کد**<sup>224</sup> به نرم افزاری اشاره دارد که با وجود قدیمی بودن، همچنان مورد استفاده قرار می گیرد چون پاسخگوی نیازهای مشتری است. شصت درصد هزینه های نگهداری نرم افزارها صرف افزودن قابلیت جدید به نرم افزارهای قدیمی یا میراث کدها می شود، درحالی که تنها ۱۷٪ هزینه ها برای رفع اشکالات خرج می شوند، پس این میراث کدها در واقع نرم افزارهای موقفي هستند.

لغف «میراث» یک بار منفی با خود دارد و نشان دهنده این است که تکامل کد به دلیل نازیبایی های موجود در طراحی و یا استفاده از فناوری های قدیمی دشوار است. در مقابل میراث کد، ما اصطلاح «زیبائُد»<sup>225</sup> را برای کدهای بادوام که به راحتی قابل بهبود هستند، به کار می بریم. با این حال توجه داشته باشید که این به معنی آن نیست که میراث کد بدترین حالت ممکن کد است، بلکه کد فوق العاده کمدوام است که به دلیل عدم توانایی پاسخگویی به نیازهای مشتری به سرعت دور ریخته می شود. ما مثال هایی که به زیبائُد می انجامند را با نماد تابلوی مونا لیزا نشانه گذاری می کنیم. به همین ترتیب، مثال های مربوط به میراث کد با چرتكه مشخص می شوند که یک وسیله محاسباتی قدیمی اما ماندگار است که در طول سال ها با تغییرات اندکی همراه بوده است.

در فصول بعدی کتاب مثال هایی از زیبائُد و میراث کد را به شما نشان می دهیم، به این امید که الهام بخش شما باشند تا طراحی های خودتان را جوړی انجام دهید که بعدها به راحتی بتوانند تکامل پیدا کنند. این مبحث به طور معمول در درس ها و کتاب های دانشگاهی نادیده گرفته می شود. ما به قدیمی، این کتاب به وجود پذیرش گسترده اهمیت بهبود میراث کدها و نرم افزارهای سه دلیل در این کتاب به این گونه نرم افزارها می پردازیم؛ اول اینکه شما می توانید هزینه ساخت یک برنامه را با یافتن کدهای موجود و باز کاربرد آن ها کاهش دهید. یک منبع خوب برای این کار، نرم افزارهای مت بن باز هستند. دوم اینکه آموختن روش تولید کدی که بتواند به راحتی توسط نسل های بعد از شما بهبود یابد، مفید است؛ از آنجایی که این مدل کد شناسی بیشتری برای دوام و داشتن عمری طولانی دارد. در آخر، برخلاف روش های طرح و ثبت، در مدل چاپک، شما از تکرار دوم به بعد به طور مداوم در حال بازبینی کد هستید تا آن را بهبود بخشدید و امکانات جدید به آن بیفزایید. بنابراین مهارت هایی که در مدل چاپک می آموزید دقیقاً همان هایی هستند که برای تکامل میراث کدها به آن ها نیاز دارید (فارغ از اینکه چگونه ساخته شده است) و این استفاده دوگانه از تکنیک های مدل چاپک، کار ما را

<sup>223</sup> Performance

<sup>224</sup> Legacy Code

<sup>225</sup> Beautiful Code

**قیمه‌ترین برنامه زنده** مورد استفاده احتمالاً موكا<sup>226</sup> یا MOCAS<sup>227</sup> (کوتاه نوشت عبارت مکانیزه کدن سرویس های مدیریت قرارداد) است که در سال ۱۹۵۸ توسط وزارت دفاع آمریکا خریداری و تا سال ۲۰۰۵ نیز همچنان از آن استفاده می شد.



**چرتكه** هنوز در بسیاری از مناطق جهان استفاده می شود، با اینکه هزاران سال عمر دارد.

برای پوشش میراث‌کدها در همین یک کتاب بسیار ساده‌تر می‌کند.

**چکیده:** نرم‌افزار موفق می‌تواند چندین دهه عمر کند و از آن انتظار می‌رود که تکامل پیدا کردد و بهبود یابد. برخلاف بخش سخت‌افزاری رایانه‌ها که در زمان تولید به کمال و مرحله پایانی طراحی رسیده است و پس از چند سال از رده خارج می‌شود. یکی از اهداف کتاب این است که به شما آموزش دهد چگونه شناسن تولید زیباگرد را بالا ببرید تا نرم‌افزار شما بتواند عمری طولانی و مفید داشته باشد.

### خودآزمایی ۱-۹-۱

برنامه‌نویسان به ندرت قصد نوشتمن کد بد را دارند. با توجه به ایده‌های مطرح شده در قسمت ۱-۵ در مورد بهره‌وری، به‌طور خلاصه توضیح دهید چگونه نرم‌افزاری که مدت‌ها پیش و هنگام نوشتمن شدنش باکیفیت محاسب می‌شده است، ممکن است امروز به عنوان نرم‌افزاری قدیمی که نگهداری آن بسیار دشوار است قلمداد شود.

به دلیل افزایش مداوم سطح تجرید ابزارهای نرم‌افزاری، امروزه توسعه‌دهندگان اغلب می‌توانند همان عملکرد را در خطوط بسیار کمتر (و زیباتری) از کد نسبت به چند دهه پیش ایجاد کنند. بنابراین در مقایسه با امکانات روز، نگهداری از کدهای قدیمی سخت‌تر است، حتی اگر در زمان نوشتمن آن‌ها ممکن است بسیار مدرن و آخرین فناوری روز تلقی شده باشند. بدون شک کدی که ما امروز می‌نویسیم در چند دهه دیگر به عنوان کدی قدیمی، منسخ و از مد افتاده در نظر گرفته خواهد شد! ■

### ۱۰-۱ گذرنی بر این کتاب و نحوه استفاده از آن

همانطور که در ابتدای این فصل در مفاهیم و پیش‌نیازها توضیح داده شد، برای تبدیل شدن به یک مهندس نرم‌افزار ماهر، علاوه بر درک مفهومی مطالب، تمرین عملی فراوان نیاز است. بنابراین، هدف ما در هر فصل این است که به شما مبانی مفهومی لازم را برای کار روی تمرین‌ها، جایی که یادگیری واقعی اتفاق می‌افتد، ارائه دهیم.

بقیه کتاب به دو بخش تقسیم شده است. بخش اول نرم‌افزار به صورت یک سرویس را توضیح می‌دهد و بخش دوم با تأکید بر روی مدل چاپک، به توضیح توسعه نرم‌افزار به‌شکل امروزی می‌پردازد. فصل ۳، بخش اول کتاب را با توضیحی در مورد معماری نرم‌افزارهای کاربردی SaaS آغاز می‌کند. این فصل همچنین به این موضوع می‌پردازد که چطور وب از مجموعه‌ای از صفحات ایستا<sup>۲۲۶</sup> به اکوسیستمی از سرویس‌هایی با مشخصه واسطه‌های برنامه‌نویسی «می‌بینی بر»<sup>۲۲۷</sup> REST تبدیل شد.

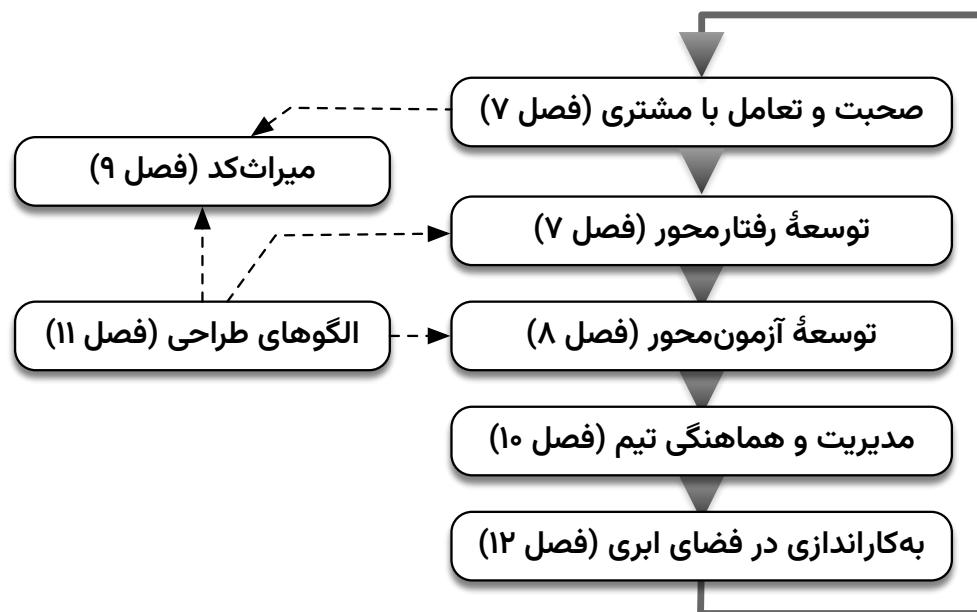
از آنجایی که زبان‌ها و چارچوب‌های برنامه‌نویسی به سرعت در حال تکامل هستند، ما باور داریم آموختن چگونگی یادگیری زبان‌ها و چارچوب‌های برنامه‌نویسی به صرفًا دانستن یک زبان و چارچوب برنامه‌نویسی خاص است. بدین ترتیب، فصل ۲ روش مدنظر ما را برای کسب این مهارت معرفی می‌کند. در این راه، ما از زبان برنامه‌نویسی رویی به عنوان مثالی برای آن دسته از برنامه‌نویسان که پیش از این با زبان برنامه‌نویسی مدرن دیگری همچون جاوا یا پایتون آشنایی دارند، استفاده می‌کنیم.

امروزه دلیل اصلی یادگیری یک زبان برنامه‌نویسی جدید اغلب میل به استفاده از یک چارچوب برنامه‌نویسی متنکی بر آن زبان است. یک چارچوب خوب هم یک معماری برنامه‌سازی خاص را تقویت می‌کند و هم از ویژگی‌های یک زبان خاص نهایت استفاده را می‌کند تا توسعه را در صورت انتباطق با آن معماری آسان کند. فصل ۴ به معرفی مفاهیم پایه‌ای چارچوب Rielz<sup>۲۲۸</sup> و قلب تپنده آن یعنی

<sup>226</sup>Static

<sup>227</sup>RESTful

<sup>228</sup>Rails (Simplified verion of Ruby on Rails)



شکل ۱-۹: یک تکرار از چرخه حیات نرمافزاری چاپک و رابطه آن با فصول این کتاب در بخش دوم، پیکانهای خطرچین نشاندهندۀ روابط فرعی و جانی بین مراحل یک تکرار هستند. در مقابل پیکانهای توپی، روند و جریان معمول را نشان می‌دهند. همانطور که بیشتر نیز شاره شد، فرایند چاپک هم بر روی نرمافزارهای کاربردی جدید و هم بر روی نرمافزارهای کاربردی قدیمی‌تر و میراث‌کدها قابل انجام است.

معماری مدل-نمای-کنترلگر<sup>۲۲۹</sup> می‌پردازد. فصل ۵ به بررسی امکانات پیشرفته‌تر این چارچوب می‌پردازد و همچنین با عمق بیشتری نشان می‌دهد که این چارچوب چگونه از ویژگی‌های زبان روبی نهایت استفاده را می‌کند. ما محتویات آموزشی مربوط به ریلز را به این علت بین دو فصل تقسیم کردیم تا کسانی که می‌خواهند هرچه سریع‌تر شروع به نوشتن یک آپ بکنند، تنها با مطالعه فصل ۴ بتوانند دست به کار شوند. اگر از پیش با روبی یا ریلز آشنایی دارید، می‌توانید از خواندن این فصول چشم‌پوشی کنید و یا صرفاً آن‌ها را سرسری مطالعه کنید.

با پیروی از همان روش مد نظرمان برای یادگیری زبان‌ها و چارچوب‌های جدید، فصل ۶ به معرفی زبان جاوااسکریپت، چارچوب چی‌کوئری و ابزار آزمون جَزْمِین می‌پردازد. قسمت توضیحات مربوط به جَزْمِین فرض می‌کند که شما آشنایی ابتدایی با فرایند آزمون دارید، پس اگر حس کردید که مطالب آن قسمت کمی برای شما جدید و ناآشنا هستند، می‌توانید مطالب آن قسمت را بعد از مطالعه فصل ۸ بررسی کنید. همانطور که چارچوب ریلز قدرت و بهره‌وری زبان روبی در تولید قسمت سرور آپ SaaS را تقویت می‌کند، چارچوب چی‌کوئری نیز قدرت و بهره‌وری زبان جاوااسکریپت را برای توسعه قسمت مربوط به کلاینت تقویت می‌کند.

با این پیش‌زمینه، شش فصل بعدی که بخش دوم کتاب را تشکیل می‌دهند، اصول مهم مهندسی نرمافزار برای ساخت و بهکاراندازی یک آپ SaaS را به کمک ابزارهای ریلز بیان می‌کنند. شکل ۱-۹ یک تکرار از چرخه حیات نرمافزاری چاپک را به تصویر کشیده است، که ما از آن به عنوان چارچوبی برای بنا کردن فصول بعدی کتاب استفاده می‌کنیم.

فصل ۷ نحوه تعامل و همکاری با مشتری را بررسی می‌کند. بنا بر توصیهٔ توسعه رفتارمحور (BDD)، توصیف موارد استفاده نرمافزار کاربردی باید به زبان ساده و غیرفنی باشد تا برای مشتریان بدون نیاز به دانش فنی قابل درک باشد. این کار با نوشتن روابیت‌های کاربری صورت می‌گیرد و در فصل ۷ خواهیم دید چگونه می‌توان این روابیت‌های کاربری را با استفاده از ابزار کیوکامبر



به آزمون‌های یکپارچگی و آزمون‌های پذیرش تبدیل کرد. این فصل همچنین به توضیح مفهوم و اصطلاح **سرعت** و نحوه استفاده از آن برای اندازه‌گیری پیشرفت پروژه با نرخ افزودن ویژگی‌ها می‌پردازد. برای دنبال کردن و محاسبه این مقدار، ابزاری به نام **پیوتوال ترکر** در این فصل نیز معرفی شود.

**فصل ۸، توسعهٔ آزمون محور (TDD)** را پوشش می‌دهد. این فصل نحوه نوشتگی کد خوب و آزمون‌پذیر<sup>۲۳۰</sup> را شرح می‌دهد و به معرفی یک ابزار آزمون به نام آر-اسپیک برای نوشتگی آزمون‌های واحد می‌پردازد. ابزار دیگری به نام سیمیل کاو<sup>۲۳۱</sup> برای اندازه‌گیری میزان پوشش آزمون نیز در این فصل معرفی می‌شود.

**فصل ۹** چگونگی برخورد با کدهایی را که از قبیل نوشتگی شده‌اند که شامل میراث‌کدها و بحث ارتقا آن‌ها نیز می‌شود، توضیح می‌دهد. این فصل توان می‌دهد که چطور از BDD و TDD هم برای فهم کد و هم بازسازی<sup>۲۳۲</sup> آن استفاده کنید. همچنین شما را با نحوه استفاده از ابزارهای کیوکامبر و آر-اسپیک که این کار را برای شما آسان‌تر می‌کنند آشنا خواهد کرد.

در **فصل ۱۰** با تکیه بر اصول اسکرام ذکر شده در بالا، توصیه‌هایی در مورد نحوه سازماندهی و کار به عنوان بخشی از یک تیم موثر و کارآرائه می‌شود. همچنین این فصل به تشریح سامانه کنترل نسخه گیت<sup>۲۳۳</sup> و سرویسی مربوط به آن با نام **گیت‌هاب** می‌پردازد تا نشان دهد چگونه اعضای یک تیم بدون دخالت در کار یکدیگر و ایجاد تداخل در روند تولید و انتشار محصول، می‌توانند بر روی ویژگی‌های مختلف یک نرم‌افزار کار کنند.

برای کمک به تمرين اصل «خودت را تکرار مکن»، **فصل ۱۱** به معرفی الگوهای طراحی می‌پردازد. الگوهای طراحی راه حل‌های ساختاری و اثبات‌شده برای مشکلات رایج در طراحی چگونگی ارتباط و همکاری کلاس‌ها با یکدیگر هستند. این فصل همچنین نشان می‌دهد چطور با به کارگیری امکانات زبان رویی می‌توان الگوهای طراحی را مورد استفادهٔ مجدد و بازکاربرد قرار داد. در این فصل رهنمودهایی نیز دربارهٔ نوشتگی کلاس‌های خوب داده شده است. این فصل شما را به اندازه کافی با **UML** (زبان مدل‌سازی یکپارچه) آشنا می‌کند تا بتوانید الگوهای طراحی را نشان دهید و نمودارهایی رسم کنید که نحوه کار کلاس‌ها را توضیح دهند.

توجه داشته باشید که **فصل ۱۱** برخلاف فصول پیشین بخش دوم کتاب که دربارهٔ فرایند توسعهٔ چابک هستند، در مورد معماری نرم‌افزار<sup>۲۳۴</sup> است. ما باور داریم که این ترتیب به شما کمک می‌کند که بتوانید یک تکرار چابک را سریع‌تر آغاز کنید و فکر می‌کنیم که هرچه تکرارهای بیشتری را انجام دهید، چرخهٔ حیات چابک را بهتر خواهید فهمید. با این حال همانطور که شکل ۹-۱ نشان می‌دهد، از آنجایی که الگوهای طراحی از اصول بنیادین در فرایندهای BDD و TDD هستند، تسلط بر آن‌ها برای نوشتگی و یا بازسازی کد بسیار مفید است.

**فصل ۱۲** توصیه‌های عملی ارائه می‌دهد چگونه ابتدا یک SaaS را در فضای ابری به کاراندازی کنیم و سپس کارایی و مقیاس‌پذیری آن را افزایش دهیم. این فصل همچنین به طور خلاصه به معرفی تعدادی از تکنیک‌های مربوط به امنیت<sup>۲۳۵</sup> و قابلیت اطمینان<sup>۲۳۶</sup> می‌پردازد که مختص به کاراندازی SaaS هستند.

ما کتاب را با یک پیش‌گفتار به پایان می‌رسانیم که نگاهی به مباحث مطرح شده در کتاب دارد و همچنین پیش‌بینی از آنچه ممکن است در آدامه رخ بدهد ارائه می‌دهد.

**پُفَک.** همانطور که کنفووسیوس<sup>۲۳۷</sup> گفت: «می‌شنوم و فراموش می‌کنم، می‌بینم و به



<sup>230</sup>Testable

<sup>231</sup>SimpleCov (Testing tool)

<sup>232</sup>Coverage (Testing)

<sup>233</sup>Refactoring (Code)

<sup>234</sup>Git (Version Control System)

<sup>235</sup>Unified Modeling Language

<sup>236</sup>Software Architecture

<sup>237</sup>Security (Computing)

<sup>238</sup>Reliability (Computing)

<sup>239</sup>Confucius (Chinese Philosopher)

یادم می‌ماند، انجام می‌دهم و می‌فهمم.» هدف این کتاب این است که به شما به اندازه‌ای اطلاعات بدهد تا به درک کافی از مفاهیم دست پیدا کنید تا بتوانید پروژه‌ها و فعالیت‌های عملی/کدنویسی (پُفَک<sup>۲۴۰</sup>) را که در متن این کتاب گنجانده شده‌اند، انجام دهید. هر تمرین پُفَک حاوی راهنمایی و نکات قابل توجهی برای خودآموزی است که برای تکمیل آن تمرین باید انجام دهید. اگر از این کتاب در کنار مطالب درسی آنلاین ارائه شده بر روی گُدیو استفاده می‌کنید (چه در محیط کلاس درس، چه به صورت خودآموز یا در دوره آموزش از راه دور [اِدْگُس](#)، جا به جای بین مطالب آموزشی محتوامحور (COD<sup>۲۴۱</sup>) و پروژه‌ها و فعالیت‌های عملی/کدنویسی (پُفَک) بسیار آسان است و تکالیف شما به طور خودکار برای شما نمره‌دهی می‌شوند. اساتید و خودآموزان، لطفاً برای اطلاعات بیشتر در مورد همه این گزینه‌ها به وبگاه این کتاب به آدرس [www.saasbook.info](http://www.saasbook.info) مراجعه کنید.

**واژه‌شناسی.** در حین فرو رفتن در این اکوسیستم غنی، با بسیاری از اصطلاحات فنی جدید (و واژگان باب روز<sup>۲۴۲</sup>) روبرو خواهید شد. برای کمک به شناسایی اصطلاحات مهم، متنی که به این صورت قالب‌بندی شده به عباراتی با مدخل‌های ویکی‌پدیا اشاره دارد. (در نسخه‌های کیندل<sup>۲۴۳</sup>، پی‌دی‌اف و گُدیو این کتاب، اصطلاحات مستقیماً به صفحه مرتبط در ویکی‌پدیا پیوند می‌خورند.) بسته به پیشینه شما، ما گمان می‌کنیم که لازم است برخی از فصول را پیش از یک بار بخوانید تا به طور کامل بر مطالب تسلط پیدا کنید.

در پایان هر فصل، یک قسمت تحت عنوان باورهای نادرست و خطرهای پنهان آورده شده است که تصویرات نادرست رایج یا مشکلاتی را توضیح می‌دهد که اگر هوشیار نباشید، به راحتی ممکن است در دام بیفتید و آن‌ها را تجربه کنید. پس از آن نکات پایانی برای ارائه منابعی برای کسانی که می‌خواهند عمیق‌تر در برخی از مفاهیم فصل تحقیق کنند، آورده شده است.

#### چکیده:

- مهندسی نرم‌افزار را فقط می‌توان با انجام دادن یاد گرفت و یادگیری از طریق انجام دادن به معنی صرفاً پیروی کردن از یک دستورالعمل و یا کمک کردن کدی از پیش نوشته شده نیست. متن این کتاب (COD) مبانی مفهومی کافی را به شما می‌دهد تا بتوانید بر روی پُفَک (پروژه‌ها و فعالیت‌های عملی/کدنویسی) کار کنید. هر دوی این‌ها برای یادگیری مطالب لازم هستند.
- اگر از این کتاب به همراه محیط توسعهٔ یکپارچه گُدیو استفاده می‌کنید (چه در محیط کلاس درس، چه به صورت خودآموز یا در دوره آموزش از راه دور [اِدْگُس](#)، تکالیف برنامه‌نویسی شما به طور خودکار برای شما نمره‌دهی می‌شوند و تمامی نرم‌افزارهای مورد نیاز از پیش نصب شده‌اند).
- هر فصل با فهرستی از مفاهیم و ایده‌های مهم آن فصل و همچنین دانش پیش‌نیاز فصل آغاز می‌شود.
- از باورهای نادرست و خطرهای پنهان غافل نشوید! حتی افراد متخصص نیز با آن‌ها برخورد می‌کنند، به همین دلیل است که یک قسمت از هر فصل را به خود اختصاص داده‌اند.

<sup>۲۴۰</sup> ما در این کتاب عبارت Coding/Hands-On Integrated Projects را به صورت «پروژه‌ها و فعالیت‌های عملی/کدنویسی» ترجمه کردیم و سرناام «پُفَک» را به عنوان معادل فارسی کوتاه‌نوشت CHIPS که در نسخهٔ اصلی کتاب مورد استفاده قرار گرفته است، در نظر گرفتیم.

<sup>241</sup> Content-Oriented Didactic

<sup>242</sup> Buzzword

<sup>243</sup> Kindle

**خودآزمایی ۱-۱۰**

کدام یک برای یادگیری سریع فرایند توسعهٔ SaaS از همگی مهمتر است: درک مبانی مفهومی، خواندن کد، یا نوشتن کد؟

◊ همگی مهم هستند. با کمی کردن کدی که دیگری نوشته چیز زیادی یاد نخواهید گرفت اگر ندانید که چرا آن کد کار می‌کند (یا نمی‌کند). از طرف دیگر، فقط خواندن درباره کدها منجر به کار کردن چیزی نمی‌شود. بررسی کدهای با کیفیت دیگران، که امیدواریم اساتید شما بر آن تأکید کنند، نه تنها نمونه‌های خوبی را به شما نشان می‌دهد، بلکه به تثبیت شدن درک شما از مبانی

مفهومی کمک می‌کند. ■

## ۱۱- باورهای نادرست و خطرهای پنهان

خداآندا به ما خردی عنایت بفرما که بتوانیم کلماتی آرام و محبت‌آمیز بر زبان بباوریم، شاید که فردا مجبور باشیم آن‌ها را مصرف کنیم!

—سناتور موریس اودال<sup>۲۴۴</sup>

همانطور که پیش‌تر اشاره شد، این قسمت که تقریباً در انتهای فصل قرار دارد، ایده‌های مطرح شده در این فصل را از دید دیگری بیان می‌کند تا خوانندگان بتوانند از استباهاتی که دیگران به آن‌ها دچار شدند، درس بگیرند. باورهای نادرست در واقع گفته‌هایی در رابطه با مباحث فعل هستند که قابل قبول به نظر می‌رسند (و یا حتی در بعضی موارد باور جمعی هستند) اما صحت ندارند. از سوی دیگر، خطرهای پنهان خطرات احتمالی در رابطه با مفاهیم هر فصل هستند که حتی با هشدار قبلی نیز ممکن است در دام آن‌ها بیفتند.



### باور نادرست: چرخه حیات چاپک بهترین روش برای توسعه انواع نرم‌افزار است

چرخه حیات چاپک برای استفاده در بسیاری از نرم‌افزارها، به ویژه SaaS مناسب است؛ که به همین دلیل ما از این روش در این کتاب استفاده می‌کنیم. با این حال، روش چاپک برای همه چیز بهترین نیست. برای مثال روش چاپک در مورد نرم‌افزارهای امنیتی و حساس ممکن است ناکارآمد باشد.

تجربه‌ای مانند نشان می‌دهد هنگامی که شما مراحل اساسی توسعه نرم‌افزار را فرا گرفته و تجربه مثبتی در استفاده از این مراحل با مدل چاپک داشته باشید، در تمامی پروژه‌های دیگر این اصول مهم مهندسی نرم‌افزار را فارغ از متداولویی مورد استفاده، رعایت می‌کنید. تمامی فضول در بخش دوم کتاب با مقایسه‌ای از دید روش طرح-و-ثبت تمام می‌شوند تا بهتر بتوانید این اصول را فرا بگیرید و همچنین بتوانید در صورت نیاز از سایر متداولویی‌ها استفاده کنید.

قطعاً روش چاپک آخرین چرخه حیات توسعه نرم‌افزاری نخواهد بود که شما خواهید دید، ما معتقدیم که متداولویی‌های جدید توسعه نرم‌افزاری نیز در پاسخ به شرایط و فرصت‌های جدید توسعه پیدا می‌کنند؛ پس آمادگی فرا گرفتن متداولویی‌ها و چارچوب‌های جدید را در آینده داشته باشید.



### خطر پنهان: نادیده گرفتن هزینه طراحی نرم‌افزار

از آجایی که تولید انبوه و نشر نرم‌افزار هزینه چندانی ندارد، این باور رایج به وجود می‌آید که تغییر آن نیز تقریباً بدون هزینه بوده و می‌توان آن را بر اساس خواسته‌های مشتری تغییر داد و «بازتولید» کرد. اما این دیدگاه هزینه طراحی و آزمون نرم‌افزار را که ممکن است قسمت عمده‌ای از کل هزینه پروژه نرم‌افزاری باشد، نادیده می‌گیرد. نداشتن هزینه تولید یکی از استدلال‌هایی است که برای منطقی جلوه دادن کپی کردن غیرقانونی نرم‌افزارها و سایر داده‌های الکترونیکی مورد استفاده قرار می‌گیرد. به نظر می‌رسد ناقضین حق تکثیر باور دارند که نباید پولی بابت توسعه داده شود و فقط تولید یک محصول فیزیکی را لایق دریافت پول می‌دانند.



### خطر پنهان: نادیده گرفتن زمینه تاریخی فناوری‌های نرم‌افزاری

کسانی که نمی‌توانند گذشته را به خاطر بیاورند محکوم به تکرار آن هستند.

—جورج سانتايانا<sup>۲۴۵</sup>

مهندسی نرم‌افزار یک رشته مهندسی نسبتاً جوان است، اما رشته‌ای است که به سرعت رو به سوی پیشرفت دارد. اگر سعی کنید فناوری‌های نرم‌افزاری را بیاموزید درحالی‌که زمینه‌های تاریخی را که شکل‌گیری آن‌ها نقش اند نادیده بگیرید، در خطر انتخاب‌های ناآگاهانه درباره ابزارهایی که باید استفاده کنید قرار می‌گیرید، و یا بدتر از آن، اصطلاحاً «چرخ را دوباره اختراع می‌کنید» بدون آنکه از تجربیات دیگران درس بگیرید. به عنوان مثال، اگر با همکاران خود در مورد مناسب بودن استفاده از نود چی‌اس<sup>۲۴۶</sup> به عنوان سرور نرم‌افزار کاربردی خود بحث می‌کنید، اما با بحث‌های طولانی مدت مربوط به «ریسمان‌ها<sup>۲۴۷</sup>» در مقابل رویدادها<sup>۲۴۸</sup> در جامعه توسعه‌دهندگان نرم‌افزارهای سیستمی آشنا نیستید، در بهترین حالت ممکن است یک بحث ناآگاهانه داشته باشید و در بدترین حالت به سرعت گرفتار بدیختی خواهد شد. بهطور مشابه، خرزش کاربرد<sup>۲۴۹</sup> در پایگاه‌های داده<sup>۲۵۰</sup> «NoSQL»<sup>۲۵۱</sup> منعکس‌کننده همان اتفاقاتی است که در نهایت منجر به ابداع **مدل رابطه‌ای**<sup>۲۵۲</sup> و چیره شدن آن بر **مدل سلسله‌مراتبی**<sup>۲۵۳</sup> می‌شود که پیش از آن معرفی شده بود و بهشدت شبیه به پایگاه‌های داده ساده و ابتدایی NoSQL است. اختراع مجدد چرخ همیشه لزوماً چیز بدی نیست. گاهی اوقات چرخ موجود واقعاً برای نیازهای شما مناسب نیست - همانطور که این گفته منتبه به داگلاس کراکفرد<sup>۲۵۴</sup> بیان می‌کند: «چیز خوبی که در مورد اختراق مجدد چرخ است، این است که می‌توانید یک چرخ گرد برای خود بسازید». ما امیدواریم که یادگیرندگان این مطالب چند دقیقه بیشتر وقت بگذراند تا دیدگاه جامعه‌تری به دست آورند؛ در مورد اینکه چرا چیزهای مختلفی که مورد بحث قرار می‌گیرند آن‌گونه که الان هستند پدید آمده‌اند (و یا چرا جور دیگری نیستند). ما بر این باوریم که این نه تنها به شما کمک می‌کند تا تصمیم بگیرید که آیا اختراق مجدد چرخ در حالت خاصی خوب است یا خیر، بلکه به شما کمک می‌کند تا از دلیستگی شدید به فناوری و پرستیدن آن اجتناب کنید. یعنی اینکه شما بخواهید یک فناوری جدید و به نظر جذاب و محبوب را مهم و بالرتش برای یادگیری تلقی کنید صرفاً به دلیل اینکه جدید (یا سریع یا سبک و یا هر چیز دیگری) است، بدون چشم‌اندازی مستدل و اگاهی از نقاط قوت و ضعف آن یا اینکه چگونه بر ایده‌های مشابهی که قبلًا بررسی شده‌اند بنا می‌شود.

### !! خطر پنهان: تمکز بیش از حد بر یادگیری سریع یک چارچوب خاص

فناوری‌های ساخت و توسعه نرم‌افزار با چنان سرعتی در حال تغییر و تکامل هستند که حتی چهار سال زمان هم کافی است تا چیزی که در گذشته به عنوان فناوری داغ و تازه<sup>۲۵۵</sup> شناخته می‌شد این عنوان را به چیزی جدیدتر و اگذار کرده باشد. در واقع، از اولین نسخه این کتاب در سال ۲۰۱۳ «فناوری داغ» برای ساخت قسمت پیشین یک آپ از پروتوتایپ<sup>۲۵۶</sup> به چی‌کوئری، سپس به آنگولا<sup>۲۵۷</sup>،

<sup>245</sup>George Santayana

<sup>246</sup>Node.js

<sup>247</sup>Thread

<sup>248</sup>Event (Computing)

<sup>249</sup>اصطلاح «خرزش کاربرد» و یا شکل کلی‌تر آن «خرش گستره»، به حالتی اشاره دارد که یک پروژه در طول زمان با افزودن کاربردهای زیادی از گستره تعریف شده اولیه آن فراتر می‌رود و مدیریت آن تا حدی از کنترل خارج می‌شود. در مورد پروژه‌های نرم‌افزاری این روند منجر به پیچیدگی پیش از حد نرم‌افزار و اصطلاحاً «نخ نرم‌افزار» می‌شود.

<sup>250</sup>Database

<sup>251</sup>Relational Model (Database Model)

<sup>252</sup>Hierarchical Model (Database Model)

<sup>253</sup>Douglas Crockford

<sup>254</sup>Prototype (JS Web Framework)

<sup>255</sup>Angular (Web Framework)

امیر جی اس<sup>۲۵۶</sup>، بکیون جی اس<sup>۲۵۷</sup> و ری اکت<sup>۲۵۸</sup> تغییر کرده است، و اکنون ویو جی اس<sup>۲۵۹</sup> یکی دیگر از مدعیان است. بنابراین، نویسنده‌گان این کتاب بر این باورند که آموختن چگونگی یادگیری چارچوبها و زبان‌های برنامه‌نویسی جدید ارزشمندتر است. انجام این کار با درک اصول اساسی طراحی و معماری نرم افزار، که زیربنای این چارچوبها و زبان‌های برنامه‌نویسی هستند، با کسب مداوم تسلط بر چارچوبها و ابزارهای متعدد، و با استفاده از رویکرد فراگیر و جهان‌شمول به این پرسش که کدام چارچوب یا زبان برنامه‌نویسی برای یک پروژه معین بهترین گزینه است.

## ۱۲-۱ نکات پایانی: مهندسی نرم افزار چیزی فراتر از برنامه‌نویسی است

نکات پایانی در انتهای هر فصل به خواننده کتاب چشم‌اندازی می‌دهد از آنچه در این فصل پوشش داده شده است: ایده‌ها یا نوآوری‌های فنی از کجا آمدند؟ با توجه به این تاریخچه، در مورد اینکه آن‌ها به چه سمتی می‌روند، چه چیزی می‌توانیم بگوییم؟ یک خواننده علاقه‌مند از کجا می‌تواند در مورد این موضوعات بیشتر بیاموزد؟ این قسمت‌ها هرگز حاوی محتوایی در مورد مهارت‌های فنی خاصی نیستند، بنابراین اگر عجله دارید، می‌توانید از خواندن آن‌ها صرف‌نظر کنید. اما اگر می‌خواهید یک متخصص کارکشته و یک طراح خوب نرم افزار و ابزار شوید، بهتر است این کار را انجام ندهید.

اگر برنامه‌نویسی مفرط صرفاً مجموعه‌ای جدید از همان شیوه‌های قدیمی است، پس چه چیز خاص و افراطی راجع به آن وجود دارد؟ جواب کنت یک<sup>۲۶۰</sup> این بود که برنامه‌نویسی مفرط تمامی اصول و شیوه‌های بدیهی و معقول را به سطوحی افراطی می‌رساند. برای مثال:

- اگر تکرارهای کوتاه خوب هستند، پس آن‌ها را نا آنچا که می‌شود کوتاه کنید. ساعت یا دقیقه یا ثانیه به جای روز و ماه و سال.

- اگر سادگی خوب است، همیشه سعی کنید ساده‌ترین چیزی را که شاید جواب دهد، انجام دهید.

- اگر انجام آزمون خوب است، پس همواره در حال انجام آن باشید. کد مربوط به آزمون را قبل از کدی که می‌خواهید مورد آزمون قرار دهید بتوانید.

- اگر بررسی و مرور کردن کد خوب است، پس به طور مداوم کد را مورد بررسی قرار دهید، به صورت دو نفره برنامه‌نویسی کنید، به شکلی که دو نفر باهم پای یک رايانه بنشينند و به صورت نوبتی کد بکدیگر را بررسی کنند.

– مایکل سواین<sup>۲۶۱</sup>، مصاحبه با کنت یک، (سواین ۲۰۰۱)

این نقل قول به خوبی منطق موجود در پس گونه برنامه‌نویسی مفرط (XP) از روش چاپک را که در این کتاب به آن پرداخته شده است، شرح می‌دهد. ما تکرارها را کوتاه می‌کنیم تا مشتری بتواند پیش‌نمونه‌های ناتمام از نرم افزار را که الیته کار می‌کند، در هر یک یا دو هفته یکبار ببیند. کد مربوط به آزمون‌ها را قبل از نوشتمن کد اصلی می‌نویسید و سپس کمترین کدی را می‌نویسید که بتواند آزمون را با موفقیت پشت سر بگذارد. برنامه‌نویسی دونفره یعنی کد تولید شده نه فقط در موقع خاص بلکه به طور مداوم مورد بازبینی و بررسی قرار می‌گیرد. روش چاپک تنها طی ده-دوازده سال از یک متدولوژی نرم افزاری نامتعارف (و مرتد) فراتر رفته و تبدیل به مدل غالب توسعه نرم افزار شده است؛ وقتی این مدل با معماري سرویس‌گرا ترکیب شود، می‌توان سرویس‌های پیچیده‌ای را با اطمینان خاطر تولید کرد.

با اینکه هیچ‌گونه وابستگی ذاتی بین SaaS، مدل چاپک و چارچوب‌های غنی مانند ریلز وجود ندارد، شکل ۱-۱۰ نشان می‌دهد که یک رابطه همکاری و هم‌افزایی خوبی در بین آن‌ها وجود دارد. توسعه چاپک به معنی پیشرفت مستمر در حین همکاری نزدیک با مشتری است و بر روی یک

<sup>256</sup> Ember.js (Web Framework)

<sup>257</sup> Backbone.js (Web Framework)

<sup>258</sup> React (JS Library)

<sup>259</sup> Vue.js (JS Web Framework)

<sup>260</sup> Kent Beck

<sup>261</sup> Michael Swaine





شكل ۱-۱۰: مثلث طلایی مهندسی و ساخت نرم افزار به صورت یک سرویس از سه جواهر درخشان مهندسی نرم‌افزار تشکیل شده است:  
 (۱) توسعهٔ چاپک بر روی بستر رایانش ابری، (۲) توسعه نرم‌افزاری چاپک و (۳) چارچوب‌ها و ابزارهایی کارا و غنی.

بستر رایانش ابری به مشتری این اجازه را می دهد تا در هر زمان و بی درنگ از آخرین نسخه نرم افزار استفاده کند، در نتیجه چرخه باز خورد تنگتر می شود و با خورد سریع تر صورت می گیرد (به فصول ۷ و ۱۲ رجوع کنید). بر روی بستر رایانش ابری با الگوی طراحی مدل-نمای-کنترلر گر (به فصل ۱۱ رجوع کنید) که توسط چارچوبهای غنی SaaS ارائه می شود، بسیار سازگار است (به فصل ۳، ۴ و ۵ رجوع کنید). ابزارها و چارچوبهای کارا و غنی که برای پشتیبانی از توسعه چابک طراحی شده اند، موانع را از سر راه عمل به مدل چابک برمی دارند (به فصول ۷، ۸ و ۱۰ مراجعه کنید). ما بر این باوریم که این سه «جواهر درخشان» که «مثلث طلایی» را تشکیل می دهند و همچنین شالوده اصلی این کتاب نیز از آن تشکیل شده است، به مهندسی و تولید نرم افزار به صورت یک سرویس زیبا منجر خواهد شد که بر طبق زمان بندی و بودجه تعیین شده اولیه انجام می گیرد.

مثلث طلایی کمک می کند تا ماهیت نوآورانه جامعه کاربران ریلز را بهتر توضیح دهیم. جامعه ای که مرتباً ابزارهای جدیدی را می سازد به راحتی بهره وری را بهبود می بخشد. ما تقریباً مطمئن هستیم که نسخه های آینده این کتاب شامل ابزارهایی خواهد بود که هنوز به وجود نیامده اند و آن قدر مفید هستند که نمی توانیم تصویر کنیم چطور زمانی کارمن را بدون آن ها پیش می بردیم! به عنوان مدرس و از آنچایی که بسیاری از دانشجویان روش های طرح-و-ثبت را بسیار کسالت آور می دانند، ما از اینکه پاسخ ها به ۱۰ پرسش اشاره شده در شکل ۱-۵، به شدت استفاده از روش چابک را برای انجام پروژه های دانشجویی توصیه می کند، خوشحالیم. با این حال، ما بر این باوریم که آشنایی با روش طرح-و-ثبت برای خوانندگان ارزشمند است چون که در مواردی این روش ممکن است گزینه بهتری باشد یا بعضی از مشتریان انجام کار را ملزم به استفاده از این روش می کنند و همچنین پرداختن به آن به توضیح بهتر روش چابک کمک می کند. بنابراین در اواخر هر فصل از بخش دوم کتاب، قسمتی را قرار داده ایم که مطالب مربوط به آن فصل را از منظر روش طرح-و-ثبت بیان می کند.

به عنوان پژوهشگر، ما معتقدیم که نرم افزارهای آینده به صورت روزافزون بر روی بستر زیرساخت های ابری ساخته و به آن ها وابسته می شوند. بنابراین از آنچایی که روش چابک سازگاری بسیار خوبی با این بسترها دارد، پیش بینی می این است که این روش در آینده بسیار محبوب تر خواهد شد. از این رو ما اکنون در نقطه زمانی خوبی قرار داریم که آینده جذابی هم برای یادگیری و هم برای آموزش توسعه نرم افزار در پیش رو داریم. چارچوبهای کارا و غنی مانند ریلز به شما کمک می کنند تا در زمان کوتاهی و با انجام دادن، ارزش این تکنولوژی را درک کنید. هدف اصلی ما از نوشتمن این کتاب این است که افراد بیشتری را از این فرصت استثنایی باخبر کنیم تا بتوانند از آن بهره مند شوند.

رایانش ابری فقط چند سال قبل از چاپ اول این کتاب وجود داشت و از آن زمان تاکنون به طرز چشمگیری تکامل یافته است. خوشة هایی از رایانه های معمولی مدت ها پایه اصلی بودند، اما رایانش ابری نحوه استفاده از آن خوشة ها را تغییر داد. تا اواخر دهه ۱۹۹۰، معمول بود که یک رایانه خاص به یک آپ SaaS خاص اختصاص داده شود که بر روی آن از پیش تمام پیش نیازها و اجزای نرم افزاری مورد نیاز برای اجرای آن آپ نصب شده باشد. در مقابل، از اواسط دهه ۲۰۰۰، فناوری ماشین مجازی این امکان را برای یک رایانه فیزیکی فراهم کرد که بتواند وانمود کند چندین رایانه است، به طوری که نرم افزارهای در حال اجرا بر روی هر رایانه مجازی باور داشتند که روی سخت افزار واقعی اجرا می شوند. مانند بسیاری دیگر از فناوری های مرتبط با SaaS، ماشین های مجازی برای دهه ها وجود داشتند - در این مورد، حداقل از دهه ۱۹۶۰ - اما کاهش هزینه های سخت افزار و سلطه معماری اینتل<sup>۲۶۲</sup> بر سرورها، ماشین های مجازی با کارایی بالا را به ابزاری کاربردی تبدیل کرد تا با استفاده از آن بتوان از چندین آپ SaaS مختلف بر روی یک رایانه میزبانی کرد، حتی اگر آن ها به سیستم های عامل و بسته های نرم افزاری متفاوتی نیاز داشته باشند. به طور معمول، یک آپ SaaS فقط به نوع ماشین مجازی که در آن اجرا می شود اهمیت می دهد و می تواند تا حد زیادی از جزئیات سخت افزار و سیستم عاملی که ماشین مجازی روی آن میزبانی می شود، بی اطلاع بماند. از اواسط دهه ۲۰۰۰، تکامل بیشتر فناوری ماشین مجازی منجر به چارچوبهای کانتینری سُبکی مانند داکر شد که بسته های نرم افزاری را از یکدیگر جدا و ایزو له می کند در حالی که یک هسته<sup>۲۶۳</sup> سیستم

262 Intel

263 Kernel (Operating System)

عامل واحدی را به اشتراک می‌گذارد. جدیدترین مرحله از مجازی‌سازی، تابع به صورت یک سرویس FaaS<sup>۲۶۴</sup> است، که در آن توسعه‌دهنده اکنون فقط کد یک یا چند تابع را مشخص می‌کند و به ازای فراخوان<sup>۲۶۵</sup> تابع هزینه پرداخت می‌کند. نمونه اولیه این مدل از مجازی‌سازی، سرویس لامبда<sup>۲۶۶</sup> از سرویس‌های وب آمازون<sup>۲۶۷</sup> است. اگرچه از FaaS با عنوان «رایانش بی‌سرور»<sup>۲۶۸</sup> نیز یاد می‌شود، البته واقعاً بدون سرور نیست، زیرا توابع باید به هر حال در جایی اجرا شوند. تمایز اصلی آن با SaaS<sup>۲۶۹</sup> است که توسعه‌دهنده‌گان با یک پیشته نرم‌افزاری<sup>۲۷۰</sup> متشکل از سرور آپ، سرور HTTP و غیره سروکار ندارند؛ آن‌ها فقط کد توابع را می‌نویسند. رایانش بی‌سرور هنوز در حال تکامل است و بسته به نوع آپی که قرار است پیاده‌سازی و به کاراندازی شود، هم مزايا و هم معایب دارد (کاسترو و دیگران<sup>۲۷۱</sup>).

در زبان برنامه‌نویسی محترم و ارجمند لیسپ، توابع به عنوان عبارات لامبدا شناخته می‌شند، زیرا این زبان بهشدت از چارچوب نظری حساب لامبدا الهام گرفته بود.

ما بر این باوریم که اگر مطالب این کتاب را یاد بگیرید و در کنار آن تکالیف و فعالیت‌های پیشنهادی (پیشک‌ها) را انجام دهید، همانطور که روش‌های معقول و اصولی مهندسی نرم‌افزار را می‌آموزید و آن‌ها را ذنبال می‌کنید، می‌توانید نسخه‌های (ساده‌شده) خودتان از سرویس‌های نرم‌افزاری محبوبی مانند فارم‌ویل<sup>۲۷۰</sup> یا توییتر را بسازید. اگرچه توانایی ساخت سرویس‌های شبیه به سرویس‌های موجود و به کاراندازی آن‌ها در فضای ابری در عرض چند ماه امری بسیار فوق‌العاده است، ما بیشتر هیجان‌زده هستیم تا ابداعاتی را که شما با بهره‌گیری از این مجموعه مهارت‌های جدید به وجود می‌آورید بینیم. ما مشتاقانه منتظریم که زیبائدهای شما و تبدیل شدن آن‌ها به کدهایی بادوام را بینیم و یکی از طرفداران پر و پا قرص آن‌ها شویم!

Christopher Alexander, Sara Ishikawa, and Murray Silverstein. *A Pattern Language: Towns, Buildings, Construction* (Cess Center for Environmental). Oxford University Press, 1977. ISBN 0195019199.

Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM (CACM)*, 53(4): 50–58, April 2010.

Luiz Andre Barroso and Urs Hoelzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines (Synthesis Lectures on Computer Architecture)*. Morgan and Claypool Publishers, 2009. ISBN 159829556X. URL <http://www.morganclaypool.com/doi/pdf/10.2200/S00193ED1V01Y200905CAC006>.

Sharon Begley. As Obamacare tech woes mounted, contractor payments soared. *Reuters*, October 17, 2013. URL <http://www.nbcnews.com/politics/politics-news/stress-tests-show-healthcare-gov-was-overloaded-v21337298>.

Jess Bidgood. Massachusetts appoints official and hires firm to fix exchange problems. *New York Times*, February 7, 2014. URL <http://www.nytimes.com/news/affordable-care-act/>.

Barry W. Boehm. Software engineering: R & D trends and defense needs. in Peter Wegner, editor, *Research Directions in Software Technology*, Cambridge, MA, 1979. MIT Press.

<sup>264</sup>Function as a Service

<sup>265</sup>Call (Function/Method)

<sup>266</sup>Lambda (AWS)

<sup>267</sup>Serverless Computing

<sup>268</sup>Software Stack

<sup>269</sup>Hypertext Transfer Protocol

<sup>270</sup>FarmVille (Online Game)

- Barry W. Boehm. A spiral model of software development and enhancement. in ACM SIGSOFT Software Engineering Notes, 1986.
- Eric Braude. *Software Engineering: An Object-Oriented Perspective*. John Wiley and Sons, 2001. ISBN 0471692085.
- Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. The rise of serverless computing. *Communications of the ACM (CACM)*, 62(12), Dec 2019.
- Robert Charette. Why software fails. *IEEE Spectrum*, 42(9):42–49, September 2005.
- Luke Chung. Too big to fire: How government contractors on HealthCare.gov maximize profits. *FMS Software Development Team Blog*, December 7, 2013. URL <http://blog.fmsinc.com/too-big-to-fire-healthcare-gov-government-contractors>.
- Michael Cormick. Programming extremism. *Communications of the ACM*, 44(6): 109–110, June 2001.
- Eric Enge. Mobile vs desktop usage in 2018: Mobile takes the lead. Stone Temple Consulting, Apr 2018. URL <https://www.stonetemple.com/mobile-vs-desktop-usage-study>.
- H-Christian Estler, Martin Nordio, Carlo A. Furia, Bertrand Meyer, and Johannes Schneider. Agile vs. structured distributed software development: A case study. in *Proceedings of the 7th International Conference on Global Software Engineering (ICGSE'12)*, pages 11–20, 2012.
- ET Bureau. Need for speed: More it companies switch to agile code development. *The Economic Times*, August 6, 2012. URL [http://articles.economictimes.indiatimes.com/2012-08-06/news/33065621\\_1\\_thoughtworks-software-development-iterative](http://articles.economictimes.indiatimes.com/2012-08-06/news/33065621_1_thoughtworks-software-development-iterative).
- Martin Fowler. The New Methodology. *martinfowler.com*, 2005. URL <http://www.martinfowler.com/articles/newMethodology.html>.
- Elizabeth Harrington. Hearing: Security flaws in Obamacare website endanger AmericansHealthCare.gov. *Washington Free Beacon*, 2013. URL <http://freebeacon.com/hearing-security-flaws-in-obamacare-website-endanger-americans/>.
- Scott Horsley. Enrollment jumps at HealthCare.gov, though totals still lag. *NPR.org*, December 12, 2013. URL <http://www.npr.org/blogs/health/2013/12/11/250023704/enrollment-jumps-at-healthcare-gov-though-totals-still-lag>.
- Alex Howard. Why Obama's HealthCare.gov launch was doomed to fail. *The Verge*, October 8, 2013. URL <http://www.theverge.com/2013/10/8/4814098/why-did-the-tech-savvy-obama-administration-launch-a-busted-healthcare-website>.
- Clay Johnson and Harper Reed. Why the government never gets tech right. *New York Times*, October 24, 2013. URL [http://www.pmi.org/en/Professional-Development/Career-Central/Must\\_Have\\_Skill\\_Agile.aspx](http://www.pmi.org/en/Professional-Development/Career-Central/Must_Have_Skill_Agile.aspx).

Jim Johnson. The CHAOS report. Technical report, The Standish Group, Boston, Massachusetts, 1995. URL <http://blog.standishgroup.com/>.

Jim Johnson. HealthCare.gov chaos. Technical report, The Standish Group, Boston, Massachusetts, October 22, 2013a. URL [http://blog.standishgroup.com/images/audio/HealthcareGov\\_Chaos\\_Tuesday.mp3](http://blog.standishgroup.com/images/audio/HealthcareGov_Chaos_Tuesday.mp3).

Jim Johnson. The CHAOS manifesto 2013: Think big, act small. Technical report, The Standish Group, Boston, Massachusetts, 2013b. URL <http://www.standishgroup.com>.

Capers Jones. Software project management practices: Failure versus success. *CrossTalk: The Journal of Defense Software Engineering*, pages 5–9, October 2004. URL <http://cross5talk2.squarespace.com/storage/issue-archives/2004/200410/200410-Jones.pdf>.

J. M. Juran and F. M. Gryna. *Juran's quality control handbook*. New York: McGraw-Hill, 1998.

Philippe Kruchten. *The Rational Unified Process: An Introduction, Third Edition*. Addison-Wesley Professional, 2003. ISBN 0321197704.

Timothy Lethbridge and Robert Laganiere. *Object-Oriented Software Engineering: Practical Software Development using UML and Java*. McGraw-Hill, 2002. ISBN 0072834951.

National Research Council. *Achieving Effective Acquisition of Information Technology in the Department of Defense*. The National Academies Press, 2010. ISBN 9780309148283. URL [http://www.nap.edu/openbook.php?record\\_id=12823](http://www.nap.edu/openbook.php?record_id=12823).

Peter Naur and Brian Randell. *Software engineering*. Scientific Affairs Div., NATO, 1969.

J. R. Nawrocki, B. Walter, and A. Wojciechowski. Comparison of CMM level 2 and extreme programming. in *7th European Conference on Software Quality*, Helsinki, Finland, 2002.

Mark Pault, Charles Weber, Bill Curtis, and Mary Beth Chrissis. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley, 1995. ISBN 0201546647.

Gerald J Popek and Robert P Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, 1974.

Project Management Institute. Must-have skill: Agile. *Professional Development*, February 28, 2012. URL [http://www.pmi.org/en/Professional-Development/Career-Central/Must\\_Have\\_Skill\\_Agile.aspx](http://www.pmi.org/en/Professional-Development/Career-Central/Must_Have_Skill_Agile.aspx).

W. W. Royce. Managing the development of large software systems: concepts and techniques. in *Proceedings of WESCON*, pages 1–9, Los Angeles, California, August 1970.

Ian Sommerville. *Software Engineering, Ninth Edition*. Addison-Wesley, 2010. ISBN 0137035152.

Matt Stephens and Doug Rosenberg. *Extreme Programming Refactored: The Case Against XP*. Apress, 2003.

Michael Swaine. Back to the future: Was Bill Gates a good programmer? What does Prolog have to do with the semantic web? And what did Kent Beck have for lunch? *Dr. Dobb's The World of Software Development*, 2001. URL <http://www.drdobbs.com/back-to-the-future/184404733>.

Andrew Taylor. IT projects sink or swim. *BCS Review*, January 2000. URL <http://archive.bcs.org/bulletin/jan00/article1.htm>.

Frank Thorp. 'Stress tests' show HealthCare.gov was overloaded. *NBC News*, November 18, 2013. URL <http://www.nbcnews.com/politics/politics-news/stress-tests-show-healthcare-gov-was-overloaded-v21337298>.

Jeff Zients. HealthCare.gov progress and performance report. Technical report, Health and Human Services, December 1, 2013. URL <http://www.hhs.gov/digitalstrategy/sites/digitalstrategy/files/pdf/healthcare.gov-progress-report.pdf>.

## پیوند ها

[https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction\\_to\\_HTML#Guides](https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML#Guides)<sup>۱</sup>  
<http://www.youtube.com/watch?v=DeBi2ZxUZiM><sup>۲</sup>  
<http://www.youtube.com/watch?v=kYUrqdUyEpI><sup>۳</sup>  
<https://youtu.be/AP71VJphbSk><sup>۴</sup>  
<https://getbootstrap.com/docs/4.0/examples/navbars/><sup>۵</sup>  
<https://sensortower.com/blog/25-top-ios-apps-and-their-version-update-frequencies><sup>۶</sup>  
<https://www.w3.org/TR/2011/WD-html5-20110525/offline.html><sup>۷</sup>  
<http://developers.slashdot.org/story/08/05/11/1759213/c><sup>۸</sup>  
<https://aws.amazon.com/lambda><sup>۹</sup>



---

## بخش اول

# نرم افزار به صورت یک سرویس: چارچوبها و زبانها

---

# چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم

هرگز نیاز نیست که بهینه‌ترین کارایی را داشته باشید، کافی است [برنامه شما] به اندازه خوبی کارایی داشته باشد... برنامه‌نویسان بیش از حد درگیر [بهبود حداکثری] کارایی هستند.

—باربارا لیسکف، ۲۰۱۱

۱-۲	پیش‌درآمد: یادگیری چگونگی یادگیری چارچوب‌ها و زبان‌های برنامه‌نویسی . . .	۵۹
۲-۲	برنامه‌نویسی دونفره . . . . .	۶۳
۳-۲	معرفی روبی، یک زبان شی‌عکرا . . . . .	۶۵
۴-۲	اصطلاحات برنامه‌نویسی روبی: حالت شاعرانه، بلوک‌ها، نوع‌دهی اردکی . . .	۷۵
۵-۲	پُفک: آشنایی با روبی . . . . .	۸۲
۶-۲	چم‌ها و باندلر: مدیریت کتابخانه‌ها در روبی . . . . .	۸۲
۷-۲	باورهای نادرست و خطرهای پنهان . . . . .	۸۶
۸-۲	نکات پایانی: چگونه می‌توان یک زبان را با جست‌وجو در گوگل یاد (ن)‌گرفت . .	۸۸

**باربارا لیسکف (۱۹۳۹-)** یک از اولين زنان بود که در ایالات متحده آمریکا و در رشته علوم رایانه مدرک دکترا گرفت (در سال ۱۹۶۸) و در سال ۲۰۰۸ جایزهٔ تورینگ را برای نوآوری‌های اساسی در طراحی زبان‌های برنامه‌نویسی دریافت کرد. ابداعات او شامل نوع داده انتزاعی و پیمایشگرها است که هر دو در زبان برنامه‌نویسی روبی نقشی اساسی دارند.



## پیش‌نیازها و مفاهیم

مهندسان نرم افزاری که قادر به یادگیری سریع و استفاده از زبان‌ها و چارچوب‌های جدید نیستند، هر چند سال یک بار در معرض خطر از رده خارج شدن یا بیکار شدن هستند. این فصل بر ایجاد این مهارت‌ها تمرکز می‌کند و با توضیح زبان روبي از دیدگاه یک توسعه‌دهنده شروع می‌شود. در فصل ۶ ما همین فرایند را برای جواه‌اسکرپت تکرار می‌کنیم.

### پیش‌نیازها:

- شما باید تجربه کار با برخی از زبان‌های شی‌ء‌گرای (OO) مدرن، مانند جاوا یا پایتون را داشته باشید و بتوانید به راحتی از مفاهیم OO مانند متغیرها و مُتّدهای کلاس در مقابل متغیرها و مُتّدهای نمونه، مُتّدهای عمومی در مقابل خصوصی و غیره استفاده کنید.

• آشنایی با انجام عملیات پایه‌ای بر روی گردآوردها مانند عملیات‌هایی که در زبان‌های برنامه‌نویسی تابعی دیده می‌شوند و توسط زبان پایتون نیز به عاریت گرفته شده‌اند مفید است، اما الزامی نیست. به عنوان مثال، `map` یک تابع (یا **عبارت لامبدا**) و یک گردآور را به عنوان ورودی می‌گیرد و یک گردآور جدید را که از اعمال تابع بر هر عنصر از گردآور اصلی حاصل می‌شود، برمی‌گرداند. `filter` یک تابع که مقدار خروجی اش ارزش بولی دارد را به همراه یک گردآور می‌گیرد و گردآور جدیدی برمی‌گرداند. این گردآور جدید متشکل است از عناصری از گردآور اصلی که تابع برای آن‌ها مقدار درست (`true`) برمی‌گرداند.

• **عبارات باقاعدہ** (به اختصار `regexes` یا `regexp`) دنباله‌ای از کاراکترها هستند که یک الگوی جست‌وجو را تعریف می‌کنند. تمام زبان‌های برنامه‌نویسی مدرن از آن‌ها استفاده می‌کنند. می‌توانید از وبگاه [Rubular](#) برای تمرین عبارات باقاعدہ در روبي استفاده کنید.

### مفاهیم:

• یادگیری یک زبان برنامه‌نویسی و یادگیری یک چارچوب برنامه‌نویسی اغلب با هم همراه هستند: شما روبي را یاد می‌گیرید تا بتوانید از ریلز استفاده کنید. این بدان معنی است که شما باید سه چیز را یاد بگیرید: زبان جدید، ساختار یا مدل معماري که چارچوب برای نرم افزارهای کاربردی توصیه می‌کند (چگونگی هماهنگی بخش‌های پویای یک نرم افزار کاربردی با یکدیگر) و در آخر نحوه استفاده چارچوب از ویژگی‌های زبان برای ارائه دادن آن ساختار.

• یادگیری یک زبان جدید مستلزم درک سازوکارهای پایه‌ای آن برای شی‌ء‌گرایی و کپسوله‌سازی (کلاس‌ها، وراشت و ترکیب)، سازوکارهای پایه‌ای دستوری آن (متغیرها، مرسومات و قواعد نام‌گذاری، کنترل جریان) و نحوه مدیریت پیچیدگی (استفاده از فضاهای نام، کتابخانه‌ها، مدیریت کتابخانه‌ها و بسته‌ها) است.

• یکی از ارکان اصلی زبان روبي این است که **همه چیز شیء است و حتی عملیات «ابتدایی» مانند جمع زدن در قالب ارسال پیامی به یک شیء برای انجام کاری**، تعریف می‌شود.

• یادگیری اصطلاحات برنامه‌نویسی که یک زبان را منحصر به فرد می‌کند جنبه کلیدی تسلط بر زبان است. اصطلاحات برنامه‌نویسی فراگیر زبان روبي که کمتر در سایر زبان‌های مدرن رایج هستند عبارت‌اند از میکس‌این‌ها (نوع دهی اردکی) و بلوک‌ها (عبارات لامبدا ناشناس).

• یادگیری یک زبان برنامه‌نویسی تا حد زیادی به یادگیری کتابخانه‌های آن و نحوه مدیریت شان مربوط می‌شود. کتابخانه‌های روبي («جم‌ها») و ابزار باندلر این امکان را فراهم می‌کنند تا بتوان وابستگی‌های یک آپ به کتابخانه‌های متعدد را (که خود نیز وابستگی‌هایی دارند) به طور دقیق مشخص کرد.



## فصل ۲. چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم

## ۱-۲ پیش‌درآمد: یادگیری چگونگی یادگیری چارچوبها و زبان‌های برنامه‌نویسی

ما از اصطلاح پُشتَّه نرم‌افزاری برای اشاره به مجموعه‌ای از فن‌آوری‌ها—معمولًاً زبان‌ها، چارچوب‌ها و زیرسازمانه‌ها—استفاده می‌کنیم که در توسعه نوع خاصی از آپ استفاده می‌شوند. در بیشتر پُشتَّه‌های نرم‌افزاری قسمت مهم و اساسی در واقع چارچوبی است برای ساخت نوع خاصی از نرم‌افزار کاربردی با استفاده از یک زبان برنامه‌نویسی خاص و تکیه بر سایر قسمت‌های دربرگیرنده پُشتَّه نرم‌افزاری است. امروزه، اکثر توسعه‌دهندگان، زبان‌های برنامه‌نویسی جدید را برای این یاد نمی‌گیرند که صرفاً به طور مستقل از آن‌ها استفاده کنند بلکه به این دلیل که می‌خواهند از یک چارچوب یا پُشتَّه نرم‌افزاری خاصی استفاده کنند: آپ‌های تلفن همراه اختصاصی سکوی آی‌اواس<sup>۳</sup> به زبان آی‌جکتیو-سی<sup>۴</sup> نوشته می‌شوند؛ آپ‌های ری‌اکت به زبان جاوا‌اسکریپت نوشته می‌شوند؛ و زبان روی<sup>۵</sup> ۱۰ سال قبل از اینکه چارچوب کارا و غنی ریلز آن را سر زبان‌ها بیندازد و محبوب کند وجود داشت.

اما همانطور که شکل ۱-۶ نشان داد، پُشتَّه‌های نرم‌افزاری و چارچوب‌ها می‌آیند و می‌روند. از همین رو، رویکرد ما از یک ضرب المثل چینی‌الهام گرفته شده است:

به فردی یک ماهی بدهید تا یک روز او را سیر کرده باشید. به آن فرد ماهیگیری بیاموزید تا یک عمر او را سیر کرده باشید.

### – ضرب المثل چین

با پیروی از این توصیه، هدف ما این نیست که به شما یک ماهی بدهیم (شما را در کمترین زمان ممکن با یک چارچوب یا پُشتَّه نرم‌افزاری خاص آشنا کنیم) بلکه بیشتر به شما کمک کنیم خودتان ماهیگیری بیاموزید—با ارائه راهنمایی در مورد چگونگی یادگیری مفاهیم مشترک تا با استفاده از آن خودتان بتوانید به سرعت موارد جدید را یاد بگیرید. در این فصل و فصل بعدی، ما همچنین یک ماهی برای شروع در قالب زبان روی و چارچوب ریلز به شما خواهیم داد. در فصل ۶، همین کار را برای جاوا‌اسکریپت و چی‌کوئری انجام خواهیم داد.

به طور مشخص، ما بحث خود را در مورد یادگیری یک زبان جدید به خانواده زبان‌های برنامه‌نویسی دستوری<sup>۶</sup> و شی‌عکرا (OO)<sup>۷</sup> محدود می‌کنیم. علاوه بر روی و جاوا‌اسکریپت که در این کتاب به معرفی آن‌ها می‌پردازیم، زبان‌های دیگر این خانواده عبارت‌اند از جاوا، پایتون، سی‌پلاس‌پلاس، سی‌شارپ<sup>۸</sup>، پل، بی‌اکالا<sup>۹</sup>، لوا<sup>۱۰</sup>، تی‌سی‌آل<sup>۱۱</sup> (که به تیکل نیز معروف است) و ده‌ها زبان دیگر. همانطور که خواهید دید، از نقطه نظر یادگیری زبان‌های جدید، همه این زبان‌ها بسیار شبیه به هم هستند، زیرا همه آن‌ها رویکردی دستوری (خطی و گام به گام) را برای حل مسائل برنامه‌نویسی می‌پرورانند. علاوه بر این، همگی آن‌ها با کپسوله‌سازی<sup>۱۲</sup> داده‌ها به همراه عملیات روی آن داده‌ها امکانات نسبتاً مشابهی را برای مدیریت پیچیدگی‌ها فراهم می‌کنند.

مهندسين نرم‌افزار ماهر می‌توانند با تسلط بر دایره واژگان فنی متشکل از سه جزء اصلی زیر، به سرعت زبان‌های جدید و پُشتَّه‌های نرم‌افزاری یا چارچوب‌هایی را که از آن‌ها استفاده می‌کنند، بیاموزند:

۱- تفاوت‌های زبان را بیاموزید: بیشتر زبان‌های دستوری شی‌عکرا دارای ابزارآلات ساده‌ای برای

<sup>1</sup> Subsystem

<sup>2</sup> iOS

<sup>3</sup> Objective-C (Programming Language)

<sup>4</sup> Imperative (Programming)

<sup>5</sup> Object-Oriented

<sup>6</sup> C# (Programming Language)

<sup>7</sup> Scala (Programming Language)

<sup>8</sup> PHP (Programming Language)

<sup>9</sup> Lua (Programming Language)

<sup>10</sup> Tcl (Programming Language)

<sup>11</sup> Encapsulation (Objec-Oriented Programming)

پیش از این، آپ‌های تلفن همراه مبتنی بر سکو خاص، گاهی اوقات با عنوان «آپ‌های بوم» شناخته می‌شدند.

اگرچه که بسیاری از این زبان‌ها عناصری از سایر خانواده‌های زبان‌های برنامه‌نویسی (مانند زبان‌های ای‌سی‌پی‌جی و پل) را دربرمی‌گیرند، اما روش اصلی استفاده از آن‌ها به صورت دستوری است.

## فصل ۲. چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم

امکانات پایه‌ای برنامه‌نویسی (متغیرها<sup>۱۳</sup>، انواع<sup>۱۴</sup> داده، کنترل جریان<sup>۱۵</sup>)، بازکاربرد (وراثت، واسطه‌ها<sup>۱۶</sup>)، مدیریت پیچیدگی (ترکیب<sup>۱۷</sup>، وراثت، نهان‌سازی داده‌ها<sup>۱۸</sup>)، اشکال‌زدایی، و استفاده از کتابخانه‌ها (اضافه کردن<sup>۱۹</sup>، مدیریت بسته‌ها و وابستگی‌ها) هستند. چه اصطلاحات یا امکاناتی از این زبان با دیگر زبان‌های متدال متفاوت است؟ مدیریت کتابخانه‌ها، که پادگیری آن‌ها معمولاً زمان بیشتری را نسبت به یادگیری زبان به خود اختصاص می‌دهد، در این زبان به چه شکل صورت می‌گیرد؟

-۲- درک خوبی از معماری نرم‌افزار ضمنی مورد استفاده توسط چارچوب به دست بیاورید: نرم‌افزار کاربردی چه ساختاری دارد و یا مجموعه‌الگوهایی که توسط چارچوب تقویت می‌شوند چه هستند؟ قسمت‌های اصلی و «نیروی محركه» یک نرم‌افزار کاربردی که با استفاده از این چارچوب ساخته شده است چه هستند، و چیدمان آن‌ها چگونه بر نحوه فرمول‌بندی عملکرد یک نرم‌افزار کاربردی از نظر آن چارچوب تأثیر می‌گذارد؟

-۳- ویژگی‌ها و امکانات زبان را با ساختار چارچوب‌ها مرتبط کنید. چگونه سازوکارهای یک زبان در فراهم کردن معماری و الگوهای مورد استفاده در یک چارچوب به برنامه‌نویسان کمک می‌کنند؟ به طور مثال، همانطور که در ادامه خواهیم دید، چارچوب ریلز به شدت بر قراردادمحوری بهجای پیکربندی<sup>۲۰</sup> متکی است: اگر از قوانین خاصی برای نام‌گذاری پیروی کنید، نیازی به ارائه فایل‌های پیکربندی ندارید که (مثلاً) توضیح دهنده کدام کلاس در آپ شما واسطه دسترسی به کدام جدول<sup>۲۱</sup> پایگاه داده است. رویی با بهره‌گیری از بازتاب و فرآبرنامه‌نویسی از قراردادمحوری بهجای پیکربندی پشتیبانی می‌کند.



بنابراین، در اینجا برنامه هشت قسمتی خودمان را برای یادگیری یک زبان برنامه‌نویسی دستوری شی‌عکرا جدید ارائه می‌دهیم:

-۱- **انواع و نوع دهی**<sup>۲۲</sup>. بررسی نوع داده و یا نوع دهی در این زبان برنامه‌نویسی به چه صورت است؟ ایستا و یا (همچون رویی و جاواسکریپت) به صورت پویا؟ آیا این زبان از دسته زبان‌های برنامه‌نویسی با نوع دهی ضعیف<sup>۲۳</sup> است یا نوع دهی قوی<sup>۲۴</sup>؟ در زبان‌های سی‌پلاس‌پلاس و جاوا، نوع یک متغیر می‌باشد در زمان کامپایل<sup>۲۵</sup> اعلان شود و در طول اجرای برنامه نمی‌تواند تغییر کند (نوع دهی ایستا). همچنین فقط اشیایی از همان نوع و یا یک زیرنوع<sup>۲۶</sup> سازگار را می‌توان به آن متغیر منتسب کرد (نوع دهی قوی). به طور مثال در جاوای، در یک گستره<sup>۲۷</sup> یکسان و مشخص، نمی‌توان ابتدا به یک متغیر عددی صحیح انتساب داد و سپس به همان متغیر یک رشته<sup>۲۸</sup> را منتسب کرد. در زبان رویی همانطور که در ادامه خواهیم دید، یک متغیر در زمان کامپایل نوع مشخص و اعلان‌شده‌ای ندارد (نوع دهی پویا) و همچنین می‌توان به

<sup>12</sup>Variable (Programming)

<sup>13</sup>Type (Programming)

<sup>14</sup>Control Flow (Programming)

<sup>15</sup>Interface (Computing)

<sup>16</sup>Composition (OOP)

<sup>17</sup>Data Hiding (Programming)

<sup>18</sup>Importing (Programming)

<sup>19</sup>Convention Over Configuration (Design Paradigm)

<sup>20</sup>Table (Database)

<sup>21</sup>Typing (Programming)

<sup>22</sup>Weakly Typed (Programming Language)

<sup>23</sup>Strongly Typed (Programming Language)

<sup>24</sup>Compile Time

<sup>25</sup>Static Typing (Type Checking)

<sup>26</sup>Subtype (Polymorphism)

<sup>27</sup>Scope (Programming)

<sup>28</sup>String (Programming)

آن هر شیئی از هر نوعی را نسبت داد (نوع دهنی ضعیف). هرچند که این سیاست امکان خطاهای مربوط به نوع داده را افزایش می‌دهد، اما همچنین باعث فراهم شدن امکانی بسیار قدرتمند برای بازکاربرد کد می‌شود که میکس‌این‌ها مشابه واسطه‌ها در زبان جawa هستند، با این تفاوت که انعطاف‌پذیری بسیار بیشتری دارند.

#### **إسکالا زبان ترکیبی جالی است:**

زبانی است که نوع دهنی پویا دارد وی در عین حال نوع دهن قوی دارد، به طوری که استنبط نوع در زمان اجرا صورت می‌گیرد تا مشخص بشود که یک عبارت خاص مجاز است با خبر.

**۲- امکانات پایه‌ای.** انواع پایه‌ای داده (اعداد، رشته‌ها، گردآوردها<sup>۲۹</sup>، و غیره) چه هستند؟ قوانین و یا قواعد مرسوم برای نام‌گذاری چیزهای مختلف (متغیرها، توابع، کلاس‌ها، فضاهای نام<sup>۳۰</sup>، و غیره) چیست؟ سازوکارهای پایه‌ای برای انتساب متغیرها، گستره متغیرها و کنترل جریان چیست؟ چه روش‌هایی برای دستکاری و ایجاد تغییرات در رشته‌ها از جمله استفاده از عبارات **باقاعدۀ<sup>۳۱</sup> و الحق رشته‌ای<sup>۳۲</sup>** وجود دارد؟

**۳- مُتَّدِّهَا.** چطور متَّدِّهَا (توابع یا رُویه‌ها) تعریف و فراخوانی می‌شوند؟ چطور آن‌ها نام‌گذاری می‌شوند؟ مُتَّدِّهای کلاس<sup>۳۳</sup> (ایستا) چگونه از مُتَّدِّهای نمونه<sup>۳۴</sup> متمایز می‌شوند؟

**۴- تجرید و کپسوله‌سازی.** کلاس‌ها چگونه تعریف می‌شوند؟ وراحت، تعریف زیرکلاس‌ها<sup>۳۵</sup> و ترکیب کلاس‌ها چگونه است؟ مُتَّدِّه‌ها و متغیرهای نمونه<sup>۳۶</sup>، مُتَّدِّه‌ها و متغیرهای کلاس (ایستا)، واسطه‌ها و سایر موارد، هر یک به چه صورتی مشخص می‌شوند؟

**۵- اصطلاحات برنامه‌نویسی<sup>۳۷</sup>.** چه اصطلاحات برنامه‌نویسی این زبان را از سایر زبان‌های برنامه‌نویسی که شما با آن‌ها آشنایی دارید متمایز می‌کند، و چگونه می‌توان از آن‌ها استفاده کرد؟ چندین نمونه برجسته از این اصطلاحات برنامه‌نویسی در زبان روبی عبارت‌اند از: نمادها<sup>۳۸</sup> (مشابه رشته‌های تغییرنایپذیر<sup>۳۹</sup>)، بلوک‌ها<sup>۴۰</sup> (همچنین با عنوان عبارت لامبدا<sup>۴۱</sup> ناشناس و یا بستار<sup>۴۲</sup> شناخته، و به‌فور در روبی برای پیاده‌سازی پیمایشگرها<sup>۴۳</sup> استفاده می‌شوند)، و اصطلاحات برنامه‌نویسی مربوط به برنامه‌نویسی تابعی برای انجام عملیات بر روی گرداوردها.

**۶- کتابخانه‌ها.** این زبان چه امکاناتی برای مدیریت کتابخانه‌ها دارد؟ نام‌گذاری، اضافه کردن و استفاده از کتابخانه‌ها و توابع موجود در آن‌ها چگونه انجام می‌شود؟ چه ابزارهایی برای مدیریت بسته‌ها<sup>۴۴</sup> (که مدیریت وابستگی‌ها نیز نامیده می‌شود) وجود دارد تا با استفاده از آن‌ها بتوان اطمینان حاصل کرد؟ اطمینان از اینکه یک نرم‌افزار کاربردی بتواند توسط سایر توسعه‌دهنده‌گان و یا در محیط عملیاتی و نهایی برای مشتری بدون مشکل، به صورت تکرارپذیر، و با نسخه‌های صحیح همه کتابخانه‌های مورد نیاز (وابستگی‌ها) به کاراندازی شود. ما به این موضوع در فصل ۱۲ بازمی‌گردیم.

**۷- اشکال‌زادایی.** چه ابزارهایی برای اشکال‌زادایی موجود است؟ آیا امکان اشکال‌زادایی مستقیم یک برنامه در حال اجرا توسط یک اشکال‌زادای برهمنکنی وجود دارد؟ آیا می‌توانید برای توقف یک

<sup>29</sup>Collection (Abstract Data Type)

<sup>30</sup>Namespace (Programming)

<sup>31</sup>Regular Expression

<sup>32</sup>String Interpolation (Programming)

<sup>33</sup>Class Method (OOP)

<sup>34</sup>Instance Method (OOP)

<sup>35</sup>Subclass (OOP)

<sup>36</sup>Instance (OOP)

<sup>37</sup>Programming Idiom

<sup>38</sup>Symbol (Programming Primitive Data Type)

<sup>39</sup>Immutable (Programming)

<sup>40</sup>Block (Programming)

<sup>41</sup>Lambda Expression

<sup>42</sup>Closure (Programming)

<sup>43</sup>Iterator

<sup>44</sup>Package (Program Distribution)

## فصل ۲. چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم

برنامه‌های در حال اجرا و بازرسی یا تغییر وضعیت آن،  **نقطه‌های توقف<sup>۴۵</sup>** و نقطه‌های مراقبت<sup>۴۶</sup> (نقطه‌های توقف مبتنی بر مقدار داده) تنظیم کنید؟ آیا دسترسی آسان به یک کنسول<sup>۴۷</sup> برهم‌کنشی، **حلقه خواندن-ارزیابی-نمایش (REPL)**<sup>۴۸</sup>، یا سازوکار دیگری برای امتحان کردن تکه‌های کوچک کد به صورت برهم‌کنشی وجود دارد؟

- **آزمون.** چگونه می‌توان آزمون‌های خودکار ساخت و اجرا کرد؟ ما در اینجا اشاره مختصراً به این موضوع خواهیم داشت اما در فصل ۸ به تفضیل به آن می‌پردازیم.

با در نظر داشتن این عبارت «انجام می‌دهم و می‌فهمم» از کنفووسیوس، ما به دنبال پاسخ دادن به این پرسش هستیم: چه ابزارهایی موجود هستند که به توسعه‌دهنده تازه‌کار این امکان را بدهد تا به سرعت با ویژگی‌ها و امکانات زبان آشنایی پیدا کند و آن‌ها را امتحان کند، بدون نیاز به دنبال کردن مراحل پیچیده‌ای برای نصب و راه‌اندازی آن بر روی رایانه خود؟  
ما از مراحل گفته شده در بالا بهره خواهیم گرفت تا صرفاً به اندازه کافی زبان روبی را بیاموزیم که بتوانیم سریعاً شروع به یادگیری و به کارگیری چارچوب توسعه SaaS محبوبی به نام ریلز کنیم. اما مراقب این نکته باشید: درحالی‌که داشتن تجربه در زبان‌های دیگر می‌تواند به شما کمک کند تا سریع‌تر یک زبان جدید را یاد بگیرید، از این خطر پنهان برای نگاشت یک‌به‌یک ویژگی‌های زبانی به زبان دیگر دوری کنید. معمول است که دو زبان دارای برخی ویژگی‌های مشترک با حداقل برخی از ویژگی‌های مشابه باشند، اما اگر هیچ تفاوت مفهومی بین دو زبان وجود نداشت، یکی از این دو را زائد بداریم، در مقابل وسوسه پرسیدن اینکه «آیا فلان ویژگی در زبان روبی با بهمن ویژگی در پایتون با جاوا یکسان است؟» مقاومت کنید. به دنبال قیاس‌ها و شباهت‌ها باشید، اما انتظار همسان بودن و یکریختی را نداشته باشید.



### چکیده‌ای از نحوه یادگیری موثر یک زبان برنامه‌نویسی جدید:

- بیشتر زبان‌های دستوری شی‌عگرا از نظر فلسفی بیشتر شبیه هم هستند تا متفاوت. به همین علت، عناصر اساسی یک زبان جدید—انواع و نوع دهنده، امکانات پایه‌ای، تعریف متّد، کنترل جریان، تجزیه و کوپوله‌سازی—ممکن‌باشد راحتی قابل یادگیری هستند.
- البته باید توجه داشت یک توسعه‌دهنده، یک زبان جدید را برای استفاده از یک چارچوب خاص یاد می‌گیرد. بنابراین، باید انتظار داشته باشید که آن زبان احتمالاً ویژگی‌ها و امکانات خاصی دارد که آن را برای آن چارچوب مناسب می‌سازد. این امکانات یا اصطلاحات برنامه‌نویسی احتمالاً به‌وقور توسط چارچوبی که به‌طور مؤثر از زبان بهره می‌گیرد، مورد استفاده قرار خواهد گرفت. این امکانات خاص احتمالاً برای شما تازه و ناآشنا خواهد بود اما تسلط بر آن‌ها برای استفاده مؤثر از زبان و خوب بهکار گرفتن آن بسیار مهم است.
- در آخر اینکه استفاده مؤثر از یک زبان همچنین مستلزم یادگیری نحوه استفاده از امکانات اشکال‌زدایی و کتابخانه‌های آن، به ویژه نحوه مدیریت وابستگی‌ها در بین کتابخانه‌های متعدد وابسته به هم است.

<sup>45</sup>Breakpoint (Programming/Debugging)

<sup>46</sup>Watchpoint (Programming/Debugging)

<sup>47</sup>Console

<sup>48</sup>Read-Eval-Print Loop

### ■ بیشتر بدانیم: تجربه کار با زبان دیگر: شمشیر دولبه

مطالعه‌ای اخیر بر روی پست‌های منتشرشده در وبگاه استک اورفلو نشان داد (شرستا و دیگران ۲۰۲۲) درحالی‌که داشتن تجربه کار با سایر زبان‌های برنامه‌نویسی می‌تواند هنگام یادگیری زبان جدید مفید باشد، اما اگر زبان آموز سعی کند صرفاً برای هر مفهومی در زبان جدید به دنبال مفهومی مشابه در زبان‌های دیگری که می‌داند بگردد، این می‌تواند بیشتر باعث سدرگمی شود. اولاً، برخی از مفاهیم در زبان جدید ممکن است هیچ مفهومی متناظر مستقیمی در زبان‌های دیگر نداشته باشند؛ ما به چند مورد از این موارد در قسمت ۴-۲ خواهیم پرداخت. ثانیاً، دو زبان ممکن است مجموعه‌ای از امکانات مشابه و مشترک برای انجام یک سری کارهای خاص داشته باشند، اما چگونگی استفاده از آنها و نحوه دسترسی آنها توسط برنامه‌نویس به طور قابل توجهی تفاوت داشته باشد. بنابراین، درحالی‌که ما شما را تشویق می‌کنیم که از تجربیات خود با زبان‌ها و چارچوب‌های دیگر برای کمک به یادگیری روبی و ریلز استفاده کنید، مراقب باشید در این دام گیر نکنید که بخواهید صرفاً مشابه فلان زبان که به آن مسلط هستید به روبی کد بنویسید.

## ۲-۲ برنامه‌نویسی دونفره

صاحب‌هکننده: در گوگل، شما یک دفتر کار مشترک دارید و حتی با هم کد می‌نویسید. سانچی قماوات: ما معمولاً با یکدیگر می‌نشینیم، یکی از ما در حال تایپ کردن است و دیگری در حال مشاهده و دنبال کردن، و در حین کار ما همواره در حال صحبت کردن هستیم و با هم آیده‌پردازی می‌کنیم.

-صاحب‌هکننده با چف دین و سانچی قماوات، پدیدآورندگان مدل برنامه‌نویسی نگاشت‌کاها<sup>۴۹</sup> (هافمن ۲۰۱۳)

نام برنامه‌نویسی مفترط (XP)، که گونه‌ای از چرخهٔ حیات چاپک است که ما در این کتاب دنبال می‌کنیم، نشان‌دهنده فاصله و تفاوت شگرفی است که این روش با روش‌های سنتی توسعه نرم‌افزار دارد. یکی از اختیارات جدید در این دنیای قشنگ نوی نرم‌افزاری، برنامه‌نویسی دونفره است. هدف این کار بهبود کیفیت نرم‌افزار با تکیه بر استفاده از دو نفر برای توسعه یک کد است. درحالی‌که برنامه‌نویسی دونفره به عنوان عُرفی در بین تیم‌ها و توسعه‌دهندگان روش چاپک پدیدار شد، نویسنده‌گان این کتاب معتقدند که این روش برای تسريع یادگیری یک زبان و چارچوب جدید نیز مناسب است. به همین دلیل است که ما در این فصل که موضوع آن آموختن چگونگی یادگیری زبان‌ها و چارچوب‌های جدید است، به معرفی این روش برنامه‌نویسی می‌پردازیم.

اگرچه برنامه‌نویسی دونفره در حقیقت یک فرایند مهندسی نرم‌افزار در نظر گرفته می‌شود تا اینکه با یک زبان خاص مرتبط باشد، ما آن را در همین ابتدای کار معرفی می‌کنیم تا استفاده زودهنگام از آن و به ویژه هنگام یادگیری یک زبان جدید را تسويق کنیم. همانطور که از نام آن پیداست، در برنامه‌نویسی دونفره، دو توسعه‌دهنده از یک رایانه مشترک استفاده می‌کنند و هر یک نقش متفاوتی را ایفا می‌کنند:

- راننده مسئول نوشتن کد است و از نظر تاکتیکی در مورد چگونگی تکمیل کار فعلی فکر می‌کند و در هنگام نوشتن کد افکار خود را با فرد دیگر به اشتراک می‌گذارد..

- ناظر یا (اگر بخواهیم قیاس دنیای اتومبیل‌رانی را دقیق‌تر دنبال کنیم) نقشه‌خوان وظیفه نظارت و بررسی خطوط کد را در همان هنگام نوشته شدن دارد. ناظر همچنین به طور استراتژیک در مورد مشکلاتی که در آینده باید به آنها رسیدگی شود فکر می‌کند و به راننده پیشنهاداتی می‌دهد.

<sup>49</sup>MapReduce (Programming Model)

## فصل ۲. چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم



شکل ۲-۳: سارا می و جی آر بوینت در شرکت پیووتال لبز در حال برنامه‌نویسی دونفره. سارا در نقش رانده و جی آر در نقش ناظر هستند. اگرچه دو صفحه‌کلید در عکس مشاهده می‌شود، اما فقط سارا در حال نوشتن کد است. همانطور که در عکس سمت راست می‌بینید، رایانه گرفته شده‌اند (در پس زمینه قابل مشاهده هستند) پکسان هستند و هیچ نرم افزار جانی غیر از ابزارهای مورد نیاز برای برنامه‌نویسی دونفره بر روی آن‌ها نصب نشده است. رایانه‌های دیگری به دور از این میزها برای سایر کارها همچون ارسال رایانه‌های تعبیه شده‌اند. عکس توسط تونیا فاکس، با اجازه و تنشکر از پیووتال لبز.

به طور معمول، دو نفر در حین انجام کار به‌طور متناوب نقش‌هایشان را عوض می‌کنند. شکل ۱-۲، مهندسان شرکت پیووتال لبز<sup>۵۰</sup> (پیدیآورندگان پیووتال ترکر) را نشان می‌دهد که بیشتر روز را به برنامه‌نویسی دونفره مشغول هستند (مور ۲۰۱۱).

برنامه‌نویسی دونفره کاری مشارکتی و مبتنی بر همکاری دوجانبه است و دو نفر می‌باشد همواره در حال صحبت کردن با یکدیگر باشند. ذهن و تلاش هر دو فرد بر روی کار در پیش‌رو متتمرکز است و همکاری دونفره باعث می‌شود که احتمال پیروی از شیوه‌های توسعه خوب افزایش یابد. اما تنها در صورتی مؤثر است که هر دو همکار در سراسر فرایند مشترک<sup>۵۱</sup> بر روی یک چیز متمرکز باشند (رودریگز و دیگران ۲۰۱۷)؛ اگر یکی از آن دو ساخت باشد و یا اینکه مشغول بررسی رایانه‌های خود باشد، این دیگر برنامه‌نویسی دونفره نیست، بلکه این صرفاً دو نفر است که در نزدیکی یکدیگر نشسته‌اند. در واقع، طبیعی است که نقشه‌خوان بیشتر از رانده صحبت کند و دائمًا بازخورد بدهد. اگر یکی از همکاران در مورد گفتوگو کنند تا دوباره با یکدیگر هماهنگ شود. اگر این دو مدت زمان مکث کنند و در این مورد گفت‌وگو کنند تا دوباره با یکدیگر هماهنگ شود. اگر این دو بازخورد باتجربه دیگری از تیم توسعه‌دهندگان کمک و مشورت بگیرند.

برنامه‌نویسی دونفره یک اثر جانبی هم دارد که آن انتقال دانش بین دو نفر است. این دانش می‌تواند اصطلاحات برنامه‌نویسی، ترقندهایی در مورد ابزاری خاص، فرایندها و گرفتهای شرکت، خواسته‌های مشتری و غیره باشد. بنابراین، برای گسترش هرچه بیشتر دانش، برخی از تیم‌ها عمدها و مرتباً اعضای مرتبط با یک کار مشخص را تعویض می‌کنند تا در نهایت همه با هم فرصلت برنامه‌نویسی دونفره را پیدا کنند. به عنوان مثال، با پیروی از جفت شدن بی‌قید و شرط<sup>۵۲</sup>، در یک تیم چهار نفره، شش حفت متفاوت ایجاد می‌شود.

مطالعاتی که به مقایسه برنامه‌نویسی دونفره و برنامه‌نویسی تک‌نفره پرداخته‌اند حاکی از کاهش زمان توسعه و بهبود کیفیت نرم افزار هستند. به عنوان مثال، کاکبرن و ویلیامز ۲۰۱۵ کاهش ۴۰٪ تا ۲۰٪ درصدی در زمان را مشاهده کردند. آن‌ها همچنین مشاهده کردند که کد اولیه در ۱۵٪ بوده است. با این حال، در مجموع (وقتی زمان هر دو عضو جدگانه در نظر گرفته شود) کارها حدود ۱۵٪ بیشتر طول کشیدند در مقایسه با حالتی که کارها به صورت تک‌نفره انجام می‌گرفتند. تعداد کثیری از برنامه‌نویسان، آزمون‌گران و مدیران با ۱۰ سال تجربه کاری در مایکروسافت گزارش

**دبلیرت در مورد برنامه‌نویسی**  
 دونفره داستان مصور دبلیرت در  
 این دو<sup>۵۳</sup> داستان<sup>۵۴</sup> با نگاهی  
 طنزآلود به نقد برنامه‌نویسی  
 دونفره می‌پردازد.

<sup>50</sup>Pivotal Lab

<sup>51</sup>Promiscuous Pairing (Pair Programming)

دادند که برنامه‌نویسی دونفره برای آن‌ها به خوبی کار می‌کند و منجر به تولید کد با کیفیت بالاتری می‌شود (بیگل و ناگاپان ۲۰۰۸). تحلیلی بر روی مجموعه‌ای از مطالعات مربوط به برنامه‌نویسی دونفره به این نتیجه رسیده است که وقتی پیچیدگی کار برنامه‌نویسی کم است (مثلًاً کارهای تک امتیازی در مقیاس پیووتال ترکر)، برنامه‌نویسی دونفره سریع‌تر بیش می‌رود و وقتی پیچیدگی کار بالا است (کارهایی با ارزش سه امتیاز در همان مقیاس) کد با کیفیت بالاتری به دست می‌آید. در هر دو مورد اما، تلاش بیشتری در مجموع نسبت به برنامه‌نویسی تک‌نفره صرف شده است (هانی و دیگران ۲۰۰۹).

تجربهٔ پیووتال لبز در این زمینه حاکی از این است که این مطالعات ممکن است تأثیر منفی عوامل حواس‌پرتی در دنیای مدرن ما و گستردگی فزایندهٔ ارتباطات (رایانه، توبیت، فیسبوک و غیره) بر بهره‌وری را نادیده گرفته باشند. برنامه‌نویسی دونفره هر دو برنامه‌نویس را مجبور می‌کند تا ساعتها پشت‌سرهم به کار مورد نظر توجه کنند و بر آن متمرکز باشند. در واقع، کارمندان جدید در پیووتال لبز کاملاً خسته به خانه می‌روند، زیرا عادت نداشتند برای یک مدت طولانی روى یک کار تمرکز کنند.

حتی اگر برنامه‌نویسی دونفره تلاش بیشتری را بطلبید، می‌توان افزایش بهره‌وری حاصله از روش چاپک و ریلز را برای برنامه‌نویسی دونفره «خرج» کرد. نتیجه این کار و سپردن توسعه کد به دو نفر می‌تواند زمان عرضه یک نرم‌افزار جدید را کاهش دهد یا کیفیت محصول نهایی را بهبود بخشد. توصیه می‌کنیم حتماً برنامه‌نویسی دونفره را امتحان کنید تا ببینید آیا آن را دوست دارید یا خیر؛ برخی از توسعه‌دهندگان که عاشق آن هستند.

**چکیده:** وقتی که زمان شروع کدنویسی فرا می‌رسد، یک رویکرد برنامه‌نویسی دونفره است که نوید کیفیت بالاتر و زمان توسعه کمتر را می‌دهد، اما شاید منجر به هزینه‌های برنامه‌نویسی بیشتری بشود به دلیل اینکه کار توسط دو نفر انجام می‌گیرد. یکی از این دو نفر نقش راننده را به عهده می‌گیرد که وظیفه دارد از نظر تاکتیکی در مورد تکمیل کار فعلی فکر کند، و دیگری نقش ناظر را دارد که به طور استراتژیک در مورد چالش‌های پیش‌رو فکر می‌کند و به راننده پیشنهاداتی می‌دهد.

## خودآزمایی ۱-۲-۲

درست یا نادرست: تحقیقات نشان می‌دهد که برنامه‌نویسی دونفره سریع‌تر و ارزان‌تر از برنامه‌نویسی تک‌نفره است.

- ◊ نادرست. با وجود اینکه آزمایش‌های دقیقی هنوز صورت نگرفته تا متخصصان را راضی کند، و مشخص نیست که آیا این آزمایش‌ها فقدان حواس‌پرتی در برنامه‌نویسی دونفره در نظر گرفته‌اند یا خیر، اتفاق نظر فعلی محققان این است که برنامه‌نویسی دونفره از برنامه‌نویسی تک‌نفره گران‌تر تمام می‌شود (مقدار نفر-ساعت بیشتری برای هر کار نیاز است). ■

## خودآزمایی ۲-۲-۲

درست یا نادرست: یک زوج در نهایت به این می‌رسند که کدام یک راننده بهتری و کدام یک ناظر بهتری است، و بعد از آن به آن نقش‌ها پاییند می‌مانند.

- ◊ نادرست. یک جفت کارا و مؤثر نقش‌هایشان را به طور متناوب عوض می‌کنند، زیرا در نهایت این برای افراد سودمندتر (و البته لذت‌بخش‌تر) است که هم نقش راننده را داشته باشند و هم نقش ناظر را. ■

## ۳- معرفی روبی، یک زبان شی‌عگرا

روبی یک زبان مینیمالیستی است: درحالی‌که کتابخانه‌های غنی دارد، اما خود زبان سازوکارهای نسبتاً کمی دارد. فلسفهٔ کلی آن را می‌توان در «شی‌عگرایی مفرط» خلاصه کرد. دو اصل به شما کمک می‌کند تا به سرعت خواندن و نوشتن به زبان روبی را یاد بگیرید:

## فصل ۲. چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم

- همه چیز شیء<sup>۵۳</sup> است- حتی یک عدد صحیح- و به معنای واقعی کلمه این‌طور است که هر عملیات منجر به یک فراخوانی مُتُّد بر روی یک شیء می‌شود و هر فراخوانی مُتُّد مقداری را بر می‌گرداند.

- رویی همانند جاوا و پایتون دارای کلاس‌های متعارف است. اما بر خلاف خصوصیات عمومی<sup>۵۴</sup> در زبان جاوا یا متغیرهای نمونه<sup>۵۵</sup> در زبان پایتون، تنها مُتُّدهای نمونه- و نه متغیرهای نمونه- در خارج از کلاس قابل دسترسی هستند. به عبارت دیگر، تمامی دسترسی‌ها از خارج از کلاس به متغیرهای نمونه، باید از طریق **متدهای دسترسی**<sup>۵۶</sup> عمومی صورت گیرد. متغیرهای نمونه قادر مُتُّد دسترسی عمومی در واقع خصوصی<sup>۵۷</sup> هستند. (پایتون نیز از رویکرد مشابهی پشتیبانی می‌کند با این تفاوت که آن را اجباری نمی‌کند).

اکنون بباید بررسی زبان رویی را مطابق با عناصر مهمی که در پیش به آن‌ها اشاره کردیم (برنامه هشت قسمتی) و در پرتوی دو اصل بالا ادامه بدهیم.

**نوع داده، نوع دهنی و نام‌گذاری.** رویی زبانی است با نوع دهنی پویا، به این معنی که متغیرها نوع ندارند، اما اشیایی که این متغیرها به آن‌ها اشاره می‌کنند دارای نوع هستند. از همین رو، عبارت `x = 'foo'` در این زبان کاملاً مجاز است. همانطور که سطر اول در شکل ۲-۲ نشان می‌دهد، یک و یا دو علامت `@` پیش از نام متغیرهای نمونه یا کلاس (ایستا) قرار می‌گیرد. اما متغیرهای محلی<sup>۵۸</sup> شامل حروف می‌شوند و اثرب از علامت `@` دیگر نیست. هر سه نوع متغیر می‌باشد با حروف کوچک شروع شوند، و نگارش به سبک ماری (مثلًا `snake_case`) به نگارش شتری (مثلًا `camelCase`)<sup>۵۹</sup> ترجیح داده می‌شود و متداوی‌تر است. سطر دوم قاعده نام‌گذاری سایر موجودیت‌ها مانند کلاس‌ها و مقدارهای ثابت<sup>۶۰</sup> را نشان می‌دهد که همگی به جز متغیرهای سراسری<sup>۶۱</sup> (که البته بهتر است هرگز از آن‌ها استفاده نکنید) می‌باشد با حرف بزرگ شروع شوند، و کلاس‌ها به صورت `UpperCamelCase` نام‌گذاری می‌شوند. (بنابراین، هرچند که به بیان دقیق، نگارش `lowerCamelCase` برای متغیرهای محلی و متغیرهای محلی قانونی است، این نوع نگارش به صورت اکید توصیه نمی‌شود، زیرا از نظر بصری تشخیص آن از `UpperCamelCase` دشوار است و به دلیل سهولتی که یک اشتباه ساده نوشتاری می‌تواند اولی را به دو می‌تغییر دهد و باعث ایجاد خطای<sup>۶۲</sup> شود). فضاهای نام برای هر گونه موجودیت نام‌گذاری شده جدآگاهه است، به طوری که `@foo`, `foo`, `FOO`، و `$FOO` همگی محذا هستند و تداخلی با یکدیگر ندارند.

در یادگیری هر زبان جدید، یک چیز آزاردهنده مرتبط با نوع داده، به خاطر سپردن این نکته است که این زبان چگونه عبارات غیربولی را مورد ارزیابی بولی<sup>۶۳</sup> قرار می‌دهد. برخی زبان‌ها دارای انواع و مقادیر بولی خاص هستند، مانند `True` و `False` در زبان پایتون (که از نوع خاص `Bool` هستند)، `true` و `false` در زبان جاوا‌اسکریپت (نوع `boolean`), و `true` و `false` در زبان رویی (به ترتیب `TrueClass` و `FalseClass`). برای جلوگیری از سردرگمی و اشتباه شدن عبارات غیربولی با چنین الفاظ<sup>۶۴</sup> بولی، توسعه‌دهندگان اغلب برای توصیف ارزش یک عبارت غیربولی (که به صورت `e` در عبارت

<sup>52</sup>Object (Programming)

<sup>53</sup>Public (Programming)

<sup>54</sup>Instance Variable (OOP)

<sup>55</sup>Accessor Method

<sup>56</sup>Private (Programming)

<sup>57</sup>Local Variable (Programming)

<sup>58</sup>این نوع نگارش به دلیل نحوه قرارگیری حروف بزرگ در میان حروف کوچک و شباهت نوشtar به کوهان شتر، «نگارش شتری» نام گرفته است. «نگارش ماری» نیز نامی نسبتاً جدیدتر است که به دلیل شباهت این نوع نوشtar به مار، در مقابل با نگارش شتری نام‌گذاری شده است.

<sup>59</sup>Constant Value (Programming)

<sup>60</sup>Global Variable (Programming)

<sup>61</sup>Error (Computing)

<sup>62</sup>Boolean (Data Type)

<sup>63</sup>Literal (Programming)

شرطی... (e) if استفاده شده است) از کلمات **حقیقی**<sup>۶۴</sup> یا **کذب**<sup>۶۵</sup> استفاده می‌کنند. متأسفانه، این درست بودن در هر زبانی متفاوت و تا حد زیادی دلخواهی و بی‌قاعده است. در زبان روبی الفاظ **nil** و **false** کذب محسوب می‌شوند، اما **سایر مقادیر**، شامل عدد صفر، یک رشته خالی، یک آرایه<sup>۶۶</sup> خالی، و غیره، همگی حقیقی هستند. در مقابل، در زبان پایتون، صفر کذب است، اما رشته خالی حقیقی محسوب می‌شود. در زبان جاواسکریپت، صفر و رشته خالی هر دو کذب هستند، همانطور که مقادیر ویژه **undefined** و **null** نیز کذب هستند، اما آرایه خالی حقیقی نمی‌باشد. در زبان‌هایی که هم نوع بولی واقعی و هم عملگر منطقی یکتاپی نمی‌باشد (معمولًاً به شکل !) وجود دارد، نوشتن **x != !x** عبارت را مجبور می‌کند که مقداری بولی داشته باشد (مثلاً اگر **x** کذب باشد، آن‌گاه **x != !x** واقعاً مقداری از نوع بولی و به ارزش نادرست دارد).

### امکانات پایه‌ای.

شکل ۲-۲ قواعد، ساختارها و عناصر پایه‌ای زبان روبی را نشان می‌دهد. همانطور که مشاهده می‌کنید همه چیز خیلی ساده است و چیز خارج از انتظاری در بین این عناصر نیست. زبان روبی دارای مقادیر بولی ویژه‌ای می‌باشد (ردیف ۳) از جمله مقدار ویژه **nil** (تهی) است، که نتیجه معمول عملیاتی است که در غیر این صورت هیچ مقدار بازگشتی معنی‌داری از آن حاصل نمی‌شود، مانند جست‌وجوی یک کلید ناموجود در یک هش<sup>۶۷</sup> یا یک مقدار ناموجود در یک آرایه.

در زبان روبی مقدار مجازی برای نشان دادن «نتیجه خالی» وجود ندارد، برخلاف **none** در زبان پایتون و یا **null** در زبان جاواسکریپت. به عبارت دیگر، یک متغیر در جاواسکریپت که مقدار آن **null** است به این معنی است که متغیر به‌حای نشان دادن «نادرستی» از منظر بولی، صرف‌فا به هیچ چیز خاصی ارجاع نمی‌دهد، درحالی‌که **nil** در زبان روبی ممکن است نادرست بودن بولی و یا متغیری را که به چیزی اشاره ندارد نشان دهد.

زبان روبی علاوه بر رشته‌ها (ردیف ۴)، نوعی از داده به نام **نماد** (ردیف ۴) نیز دارد، به‌طور مثال **octocat**، که در واقع یک «نشانه» تغییرپذیر است که مقدارش همان خودش است. از این نوع معمولاً در جاهایی که نوع شمارشی<sup>۶۸</sup> نیاز است استفاده می‌شود، همانند نوع **enum** در زبان سی یا جاوا، هرچند که کاربردهای دیگری نیز دارد. نمادها رشته نیستند، ولی همانطور که در شکل نشان داده شده است، این دو نوع به راحتی می‌توانند به یکدیگر تبدیل شوند.

ردیف ۶ و شکل ۳-۲ پشتیبانی و امکانات سرراست زبان روبی برای استفاده از عبارات باقاعده و گرفتن نتایج مطابقت‌های آن‌ها را خلاصه می‌کند. با توجه به حجم پردازش متنی که توسط آپ‌های **SaaS** مدرن انجام می‌شود، تسلط بر عبارات باقاعده و یادگیری اینکه چگونه یک زبان جدید امکان استفاده از عبارات باقاعدۀ را فراهم می‌کند، برای برنامه‌نویسان ضروري است. گردآوردها (ردیفهای ۷ تا ۹: آرایه‌ها و هش‌ها) می‌توانند کلیدها و مقادیر مختلف را ترکیب کنند. به‌طور خاص هش‌ها که در زبان‌های دیگر آرایه‌های انجمانی<sup>۶۹</sup> یا جداول درهم‌سازی<sup>۷۰</sup> نیز نامیده می‌شوند، در روبی بسیار پرکاربرد هستند و همه جا حاضرند.

هر عبارت در زبان روبی مقداری را برمی‌گردد. انتساب‌ها مقدار سمت چپ خود را برمی‌گرداند، یعنی مقدار متغیر یا چیز دیگری که انتساب به آن اختصاص داده شده است.

**مُتّدّها.** تعریف یک متّد با **def method\_name(arg1,...,argN)** شروع و با **end** پایان می‌یابد. همه مُتّدّها مقداری را برمی‌گردانند؛ اگر یک متّد به صورت صریح از دستور **return** برای بازگرداندن مقداری استفاده نکند، آن‌گاه مقدار آخرین عبارت ارزیابی‌شده در متّد به عنوان مقدار بازگشتی این متّد بازگردانده می‌شود، که این خود همانطور که پیش‌تر گفته شد تعریفی مشخص دارد و در زبان روبی همه دستورات مقداری را برمی‌گردانند.

همه چیز در زبان روبی یک شیء تمام عیار است که نمونه‌ای از یک کلاس به شمار می‌آید، حتی یک عدد صحیح ساده. همچنین هر عملیاتی، بدون استثنای، با فراخوانی یک متّد بر روی یک شیء

<b>زنان</b>	<b>إسمال تاک</b>	، که الهام بخش
مدل شیء‌گرایی استفاده شده		
در روبی است، خود از زبان		
<b>سیمولو</b> ایده گرفته است که		
اولین زبان برنامه‌نویسی شیء‌گرا		
است و محتویات آن برای		
تللاش‌هایشان موفق به کسب		
چایزه تورینگ شدند.		

<sup>64</sup>Truthy (Programming)

<sup>65</sup>Falsy (Programming)

<sup>66</sup>Array (Data Structure)

<sup>67</sup>Hash (Ruby Collection)

<sup>68</sup>Enumeration (Programming)

<sup>69</sup>Associative Array (Data Structure)

<sup>70</sup>Hash map/table (Data Structure)

local_variable, @@class_variable, @instance_variable ClassName, CONSTANT, \$GLOBAL, \$global	۱- متغیرها ۲- مقدارهای ثابت
nil و false معادل مقدار بولی نادرست هستند؛ true و هر چیز دیگری (عدد صفر، رشته خالی و غیره) مقدار درست دارند.	۳- مقدارهای بولی
"string", 'also a string', %q{like single quotes}, %Q{like double quotes}, :symbol کاراکترهای خاص (\n) در رشته‌هایی که بین علامت نقل قول دو تایی آمده‌اند، با معادلشان جایگزین می‌شوند، اما نه هنگامی که مابین علامت نقل قول منفرد آمده‌اند.	۴- رشته‌ها و نمادها
@foo = 3 "Answer is #{@foo}"; %Q{Answer is #{@foo+1}} "hello" =~ /lo/ "hello".match(Regexp.new 'lo') a = [1, :two, 'three'] ; a[1] == :two h = {a =>1, 'b' =>"two"} ; h['b'] == "two" ; h.has_key?(:a) == true h = {a: 1, 'b': "two"}	۵- عبارات دستوری داخل رشته‌های بین علامت نقل قول دو تایی ۶- انطباق با عبارات باقاعده (شکل ۳-۲) ۷- آرایه‌ها ۸- هش‌ها ۹- هش‌ها (نوشتار دیگر، رویی ۱/۹ به بعد)
def method(arg, arg) ... end (برای داشتن تعداد آرگومان متغیر از *args استفاده کنید)	۱۰- متّد نمونه
def ClassName.method(arg, arg) ... end def self.method(arg, arg) ... end	۱۱- متّد کلاس (ایستا)
def setter=(arg, arg) ... end def boolean_method?(arg, arg) ... end def dangerous_method!(arg, arg) ... end	۱۲- نام‌های متّدهای خاص (استفاده از ? و ! در پایان نام متّدها اختیاری اما رایج و مصطلح است)

عبارات شرطی	تکرار (رجوع شود به قسمت ۴-۲)	استثنایها
if شرط unless (یا) شرط دستورها [ elseif شرط دستورها ] [ else دستورها ] end	while شرط (or until) دستورها end 1..upto(10) do  i  ... end 10..times do ... end collection.each do  elt  ... end	begin دستورها rescue AnError => e :AnError داشت یک استثنای از کلاس rescue داشت [ensure [این قسمت همواره اجرا می‌شود end

شکل ۲-۲: ساختارهای کنترلی و عناصر بایه‌ای زبان رویی، به همراه موارد اختیاری مربوط به آن‌ها در داخل [قلاب‌های چهارگوش]. دستورات (معمول) با خطوط جدید و یا (به ندرت) با علامت نقطه‌ویرگول (;) از هم جدا می‌شوند. تعریفگی اهمیتی ندارد. علاوه بر انواع پایه‌ای داده و انواع گردآوردهای معمول، هش‌ها (کلاس Hash رویی، که با عنوان آرایه‌های انجمنی یا جداول درهم‌سازی نیز شناخته می‌شوند) همه جا وجود دارند.

نماد	معنی و کاربرد					
*	صفر یا بیشتر					شمارش
+	یک یا بیشتر					
?	صفر یا یک بار					
^	شروع خط، همچنین عدم حضور در مجموعه					شروع و پایان، مجموعه، بازه و سازه کاراکترها
\$	پایان خط					
( )	گروه، همچنین مقدار منطبق این گروه را در روبی ثبت می‌کند					
[ ]	مجموعه‌ای از کاراکترها					
[x-y]	بازه‌ای از کاراکترها					
	(یا (بول))					
[^ ]	عدم حضور (خلاف) در مجموعه					
.	هر کاراکتری					
	(به جز کاراکتر خط‌جدید)					
\	برای انطباق کاراکترهای خاص، همچنین برای استفاده در کلاس‌ها					
i	با قرار دادن در انتهای الگو عدم حساسیت مودی (بزرگی/کوچکی حروف) فعال می‌شود					کلاسیفیک
\d	ارقام دستگاه ددهی به لاتین ([۰-۹])					
\D	جیزی به حجز ارقام ددهی به لاتین ([۰-۹])					
\s	کاراکترهای فاصله خالی (whitespace)					
\S	به جز کاراکترهای فاصله خالی					
\w	کاراکترهای حرفی عددی (کلمه‌ای) ([۰-۹_])					
\W	به جز کاراکترهای حرفی عددی (غیرکلمه‌ای) ([۰-۹_])					
\n	کاراکتر خط‌جدید					

شکل ۳-۲: همانند اکثر زبان‌های برنامه‌نویسی مدرن، روبی از عبارات باقاعدۀ سازگار با پرل (PCRE)، که معمولاً به اختصار تحت عنوانین `regexp(s)` یا `regexp(es)` نام PCRE از قابلیت‌های گسترده‌ای که زبان اسکریپتنویسی پرل برای اولین بار برای کار با عبارات باقاعدۀ ارائه کرد الهام گرفته شده است.

## فصل ۲. چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم

انجام می‌شود. عبارت `(obj meth)` می‌شود `obj` فراخوانی می‌کند. در این حالت به شیء `obj` گیرنده گفته می‌شود و از آن انتظار می‌رود تا پاسخگوی فراخوانی `meth` باشد. به عنوان مثال، عبارت `(class 5)` فراخوانی می‌شود `class` را بدون هیچ آرگومانی به شیء ۵ می‌فرستد. می‌شود `class` استفاده شده در این مثال کلاسی که شیء به آن تعلق دارد را برمی‌گرداند، که در این مورد `Fixnum` است.

همانطور که در قسمت بعدی با جزئیات بیشتری خواهیم دید، در زبان رویی این امکان وجود دارد تا پرانتزهایی که معمولاً در طرف لیست آرگومانها استفاده می‌شود را حذف کرد، و این کار منجر به ابهام در هنگام تجزیه<sup>۷۱</sup> نمی‌شود. بنابراین دستور `5.class` معادل `(5.class)` است.

علاوه بر این، از آن جایی که همه چیز در زبان رویی یک شیء است، نتیجه هر عبارت، طبق تعریف، چیزی است که می‌توانید متدهای دیگر را بر روی آن فراخوانی کنید. از این رو `Fixnum superclass(5.class)` با ارسال فراخوانی می‌شود `superclass` بدون هیچ آرگومانی به `Fixnum` (شیئی که نماینده کلاس است که ۵ به آن تعلق دارد)، به شما می‌گوید که آبرکلاس<sup>۷۲</sup> `Fixnum` چه کلاسی است. فراخوانی می‌شدها به سمت چپ عبارت برمی‌گردند، بنابراین مثال بالا را می‌توان به شکل `5.class.superclass` و بدون هیچ پرانتزی نوشت. این گونه نوشтар و زنجیره کردن می‌شدها در زبان رویی بسیار رایج و مصطلح است.

### ■ بیشتر بدانیم: بازتاب

این مثال نگاه اجمالی به امکان و توانایی جامع و گسترده زبان رویی در زمینه بازتاب، یا به عبارت دیگر توانایی از اشیا در مورد خودشان پرسیدن، می‌اندازد. عبارت `('class' respond_to?')` به شما می‌گوید که آیا شیء ۵ می‌تواند پاسخگوی می‌شود `class` باشد یا خیر. عبارت `5.methods` تمامی متدهایی را که شیء ۵ به آن‌ها پاسخ می‌دهد، از جمله آن‌هایی که توسط اعداد آن تعریف شده‌اند، فهرست می‌کند. عبارت `('+' method)` نشان می‌دهد که می‌شود + در کلاس `Fixnum` تعریف شده است، در حالی که `('ceil' method)` نشان می‌دهد که می‌شود `ceil` در کلاس `Integer` تعریف شده است، که جد کلاس `Fixnum` است.

همانطور که شکل ۴-۲ نشان می‌دهد<sup>۷۳</sup>، حتی عملیات ساده ریاضی و ارجاعات آرایه در واقع فراخوانی‌های می‌شود بر روی اشیایی (گیرنده‌ها) هستند. از این رو، مفاهیمی همچون **تبدیل نوع**<sup>۷۴</sup> به ندرت در زبان رویی کاربرد دارند: درحالی‌که شما قطعاً می‌توانید از `5.to_s` یا `5.to_i` برای تبدیل بین رشته‌ها و اعداد صحیح استفاده کنید، نوشتن `a+b` به معنای فراخوانی `+` بر گیرنده `a` است، بنابراین رفتار آن کاملاً به چگونگی پیاده‌سازی می‌شود `+ نمونه + توسط کلاس a` (یا یکی از اعداد<sup>۷۵</sup> و یا میکس این‌های آن) دارد. پس هر دو عبارت `3+2` و `"foo"+"bar"` در زبان رویی کاملاً مجاز هستند، با این تفاوت که عبارت اول `+` را مطابق با تعریفی که در کلاس `Numeric` (جد کلاس `Fixnum`) آمده است فراخوانی می‌کند، و اما عبارت دوم `+` را مطابق تعریف ارائه شده توسط کلاس `String` فراخوانی می‌کند. رویی بازها<sup>۷۶</sup> می‌شوند `method` نماینده `method` از کلاس `ClassName` را با `ClassName#method` نشان می‌دهند و می‌شوند کلاس (یا می‌شوند) از کلاس `ClassName` را با `ClassName.method` نشان می‌دهند. بنابراین می‌توان گفت که عبارت `3+2` منجر به فراخوانی `Fixnum#+` بر گیرنده ۳ می‌شود.

**تجزید و کپسوله‌سازی.** زبان رویی از وراثت مرسوم پشتیبانی می‌کند، به این صورت که `SubFoo<Foo` class به این معنی است که `SubFoo` یک زیرکلاس از `Foo` است. یک کلاس حداکثر می‌تواند از یک آبرکلاس وراثت کند (رویی فاقد وراثت چندگانه است)، و همه کلاس‌ها در نهایت از `BasicObject` که گاهی کلاس ریشه نیز نامیده می‌شود و هیچ آبرکلاسی ندارد وراثت می‌کند. مانند

یک کلاس رویی مانند `Fixnum` خود نمونه‌ای از `Class` است، که کلاسی است که نمونه‌های آن نیز کلاس هستند (اصطلاحاً یک **فراکلاس** است). اگر خود زبان‌های برنامه‌نویسی و خیلی اهل این جور چیزها نیاورید و سعی نکنید همین الان را کامل بفهمید، وگرنه سرتان درد خواهد گرفت.

شما می‌توانید این را با ارزیابی عبارات `('+' method)` و `"foobar".method(:+)` راستی‌آزمایی کنید.

<sup>71</sup>Parsing (Computing)

<sup>72</sup>Superclass (OOP)

<sup>73</sup>عبارت «قند نحوی» که ما در این کتاب به عنوان معادل فارسی **Syntactic Sugar** در نظر گرفته‌ایم به قواعد و دستورات خاصی در زبان‌های برنامه‌نویسی اطلاق می‌شود که برای آسان کردن کار برنامه‌نویسان در نظر گرفته شده است. این دستورات معمولاً منجر به واضح‌تر و مختصرتر شدن کد می‌شوند و به نوعی باعث «شیرین‌تر» شدن زبان برای برنامه‌نویسی می‌شوند.

<sup>74</sup>Type Casting/Conversion

<sup>75</sup>Ancestor (OOP)

<sup>76</sup>Rubyist

استفاده صريح از send	بدون قند	با قند
10.send(:modulo, 3)	10.modulo(3)	10 % 3
5.send(:+, 3)	5.+(3)	5+3
x.send(:==, y)	x.==(y)	x == y
a.send(:*, x).send(:+, y)	a.*(x).+(y)	a * x + y
a.send(:+, x.send(:*, y))	a.+(x.*(y))	a + x * y
(اولویت عملگرها حفظ شده است)		
x.send(:[], 3)	x.[](3)	x[3]
x.send(:[]=, 3, 'a')	x.[]= (3, 'a')	x[3] = 'a'
Regexp.send(:new, 'abc')	Regexp.new("abc")	%r{abc} ./abc/
str.send(:match, regex)	str.match(regex)	str =~ regex
regex.send(:match, str)	regex.match(str)	regex =~ str
Regexp.send(:last_match, n)	Regexp.last_match(n)	\$1...\$n (ذخیره مقدار منطبق)

شکل ۴-۲: ستون اول «قند نحوی» زبان روبی را برای عملیات معمول نشان می‌دهد. ستون دوم معادل همان عملیات را به صورت فراخوانی صريح مُنْد نشان می‌دهد. ستون سوم چگونگی انجام همان فراخوانی مشابه را با استفاده از مُنْد send زبان روبی، که یک رشته (به صورت مصطلح تر) یک نماد را به عنوان آرگومان می‌پذیرد، نشان می‌دهد.

**میکس‌این‌ها** که به زودی آن‌ها را توضیح خواهیم داد، می‌توانند یک فراخوانی مُنْد تعریف نشده را پاسخ‌گو باشند، پیش از اینکه به آبرکلاس واگذاری شود.

بسیاری از زبان‌هایی که از وراثت پشتیبانی می‌کنند، اگر یک شیء برای مُنْدی که در کلاس‌ش تعریف نشده است فراخوانی دریافت کند، فراخوانی به آبرکلاس‌ش منتقل می‌شود و به همین ترتیب تا زمانی که به کلاس رسیده برسد یا یک استثنای<sup>۷۷</sup> از نوع مُنْد تعریف نشده (undefined method) (اتفاق بیفتند، ادامه می‌یابد. مُنْد سازنده<sup>۷۸</sup> پیش‌فرض برای یک کلاس باید initialize نام داشته باشد، اما همیشه به صورت new Foo فراخوانی می‌شود—این یک ویژگی خاص از زبان روبی است. کلاس‌ها می‌توانند هم مُنْدهای کلاس (ایستا) و هم مُنْدهای نمونه داشته باشند. همچنین آن‌ها می‌توانند هم متغیرهای کلاس<sup>۷۹</sup> (ایستا) و هم متغیرهای نمونه داشته باشند. نام متغیرهای کلاس با @ و نام متغیرهای نمونه با @ شروع می‌شود. نام مُنْدهای کلاس و مُنْدهای نمونه اما تفاوتی ندارد و مشابه یکدیگر هستند.

احتمالاً بزرگترین شگفتی برای تازه‌واردانی که در مورد سازوکار کلاس زبان روبی یاد می‌گیرند این است که هیچ دسترسی مستقیمی به متغیرهای کلاس یا متغیرهای نمونه از خارج از کلاس وجود ندارد. در زبان‌های برنامه‌نویسی دیگر، می‌توان متغیرهای نمونه از یک کلاس را به صورت عمومی تعریف (اعلان<sup>۸۰</sup>) کرد، مانند خصوصیات در زبان جاوا. در زبان روبی، دسترسی به وضعیت و مقادیر درون یک کلاس و یا نمونه باید از طریق مُنْدهای مقدارگیر<sup>۸۱</sup> و مقدارده<sup>۸۲</sup> باشد، که به آن‌ها در کار هم مُنْدهای دسترسی نیز گفته می‌شود. شکل ۲-۵ مثال‌هایی از مقدارگیرها (خطوط ۱۰-۱۶ و ۱۲-۱۳) و مقداردها (خطوط ۱۳-۱۵): توجه داشته باشید که مُنْدهای مقدارده به صورت مرسوم دارای نام‌هایی هستند که به = ختم می‌شوند، که دستوری مانند خط ۱۳ را ممکن می‌کنند، و یک مُنْد نمونه ساده را که به متغیرهای نمونه دسترسی دارد (خط ۱۸) نشان می‌دهد. از نقطه نظر فراخوانی‌دهنده<sup>۸۳</sup> در خطوط ۱۳-۱۴، به هیچ وجه نمی‌توان گفت که آیا یک مُنْد صرفاً یک «لفاف‌پیچ»<sup>۸۴</sup> برای دسترسی به یک متغیر نمونه است (مثل title) و یا آن مُنْد مقدار بازگشتنی و نتیجه خود را با انجام محاسباتی تولید می‌کند (مثل full\_title). این انتخاب در طراحی، موضوع سخت‌گیرانه روبی را در مورد اصل دسترسی یکنواخت<sup>۸۵</sup> نشان می‌دهد. این اصل به یکی از جنبه‌های کپسوله‌سازی در برنامه‌نویسی

<sup>77</sup>Exception

<sup>78</sup>Constructor (OOP)

<sup>79</sup>Class Variable (OOP)

<sup>80</sup>Declaration

<sup>81</sup>Getter Method (OOP)

<sup>82</sup>Setter Method (OOP)

<sup>83</sup>Caller (Function/Method)

<sup>84</sup>Wrapper (Programming)

<sup>85</sup>Uniform Access Principle (Programming)

## فصل ۲. چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم

<https://gist.github.com/450367d90330578a8ee4d355979fc7d1>

```

1 class Movie
2   def initialize(title, year)
3     @title = title
4     @year = year
5   end
6   # class (static) methods - 'self' refers to the actual class
7   def self.find_in_tmdb(title_words)
8     # call TMDb to search for a movie...
9   end
10  def title
11    @title
12  end
13  def title=(new_title)
14    @title = new_title
15  end
16  def year ; @year ; end
17  # note: no way to modify value of @year after initialized
18  def full_title ; "#{@title} (#{@year})"; end
19 end
20
21 # A more concise and Rubyistic version of class definition:
22 class Movie
23   def self.find_in_tmdb(title_words)
24     # call TMDb to search for a movie...
25   end
26   attr_accessor :title # can read and write this attribute
27   attr_reader :year # can only read this attribute
28   def full_title ; "#{@title} (#{@year})"; end
29 end
30
31 # Example use of the Movie class
32 beautiful = Movie.new('Life is Beautiful', '1997')
33 beautiful.title = 'La vita e bella'
34 beautiful.full_title # => "La vita e bella (1997)"
35 beautiful.year = 1998 # => ERROR: no method 'year='

```

شکل ۲-۵: تعریف یک کلاس ساده در روی که نشان می‌دهد تنها راه دسترسی به متغیرهای نمونه از خارج از کلاس استفاده صریح از مُندهای مقدارگیر و مقدارده است، و البته اینکه روی میابندهای (خطوط ۲۷-۲۶) اراهه می‌دهد تا نیازی به تعریف تک‌تک مُندهای دسترسی به صورت صریح نباشد. به جای تعیین کردن و تمایز قائل شدن بین متغیرهای نمونه و متغیرهای کلاس «خصوصی» در مقابل «عمومی»، صرف کافی است که مُندهای دسترسی عمومی ( فقط خواندنی، فقط نوشتنی، یا خواندنی/نوشتنی) برای وضعیت و مقادیری که باید به صورت عمومی در دسترس باشند، فراهم کرد.

شیءگرا مربوط می‌شود: تعیین جزئیات پیاده‌سازی عملیات، وضعیت و مقادیر درونی یک شیء از خارج از آن شیء باید غیرممکن باشد. مراقب باشید! اگر به زبان‌های برنامه‌نویسی جاوا یا پایتون عادت دارید، به راحت (و البته به اشتباه) ممکن است که دستور آمده در خط ۳۳ را به عنوان انتساب به یک خصوصیت یا متغیر نمونه در نظر بگیرید، اما این فقط یک فراخوانی مُنده است و در واقع می‌تواند به صورت beautiful.send('title=', 'La vita e bella') باشد که هر متغیر نمونه‌ای که قبلًا مقداردهی نشده باشد، بی‌سر و صدا (بدون هیچ خطابی) به nil ارزیابی می‌شود.

**چکیده:**

- همه چیز در روبی یک شیء است، حتی انواع پایه‌ای داده مانند اعداد صحیح، اشیا در روبی نوع دارند، اما متغیرها که به آن‌ها اشاره می‌کنند فاقد نوع هستند.
- هر عملیات با فراخوانی یک متُد بر روی یک شیء انجام می‌شود؛ عبارت `b.a` به معنای «فراخوانی متُد `b` روی شیء `a`» است. در اینجا به شیء `a`، گیرنده گفته می‌شود و اگر نتواند فراخوانی متُد را پاسخ‌گو باشد، فراخوانی را به آبرکلاس خود واگذار می‌کند. به این فرایند جست‌وجوی متُد در گیرنده می‌گویند.
- هر دستور روبی عبارتی است که دارای یک مقدار تعریف شده و مشخص است (که ممکن است `nil` باشد).
- روبی به عنوان یک زبان برنامه‌نویسی شی‌عکرا امکان تعریف کلاس و وراثت تکی را به همراه متغیرها و متدهای نمونه و کلاس معمول در این دسته زبان‌ها دارد.
- اصطلاحات برنامه‌نویسی در زبان روبی شامل استفاده از نمادها، استفاده از آرگومان‌های مبتنی بر کلیدواژه برای متدها، و حالت شاعرانه است. حالت شاعرانه اجازه می‌دهد پرانتزهای اطراف آرگومان‌های متُد و آکولادهای اطراف هش را زمانی که کد حاصل از نظر دستوری بدون ابهام است، حذف کنید.
- نماد یک نوع داده مصطلح و رایج در زبان روبی است. نماد رشته‌ای تغییرناپذیر است که مقدار آن خودش است. نمادها معمولاً در روبی برای نشان دادن «خاص بودن» استفاده می‌شوند، مانند عضوی از مجموعه‌ای از انتخاب‌های ثابت بودن (مانند نوع شمارشی)، و یا در هنگام فراخوانی متدها و برای پاس دادن آرگومان‌ها نام‌گذاری شده و خاص (کلیدواژه‌ها) به آن‌ها استفاده می‌شود.
- زبان روبی دارای امکان و توانایی جامع و گسترده در زمینه بازتاب است، که به شما این امکان را می‌دهد تا از اشیا در مورد خودشان بپرسید.
- مثالی از فرابرنامه‌نویسی است: در زمان اجرا، کد جدیدی ایجاد می‌کند که این کد مربوط به مقدارگیرها و مقداردها برای یک متغیر نمونه است. این سبک از فرابرنامه‌نویسی در روبی بسیار رایج است.

**■ بیشتر بدانیم: در زبان روبی، تمام برنامه‌نویسی فرابرنامه‌نویسی است**

مثالی است از **attr\_accessor** مثالی است از **attr\_accessor** (ایجاد کد در زمان اجرا که متدهای جدیدی را تعریف می‌کند)، زیرا که **attr\_accessor** در واقع چیزی نیست که در خود زبان روبی تعریف و تعییه شده باشد، بلکه صرفاً یک فراخوانی متُد معمولی است که متدهای مقدارگیر و مقدارده را در لحظه تعریف می‌کند. یعنی اینکه `foo attr_accessor :foo` و `foo` را تعریف می‌کند که وظیفه مقداردهی به متغیر نمونه `@foo` یا گرفتن مقدار آن را دارند. در واقع، می‌توان گفت که **همه** برنامه‌نویسی به زبان روبی به نوعی فرابرنامه‌نویسی است، چون که حتی تعریف کلاسی که در شکل ۵-۲ آمده است، یک تعریف کلاس و یا اعلان مشابه چیزی که در زبان جاوا داریم نیست، بلکه در واقع کدی است که در زمان اجرا یک شیء جدید را به عنوان نماینده کلاس `Movie` ساخته و نام `Movie` را به آن پیوند می‌زند. از آنجایی که تعریف یک کلاس در زمان اجرا اتفاق می‌افتد، پس شما می‌توانید بعداً آن را تغییر بدهید، همانطوری که شکل ۶-۲ نشان می‌دهد.

**خودآزمایی ۱-۳-۲**

معادل عبارات زیر با استفاده صریح از `send` چیست: `x[0] = 'foo'`، `a == b`، `a < b`، `a[x[0]]` و `'foo'.`

## فصل ۲. چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم

<https://gist.github.com/8d6e400be9b2c2b73dd9635070114cb6>

```

1 # Time#now, Time#+ and Time#- represent time as 'seconds since 1/1/'70'
2 class Fixnum
3   def seconds ; self ; end
4   def minutes ; self * 60 ; end
5   def hours   ; self * 60 * 60 ; end
6   def ago     ; Time.now - self ; end
7   def from_now ; Time.now + self ; end
8 end
9 Time.now          # => 2018-11-22 16:58:04 +0100
10 5.minutes.ago    # => 2018-11-22 16:53:12 +0100
11 5.minutes - 4.minutes # => 60
12 3.hours.from_now # => 2018-11-22 19:58:45 +0100

```

شکل ۶-۲: باز کردن مجدد کلاس `Fixnum`، که یکی از کلاس‌های اصلی و مرکزی روبی است، و افزودن شش مُثُد نمونه جدید به آن، دستورهای زیبایی برای انجام محاسبات زمانی به دست می‌آوریم. (ریلز شامل نسخه کامل تری از این تسویه‌لات است.) یونیکس در سال ۱۹۷۰ ابداع شد، بنابراین طراحان آن تصمیم گرفتند زمان را به صورت تعداد ثانیه‌هایی که از نیمه شب ۱۹۷۰/۰۱/۰۱ به وقت گرینویچ (که به آن گاهی آغاز دوران می‌گویند) گذشته است نشان دهند.

■ `x.send(:[] =, 0, 'foo')`, `x.send(:[], 0)`, `a.send(:==, b)`, `a.send(:<, b)` ◇

### ۲-۳-۲ خودآزمایی

با استفاده از یک مفسر برهمنشی روبی راستی‌آزمایی کنید که خروجی عبارت ۵/۴ عدد ۱ می‌شود، اما خروجی `5/4.0` و `5.0/4` هر دو عدد ۱.۲۵ می‌شود. این رفتار را با مشخص کردن اینکه مُثُد / کدام کلاس در هر مورد فراخوانی می‌شود و اینکه فکر می‌کنید که چگونه آرگومان خود را بررسی می‌کند، توضیح دهد.  
 ◇ در `5/4` و `5.0/4`، مُثُد نمونه / مربوط به کلاس `Integer` بر روی گیرنده ۵ فراخوانی می‌شود. آن مُثُد اگر آرگومان آن نیز یک عدد صحیح باشد، تقسیم عدد صحیح را انجام می‌دهد، اما اگر آرگومان آن یک عدد اعشاری ممیز شناور باشد، گیرنده را نیز به یک عدد اعشاری ممیز شناور تبدیل می‌کند. و تقسیم ممیز شناور را انجام می‌دهد. در `5.0/4`، مُثُد / مربوط به کلاس `Float` فراخوانی می‌شود، که همیشه تقسیم ممیز شناور را انجام می‌دهد. ■

### ۳-۳-۲ خودآزمایی

چرا عبارت `movie.year=1998` معادل عبارت `movie.year=1998` نیست؟  
 ◇ دستوری به شکل `b.a.b` به این معنی است که «مُثُد `b` را بر روی گیرنده `a` فراخوانی کن»، اما اگر `@year` اسم یک متغیر نمونه است، در حالی که `year` اسم یک مُثُد نمونه است. ■

### ۴-۳-۲ خودآزمایی

فرض کنید که ما خط ۱۲ را از کد آمده در شکل ۵-۲ حذف کنیم. آن‌گاه نتیجه اجرای دستور `Movie.new('Inception', 2011).year` چه خواهد بود؟  
 ◇ این در زبان روبی منجر به خطأ خواهد شد که مُثُد `year` تعریف نشده است. ■

### ۵-۳-۲ خودآزمایی

در کدی که در شکل ۶-۲ آورده شده است، آیا `Time.now` یک مُثُد کلاس است و یا یک مُثُد نمونه؟  
 ◇ با توجه به اینکه گیرنده آن کلاسی با نام `Time` است، می‌توان دریافت که این مُثُد یک مُثُد کلاس است. ■

**خودآزمایی ۶-۳-۲**

چرا عبارت `superclass` ۵ منجر به خطا «مِنْد تعریف نشده» می‌شود؟ (راهنمایی: به این فکر کنید که بین فراخوانی `superclass` بر روی عدد ۵ و فراخوانی `superclass` بر روی شیء بازگشتی از `class`. ۵ چه تفاوتی وجود دارد.)  
◇ یک مِنْد است که بر روی کلاس‌ها تعریف شده است. ۵ یک شیء است و نه یک کلاس، از این رو نمی‌توان `superclass` را بر روی آن فراخوانی کرد. ■

**خودآزمایی ۷-۳-۲**

از بین عبارات زیر، کدامین معادل یکدیگر هستند؟ (الف) `foo` (ب) `(p) %q{foo}` (پ) `%Q{foo}` (ت) `foo.to_s` (ث) `'foo'.to_sym`  
◇ (الف) و (ت) معادل یکدیگر هستند؛ همچنین (ب)، (پ)، و (ث) نیز معادل یکدیگر هستند. ■

**خودآزمایی ۸-۳-۲**

مقدار ذکر شده در \$1 وقتی که رشته ۱ to 25 در برابر عبارات باقاعدۀ زیر تطبیق داده شود، چه خواهد بود؟  
(الف) `/(\d+)/$`  
(ب) `/^\d+[^0-9]+/`  
◇ (الف) رشته "1" (ب) رشته "1" (پ) رشته "1" (ث) نیز شامل فاصله‌های دو طرف ■

**خودآزمایی ۹-۳-۲**

خط ۱۸ در کد شکل ۵-۲ را در نظر بگیرید. توضیح دهید که چرا روش زیر یک روش جایگزین قابل قبول برای تعریف مِنْد `full_title` است. همچنین مزایا و معایب آن را در مقایسه با روشی که در شکل نشان داده شده است، نام ببرید:

```
def full_title ; "#title (#year)" ; end
```

◇ این نسخه از کد، با فراخوانی مِنْدهای `title` و `year` به متغیرهای نمونه دسترسی پیدا می‌کند، در مقایسه با کد قبلی که مستقیماً به متغیرهای نمونه دسترسی داشت. این کار باعث جداسازی پیاده‌سازی این مِنْد از جزئیات پیاده‌سازی و چگونگی نگهداری مقادیر آن شیء فیلم (عنوان و سال ساخت) می‌شود. ■

**۴-۲ اصطلاحات برنامه‌نویسی روبی: حالت شاعرانه، بلوک‌ها، نوع دهی اردکی**

یک «اصطلاح برنامه‌نویسی» در واقع الگو و روشی متدابول در بین کاربران با تجربهٔ یک زبان برنامه‌نویسی است برای انجام کاری یا بیان چیزی. در حالی‌که ممکن است راههای دیگری برای انجام همان کار وجود داشته باشد، روش مصطلح راهی است که به راحتی برای سایر کاربران با تجربهٔ زبان قابل درک و مشخص است. هدف شما در حین یادگیری یک زبان جدید باید این باشد که باد بگیرید «به آن زبان فکر کنید»، یعنی اصطلاحات برنامه‌نویسی آن را به خوبی درک و استفاده کنید؛ یا به عبارت دیگر، از افتادن در این دام معروف «شما می‌توانید در هر زبان برنامه‌نویسی به سبک فورترن بنویسید»<sup>۸۶</sup>، اجتناب کنید. در این قسمت، سه تا از اصطلاحات برنامه‌نویسی اصلی روبی را بررسی می‌کنیم: پاس دادن آرگومان‌ها به مِنْدها («حالت شاعرانه»<sup>۸۷</sup> و پارامترهای نام‌گذاری شده)، بلوک‌ها و نوع دهی اردکی.<sup>۸۸</sup>

<sup>86</sup>Poetry Mode (Programming Style in Ruby)

<sup>87</sup>Duck Typing (Programming)

## فصل ۲. چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم

<https://gist.github.com/c0ec0b8c12c435990319416c056340a3>

```

1 link_to('Edit', {:controller => 'students', :action => 'edit'})
2 link_to 'Edit', :controller => 'students', :action => 'edit'
3 link_to 'Edit', controller: 'students', action: 'edit'
```

شکل ۷-۲: سه فراخوانی مُنْد link\_to (که بعداً در قسمت ۴-۴ با آن آشنا خواهیم شد). هر سه از نظر دستوری مجاز هستند و معادل یکدیگرند و یک رشته و یک هش را به عنوان آرگومان ورودی می‌گیرند. اولی همۀ پرانتزها و آکولادها دارد، دومی پرانتزها اطراف آرگومان‌های فراخوانی و همچنین آکولادهای اطراف آرگومان هش آخر را حذف کرده است، و سومی از دستور جایگزین (که از روی نسخه ۲.۰ به بعد امکان دارد) برای کلیدهای هش در آرگومان دوم استفاده کرده است.

<https://gist.github.com/4b32d51d724a86029604539dd465c7d6>

```

1 # Using 'named keyword' arguments
2 def greet(name, last_name: "", greeting: "Hi")
3   "#{greeting}, #{name} #{last_name}!"
4 end
5 greet("Dave")           # => "Hi, Dave!"
6 greet("Dave", last_name: "Fox") # => "Hi, Dave Fox!"
7 greet("Dave", greeting: "Yo")  # => "Yo, Dave!"
8 greet("Dave", greeting: "Hey", last_name: "Patterson")
9   # => "Hey, Dave Patterson!" - order of keyword args irrelevant
10 greet(greeting: "Yo")      # ArgumentError, since first arg is required
```

شکل ۷-۳: استفاده از آرگومان‌های کلیدوازه‌ای یا پارامترهای نام‌گذاری شده به شما امکان می‌دهد مُنْد های را تعریف کنید که در آن برخی از آرگومان‌ها اختیاری هستند با مقادیری پیش‌فرض را در نظر می‌گیرند. پارامترهای نام‌گذاری شده می‌توانند وضوح را برای مُنْد های که چندین آرگومان می‌گیرند بهبود بخشند، اگرچه در فصل ۹ خواهیم دید که معمولاً می‌باشد تعداد آرگومان‌هایی را که یک مُنْد می‌پذیرد به حداقل رساند.

**حال شاعرانه و پارامترهای نام‌گذاری شده.** آشکال ۷-۲ و ۸-۲ دو اصطلاح برنامه‌نویسی فراگیر در زبان روبي برای فراخوانی مُنْد را نشان می‌دهند. اصطلاح برنامه‌نویسی اول **حال شاعرانه** است که این امکان را می‌دهد تا پرانتزهای اطراف آرگومان‌ها را در هنگام فراخوانی یک مُنْد، در صورتی که تجزیه آن بدون ابهام باشد، حذف کرد. علاوه بر این، زمانی که آخرین آرگومان در فراخوانی یک مُنْد، یک هش باشد، می‌توان آکولادهای اطراف هش را نیز حذف کرد.

در نسخه‌های اولیه روبي، برنامه‌نویسان اغلب از هش به عنوان آرگومان برای شبیه‌سازی **پارامترهای نام‌گذاری شده** (که آرگومان‌های کلیدوازه‌ای نیز نامیده می‌شود) استفاده می‌کردند که در زبان‌هایی مانند پایتون، سی شارپ و غیره وجود دارد. به عنوان مثال، مستندات مُنْد link\_to که در شکل ۷-۲ استفاده شده، به ما می‌گوید که **action** و **controller** است: تنها دو مورد از مقادیر اضافی (و اختیاری) ممکن هستند که می‌توانند به عنوان کلید در یک هش مورد استفاده قرار بگیرند و به این مُنْد پاس داده شوند. امکان استفاده از پارامترهای نام‌گذاری شده واقعی (و نه شبیه‌سازی شده) بعداً در نسخه ۲.۰ روبي فراهم شد، همان‌طور که شکل ۸-۲ نشان می‌دهد. با وجود اين، مقدار زيادي از کدهای روبي که پيش از نسخه ۲.۰ نوشته شده‌اند، هنوز از هش برای ارسال آرگومان‌های اختیاری يا ارائه مقادير پيش‌فرض برای آرگومان‌ها استفاده می‌کنند.

**بلوک‌ها.** روبي از لفظ **بلوک** تا حدی متفاوت از زبان‌های دیگر استفاده می‌کند. در روبي، یک **بلوک** صرفاً یک مُنْد بدون نام است، که در دایره واژگان حوزه زبان‌های برنامه‌نویسي، تحت عنوان **عبارت لامبدي بـنام** نيز شناخته می‌شود. یک **بلوک** همانند یک مُنْد بنام معمولي، داراي آرگومان است و می‌تواند از متغیرهای محلی استفاده کند.

همان‌طور که شکل ۹-۲ نشان می‌دهد، يك از رايچ‌ترین کاربردهای **بلوک‌ها**، پياده‌سازی پيمایش يك داده‌ساختار<sup>۸۸</sup> است. مُنْد **نمونه** each، که در تمام کلاس‌های روبي که گرداور德 هستند، وجود دارد، تنها يك آرگومان به عنوان ورودي می‌گيرد که يك **بلوک** (عبارت لامبدي بـنام) است و هر يك از اعضائي گرداورد به آن **بلوک** ارسال می‌شود. each نمونه‌ای است از يك **پيمایشگر داخلی**. به قول

<https://gist.github.com/c3ea17d28f213c3e08e5503da3f210cf>

```

1 def print_movies(movie_list)
2   movie_list.each do |m|
3     puts "#{m.title} (rated: #{m.rating})"
4   end
5 end

```

شکل ۹-۲: مُتْد each یک آرگومان می‌بیند—یک بلوک—و هر عنصر از گردآورده را به‌نوبت به بلوک پاس می‌دهد. محدوده هر بلوک با `m` و مشخص می‌شود و هر آرگومان مورد انتظار بلوک در بین ادو خط عمودی `|` که بعد از `m` آمداند، قرار می‌گیرد. در هر تکرار، `m` مقداردهی می‌شود و به عنصر بعدی از `movie_list` اشاره می‌کند.

رویی بازها، گردآوردهای رویی «خودشان نحوه پیمایش‌شان را مدیریت می‌کنند»، زیرا این به گیرنده each بستگی دارد که چگونه آن مُتْد را برای به‌دست آوردن هر عنصر گردآورده پیاده‌سازی کند. (در واقع، در شکل ۹-۲، ما حتی نمی‌توانیم بگوییم که نوع داده `movie_list` چیست.)

پیمایشگرهای داخلی اولین بار در زبان برنامه‌نویسی سی‌ال‌پی پدیدار شدند. این زبان ابزار اولیه‌ای بود برای تحقیق در مورد نهان‌سازی داده‌ها که نهایتاً جایزه توپیگ را برای بارگذاشت لیسکف به ارمغان آورد.

شکل ۱۰-۲ یک مثال ساده از یک عملگر گردآورده را نشان می‌دهد که می‌تواند برای هر گردآورده که each را بعنوان راهی برای پیمایش خود پیاده‌سازی کرده باشد، مورد استفاده قرار گیرد. یک بار دیگر به این توجه کنید که ما هیچ ایده‌ای نداریم که این گردآورده چگونه پیاده‌سازی شده است: تنها چیزی که باید بدانیم این است که مُتْد نمونه each را برای پرشمردن عناصر خود پیاده‌سازی می‌کند. زبان رویی طیف گسترده‌ای از این قبیل مُتْدها را برای گردآوردها ارائه می‌دهد؛ شکل ۱۱-۲ برخی از کاربردی‌ترین آن‌ها را فهرست می‌کند. پس از کمی تمرین، شما برای بیان اجرای عملیات بر روی گردآوردها، بهطور خودکار شروع به استفاده از این اصطلاحات تابعی به جای حلقه‌های دستوری خواهید کرد.<sup>۸۹</sup> اگرچه در زبان رویی استفاده از دستور `for i in collection` نیز امکان پذیر است، اما each به ما امکان می‌دهند تا بهتر از **نوع‌دهی اردکی**، که در ادامه به آن خواهیم پرداخت، بهره ببریم و بازکاربردی‌زیری کد را بهبود ببخشیم.



**نوع‌دهی اردکی.** ممکن است تعجب کنید که مُتْدهای مربوط به گردآوردها که خلاصه‌ای از آن‌ها در شکل ۱۱-۲ آمده است (و تعدادی دیگر که در شکل نیستند) بخشی از کلاس `Array` (آرایه) رویی نیستند. در واقع، آن‌ها حتی بخشی از هیچ ابرکلاس دیگری هم نیستند که `Array` و دیگر انواع گردآورده از آن وراثت می‌کنند. در عوض، آن‌ها از یک سازوکار قدرتمندتری برای بازکاربرد بهره می‌برند: **میکس‌این<sup>۹۰</sup>** مجموعه‌ای نام‌گذاری شده از مُتْدهای مرتبط است که می‌توان به هر کلاسی که «قرارداد» مربوط به آن میکس‌این را برآورده می‌کند اضافه کرد. یک **پودمان<sup>۹۱</sup>** مورد نظر زبان رویی برای بسته‌بندی گروهی از مُتْدها به صورت یک کلاس قرار می‌گیرد، مُتْدهای نمونه، مُتْدهای کلاس و متغیرهای آن پودمان را با آن کلاس درمی‌آمیزد. مُتْدهای مربوط به گردآوردها در شکل ۱۱-۳ در **پودمانی به نام Enumerable** (قابل شمارش) تعریف شده‌اند که بخشی از کتابخانه استاندارد<sup>۹۲</sup> زبان رویی است و به صورت میکس‌این با تمام کلاس‌های گردآورده رویی ترکیب شده است. همانطور که مستندات<sup>۹۳</sup> آن بیان می‌کند، زیرا که با آن ترکیب می‌شود را ملزم به ارائه پیاده‌سازی برای مُتْد each می‌کند، مادامی که کلاسی مُتْد نمونه each را تعریف می‌کند، می‌تواند با این میکس‌این پیاده‌سازی شده‌اند. تعریف شده‌اند که کلاس یا میکس‌این هیچ یک مقصود و عزم خود را از پیش اعلام کنند. برای مثال، مُتْد each در کلاس `Array` رویی برروی عناصر آرایه پیمایش می‌کند، درحالی‌که مُتْد

این دسته از اصطلاحات و نحوه برنامه‌نویسی از برنامه‌نویسی از برخلاف برنامه‌نویسی دستوری است.<sup>۸۹</sup>

در زبان انگلیسی، **Mixin** یا **Mix-in** به درهم‌آمیختن، ترکیب کردن و مخلوط کردن اشاره دارد که اولین بار یک بسته‌فروش برای مخلوط بستن با شکلات یا بیسکویت این نام را برگزید. در حوزه برنامه‌نویسی شیء‌گرا نیز این لفظ برای توصیف ترکیب کردن رفتار یک پودمان با یک کلاس استفاده می‌شود. در ترجمه‌این کتاب ما از معادل «میکس‌این» استفاده کردیم و از جایگزین دیگری هنوز انتخاب نکردیم.

<sup>۹۱</sup>Module (Programming)

<sup>۹۲</sup>Standard Library (Programming)

## فصل ۲. چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم

<https://gist.github.com/3f068204f819ace57403adfd66652424>

```

1 # find largest element in a collection
2 def maximum(collection)
3   result = collection.first
4   collection.each do |item|
5     result = item if item > result
6   end
7   result
8 end
9 maximum([3,4,2,1])      # => 4
10 maximum(["a","x","b"]) # => "x"
11 maximum([RomanNumeral.new('XL'), RomanNumeral.new('LI')]) # => 'LI'
12
13 class RomanNumeral
14   include Comparable
15   attr :value
16   def initialize(roman_numeral_string)
17     @orig_string = roman_numeral_string
18     @value = RomanNumeral.convert_from_roman(roman_numeral_string)
19   end
20   def <=>(other)
21     @value <=> other.value
22   end
23   def to_s
24     @orig_string
25   end
26   def self.convert_from_roman(str)
27     # ...code to convert Roman numerals from strings...
28   end
29 end

```

شکل ۲-۲: این مثال با جستجو در یک گردآورده، عنصری که حداقل ارزش را دارد پیدا می‌کند. این مفهوم فارغ از نوع داده عناصر گردآورد عمل می‌کند، مادامی که آنها به عملگر `>` پاسخ می‌دهند. این مفهوم حتی روی اعداد رومی نیز کار می‌کند؛ فقط کافی است یک کلاس داشته باشیم که عملگر `>` را به طور صریح تعریف می‌کند یا صرفاً عملگر `<=>` را تعریف می‌کند و با درآمیختن با پودمان `RomanNumeral` `Comparable` عملگرهای `<` و `>` و غیره را تعریف می‌کند.

مُنْد	بلوک؟	یک گرداورد جدید شامل عناصر ذکرشده بازمی‌گرداند
	۱	عناصر حاصل از اعمال دستورات بلوک بر هر عنصر از <code>c</code>
	۱	زیرمجموعه‌ای از <code>c</code> که بلوک برای آن عنصر با مقدار درست ( <code>true</code> ) ارزیابی می‌شود
	۱	زیرمجموعه‌ای از <code>c</code> که عناصری که بلوک برای آن‌ها به مقدار درست ( <code>true</code> ) ارزیابی می‌شود از آن حذف شده است
		تمام عناصر <code>c</code> پس از حذف موارد تکراری
		عناصر <code>c</code> به ترتیب معکوس
		تمام عناصر غیرتهی <code>c</code> (مقدارشان <code>nil</code> نیاشد)
		عناصر <code>c</code> و همه آرایه‌های زیرمجموعه‌آن که به طور بازگشتی همگی مسطح شده‌اند تا هیچ عنصری از جنس آرایه باقی نمانده باشد
	۱	دو گرداورد، اول شامل عناصری از <code>c</code> که بلوک برای آن‌ها به مقدار درست ( <code>true</code> ) ارزیابی می‌شود و دومی شامل عناصری که بلوک برای آن‌ها به مقدار نادرست ( <code>false</code> ) ارزیابی می‌شود
۲	c.sort	عناصر <code>c</code> به صورت مرتب‌شده بر اساس بلوکی که دو آرگومان ورودی می‌گیرد و مقادیر <code>-1</code> ، <code>0</code> را به ترتیب برای وقتي که عنصر اول باید اول قرار داده شود، عنصر دوم باید اول قرار گیرد یا اینکه هر دو می‌توانند با هر ترتیبی قرار گیرند، برمی‌گرداند.
		مندهای زیر نیازمند این هستند که عناصر گرداورد به فراخوانی <code>&lt;=&gt;</code> پاسخ بدنهند؛ رجوع شود به قسمت ۴-۲.
	c.sort	اگر <code>sort</code> بدون یک بلوک فراخوانی شود، آن‌گاه عناصر با توجه به اینکه چطور به <code>&lt;=&gt;</code> پاسخ می‌دهند مرتب می‌شوند.
۱	c.sort_by	دستورات داخل بلوک را ببروی تک‌تک عناصر <code>c</code> اعمال و سپس نتیجه را مرتب می‌کند. به طور مثال، <code>{ m.title   m.movies.sort_by { a, b } a &lt;= b }</code> اشیایی از جنس <code>Movie</code> را با توجه به اینکه عنوان‌شان ( <code>title</code> ) چطور به <code>&lt;=&gt;</code> پاسخ می‌دهد مرتب می‌کند.
	c.max, c.min	بزرگترین یا کوچکترین عنصر گرداورد

شکل ۱۱-۲: تعدادی از مندهای رایج رویی برای کار با گرداوردها. برای مندهایی که یک بلوک را به عنوان ورودی می‌گیرند، ستون «بلوک» تعداد آرگومان‌های مورد انتظار بلوک را شناسان می‌دهد. خانهٔ خالی به این معنی است که آن مُنْد بلوک ورودی نمی‌گیرد. برای مثال، فراخوانی `مند sort` که بلوک آن دارای `۲` آرگومان است، می‌تواند به این شکل باشد: `c.sort { a, b } a <= b`. همهٔ این مُنْد‌ها به جای تغییر فراخوانی‌گیرنده، یک شیء جدید را برمو گردانند، اما برخی از مندهای یک نوع مخرب نیز دارند که به ! ختم می‌شود (برای مثال `sort!`) که فراخوانی‌گیرنده را مستقیماً تغییر می‌دهد و همچنین آن را به عنوان مقدار اصلاح شده برمو گرداند. در استفاده از مندهای مخرب احتیاط زیاد به خرج بدهید.



## فصل ۲. چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم

در کلاس `I` روی خطوط یک فایل<sup>۹۳</sup> یا سایر جریان‌های ورودی/خروجی پیمایش می‌کند. میکس‌این‌ها امکان استفاده مجدد از مجموعه‌ای از رفتارها را در کلاس‌های مختلفی که ارتباط دیگری با یکدیگر ندارند، فراهم می‌کنند.

به همین ترتیب، اگر یک کلاس `عملگر`<sup>۹۴</sup> معروف به «عملگر فضاییما»<sup>۹۵</sup> را تعریف کند، که مقادیر `-1`, `0`, `1` را با توجه به اینکه آرگومان دومش کوچکتر، برابر با و یا بزرگ‌تر از آرگومان اولش باشد، برمی‌گردداند، آن‌گاه این کلاس می‌تواند با پومن `Comparable` درآمیزد که با استفاده از `<=`, `==`, `>`, `<`, `=` و `between?` را تعریف می‌کند. به طور مثال، کلاس `Time` با تعریف `<=` و `Comparable` به شما این امکان را می‌دهد تا دستوری مثل `((23:15) Time.parse("19:00"), Time.parse("23:15").between?(Time.now))` را بنویسید.<sup>۹۶</sup>

اصطلاح «نوع‌دهی اردکی» توصیفی رایج از این قابلیت است، زیرا «اگر چیزی شبیه یک اردک باشد و مانند یک اردک صدا درمی‌آورد، می‌توان گفت واقعاً یک اردک است.» از نقطه‌نظر `Enumerable`، اگر یک کلاس دارای مُنْد `each` باشد، می‌توان گفت که آن یک گردآورده است، بنابراین به اجازه ارائه مُنْد‌های دیگری را می‌دهد که بر پایه `each` پیاده‌سازی شده‌اند. وقتی برنامه‌نویسان روی می‌گویند که یک کلاس «مانند `Array` صدا درمی‌آورد»، معمولاً منظورشان این است که لزوماً یک `Array` یا از نسل `Array` نیست، اما به اکثر مُنْد‌های مشابهی که رویی از پارامترهای `Array` پاسخ می‌دهد، آن نیز پاسخ می‌دهد. بنابراین می‌توان از آن در هر جایی استفاده کرد که از `Array` استفاده می‌شود.

### چکیده

- حالت شاعرانه این امکان را می‌دهد تا پرانتزهای اطراف آرگومان‌های فراخوانی مُنْد و همچنین آکولادهای اطراف یک هش را زمانی که آخرین آرگومان فراخوانی مُنْد است، حذف کرد. پیش از این مرسوم بود که از هش برای شبیه‌سازی پارامترهای نام‌گذاری شده یا آرگومان‌های کلیدوازه‌ای استفاده کرد، اما از آنجایی که رویی از پارامترهای نام‌گذاری شده واقعی با شروع نسخه ۲/۰ پشتیبانی می‌کند، اکنون این نوع استفاده توصیه نمی‌شود.
- رویی ایده‌های بسیار خوبی را از برنامه‌نویسی **تابعی** وام گرفته است، بهویژه استفاده از **بلوک‌ها** – تکه‌کدهای پارامترداری که عبارت `لامبда` نامیده می‌شوند و گستره تعریف شده خود را به همراه دارند و از همین رو می‌توان به آن‌ها **بستان** نیز گفت.
- به دلیل **نوع‌دهی پویا** در زبان رویی، فراخوانی یک مُنْد بروی یک شیء مجاز است مادامی که فراخوانی‌گیرنده بدون توجه به کلاسش به مُنْد پاسخ دهد، رفتاری که تحت عنوان **نوع‌دهی اردکی** نیز شناخته می‌شود.
- یک میکس‌این با استفاده از نوع‌دهی اردکی این امکان را فراهم می‌کند تا مجموعه‌ای از مُنْد‌های مرتبط را به هر کلاسی اضافه کرد که قراردادی را برآورده می‌کند؛ برای مثال، `Enumerable` فقط کلاسی که آن را مورد استفاده قرار می‌دهد ملزم می‌کند تا به مُنْد `each` پاسخ دهد. با قرار دادن عبارت `include ModuleName` پس از دستور `class ClassName` می‌توان یک پومن را با یک کلاس ترکیب کرد.
- برخلاف واسطه‌ها در زبان جاوا، میکس‌این‌ها نیازی به اعلان رسمی ندارند. اما از آنجایی که زبان رویی نوع‌دهی ایستانا ندارد، این مسئولیت شماست که اطمینان حاصل کنید که کلاسی که از میکس‌این استفاده می‌کند، شرایط ذکر شده در مستندات میکس‌این را برآورده می‌کند، در غیر این صورت، با خطای زمان اجرا مواجه خواهد شد.

<sup>۹۳</sup>File

<sup>۹۴</sup>Spaceship Operator (Programming)

<sup>۹۵</sup>این دستور با مقایسه زمان حال با ۱۹:۰۰ و ۲۳:۱۵، بررسی کند که آیا زمان فعلی بین این دو ساعت است یا خیر.

### ■ بیشتر بدانیم: بلوک‌ها در واقع بستار هستند

ترکیب بلوک‌ها و پیمایشگرها مانند `each`، که روش بیان اکثر عملیات برروی گردآوردها در رویی است، تکنیکی است که رویی از **برنامه‌نویسی تابعی** وام گرفته است. یک بلوک در رویی در واقع یک بستار است: هر زمان که بلوک اجرا شود، می‌تواند کل دامنهٔ واژگانی موجود در محل تعریف بلوک (جایی در متن کد برنامه که بلوک نوشته شده است) را «بینید» و به آن دسترسی داشته باشد. به عبارت دیگر، گویی بلوک یک «برگرفت» از گستره تهیه می‌کند، که بعداً هر زمان که بلوک اجرا شود، آن را بازسازی می‌کند. این ویژگی توسط بسیاری از ویژگی‌های ریلز بهشت مورد استفاده قرار گرفته است تا از تکرار پیرویه دستورات پیشگیری کند (در واقع میزان `DRY` بودن را بهبود می‌دهد). از جمله این ویژگی‌ها می‌توان به یندر کردن نماها (که در قسمت ۴-۴ خواهیم دید) و تأییدیه‌های اجازه می‌دهند تا تعریف پالایه‌های مربوط به کنترل‌گرها (قسمت ۱-۵)، اشاره کرد. این ویژگی‌ها اجازه می‌دهند تا تعريف چیزی را که قرار است اتفاق بیفت، از زمان و جایی که در ساختار برنامه رخ می‌دهد، جدا کرد.



### خودآزمایی ۱-۴-۲

یک خط به زبان رویی بنویسید که بررسی کند رشتة `s` واروخوانه (یا پالیندروم) است یا نه. رشتہ‌ای واروخوانه است که از راست به چپ و یا چپ به راست به یک صورت خوانده شود. راهنمایی: از متدهای شکل ۱-۲ استفاده کنید و فراموش نکنید که حروف بزرگ و کوچک (در زبان انگلیسی) اهمیت ندارند؛ به طور مثال، `ReDivider` یک واروخوانه است.

`s.downcase == s.downcase.reverse`

ممکن است فکر کنید که می‌توانستید بنویسید `s.reverse=~Regexp.new(s)`، اما اگر `s` حاوی کاراکترها یا حروف خاصی مثل `$` باشد که در عبارات باقاعدۀ معنای خاصی دارند، آن‌گاه این کد درست کار نمی‌کند. ■

### خودآزمایی ۲-۴-۲

فرض کنید که کلاس `Foo` را که تعریفی برای متد `each` ارائه نمی‌دهد، با `Enumerable` ترکیب کرده باشیم. آن‌گاه اگر دستور `{ |elt| puts elt }` را اجرا کنیم، چه خطای خواهیم گرفت؟

◊ متد `map` مربوط به `Enumerable` تلاش می‌کند که متد `each` را برروی گیرنده آن فراخوانی کند، اما از آنجایی که شیء تازه تعریف شده از جنس `Foo` تعریفی برای متد `each` ارائه نمی‌دهد، خطای متد تعریف‌نشده (`Undefined Method`) اتفاق می‌افتد. ■

### خودآزمایی ۳-۴-۲

کدام عبارت صحیح است و چرا؟

(الف) `include 'enumerable'`

(ب) `include Enumerable`

◊ گزینه (ب) صحیح است، چون‌که `include` انتظار دارد بعد از آن نام یک پودمان بیاید که (همانند نام یک کلاس) از نوع ثابت است و نه یک رشتۀ. ■

## فصل ۲. چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم

<https://gist.github.com/2b1976e79ef3c4b6bd6bb6b8e6dd54ab>

```

1 require "stripe"           # makes gem's definitions available within this file
2 Stripe.api_key = "sk_test_4eC39HqLyjWDarjtT1zdP7dc"
3 Stripe::Charge.create({
4   :amount => 2000,
5   :currency => "usd",
6   :source => "tok_mastocard", # obtained with Stripe.js
7   :description => "Charge for jenny.rosen@example.com"
8 })

```

شکل ۲-۱: برای استفاده از یک چم، ابتدا می‌باشد آن را نصب کنید. این کار را می‌توان با مستقیماً با استفاده از ابزار خط فرمان `gem` به صورت `'stripe'` (gem install stripe) انجام داد یا ترجیحاً از طریق ابزار باندلر که در ادامه به آن پرداخته‌ایم. دستور `gem help` از ابزار خط فرمان `gem` که جمهای نصب شده را به صورت دستی مدیریت می‌کند، ارائه توضیحات مختصی در مورد چگونگی استفاده از ابزار خط فرمان `gem` که مخزن اصلی چم‌ها است، نصب می‌کند.

## ۵-۲ پُفَک: آشنایی با روبي

### ۵-۲ پُفَک: آشنایی با روبي

<https://github.com/saasbook/hw-ruby-intro>

با نوشتن و اجرا کردن کدهایی به زبان روبي، اصول اولیه مربوط به کنترل جریان، کلاس‌ها، و راثت، متدهای دسترسی، عبارات باقاعدۀ، دستکاری و ایجاد تغییرات در رشته‌ها، نمادها و همچنین استفاده‌های رایج از بلوک‌ها به همراه پیمایشگرها و گردآوردها را تمرین کنید. خود را با مفسر برهم‌کنشی روبي یا همان `irb` و همچنین `byebug` آشنا کنید. با آر-إسپِک که ابزاری است برای ایجاد آزمون‌های خودکار آشنایی پیدا کنید و با نوشتن کد مناسب، آزمون‌هایی را که ما تعییه کرده‌ایم با موفقیت بگذرانید.

## ۶-۲ چم‌ها و باندلر: مدیریت کتابخانه‌ها در روبي

**کتابخانه‌ها.** کتابخانه استاندارد زبان روبي<sup>۹۷</sup> شامل تعداد زیادی کلاس کاربردی و مفید است که مواردي همچون خواندن و نوشتن از فایل‌ها و شبکه<sup>۹۸</sup>، کار با زمان و تاریخ، کار با رشته‌ها و گردآوردها و بسیاری موارد دیگر را پوشش می‌دهد.

در زبان برنامه‌نویسي روبي، يك کتابخانه خارجي به صورت يك چم<sup>۹۷</sup> بسته‌بندی و عرضه می‌شوند. يك چم<sup>۹۸</sup> در برگیرنده مجموعه‌اي از کلاس‌ها است با واسطه‌های واضح و مشخص. چم‌ها می‌توانند خيلي ساده باشند و صرفاً برخی کلاس‌های موجود را تغیير دهنند تا چندتابع کاربردی به آن‌ها اضافه کنند، يا حتی به پیچیدگی يك چارچوب كامل باشند: چارچوب ریلز خود به عنوان يك چم عرضه می‌شود که به چندین چم دیگر نیز وابستگی دارد. همانطور که شکل ۱۲-۲ نشان می‌دهد، دستور `require` در روبي مشابه دستور `import` در زبان پايتون عمل می‌کند و کلاس‌ها و توابع يك چم را در يك فايل که روبي در دسترس قرار می‌دهد و استفاده از آن‌ها را ممکن می‌کند.

البته نقطه قوت و متمایزکننده روبي در نحوه مدیریت وابستگی‌ها مابین چم‌ها است. گیتلب<sup>۹۹</sup> که يك نرم‌افزار کاربردی محبوب متن باز نوشته شده با استفاده از ریلز است، به حدود ۴۰۰ چم متکی



<sup>۹۷</sup>Network (Computing)

<sup>۹۸</sup>Gem (Ruby)

<sup>۹۹</sup>لطف چم به معنی سنگ قیمتی یا گوهر است که الهام‌گرفته شده از نام خود زبان روبي (Ruby) است که به معنی سنگ قیمتی باقوت است.

<sup>۱۰۰</sup>GitLab

است و از آن‌ها استفاده می‌کند. از آنجایی که برخی از این چم‌ها به نوبه خود به چم‌های دیگری نیز متکی هستند، در مجموع گیتبل به بیش از ۸۰۰ چم وابسته است که بسیاری از آن‌ها دائمًا در حال تکامل هستند. بنابراین، این بسیار حائز اهمیت است که بتوان تعیین کرد آپ با کدام نسخه(ها) از کتابخانه‌ها توسعه یافته و مورد آزمون قرار گرفته است، تا بتوان اطمینان حاصل کرد که وقتی آپ توزیع و به کاراندازی می‌شود، در هر محیطی که در آن اجرا می‌شود، رفتار یکسانی داشته باشد.

 برای مدیریت وابستگی‌های پیچیده، به یک سامانه مدیریت وابستگی‌ها یا سامانه مدیریت بسته<sup>۱۰۰</sup>، مانند `npm` برای پایتون، `bundler`<sup>۱۰۱</sup> برای نود چی‌اس یا آپاچی می‌بین<sup>۱۰۲</sup> برای جاوا نیاز داریم. سامانه مدیریت بسته زبان روبي، باندلر<sup>۱۰۳</sup> است که خود یک چم است. می‌توانید با اجرای دستور `gem install bundler` در خط فرمان<sup>۱۰۴</sup>، باندلر را نصب کنید و مدیریت وابستگی‌های پروژه‌تان را به آن بسپارید.

برای استفاده از باندلر در پروژه‌ای به زبان روبي، می‌بایست فایلی به نام `Gemfile` را در دایرکتوری<sup>۱۰۵</sup> اصلی خود (بالاترین سطح) داشته که وابستگی‌های آپ را به کتابخانه‌های خاص ثبت می‌کند. باندلر این فایل را می‌خواند و سعی می‌کند مجموعه‌ای از نسخه‌های کتابخانه‌های مورد استفاده را تنظیم کند، به نوعی که تمام محدودیت‌های در فایل آمده را لحاظ کرده باشد. برای مثال، اگر آپ به نسخه<sup>۱۰۶</sup> ۳.۰  $\geq$  ۴.۰  $\leq$  از کتابخانه X وابستگی دارد و در عین حال به کتابخانه Y نیز وابستگی دارد که آن کتابخانه خود به نسخه<sup>۱۰۷</sup> ۳.۵ کتابخانه X نیاز دارد، آنگاه نسخه<sup>۱۰۸</sup> ۳.۵ کتابخانه X نصب خواهد شد تا همه محدودیت‌ها برآورده شوند. باندلر همچنین می‌تواند تشخیص دهد که چه زمانی برآورده کردن همه محدودیت‌ها غیرممکن است. به طور کلی، هنگامی که یک پروژه جدید روبي را شروع می‌کنید، بلافضلله یک `Gemfile` برای آن ایجاد می‌کنید و همچنین زمانی که پروژه روبي شخص دیگری را بارگیری<sup>۱۰۹</sup> می‌کنید، پیش از آن که کار برروی آن را شروع کنید، ابتدا `bundle install` را در دایرکتوری اصلی پروژه اجرا می‌کنید تا به باندلر امکان پیدا کردن و بارگیری همه کتابخانه‌های لازم را بدهد.

سپس باندلر ترتیبی می‌دهد تا همه چم‌های مورد نیاز را با نسخه‌های مناسب نصب کند. باندلر همچنین نتیجه کار را در `Gemfile.lock` ثبت می‌کند. هم `Gemfile` و هم `Gemfile.lock` باید به عنوان بخشی از مجموعه کد ذخیره شوند؛ زیرا `Gemfile` به تنها یک صرفاً محدودیت‌هایی را مشخص می‌کند که کدام نسخه‌ها می‌توانند در صورت سازگاری استفاده شوند، و این `.lock` است که ثبت و ضبط می‌کند که کدام نسخه از کتابخانه‌ها در حقیقت برای توسعه استفاده شده‌اند.

به طور معمول و البته فرایاندهای، شماره‌های نسخه‌های کتابخانه‌ها از نسخه‌گذاری معنایی<sup>۱۱۰</sup> پیروی می‌کنند، نه فقط برای چم‌های روبي، بلکه در سایر زبان‌ها نیز این روش متدالوی است. بر اساس نسخه‌گذاری معنایی<sup>۱۱۱</sup>، ترتیب معمول این است که شماره نسخه طبق قالب `major.minor.patch`<sup>۱۱۲</sup> مشخص می‌شود، که در آن هر بخش یک عدد صحیح است، مانند ۱.۲.۳.۱. تغییرات در مقدار `patch` معمولاً همراه با رفع اشکال جزئی در عین سازگاری یا همسازی با قبل<sup>۱۱۳</sup> است، از جمله وصله‌های امنیتی، که قابلیت‌ها یا معنای اصلی چم را تغییر نمی‌دهند. تغییرات در `minor`<sup>۱۱۴</sup> معمولاً نشان‌دهنده اضافه شدن قابلیت‌های جدید با حفظ همساری با قبل است. تغییرات در بخش `major` نشان می‌دهد که واسط برنامه‌نویسی (API)<sup>۱۱۵</sup>-تریقه‌ای که شما توابع کتابخانه را فراخوانی‌ها می‌کنید-به گونه‌ای تغییر کرده است که ممکن است سازگاری با نسخه‌های

<sup>100</sup>Package Manager System

<sup>101</sup>Apache Maven

<sup>102</sup>Bundler (Ruby)

<sup>103</sup>Command-line

<sup>104</sup>Directory (Computing)

<sup>105</sup>Download

<sup>106</sup>Semantic Versioning

<sup>۱۱۰</sup>اچپ به راست به معنی اصلی، جزئی و وصله. در ترجمه این کتاب، ما شماره نسخه‌ها را نیز به همین ترتیب از چپ به راست نوشتایم.

<sup>108</sup>Backward Compatibility

<sup>109</sup>Application Program Interface

<code>gem install name [-v version]</code>	نسخه <code>version</code> (پیش‌فرض: آخرين) چم با نام <code>name</code> را نصب می‌کند. چم‌ها به صورت پیش‌فرض از <a href="http://rubygems.org">rubygems.org</a> باگیری می‌شوند.
<code>bundle install [--without env]</code>	اگر <code>Gemfile</code> تغییر نکرده است، <code>Gemfile.lock</code> را بررسی می‌کند و مطمئن می‌شود که نسخه‌های صحیح چم‌ها مشخص شده در آن فایل نصب شده است. اگر تغییر کرده باشد، گراف وابستگی‌ها را از نو می‌سازد تا <code>Gemfile.lock</code> جدیدی تولید شود، سپس مطمئن می‌شود که چم‌های صحیح نصب شده‌اند. اگر گزینه <code>--without</code> داده شده باشد، آن گاه نیازی به نصب چم‌های مرتبط با محیط <code>env</code> نیست و آن‌ها در نظر گرفته نمی‌شوند.
<code>bundle update gemnames</code>	باندلر را مجاب می‌کند تا چم‌هایی را که نامشان آمدۀ است ( <code>gemnames</code> ) به روزرسانی کند و وابستگی‌ها را از نو محاسبه کند. اگر گزینه <code>all</code> -- به جای <code>gemnames</code> داده شود، آن‌گاه تمامی چم‌ها می‌بایست به روزرسانی شوند در حالی‌که <code>Gemfile.lock</code> نادیده گرفته می‌شود و سپس از نو ساخته می‌شود.
<code>bundle exec command</code>	دستور <code>command</code> را با در نظر گرفتن شرایط تعریف‌شده اجرا می‌کند. این به این معنی است که باندلر ابتدا اطمینان حاصل می‌کند که نسخه‌های صحیح چم‌ها باگیری شده و قبل از اجرای دستور فعل هستند. به عنوان مثال، اگر دستور <code>command</code> به نسخه خاصی از یک چم متکی باشد، اما شما نسخه‌های دیگری از آن را نیز نصب کرده باشید، بدون استفاده از <code>bundle exec</code> ممکن است هنگام اجرای دستور، نسخه اشتباهی از آن چم فعل باشد.
<code>bundle help</code>	راهنمای و توضیحات مربوط به نحوه کار با باندلر را نمایش می‌دهد؛ همچنین می‌توانید به وبگاه باندلر <a href="#">ک</a> رجوع کنید.

شکل ۲-۱۳: دستورات پرکاربرد برای کار با چم‌ها و باندلر. ما در ادامه و در قسمت ۱-۴ با «محیط‌های» باندلر نیز آشنا خواهیم شد.

قبلی را از بین ببرد.

از آنجایی که شکستن سازگاری یک تصمیم مهم است که ممکن است بر هزاران آپ که از این کتابخانه استفاده می‌کنند تأثیر بگذارد، یک روش معمول این است که چنین تغییراتی ابتدا به عنوان هشدارهای **منسوخ شدن**<sup>۱۱۰</sup> در یک نسخه جزئی جدید یا نسخه وصله<sup>۱۱۱</sup> ظاهر شوند. چنین هشدارهایی معمولاً به صورت پیام‌هایی با این چنین مضمونی «هشدار: این قابلیت در نسخه اصلی بعدی این کتابخانه متفاوت عمل می‌کند [ایا حذف می‌شود]» در زمان ساخت (زمان کامپایل) یا زمان اجرا<sup>۱۱۲</sup> منتشر و ظاهر می‌شوند. به عنوان یک قاعدة کلی، هشدارهای منسوخ شدن بعد از اینکه نسخه اصلی تغییر کند به خطا تبدیل می‌شوند. بنابراین، یک توسعه‌دهنده محتاط و هوشمند وقتی که با هشدار<sup>۱۱۳</sup> منسوخ شدن مواجه می‌شود، به مستندات رجوع می‌کند تا مشخص کند که آیا راهی برای تغییر کد فعلی وجود دارد تا از نسخه پیش رو این قابلیت یا واسطه برنامه‌نویسی جدید قابلیت مدنظر استفاده کند.

در واقع، هنگام ارتقا به یک نسخه اصلی جدید، بهترین روش این است که ابتدا به صورت تدریجی ارتقا دهید تا بتوانید پیش از تغییر نسخه اصلی، موارد منسوخ شده را شناسایی و برطرف کنید. برای مثال، فرض کنید آپ شما از نسخه ۲.۷.۳ فلان چم استفاده می‌کند، اما آخرین نسخه ۳.۱.۵ است و شما هنوز به روزرسانی انجام نداده‌اید:

- ابتدا می‌بایست به آخرین نسخه که نسخه اصلی آن هنوز ۲ است به روزرسانی کنید-فرض کنید که این نسخه ۲.۸.۱ است.

- هرگونه هشدار منسوخ شدن ایجاد شده پس از این ارتقا را شناسایی و به آن رسیدگی کنید.

- حال به اولین نسخه‌ای ارتقا دهید که نسخه اصلی آن ۳ است. به احتمال زیاد این نسخه ۳.۰.۰ است، اما ممکن است متفاوت باشد. مطمئن شوید که همه چیز با این نسخه اصلی جدید به خوبی کار می‌کند.

- سپس، به آخرین نسخه جزئی ارتقا دهید-در مثال ما، احتمالاً ۳.۱.۰ است. اطمینان حاصل کنید که همه چیز به خوبی کار می‌کند.

- در نهایت، به نسخه ۳.۱.۵، آخرین نسخه وصله منتشرشده، به روزرسانی کنید.

البته، در یک دنیای ایده‌آل، توسعه‌دهنده به مرور و به طور مرتباً چم را به روزرسانی می‌کند، بنابراین لازم نیست همه این مراحل را یکجا انجام شوند. در فصل ۸، ما به تفصیل به این خواهیم پرداخت که چطور می‌توان اطمینان حاصل کرد از اینکه «همه چیز به خوبی کار می‌کند»؛ زیرا این یکی از نقش‌های کلیدی داشتن یک مجموعه آزمون قوی است.

<sup>110</sup>Deprecation (Software)

<sup>111</sup>Patch (Computing)

<sup>112</sup>Runtime

<sup>113</sup>Warning (Computing)

باندلر از نوشтар ۲> به معنای «آخرین نسخه‌ای که نسخه اصلی آن ۲ است» است. این معملاً هشدار منسوخ شدن ایجاد شده پس از این ارتقا را شناسایی و به آن رسیدگی کنید.
حال به اولین نسخه‌ای ارتقا دهید که نسخه اصلی آن ۳ است. به احتمال زیاد این نسخه ۳.۰.۰ است، اما ممکن است متفاوت باشد. مطمئن شوید که همه چیز با این نسخه اصلی جدید به خوبی کار می‌کند.
سپس، به آخرین نسخه جزئی ارتقا دهید-در مثال ما، احتمالاً ۳.۱.۰ است. اطمینان حاصل کنید که همه چیز به خوبی کار می‌کند.
در نهایت، به نسخه ۳.۱.۵، آخرین نسخه وصله منتشرشده، به روزرسانی کنید.
البته، در یک دنیای ایده‌آل، توسعه‌دهنده به مرور و به طور مرتباً چم را به روزرسانی می‌کند، بنابراین لازم نیست همه این مراحل را یکجا انجام شوند. در فصل ۸، ما به تفصیل به این خواهیم پرداخت که چطور می‌توان اطمینان حاصل کرد از اینکه «همه چیز به خوبی کار می‌کند»؛ زیرا این یکی از نقش‌های کلیدی داشتن یک مجموعه آزمون قوی است.

### چکیده‌ای از کتابخانه‌ها و مدیریت وابستگی‌ها در روبی:

- علاوه بر کتابخانه‌های استاندارد یک زبان، که معمولاً با توزیع خود زبان منتشر می‌شود و در دسترس است، بیشتر زبان‌ها از مجموعه‌ای گسترده از کتابخانه‌های خارجی دربرگیرنده کدهای کمکی برخوردار هستند. در زبان، روبی، کتابخانه‌های خارجی به عنوان چهارهای رویی عرضه و توزیع می‌شوند.
- پیگیری این که یک آپ به کدام نسخه (ها) از چه کتابخانه‌های وابسته است بسیار مهم است. ابزار باندلر تا حد زیادی این را خودکار می‌کند؛ توسعه‌دهنده این امکان را دارد تا محدودیت‌هایی را برای نسخه‌های کتابخانه‌ها مشخص کند و سپس باندلر مجموعه‌ای از نسخه‌ها را پیدا می‌کند که تمامی محدودیت‌ها را برآورده می‌کند.
- اکثر کتابخانه‌ها از نسخه‌گذاری معنایی پیروی می‌کنند که به صورت `major.minor.patch` (وصله.جزئی.اصلی) نسخه‌بندی می‌شوند. در این روش وصله‌ها اشکالات را برطرف می‌کنند، نسخه‌های جزئی قابلیت جدیدی اضافه می‌کنند و نسخه‌های اصلی کتابخانه را به گونه‌ای تغییر می‌دهند که ممکن است سازگاری با نسخه‌های اصلی قبلی را از بین ببرد.
- برای آماده‌سازی توسعه‌دهنگان برای چنین تغییرات اساسی، نسخه‌های جزئی و یا وصله‌های جدید اغلب پیام‌هایی را نمایش می‌دهند که به توسعه‌دهنده درباره استفاده از قابلیت‌هایی که در شرف تغییر ناسازگار است، هشدار می‌دهند. به روزرسانی‌های مکرر و به موقع این احتمال را به حداکثر می‌رساند که بتوانید موارد منسوخ شده را قبل از تبدیل شدن به خطأ ببینید و متعاقباً اصلاحات لازم را انجام دهید؛ در `Gemfile` که برای باندلر آماده می‌کنید مشخص می‌کند که چه به روزرسانی‌هایی «ایمن» و مجاز هستند.

## ۷-۲ باورهای نادرست و خطرهای پنهان

### ! خطر پنهان: همیشه نقش ناظر را در برنامه‌نویسی دونفره داشته باشد.

اگر از بین یکی از دو همکار یک نفر تجربه بسیار بیشتری داشته باشد، وسوسه‌انگیز است که نقش راننده را به کلی به او بسپارید و نفر کم‌تجربه‌تر در واقع ناظر دائمی شود. این یک رابطه سالم کاری نیست و احتمالاً منجر به عدم مشارکت عضو کم‌تجربه‌تر خواهد شد.

### ! خطر پنهان: هنگام یادگیری یک زبان جدید، کورکورانه صرفاً آموزش‌هایی را به سبک کتاب آشپزی دنبال کنید.

اسطورةٌ علوم رایانه و برندهٔ جایزهٔ تورینگ، دانلد کنوث که چندین کتاب مرجع بر روی مبانی نظری علوم رایانه نوشته است، می‌گوید که نوشتن کد «سخت‌تر از هر کار دیگری است که من تا به حال انجام داده‌ام». پیتر نورویگ، مدیر تحقیقات شرکت گوگل، به شیوه‌ایی گفته است<sup>۳</sup> که هیچ میانبری برای حل چنین چالشی وجود ندارد: این کار نیاز به مطالعه و درک عمیق و تمرين مداوم زیادی دارد. دنبال کردن یک آموزش گام‌به‌گام بدون درک سازوکارهای زیربنایی که توضیح داده شده است، این امکان را به شما می‌دهد تا چیزی را به سرعت آماده و بر روی صفحه نمایش دهید، اما شما نمی‌توانید بفهمید که چگونه به آنجا رسیده است و نمی‌توانید این موفقیت را با آپهای خود تکرار کنید.

### ! خطر پنهان: فراموش کنید که کامپایلر شما را نجات نخواهد داد.

در زبان‌های برنامه‌نویسی با نوع دهی قوی یا نوع دهی ایستا، کامپایلر معمولاً می‌تواند تشخیص

دهد که آیا به یک متغیر از یک نوع خاص، مقداری از نوع ناسازگار به اشتباه تخصیص داده شده است، به عنوان مثال، نوشتن کد `x=foo` وقتی که `foo` یک مقدار عددی برمی‌گرداند اما `x` به عنوان یک متغیر رشته‌ای تعریف شده است. در زبان‌های برنامه‌نویسی با نوع دهنده ضعیف یا نوع دهنده پویا (روبی هر دو را دارد)، چنین بررسی‌های «زمان کامپایل» وجود ندارند—در عوض شما یک خطای زمان احرا دریافت خواهید کرد. بنابراین باید در آزمون و طراحی کد خود بسیار مراقب باشید. فصل ۸ تکنیک‌هایی را برای اطمینان از اینکه کد شما به خوبی مورد آزمون قرار گرفته است، معرفی می‌کند. بحث بر سر مزیت‌های نسبی نوع دهنده ایستا در مقابل نوع دهنده پویا یک «جنگ مقدس»<sup>۶</sup> طولانی‌مدت در میان برنامه‌نویسان است که ما در اینجا به آن نمی‌پردازیم.

### خطر پنهان: به سبک پایتون، کد روبی بنویسید.

یادگیری زیر و بم یک زبان برنامه‌نویسی جدید، اصطلاحات برنامه‌نویسی آن و اینکه با زبان‌های دیگر چه تفاوت‌هایی دارد به زمان و تمرین نیاز دارد. در اینجا چندین اشتباه رایج برنامه‌نویسان باسابقهٔ پایتون را که در برنامه‌نویسی با روبی تازه‌کار هستند با هم مرور می‌کنیم:

- خواندن و تعبیر کردن عبارتی مثل `person.age` به صورت «خصوصیت `age` از شیء `person`» به جای «فراخوانی متّد نمونه `age` بر روی شیء `person`».

• خیال کنید که عبارت `person.age=40` صرفاً یک انتساب ساده به یک خصوصیت است، در حالی‌که این یک فراخوانی متّد است. در حقیقت اینجا متّد `age` با آرگومان ورودی `40`، بر روی شیء `person` فراخوانی می‌شود که این متّد هر کاری ممکن است بکند. کدهای نوشته‌شده به زبان روبی اغلب از این سازوکار به عنوان راهی برای ارائه قند نحوی برای «انتساب‌هایی» استفاده می‌کنند که می‌توانند اثر جانی داشته باشند.

- فراموش کردن این نکته که متغیر نمونه‌ای که مقداردهی نشده است، بی سر و صدا (بدون هیچ خطابی) به `nil` ارزیابی می‌شود.

• در نظر گرفتن `attr_accessor` به عنوان اعلان وجود خصوصیات. این میان‌بر و سایر موارد مشابه، کار شما را راحت می‌کنند، اگر بخواهید برای یک کلاس یک خصوصیت عمومی تعریف کنید که برای عموم قابل خواندن یا نوشتن باشد. اما در زبان روبی به هیچ وجه نیازی به «اعلان» یک خصوصیت ندارید (صرفًا وجود متغیر نمونه کفایت می‌کند) و به احتمال زیاد برخی از خصوصیات اصلاً نباید به صورت عمومی قابل مشاهده باشند. در مقابل این وسوسه مقاومت کنید که از `attr_accessor` بخواهید به همان سبکی که در جاوا برای خصوصیات اعلان می‌نوشتید، استفاده کنید.

- نوشتن حلقه‌های فور صریح به جای استفاده از یک پیمایشگر مثل `each` و متّدهای مربوط به گردآوردها که از طریق میکس‌این‌های مانند `Enumerable` از این پیمایشگر نهایت استفاده را می‌کنند. از متّدهای کاربردی مانند `select`, `all?`, `any?`, `map` و غیره استفاده کنید.

• استفاده از نگارش شتری (`lowerCamelCase`) به جای نگارش ماری (`snake_case`) برای نام‌گذاری متغیرها. ممکن است بی‌اهمیت و ناچیز به نظر برسد، اما برنامه‌نویس با تجربه خواندن کدی را که رسم و رسوم نوشترانی یک زبان را نقض می‌کند، بسیار آزاردهنده می‌داند، درست همانطور که نوازنده‌گان با تجربه با شنیدن یک نت خارج از گام، به لزه می‌افتد. اگر در مورد چیزی شک دارید، کدهای روبی دیگری را که کار مشابه کار مورد نظر شما انجام می‌دهند پیدا و از آن‌ها الگوپردازی کنید.

### خطر پنهان: خیال کنید که رشته‌ها و نمادها معادل و قابل جایگزینی با یکدیگرند.

در حالی‌که بسیاری از متّدهای ریلز به طور خاص جوری ساخته شده‌اند که هم یک رشته و هم یک نماد را به عنوان ورودی بپذیرند، این دو به طور کلی قابل تعویض و جایگزینی با یکدیگر

## فصل ۲. چگونه یک زبان برنامه‌نویسی جدید یاد بگیریم

نیستند. متنی که انتظار یک رشته را دارد، در صورت دریافت یک نماد، بسته به نوع پیاده‌سازی اش، ممکن است با خطای مواجه بشود. برای مثال، خروجی عبارت ('foo', 'bar'].`.include?`('foo') است، درحالی‌که خروجی عبارت (:foo', 'bar'].`.include?`(:) کذب (Falsy) حقیقی (Truthy) است.

### خطر پنهان: استفاده از یک متغیر محلی به جای فراخوانی یک متند است.

فرض کنید در کلاس C یک متند با نام `x` تعریف شده است. در یک متند نمونه از کلاس C، نوشتند عبارت `3=x` به معنی فراخوانی متند `=x` با آرگومان 3 نخواهد بود؛ بلکه یک متغیر محلی `x` را با عدد ۳ مقداردهی می‌کند که این احتمالاً آن چیزی نیست که شما می‌خواستید. برای به دست آوردن نتیجهٔ مورد نظر می‌باشد از عبارت `self.x=3` استفاده کنید که فراخوانی متند `x` را به صورت صریح و واضح انجام می‌دهد.

### خطر پنهان: اشتباه گرفتن `require` با `.include`

دستور `require` یک فایل روبی دلخواه (ممعلولاً فایل اصلی یک چم) را بارگذاری می‌کند، درحالی‌که `include` یک پودمان را با کلاس فعلی درمی‌آمیزد. در هر دو مورد، زبان روبی قوانین خاص خود را برای مکان‌یابی فایلهای حاوی کد دارد. مستندات رسمی زبان روبی به چگونگی استفاده از `$LOAD_PATH`<sup>۱۴</sup> پرداخته است، اما در عمل نماید آن را مستقیماً دستکاری کنید، اگر از `RILZ` به عنوان چارچوب کاری خود و از `BANDL` برای مدیریت چم‌های خود استفاده می‌کنید.

## ۸-۲ نکات پایانی: چگونه می‌توان یک زبان را با جستجو در گوگل یاد (ن)گرفت

به نظر نویسنده‌گان این کتاب، توصیه‌های این قسمت را می‌توان به خوبی و به طور ادبیانه‌ای با دو نقل قول زیر بیان کرد. اولین نقل قول از Tam Nait<sup>۱۵</sup> است که یکی از طراحان اصلی ماشین‌های لیسپ<sup>۱۶</sup> (ابانه‌های تحقیقاتی که در MIT<sup>۱۷</sup> که برای بهینه‌سازی اجرای برنامه‌های نوشته شده به زبان برنامه‌نویسی لیسپ<sup>۱۸</sup> طراحی شده‌اند) بوده است.

یک تازه‌کار سعی می‌کرد یک ماشین لیسپ را که درست کار نمی‌کرد، [صرفًا] با خاموش و روشن کردن آن درست کند. Tam نایت پس از دیدن آنچه آن دانشجو در حال انجام دادن بود، با حالت سختگیرانه‌ای به او گفت: «شما نمی‌توانید صرفًا با خاموش-روشن کردن این دستگاه و بدون درک درستی از اینکه چه ابرادی وجود دارد آن را درست کنید.» Tam نایت سپس ماشین را خاموش و روشن کرد و ماشین بدون ایرادی شروع به کار کرد.

—قسمت «AI Koans» از کتاب واژه‌نامه هکر جدید<sup>۱۹</sup>

برای ما، نقل قول بالا خطرات کبی کردن کد به صورت کورکورانه، بدون اینکه بدانیم چگونه کار می‌کند، را نشان می‌دهد: ممکن است در ابتدا به نظر برسد که کد کار می‌کند، اما اگر نمی‌دانید چرا، یا اگر چیز دیگری را خراب کند، ممکن است در آینده دچار مشکل شوید و مطمئناً چیز زیادی یاد نخواهید گرفت. درحالی‌که ویگاه‌های پرسش و پاسخ برنامه‌نویسی مانند استک اورفلو<sup>۲۰</sup> هم برای پرسش و هم برای پیدا کردن تکه‌کدهایی برای انجام یک سری کارهای خاص ارزشمند هستند، شما باید به دنبال یک الگوی کلی باشید، مطابق با آنچه می‌خواهید انجام دهید؛ پس از اینکه به طور کامل

<sup>۱۴</sup>مسیرهایی که در زمان اجرا برای بارگذاری به آن‌ها رجوع می‌شود.

<sup>۱۵</sup>Tom Knight

<sup>۱۶</sup>Lisp Machine

<sup>۱۷</sup>Massachusetts Institute of Technology

<sup>۱۸</sup>Lisp (Programming Language)

<sup>۱۹</sup>The New Hacker's Dictionary (Book)

درک کردید که آن نمونه کد چگونه کار می‌کند، آن را با شرایط خاص مورد نیاز خود تطبیق دهید. به عبارت دیگر، به دنبال چگونگی (روش انجام یک کار) باشید و نه چیستی (جواب نهایی). نقل قول دوم از نویسنده کتاب کلیسای جامع و بازار<sup>۱۲۰</sup> (ریموند ۲۰۰۱)، شرحی اولیه از مزایای بالقوه توسعه متن باز، است:

برنامه‌های زشت مانند پل‌های معلق زشت هستند: آن‌ها بسیار بیشتر از همنوعان زیبای خود در معرض فرو ریختن هستند، زیرا نحوه درک انسان‌ها (هویته مهندسان) از زیبایی با توانایی ما در برداش و درک پیچیدگی ارتباط نزدیک دارد. زبانی که نوشتن کدی زیبا و بارزه را ساخت می‌کند، در واقع نوشتن کد خوب را ساخت می‌کند.

ایران اس. ریموند<sup>۱۲۱</sup>

یادگیری استفاده از یک زبان جدید و استفاده حداکثری از اصطلاحات برنامه‌نویسی آن یک مهارت حیاتی برای متخصصان حوزه نرم‌افزار است. این البته کار آسانی نیست، اما امیدواریم نشان دادن و تمرکز بر ویژگی‌های منحصر به فرد و زیبایی روبی و جاواسکریپت، کنگاوری فکری شما و نه ناله‌های تسلیم و کناره‌گیری‌تان را برانگیزد. همچنین امیدواریم ارزش استفاده از ابزارهای تخصصی گوناگون و انتخاب مناسب‌ترین و کارآمدترین مورد برای هر کاری را فرا بگیرید.

اگر این اولین رویارویی شما با روبی است، برنامه‌نویسی منابعی و بلوک‌ها در روبی و بستارها در جاواسکریپت ممکن است کمی به زمان نیاز داشته باشید تا به آن‌ها عادت کنید. اما مانند هر زبان دیگری، عدم یادگیری استفاده صحیح از اصطلاحات برنامه‌نویسی ممکن است منجر به از دست دادن فرصت استفاده از سازوکاری در زبان جدید شود که راه حل زیباتری ارائه می‌دهد.

بنابراین توصیه ما این است که در حین یادگیری یک زبان جدید به پشتکار و تلاش خود آنقدر ادامه بدھید تا زمانی که با اصطلاحات برنامه‌نویسی آن راحت کنار بیایید و بتوانید از آن‌ها به راحتی استفاده کنید. در مقابل این وسوسه مقاومت کنید که کد خود از زبان‌های دیگر به زبان مقصود صرفاً «نویسه‌گردانی» کنید<sup>۱۲۲</sup>، بدون اینکه ابتدا در نظر بگیرید که آیا روشی متعارف بر مبنای اصطلاحات برنامه‌نویسی زبان جدید برای بیان آنچه در زبان مقصود نیاز دارید وجود دارد یا خیر.

اگر می‌خواهید با زبان‌های بیشتری آشنا بشوید و مهارت‌های مربوط به برنامه‌نویسی خود را گسترش دهید، ما کتاب هفت زبان در هفت هفته<sup>۱۲۳</sup> (تیت ۲۰۱۰) را توصیه می‌کنیم، که خواننده را با مجموعه‌ای از زبان‌ها آشنا می‌کند که هر یک راهکارهای کاملًا متفاوتی را برای بیان کارها در برنامه‌نویسی می‌طلبند.

در نهایت، ما این منابع را برای جزئیات بیشتر در مورد زبان روبی توصیه می‌کنیم. مجددًا، لازم به ذکر است که ما فرض می‌کنیم که روبی اولین زبان برنامه‌نویسی که شما می‌خواهید فرا بگیرید نیست و این کتاب و این دوره آموزشی اولین مواجهه شما با برنامه‌نویسی نیستند، بنابراین به منابعی که برای مبتدیان در نظر گرفته شده است اشاره نمی‌کنیم.

- کتاب برنامه‌نویسی روبی<sup>۱۲۴</sup> و همچنین کتاب زبان برنامه‌نویسی روبی (فلانگان و ماتسوموتو ۲۰۰۸)، که توسط مختار روبی، یوکیهیرو ماتسوموتو<sup>۱۲۵</sup> تالیف شده است، مراجعی معتبر و کامل برای زبان روبی هستند.

- مستندات زبان روبی که به صورت آنلاین در دسترس قرار دارد، جزئیات زبان، کلاس‌های آن و کتابخانه‌های استاندارد آن را ارائه می‌دهد. چند مورد از کاربردی‌ترین کلاس‌ها عبارت‌اند از IO (ورودی/خروجی مربوط به فایل‌ها و شبکه، شامل فایل‌های CSV)، Set (عملیات مربوط

<sup>120</sup>The Cathedral and the Bazaar (Book)

<sup>121</sup>Eric S. Raymond

<sup>122</sup>نویسه‌گردانی در زبان‌شناسی به نوشتن کلمه‌ای از زبانی با الفبایی متفاوت در زبانی دیگر به‌نوعی که حروف به‌طور منتظر از زبان و الفبای اول به زبان و الفبای دوم نگاشت شوند، گفته می‌شود. در اینجا نویسه‌گردانی به این اشاره دارد که کدی را در زبانی جدید به سبک زبانی دیگر که از پیش می‌دانیم بنویسیم، بدون اینکه از امکانات متناسب آن در زبان جدید استفاده کنیم. یا به عبارت دیگر بدون توجه به اصطلاحات برنامه‌نویسی زبان جدید، آن کد را صرفاً به سبک متعارف در زبان قبلی بنویسیم.

<sup>123</sup>Seven Languages In Seven Weeks (Book)

<sup>124</sup>Yukihiro "Matz" Matsumoto



به گردآوردن مجموعه مانند تفاضل مجموعه‌ها، اشتراک مجموعه‌ها و غیره) و Time (کلاس استاندارد برای بیان زمان، که ما استفاده از آن را به جای Date توصیه می‌کنیم حتی اگر می‌خواهید فقط با تاریخ بدون داشتن زمان کار کنید). این موارد حکم مرجع را دارند، نه مطالب آموزشی.

• کتاب **یادگیری روبي**<sup>۱۲۵</sup> (Fitzgerald ۲۰۰۷) رویکردی بیشتر به سبک آموزشی برای یادگیری این زبان دارد.

• کتاب **راه و روش روبي**<sup>۱۲۶</sup> یک مرجع دانشنامه‌ای و جامع است هم برای خود زبان روبي و هم برای چگونگی استفاده از اصطلاحات برنامه‌نویسي روبي برای حل بسیاری از مسائل کاربردي برنامه‌نویسي است.

• کتاب **بهترین راهکارهای استفاده از روبي**<sup>۱۲۷</sup> (Braun ۲۰۰۹) بر اینکه چطور می‌توان بهترین استفاده را از «ابزارهای قدرتمند» روبي مانند بلوک‌ها، پودمان‌ها/نوع‌دهی اردکی، فرآبرنامه‌نویسي و غیره داشت، تمرکز دارد. اگر می‌خواهید مانند یک روبي باز قهار کد روبي بنویسید، این کتاب یک منبع عالی است.

• بسیاری از تازه‌کارهای زبان روبي با دستور yield مشکل دارند، مخصوصاً از آنجایی که در زبان‌های جاوا، سی یا سی‌پلاس‌پلاس معادلی برای آن وجود ندارد (اگرچه نسخه‌های اخیر پایتون و جاوا اسکریپت سازوکارهای مشابهی دارند). مقاله **همروال**<sup>۱۲۸</sup> در وبکی‌پدیای انگلیسی، مثال‌های خوبی را از سازوکارهای همروالی که yield پشتیبانی می‌کند ارائه می‌دهد.

Andrew Begel and Nachiappan Nagappan. Pair programming: What's in it for me? in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 120–128, Kaiserslautern, Germany, October 2008.

Gregory T Brown. *Ruby Best Practices*. O'Reilly Media, 2009. ISBN 0596523009.

A. Cockburn and L. Williams. The costs and benefits of pair programming. *Extreme Programming Examined*, pages 223–248, 2001.

Michael James Fitzgerald. *Learning Ruby*. O'Reilly Media, 2007. ISBN 0596529864.

David Flanagan and Yukihiro Matsumoto. *The Ruby Programming Language*. O'Reilly Media, 2008. ISBN 0596516177.

J. Hannay, T. Dyba, E. Arisholm, and D. Sjoberg. The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, 51(7):1110–1122, July 2009.

Leah Hoffmann. Q&a: Big challenge. *Communications of the ACM (CACM)*, 56(9):112–ff, September 2013.

Joe Moore. ipad 2 as a remote presence device? *Pivotal Labs*, 2011. URL <http://pivotallabs.com/blabs/categories/pair-programming>.

<sup>125</sup>Learning Ruby (Book)

<sup>126</sup>The Ruby Way (Book)

<sup>127</sup>Ruby Best Practices (Book)

<sup>128</sup>Coroutine (Programming)

Eric S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly Media, Inc., 2001.

Fernando J. Rodríguez, Kimberly Michelle Price, and Kristy Elizabeth Boyer. Exploring the pair programming process: Characteristics of effective collaboration. in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE '17*, page 507–512, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346986. doi:10.1145/3017680.3017748. URL <https://doi.org/10.1145/3017680.3017748>.

Nischal Shrestha, Colton Botta, Titus Barik, and Chris Parnin. Here we go again: Why is it difficult for developers to learn another programming language? *Communications of the ACM*, 65(3), March 2022. URL <https://cacm.acm.org/magazines/2022/3/258915-here-we-go-again/fulltext>.

Bruce Tate. *Seven Languages in Seven Weeks: A Pragmatic Guide to Learning Programming Languages (Pragmatic Programmers)*. Pragmatic Bookshelf, 2010. ISBN 193435659X. URL <https://www.amazon.com/Seven-Languages-Weeks-Programming-Programmers/dp/193435659X>.

## پیوند ها

[https://rubular.com<sup>۱</sup>](https://rubular.com)  
[http://dilbert.com/strips/comic/2003-01-09/<sup>۲</sup>](http://dilbert.com/strips/comic/2003-01-09/)  
[http://dilbert.com/strips/comic/2003-01-11/<sup>۳</sup>](http://dilbert.com/strips/comic/2003-01-11/)  
[https://code.visualstudio.com<sup>۴</sup>](https://code.visualstudio.com)  
[https://sublimetext.com<sup>۵</sup>](https://sublimetext.com)  
[https://queue.acm.org/detail.cfm?id=1039535<sup>۶</sup>](https://queue.acm.org/detail.cfm?id=1039535)  
[http://ruby-doc.org/core-2.5.1/Enumerable.html<sup>۷</sup>](http://ruby-doc.org/core-2.5.1/Enumerable.html)  
[https://ruby-doc.org/stdlib<sup>۸</sup>](https://ruby-doc.org/stdlib)  
[https://rubygems.org<sup>۹</sup>](https://rubygems.org)  
[https://rubygems.org<sup>۱۰</sup>](https://rubygems.org)  
[https://bundler.io<sup>۱۱</sup>](https://bundler.io)  
[https://semver.org/spec/v2.0.0.html<sup>۱۲</sup>](https://semver.org/spec/v2.0.0.html)  
[http://norvig.com/21-days.html<sup>۱۳</sup>](http://norvig.com/21-days.html)  
[https://wiki.c2.com/?HolyWar<sup>۱۴</sup>](https://wiki.c2.com/?HolyWar)  
[https://stackoverflow.com<sup>۱۵</sup>](https://stackoverflow.com)  
[http://ruby-doc.org/docs/ProgrammingRuby<sup>۱۶</sup>](http://ruby-doc.org/docs/ProgrammingRuby)  
[http://ruby-doc.org/<sup>۱۷</sup>](http://ruby-doc.org/)

# معماری نرم افزار کاربردی :SaaS ریزسرویس‌ها، API‌ها، و

فکر می‌کنیم مهم‌ترین ایده خوب در یونیکس، واسط ساده و تمیز آن بود: `open` (باز کردن)، `close` (بستن)، `read` (خواندن) و `write` (نوشتن).

—یونیکس و فراتر از آن: مصاحبه‌ای با کِن تامپسون، مجله IEEE Computer، جلد ۳۲، شماره ۵، ۱۹۹۹ مه

دینیس ریچی (۱۹۴۱-۲۰۱۱) و کِن تامپسون (۱۹۴۳-). در سال ۱۹۸۳ به طور مشترک، به پاس مشارکت‌های بنیادین‌شان در طراحی سیستم‌های عامل به طور کلی و همچنین ابداع سیستم عامل یونیکس به طور خاص، جایزه تویرینگ را دریافت کردند.



۱-۱	معماری کلاینت-سرور در وب	9۱
۲-۳	ارتباطات در SaaS از خطوط سیری HTTP استفاده می‌کند	9۷
۳-۳	پُفَک: HTTP و URI‌ها	1۰۲
۴-۳	از وبگاه‌ها تا ریزسرویس‌ها: معماری سرویس‌گرا	1۰۳
۵-۳	واسطه‌های برنامه‌نویسی مبتنی بر REST: همه چیز نوعی منبع است	1۰۷
۶-۳	URI‌های مبتنی بر REST، فراخوانی واسطه‌های برنامه‌نویسی و JSON	1۱۳
۷-۳	پُفَک: ساخت و به کاراندازی یک آپ SaaS ساده	1۱۷
۸-۳	باورهای نادرست و خطرهای پنهان	1۱۷
۹-۳	نکات پایانی: سیر تحول از واسط دروازه مشترک تا معماری سرویس‌گرا	1۱۹

## پیش‌نیازها و مفاهیم

### مفاهیم:

• آپ‌های SaaS از الگوی **کلاینت-سرور** پیروی می‌کنند که در آن یک کلاینت درخواست‌هایی ارسال می‌کند و یک سرور به درخواست‌های تعداد زیادی کلاینت پاسخ می‌دهد.

• در معماری وب، دو مفهوم بنیادی وجود دارد: پروتکل انتقال ابرمن (HTTP) که یک پروتکل درخواست-پاسخ برای انتقال محتوای وب است، و شناسانه‌های منبع یکنواخت (URI)، که یک منبع خاص در وب را نام‌گذاری کرده و ممکن است پارامترهایی برای دسترسی به آن مشخص کند. یکی از ساختارهای پایه‌ای SaaS، یک خط سیر HTTP است که ترکیبی از یک روش درخواست مانند POST یا GET URL است.

• با گذشت زمان، وب از مجموعه‌ای از منابع ایستا به مجموعه‌ای از برنامه‌ها (سرویس‌ها) تبدیل شد که می‌توانستند از راه دور و از طریق یک مرورگر وب یا توسط یک آپ SaaS مورد دسترسی قرار گیرند. این تغییر رویه به سمت **معماری سرویس‌گرا (SOA)**، زمینه‌ساز رشد و گسترش ریزسرویس‌ها شد.

• واسط برنامه‌نویسی (API) برای یک ریزسرویس یک توصیف رسمی از عملیات‌هایی است که آن ریزسرویس قادر به انجام آنهاست. طبق رویکرد طراحی «API-محور»، می‌توان با تمرکز بر مؤلفه‌های کوچک‌تر و واسطه‌های برنامه‌نویسی ارتباطی بین آنها، حتی هنگام ساختن یک آپ SaaS بزرگ، یک طراحی پودمانی‌تر ایجاد کرد.

• از آنجا که اکثر آپ‌های موبایل، در واقع کلاینت‌های SaaS هستند و به همان روشنی که یک ریزسرویس به یک آپ SaaS متصل می‌شود، از آن سرویس می‌گیرند، رویکرد طراحی API-محور که در این فصل ارائه شده، یک مکمل عالی برای رویکرد طراحی «تلفن‌همراه محور» مطرح شده در فصل ۱ محسوب می‌شود.

### ۱-۳ معماری کلاینت-سرور در وب

هر بار که از یک مرورگر وب برای بازدید از یک وبگاه استفاده می‌کنید یا از یک آپ تلفن همراه که از خدمات رایانش ابری کمک می‌گیرد، برای مصارفی مثل دریافت آخرین پیش‌بینی‌های آب و هوای بهره می‌گیرید، در واقع با **کلاینت SaaS** خود به یک سرور متکی بر مدل نرم‌افزار به صورت یک سرویس (SaaS) درخواست ارسال می‌کنید. پیاده‌سازی نرم‌افزار به صورت یک سرویس بروزی پروتکل‌های وب معمول‌ترین نمونه از پیاده‌سازی **معماری کلاینت-سرور**<sup>۱</sup> است. در این معماری، کلاینت‌ها برنامه‌هایی هستند که مسئولیت درخواست اطلاعات از سرورها را به عهده دارند و معمولاً به کاربران اجازه برهمنکش با آن اطلاعات را می‌دهند. سرورها در سوی دیگر، برنامه‌هایی هستند که مسئولیت فراهم کردن اطلاعات به صورت کارآمد در حجم بالا برای چندین کلاینت به صورت همزمان را به عهده دارند.

امروزه کلاینت‌های SaaS شکل‌های مختلفی به خود می‌گیرند. وقتی می‌خواهید از سرویس نقشه گوگل (یا همان گوگل مپ<sup>۲</sup>) استفاده کنید، فرقی نمی‌کند که نقشه را از طریق یک مرورگر روی رایانه شخصی باز کنید، یا از طریق یک مرورگر روی گوشی هوشمند و یا حتی مستقیماً از طریق آپ مخصوص گوشی هوشمند؛ چرا که در هر سه حالت از یک کلاینت SaaS استفاده می‌کنید. با وجود تمام تفاوت‌های بین کلاینت‌ها که ناشی از اهداف و طرز استفاده‌های متفاوت‌شان است، همگی به یک سرور SaaS یکسان برای دریافت سرویس نقشه گوگل متصل می‌شوند.

برخلاف نرم‌افزار کلاینت که معمولاً یک آپ مستقل است که بر روی یک دستگاه واحد مانند رایانه شخصی یا گوشی هوشمند اجرا می‌شود، «سرور» عموماً مجموعه‌ای از رایانه‌هایی است که چندین مولفه نرم‌افزاری مختلف را اجرا می‌کنند (که در زمان مناسب با آن‌ها آشنا خواهیم شد) که مجموعاً عملکرد وبگاه واقعی را تشکیل می‌دهند. نحوه توزیع این مولفه‌ها<sup>۳</sup> بر روی یک یا چند رایانه به نوع محیط میزبانی و تعداد کاربرانی که آپ باید به آن‌ها خدمات خود را ارائه کند بستگی دارد. در هر صورت، «سرور» به عنوان یک موجودیت منطقی واحد برای کلاینت ظاهر می‌شود، تا کلاینت بتواند با خیالی راحت از جزئیات نحوه به کاراندازی سرور (اجزای داخلی، همبندی<sup>۴</sup>، نحوه اتصال بین اجرا و غیره) بی‌اطلاع بماند. شما حتی می‌توانید یک سرور کوچک را روی رایانه شخصی خود به کار بیندازید که توانایی آن فقط به اندازه‌ای کافی باشد که بتواند به یک و فقط یک کاربر (مثلاً خود شما به عنوان یک توسعه‌دهنده) در هنگام توسعه و آزمون یک آپ SaaS کمک کند.

ایجاد تمایز بین کلاینت و سرور این فرصت را به هر دو نوع برنامه می‌دهد تا مرکز را روی وظایف تخصصی خود قرار دهند: کلاینت می‌تواند یک رابط کاربری واکنش‌گرا<sup>۵</sup> و جذاب داشته باشد، در حالی که سرور بر روی ارائه کارآمد به حداقل تعداد کلاینت‌ها به طور همزمان تمرکز می‌کند. بنابراین معماری کلاینت-سرور، اولین نمونهٔ ما از یک **الگوی طراحی** است. یک ساختار، رفتار، استراتژی یا تکنیک با قابلیت استفاده چندباره که با جدا کردن چیزهایی که تغییر می‌کنند از چیزهایی که ثابت می‌مانند، راحلی اثبات شده برای مجموعه‌ای از مشکلات مشابه را در بر می‌گیرد. در مورد معماری کلاینت-سرور، آنچه ثابت می‌ماند، جداسازی دغدغه‌ها بین کلاینت و سرور است، علی‌رغم تغییراتی که در پیاده‌سازی جزئیات کلاینت‌ها و سرورها ایجاد می‌شود.

**«رایانه شخصی خود» که در این کتاب به آن اشاره می‌کنیم، در واقع محیط است که شما در آن به توسعه می‌پردازید. این محیط می‌تواند حتی یک محیط توسعهٔ یکپارچه (IDE) باشد که بر روی زیرساخت ابری در حال اجرا است و شما از طریق مرورگر به آن دسترسی دارید.**

البته کلاینت-سرور تنها معماری و الگوی طراحی موجود در سرویس‌های مبتنی بر اینترنت نیست. در **معماری همتا**<sup>۶</sup> که در بیت‌تورنت<sup>۷</sup> استفاده می‌شود، هر عضو هم کلاینت و هم سرور است و هر کسی می‌تواند از دیگران اطلاعات بخواهد. در چنین سامانه‌ای که یک برنامه باید هم به عنوان کلاینت و هم به عنوان سرور عمل کند، تخصصی کردن برنامه برای انجام هر یک از کارها

از کلاینت با نام سرویس‌گیرنده یا کارخواه و از سرور با نام سرویس‌دهنده یا کارساز نیز یاد می‌شود. از این رو، این نوع معماری را معماری سرویس‌گیرنده-سروری می‌نامیم.

<sup>1</sup>Google Maps (Web Mapping Service)

<sup>2</sup>Component (Computing)

<sup>3</sup>Topology (Networking)

<sup>4</sup>Responsive (Web Design)

<sup>5</sup>Peer-to-peer (Networking)

<sup>6</sup>BitTorrent

سال	سامانه	کلاینت	سرور	پروتکل‌ها
۱۹۶۰	سابری، سامانه خرید بلیت برای شرکت هوایپیمای امریکن ایرلاینز	پایانه‌های سفارشی الکترومکانیکی مستقر در آزادس‌های مسافرتی	دو عدد بزرگ‌رایانه آی‌بی‌ام ۷۰۹۰	پروتکل سفارشی مبتنی بر <b>FM</b> بر روی خطوط تلفن اجراهای
۱۹۷۱	<b>FTP</b> که به کاربران امکان بارگیری فایل‌ها از روی سرور را می‌داد	در ابتدا، کلاینت مبتنی بر خط فرمان <b>ftp</b> : امروزه، کلاینت‌های مبتنی بر خط فرمان <b>NcFTP</b> ، <b>cURL</b> و <b>WinSCP</b> ، رابطه‌ای کاربری گرافیکی مثل سایبردک، فوج، و همچنین تمامی مرورگرهای وب	بسته‌های نرم‌افزاری سرور مختلف نظیر <b>vsftpd</b> یونیکس، فایل‌زیلا و	پروتکل <b>FTP</b> مبتنی بر ASCII بر بستر مجموعه پروتکل اینترنت (TCP/IP)
۱۹۸۳	نرم‌افزار تحت <b>NetWare</b> شبکه شرکت نوول، که به رایانه‌های شخصی با سیستم‌های عامل <b>MS-DOS</b> و <b>CP/M</b> اجازه به اشتراک‌گذاری فایل‌ها بر روی یک سرور را می‌داد	یک نرم‌افزار سفارشی قابل اجرا بر روی سیستم <b>MS-DOS</b>	دستگاه (فایل‌سرور) تعبيه شده سفارشی شرکت نوول بر اساس ریزپردازنده موتورولا ۶۸۰۰	پروتکل‌های سفارشی بر روی رابط شبکه سفارشی سازگار با رایانه شخصی
۱۹۸۴	<b>POP</b> پستی، پروتکل دفتر جداسازی کلاینت‌های رایانه‌های از سرورها ممکن کرد	نرم‌افزاری مختلف نظیر یودارا، تاندربرید، اپل میل، مایکروسافت اوتلوك، ilm، پاین و یوریکا	بسته‌های نرم‌افزاری سرور مختلف نظیر آپاچی چیمز، انجین‌اکس، یودارا و کیوبایر	پروتکل POP مبتنی بر TCP/IP ASCII بر بستر عدمتاً توسط پروتکل <b>IMAP</b> جایگزین شده است
۱۹۹۰	شبکه جهانی وب	نرم‌افزاری مختلف نظیر موزاییک، نتسکیپ نویگیتو، اینترنت اکسپلورر ماکروسافت، موزیلا فایرفاکس، گوگل کروم و اپل سافاری	بسته‌های نرم‌افزاری سرور مختلف نظیر سرور وب آپاچی، سرور اطلاعات اینترنتی ماکروسافت و انجین‌اکس	پروتکل انتقال ابرمن <b>HTTP</b> (HTTP) مبتنی بر ASCII بر بستر TCP/IP

شکل ۱-۳: وب واردت تاریخچه‌ای طولانی و غنی از سامانه‌های کلاینت-سرور رایانه‌ای است. درحالی‌که می‌توان گفت همهٔ این مثال‌ها ایدهٔ نرم‌افزار به صورت یک سرویس (SaaS) را منعکس می‌کنند، امروزه این اصطلاح به سامانه‌های کلاینت-سرور ساخته شده با استفاده از **HTTP** و سایر استانداردها و پروتکل‌های وب اشاره دارد.

واقعاً دشوارتر است. در روزهای ابتدایی دنیای رایانه، معماری‌های مبتنی بر الگوی کلاینت-سرور منطقی‌تر و مناسب‌تر به نظر می‌آمدند، چرا که سخت‌افزار مورد استفاده در کلاینت می‌باشد از سرور ارزان‌تر تمام می‌شد تا سرویس‌دهندگان بتوانند تنها با یک یا تعداد کمی سرور گران‌قیمت پاسخ‌گوی تعداد زیادی کلاینت باشند. امروزه، با کاهش هزینه‌های سخت‌افزاری که منجر به ظهور گوشی‌های هوشمند و مروگرهای وب قدرتمند شده است که از پویانمایی<sup>۸</sup> و جلوه‌های سه‌بعدی پشتیبانی می‌کنند. توصیف دقیق‌تر شاید این باشد که کلاینت‌ها و سرورها تقریباً به یک اندازه پیچیده هستند اما همچنان برای نقش‌های بسیار متفاوت خود تخصصی به شمار می‌روند. در واقع، ما آن نقش‌های متمایز را در الگوهای طراحی که در چارچوب‌های سمت کلاینت (آنگولار، ریاکت و غیره) پدیدار می‌شوند، در مقابل نقش‌هایی که در چارچوب‌های سمت سرور پدیدار می‌شوند (ریلز، چنگو<sup>۹</sup>، نود چی‌اس و غیره) مشاهده خواهیم کرد. حتی عباراتی مانند «Client Push»<sup>۱۰</sup> منعکس‌کننده این پیش‌فرض است که کلاینت‌ها از سرورها متمایز هستند.

اگرچه سامانه‌های کلاینت-سرور مدت‌ها قبل از ظهور نرم‌افزار به صورت یک سرویس، وب یا حتی اینترنت به وجود آمدند، به‌دلیل فرآگیر بودن وب، ما از اصطلاح «نرم‌افزار به صورت یک سرویس» (SaaS) به معنای «سامانه‌های کلاینت-سرور ساخته شده بر اساس استانداردهای باز شبکه جهانی وب» استفاده می‌کنیم. طبق این تعریف، سرویس‌های تحت وب به‌وسیلهٔ مجموعه‌ای از قالب‌های داده<sup>۱۱</sup> و پروتکل‌ها<sup>۱۲</sup> که در این فصل مورد بررسی قرار خواهیم داد، عموماً توسط مروگرهای آپ‌های گوشی‌های هوشمند قابل دسترسی هستند.

### چکیده:

- آپ‌های SaaS تحت وب، نمونه‌هایی از الگوی معماری کلاینت-سرور هستند که در آن کلاینت (سرویس‌گیرنده) معمولاً به‌طور تخصصی وظيفة ارتباط و برهمنکش با کاربر و ارسال درخواست‌های کاربر به سرور (سرویس‌دهنده) را به عهده دارد، درحالی‌که سرور روی رسیدگی به حجم زیادی از چنین درخواست‌هایی تمرکز دارد.
- میراث به جا مانده از وب یعنی معماری کلاینت-سرور به‌طور فرآگیر در سرتاسر پُشته‌های نرم‌افزاری، پروتکل‌ها و اصطلاحاتی که ما در این کتاب به آن‌ها برمی‌خوریم مشهود است.
- امروزه مروگر وب به‌عنوان یک «کلاینت همگانی و فرآگیر» شناخته می‌شود، چرا که برخلاف استانداردهای خصوصی (متعلق به شرکتی خاص) مورد استفاده در برخی آپ‌های کلاینت-سرور قدیمی، آپ‌های تحت وب از استانداردهای باز بهره می‌برند که به هر کسی اجازهٔ پیاده‌سازی بدون حق امتیاز را می‌دهند.
- یک جایگزین برای معماری کلاینت-سرور، معماری همتابه‌همتا است که در آن همهٔ اعضاء هم به‌عنوان کلاینت و هم به‌عنوان سرور عمل می‌کنند. درحالی‌که می‌توان گفت این معماری انعطاف‌پذیرتر است، تخصصی کردن نرم‌افزار برای انجام مناسب هر یک از کارها را دشوار می‌کند.

### ۱-۱-۳ خودآزمایی

تفاوت اصلی بین نقش کلاینت‌ها و سرورها در SaaS چیست؟

- ◊ یک کلاینت SaaS به‌گونه‌ای بهینه شده است تا امکان برهمنکش کاربر با اطلاعات را فراهم کند، درحالی‌که یک سرور SaaS برای سرویس‌دهی به تعداد زیادی از کلاینت‌ها به‌طور همزمان بهینه شده است. ■

<sup>8</sup>Animation

<sup>9</sup>Django

<sup>10</sup>Data Format

<sup>11</sup>Protocol (Computing)

**شاپیستگی ۲-۱-۳**

برخی از مزایای یک پروتکل بدون حالت کلاینت-سرور را شرح دهد.

**۲-۳ ارتباطات در SaaS از خطوط سیری HTTP استفاده می‌کند**

کلاینت‌های SaaS چطور با سرورهای SaaS ارتباط برقرار می‌کنند؟ **پروتکل شبکه**، مجموعه‌ای از قوانین ارتباطی است که عوامل شرکت‌کننده در یک شبکه در مورد آن توافق دارند. پروتکل اصلی که همه رایانه‌های موجود در اینترنت را به هم پیوند می‌دهد **مجموعه پروتکل اینترنت** یا همان **TCP/IP**<sup>۱۲</sup> است که در برگیرنده پروتکل کنترل انتقال (TCP<sup>۱۳</sup>) و پروتکل اینترنت (IP<sup>۱۴</sup>) است. به طرف ارتباط اجازه می‌دهد تا توالی‌های مرتب‌شده‌ای از بایت‌ها<sup>۱۵</sup> را در هر دو جهت به‌طور هم‌زمان مبادله کنند (**کاملاً دوطرفه**<sup>۱۶</sup>، مشابه یک مکالمه تلفنی که در آن هر دو طرف می‌توانند هم‌زمان صحبت کنند و گوش دهند). اگر برنامه‌ای در یک سمت اتصال TCP/IP (مثلًا کلاینت) رشته‌ای از حروف را ارسال کند، سرور دقیقاً همان رشته حروف را دریافت می‌کند و بالعکس. نقش دو طرف ارتباط را از هم متمایز نمی‌کند—برای آن مهم نیست که یکی سرور و یکی کلاینت است یا اینکه آن دو در واقع دو سمت یک رابطه همتا هستند—و هیچ محدودیتی روی ماهیت داده‌های منتقل شده قرار نمی‌دهد. تدوین قوانین ارتباطی به عهده خود برنامه‌هایی است که در بستر TCP/IP مشغول به تبادل داده هستند. همانطور که خواهیم دید، در مورد سرورهای وب<sup>۱۷</sup> و مرورگرهای وب، این قوانین توسط **پروتکل انتقال ابرمن**<sup>۱۸</sup> (HTTP) تعریف می‌شوند.

رایانه‌ها در یک شبکه مبتنی بر TCP/IP چطور با یکدیگر ارتباط برقرار می‌کنند؟ هر رایانه دارای یک

**نشانه آپی**<sup>۱۹</sup> است که از چهار بایت که از هم با نقطه جدا شده‌اند (مثل ۱۷۲.۳۲.۲۴۴.۱)، تشکیل شده است. در بیشتر اوقات از نشانه‌های آپی به‌طور مستقیم استفاده نمی‌کنیم. سرویس اینترنتی DNS تحت عنوان **سامانه نام دامنه**<sup>۲۰</sup> (DNS)، که خود نیز یک پروتکل مبتنی بر بستر TCP/DARP دارد، به‌طور خودکار در پشت صحنه **نامهای میزبان**<sup>۲۱</sup>، مثل www.eecs.berkeley.edu، را به نشانه‌های آپی مربوط به آن تبدیل می‌کند. وقتی نام وبگاهی نظری www.eecs.berkeley.edu را در نوار آدرس<sup>۲۲</sup> مرورگر خود قرار می‌دهید، مرورگر به‌طور خودکار با سرور DNS ارتباط برقرار می‌کند تا نام و بگاه را به نشانه آپی که در این مثال ۱۷۲.۳۲.۲۴۴.۱۷۲ معرف می‌کند. بنا به دلایل تاریخی که مربوط به طراحی IP است، نشانه آپی ۱۷۰.۰.۰.۱ و نام میزبان localhost (میزبان محلی) همیشه معرف همان رایانه‌ای خواهد بود که آپ روی آن در حال اجراست. این قابلیت به شما اجازه می‌دهد آپهای تحت وب را با اتصال به سروی که روی رایانه شخصی خودتان کار می‌کند، توسعه و مورد آزمون قرار دهید: از دیدگاه کلاینت، چنین سروی شبیه به سروری در بستر رایانش ابری است که توسط هزاران رایانه پشتیبانی می‌شود، عمل می‌کند. یعنی یک برنامه سرور مبتنی بر TCP/IP بدون در نظر گرفتن اینکه نرم‌افزار سرور در کجا اجرا و چگونه روی یک یا چند رایانه توزیع می‌شود، همان تجربه (یا تحرید) را برای کلاینت فراهم می‌کند.

از آنجایی که چندین برنامه مبتنی بر TCP/IP می‌توانند به‌طور هم‌زمان روی یک رایانه اجرا شوند (مثلًا یک سرور وب و یا یک سرور رایانه)، نشانه آپی به تنهایی برای تفکیک کردن آن‌ها کافی

۱۲Internet Protocol Suite (Transmission Control Protocol/Internet Protocol)

۱۳Transmission Control Protocol

۱۴Internet Protocol

۱۵Byte (Computing)

۱۶Full Duplex (Communication)

۱۷Web Server

۱۸HyperText Transfer Protocol (HTTP)

۱۹IP Address (Networking)

۲۰Domain Name System

۲۱Hostname (Computing)

۲۲Address Bar (Browser)

**هشتتایپ (Octet)** (Octet) اصطلاحی است که در ادبیات شبکه گاهی اوقات به یک گروه ۸ بیت اطلاق می‌شود—میراثی به جا مانده از دوران پیش از اینکه بزرگ‌رایانه System/360 شرکت آی‌بی‌ام باشد که ۸ بیت را به عنوان یک استاندارد برای همگان جا بیندازند.

نیست. بنابراین، برقراری یک ارتباط موفق TCP/IP به یک شماره درگاه<sup>۲۳</sup>، که عددی است بین ۱ تا ۶۵۵۳۵ هم نیاز دارد تا مشخص کند کدام برنامه در حال اجرا بر روی سرور همان مقصود ارتباطی مورد نظر است. یک برنامه باید حتماً روی همان شماره درگاه مورد نظر بر روی سرور در حال گوشش دادن باشد تا بتواند درخواست ارتباط را بپذیرد. سرورهای وبی که در محیط‌های عملیاتی و نهایی اجرا می‌شوند، به طور پیش‌فرض روی درگاه شماره ۸۰ برای ارتباط‌های مبتنی بر HTTP و روی درگاه شماره ۴۴۳ برای ارتباط‌های مبتنی بر HTTPS گوش می‌دهند. در فصل ۱۲ توضیح خواهیم داد که ارتباط‌های مبتنی بر HTTPS (که کوتاه‌نوشت معادل پروتکل امن انتقال ابرمن<sup>۲۴</sup> است) از رمزگاری بر پایه کلید عمومی<sup>۲۵</sup> برای رمزگذاری<sup>۲۶</sup> ارتباط و محافظت در برابر استراق سمع اطلاعات استفاده می‌کند. هنگامی که در محیط توسعهٔ شخصی خود یک سرور برای مورد آزمون قرار دادن آپ خود راه‌اندازی می‌کنید، شماره درگاه مورد استفادهٔ سرور برای «گوش دادن» به عوامل تغییر محیط توسعهٔ یکپارچه<sup>۲۷</sup> مورد استفاده شما، چارچوبی که استفاده می‌کنید (مثلًاً چارچوب ریلز از درگاه شماره ۳۰۰۰ استفاده می‌کند) و یا حتی نحوهٔ راه‌اندازی و اجرای سرور بستگی خواهد داشت.

**پروتکل HTTP** که مجموعه‌ای از قوانین برقراری ارتباط برای ایجاد درخواست و دریافت پاسخ است، تعریف دقیق و مشخص دارد که می‌توان آن را در مراحل زیر خلاصه کرد:

۱- کلاینت با تعیین نشانهٔ آی‌پی و شماره درگاه (ممکن‌باش<sup>۲۸</sup>) اتصال TCP/IP به سرور را آغاز می‌کند. اگر بر روی رایانهٔ مقصود با نشانهٔ آی‌پی مورد نظر، پردازهای که به عنوان یک سرور HTTP در حال گوش دادن روی درگاه مشخص شده باشد وجود نداشته باشد، کلاینت بلاfacile با خطای مواجه می‌شود. اکثر مرورگرها این خطای را با پیام «دسترسی به این ویگاه امکان‌پذیر نیست» یا «اتصال رد شد» گزارش می‌دهند.

۲- در غیر این صورت، اگر اتصال موفقیت‌آمیز باشد، کلاینت بلاfacile یک درخواست HTTP ارسال می‌کند که قصد خود برای انجام عملیات مورد نظرش روی یک منبع<sup>۲۹</sup> را توضیح می‌دهد. منبع هر چیزی است که آپ سرور آن را تحت مدیریت خود دارد—یک صفحهٔ وب، یک تصویر و یا فرم ارسالی که یک حساب کاربری جدید ایجاد می‌کند، همگی نمونه‌هایی از یک منبع هستند.

۳- سرور یک پاسخ HTTP ارائه می‌دهد که یا درخواست کلاینت را برآورده می‌کند یا هرگونه خطای که مانع از موفقیت درخواست شده است را گزارش می‌دهد. پاسخ همچنین ممکن است شامل اطلاعاتی به شکل یک کوکی<sup>۳۰</sup> HTTP باشد که به سرور اجازه می‌دهد همین کلاینت را در ارتباط‌ها و برهمنکش‌های بعدی به درستی شناسایی کند.

درخواست HTTP از جانب کلاینت در مرحله ۲ چه شکل و ظاهری دارد؟ یک درخواست HTTP از یک خط سیر<sup>۳۱</sup>، احتمالاً تعدادی سرآیند<sup>۳۲</sup>، و شاید یک متن<sup>۳۳</sup> درخواست تشکیل شده است، که همگی آن‌ها صرفاً رشته‌هایی از داده هستند که بر روی یک اتصال TCP/IP ارسال می‌شوند. همان‌طور که در شکل ۲-۳ مشاهده می‌کنید، یک خط سیر<sup>۳۴</sup> HTTP از یک روش درخواست<sup>۳۵</sup>—که

۶۵۵۳۵ (۱ - ۶) بزرگترین  
 مقدار بی‌علامت است که در  
 فضای ۱۶ بیتی که در طراحی  
 اصلی پروتکل اینترنت به شماره  
 درگاه اختصاص داده شده بود،  
 جای می‌گیرد.

<sup>23</sup>Port (Networking)

<sup>24</sup>HyperText Transfer Protocol Secure (HTTPS)

<sup>25</sup>Public-key Cryptography

<sup>26</sup>Encryption

<sup>27</sup>Integrated Development Environment (IDE)

<sup>28</sup>Connection (Networking)

<sup>29</sup>Resource (Computing)

<sup>30</sup>Cookie (HTTP)

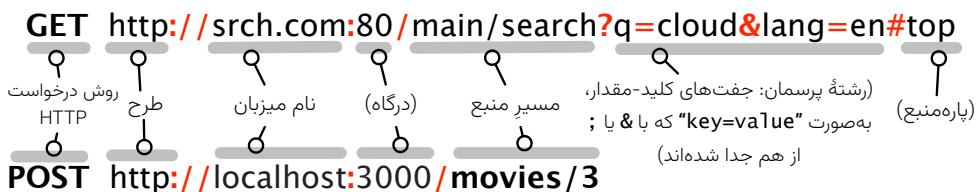
<sup>31</sup>Route

<sup>32</sup>Header (Computing)

<sup>33</sup>Body {of request/message/packet}

<sup>34</sup>استفاده از واژه «مسیر» در زبان فارسی به عنوان معادل هر دو واژه Route و Path معمول است، اما در ترجمه‌ای کتاب، ما برای متمایز بودن این واژگان از یکدیگر، از دو معادل مختلف استفاده کردیم. از این رو، ما «خط سیر» و «مسیر» را به ترتیب معادل Route و Path در نظر گرفته‌ایم. یک خط سیر در واقع در برگیرنده یک مسیر به همراه روش درخواست و سایر مؤلفه‌های دیگر است که در شکل ۲-۳ توضیح داده شده‌اند.

<sup>35</sup>Method (HTTP)



شکل ۲-۳: یک خط سیری HTTP از یک روش درخواست **HTTP** و یک **URI** تشکیل شده است و تمایل کلاینت را برای انجام عملیاتی روی منبع مشخص شده توسط **URI** بیان می‌کند. یک **URI** مانند **http://** با کلمه مخصوص کننده طرح یا پروتکل مورد استفاده برای دسترسی به منبع است، شروع می‌شود و شامل مؤلفه‌های فوق است. مؤلفه‌های اختیاری درون پرانتز هستند؛ اگر شماره درگاه حذف شود، به طور پیش‌فرض ۸۰ برای **HTTP** و ۴۴۳ برای **HTTPS** در نظر گرفته می‌شود. یک **URI** ناقص یک یا همهٔ مؤلفه‌های سمت چپ را حذف می‌کند که در این صورت آن مؤلفه‌ها نسبت به یک **URI** پایه که توسط نرم‌افزار کاربردی تعیین شده است، مشخص می‌شوند. بهتر است همواره از **URI**‌های کامل استفاده شود.

**URL یا URI ؟ URL‌ها گاهی**

اوقات با عنوان مکان یا بین منبع یکنواخت (**URL**) نیز شناخته می‌شوند. علیرغم تمایزهای فنی طبیعی بین این دو، برای اهداف ما در این کتاب، این اصطلاحات می‌توانند بهجای یکدیگر استفاده شوند. ما از **URI** استفاده می‌کنیم زیرا کل تر است و همچنین با اصطلاحات مورد استفاده اکثر کتابخانه‌ها مطابقت دارد.

ممکن‌باشی از روش‌های درخواست **GET** (دریافت)، **PUT** (ارسال)، **PATCH** (گذاشت)، **DELETE** (حذف) است—و یک **شناسانه منبع یکنواخت URI**<sup>۳۶</sup> تشکیل شده است. شما **URI** را به عنوان رشتۀ‌هایی که معمولاً با **http://** شروع می‌شوند و درون نوار آدرس مرورگر خود قرار می‌دهید می‌شناسید. اما نکته مهم این است که ترکیب روش درخواست **HTTP** و **URI** است که یک خط سیری را تعریف می‌کند: همان **URI** با روش‌های درخواست **HTTP** می‌تواند معانی مختلفی برای **SaaS** داشته باشد. ما در ادامه این فصل، در مورد مفهوم و ساختار خطوط سیری بیشتر صحبت خواهیم کرد اما به طور کلی، معمولاً به معنای «ارائهٔ یک کمی از منبع درخواستی به کلاینت بدون اعمال تغییری در منبع و یا هرگونه اثر جانبی» است، در حالی که **POST**، **PATCH**، **PUT** و **DELETE** معمولاً برای انجام عملیاتی استفاده می‌شوند که یک منبع را ایجاد می‌کند، تغییر می‌دهد و یا حذف می‌کند. هنگامی که از یک **URI** با قرار دادن آن در نوار آدرس مرورگر بازدید می‌کنید، مرورگر شما درخواستی از نوع **GET** به آن **URI** می‌دهد. وقتی که اطلاعات یک فرم را بعد از تکمیل ارسال می‌کنید، بسته به نحوه پیاده‌سازی صفحه، مرورگر ممکن است یک درخواست **GET** را به **POST**، **PATCH** یا **PUT** موردنظر بدهد. به دلایل تاریخی، اکثر مرورگرها مستقیماً درخواستی از نوع **PUT**، **PATCH** یا **DELETE** ایجاد نمی‌کنند. به زودی دربارهٔ نحوه کاربرد آن‌ها صحبت خواهیم کرد.

در تمرین پُنک بعدی، به طور عملی کار با **HTTP** را تمرین خواهید کرد، اما دانستن این پیش‌زمینه شما را در مسیر صحیح راهنمایی خواهد کرد. سادگی **HTTP** از دو ویژگی ناشی می‌شود. اول، یک پروتکل **درخواست-پاسخ** است: هر ارتباط و برهمنکش **HTTP** با درخواست کلاینت شروع می‌شود که سرور به آن پاسخ می‌دهد (مگر اینکه سرور دچار مشکل شده یا شبکه از دسترس خارج شود). یک درخواست **HTTP** حتماً باید شامل خط سیری و شماره نسخه پروتکل **HTTP** باشد و معمولاً سرآیندهایی را نیز به همراه دارد که اطلاعاتی در مورد کلاینت ارائه می‌دهند. شماره نسخه پروتکل **HTTP** به سرور می‌گوید که کلاینت کدام امکانات **HTTP** را می‌تواند درک کند، به طوری که (به عنوان مثال) سرورها می‌توانند از استفاده از امکانات جدیدتر **HTTP** اجتناب کنند، وقتی که کلاینت فقط یک نسخه پروتکل قدیمی‌تر را درک می‌کند. پاسخ سرور حتماً باید شامل شماره نسخه **HTTP** و یک کد وضعیت<sup>۳۷</sup> سه رقمی باشد که نتیجهٔ عملیات درخواستی را نشان می‌دهد. هر آنچه بعد از کد وضعیت در اولین خط پاسخ قرار گیرد اختیاری است و نادیده گرفته می‌شود، اما اغلب برای ارائهٔ یک پیام ساده قابل درک برای انسان از تفسیر کد وضعیت استفاده می‌شود. پیام پاسخ همچنین شامل سرآیندهایی است که بقیه اطلاعات درون پاسخ را توصیف می‌کنند، و به دنبال آن یک خط خالی و سپس محتوای داخلی پاسخ قرار می‌گیرد.

دوم، **HTTP یک پروتکل بدون حالت**<sup>۳۸</sup> است: هر درخواست **HTTP** مستقل و بی‌ارتباط با تمام درخواست‌های قبلی است. با این وصف، چگونه یک وبگاه می‌تواند اطلاعاتی مانند اینکه آیا شما

<sup>36</sup>Uniform Resource Identifier

<sup>37</sup>Status Code (HTTP)

<sup>38</sup>Stateless (Protocol)

احراز هویت شده‌اید را پیگیری کند؟ HTTP روشی مبتنی بر **کوکی** را برای این منظور فراهم می‌کند. اولین باری که یک کلاینت درخواستی را به یک سرور خاص ارسال می‌کند، سرور می‌تواند در پاسخ یک سرآیند با عنوان: **Set-Cookie** را قرار دهد که شامل اطلاعاتی است که سرور می‌تواند برای شناسایی کلاینت در درخواست‌های HTTP آنی استفاده کند. این مسئولیت کلاینت است که آن اطلاعات را جایی ذخیره کند تا از طریق یک سرآیند با عنوان **Cookie** در درخواست‌های بعدی به همان سرور ارسال کند. درست مانند برگ شماره‌ای که برای کلاینت معنای خاصی نداشته باشد و اگر دریافت می‌کنید، محتوای کوکی باید طوری باشد که برای تشخیص کلاینت توسعه سرور کافی محتوای آن دستکاری شود مشخص شود، ولی در عین حال برای تشخیص کلاینت پیشنهاد شده باشد تا امکان ارتباط دنباله‌دار بین سرور و آن کلاینت خاص در یک **نیشت<sup>۳۹</sup>** پیوسته فراهم شود. بنابراین، پروتکلهای بدون حالت، طراحی سرور را به قیمت پیچیدگی طراحی نرم‌افزارهای کاربردی، ساده‌تر می‌کنند، اما خوبی‌های موققی مانند ریزش‌ها را از بسیاری از این پیچیدگی‌ها محافظت می‌کند.

**لغت کوکی** از گذشته دور در اصطلاحات هکری به معنای «بک‌تکه‌ای از داده‌های غیرقابل تفسیر است که باید بعداً برای شناسایی خود یا دستیابی به یک کار ارائه شود» بوده است.

## چکیده

- مرورگرهای وب و سرورهای وب از **پروتکل انتقال ابرمنتن (HTTP)** برای برقراری ارتباط استفاده می‌کنند. HTTP با اتکا بر **TCP/IP** (مجموعه پروتکل اینترنت): پروتکل کنترل انتقال و پروتکل اینترنت) به طور قابل اعتماد یک توالی مرتباً شده از بایت‌ها را می‌مبدله کند.
- هر رایانه در شبکه TCP/IP دارای یک **نشانه آی‌پی** مثل ۱۷۲.۳۲.۲۴۴.۱۷۲ است، هر چند که **سامانه نام دامنه (DNS)** امکان استفاده از نام‌هایی که خواندن آن‌ها برای انسان‌ها راحت‌تر است را به جای نشانه آی‌پی می‌دهد. نام ویژه localhost (میزبان محلی) به رایانه در حال استفاده (محلی) اشاره دارد و به نشانه آی‌پی ویژه ۱۲۷.۰.۰.۱ ترجمه می‌شود.
- هر نرم‌افزار کاربردی که روی یک رایانه خاص اجرا می‌شود باید بروی یک **درگاه TCP** مشخص و متمایز، شماره‌گذاری شده از ۱ تا ۶۵۵۳۵ (۱ - ۲<sup>۱۶</sup>)، «گوش دهد». درگاه شماره ۸۰ توسط سرورهای HTTP (سرورهای وب) استفاده می‌شود.
- یک **شناسانه منبع یکنواخت (URI)** یک منبع موجود در اینترنت را نام می‌برد. تفسیر نام منبع از یک نرم‌افزار کاربردی به نرم‌افزار کاربردی دیگر متفاوت است.
- یک خط سیر HTTP از یک روش درخواست (مانند GET یا POST) به همراه یک URI تشکیل شده است. یک URI یکسان با روش‌های درخواست مختلف منجر به خطوط سیری مختلفی می‌شود که ممکن است در یک آپ SaaS خاص رفتار یکسانی داشته با نداشته باشند.
- یک پروتکل بدون حالت است که هر درخواست مستقل از هر درخواست دیگری است، حتی از همان کلاینت. **کوکی‌های HTTP** امکان ایجاد وابستگی و ارتباط بین درخواست‌های HTTP از یک کلاینت را فراهم می‌کنند. این مسئولیت مرورگر است که یک کوکی از سرور HTTP را پذیرد و اطمینان حاصل کند که آن کوکی را در درخواست‌های بعدی ارسال شده به آن سرور بگنجاند.

<sup>۳۹</sup>Session (Computing)

### ■ بیشتر بدانیم: چندخانگی و IPv6

ما در توضیحاتمان در مورد بسیاری از جنبه‌های TCP/IP بهشدت ساده‌سازی کردیم، از جمله استفاده از چندخانگی (Multihoming) برای اتصال دستگاهها به چندین شبکه به صورت همزمان و کاهش تدریجی استفاده از نسخه فعلی پروتکل اینترنت (IPv4) و جایگزینی آن توسعه نسخه ۶ آن (IPv6)، که از قابل متفاوتی برای نشانه‌های آی‌پی استفاده می‌کند. با این حال، از آنجایی که توسعه دهنده‌گان آپ‌های SaaS به ندرت مستقیماً با نشانه‌های آی‌پی سروکار دارند، این ساده‌سازی‌ها، توضیحات ما را تغییر چندانی نمی‌دهند.

### خودآزمایی ۱-۲-۳

آیا DNS یک پروتکل کلاینت-سرور است؟ چرا؟

- ◊ بله. کلاینت‌های DNS درخواست ترجمه نام‌ها را ارسال می‌کنند و سرورهای DNS به این درخواست‌ها پاسخ می‌دهند که البته برای آماده‌سازی پاسخ، گاهی نیاز به مشورت کردن با سایر سرورها نیز دارند (که در این حالت در واقع به طور موقت به عنوان کلاینت عمل می‌کنند). ■

### خودآزمایی ۲-۲-۳

آیا می‌توان که یک اتصال TCP بدون مشخص کردن شماره درگاهی ایجاد کرد؟ در صورت امکان، چه اتفاقی می‌افتد؟

- ◊ همه اتصالات TCP باید یک شماره درگاه را مشخص کنند. با این حال، انواع خاصی از کلاینت‌ها (مرورگرهای وب، رایانمeh خوانها و غیره) شماره‌های درگاه پیش‌فرض مورد استفاده برای آن سرویس‌ها را می‌دانند، بنابراین کاربران چنین کلاینت‌هایی به ندرت نیاز به دانستن این اطلاعات پیدا می‌کنند. ■

### خودآزمایی ۳-۲-۳

درست یا نادرست: HTTP به عنوان یک پروتکل، درکی از مفهوم «نشست» که متشکل است از دنباله‌ای از درخواست‌های HTTP مرتبط به یک وبگاه یکسان، را ندارد.

- ◊ درست. HTTP یک پروتکل بدون حالت است که هر درخواست از سایر درخواست‌های کلاینت کاملاً مستقل است. بنابراین سازوکاری نظیر استفاده از کوکی‌های HTTP برای ایجاد نشست به صورت انتزاعی لازم است. ■

### خودآزمایی ۴-۲-۳

بسیاری از سرورهای HTTP به کمک کوکی‌ها می‌توانند یک کلاینت را در درخواست‌های مکرر شناسایی کنند، به عنوان مثال، برای ردیابی اطلاعاتی مانند اینکه آیا آن کاربر به سامانه وارد شده است یا نه. حال اگر در مرورگر خود به کلی کوکی‌ها را غیرفعال و از چنین وبگاهی بازدید کنید، چه اتفاقی رخ خواهد داد؟

- ◊ امتحان کنید و ببینید. از یک موتور جستجو برای یافتن دستورالعمل نحوه غیرفعال کردن کامل کوکی‌ها (به طور موقت) در مرورگر خود استفاده و سعی کنید به وبگاهی که در آن یک حساب کاربری دارید وارد شوید. فراموش نکنید وقتی آزمایش خود را به پایان رسانید، کوکی‌ها را دوباره فعال کنید. ■

### شایستگی ۵-۲-۳

قسمت‌های ضروری یک درخواست HTTP را شناسایی کنید.

### شایستگی ۶-۲-۳

یک URI را با در کنار هم گذاشتن اجزای تشکیل‌دهنده آن (روش درخواست HTTP، پروتکل،

میزبان، مسیر، پارامترها) بسازید.

### شاپایستگی ۷-۲-۳

یک URI را به اجزای سازنده آن تجزیه کنید.

### شاپایستگی ۸-۲-۳

پیامدهای بدون حالت بودن HTTP را بر نحوه پیگیری وضعیت کاربران در آپ‌های SaaS شناسایی کنید.

## ۳-۳ پُفک: HTTP و URI‌ها

### ۳-۳ پُفک: HTTP و URI‌ها

<https://github.com/saasbook/hw-http-intro>

به کمک دستورها و ابزارهای قادرمند مبتنی بر خط فرمان که تمام توسعه‌دهندگان SaaS باید با آن‌ها آشناشی داشته باشند، آدرس‌های URI را بسازید و درخواست‌های HTTP را به صورت مستقیم ایجاد کنید. محتویات درخواست و پاسخ شامل سرآیدهای، کدهای خطأ و کوکی‌ها را بررسی کرده و بیاموزید.

## ۴-۳ از وبگاه‌ها تا ریزسرویس‌ها: معماری سرویس‌گرا

هیچ‌کس نباید یک پروژهٔ خیلی بزرگ را شروع کند. باید با یک پروژهٔ کوچک و پیش‌پاافتاده آغاز کنید و هرگز انتظار نداشته باشید که ان پروژهٔ بزرگ شود. اگر چنین انتظاری داشته باشید، فقط بیش از حد به طراحی می‌پردازید و به طور کلی تصویر می‌کنید که اهمیت پروژه بالاتر از حد واقعی آن در آن مرحله است. یا بدتر از آن، ممکن است از حجم کاری که در ذهن مجسم می‌کنید هراسان شوید و پا پس بکشید. بنابراین کوچک شروع و برروی جزئیات تمرکز کنید. به تصویر کلی با طراحی پرزرق‌ویرق فکر نکنید. اگر پروژهٔ شما پاسخگوی نیازهای نسبتاً فوری نباشد، احتمالاً بیش از حد به طراحی آن پرداخته شده است. و البته انتظار نداشته باشید که دیگران فوراً به شما ملحق شوند و کمک کنند. کارها این‌گونه پیش نمی‌رونند. شما باید ابتدا چیزی نیمه‌کاره و نسبتاً مفید داشته باشید و سپس دیگران می‌گویند: «او، این تقریباً به درد من می‌خورد» و آن وقت در پروژه درگیر می‌شوند و در توسعه آن مشارکت می‌کنند.

لینوس توروالدز<sup>۴۰</sup> در مصاحبه با پرستون سن پیر<sup>۴۱</sup>، منتشر شده در وبگاه Linux Times، به تاریخ ۲۵ اکتبر ۲۰۰۴

هنگامی که وب در سال ۱۹۹۰ شروع به کار کرد، سرورهای HTTP اساساً برای ارائه محتواهای ایستا (در اصل متن و تصاویر مقالات علمی) که مرورگرها نمایش می‌دادند وجود داشتند. هر بار که مرورگر درخواست HTTP دیگری می‌کرد، سرور یک صفحهٔ وب جدید را به مرورگر برای نمایش ارائه می‌داد. اما ظهور SaaS در حدود سال ۱۹۹۵ به سرعت کاربرد و عملکرد سرورها را تغییر داد: سرورها به جای بازگرداندن نسخه‌های ساده از صفحات وب با محتواهای ایستا، اکنون برنامه‌های را اجرا و صفحات HTML را «در لحظه» ایجاد می‌کردند که رابط کاربری آن برنامه را پیاده‌سازی می‌کرد. با این حال،

<sup>40</sup>Linus Torvalds

<sup>41</sup>Preston St. Pierre

هنوز هم چنین بود که هر درخواست HTTP جدید منجر به بارگیری دوباره و نمایش یک صفحهٔ جدید در مرورگر می‌شد. مرحلهٔ بعدی تکامل، ظهور **ایچکس<sup>۴۳</sup>** بود. اگر یک مرورگر از **ایچکس** پشتیبانی می‌کرد، صفحات می‌توانستند شامل کدهای نوشته شده به زبان جاوااسکریپت باشند و این کدها می‌توانستند درخواست‌های HTTP بعدی را بدون ایجاد بارگیری مجدد کل صفحه به سرور ارسال کنند. در پاسخ به این درخواست‌ها، سرور نه یک صفحهٔ HTML، بلکه یک داده‌ساختار در قالب XML یا **جی‌سان<sup>۴۴</sup>** را برمی‌گرداند که به‌زودی با هر دوی آن‌ها آشنا خواهیم شد. کد جاوااسکریپت در حال اجرا در مرورگر از آن داده‌ها برای تعیین نحوهٔ تغییر ظاهر یا رفتار صفحهٔ نمایش داده شده استفاده می‌کرد، بدون اینکه باعث بارگیری مجدد کل صفحه شود.

**ایچکس** نقطهٔ عطفی در رابطهٔ بین یک وبگاه و یک کلاینت بود: به جای دریافت صفحات HTML و انجام وظیفه به عنوان صرفاً یک موتور نمایش، کلاینت اساساً یک تابع را بر روی سرور دور دست فراخوانی می‌کرد و انتظار داشت که یک سری داده را دریافت کند، گویی کلاینت یک تابع را از کتابخانه‌ای فراخوانی می‌کرد. این تغییر دیدگاه، راه را برای نگرشی نو به دنیای وب باز کرد: دیدن آن به عنوان مجموعه‌ای از سرویس‌های مستقل که می‌توانند برای تولید یک وبگاه بزرگ‌تر کنار هم قرار گیرند—چیزی که به **معماری سرویس‌گرا (SOA)** معروف شد.

معماری سرویس‌گرا مدت‌ها بود که از عدم شفافیت، بی‌برنامگی و رهبری درست رنج می‌برد.... حتی ممکن بود به کلی نابود شود—نه به خاطر بی‌اهمیتی یا کمبود بنیه، بلکه صرفاً به دلیل گسترش به ظاهر بی‌پایان اطلاعات نادرست و سردرگمی ایجاد شده.

—توماس ارل<sup>۴۵</sup>، دریاره منشور معماری سرویس‌گرا، ۲۰۱۰

**اینترنت اکسپلورر مکروسافت**  
نسخه<sup>۵</sup>، که به‌نوعی جد مایکروسافت اچ محسوب می‌شود، اولین مرورگر بود که در سال ۱۹۹۸ از **ایچکس** پشتیبانی کرد. سرویس گوگل میز، که در سال ۲۰۰۵ راه‌اندازی شد، نمایشی چشمگیر از استفاده از **ایچکس** برای ساخت آپلهای تحت وبی بود که به معنای واقعی برهم‌کنشی بودند.

معماری سرویس‌گرا به عنوان یک الگوی معماري ممکن است دیدگاه تازه‌ای در ساختاربندی وب باشد، اما مطمئناً ایدهٔ تازه‌ای نبود. علی‌رغم تردیدهای اولیه در مورد اینکه آیا **SOA** چیزی بیش از یک «واژهٔ باب روز» برای بازاریابی است، یک شرکت بسیار برجسته (بی‌سروصدا و در داخل) مشغول ملموس و محکم‌تر کردن پایه‌های **SOA** بود. غول تجارت الکترونیک<sup>۴۶</sup>، آمازون، وبگاه خردفروشی خود را در سال ۱۹۹۵ راه‌اندازی کرد که به صورت یک نرم‌افزار کاربردی **تک‌سنگی<sup>۴۷</sup>** کار می‌کرد، یعنی یک نرم‌افزار کاربردی یکپارچه و بزرگ که مدیریت تمام جنبه‌های وبگاه را در اختیار داشت. به نقل از ویلگ<sup>۴۸</sup> استیو پِیگ<sup>۴۹</sup>، از کارمندان اسبق آمازون<sup>۵۰</sup>، در سال ۲۰۰۲ مدعی‌عامل و بنیان‌گذار آمازون، دستور به عمل تغییراتی در روند کاری این شرکت داد که امروزه عملًاً آن‌ها را **SOA** می‌نامیم. استیو پِیگ ادعا می‌کند که جف بِیِرُوس<sup>۵۱</sup> یک رایانه‌ای برای همهٔ کارمندان ارسال می‌کند که نکات اصلی آن طبق توضیحات استیو پِیگ مختصراً به شرح زیر است:

از این پس تمامی تیم‌های مسئول زیرسامانه‌های مختلف آمازون موظف‌اند داده‌ها و قابلیت‌های زیرسامانهٔ خود را فقط از طریق واسطه‌های به صورت سرویس در دسترس دیگران قرار دهند. هیچ زیرسامانه‌ای مجاز به دسترسی مستقیم به داده‌هایی که «متعلق» به زیرسامانهٔ دیگری است نخواهد بود؛ تنها راه دسترسی، از طریق واسطه خواهد بود که عملیات مشخصی را برروی آن داده‌ها فراهم می‌کند. افزون بر این، هر یک از این واسطه‌ها باید به‌گونه‌ای طراحی شوند که در آینده بتوانند برای توسعه‌دهندگان بیرونی نیز در دسترس قرار گیرند، نه فقط برای استفادهٔ داخلی در آمازون.

<sup>42</sup>AJAX (Asynchronous JavaScript And XML)

<sup>43</sup>JSON (JavaScript Object Notation)

<sup>44</sup>Thomas Erl

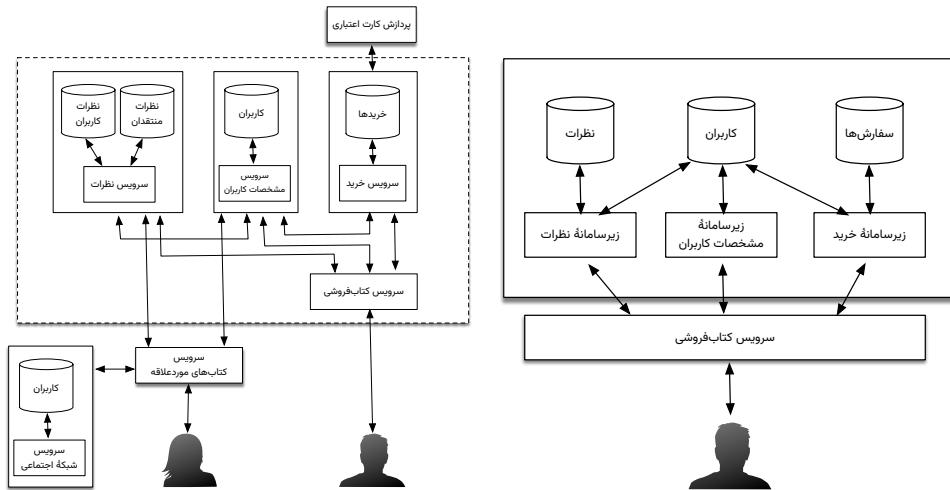
<sup>45</sup>E-commerce

<sup>46</sup>تک‌سنگ (**Monolith**) عارضه‌ای زمین‌شناختی است که از یک تودهٔ منفرد سنگ یا صخرهٔ عظیم‌الجهة تشکیل شده‌است. از این رو، در ادبیات حوزهٔ مهندسی نرم‌افزار برای توصیف نرم‌افزارهایی که به صورت یک تکه ساخته شده‌اند و حجم بزرگی دارند از این لفظ استفاده می‌شود. ما نیز معادل فارسی زمین‌شناختی این کلمه را در ترجمه در نظر گرفته‌ایم.

<sup>47</sup>Weblog

<sup>48</sup>Steve Yegge

<sup>49</sup>Jeff Bezos



شکل ۳-۳: سمت راست: نسخه یکتکه و ایزوله شده یک سرویس کتاب فروشی فرضی که همه زیرسامانه های آن در پشت یک واسط برنامه نویسی واحد هستند. سمت چپ: نسخه مبتنی بر معماری سرویس گرایی یک سرویس کتاب فروشی فرضی که در آن زیرسامانه ها از یکدیگر مستقل هستند و از طریق واسطه های برنامه نویسی در دسترس هستند.

در این خطمشی، جف بیزوس وجه تمایز کلیدی معماری سرویس گرا را به خوبی بیان می کند: تنها راهی که یک سرویس می تواند به داده های سرویس دیگری دسترسی پیدا کند، این است که عملیات مشخصی از طریق یک واسط خارجی که آن عملیات را ارائه می دهد، درخواست کند. به عنوان مثال، فرض کنید می خواهیم یک سرویس کتاب فروشی ساده ایجاد کنیم که در آن کاربران بتوانند نظرات خود درباره کتاب های خریداری شده شان را ارسال و لیست از کتاب های مورد علاقه شان را ذخیره کنند. ما به سه زیرسامانه نیاز داریم: نقد و بررسی کتاب (نظرات)، مشخصات کاربران و خرید. سمت راست شکل ۳-۳، نسخه یکتکه و ایزوله شده را نشان می دهد، شبیه به نحو کار ویگاه آمازون در سال ۱۹۹۵. هر زیرسامانه می تواند به اطلاعات یک زیرسامانه داخلی دیگر دسترسی داشته باشد. به عنوان مثال، زیرسامانه نظرات می تواند مستقیماً مشخصات یک کاربر را از زیرسامانه کاربران استخراج کند. تنها واسط قابل مشاهده خارجی خود «کتاب فروشی» است.

به عبارت دیگر، شما به عنوان یک توسعه دهنده و ب مستقل نمی توانید به سرویس مشخصات کاربری آمازون برای مدیریت کاربران در وبگاه تجارت الکترونیک خود دسترسی داشته باشید. این قابلیت فقط برای استفاده داخلی آمازون است. در مقابل، سمت چپ شکل ۳-۳، نسخه مبتنی بر معماری سرویس گرایی سرویس کتاب فروشی را نشان می دهد که در آن همه زیرسامانه ها مجزا و مستقل هستند. با وجود اینکه همه زیرسامانه ها در داخل «محدوده» کتاب فروشی قرار دارند، که به صورت مستطیل نقطه چین شده نشان داده شده است، آن ها طوری با یکدیگر برهمنکش دارند که گویی از هم جدا هستند. به عنوان مثال، اگر زیرسامانه نظرات بخواهد مشخصات کاربر را به روزرسانی کند تا نشان دهد کاربر نظری را نوشته است، نمی تواند مستقیماً به پایگاه داده کاربران دسترسی پیدا کند. در عوض، باید از **سرویس** مشخصات کاربران بخواهد که اطلاعات کاربر را از طریق هر واسطی که برای این منظور ارائه شده است، بروز کند. اگر چنین عملیاتی ارائه و تعریف نشده است، آن گاه تیم مسئول سرویس نظرات باید با تیم مربوط به پایگاه داده مشخصات کاربران مذاکره و آن ها را مجاب به ارائه عملیات لازم از طریق یک واسط کند.

یک **ریزسرویس**، سرویسی مستقل است که فقط یک نوع کار را انجام می دهد، و همچنین طوری طراحی شده است تا (شاید با پرداخت هزینه های) توسط هر سرویس خارجی دیگری قابل دسترسی باشد و یا در آن ها گنجانده و ادغام شود. در مثال کتاب فروشی، سرویس های مربوط به مشخصات کاربران و ثبت سفارش می توانند به صورت ریزسرویس ساخته شوند. اگرچه هیچ معیار سرراست و

معایب	مزایا
کارایی: هر فراخوانی یک سرویس شامل هزینه‌ای اضافه برای عبور از لایه‌های پُشتۀ نرم‌افزاری مربوط به رابط شبکه دارد. این هزینه اضافی معماری سرویس‌گرای می‌تواند منجر به نزول کارایی و سرعت کلی آپ بشود.	بازکاربردپذیری: دیگران می‌توانند مانند شکل ۳-۳، سرویس‌های موجود را برای ایجاد آپ جدید با یکدیگر ترکیب کرده. همچنین از آن جایی که جزئیات پیاده‌سازی هر ریزسرویس کاملاً در پشت واسطه برنامه‌نویسی آن پنهان است، می‌توان از مناسب‌ترین زبان یا چارجوب در پیاده‌سازی آن استفاده کرد.
مدیریت خرابی جزئی یا مقطعی: یک سیستم تک‌سنگی یا کار می‌کند یا خیر. ولی در یک سیستم مبتنی بر SOA ممکن است برخی سرویس‌ها کار کنند در حالی که برخی دیگر دچار اشکال و خرابی باشند که قابل اعتماد بودن سرویس را به چالش می‌کشد.	ازمون آسان‌تر: یک ریزسرویس فقط یک کار را انجام می‌دهد، بنابراین مورد آزمون قرار دادن هر ریزسرویس آسان‌تر است.
حجم کاری بیشتر برای توسعۀ نرم‌افزار: شما باید یک واسطه برای هر مؤلفه از سرویس خود طراحی و پیاده‌سازی کنید، به جای اینکه یک واسطه واحد برای کل وبگاه طراحی کنید. خوشبختانه راهکاری به نام REST، که در ادامه به آن خواهیم پرداخت، این کار را ساده می‌کند.	انطباق بیشتر با روش چاپک: فصل ۱ نشان می‌دهد که توسعۀ نرم‌افزاری چاپک با پروژه‌ها و تیمهای کوچک تا متوسط بهترین کارایی را دارد. SOA اجزاهه می‌دهد تا سرویس‌های بزرگ را با ترکیب کردن سرویس‌های کوچک‌تر ساخت. آن‌گاه ساخت و اداره این سرویس‌های کوچک‌تر را می‌توان به تیم‌های کوچک‌تر سپرد که روش چاپک را دنبال می‌کنند.
توسعۀ دهنده‌گان باید با اصول عملیات فناوری اطلاعات (برای راهاندازی و ادارۀ نرم‌افزار) آشنایی پیدا کنند و بالعکس، کسانی که پیش از این مسئول عملیات فناوری اطلاعات بودند می‌باشند با اصول توسعۀ نرم‌افزار آشنا باشند. این یک واقعیت در مورد SaaS امروزی است که تحت عنوان توسعۀ عملیات (Dev/Ops) شناخته می‌شود و ما در فصل ۱۲ به آن خواهیم پرداخت.	«خودتان می‌سازید، خودتان هم آن را اجرا و اداره می‌کنید!» (همان‌طور که ورنر فوگلز، مدیر ارشد فناوری سرویس‌های وب آمازون، بیان کرد): همان تیم جمیع‌جور پشت هر ریزسرویس، وظیفه توسعه، آزمون، راهاندازی و اداره آن را به عهده دارد، که این امکان را فراهم می‌کند تا در پاسخ به درخواست‌های مشتریان ریزسرویس را با سرعت بیشتری بهبود داد.

شكل ۴-۳: هر مزیت معماری سرویس‌گرای (SOA) که در ستون سمت راست لیست شده است، هزینه‌ای را به همراه خود دارد که در ستون سمت چپ آورده شده است. با وجود این، اگر SOA به خوبی انجام شود، در عمل به نظر می‌رسد که مزایای آن بیشتر از معایب باشد.

مشخصی وجود ندارد که حد و اندازه یک ریزسرویس را از یک سرویس متمایز کند، پیشوند ریز معنی تشویق به «تکیه کردن افراطی بر اصول SOA» است که در آن هر ریزسرویس یک وظیفهٔ خاص دارد و مسئول یک قابلیت محدود و تعریف شده است.<sup>۵۰</sup> به عنوان مثال، هنگامی که از گوگل میز در مرورگر استفاده می‌کنید، شبیه یک آپ واحد به نظر می‌رسد، اما «دسته‌هایی» از قابلیت‌های مرتبط که آپ از آن‌ها استفاده می‌کند، (ترسیم نقشه، محاسبه مسیرهای رانندگی، **کدبندی مکانی**<sup>۵۱</sup> نشانی‌ها و غیره) به صورت ریزسرویس‌هایی مجزا از یکدیگر برای کد جاوا‌اسکریپت پیاده‌سازی‌کننده، پدیدار می‌شوند. یک قانون سرانگشتش این است که محدودهٔ مسئولیت یک ریزسرویس باید منحصر به مجموعه‌ای از عملیات مرتبط روی گروهی از منابع نزدیک به هم باشد که به نوعی به یکدیگر گره خورده‌اند. با تمام این صحبت‌ها، از آن جایی که هیچ تمایز سفت و سختی بین یک سرویس و یک ریزسرویس وجود ندارد، ما در ادامه از واژهٔ **سرویس** استفاده خواهیم کرد و همچنین این تعریف نسبی ارائه شده در کتاب ناداریشونی و دیگران می‌کنیم: «یک (ریز)سرویس مؤلفه‌ای است با امکان به کاراندازی مستقل و با محدودهٔ کاری معین که از طریق ردوبدل کردن پیام، امکان تعامل‌پذیری (با سایر سرویس‌ها) را فراهم می‌کند.» بر اساس این تعریف، می‌توانیم بینیم که نسخهٔ تک‌سنگی<sup>۵۲</sup> کتاب‌فروشی ما نه تنها به دلیل اندازه‌اش، بلکه از آن جایی که مؤلفه‌های آن دسترسی مشترکی به پایگاه‌های داده دارند و در نتیجه امکان به کاراندازی مستقل را ندارند، نمی‌تواند «ریز» تلقی شود.

<sup>۵۰</sup> این توصیف در مورد پیشوند micro در زبان انگلیسی بوده است که در زبان فارسی و در ترجمهٔ ما نیز لحاظ شده است.

<sup>51</sup> Geocoding

<sup>52</sup> Monolithic

همانطور که شکل ۴-۳ نشان می‌دهد، انتخاب یک معماری سرویس‌گرا به جای یک معماري تک‌سنگی، هم مزایایی دارد و هم معایبی. به طور خلاصه، ما می‌توانیم ریزسرویس‌ها را به عنوان تجلی اعمال کردن برنامه‌نویسی مفطر (XP) بر معماری سرویس‌گرا در نظر بگیریم؛ اگر قابلیت تکامل مستقل سرویس‌ها برای ما مطلوب است، پس هر کدام را تا حد امکان فشرده کنیم تا این استقلال به حد اکثر برسد. ریزسرویس‌ها ممکن است نتیجهٔ یک طراحی ازپیش‌انجام شده باشند یا در اثر تقسیم کردن یک سرویس بزرگ موجود به وجود بیایند، درست مشابه آنچه در توییتر اتفاق افتاد. در حوالی سال ۲۰۱۳<sup>۵۴</sup>، آن‌ها نرم‌افزار کاربردی تک‌سنگی نوشته شده بر پایهٔ ریز خود را، که به طنز آن را بین خودشان «مونوریل»<sup>۵۵</sup> خطاب می‌کردند، به تعداد زیادی ریزسرویس که اکثراً به زبان‌های جاوا، اسکالا یا کلوژ<sup>۵۶</sup> نوشته شده بودند، تقسیم کردند (کریکوریان ۲۰۱۳).

### چکیدهٔ معماری سرویس‌گرا و ریزسرویس‌ها

- اگرچه این اصطلاح در میان انبوهی از سردگرمی‌ها تقریباً به فراموشی سپرده شده بود، معماري سرویس‌گرا (SOA) صرفاً به رویکردی در توسعهٔ نرم‌افزار اشاره دارد که در آن زیرسamanه‌ها تنها از طریق واسطه‌های خارجی می‌توانند به داده‌های یکدیگر دسترسی پیدا کنند.
- از سال ۱۹۹۰ تا ۲۰۱۰، وب دست‌خوش تحول شد: از ارائه محتوای ایستا (وب/۰/۱) به ارائه رابطه‌ای کاربری بوسیلهٔ SaaS (SaaS)، سپس به امکان برهمنشی‌های مداوم پس از اولین بارگیری صفحه (ایچ‌کس) و در نهایت به معماری و ساخت آپ‌های بزرگ با ترکیب سرویس‌های مستقل (SOA) تبدیل شد.
- اگرچه مرز روشی وجود ندارد که یک سرویس را از یک ریزسرویس متمایز کند، یک ریزسرویس باید مجموعه‌ای از عملیات مشخصی را بر روی گروهی از منابع مرتبط انجام دهد. یک ریزسرویس همچنین باید به طور مستقل و کاملاً مجزا به کاراندازی و اداره شود (معمولًاً توسط همان تیمی که آن را می‌سازد) و باید به گونه‌ای طراحی شود که به راحتی در سایر سرویس‌های خارجی گنجانده و ادغام شود.

### ■ بیشتر بدانیم: ریزسرویس‌ها و فلسفهٔ یونیکس

تصمیم‌های دقیق در طراحی یونیکس باعث تاثیرگذاری فراگیر این سیستم عامل در سرتاسر علم مهندسی نرم‌افزار شده است. به طور خاص، «فلسفهٔ یونیکس» ساخت مؤلفه‌های ساده‌ای را ترویج می‌کند که فقط یک کار را انجام می‌دهند و به راحتی قابل ترکیب کردن هستند، به این ترتیب که خروجی هر مؤلفه باید ورودی برای یک مؤلفه دیگر باشد. ریزسرویس‌ها را می‌توان به عنوان پیروزی و موفقیت فلسفهٔ یونیکس در دنیای SaaS در نظر گرفت: یک ریزسرویس باید فقط یک کار را به خوبی و انجام دهد و کمترین فرض ممکن را در مورد نحوهٔ ادغام آن در یک معماری سرویس‌گرا بزرگ‌تر داشته باشد.

<sup>۵۴</sup>MonoRail (MonoRail) نوعی از راه‌آهن و قطاریست که مسیر آن از یک ریل تشکیل شده است. MonoRail در اینجا در واقع بازی با کلمات تک‌سنگی (Monolithic) و چارچوب ریز (Rails) است و از ترکیب این دو کلمه ساخته شده است.

<sup>۵۵</sup>Clojure [Lisp Dialect]

**خودآزمایی ۱-۴-۳**

یک برداشت دیگر از SOA می‌تواند این باشد که این یک روش معقول و واضح برای بهبود بهره‌وری برنامه‌نویسی است. بر این اساس، کدام سازوکار بهره‌وری را به بهترین نحو نشان می‌دهد:وضوح به کمک اختصار، سنتر، بازکاربرد و یا خودکارسازی و استفاده از ابزارها؟  
بازکاربرد! هدف از نمایان کردن و در دسترس گذاشتن واسطه‌های داخلی برای دنیای خارج این است که برنامه‌نویسان بتوانند از تجربیات و محصولات یکدیگر بهره ببرند و یا به‌اصطلاح بر شانه‌های یکدیگر بایستند. ■

**شاپیستگی ۲-۴-۳**

ویژگی‌های) کلیدی و متمایزکننده معماری سرویس‌گرا را مرور کنید.

**۵-۳ واسطه‌های برنامه‌نویسی مبتنی بر REST: همه چیز نوعی منبع است**

یک واسط برنامه‌نویسی که قابل فهم نباشد، قابل استفاده نیست.

— جیمز گاسلینگ<sup>۵۵</sup>، مخترع زبان برنامه‌نویسی جawa

مبنای معماری سرویس‌گرا این است که هر سرویس مجموعه‌ای مشخص و تعریف شده از عملیات‌ها را برروی یک یا چند نوع منبع مرتبط ارائه می‌دهد—مشابه یک کتابخانه برای یک زبان برنامه‌نویسی. به عبارت دیگر، کلاینت‌ها باید بتوانند تابع موردنظر برروی سرور را بنامند و فراخوانی کنند، آرگومان‌هایی را به آن ارسال، مقادیر بازگشته را دریافت، استثناهای سرور (خطاهای زمان اجرا) را شناسایی و مدیریت کنند وغیره؛ درست همان‌طور که یک نرم‌افزار کاربردی تابعی از یک کتابخانه را فراخوانی می‌کند، با این تفاوت که همه‌ای این موارد تحت محدودیت‌های استفاده از HTTP برای ارتباط انجام می‌گیرد. اصطلاح API با واسط برنامه‌نویسی به «قرارداد» بین فراخوانی‌دهنده و فراخوانی‌گیرنده<sup>۵۶</sup> اشاره دارد، خواه این فراخوانی‌دهنده یک برنامه باشد که تابعی را از یک کتابخانه فراخوانی کرده، یا یک کلاینت SaaS که سرویسی را بر روی یک سرور SaaS فراخوانی می‌کند، همان‌طور که در شکل ۵-۳ نشان داده شده است.

متأسفانه، ردیفهای دوم و سوم آورده شده در شکل مشکل‌ساز هستند، چرا که HTTP راهی برای «نام‌گذاری یک تابع دوردهست» یا «ارسال پارامترها» تعریف نمی‌کند، زیرا آن وظایف هرگز بخشی از طراحی اولیه آن نبوده‌اند. به طور خاص، مشخصات HTTP و URI هیچ قراردادی در مورد معناشناصی (مفهوم ضمنی) نحوه ساختاردهی یک URI یا چگونگی انجام این وظایف ارائه نمی‌دهند. از همان ابتدا، این امر به‌طور گستردۀ پذیرفته شد که استاندارد سازی این قراردادها برای چنین ارتباطاتی می‌تواند امکان ایجاد اکوسیستمی را فراهم کند که در آن هر کلاینت، نه فقط یک مرورگر وب، بتواند به روش‌های مختلف از یک سرور استفاده کند.

در دنیای ریزسرویس‌ها، این قراردادها عموماً توسط REST<sup>۵۷</sup>، که کوتاه‌نوشت عبارت انتقال بازنمودی حالت است، بیان می‌شوند. در سال ۲۰۰۰، روی فیلیدینگ<sup>۵۸</sup>، یک محقق علوم رایانه، REST را در رساله دکتری خود به عنوان راهی برای نگاشت درخواست‌ها به گُشنهای<sup>۵۹</sup> پیشنهاد کرد که به‌ویژه برای معماری سرویس‌گرا مناسب است. REST یک استاندارد نیست، بلکه یک رویکرد طراحی در مورد نحوه ساخت یک سرویس است و به تبع آن، در مورد شکل ظاهری واسط برنامه‌نویسی آن سرویس است. ایده روی فیلیدینگ این بود که موجودیت‌های مختلفی که توسط یک آپ تحت وب مدیریت می‌شوند، می‌بایست به عنوان منابع نمایش داده و بازنمایی شوند (از این رو، بازنمودی).

<sup>55</sup>James Gosling

<sup>56</sup>Callee (Function/Method)

<sup>57</sup>Representational State Transfer

<sup>58</sup>Roy Fielding

<sup>59</sup>Action (Software Engineering)

برنامهٔ پایتون (فراخوانی‌دهنده) مُثُدی از یک بستهٔ یا کتابخانهٔ پایتون (فراخوانی‌گیرنده) را فراخوانی می‌کند	یک کلاینت SaaS (فراخوانی‌دهنده) سرویس SaaS دیگری (فراخوانی‌گیرنده) را فراخوانی می‌کند
۱- فراخوانی‌دهنده چگونه فراخوانی‌گیرنده را شناسایی می‌کند؟	یک نقطهٔ پایانی یک آدرس منطقی است که کلاینت‌ها برای استفاده از سرویس، با آن تماس برقرار می‌کنند. این آدرس معمولاً به‌شکل یک «URI پایه» است که در واقع یک پیشوندی از URI (شامل نام میزبان ریزسرویس و در صورت لزوم، شماره درگاه) به‌شمار می‌رود که بین همهٔ فراخوانی‌های API به آن نقطهٔ پایانی مشترک است.
۲- چه عملیاتی فراخوانی شده است؟	نام مُثُد که در کد آمده است. به طور مثال: <code>numpy.array(...)</code>
۳- فراخوانی‌دهنده چگونه پارامترهای الزامی و اختیاری فراخوانی مُثُد ارسال می‌شوند. به طور مثال: <code>([[1, 2, 3]])</code>	پارامترها ممکن است که به عنوان بخشی از مسیر URI، به‌شکل جفت‌های کلید-مقدار در قسمت رشتهٔ پرسمان از URI یا به عنوان پایه‌بار در قالب JSON یا XML در متن درخواست ارسال شوند.
۴- فراخوانی‌دهنده چگونه مقدار بازگشتی را دریافت می‌کند؟	سرویس معمولاً یک داده‌ساختار در قالب XML یا JSON بازمی‌گرداند.
۵- فراخوانی‌گیرنده چگونه خط را اعلام می‌کند؟	سرویس یک کد وضعیت HTTP مناسب را برای نشان دادن نوع خطاب بازمی‌گرداند و معمولاً یک پیام خطاب به عنوان بخشی از داده‌ساختار بازگشتی ارائه می‌دهد.

شکل ۳-۵: واسطه‌های برنامه‌نویسی هر نوع فرادراد بین فراخوانی‌گیرنده و فراخوانی‌دهنده را توصیف می‌کنند، خواه این فراخوانی یک سرویس در معماری سرویس‌گرا باشد یا یک تابع کتابخانه‌ای در پایتون. مستندات API مشخص می‌کند که چه عملیاتی موجود است، URI‌ها چگونه باید برای فراخوانی آن عملیات ساخته شوند، خطاهای گزارش می‌شوند و چه الزامات عملیاتی دیگری باید رعایت شوند (برای مثال، محدود کردن تعداد فراخوانی‌ها در روز برای یک ریزسرویس یا ارسال کدن یک شناسه حساب یا گذرواژه با هر فراخوانی API).

همچنین خطوط سیر به گونه‌ای ساخته شوند که هر درخواست HTTP شامل تمام اطلاعات لازم برای شناسایی هم منبع و هم گنیش‌هایی که قرار است ببروی آن انجام شود باشد، که این گنیش ممکن است منجر به تغییر حالت در یک یا چند منبع شود (از این رو، انتقال حالت). یک واسط برنامه‌نویسی که از رهنمودهای روی فیلیدینگ پیروی کند، «مبتنی بر REST» یا RESTful نامیده می‌شود و خطوط سیری (روش‌های درخواست HTTP به علاوه URI) که توسط واسط برنامه‌نویسی برای انجام گنیش‌های خاص تعریف شده‌اند، خطوط سیر مبتنی بر REST نامیده می‌شوند.

اگرچه توضیح REST ساده است، اما این اصل به شکلی غیرمنتظره برای ساده‌سازی و سازمان‌دهی نرم‌افزارهای کاربردی SaaS قدرتمند است، چرا که طراح آپ را وادار می‌کند با دقت یه این فکر کند که هر موجودیتی که توسط آپ مدیریت می‌شود جگونه‌ی منبع یک عنوان یک منبع بازنمایی شود، چه عملیاتی می‌توان روی آن منبع انجام داد، و چه شرایط یافرضیات باید برقرار باشند تا درخواست برای چنین عملیاتی به صورت مستقل و خودکفا باشد. هر عملیات بر پایه واسط برنامه‌نویسی مبتنی بر REST، باید طوری طراحی شود تا پاسخ به پرسش‌های زیر به راحتی قابل دستیابی باشد:

- ۱- منبع اصلی که تحت تأثیر این عملیات قرار می‌گیرد چیست؟
- ۲- عملیاتی که باید روی آن منبع انجام شود چیست؟ نتایج احتمالی چه هستند؟ چه اثرات و عوارض جانبی احتمالی وجود دارد؟
- ۳- برای انجام عملیات، چه داده‌های دیگری لازم است؟ و در صورت لزوم، این اطلاعات چگونه مشخص می‌شوند؟

به عنوان مثال، به پاسخ پرسش‌های بالا در مورد فرایند ارسال نقد برای فیلم «۲۰۰۱: ادیسه فضایی»<sup>۶۰</sup>، که از فیلم‌های کلاسیک مورد علاقهٔ یکی از نویسندهای این کتاب است توجه کنید:

- ۱- منبع اصلی در این مثال، یک نقد یا نظر جدید برای فیلم «۲۰۰۱: ادیسه فضایی» است که ممکن است مثلاً شامل یک امتیاز عددی و یک متن چندخطی باشد.

۲- عملیات مورد نظر، ایجاد یک نقد جدید با استفاده از اطلاعات داده شده است. یکی از نتایج احتمالی موفقیت است، با این اثر جانبی که یک نقد جدید ایجاد می‌شود. نتیجه احتمالی دیگر این است که ایجاد این نقد به دلایل مختلفی ممکن است شکست بخورد (شاید این کلایینت مجاز به ارسال نقد نباشد، یا پایگاه داده پر است و جا ندارد، یا دیگر اجازه ارسال نقد برای این فیلم وجود ندارد)، در این صورت، هیچ اثر و عارضهٔ جانبی نخواهد داشت.

۳- علاوه بر اطلاعات مربوط به خود نقد، داده اضافی دیگری که می‌تواند لازم باشد، نوعی دادهٔ شناسایی برای این است که نشان دهد این نقد به کدام فیلم مرتبط است. همانطور که خواهیم دید، این شناسه احتمالاً به عنوان بخشی از خط سیر ارسال می‌شود، یا به عنوان یک مؤلفه در بخش مسیر<sup>۶۱</sup> URI یا به عنوان یک پارامتر در بخش رشتۀ پرسمان متعلق به URI. همچنین، اگر این آپ این امکان را به صورت اختیاری داشته باشد تا نام یا شناسه نویسنده نقد نیز به همراه نقد آورده شود، آنگاه شناسه نویسنده نیز جزئی از اطلاعات ضروری خواهد بود.

مستندات API باید توضیح دهد که چه عملیاتی موجود است و چگونه آرگومان‌های اجباری و اختیاری برای هر عملیات می‌باشد. همانطور که در فصل ۴ خواهیم آموخت، ریلز و سایر چارچوب‌ها به طور پیش‌فرض از خطوط سیر مبتنی بر REST پشتیبانی می‌کنند و از سازوکارهایی برای تعریف آسان آن برخوردار هستند.

در پایه‌ای ترین شکل خود، یک واسط برنامه‌نویسی مبتنی بر REST حداقل پنج عملیات را ببروی یک منبع تعریف می‌کند که با سروازۀ CRUDI<sup>۶۲</sup> (ایجاد، خواندن، به‌روزرسانی، حذف و فهرست کردن)

<sup>۶۰</sup>2001: A Space Odyssey (Movie)

<sup>۶۱</sup>Path (Computing)

<sup>۶۲</sup>Create, Read, Update, Delete, and Index

شناخته می‌شوند: ایجاد یک نمونه جدید از یک منبع، خواندن یا بازیابی یک نسخه از منبع، بهروزسازی و ایجاد تغییرات در یک منبع، حذف یک منبع و در آخر، فهرست کردن تمام منابع موجود از یک نوع خاص که ممکن است بر اساس معیارهای خاصی فیلتر شود. بسیاری از واسطه‌های برنامه‌نویسی پا را فراتر گذاشته و عملیات بیشتر و بهخصوصی را برای منابع مورد استفاده توسعه آن سرویس تعریف می‌کنند. تقریباً همیشه، به هر منبع یک شناسه یکتا (ممولاً یک عدد) داده می‌شود که به عنوان شناسه دائمی آن منبع عمل می‌کند و هرگز برای منبع دیگری، حتی پس از حذف، استفاده نمی‌شود. یک الگوی استفاده متدالوی برای واسطه‌های برنامه‌نویسی مبتنی بر REST این است که فهرستی از منابع (به همراه شناسه‌هایشان) را مطابق با معیارهای یک عملیات جست‌وجو بازگرداند؛ سپس کلاینت می‌تواند منابع مورد نظر را یک‌به‌یک از طریق شناسه‌های آن‌ها بازیابی کند. مطابق با قسمت ۳-۲، خطوط سیر مبتنی بر REST که گنیش‌های آن‌ها اثر جانی ندارند، معمولاً از روش درخواست GET استفاده کرده، درحالی‌که آن‌هایی که اثرات جانبی دارند از روش‌های درخواست DELETE، POST، PATCH یا PUT استفاده می‌کنند.

اکنون می‌توانیم به‌طور مشخص به این بپردازیم که چگونه واسطه‌های برنامه‌نویسی یک سرویس مبتنی بر REST به الزامات آورده شده در ردیفهای ۱ تا ۳ در شکل ۵-۳ پاسخ می‌دهند. (قویاً توصیه می‌کنیم که در حین خواندن ادامه این قسمت، مستندات API آورده شده در وبگاه مختلف یک URI را نشان می‌دهد. واسطه‌های برنامه‌نویسی مبتنی بر REST چندین قرارداد را برای نحوه ساخت این اجزا هنگام انجام یک فراخوانی API رعایت می‌کنند:

- بخش نام میزبان از URI بیانگر این است که چه سرور یا سرویس دهنده‌ای این سرویس را ارائه می‌دهد: <https://api.themoviedb.org>

• علاوه بر این، بیشتر سرویس دهنده‌هایی که واسطه‌های برنامه‌نویسی مبتنی بر REST می‌دهند، یک **URI پایه** یا پیشوندی مشترک برای URI مشخص می‌کنند که باید در همه فراخوانی‌های API به URI اضافه شود. در مستندات API می‌توانید ببینید که تمام URI‌ها با <https://api.themoviedb.org/4/> شروع می‌شوند. به این URI پایه (نام میزبان به همراه پیشوند)، گاهی **نقطه پایانی**<sup>۶۳</sup> واسطه برنامه‌نویسی گفته می‌شود؛ که یک آدرس منطقی است که کلاینت‌ها برای استفاده از سرویس، با آن تماس برقرار می‌کنند.

• در این مورد، مستندات API به ما می‌گوید که پیشوند مشترک در نام نقطه پایانی (4) به شماره نسخه API اشاره دارد. قرار دادن شماره نسخه به عنوان بخشی از URI این امکان را فراهم می‌کند که API با حفظ سازگاری با کلاینت‌های قدیمی‌تر، تکامل یابد. همچنین ممکن است نمونه‌های مختلف دیگری مانند <https://themoviedb.org/api/v4/> و <https://api.themoviedb.org/v4/> کنید.

• مؤلفه‌های بخش مسیر URI که پس از پیشوند می‌آیند، عملیات مورد نظر برای اجرا و منبعی که باید بر روی آن عملیات انجام شود را مشخص می‌کنند. مستندات API اغلب از نشان دونقطه (: ) یا آکولاد برای نشان دادن یک مؤلفه URI متناظر با شناسه منبع استفاده می‌کند. به عنوان مثال، در مستندات API ممکن است به این اشاره شده باشد که خط سیر اطلاعات کامل فیلم خاصی را درخواست می‌کند که شناسه عددی آن به جای {movie\_id} در URI جایگزین [https://api.themoviedb.org/v4/movie/{movie\\_id}](https://api.themoviedb.org/v4/movie/{movie_id}) باشد.

در قسمت بعدی، نحوه دقیق قالب‌بندی داده‌های مرتبط با این درخواست‌ها و نحوه رسیدگی به خط‌ها را توضیح می‌دهیم (ردیفهای ۴ و ۵ در شکل ۵-۳).

<sup>63</sup>Endpoint (Communication)

## مفرد یا جمع؟ در برخی از سیکهای URI های REST

جمله آنچه در یاز استفاده می شود، از نامهای جمع برای منابعی استفاده می شود که بیش از یک نمونه از آن نوع می توان وجود داشته باشد، مانند `movies/35` و `GET /movies`؛ از نامهای مفرد برای منابعی که فقط یک نمونه از آن وجود دارد، مانند `GET /homepage`

یکی از ویژگی های رایج اما نه همه گیر واسطه های برنامه نویسی مبتنی بر REST این است که ساختار مسیر URI، خود اطلاعاتی را درباره روابط بین انواع منابع نشان می دهد. برای مثال، خط سیر `GET /movie/{movie_id}/reviews` فیلم<sup>۶۴</sup>، تمام نقد های یک فیلم خاص را بازیابی می کند - در واقع اجرای عملیات فهرست کردن (`Index`) بر روی نقد ها، منتهی محدود به یک فیلم خاص. ساختار URI نشان می دهد که همان فیلم می تواند نقد های زیادی مرتبط با خود داشته باشد (یک رابطه به اصطلاح "has-many" یا "یک-به-چند") که در فصل ۵ با آن آشنا خواهیم شد). به طور مشابه، یک خط سیر فرضی مانند `GET /movies/5/reviews/22` برای درخواست محتوای نقد با شناسه ۲۲ مرتبط با فیلم با شناسه ۵، ممکن است زائد به نظر برسد، زیرا شناسه یک نقد به خودی خود منحصر به فرد است؛ اما باز هم ساختار خط سیر یک رابطه ای را که در غیر این صورت آشکار نبود، نشان می دهد. اما تمام سرویس ها و وبگاه های مبتنی بر REST این رویه را دنبال نمی کنند: خط سیر سرویس پایگاه داده فیلم برای یک نقد خاص در واقع فقط `GET /reviews/{review_id}` است و برخی از خطوط سیر این سرویس از چندین مؤلفه مسیر (عبارات جدا شده با خط مورب<sup>۶۵</sup>) برای بیان عملیات های فرعی مختلف بر روی یک منبع استفاده می کنند، نه برای بیان روابط بین انواع منابع.

شما می توانید با مطالعه مستندات مربوط به واسط برنامه نویسی سرویس پایگاه داده فیلم تأیید کنید که درخواست برای بازیابی تمام نقد های یک فیلم، در واقع بخشی از محتوای هر نقد را نیز برمی گردد. در یک واسط برنامه نویسی مبتنی بر REST «خالص»، چنین فراخوانی از نوع فهرست کردن ممکن است تنها یک فهرست از شناسه های نقد ها را بازگردد و کلاینت سپس بتواند محتوای هر نقد را به طور جداگانه و با استفاده از شناسه آن ها بازیابی کند. ممکن است که واسط برنامه نویسی این سرویس به دنبال کارآمدتر کردن کارها باشد و اطلاعات کافی را برای هر نقد بازگردد تا کلاینت بتواند تصمیم بگیرد که کدام نقد ها ارزش دریافت جزئیات بیشتر را دارند. اما اگر یک فیلم هزاران نقد داشته باشد، بازگردن محتوای نقد ها به جای شناسه های آن ها ممکن است سنگین و غیر عملی شود.

ممکن است که طراحی مبتنی بر REST یک انتخاب بدیهی به نظر برسد، اما تا زمانی که روی فیلدهای REST را به صورت غیر مبتنی بر REST طراحی می شدند. شکل ۶-۳ نشان می دهد که چگونه یک وبگاه تجارت الکترونیک فرضی غیر مبتنی بر REST ممکن است قابلیت هایی مانند ورود کاربر، افزودن یک کالای خاص به سبد خرید و ادامه فرایند خرید تا پرداخت را پیاده سازی کند. برای وبگاه فرضی غیر مبتنی بر REST، هر مرحله پس از ورود به سامانه (مرحله ۱) به اطلاعات ضمنی وابسته است: مرحله ۲ فرض می کند که وبگاه «به خاطر می آورد» که کاربر وارد شده فعلی کیست تا صفحه خوش آمدگویی را به او نشان دهد. همچنین، مرحله ۵ فرض می کند که وبگاه، برای تکمیل فرایند خرید و پرداخت، «به خاطر می آورد» که چه کسی کالاهایی را به سبد خرید اضافه کرده است. در مقابل، هر URI برای وبگاه مبتنی بر REST شامل اطلاعات کافی برای انجام درخواست بدون نیاز به تکیه بر اطلاعات ضمنی است: پس از اینکه دیو (Dave) وارد سامانه می شود، این حقیقت که شناسه کاربری او است، در هر درخواست وجود دارد، و سبد خرید او به صراحت توسط شناسه کاربری اش مشخصی شده است، نه اینکه به صورت ضمنی و بر اساس مفهوم کاربر وارد شده فعلی مشخص شده باشد.

<sup>64</sup>The Movie Database (TMDb)

<sup>65</sup>Slash (Punctuation)

URI مرتبط با REST و بگاه مبتنی بر REST	URI مرتبط با REST و بگاه غیرمبتنی بر REST	
POST /login/dave	POST /login/dave	۱- ورود به وبگاه
GET /user/301/welcome	GET /welcome	۲- صفحه خوشآمدگویی
POST /user/301/add/427	POST /add/427	۳- افزودن کالای شماره ۴۲۷ به سبد خرید
GET /user/301/cart	GET /cart	۴- مشاهده سبد خرید
POST /user/301/checkout	POST /checkout	۵- پرداخت

شکل ۶-۳: درخواست‌ها و خطوط سیر غیرمبتنی بر REST به نتایج درخواست‌های قبلی واپس‌تَه‌اند، اما این واپتگی‌ها را به صورت صريح به عنوان يخشی از درخواست فعلی نشان نمی‌دهند. در يك معماري سرويس‌گرا، يك کلاینت و بگاه مبتنی بر REST می‌تواند مستقیماً درخواست مشاهده سبد خرید را ارسال کند (خط ۴)، اما يك کلاینت و بگاه غیرمبتنی بر REST يابد ابتدا مراحل ۱ تا ۳ را انجام دهد تا اطلاعات ضمنی لازم برای مرحله ۴ فراهم شود.

### چکیده

- برای اینکه بتوان يك يا چند آپ SaaS را به عنوان «سرویس» در نظر گرفت که قادر به پذیرش فراخوانی‌های رویه‌ای دوردست از سوی يك کلاینت باشند، این موارد لازم‌اند: امکان شناسایی سرویس و همچنین عملیاتی (کدام تابع) که قرار است فراخوانی شود، ارسال داده به سرویس و دریافت داده از آن و در نهایت، مدیریت خطاهای.
- اگرچه هیچ استاندارد اجباری‌ای در مورد نگاشت بین خطوط سیر HTTP و عملیات تعریف شده واسطه‌های برنامه‌نویسی يك سرویس وجود ندارد، REST (انتقال بازنمودی حالت) به عنوان يك روش ساده و منسجم برای انجام این کار مطرح شده است که به خوبی با فناوری‌های وب سازگار است.
- ایده کلیدی در REST این است که هر نوع موجودیت مدیریت شده توسط سرویس، به عنوان يك منبع معرفی و نشان داده شود و مجموعه‌ای محدود از عملیات (معمولًا ایجاد، خواندن، به روزرسانی، حذف و فهرست کردن) که می‌توان روی آن منبع انجام داد، ارائه شود. درخواست‌های مبتنی بر REST آن‌گاه شامل تمام اطلاعات موردنیاز برای انجام گیش مشخص شده روی آن منبع خواهند بود.

### ■ بیشتر بدانیم: جداسازی فرمان-درخواست

پرتراند مایر، پژوهشگر بر جسته حوزه مهندسی نرم‌افزار، از مدت‌ها پیش بر اصل جداسازی فرمان-درخواست تأکید کرده است (مایر ۱۹۹۷): يك مُند یا عملیات باید یا منجر به تغییر داده‌ها شود (فرمان) یا فقط داده‌ها را بخواند (درخواست)، اما نه هر دو. REST این اصل را رعایت می‌کند، نه تنها با جدا کردن این دو نوع عملیات، بلکه طبق قواعد «REST خالص»، به هر عملیات تنها يك مسئولیت اختصاص داده می‌شود: يك فراخوانی واسط برنامه‌نویسی، یا داده‌ای را بازمی‌گرداند (یك منبع خاص یا احتمالاً مجموعه‌ای فیلترشده از منابعی از يك نوع خاص)، یا يك منبع از يك نوع خاص را ایجاد، به روزرسانی یا حذف می‌کند.

### ۱-۵-۳ خودآزمایی

کدام يك از اين خطوط سیر برای به روزرسانی اطلاعات فيلم با شناسه ۳۵ از اصول خوب HTTP و REST پیروی می‌کند:

- |                      |                     |
|----------------------|---------------------|
| (الف) POST /movie/35 | (ب) POST /movies/35 |
| (پ) PUT /movie/35    | (ت) PUT /movies/35  |
| (ث) GET /movie/35    | (ج) GET /movies/35  |

◊ به حز (ث) و (ج)، همه آنها از شیوه‌های قابل قبولی پیروی می‌کنند. اینکه از کلمهٔ مفرد یا جمع استفاده شود، یک موضوع سلیقه‌ای و قراردادی است، اما GET نباید برای خطوط سیری استفاده شود که گیش‌هایشان دارای اثرات و عوارض جانبی هستند. ■

### شاپیستگی ۲-۵-۳

بخش‌های حداقلی موردنیاز برای یک خط سیر HTTP مبتنی بر REST را مرور کنید.

### شاپیستگی ۳-۵-۳

نقش روش‌های درخواست (GET، POST و غیره) را در نحوهٔ استفاده از خطوط سیر توسط آپ‌های SaaS مرور کنید.

## ۶-۳ URIهای مبتنی بر REST، فراخوانی واسطه‌های برنامه‌نویسی و JSON

در بررسی چگونگی استفاده از مجموعه‌ای از سرویس‌هایی که تحت یک معماری سرویس‌گرا ارائه شده‌اند به عنوان یک بستر برنامه‌نویسی، به ردیفهای ۱ تا ۳ از شکل ۵-۳ پرداخته‌ایم. در ادامه، نحوه ارسال یا دریافت داده از این سرویس‌ها و برخی ملاحظات عملیاتی مانند احراز مجوز دسترسی<sup>۶۶</sup> (آیا کلاینت مجاز است این واسطه برنامه‌نویسی را روی این منبع فراخوانی کند؟) و نحوه مدیریت خطاهای توپیجی می‌دهیم.

در یک نگاه کلی، سه روش برای ارسال پارامترها از کلاینت به یک سرویس بر بستر HTTP وجود دارد: به طور مستقیم در خود URI، در متن درخواست (برای درخواست‌های POST یا PUT)، و به ندرت، به عنوان مقدار یکی از سرآیندهای HTTP. وقتی که تعداد پارامترها کم است و بهوژه وققی که پارامترها از نوع‌های ساده‌ای مانند رشته یا اعداد هستند، اغلب می‌توان آن‌ها را به صورت پارامترهای تعییشده در URI ارسال کرد، همانند آنچه در شکل ۲-۳ نشان داده شد:

`param1=value1&param2=value2&...&paramN=valueN`

این حالت برای درخواست‌های GET مرسوم است که معمولاً در آن‌ها داده‌هایی را بر اساس یک شناسه و شاید چندین پارامتر اختیاری درخواست می‌کنیم. به عنوان مثال، با رجوع به مستندات واسطه برنامه‌نویسی سرویس TMDb<sup>۶۷</sup> خواهید دید که خط سیر زیر، فیلمی را در JSTW وجود می‌کند که عنوان آن با رشتهٔ پرسمان «Batman Returns» مطابقت دارد:

`GET /search/movies?query=Batman+Returns`

وقتی داده‌های ارسالی پیچیده‌تر هستند، یا زمانی که عملیات واسطه برنامه‌نویسی شامل یک روش درخواست تغییردهندهٔ حالت مانند POST یا PUT است، آنگاه داده‌ها به عنوان بخشی از متن درخواست ارسال می‌شوند، همان‌طور که مرورگرها هنگام ارسال مقادیر واردشده در یک فرم عمل می‌کنند. (به یاد داشته باشید که درخواست‌های GET هیچ متن درخواستی ندارند). این داده‌ها چگونه به سرور ارائه می‌شوند؟ در حالی که گزینه‌های زیادی برای این کار وجود دارد، هیچ شکی نیست که جامعهٔ کاربران معماري سرویس‌گرا به سرعت ببروی JSON<sup>۶۸</sup> (جی‌سان) به عنوان قالب معمول تبدیل داده، توافق کرده‌اند. جی‌سان به این نام شناخته می‌شود چون نحوهٔ نگارش آن شبیه به (اما نه کاملاً) یک شیء در جاوا اسکریپت است-مجموعه‌ای از چفت‌های کلید-مقدار<sup>۶۹</sup> بدون ترتیبی مشخص، مشابه هش در زبان برنامه‌نویسی رویی، dict در پایتون یا HashMap در جاوا. در جی‌سان، هر کلید (یا «خانه»، همان‌طور که در فصل ۶ خواهیم آموخت) باید یک رشتهٔ مابین علامت

همانند خود جاوا اسکریپت، استاندارد JSON تحت نظر ECMA یا انجمن تولیدکنندگان رایانهٔ اروپا قرار دارد.

<sup>66</sup>Authorization

<sup>67</sup>The Movie Database

<sup>68</sup>JavaScript Object Notation

<sup>69</sup>Key-Value Pair

نقل قول دوتایی باشد ("string"). مقدار آن نیز می‌تواند یک نوع داده ساده (رشته، عدد، true، null، false) باشد که هر یک از عناصر آن می‌توانند هر یک از این انواع داده باشند، یا یک شیء دیگر باشد که خانه‌های آن داده‌هایی از همین انواع داده‌ای باشند. ویگاه رسمی چی‌سان<sup>۷۰</sup> چند مثال ساده ارائه می‌دهد که می‌توانید به آن‌ها رجوع کنید. به خاطر محبوبیت JSON به عنوان قالب داده پیش‌فرض برای معماری سرویس‌گر، تقریباً تمام زبان‌های برنامه‌نویسی مدرن شامل کتابخانه‌هایی برای تولید و تجزیه JSON هستند. نوشتمن فاصله‌های خالی<sup>۷۱</sup> (کاراکترهای فاصله)،<sup>۷۲</sup> جهش<sup>۷۳</sup> و خط نو<sup>۷۴</sup> در JSON اختیاری است و بیشتر سرویس‌ها متن چی‌سانی که برمی‌گرانند قادر فاصله‌های خالی است. ابزارهای خط فرمان یونیکس<sup>۷۵</sup> مانند json\_pp و افزونه‌های مرورگر JSONView مانند فاصله‌گذاری و تورفتگی‌ها<sup>۷۶</sup> را هنگام نمایش بازمی‌گردانند تا متن چی‌سان خواناتر شود. توجه داشته باشید که درخواست‌هایی که نیاز به ارسال داده‌های چی‌سان دارند، ممکن است همچنین اجازه دهنده (یا الزام کننده) که برخی از مقادیر پارامترها به صورت تعییشده در URI ارسال شوند؛ همواره با رجوع به مستندات مربوط به واسطه برنامه‌نویسی، جزئیات نحوه انجام این کار را بررسی کنید.

گاهی اوقات برای ارسال انواع سیار خاصی از پارامترها از سرآیندهای HTTP استفاده می‌شود. برای مثال، برخی از واسطه‌های برنامه‌نویسی از شما می‌خواهند که اگر پایه‌بار<sup>۷۷</sup> درخواست شما در واقع یک چی‌سان است، آنگاه یک سرآیند HTTP با مقدار Content-Type: application/json به درخواست خود اضافه کنید، درحالی‌که برخی دیگر نیازی به این کار ندارند. در نهایت، تقریباً همه واسطه‌های برنامه‌نویسی نیاز به احراز مجوز دسترسی دارند—کلاینت برای انجام هر فرآخوانی واسط برنامه‌نویسی باید ثابت کند که حق انجام آن کار را دارد. درحالی‌که طرح‌های احراز مجوز دسترسی متنوعی وجود دارند، رایج‌ترین روش گنجاندن یک کلید واسط برنامه‌نویسی معمولاً از متمایزکننده آن است) به همراه هر درخواست است. کلیدهای واسط برنامه‌نویسی معمولاً از قبیل به صورت دستی توسط کاربر درخواست می‌شوند و ممکن است رایگان یا پولی باشند (برای سرویس TMDb دریافت کلید رایگان است) و سرویس ممکن است محدودیت‌هایی را مانند تعداد فرآخوانی‌های مجاز در روز اعمال کند. بسته به واسط برنامه‌نویسی، این کلید ممکن است به عنوان یک آرگومان در URI (همانند TMDb)، به عنوان مقدار یک سرآیند HTTP (Authorization:) (یا هر دو روش ارسال شود).

با جمع‌بندی تمام این موارد، شکل ۷-۳ استفاده از ابزار curl را برای انجام دنبالهای از درخواست‌ها برای واسط برنامه‌نویسی مبتنی بر REST نشان می‌دهد که در آن از واسط برنامه‌نویسی مربوط به سرویس TMDb استفاده شده است. همگی این درخواست‌ها به جز آخرین درخواست از نوع GET هستند. توجه خاصی به این نکته داشته باشید که کلید واسط برنامه‌نویسی به عنوان یک پارامتر الزامی برای هر درخواست است، در URI گنجانده شده است. همچنین به قالب شیء نوشته شده در خط ۱۴ که نمایانگر امتیاز دلخواه شما برای یک فیلم است و به عنوان پایه‌بار یا متن درخواست ارسال می‌شود توجه داشته باشید و با رجوع به مستندات واسط برنامه‌نویسی می‌توانید صحبت آن را بررسی کنید.

اگر خطای رخ دهد چه؟ به خاطر بیاورید همانطور که در قسمت ۲-۳ توضیح داده شد، هر پاسخ HTTP با یک کد وضعیت سه‌ رقمی شروع می‌شود؛ این کدها توسط کنسرسیوم جهانی و<sup>۷۸</sup> فهرست‌بندی و نگهداری می‌شوند.<sup>۹</sup> سرویس‌ها از این کدهای وضعیت برای نشان دادن انواع مختلف خطاهای استفاده می‌کنند:

• کدهای وضعیت سری ۲۰۰ (2xx) نشان‌دهنده موفقیت هستند. به عنوان مثال، کد



در فصل ۱۲ به این خواهیم پرداخت که استفاده از HTTPS چرا و چگونه امکان ارسال این اطلاعاتی که عملًا حکم گذروزه را دارد به عنوان بخشی از درخواست وب را فراهم می‌کند.

**گستره‌ها** به فرآخوانی‌دهنده اجاهه می‌دهند در زمانی که نمودافزار امنیتی را درخواست می‌کنند، مشخص کند که چه نوع عملیاتی را می‌خواهد با استفاده از واسط برنامه‌نویسی انجام دهد. واسط برنامه‌نویسی گیت‌هاب<sup>۵</sup> اقسام مختلفی از گستره‌ها را پشتیبانی می‌کند؛ اما واسطه‌های برنامه‌نویسی ساده‌تر مانند TMDb معمولاً از گستره‌ها پشتیبانی نمی‌کنند.

<sup>70</sup>Whitespace (Character)

<sup>71</sup>Space (Character)

<sup>72</sup>Tab (Character)

<sup>73</sup>Newline (Character)

<sup>74</sup>Unix (OS)

<sup>75</sup>Indentation (Programming)

<sup>76</sup>Payload (Networking)

<sup>77</sup>World Wide Web Consortium (W3C)

<https://gist.github.com/c428409170320d32ba60934e1d29b190>

```

1 # set endpoint for TMDb API
2 export BASE=https://api.themoviedb.org/3
3 # set our API key for use in other calls
4 export KEY="my API key here"
5 # Search for a movie by keywords
6 curl "$BASE/search/movie?api_key=$KEY&query=Batman+Returns"
7 # For better legibility, pipe the output to json_pp:
8 curl "$BASE/search/movie?api_key=$KEY&query=Batman+Returns" | json_pp
9 # Start a new guest session
10 curl "$BASE/authentication/guest_session/new?api_key=$KEY" | json_pp
11 # capture the guest session ID from Curl's output:
12 export SESSION=e91f07cca8166b7b1e707d8a826e8a38
13 # Create a file containing the JSON object for rating a movie:
14 echo '{ "value": 6.5 }' > myrating.json
15 # Use Curl to POST a movie rating request using the file's contents:
16 curl -X POST -H "Content-Type: application/json" -d @myrating.json \
17 "$BASE/movie/364/rating?api_key=$KEY&guest_session_id=$SESSION"

```

شکل ۷-۳: دستورات این اسکریپت شل را خط به خط امتحان کنید. توجه داشته باشید که مقدار KEY را با کلید واسط برنامه نویسی خود در سرویس **TMDb** جایگزین کنید. پس از اجرای خط ۸، باید به صورت دستی شناسه صحیح فیلم را از محتوای پاسخ دریافتی که به صورت چیزیان است، پیدا کنید (عدد ۳۶۴) و بعداً در خط ۱۷ از آن استفاده کنید. پس از اجرای خط ۱۰ نیز شناسه نشست صحیح کاربر مهمان را استخراج کرده تا در خط ۱۲ از آن استفاده کنید. اگر شما ابزار json\_pp را نصب ندارید، می‌توانید آن را از دستورات حذف کنید.

۲۰۰ (موفقیت‌آمیز یا "OK") به طور معمول برای نشان دادن موفقیت یک درخواست GET استفاده می‌شود، در حالی که کد ۲۰۱ ("Created" یا ساخته شد) برای نشان دادن موفقیت یک درخواست POST که یک منبع جدید ایجاد می‌کند، رایج‌تر است.

- کدهای وضعیت سری ۳xx (3xx) نشان‌دهنده این هستند که کلاینت باید اقدامات بیشتری برای تکمیل درخواست انجام دهد یا به عبارت دیگر، تغییرمسیر<sup>۷۸</sup> می‌باشد صورت گیرد. ممکن است منبع درخواست‌شده به یک URI دیگر منتقل شده باشد که در متن پاسخ مشخص می‌شود.

- کدهای وضعیت سری ۴xx (4xx) نشان‌دهنده این هستند که سرویس (به سبب وجود مشکلی در درخواست) با یک خطأ در حین پردازش درخواست موواجه شده است. کد وضعیت ۴۰۰ به این معناست که درخواست نامعتبر و از نظر نحوی نادرست بوده است. اما کدهای دیگری هم در این سری هستند که با وجود معتبر بودن درخواست، نشان‌دهنده مشکلات دیگری هستند، از جمله کد وضعیت ۴۰۱ ("Unauthorized" یا غیرمجاز) که بیانگر وجود خطأ در احرار مجوز دسترسی است، کد وضعیت ۴۰۲ ("Payment Required") یا پرداخت لازم است) و موارد دیگر.

- کدهای وضعیت سری ۵xx (5xx) نشان‌دهنده این هستند که مشکلی در زیرساخت خود سرویس وجود دارد—یک خطأ که منجر به این شده که سرور حتی نتواند این فراخوانی را به پایان برساند، مثلاً سرور با یک خطأ داخلی بسیار شدید موواجه می‌شود که حتی نمی‌تواند توضیح دهد چه مشکلی پیش آمده است.

در صورت بروز خطأ (هر کد وضعیتی به جز سری ۲۰۰)، معمولاً متن پاسخ حاوی یک پیامی است که توضیح می‌دهد چه مشکلی رخ داده است. بسته به واسط برنامه نویسی، متن پاسخ ممکن است تنها شامل این رشته باشد یا معمولاً شامل یک شیء چیزیان با یک جفت کلید-مقدار که مقدار آن یک رشته است و کلید آن error یا message یا چیزی مشابه آن است.

### چکیده

- یکی از راه‌های اعمال اصول طراحی مبتنی بر REST بر خطوط سیر HTTP، استفاده از روش درخواست (GET، POST و غیره) و قسمت مسیر URL برای بیان منبع و عملیاتی که باید انجام شود است. در درخواست‌های GET، آرگومان‌های اختیاری می‌توانند مستقیماً در URI تعبیه شوند (به‌شکل `?param1=value1&...&paramN=valueN`). اما در درخواست‌های POST یا PUT، که معمولاً برای ارسال فرم استفاده می‌شوند، مقادیر تعبیه شده برای قسمت‌های فرم و آرگومان‌های اختیاری اضافی می‌توانند به عنوان بخشی از متن درخواست ارسال شوند.
- درخواست‌های ارسالی به یک سرویس ممکن است مستلزم ارائه نوعی مدارک هویتی باشند تا ثابت کنند که کلاینت مجاز به استفاده از این سرویس است. طرح‌های متعددی برای این منظور وجود دارد، اما یکی از ساده‌ترین آن‌ها احراز هویت پایه HTTP است که در آن یک نام کاربری و گذرواژه (که احتمالاً پیش‌تر توسط مرورگر از کاربر گفته شده است) در سرآیندهای HTTP تعبیه می‌شوند. استفاده از پروتکل امن انتقال ابرمن (HTTPS) که در فصل ۱۲ توضیح داده می‌شود، تضمین می‌کند که شنودکنندگان نتوانند این داده‌های حساس را بخوانند.
- چی‌سان (JSON) که بر اساس قواعد زبان برنامه‌نویسی جاوا اسکریپت طراحی شده است، به محبوب‌ترین قالب داده برای ارسال و دریافت داده در سرویس‌های تشکیل‌دهنده SaaS تبدیل شده است.
- هیچ‌کس تصویب نکرده است که REST ده‌ها طرح پیشنهادی دیگر را در رقابت شکست دهد و به روش ارجح برای طراحی سرویس‌ها تبدیل شود. بلکه REST به این دلیل به‌طور گسترده‌پذیرفته شد که فهم و پیاده‌سازی آن ساده بود، به‌خوبی با پروتکل‌های زیربنایی وب تطبیق داشت و با محدودیت‌های مربوط به مالکیت معنوی مواجه نبود.

### ■ بیشتر بدانیم: چرا REST پیروز شد؟

جادهٔ منتهی به ریزسرویس‌های امروزی مملو است از نام‌های اختصاری قراردادها و استانداردهایی که برای ایجاد سازگاری و همکاری پیشنهاد شدند، اما مانند زباله‌های پراکنده در مسیر، هرگز نتوانستند جایگاهی برای خود پیدا کنند و هیچ‌گاه به‌طور کامل مورد پذیرش قرار نگرفته‌اند: **WSDL**, **SOAP**, **CORBA**, **Jini**, **DCOM**, **XML-RPC**, **UDDI** تنها چند نمونه از آن‌ها هستند. از آنجا که هیچ‌نهاد واحدی بر اینترنت کنترل ندارد و نمی‌تواند «برندهای را انتخاب کند»، معمولاً برندۀای از طریق اجماع نسبی و به دلایل عملی پذیراد می‌شون: اینکه درک و استفاده از آن برای توسعه‌دهندگان آسان باشد (به‌ویژه برای راه‌اندازی سریع موارد رایج و ساده)، اینکه با زیرساخت فناوری موجود (در این مورد پروتکل‌ها و استانداردهای وب، به‌ویژه HTTP) به‌خوبی سازگار باشد و اینکه توسعه آن مشمول هزینه‌ها یا مجوزهای محدودکننده و گران نباشد. REST این معیارها را برآورده می‌کند—به اندازه‌ای ساده است که بتوان آن را در این یک قسمت از این کتاب توضیح داد—اما تطابق و هم‌خوانی مناسب آن با HTTP از طریق نوعی «تقلیب» به دست آمده است: REST با رویکردی بازنگرانه و با درس گرفتن از گذشته، یک جور کدگذاری از رویه‌ها و راهکارهایی است که در زمان رشد و بلوغ وب مؤثر شناخته شدند، از جمله تأکید بر طراحی بدون حالت، راهکاری برای ساخت URL‌هایی که به‌خوبی با ذخیره‌سازی موقت وب سازگار است (فصل ۱۲)، عدم اتكا به مفهوم ضمنی (نشست) و موارد دیگر. بیشتر پروتکل‌های رقیب یکی با بیشتر از این درس‌ها را نادیده گرفتند و بنابراین نتوانستند به «نقطه طلایی» دست یابند.

### خودآزمایی ۱-۶-۳

شما با استفاده از واسط برنامه‌نویسی سرویس TMDb درخواستی را برای این سرویس

ارسال می‌کنید و پس از آن پاسخی با کد وضعیت ۴۰۰ دریافت می‌کنید. با فرض اینکه سرویس *TMDB* از معانی رسمی ارائه شده توسط کنسرسیوم جهانی وب برای کدهای وضعیت پیروی می‌کند، کدام یک از موارد زیر ممکن است دلیل بروز خطا باشد: (الف) درخواست شما نامعتبر و از نظر نحوی نادرست بوده است، پس رسیدگی به آن ممکن نبوده است؛ (ب) شما فراموش کرده‌اید کلید واسط برنامه‌نویسی خود را به همراه درخواست ارسال کنید؛ (پ) درخواست و کلید واسط برنامه‌نویسی شما نادرست نبوده‌اند و هر دو معتبر بوده‌اند، اما شما در حال تلاش برای انجام عملیاتی هستید که مجاز به انجام آن نیستید.

◊ احتمالاً مورد (الف) اتفاق افتاده است. در دو حالت دیگر، کد وضعیت ۴۰۱ ("Unauthorized") یا غیرمجاز) احتمالاً بازگردانده می‌شود. ■

### شاپیستگی ۲-۶-۳

`build.api.request`

با توجه به مشخصات یا مستندات یک واسط برنامه‌نویسی، خط سیری بسازید که درخواست خاصی را در برابر آن واسط برنامه‌نویسی انجام دهد.

## ۷-۳ پُنک: ساخت و بهکاراندازی یک آپ *SaaS* ساده



### پُنک ۷-۳: ساخت و بهکاراندازی یک آپ *SaaS* ساده

<https://github.com/saasbook/hw-sinatra-saas-wordguesser>

یک آپ *SaaS* بسازید و بهکاراندازی کنید که یک بازی حدس کلمه را ارائه می‌دهد که در آن کاربران با حدس حروف، کلمه مورد نظر را پیدا می‌کنند. برای ساخت آن از زبان برنامه‌نویسی رویی و سیناترا، چارچوب ساده ساخت آپ‌های *SaaS* استفاده کنید. در طراحی خود به این نکات توجه کنید: چگونه گنیش‌های بازی به خطوط سیر *HTTP* نگاشت می‌شوند؛ چگونه وضعیت فعلی بازی نمایش داده می‌شود؛ چگونه از کوکی‌ها برای مدیریت وضعیت بازی استفاده می‌شود؛ و چگونه تقلب را شناسایی و جلوگیری می‌کنید (به این معنی که نمی‌توانید به هیچ‌یک از کلاینت‌های *HTTP* اعتماد کنید).

### شاپیستگی ۱-۷-۳

چندین قابلیت ساده به بازی حدس کلمه اضافه کنید، مثلًاً اینکه به بازیکن اجازه دهید که قبل از حدس زدن تمامی حروف کلمه، اگر بخواهد بتواند به صورت فوری کلمه را حدس بزند. ■

## ۸-۳ باورهای نادرست و خطرهای پنهان

### باور نادرست: تقسیم یک سرویس بزرگ «تکسنگی» به چندین ریزسرویس باعث کاهش پیچیدگی می‌شود.



پیچیدگی منطق کسب‌وکار<sup>۷۹</sup> یک سامانه با تقسیم آن یا طراحی آن بر اساس معماری سرویس‌گرا از بین نمی‌رود؛ بلکه این پیچیدگی تنها در میان ریزسرویس‌ها توزیع می‌شود و به‌ویژه در نحوه مدیریت برهم‌کنش‌های میان این ریزسرویس‌ها نمود پیدا می‌کند. دو الگوی اصلی ساختاری برای

<sup>79</sup>Business/Application Logic

هماهنگی میان ریزسرویس‌ها عبارت‌اند از: سازآرایی (که در آن لایه ترکیب دارای پیچیدگی بیشتری است و بخش عمده‌ای از منطق کسب‌وکار را در خود جای می‌دهد) و کارآرایی (که در آن سرویس‌ها بدون وجود یک لایه ترکیب جداگانه با یکدیگر برهمنکنش دارند).<sup>۸۰</sup> در نهایت، تقسیم یک سرویس بزرگ به ریزسرویس‌ها نیازمند تفکر دقیق درباره مرز بین پودمان‌ها است، همان‌گونه که در طراحی یک سرویس بزرگ نیز باید انجام می‌شد.

**باور نادرست: من با انتشار API سرویسم باعث می‌شوم که سرویسم مبتنی بر REST**

**شود (یا آن را به یک ریزسرویس تبدیل می‌کنم، یا آن را سازگار به اصول معماری سرویس‌گرا می‌کنم و غیره).**



انتشار یک واسط برنامه‌نویسی صرفاً به این معناست که فراخوانی‌های خارجی محاز هستند؛ میزان قابل فهم بودن و قابل استفاده بودن واسط برنامه‌نویسی تعیین می‌کند که آیا واسط برنامه‌نویسی یا سرویس ارائه شده به‌طور گسترده مورد استفاده قرار می‌گیرد یا خیر، زیرا واسط برنامه‌نویسی در واقع حکم مذاکرات مرزی برای کاری که سرویس انجام می‌دهد (خروجی مورد انتظار) و نحوه دسترسی فراخوانی‌دهنده به آن (ورویدی مورد انتظار) را دارد. REST تنها روش خوب برای طراحی واسط برنامه‌نویسی نیست، اما روش‌های زیادی برای طراحی بد یک واسط برنامه‌نویسی وجود دارد و پیروی دقیق از اصول REST احتمال انتخاب یکی از آن روش‌های بد را کاهش می‌دهد.

**باور نادرست: من یک API برای سرویسم منتشر نکرده‌ام، بنابراین کلاینت‌ها نمی‌توانند به آن دسترسی داشته باشند.**



هر وبگاهی که به‌طور عمومی قابل دسترسی باشد، به صورت ضمنی از یک واسط برنامه‌نویسی مبتنی بر HTTP برخوردار است، زیرا می‌توان با ساختن URL‌های خاص، با وبگاه ارتباط برقرار کرد و از آن اطلاعات را دریافت کرد. البته چنین واسط برنامه‌نویسی «تصادفی» به‌ندرت از اصول طراحی خوب پیروی می‌کند؛ یک کلاینت که می‌خواهد از آن استفاده کند ممکن است مجبور شود (به عنوان مثال) صفحه HTML بازگشتی را تجزیه کند تا اطلاعات مورد نیاز خود را استخراج کند، فرایندی که گاهی تراشیدن<sup>۸۱</sup> وب نامیده می‌شود. با این حال، اگر سرویس یا آپ شما به صورت عمومی در دسترس است، یک واسط برنامه‌نویسی دارد، چه بخواهید و چه نخواهید. اگر قصد دارید آپ شما به عنوان یک سرویس یا ریزسرویس توسعه سایرین به صورت برنامه‌نویسی شده استفاده شود، باید مطابق با رهنمودهای این فصل یک واسط برنامه‌نویسی طراحی و منتشر کنید.

**خطر پنهان: تفکر بر اساس URLs، گُنیش‌های کاربر و نمایه‌ها به جای تفکر بر اساس**

**منابع.**



یک دنباله از مراحل در یک وبگاه فرضی تجارت الکترونیک را در نظر بگیرید: در مرحله ۱، کاربر به صفحه یک کالا مراجعه می‌کند؛ در مرحله ۲، یک کالا را به سبد خرد اضافه می‌کند؛ شاید مراحل ۱ و ۲ را برای اضافه کردن چندین کالا تکرار کند؛ در مرحله ۳، هزینه کالا(ها) را پرداخت می‌کند. اگر منطق کسب‌وکار چنین آپ را از دیدگاه متمنکر بصفحات وب و توالی آن‌ها طراحی کنید، ممکن است وسوسه شوید که تنها مراحلی که کاربر در آن است را ردیابی کنید (متلاً به عنوان بخشی از نشست یا با قرار دادن شماره مرحله به عنوان یک پارامتر در URL<sup>۸۲</sup>) و منطق برنامه را طوری طراحی کنید که با توجه به هر مرحله، به گُنیش داخلی مناسب آن مرحله هدایت شود. اما چنین رویکردی شما را مجبور به تفکر درباره پرسش‌های مهمی از این دست نمی‌کند؛ اگر کاربر وبگاه را ترک کند و

<sup>۸۰</sup> ما دو واژه «سازآرایی» و «کارآرایی» را به ترتیب معادل Choreography و Orchestration در ترجمه این کتاب نظر گرفته‌ایم. در موسیقی، «Orchestration» به تنظیم و هماهنگ‌سازی سازها و قطعات مختلف موسیقی توسط یک رهبر ارکستر اشاره دارد. این مفهوم در زمینه معماری سرویس‌گرا بر هماهنگی متمنکر و مدیریت منطقی ریزسرویس‌ها توسط یک لایه ترکیب مرکزی دلالت می‌کند. از سوی دیگر، «Choreography» در رقص به برهم‌کنش‌های هماهنگ و بدو نیاز به مدیریت مرکزی میان اعضای گروه اشاره دارد. در معماری سرویس‌گرا این مفهوم به برهم‌کنش‌های غیرمتمنکر و خودگردان میان ریزسرویس‌ها اشاره دارد. سعی شده که هر دو واژه ارتباط نزدیک با ریشه‌های هنری این اصطلاحات در موسیقی و رقص داشت باشند و در عین حال مفهوم فنی و نحوه همکاری و برهم‌کنش‌ها در معماری سرویس‌گرا منعکس کنند.

<sup>81</sup> Scraping [Web/Data]

<sup>82</sup> Uniform Resource Locator

بعداً بازگردد، چگونه می‌توانیم اطمینان حاصل کنیم که محتوای سبد خرد او تغییر نکرده است؟ اگر کاربر سفارش را بدون پرداخت رها کند، چه زمانی تصمیم می‌گیریم که آن را حذف کنیم؟ اگر مرحله پرداخت شکست بخورد، آیا می‌توانیم به راحتی کاربر را به تکرار فقط همان مرحله هدایت کنیم بدون اینکه دوباره از تمام مراحل قبلی عبور کند؟ در مقابل، یک رویکرد طراحی واسط برنامه‌نویسی مبتنی بر REST با این پرسش آغاز می‌شود: یک سفارش چه نوع منبعی است و چه عملیاتی روی آن قابل انجام است؟ یک کالا چه نوع منبعی است و چه عملیاتی روی آن قابل انجام است؟ چه چیزی ممکن است در هر نوع عملیات اشتباه پیش برود؟ هر نوع منبع چگونه روی سرور ذخیره می‌شود و تحت چه شرایطی می‌توان آن را حذف کرد؟ کلاینت چگونه می‌تواند به منبع خاصی که قبلاً ایجاد شده اشاره کند، حتی اگر کاربر برای مدت طولانی از رایانه خود دور بوده باشد؟ پاسخ‌های سنجیده به چنین پرسش‌هایی منجر به طراحی سرویس‌های تزویجی می‌شود که هم به عنوان یک سرویس یا ریزسرویس مستقل و هم به عنوان سرویس پشتیبان یک تجربه مبتنی بر مرورگر مناسب است.

### **خطر پنهان: طراحی ضعیف API بهدلیل درک نادرست از حوزه یا موارد استفاده کاربران.**

ما مشاهده کردیم که فراخوانی واسط برنامه‌نویسی سرویس TMDb برای «دربیافت نقدهای مرتبط با این فیلم» در واقع محتوای نقدها را بازمی‌گرداند، نه فقط شناسه‌های آن‌ها را. این طراحی زمانی منطقی است که بیشتر درخواست‌های این نوع احتمالاً به بررسی محتوای اکثر نقدها منجر شود. اما اگر مورد استفاده رایج‌تر (برای مثال) نمایش جزئیات نقدهایی با ویژگی‌های خاصی مانند امتیاز عددی باشد، یک API کارآمدتر می‌تواند امکان مشخص کردن این گزینه‌ها را برای محدود کردن نتایج فراهم کند.

به طور کلی، درک ناکافی از نحوه استفاده کاربران از واسط برنامه‌نویسی شما ممکن است به طراحی ناکارآمد منابع در کد داخلی برنامه منجر شود؛ یعنی منابع مورد نیاز به درستی شناسایی نشده و برهمنکش بین آن‌ها به‌شکل مناسب برای پاسخگویی به نیازهای کاربران مدل‌سازی نشده است. اما همان‌طور که خواهیم دید، خبر خوب این است که طراحی و نمایش داخلی منابع تا زمانی که جزئیات آن به API نشست نکرده باشد، قابل تغییر است.

### **۹-۳ نکات پایانی: سیر تحول از واسط دروازه مشترک تا معماری سرویس‌گرا**

از آن جایی که در روزهای ابتدایی وب، از آن برای ارائه مجموعه‌ای از فایل‌های ایستای ذخیره‌شده در یک سامانهٔ فایل‌بندی<sup>۸۳</sup> استفاده می‌شد، سرووهای اولیه‌ای که خدمات وب و HTTP ارائه می‌دادند، مانند سرور وب آپاچی<sup>۸۴</sup>، می‌توانستند به گونه‌ای پیکربندی شوند که یک زیرساخت از سامانهٔ فایل‌بندی را به عنوان یک درخت قابل مرور از فایل‌ها و دایرکتوری‌ها ارائه دهند. به همین دلیل، URI اولیه معمولاً ساختار سلسله‌مراتبی سامانهٔ فایل‌بندی را تقلید می‌کردند. به عنوان مثال، یک URI مانند <http://www.cs.berkeley.edu/reports/1997/daedalus.ps> به احتمال زیاد به فایل واقعی اشاره داشت که در زیرساختهایی به نام [reports/1997/daedalus.ps](http://reports/1997/daedalus.ps) در رایانه‌ای باشد، قابل تغییر است.

در سال ۱۹۹۳، **واسط دروازه مشترک** یا پروتکل CGI<sup>۸۵</sup> نخستین نشانه‌ها برای ظهور SaaS بود و به‌نوعی نقطه‌ای ططفی در تاریخ وب محسوب می‌شد. یک سرور وب مجهز به CGI می‌توانست برخی از URI‌ها را نه به عنوان نام فایل که در آن سرور ذخیره شده است، بلکه به عنوان یک دستور برای اجرای یک برنامه و ارسال خروجی آن به کلاینت تفسیر کند. در حالی‌که هیچ قراردادی برای نحوه ساخت چنین URI‌هایی پیشنهاد نشده بود، یک روش رایج این بود که تمام این «برنامه‌های CGI» را در یک زیرساختهای واحد که اغلب cgi-bin نامیده می‌شد، قرار داد و از ترکیبی از مؤلفه‌های قسمت مسیر URI و پارامترهای موجود در رشتهٔ پر اسمان برای «ارسال آرگومان‌ها» به برنامهٔ موردنظر برای اجرا

<sup>83</sup>File System

<sup>84</sup>Apache Httpd (SW)

<sup>85</sup>Common Gateway Interface (CGI)

استفاده شود. برنامه CGI باید یک پاسخ HTTP کامل و درست، شامل سرآیندهای HTTP متناسب پاسخ و یک پایه‌بار که مثلاً می‌توانست یک صفحه HTML باشد را تولید می‌کرد. با گسترش استفاده از SaaS به عنوان یک روش رایج برای پیاده‌سازی و به‌کاراندای ویگاه‌ها، چارچوب‌هایی تحت عنوان **سورهای نرم‌افزارهای کاربردی**<sup>۸۶</sup> آغاز به ظهور کردند که به‌طور خودکار بخشی از این «کار گل»<sup>۸۷</sup> یا زیرساخت‌های فنی مانند ساختن پاسخ مناسب HTTP یا رسیدگی و مدیریت خطاهای رایج HTTP را انجام می‌دادند و به برنامه‌نویس اجزه می‌دادند تنها بر روی محتوا تمرکز کنند.

رشد سریع محبوبیت مدل «ریزسرویس‌های با واسطه‌های برنامه‌نویسی مبتنی بر REST» باعث شد تا توجه و تمرکز مجددی به طراحی واسطه برنامه‌نویسی و ابزارهایی برای پشتیبانی از آن شده است. مقاله‌جاشوا بِلَک<sup>۸۸</sup> با عنوان چگونه یک واسطه برنامه‌نویسی خوب طراحی کنیم و چرا این اهمیت دارد (پلاک ۶۰۰۲)<sup>۸۹</sup> و سخنرانی مربوط به آن که در گوگل ارائه شد، دیدگاه جامع و خوبی از نحوه تفکر در مورد طراحی واسطه برنامه‌نویسی ارائه می‌دهد. او تصمیم‌های مربوط به طراحی واسطه برنامه‌نویسی را با تصمیم‌های طراحی زبان‌های برنامه‌نویسی که دهه‌های است مورد بحث قرار گرفته‌اند مقایسه می‌کند و تعریفی عملی و مختصر از API ارائه می‌دهد: «روش‌های عملیاتی که مؤلفه‌های یک سامانه از طریق آن با یکدیگر تعامل می‌کنند.» (سخنرانی دیگری<sup>۹۰</sup> از همین نویسنده، تاریخچه‌ای طنزآمیز و دیدگاهی منتقدانه از واسطه‌های برنامه‌نویسی از آغاز ۱۹۵۰ میلادی ارائه می‌دهد.) علاوه بر این، از آنجا که API به نوعی یک «قرارداد» نمایان مابین سرویس‌دهنده و فرآخوانی‌دهنده‌های سرویس است، مستندسازی رسمی API اهمیت بیشتری یافته است. همچنین اطمینان از اینکه همزمان با تکامل سرویس، مستندات واسطه برنامه‌نویسی به روز بماند نیز اهمیت بالایی دارد. ابزارهای OpenAPI<sup>۹۱</sup> (که قبلًا به نام سواگر<sup>۸۸</sup> شناخته می‌شد) شامل یک ویرایشگر API برای طراحی واسطه‌های برنامه‌نویسی متنطبق با مشخصات API، یک مولد خودکار که برای تولید خُرده‌کدهای سمت سرور و کلاینت برای استفاده از API و ابزارهایی برای استخراج و انتشار خودکار مستندات از توضیحات OpenAPI بهصورت «زند»<sup>۹۲</sup> هستند.

یک جایگزین نسبتاً جدید برای واسطه‌های برنامه‌نویسی مبتنی بر REST (که کاملاً بر پایه رُویه‌ها هستند)، گرفکیوال<sup>۹۰</sup> است که به جای تمرکز فرآخوانی رُویه‌ها، بر توصیف داده‌ساختارها متمرکز است. در یک واسطه برنامه‌نویسی مبتنی بر REST، سرور تصمیم می‌گیرد که چه عملیات‌هایی را ارائه دهد و چه داده‌ساختارهایی برای فرآخوانی آن‌ها مورد نیاز است. در مقابل، یک کلاینت گرفکیوال داده‌ساختارهای مورد نیاز خود را تعریف می‌کند و همان داده‌ساختارها از سمت سرور بازگردانده می‌شوند. اگرچه غنایی و در عین حال پیچیدگی گرفکیوال ممکن است برای واسطه‌های برنامه‌نویسی ساده ارزش نداشته باشد، اما برای سرویس‌های اصطلاحاً داده‌بر<sup>۹۳</sup>، یک جایگزین جالب و رو به رشد نسبت به REST محسوب می‌شود.

در نهایت شایان ذکر است که این نوع واسطه‌های برنامه‌نویسی تنها جدیدترین تجلی یک عامل کلیدی در موقوفیت وب هستند: جداسازی اجزایی که تغییر می‌کنند از آن‌هایی که ثابت می‌مانند. مجموعه پروتکل اینترنت (TCP/IP)، HTML و HTTP ممکن بازنگری عمده را پشت سر گذاشته‌اند، اما همگی راههایی برای تشخیص نسخه مورد استفاده دارند، به‌طوری که کلاینت می‌تواند بفهمد آیا با یک سرور قدیمی‌تر در حال ارتباط است یا خیر و رفتار خود را بر این اساس تنظیم کند. امروزه واسطه‌های برنامه‌نویسی امکان جداسازی واسطه از پیاده‌سازی را در سطح سرویس‌های کامل فراهم کرده‌اند. اگرچه کار با نسخه‌های مختلف پروتکل‌ها و زبان‌ها بار اضافی بر مروگرها و سایر کلاینت‌ها تحمیل می‌کند، اما به نتیجه‌ای شگفت‌انگیز منجر شده است: یک صفحه وب که در سال ۲۰۱۹ میلادی ایجاد شده و از زبانی مبتنی بر فناوری دهه ۱۹۶۰ استفاده می‌کند، می‌تواند با استفاده از پروتکل‌های شبکه توسعه‌یافته در سال ۱۹۶۹ بازیابی شود و توسط مروگری که برای اولین بار در سال ۱۹۹۲ ایجاد شده نمایش داده شود. جداسازی اجزای متغیر از ثابت بخشی از مسیر ساخت

**لفظ bin** در حوزه رایانش، تقریباً همیشه معادل کوتاه‌نوشت فایل اجراذیز دودوبی<sup>۹۴</sup> یا Binary است. این در حالیست که برنامه‌های CGI لزوماً در ادامه دیگر فایلهای اجراذیز دودوبی نبودند و بهجای آن‌ها از اسکریپتهای شل که به زبان‌های مانند بیل یا پایتون نوشته شده بودند استفاده می‌شد.

<sup>86</sup>Application Server

<sup>87</sup>Joshua Bloch

<sup>88</sup>Swagger

<sup>89</sup>Editor (Programming)

<sup>90</sup>GraphQL

<sup>91</sup>Data-Intensive

**تیم پرنز-لی**، دانشمند علوم

رايانه در سین<sup>۵</sup>، توسعه HTTP و HTML را در سال ۱۹۹۰ رهبری کرد. تمام استانداردهای باز وب، از جمله موارد ذکر شده در اینجا، اکنون تحت نظرارت کنسرسیوم جهانی وب (W3C)<sup>۶</sup> که یک نهاد غیرانتفاعی و بی طرف است، قرار دارند.

نرم افزارهای بادوام است.

Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Trans. Comput. Syst.*, 2(1):39–59, February 1984. ISSN 0734-2071. doi:10.1145/2080.357392. URL <http://doi.acm.org/10.1145/2080.357392>.

Joshua Bloch. How to design a good api and why it matters. in *Proc. 21st ACM SIGPLAN Conference (OOPSLA)*, pages 506–507, Portland, Oregon, 2006. URL <http://portal.acm.org/citation.cfm?id=1176617.1176622>.

Raffi Krikorian. Personal communication, 2013.

Bertrand Meyer. *Object-Oriented Software Construction*. Prentice-Hall, 1997.

Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, and Mike Amundsen. *Microservice Architecture*. O'Reilly Media, Sebastopol, CA, 2016.

## پیوند ها

<http://www.catb.org/~esr/jargon/html/M/magic-cookie.html><sup>۷</sup>  
<https://gist.github.com/chitchcock/1281611><sup>۸</sup>  
<https://developers.themoviedb.org><sup>۹</sup>  
<https://json.org/example.html><sup>۱۰</sup>  
<https://docs.github.com/en/developers/apps/scopes-for-oauth-apps><sup>۱۱</sup>  
<https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html><sup>۱۲</sup>  
<https://www.youtube.com/watch?v=heh40eB9A-c><sup>۱۳</sup>  
<https://www.youtube.com/watch?v=ege-kub1qtk><sup>۱۴</sup>  
<https://swagger.io/tools><sup>۱۵</sup>  
<http://info.cern.ch><sup>۱۶</sup>  
<http://w3.org><sup>۱۷</sup>

# ریلز، یک چارچوب مدل-نمای کنترل‌گر

این فصل در ویرایش‌های بعدی از این کتاب گنجانده خواهد شد. شما می‌توانید برای کسب اطلاع از بهروزرسانی‌های آتی به وبگاه کتاب<sup>۱</sup> مراجعه کنید.

## پیوندها

<https://saasbook.info/translation-fa><sup>۱</sup>



# تجزیدهای پیشرفته برنامهنویسی برای SaaS

این فصل در ویرایش‌های بعدی از این کتاب گنجانده خواهد شد. شما می‌توانید برای کسب اطلاع از بهروزرسانی‌های آتی به وبگاه کتاب<sup>۱</sup> مراجعه کنید.

## پیوندها

<https://saasbook.info/translation-fa><sup>۱</sup>



# مقدمه‌ای بر جاوا اسکریپت

این فصل در ویرایش‌های بعدی از این کتاب گنجانده خواهد شد. شما می‌توانید برای کسب اطلاع از بهروزرسانی‌های آتی به وبگاه کتاب<sup>۱</sup> مراجعه کنید.

## پیوندها

[https://saasbook.info/translation-fa<sup>۱</sup>](https://saasbook.info/translation-fa)

---

## بخش دوم

# توسعه نرم افزار با رویکردی چاپک

---



# توسعهٔ رفتارمحور و روایت‌های کاربری

این فصل در ویرایش‌های بعدی از این کتاب گنجانده خواهد شد. شما می‌توانید برای کسب اطلاع از بهروزرسانی‌های آتی به وبگاه کتاب<sup>۱</sup> مراجعه کنید.

## پیوندها

[https://saasbook.info/translation-fa<sup>۱</sup>](https://saasbook.info/translation-fa)



۸

## توسعه آزمون محور

این فصل در ویرایش‌های بعدی از این کتاب گنجانده خواهد شد. شما می‌توانید برای کسب اطلاع از بهروزرسانی‌های آتی به وبگاه کتاب<sup>۱</sup> مراجعه کنید.

پیوندها

<https://saasbook.info/translation-fa><sup>۱</sup>



# میراث‌کد، بازسازی، و روش‌های چابک

این فصل در ویرایش‌های بعدی از این کتاب گنجانده خواهد شد. شما می‌توانید برای کسب اطلاع از بهروزرسانی‌های آتی به وبگاه کتاب<sup>۱</sup> مراجعه کنید.

## پیوندها

[https://saasbook.info/translation-fa<sup>۱</sup>](https://saasbook.info/translation-fa)



# تیم‌های چابک

این فصل در ویرایش‌های بعدی از این کتاب گنجانده خواهد شد. شما می‌توانید برای کسب اطلاع از بهروزرسانی‌های آتی به وبگاه کتاب<sup>۱۰</sup> مراجعه کنید.

## پیوندها

<https://saasbook.info/translation-fa><sup>۱۰</sup>



# الگوهای طراحی برای SaaS

این فصل در ویرایش‌های بعدی از این کتاب گنجانده خواهد شد. شما می‌توانید برای کسب اطلاع از بهروزرسانی‌های آتی به وبگاه کتاب<sup>۱</sup> مراجعه کنید.

پیوندها

<https://saasbook.info/translation-fa><sup>۱</sup>



# توسعه/عملیات: به کاراندازی، کارابی، قابلیت اطمینان و امنیت کاربردی

این فصل در ویرایش‌های بعدی از این کتاب گنجانده خواهد شد. شما می‌توانید برای کسب اطلاع از به روزرسانی‌های آتی به وبگاه کتاب<sup>۱</sup> مراجعه کنید.

## پیوندها

<https://saasbook.info/translation-fa><sup>۱</sup>



## پس‌گفتار

این فصل در ویرایش‌های بعدی از این کتاب گنجانده خواهد شد. شما می‌توانید برای کسب اطلاع از بهروزرسانی‌های آتی به وبگاه کتاب<sup>۱۰</sup> مراجعه کنید.

### پیوندها

[https://saasbook.info/translation-fa<sup>۱۱</sup>](https://saasbook.info/translation-fa)



# واژه‌نامهٔ انگلیسی به فارسی

## Numbers

2001: A Space Odyssey (Movie) ..... فیلم ۲۰۰۱: ادیسهٔ فضایی

## A

Abstract Data Type .....	نوع دادهٔ انتزاعی .....
Abstraction (Computer Science) .....	تجزیه (علوم رایانه) .....
Acceptance Testing .....	آزمون پذیرش .....
Accessibility .....	دسترسی پذیری .....
Accessor Method .....	متند دسترسی .....
Active Server Pages (ASP.NET) .....	صفحه‌های سرور فعال .....
Action (Software Engineering) .....	گُش .....
Address Bar (Browser) .....	نوار آدرس .....
Affordable Care Act .....	لایحهٔ مراقبت مقررین به صرفه .....
Agile .....	چابک .....
Agile Alliance .....	اتحادیهٔ روش چابک .....
Agile Manifesto .....	منشور چابک .....
Authentication .....	احراز هویت .....
Authorization .....	احراز محوظ دسترسی .....
Automation .....	خودکارسازی .....
Availability .....	دسترسی پذیری .....
AJAX (Asynchronous JavaScript And XML) .....	ایجکس .....
Amazon Web Services (AWS) .....	سرویس‌های وب آمازون .....
Amazon.com .....	آمازون (شرکت حوزهٔ فناوری) .....
American Airlines .....	امري肯 ايрапلینز .....
Ancestor (OOP) .....	جد .....
Andy Hunt .....	آندي هانت .....
Android (OS) .....	اندرويد .....
Angular (Web Framework) .....	آنگولار .....
Animation .....	پويانامي .....
AOL .....	اى اوال .....

Apache Httpd (SW)	سرور وب آپاچی
Apache James (SW)	آپاچی جیمز
Apache Maven	آپاچی میون
App (Application Software/Program)	اپ
App Store	فروشگاه نرم افزار کاربردی
Apple	اپل
Apple Mail (SW)	اپل میل
Apple Safari (SW)	اپل سافاری
Application (Software/Program)	نرم افزار کاربردی
Application Programming Interface	واسطه برنامه نویسی
Application Server	سرور نرم افزارهای کاربردی
Aqueduct of Segovia	آبگذر سگوبیا
Argument (Programming)	آرگومان (برنامه نویسی)
Ariane 4 Rocket	موشک آریان ۴
Ariane 5 Rocket	موشک آریان ۵
Armando Fox	آرماندو فاکس
Array (Data Structure)	آرایه
Assembly Language	زبان اسembly
Assertion (Computer Programming)	بايسٽگي (برنامه نویسي)
Associative Array (Data Structure)	آرایه انجماني
Attribute (Computing)	خصوصيت

## B

Backbone.js (Web Framework)	بک بون جی اس
Backend	پسین
Backward Compatibility	هم سازی با قبل
Barbara Liskov	باربارا لیسکف
Basic Authentication (HTTP)	احراز هویت پایه
Beautiful Code	زیبایی
Behavior-Driven Development	توسعه رفتار محور
Bertrand Meyer	پرتراند مایر
Bug (Computer)	اشکال
Buzzword	واژه باب روز
Bundler (Ruby)	باندلر
Business Modeling	مدل سازی کسب و کار
Business/Application Logic	منطق کسب و کار
Byte (Computing)	بایت
Big Design Up Front	طرایح عظیم از ابتدا
Binary (Computing)	دودویی
Bit (Computing)	بیت
Bit Blit	بیت بلیت
Bitmap	بیت مپ
BitTorrent	بیت تورنت
Block (Programming)	بلوک

Body (of request/message/packet) .....	متن
Bookmark (Web) .....	نشانک
Boolean (Data Type) .....	بولی
Bootstrap (Front-end Framework) .....	بوتاسترپ
Bottleneck .....	گلوگاه
Breakpoint (Programming/Debugging) .....	نقطهٔ توقف
British Computer Society .....	انجمن رایانهٔ بریتانیا
Browser (Web) .....	مرورگر

## C

C (Programming Language) .....	سی
C# (Programming Language) .....	سی شارپ
C++ (Programming Language) .....	سی پلاس پلاس
Caching (Software) .....	ذخیره‌سازی موقت
Call (Function/Method) .....	فراخوانی
Callee (Function/Method) .....	فراخوانی‌گیرنده
Caller (Function/Method) .....	فراخوانی‌دهنده
Camel Case (Programming) .....	نگارش شتری
Capability Maturity Model .....	مدل بلوغ قابلیت
Carnegie Mellon University .....	دانشگاه کارنگی ملون
Cascading Style Sheets .....	شیوه‌نامهٔ آبشاری
CERN .....	سین
CGI Group (Information Technology Company) .....	شرکت فناوری گروه سی‌جی‌آی
Cucumber (Software tool) .....	کیوکامبر (ابزاری برای توسعهٔ رفتارمحور)
Custom (Customized) .....	سفارشی
Cyberduck (SW) .....	سایپرداک
Character (Computing) .....	کاراکتر (رایانش)
Chief Technology Officer .....	مدیر ارشد فناوری
Choreography (Microservices) .....	کارآرایی
Clarity .....	وضوح
Class (Computer Programming) .....	کلاس (برنامه‌نویسی)
Class Variable (OOP) .....	متغیر کلاس
Class Method (OOP) .....	چند کلاس
CLU (Programming Language) .....	سی‌ال‌یو
Client .....	کلاینت
Cloud Computing .....	رایانش ابری
Closure (Lisp Dialect) .....	کلوژر (گویشی از زبان برنامه‌نویسی لیسپ)
Closure (Programming) .....	بستار
COBOL (Programming Language) .....	کوبال (زبان برنامه‌نویسی)
Codebase .....	مجموعهٔ کد
Coding/Hands-On Integrated Projects (CHIPS) .....	پروژه‌ها و فعالیتهای عملی/کدنویسی
Codio .....	کدیو
Coverage (Testing) .....	بوشش
Collection (Abstract Data Type) .....	گردآورد

Comic Strip .....	داستان مصور.....
Command-line .....	خط فرمان.....
Command-Query Separation .....	جداسازی فرمان-درخواست.....
Common Gateway Interface (CGI) .....	واسطه دروازه مشترک.....
CompuServe .....	کامپیوسرо.....
Computer .....	رایانه.....
Computer Cluster .....	خوش.....
Computer Programming .....	برنامه‌نویسی رایانه‌ای.....
Computer Science .....	علوم رایانه.....
Computing .....	رایانش.....
Compile Time .....	زمان کامپایل.....
Compiler .....	کامپایل.....
Component (Computing) .....	مؤلفه.....
Composition (OOP) .....	ترکیب.....
Conciseness .....	اختصار.....
Confucius (Chinese Philosopher) .....	کنفوسیوس.....
Configuration .....	بیکربندی.....
Convention Over Configuration (Design Paradigm) .....	قراردادمحوری بهجای پیکربندی.....
Connection (Networking) .....	اتصال.....
Console .....	کنسول.....
Constant Value (Programming) .....	مقدار ثابت.....
Construction (RUP phase) .....	ساخت (فازی از آریوبی).....
Constructor (OOP) .....	سازنده.....
Container (Virtualization) .....	کانتینر.....
Content-Oriented Didactic (COD) .....	آموزشی محتوامحور.....
Control Flow (Programming) .....	کنترل جریان.....
Controller (MVC) .....	کنترل‌گر.....
Cookie (HTTP) .....	کوکی.....
Croutine (Programming) .....	همروال.....
Cray .....	کری.....
Create (CRUD Action) .....	ایجاد (گیش CRUD).....
Create, Read, Update, Delete, Index .....	ایجاد، خواندن، بهروزسازی، حذف و فهرست کردن.....
Credentials (Security) .....	(CRUDI).....
	مدارک هویتی.....

## D

Dave Thomas .....	دیو توماس.....
David Patterson .....	دیوید پترسون.....
Data Format .....	قالب داده.....
Data Hiding (Programming) .....	نهانسازی دادهها.....
Data Structure .....	داده‌ساختار.....
Database .....	پایگاه داده.....
Datacenter .....	مرکز داده.....
Data-Intensive .....	داده‌بر.....

Debugger .....	اسکال زدا .....
Debugging .....	اشکال زدایی .....
Declaration .....	اعلان .....
Develop (Software) .....	توسعه .....
Developer (Software) .....	توسعه دهنده .....
Delete (CRUD Action) .....	حذف (گُنیش (CRUD .....
DELETE (HTTP Method) .....	حذف (روش درخواست) .....
Dennis Ritchie .....	دنیس ریچی .....
Deployment (Software) .....	به کاراندازی .....
Deprecation (Software) .....	منسوخ شدن .....
Design Pattern .....	الگوی طراحی .....
Desktop .....	دسکتاپ .....
Duck Typing (Programming) .....	نوع دهی اردکی .....
Dynamic .....	پویا .....
Dynamic Typing (Type Checking) .....	نوع دهی پویا .....
Dilbert (Comic Strip) .....	دیلبرت .....
Directory (Computing) .....	دایرکتوری .....
Django .....	جنگو .....
Document (Computing) .....	سنند .....
Document Type Declaration (XML Instruction) .....	اعلان نوع سنند .....
Documentation (Act of Documenting) .....	مستندسازی .....
Documentation (Set of Documents) .....	مستندات .....
Docker .....	دایکر .....
Douglas Crockford .....	داگلاس کراکفرد .....
Download .....	بارگیری .....
Domain Name System (DNS) .....	سامانه نام دامنه .....
Donald Knuth .....	دانلد کنوث .....
Don't Repeat Yourself (DRY) .....	خودت را تکرار مکن .....
Dropdown Menu .....	منوی کشویی .....

## E

eBay .....	ای بی .....
E-commerce .....	تجارت الکترونیک .....
Economies of Scale .....	صرفه به مقیاس .....
edX .....	إدكس .....
Editor (Programming) .....	ویرایشگر .....
Eudora (Email Client) .....	یودارا .....
Eureka (SW) .....	یوریکا .....
European Computer Manufacturers Association .....	انجمان تولیدکنندگان رایانه اروپا .....
	(ECMA)
Event (Computing) .....	رویداد .....
Exception .....	استثناء .....
Executable .....	اجراپذیر .....
Extensible Markup Language .....	زبان نشانه‌گذاری گسترش‌پذیر .....

Extention (Software) .....	افزونه .....
Extreme Programming .....	برنامه‌نویسی مفروط .....
Elaboration (RUP phase) .....	تحلیل جزئیات (فازی از آریوچی) .....
Element .....	عنصر .....
Elm (SW) .....	الم .....
Emacs (Editor) .....	ایمکس .....
Email .....	رایانامه .....
Email Reader .....	رایانامه‌خوان .....
Ember.js (Web Framework) .....	امیر جی اس .....
Empire State Building .....	ساختمان امپایر استیت .....
Encapsulation (Objec-Oriented Programming) .....	کپسوله‌سازی .....
Encoding (Computing) .....	کدگذاری .....
Encryption .....	رمزگذاری .....
Endpoint (Communication) .....	نقطه پایانی .....
Enumerable .....	قابل شمارش .....
Enumeration (Programming) .....	شمارشی .....
Enterprise Java Beans (EJB) .....	بین‌های سازمانی جاوا .....
Epoch (Computing) .....	دوران .....
Eric Braude .....	اریک براد .....
Eric S. Raymond .....	اریک اس. ریموند .....
Error (Computing) .....	خطا .....
Ethernet .....	ایترنت .....

## F

Facebook .....	فیسبوک .....
False .....	نادرست .....
Falsy (Programming) .....	کذب .....
FarmVille (Online Game) .....	فارم ویل .....
Feature Creep .....	خرش کاربرد .....
Fetch (SW) .....	فچ .....
Full Duplex (Communication) .....	کاملاً دوطرفه .....
Function (Programming) .....	تابع ( برنامه‌نویسی ) .....
Function as a Service .....	تابع به صورت یک سرویس .....
Functional Programming .....	برنامه‌نویسی تابعی .....
File .....	فایل .....
File Server (Networking) .....	فایل‌سرور .....
File System .....	سامانه فایل‌بندی .....
File Transfer Protocol (FTP) .....	پروتکل انتقال فایل .....
FileZilla (SW) .....	فایل‌زیلا .....
Filter (for Controllers in Rails) .....	پالایه .....
Floating Point .....	ممیز شناور .....
For Loop .....	حلقه فور .....
Formal Methods .....	روش‌های صوری .....
Fortran (Programming Language) .....	فورترن ( زبان برنامه‌نویسی ) .....

Framework .....	چارچوب .....
Fred Brooks, Jr. .....	فِرد بروکس جونیور .....
Frontend .....	پیشین .....

## G

Gem (Ruby) .....	جم .....
Geocoding .....	کدیندی مکانی .....
George Santayana .....	جورج سانتایانا .....
Gerald Weinberg .....	جرالد واینبرگ .....
GET (HTTP Method) .....	دریافت (روش درخواست) .....
Getter Method (OOP) .....	مقدارگیر .....
Git (Version Control System) .....	گیت .....
GitHub .....	گیت‌هاب .....
GitLab .....	گیتلب .....
Global Variable (Programming) .....	متغیر سراسری .....
Gmail (Google Mail Service) .....	جی‌میل .....
Google .....	گوگل .....
Google App Engine .....	موتور اجرای برنامه گوگل .....
Google Calendar (Digital Calendar Service) .....	تقویم گوگل .....
Google Chrome (SW) .....	گوگل کروم .....
Google Docs (Online Word Processor) .....	گوگل داکس .....
Google Maps (Web Mapping Service) .....	گوگل مپ .....
Google News (News Aggregator Service) .....	گوگل نیوز .....
Google Search .....	جست‌وجوگر گوگل .....
Grace Murray Hopper .....	گریس موری هاپر .....
Graphical User Interface (GUI) .....	رابط کاربری گرافیکی .....
GraphQL .....	گرفکیوال .....
Greenwich Mean Time (GMT) .....	به وقت گرینویچ .....

## H

Hacker .....	هَكْر .....
Hardware .....	سخت‌افزار .....
Hash (Ruby Collection) .....	هَش .....
Hash map/table (Data Structure) .....	جدول درهم‌سازی .....
Header (Computing) .....	سرآیند .....
HyperText Markup Language .....	زبان نشانه‌گذاری ابرمتنی .....
HyperText Transfer Protocol (HTTP) .....	پروتکل انتقال ابرمتن .....
HyperText Transfer Protocol Secure (HTTPS) .....	پروتکل امن انتقال ابرمتن .....
Hierarchical Model (Database Model) .....	مدل سلسله‌مراتبی .....
Higher-order Function .....	تابع مرتبه بالاتر .....
Host (Computing) .....	میزبان .....

Hostname (Computing) .....	نام میزبان.....
Hotmail .....	هاتمیل.....

## I

IBM .....	آی‌بی‌ام.....
IBM's Generalized Markup Language .....	زبان نشانه‌گذاری تعمیم‌یافته آی‌بی‌ام.....
Icon (Computing) .....	آیکن.....
Immutable (Programming) .....	تغییرناپذیر.....
Imperative (Programming) .....	دستوری.....
Importing (Programming) .....	اضافه کردن.....
Inception (RUP phase) .....	آغازین.....
Index (CRUDI Action) .....	فهرست کردن (گُنش (CRUDI).....
Indentation (Programming) .....	تورفتگی.....
Information Technology .....	فناوری اطلاعات.....
Inheritance (OOP) .....	وراثت.....
Input/Output (I/O) .....	ورودی/خروجی.....
Instance (OOP) .....	نمونه.....
Instance Variable (OOP) .....	متغیر نمونه.....
Instance Method (OOP) .....	مِتْد نمونه.....
Integrated Development Environment (IDE) .....	محیط توسعه یکپارچه.....
Integration .....	یکپارچه‌سازی.....
Integration Testing .....	آزمون یکپارچگی.....
Intel .....	اینتل.....
Interactive .....	برهم‌کنشی.....
Interaction .....	برهم‌کنش.....
Interface (Computing) .....	واسط (رایانش).....
Internet Protocol (IP) .....	پروتکل اینترنت.....
Internet Protocol Suite (TCP/IP) .....	مجموعه پروتکل اینترنت.....
Interoperable .....	تعامل‌پذیر.....
Interpreter .....	مفسر.....
iOS .....	آی‌اواس.....
IP Address (Networking) .....	نشانه آی‌بی.....
iPhone .....	آیفون.....
IT Operations .....	عملیات فناوری اطلاعات.....
Iteration .....	تکرار.....
Iterator .....	پیمایشگر.....

## J

Java (Programming Language) .....	جاوا (زبان برنامه‌نویسی).....
JavaScript .....	جاوا‌اسکریپت.....
JavaServer Pages (Jakarta Server Pages) .....	صفحات سرور جاکارتا.....

Jay Leno .....	جی لنو (کمدین)
James Gosling .....	جیمز گاسلینگ
Jasmine (JavaScript Testing Tool) .....	چرمین
Jeff Bezos .....	جف بیزوس
Jeff Dean .....	جف دین
Just-in-time .....	بهنگام
John McCarthy .....	جان مک‌کارتی
Joshua Bloch .....	جاشوا بلک
jQuery .....	جی‌کوئری
JR Boyens .....	جی‌آر بوینز
JSON (JavaScript Object Notation) .....	جی‌سان

## K

Kayak (Travel Search Engine) .....	کایاک
Kanban .....	کان‌بان
Keyboard .....	صفحه کلید
Key-Value Pair .....	جفت کلید-مقدار
Ken Thompson .....	کن تامپسون
Kent Beck .....	کنت بک
Kernel (Operating System) .....	هسته
Kindle .....	کیندل

## L

Lambda (AWS) .....	لامبدا
Lambda Calculus .....	حساب لامبدا
Lambda Expression .....	عبارت لامبدا
Laptop .....	لپ‌تاپ
Learning Ruby (Book) .....	کتاب یادگیری روی
Legacy Code .....	میراث‌کد
Lexical Scope (Programming) .....	دامنه واژگانی
Lua (Programming Language) .....	لوآ (زبان برنامه‌نویسی)
Luiz Barroso .....	لوئیز باروسو
Library (Computing) .....	کتابخانه (رایانش)
Life Cycle .....	چرخه حیات
Linus Torvalds .....	لینوس توروالدز
Link (Web Hyperlink) .....	پیوند
Lisp (Programming Language) .....	لیسپ (زبان برنامه‌نویسی)
Lisp Machine .....	ماشین لیسپ
Literal (Programming) .....	لفظ
Literate Programming .....	برنامه‌نویسی ادبیانه
Local Variable (Programming) .....	متغیر محلی

میزبان محلی ..... Localhost (Computing)

## M

Mainframe Computer .....	بزرگ‌رایانه
Maintenance .....	نگهداری
Makefile .....	میک‌فایل
Malware .....	بدافزار
MapReduce (Programming Model) .....	نکاشت‌کاهش
Marcus Vitruvius Pollio .....	مارکوس ویتروویوس پولیو
Markup Language .....	زبان نشانه‌گذاری
Mars Climate Orbiter .....	مدارگرد اقلیمی مریخ
Massachusetts Institute of Technology .....	مؤسسه فناوری ماساچوست
Mechanism .....	سازوکار
Metaclass (OOP) .....	فراکلاس
Metaprogramming .....	فرابرnameه‌نویسی
Method (HTTP) .....	روش درخواست
Method (OOP) .....	مِند
Multihoming (Networking) .....	چندخانگی
Michael Swaine .....	مایکل سوانین
Microprocessor .....	ریزپردازنده
Microservice .....	ریزسرویس
Microsoft .....	مایکروسافت
Microsoft Azure .....	مایکروسافت آژور
Microsoft Edge .....	مایکروسافت اج
Microsoft Word .....	مایکروسافت وُرد
Microsoft Internet Explorer (SW) .....	اینترنت اکسپلورر ماکروسافت
Microsoft Internet Information Server (SW) .....	سرور اطلاعات اینترنتی ماکروسافت
Microsoft Outlook (SW) .....	مایکروسافت اوت‌لوك
Microsoft's .NET Framework .....	چارچوب دات‌نت شرکت ماکروسافت
Mix-in (OOP) .....	میکس‌این
MOCAS .....	موکاس
Model (MVC) .....	مدل
Model–View–Controller (Software Architecture) .....	مدل-نماینده-کنترل‌گر
Modular .....	پودمانی
Module (Programming) .....	پودمان
Module Testing .....	آزمون پودمان
Mozilla Firefox (SW) .....	موزیلا فایرفاکس
Mona Lisa (Leonardo da Vinci's Painting) .....	مونا لیزا
Monolithic .....	تک‌سنگی
Moore's Law .....	قانون مور
Morris Udall .....	موریس اودال
Motorola (Company) .....	موتورولا

## N

Namespace (Programming) .....	فضای نام
NCSA Mosaic (SW) .....	موزاییک
Newline (Character) .....	خط نو
Network (Computing) .....	شبکه
Network Interface .....	رابط شبکه
Netscape Navigator (SW) .....	نت اسکیپ نویگیتور
Nginx (SW) .....	انجین اکس
Nil (Programming) .....	نهی
Node.js .....	نود جی اس
Novell (Company) .....	نوول

## O

Object (Programming) .....	شیء (برنامه‌نویسی)
Objective-C (Programming Language) .....	آبجکتیو-سی
Object-oriented (Programming) .....	شی‌عکرا (برنامه‌نویسی)
Octet (Computing) .....	هشت‌تایی
Overhead .....	سریار
Online .....	آنلاین
Open Source (Software) .....	متن باز
Open Standard .....	استاندارد باز
Operating System .....	سیستم عامل
Orchestration (Microservices) .....	سازآرایی

## P

Package (Program Distribution) .....	بسته
Package Manager System .....	سامانه مدیریت بسته
Pay-as-you-go .....	پرداخت به میزان مصرف
Payload (Networking) .....	پایه‌بار
Pair Programming .....	برنامه‌نویسی دونفره
Palindrome .....	واروخوانه
Parsing (Computing) .....	تجزیه
Password .....	گذرواژه
Patch (Computing) .....	وصله
PATCH (HTTP Method) .....	ترمیم (روش درخواست)
Path (Computing) .....	مسیر (رایانش)
Peer-to-peer (Networking) .....	همتابه‌همتا
Performance .....	کارایی
Perl (Programming Language) .....	پرل (زبان برنامه‌نویسی)
Perl Compatible Regular Expressions (PCRE) .....	عبارات باقاعدۀ سازگار با پرل
Personal Computer .....	رایانه شخصی

Peter Norvig .....	پیتر نورویگ.....
Public (Programming) .....	عمومی.....
Public-key Cryptography .....	رمزگاری بر پایه کلید عمومی
PUT (HTTP Method) .....	گذاشتن (روش درخواست)
Python (Programming Language) .....	پایتون.....
PHP (Programming Language) .....	پاچ پی (زبان برنامه نویسی)
Pivotal Lab .....	پیووتال لبز.....
Pivotal Tracker .....	پیووتال ترکر (ابزار مدیریت پروژه)
Pine (SW) .....	پاین.....
Pipeline .....	خط لوله.....
Pitfall .....	خطر پنهان.....
Plan-and-Document .....	طرح و ثبت.....
Platform (Computing) .....	سکو.....
Poetry Mode (Programming Style in Ruby) .....	حالت شاعرانه.....
Port (Networking) .....	درگاه.....
POST (HTTP Method) .....	ارسال (روش درخواست)
Post Office Protocol (POP) .....	پروتکل دفتر پستی.....
Preston St. Pierre .....	پرستون سن پیر.....
Private (Programming) .....	خصوصی.....
Procedure (Programming) .....	رویه.....
Process .....	فرایند.....
Process (OS Concept) .....	پردازه.....
Processor (Computing) .....	پردازنده.....
Production (Environment) .....	عملیاتی.....
Program (Computer) .....	برنامه.....
Programmer .....	برنامه نویس.....
Programming .....	برنامه نویسی.....
Programming Idiom .....	اصطلاح برنامه نویسی.....
Programming Language .....	زبان برنامه نویسی.....
Programming Paradigm .....	پارادایم برنامه نویسی.....
Programming Ruby (Book) .....	کتاب برنامه نویسی روبی.....
Progressive Web Application .....	برنامه وب پیشرو.....
Promiscuous Pairing (Pair Programming) .....	جفت شدن بی قید و شرط.....
Protocol (Computing) .....	پروتکل.....
Prototype .....	پیش نمونه.....
Prototype (JS Web Framework) .....	پروتوتاپ.....

## Q

Quality Assurance (QA) .....	تضمین کیفیت .....
Query (Computing) .....	پرسمان .....
Qpopper (SW) .....	کیوپاپ .....

## R

Rails (Simplified verion of Ruby on Rails) .....	ریلز (صورت ساده‌شدهٔ روئی آن ریلز)
Rational Unified Process .....	فرایند یکپارچهٔ رشنال
React (JS Library) .....	ری اکت
Read (CRUD Action) .....	خواندن (گُبیش CRUD)
Read-Eval-Print Loop .....	حلقهٔ خواندن-ارزیابی-نمایش
Real-time .....	بلادرنگ
Redundancy .....	افزونگی
Redirection (Networking) .....	تغییرمسیر
Refactoring (Code) .....	بازسازی
Reflection (Computer Programming) .....	بازتاب
Regular Expression .....	عبارت باقاعدہ
Reusability .....	پاکاربردپذیری
Reuse .....	بازکاربرد
Relational Model (Database Model) .....	مدل رابطه‌ای
Reliability (Computing) .....	قابلیت اطمینان
Remote (Networking) .....	دوردست
Remote Procedure Call (RPC) .....	فراخوانی رویه‌ای دوردست
Render (Computing) .....	رندر
Renee McCauley .....	رئنی مک کاولی
Repository (Version Control) .....	مخزن
Representational State Transfer (REST) .....	انتقال بازنمودی حالت
Resource (Computing) .....	منبع
Responsive (Web Design) .....	واکنش‌گرا
Responsive Web Design .....	طراحی وب واکنش‌گرا
RESTful .....	میتنی بر REST
Ruby (Programming Language) .....	روئی
Ruby Best Practices (Book) .....	کتاب بهترین راهکارهای استفاده از روئی
Ruby On Rails .....	روئی آن ریلز
Rubyist .....	روئی‌باز
Runtime .....	زمان اجرا
Runtime System .....	سامانهٔ زمان اجرا
Robert Laganiere .....	رابرت لaganیر
Route .....	خط سیر
Roy Fielding .....	روی فیلدینگ
Rolling Stones .....	گروه موسیقی رولینگ استونز
RSpec (Testing tool) .....	آر-اسپیک

## S

Sabre (Travel Reservation System) .....	سابری
Salesforce .....	سینیلفورس (شرکت حوزهٔ فناوری)
Sanjay Ghemawat .....	سانجی قماوات
Sarah Mei .....	سارا می

Scala (Programming Language) .....	اسکالا (زبان برنامه‌نویسی)
Scalability .....	مقیاس‌پذیری
Scalable .....	مقیاس‌پذیر
Scale .....	مقیاس
Scheme (Part of URI) .....	طرح
Scope (Programming) .....	گستره
Scope Creep .....	خرش گستره
Scraping (Web/Data) .....	تراشیدن
Scrum .....	اسکرام
ScrumBan .....	اسکرامبان
Scripting Language .....	زبان اسکریپت‌نویسی
Search Engine .....	موتور جست‌وجو
Security (Computing) .....	امنیت
Seven Languages In Seven Weeks (Book) .....	کتاب هفت زبان در هفت هفته
Selector (CSS) .....	انتخابگر
Self-Organizing (Team Management) .....	خودسازمانده
Semantic Versioning .....	نسخه‌گذاری معنایی
Sentinel Value (Programming) .....	مقدار قرأول
Server .....	سرور
Serverless Computing .....	رایانش بی‌سرور
Service-oriented Architecture .....	معماری سرویس‌گرا
Serial .....	ترتیبی
Session (Computing) .....	نشست
Setter Method (OOP) .....	مقدارده
Subclass (OOP) .....	زیرکلاس
Sublime Text (Editor) .....	سابالایم تکست
Subsystem .....	زیرسامانه
Subtype (Polymorphism) .....	ریزنوغ
Superclass (OOP) .....	آبرکلاس
Supercomputer .....	آبررایانه
Swagger .....	سواگر
Switch (Networking) .....	سوئیچ
Symbol (Programming Primitive Data Type) .....	نماد
Syntactic Sugar (Programming) .....	قند نحوی
Syntax Highlighting .....	برجسته‌سازی نحوی
Synthesis .....	سنتر
System .....	سامانه
System Testing .....	آزمون سامانه
Syntax (Programming) .....	قواعد
Shell Script (Computing) .....	اسکریپت شل
Shimon Peres .....	شیمون پرز
Simula (Programming Language) .....	سیمولا
SimpleCov (Testing tool) .....	سیمپل کاو
Sinatra (Framework) .....	سیناترا
Slash (Punctuation) .....	خط مورب (نشان سجاوندی)
Smalltalk (Programming Language) .....	اسمال‌تاک
Smartphone .....	گوشی هوشمند

Snake Case (Programming)	نگارش ماری
Snapshot (Computing)	برگرفت
Software	نرم‌افزار
Software Architecture	معماری نرم‌افزار
Software as a Product	نرم‌افزار به صورت یک محصول
Software as a Service	نرم‌افزار به صورت یک سرویس
Software Bloat	نفخ نرم‌افزار
Software Development	توسعه نرم‌افزار
Software Development & IT Operations (Dev/Ops)	توسعه/عملیات
Software Engineering	مهندسی نرم‌افزار
Software Stack	پُشتئه نرم‌افزاری
Space (Character)	فاصله
Spaceship Operator (Programming)	عملگر فضایپما
Specification	مشخصه
Spiral Model	مدل مارپیچی
Spring	اسپرینگ
Sprint	اسپرینت
StackOverflow (Website)	إسٽك أُورفِلو
Stakeholder	ذی‌نفع
Standard Generalized Markup Language	زبان نشانه‌گذاری تعمیم‌یافته استاندارد
Standard Library (Programming)	کتابخانه استاندارد
Stateless (Protocol)	بدون حالت
Status Code (HTTP)	کد وضعیت
Static	ایستا
Static Typing (Type Checking)	نوع‌دهی ایستا
Steve Yegge	استیو یگ
Steven Ratkin	استیون راتکین
Stub (Programming)	خُردک
Stylesheet (Web Development)	شیوه‌نامه
String (Programming)	رشته
String Interpolation (Programming)	الحق رشته‌ای
Strongly Typed (Programming Language)	نوع‌دهی قوی

## T

Tab (Character)	جهش
Table (Database)	جدول (پایگاه داده)
Tablet	تبلت
Tag	تگ
Tcl (Programming Language)	تی‌سی‌ال (زبان برنامه‌نویسی)
Technology	فناوری
Test	آزمون
Test Case	آزمایه
Testable	آزمون‌پذیر
Test-Driven Development	توسعه آزمون محور

Tester .....	آزمون‌گر.....
Testing .....	آزمون.....
Turing Award .....	جایزهٔ تورینگ.....
Twitter .....	توبیتر.....
Type (Programming) .....	نوع.....
Type Casting/Conversion .....	تبديل نوع.....
Typing (Programming) .....	نوع دهی.....
Typography .....	تایپوگرافی.....
The Cathedral and the Bazaar (Book) .....	کتاب کلیسای جامع و بازار.....
The Movie Database (TMDb) .....	پایگاه داده فیلم.....
The New Hacker's Dictionary (Book) .....	کتاب واژه‌نامه هکر جدید.....
The Ruby Way (Book) .....	کتاب راه و روش روبي.....
The Ruby Programming Language (Book) .....	کتاب زبان برنامه‌نويسی روبي.....
Therac-25 (radiation therapy machine) .....	ماشين پرتو درمانی تراک-۲۵.....
Thunderbird (SW) .....	ثاندربرد.....
Thomas Erl .....	توماس إرل.....
Thread .....	ریسمان.....
Tim Berners-Lee .....	تیم بربرز-لی.....
Time-sharing .....	اشتراك زمانی.....
Timothy Lethbridge .....	تیموتو لیتبریج.....
Toyota (Company) .....	توبوتا.....
Token (Security) .....	نمودافزار.....
Tom Knight .....	تام نایت.....
Tonia Fox .....	تونیا فاکس.....
Topology (Networking) .....	هم‌بندی.....
Transition (RUP phase) .....	گذار.....
Transmission Control Protocol (TCP) .....	پروتکل کنترل انتقال.....
True .....	درست.....
Truthy (Programming) .....	حقیقی.....

## U

Unified Modeling Language (UML) .....	زبان مدل‌سازی یکپارچه.....
Uniform Access Principle (Programming) .....	اصل دسترسی یکنواخت.....
Uniform Resource Identifier (URI) .....	شناسانهٔ منبع یکنواخت.....
Uniform Resource Locator (URL) .....	مکان‌یاب منبع یکنواخت.....
University of Nebraska .....	دانشگاه نبراسکا.....
Unix (OS) .....	یونیکس.....
Unit Testing .....	آزمون واحد.....
Unsigned (Computing) .....	بی‌علامت.....
Update (CRUD Action) .....	به‌روزرسانی (گیش CRUD).....
US National Academies .....	آکادمی‌های ملی آمریکا.....
Use case .....	مورد استفاده.....
User (Computing) .....	کاربر.....
User Experience (UX) .....	تجربهٔ کاربری.....

User Interface .....	رابط کاربری .....
User Story .....	روایت کاربری .....
Username .....	نام کاربری .....
USS Hopper .....	بواسس اس هاپر .....
Utilization .....	بهره‌وری .....

## V

Validation .....	اعتبارسنجی .....
Validation (for Models in Rails) .....	تائیدیه .....
Variable (Programming) .....	متغیر .....
Velocity (Software Development Metric) .....	سرعت .....
Verification .....	درستی‌سنجی .....
Version Control .....	کنترل نسخه .....
Vue.js (JS Web Framework) .....	ویو جی اس .....
View (MVC) .....	نمای .....
Virtual Case File (FBI Software) .....	نرم‌افزار پروندهٔ مجازی افبی‌آی .....
Virtual Machine .....	ماشین مجازی .....
Virtual Machine Monitor .....	ناظر ماشین مجازی .....
Virtualization .....	مجازی‌سازی .....
Visual Studio Code (Editor) .....	ویژوال استودیو کد .....

## W

Warehouse Scale Computer .....	رايانه‌انبارگون .....
Warning (Computing) .....	هشدار .....
Watchpoint (Programming/Debugging) .....	نقطهٔ مراقبت .....
Waterfall Model .....	مدل آشیاری .....
Weakly Typed (Programming Language) .....	نوع‌دهی ضعیف .....
Web (World Wide Web) .....	وب .....
Web Portal .....	درگاه وب .....
Web Server .....	سرور وب .....
Weblog .....	بلاگ .....
Webpage .....	صفحهٔ وب .....
Website .....	وبگاه .....
Weinberg .....	واینبرگ .....
Werner Vogels .....	ورنر فوگلز .....
Whitespace (Character) .....	فاصلهٔ خالي .....
Wikimedia Commons .....	ويکي‌انبار .....
Wikipedia .....	ويکي‌پديا .....
Word Processor (Software) .....	واژه‌پرداز (نرم‌افزار) .....
Workflow .....	گردش کار .....
World Wide Web .....	شبکهٔ جهانی وب .....

کنسرسیوم جهانی وب ..... World Wide Web Consortium (W3C)  
لگافپیچ ..... Wrapper (Programming)

Y

یاهو! ..... Yahoo!  
یوکیهیرو ماتسوموتو ..... Yukihiro "Matz" Matsumoto  
یوتیوب ..... YouTube (Online Video Sharing Platform)

Z

زند ..... Zend





# واژه‌نامهٔ فارسی به انگلیسی

default

2001: A Space Odyssey (Movie) ..... فیلم ادیسهٔ فضایی

۷

Objective-C (Programming Language)	آجکتیو-سی
Aqueduct of Segovia	آب‌گذر سگوبیا
Apache James (SW)	آپاچی جیمز
Apache Maven	آپاچی میون
RSpec (Testing tool)	آر-اسپیک
Array (Data Structure)	آرایه
Associative Array (Data Structure)	آرایهٔ انجمنی
Argument (Programming)	آرگومان (برنامه‌نویسی)
Armando Fox	آرماندو فاکس
Test Case	آزمایه
Test	آزمون
Acceptance Testing	آزمون پذیرش
Module Testing	آزمون بودمان
System Testing	آزمون سامانه
Unit Testing	آزمون واحد
Integration Testing	آزمون یکپارچگی
Testable	آزمون‌پذیر
Tester	آزمون‌گر
Inception (RUP phase)	آغازین
US National Academies	آکادمی‌های ملی آمریکا
Amazon.com	آمازون (شرکت حوزهٔ فناوری)
Content-Oriented Didactic (COD)	آموزشی محتوامحور
Online	آنلاین
iOS	آی‌اواس
IBM	آی‌بی‌ام
iPhone	آیفون

## آیکن Icon (Computing) .....

Supercomputer .....	آبرایانه
Superclass (OOP) .....	آبرکلاس
App (Application Software/Program) .....	آپ
Apple .....	اپل
Apple Safari (SW) .....	اپل سافاری
Apple Mail (SW) .....	اپل میل
Agile Alliance .....	اتحادیه روش چابک
Connection (Networking) .....	اتصال
Executable .....	اجرا بدیز
Authorization .....	احراز مجوز دسترسی
Authentication .....	احراز هویت
Basic Authentication (HTTP) .....	احراز هویت پایه
Conciseness .....	اختصار
edX .....	ایدکس
POST (HTTP Method) .....	ارسال (روش درخواست)
Eric S. Raymond .....	اریک اس. ری蒙د
Eric Braude .....	اریک براد
Sprint .....	اسپرینت
Spring .....	اسپرینگ
Open Standard .....	استاندارد باز
Exception .....	استثنای
StackOverflow (Website) .....	استک اُرفلو
Steve Yegge .....	استیو یگ
Steven Ratkin .....	استیون راتکین
Scala (Programming Language) .....	اسکالا (زبان برنامه نویسی)
Scrum .....	اسکرام
ScrumBan .....	اسکرامبان
Shell Script (Computing) .....	اسکریپت شل
Smalltalk (Programming Language) .....	اسمال تاک
Time-sharing .....	اشتراك زمانی
Bug (Computer) .....	اشکال
Debugger .....	اشکال زدا
Debugging .....	اشکال زدایی
Programming Idiom .....	اصطلاح برنامه نویسی
Uniform Access Principle (Programming) .....	اصل دسترسی یکنواخت
Importing (Programming) .....	اضافه کردن
Validation .....	اعتبارسنجی
Declaration .....	اعلان
Document Type Declaration (XML Instruction) .....	اعلان نوع سند
Redundancy .....	افزونگی
Extention (Software) .....	افزونه

String Interpolation (Programming) .....	الحاق رشته‌ای
Design Pattern .....	الگوی طراحی
Elm (SW) .....	إلم
Ember.js (Web Framework) .....	امبر جی اس
American Airlines .....	امریکن ایرلاینز
Security (Computing) .....	امنیت
Selector (CSS) .....	انتخابگر
Representational State Transfer (REST) .....	انتقال بازنمودی حالت
European Computer Manufacturers Association .....	انجمن تولیدکنندگان رایانه اروپا (ECMA)
British Computer Society .....	انجمن رایانه بریتانیا
Nginx (SW) .....	إنجينكس
Android (OS) .....	اندروید
Andy Hunt .....	آندری هانت
Angular (Web Framework) .....	آنگولار
AOL .....	ای او ال
eBay .....	ای بی
Ethernet .....	ایترنوت
Create (CRUD Action) .....	ایجاد (گُنیش CRUD)
Create, Read, Update, Delete, Index .....	ایجاد، خواندن، بروزرسانی، حذف و فهرست کردن (CRUDI)
AJAX (Asynchronous JavaScript And XML) .....	ایچکس
Static .....	ایستا
Emacs (Editor) .....	ایمکس
Microsoft Internet Explorer (SW) .....	اینترنت اکسپلورر ماسکروسافت
Intel .....	اینتل

## ب

Barbara Liskov .....	باربارا لیسکف
Download .....	بارگیری
Reflection (Computer Programming) .....	بازتاب
Refactoring (Code) .....	بازسازی
Reuse .....	بازکاربرد
Reusability .....	بازکاربردپذیری
Bundler (Ruby) .....	باندلر
Byte (Computing) .....	بايت
Assertion (Computer Programming) .....	بایستگی (برنامه‌نویسی)
Malware .....	بدافزار
Stateless (Protocol) .....	بدون حالت
Bertrand Meyer .....	برتراند مایر
Syntax Highlighting .....	برجسته‌سازی نحوی
Snapshot (Computing) .....	برگرفت
Program (Computer) .....	برنامه
Progressive Web Application .....	برنامهٔ وب پیشرو

Programmer .....	برنامه‌نویس .....
Programming .....	برنامه‌نویسی .....
Literate Programming .....	برنامه‌نویسی ادبیانه .....
Functional Programming .....	برنامه‌نویسی تابعی .....
Pair Programming .....	برنامه‌نویسی دونفره .....
Computer Programming .....	برنامه‌نویسی رایانه‌ای .....
Extreme Programming .....	برنامه‌نویسی مفرط .....
Interaction .....	برهمکنش .....
Interactive .....	برهمکنشی .....
Mainframe Computer .....	بزرگ‌رایانه .....
Closure (Programming) .....	بستار .....
Package (Program Distribution) .....	بسته .....
Backbone.js (Web Framework) .....	بک‌بون جی‌اس .....
Real-time .....	بلادرنگ .....
Block (Programming) .....	بلوک .....
Bootstrap (Front-end Framework) .....	بوت‌استرپ .....
Boolean (Data Type) .....	بولی .....
Greenwich Mean Time (GMT) .....	به وقت گرینویچ .....
Update (CRUD Action) .....	بهروزرسانی (گیش CRUD) .....
Utilization .....	بهره‌وری .....
Deployment (Software) .....	بهکاراندازی .....
Just-in-time .....	بهنگام .....
Bit (Computing) .....	بیت .....
Bit Blit .....	بیتبلیت .....
BitTorrent .....	بیت‌تورنت .....
Bitmap .....	بیتم .....
Unsigned (Computing) .....	بی‌علامت .....
Enterprise Java Beans (EJB) .....	بین‌های سازمانی جاوا .....

## پ

Programming Paradigm .....	پارادایم برنامه‌نویسی .....
Filter (for Controllers in Rails) .....	پالای .....
Python (Programming Language) .....	پایتون .....
Database .....	پایگاه داده .....
The Movie Database (TMDb) .....	پایگاه داده فیلم .....
Pine (SW) .....	پاین .....
Payload (Networking) .....	پایه‌بار .....
Pay-as-you-go .....	پرداخت به میزان مصرف .....
Processor (Computing) .....	پردازنده .....
Process (OS Concept) .....	پردازه .....
Preston St. Pierre .....	پرستون سن‌پیر .....
Query (Computing) .....	پرسمان .....
Perl (Programming Language) .....	پل (زبان برنامه‌نویسی) .....
Protocol (Computing) .....	پروتکل .....

HyperText Transfer Protocol Secure (HTTPS)	پروتکل امن انتقال ابرمتن
HyperText Transfer Protocol (HTTP)	پروتکل انتقال ابرمتن
File Transfer Protocol (FTP)	پروتکل انتقال فایل
Internet Protocol (IP)	پروتکل اینترنت
Post Office Protocol (POP)	پروتکل دفتر پستی
Transmission Control Protocol (TCP)	پروتکل کنترل انتقال
Prototype (JS Web Framework)	پروتوتایپ
Coding/Hands-On Integrated Projects (CHIPS)	پروژه‌ها و فعالیت‌های عملی/کدنویسی
Backend	بَكَان
Software Stack	پُشتَئَة نرم‌افزاری
Module (Programming)	پُودمان
Modular	پُودمانی
Coverage (Testing)	پوشش
Dynamic	پویا
Animation	پویانمایی
PHP (Programming Language)	پیاج‌بی (زبان برنامه‌نویسی)
Peter Norvig	پیتر نورویگ
Prototype	پیش‌نمونه
Frontend	پیشین
Configuration	پیکربندی
Iterator	پیمایشگر
Link (Web Hyperlink)	پیوند
Pivotal Tracker	پیووچال ترکر (ابزار مدیریت پروژه)
Pivotal Lab	پیووچال لَب

## ت

Validation (for Models in Rails)	تأییدیه
Function (Programming)	تابع (برنامه‌نویسی)
Function as a Service	تابع به صورت یک سرویس
Higher-order Function	تابع مرتبه بالاتر
Tom Knight	تام نایت
Thunderbird (SW)	تادِرِید
Typography	تایپوگرافی
Type Casting/Conversion	تبديل نوع
Tablet	تبلت
E-commerce	تجارت الکترونیک
User Experience (UX)	تجربه کاربری
Abstraction (Computer Science)	تجريد (علوم رایانه)
Parsing (Computing)	تجزیه
Elaboration (RUP phase)	تحلیل جزئیات (فازی از آریوپی)
Scraping (Web/Data)	تراشیدن
Serial	ترتیبی
Composition (OOP)	ترکیب
PATCH (HTTP Method)	ترمیم (روش درخواست)

Quality Assurance (QA)	تضمین کیفیت
Interoperable	تعامل پذیر
Redirection (Networking)	تغییرمسیر
Immutable (Programming)	تغییرناپذیر
Google Calendar (Digital Calendar Service)	تقویم گوگل
Iteration	تکرار
Monolithic	تکسنگی
Tag	تگ
Indentation (Programming)	تورفندگی
Develop (Software)	توسعه
Test-Driven Development	توسعه آزمون محور
Behavior-Driven Development	توسعه رفتار محور
Software Development	توسعه نرم افزار
Software Development & IT Operations (Dev/Ops)	توسعه/عملیات
Developer (Software)	توسعه دهنده
Thomas Erl	توماس ارل
Tonia Fox	تونیا فاکس
Toyota (Company)	توبوتا
Twitter	توییتر
Nil (Programming)	تهی
Tcl (Programming Language)	تیسی ال (زبان برنامه نویسی)
Tim Berners-Lee	تیم بربنر-لی
Timothy Lethbridge	تیموثی لیثبریج

## ج

Joshua Bloch	جاشوا بلک
John McCarthy	جان مک کارتی
Java (Programming Language)	جاوا (زبان برنامه نویسی)
JavaScript	جاوا اسکریپت
Turing Award	جایزه تورینگ
Ancestor (OOP)	جد
Command-Query Separation	جداسازی فرمان- درخواست
Table (Database)	جدول (پایگاه داده)
Hash map/table (Data Structure)	جدول درهم سازی
Gerald Weinberg	جرالد واینبرگ
Jasmine (JavaScript Testing Tool)	جَزمِین
Google Search	جست و جوگر گوگل
Jeff Bezos	جف بیروس
Jeff Dean	جف دین
Promiscuous Pairing (Pair Programming)	جفت شدن بی قید و شرط
Key-Value Pair	جفت کلید- مقدار
Gem (Ruby)	جم
Django	جنگو
George Santayana	جورج سانتایانا

Tab (Character) .....	جہش .....
Jay Leno .....	چی لنو (کمدین) .....
JR Boyens .....	چی آر بونز .....
JSON (JavaScript Object Notation) .....	چی سان .....
jQuery .....	چی کوئری .....
James Gosling .....	چیمز گاسلینگ .....
Gmail (Google Mail Service) .....	چی میل .....

**ج**

Agile .....	چابک .....
Framework .....	چارچوب .....
Microsoft's .NET Framework .....	چارچوب داتنت شرکت ماکروسافت .....
Life Cycle .....	چرخهٔ حیات .....
Multihoming (Networking) .....	چندخانگی .....

**ح**

Poetry Mode (Programming Style in Ruby) .....	حالت شاعرانه .....
DELETE (HTTP Method) .....	حذف (روش درخواست) .....
Lambda Calculus .....	حساب لامبدا .....
Truthy (Programming) .....	حقیقی .....
Read-Eval-Print Loop .....	حلقهٔ خواندن-ارزیابی-نمایش .....
For Loop .....	حلقهٔ فور .....

**خ**

Stub (Programming) .....	خُردہ کد .....
Feature Creep .....	خرش کاربرد .....
Scope Creep .....	خرش گستره .....
Private (Programming) .....	خصوصی .....
Attribute (Computing) .....	خصوصیت .....
Route .....	خط سیر .....
Command-line .....	خط فرمان .....
Pipeline .....	خط لولہ .....
Slash (Punctuation) .....	خط مورب (نشان سجاوندی) .....
Newline (Character) .....	خط نو .....
Error (Computing) .....	خطا .....
Pitfall .....	خطر پنهان .....
Read (CRUD Action) .....	خواندن (کُشش (CRUD (CRUD Action) .....
Don't Repeat Yourself (DRY) .....	خودت را تکرار مکن .....

Self-Organizing (Team Management) .....	خودسازمانده
Automation .....	خودکارسازی
Computer Cluster .....	خوشه

د

Data-Intensive .....	داده‌بر
Data Structure .....	داده‌ساختار
Comic Strip .....	داستان مصور
Docker .....	داکر
Douglas Crockford .....	داگلاس کرافرد
Lexical Scope (Programming) .....	دامنه واژگانی
Carnegie Mellon University .....	دانشگاه کارنگی ملون
University of Nebraska .....	دانشگاه نبراسکا
Donald Knuth .....	دانلد کنوث
Directory (Computing) .....	دایرکتوری
True .....	درست
Verification .....	درستی‌سنجدی
Port (Networking) .....	درگاه
Web Portal .....	درگاه وب
GET (HTTP Method) .....	دربافت (روش درخواست)
Availability .....	دسترسی‌پذیری
Imperative (Programming) .....	دستوری
Desktop .....	دسکتاپ
Dennis Ritchie .....	دنیس ریچی
Binary (Computing) .....	دودوبی
Epoch (Computing) .....	دوران
Remote (Networking) .....	دوردست
Dilbert (Comic Strip) .....	دیلبرت
Dave Thomas .....	دیو توماس
David Patterson .....	دیوید پترسون

ذ

Caching (Software) .....	ذخیره‌سازی موقت
Stakeholder .....	ذی‌نفع

ر

Robert Laganiere .....	رابرت لaganiere
Network Interface .....	رابط شبکه

User Interface .....	رابط کاربری .....
Graphical User Interface (GUI) .....	رابط کاربری گرافیکی .....
Email .....	ایمیل .....
Email Reader .....	رایانامه‌خوان .....
Computing .....	رایانش .....
Cloud Computing .....	رایانش ابری .....
Serverless Computing .....	رایانش بی‌سرور .....
Computer .....	رایانه .....
Warehouse Scale Computer .....	رایانهٔ انبارگون .....
Personal Computer .....	رایانهٔ شخصی .....
String (Programming) .....	رشته .....
Encryption .....	رمزگذاری .....
Public-key Cryptography .....	رمزگاری بر پایهٔ کلید عمومی .....
Render (Computing) .....	رندر .....
Renee McCauley .....	رینی مک کاولی .....
User Story .....	روایت کاربری .....
Ruby (Programming Language) .....	روبی .....
Ruby On Rails .....	روبی آن ریلز .....
Rubyist .....	روبی‌باز .....
Method (HTTP) .....	روش درخواست .....
Formal Methods .....	روش‌های صوری .....
Roy Fielding .....	روی فیلدینگ .....
Event (Computing) .....	رویداد .....
Procedure (Programming) .....	رویه .....
React (JS Library) .....	ریاکت .....
Microprocessor .....	ریزپردازنده .....
Microservice .....	ریزسرویس .....
Thread .....	ریسمان .....
Rails (Simplified verion of Ruby on Rails) .....	ریلز (صورت ساده‌شدهٔ روبی آن ریلز) .....

ز

Scripting Language .....	زبان اسکریپتنویسی .....
Assembly Language .....	زبان آسیمبلی .....
Programming Language .....	زبان برنامه‌نویسی .....
Unified Modeling Language (UML) .....	زبان مدل‌سازی یکپارچه .....
Markup Language .....	زبان نشانه‌گذاری .....
HyperText Markup Language .....	زبان نشانه‌گذاری ابرمتندی .....
IBM's Generalized Markup Language .....	زبان نشانه‌گذاری تعمیم‌یافتهٔ آی‌بی‌ام .....
Standard Generalized Markup Language .....	زبان نشانه‌گذاری تعمیم‌یافتهٔ استاندارد .....
Extensible Markup Language .....	زبان نشانه‌گذاری گسترش‌پذیر .....
Runtime .....	زمان اجرا .....
Compile Time .....	زمان کامپایل .....
Zend .....	زند .....
Beautiful Code .....	زیباگد .....

Subsystem .....	زیرسامانه
Subclass (OOP) .....	زیرکلاس
Subtype (Polymorphism) .....	زیرنوع

## س

Sabre (Travel Reservation System) .....	سابری
Sublime Text (Editor) .....	سابلایم تکست
Construction (RUP phase) .....	ساخت (فازی از آریوپی)
Empire State Building .....	ساختمان امپایر استیت
Sarah Mei .....	سارا می
Orchestration (Microservices) .....	سازارآبادی
Constructor (OOP) .....	سازنده
Mechanism .....	سازوکار
System .....	سامانه
Runtime System .....	سامانه زمان اجرا
File System .....	سامانه فایل‌بندی
Package Manager System .....	سامانه مدیریت بسته
Domain Name System (DNS) .....	سامانه نام دامنه
Sanjay Ghemawat .....	سانجی قماوات
Cyberduck (SW) .....	سایپرداک
Hardware .....	سخت‌افزار
Header (Computing) .....	سرآیند
Overhead .....	سربار
Velocity (Software Development Metric) .....	سرعت
CERN .....	سین
Server .....	سرور
Microsoft Internet Information Server (SW) .....	سرور اطلاعات اینترنتی ماکروسافت
Application Server .....	سرور نرم‌افزارهای کاربردی
Web Server .....	سرور وب
Apache Httpd (SW) .....	سرور وب آپاچی
Amazon Web Services (AWS) .....	سرвис‌های وب آمازون
Custom (Customized) .....	سفارشی
Platform (Computing) .....	سکو
Synthesis .....	سنتر
Document (Computing) .....	سند
Swagger .....	سواگر
Switch (Networking) .....	سوئیچ
C (Programming Language) .....	سی
C# (Programming Language) .....	سی شارپ
CLU (Programming Language) .....	سی‌ال‌بو
C++ (Programming Language) .....	سی‌پلاس‌پلاس
Operating System .....	سیستم عامل
Salesforce .....	سینلزفورس (شرکت حوزه فناوری)
SimpleCov (Testing tool) .....	سیمپل‌کاو

Simula (Programming Language) .....	سیمولا
Sinatra (Framework) .....	سیناترا

## ش

Network (Computing) .....	شبکه
World Wide Web .....	شبکهٔ جهانی وب
CGI Group (Information Technology Company) .....	شرکت فناوری گروه سی‌جی‌آی
Enumeration (Programming) .....	شمارشی
Uniform Resource Identifier (URI) .....	شناسانهٔ منبع یکنواخت
Object (Programming) .....	شیء (برنامه‌نویسی)
Object-oriented (Programming) .....	شیء‌گرا (برنامه‌نویسی)
Shimon Peres .....	شیمون پرز
Stylesheet (Web Development) .....	شیوه‌نامه
Cascading Style Sheets .....	شیوه‌نامهٔ آبشاری

## ص

Economies of Scale .....	صرفه به مقیاس
JavaServer Pages (Jakarta Server Pages) .....	صفحات سرور جاکارتا
Webpage .....	صفحهٔ وب
Keyboard .....	صفحه کلید
Active Server Pages (ASP.NET) .....	صفحه‌های سرور فعال

## ط

Big Design Up Front .....	طراحی عظیم از ابتدا
Responsive Web Design .....	طراحی وب واکنش‌گرا
Scheme (Part of URI) .....	طرح
Plan-and-Document .....	طرح-و-ثبت

## ع

Perl Compatible Regular Expressions (PCRE) .....	عبارات باقاعدهٔ سازگار با پرل
Regular Expression .....	عبارت باقاعده
Lambda Expression .....	عبارت لامبدا
Computer Science .....	علوم رایانه
Spaceship Operator (Programming) .....	عملگر فضایپیما
IT Operations .....	عملیات فناوری اطلاعات

Production (Environment) .....	عملیاتی
Public (Programming) .....	عمومی
Element .....	عنصر

## ف

FarmVille (Online Game) .....	فารم ویل
Space (Character) .....	فاصله
Whitespace (Character) .....	فاصلهٔ خالی
File .....	فایل
FileZilla (SW) .....	فایل زیلا
File Server (Networking) .....	فایل سرور
Fetch (SW) .....	فیچ
Metaprogramming .....	فرابرnameهنویسی
Call (Function/Method) .....	فراخوانی
Remote Procedure Call (RPC) .....	فراخوانی رویه‌ای دوردست
Caller (Function/Method) .....	فراخوانی دهنده
Callee (Function/Method) .....	فراخوانی گیرنده
Metaclass (OOP) .....	فرآکلاس
Process .....	فرایند
Rational Unified Process .....	فرایند یکپارچه رشنال
Fred Brooks, Jr. .....	فرد بروکس جونیور
App Store .....	فروشگاه نرم‌افزار کاربردی
Namespace (Programming) .....	فضای نام
Technology .....	فناوری
Information Technology .....	فناوری اطلاعات
Fortran (Programming Language) .....	فورترن (زبان برنامه‌نویسی)
Index (CRUDI Action) .....	فهرست کردن (گیش CRUDI)
Facebook .....	فیسبوک

## ق

Enumerable .....	قابل شمارش
Reliability (Computing) .....	قابلیت اطمینان
Data Format .....	قالب داده
Moore's Law .....	قانون مور
Convention Over Configuration (Design Paradigm) .....	قراردادمحوری به جای پیکربندی
Syntactic Sugar (Programming) .....	قند نحوی
Syntax (Programming) .....	قواعد

ک

Choreography (Microservices) .....	کارآرایی .....
Character (Computing) .....	کاراکتر (رایانش) .....
Performance .....	کارایی .....
User (Computing) .....	کاربر .....
Compiler .....	کامپایلر .....
CompuServe .....	کامپیوسر .....
Full Duplex (Communication) .....	کاملاً دوطرفه .....
Kanban .....	کانبان .....
Container (Virtualization) .....	کانتینر .....
Kayak (Travel Search Engine) .....	کایاک .....
Encapsulation (Object-Oriented Programming) .....	کپسوله‌سازی .....
Programming Ruby (Book) .....	کتاب برنامه‌نویسی روبی .....
Ruby Best Practices (Book) .....	کتاب بهترین راهکارهای استفاده از روبی .....
The Ruby Way (Book) .....	کتاب راه و روش روبی .....
The Ruby Programming Language (Book) .....	کتاب زبان برنامه‌نویسی روبی .....
The Cathedral and the Bazaar (Book) .....	کتاب کلیساي جامع و بازار .....
The New Hacker's Dictionary (Book) .....	کتاب واژه‌نامه هکر جدید .....
Seven Languages In Seven Weeks (Book) .....	کتاب هفت زبان در هفت هفته .....
Learning Ruby (Book) .....	کتاب یادگیری روبی .....
Library (Computing) .....	کتابخانه (رایانش) .....
Standard Library (Programming) .....	کتابخانه استاندارد .....
Status Code (HTTP) .....	کد وضعیت .....
Geocoding .....	کدبندی مکانی .....
Encoding (Computing) .....	کدگذاری .....
Codio .....	گدیو .....
Falsy (Programming) .....	کذب .....
Cray .....	کری .....
Class (Computer Programming) .....	کلاس ( برنامه‌نویسی ) .....
Client .....	کلاینت .....
Clojure (Lisp Dialect) .....	کلوژر ( گویشی از زبان برنامه‌نویسی لیسپ ) .....
Ken Thompson .....	کن تامپسون .....
Kent Beck .....	کنٹ بِک .....
Control Flow (Programming) .....	کنترل جریان .....
Version Control .....	کنترل نسخه .....
Controller (MVC) .....	کنترل‌گر .....
World Wide Web Consortium (W3C) .....	کنسرسیوم جهانی وب .....
Console .....	کنسول .....
Action (Software Engineering) .....	کُش .....
Confucius (Chinese Philosopher) .....	کنفوتسیوس .....
COBOL (Programming Language) .....	کوبال ( زبان برنامه‌نویسی ) .....
Cookie (HTTP) .....	کوکی .....
Kindle .....	کیندل .....
Qpopper (SW) .....	کیوپاپر .....
Cucumber (Software tool) .....	کیوکامبر ( ابزاری برای توسعهٔ رفتارمحور ) .....

## گ

Transition (RUP phase) .....	گذار.....
PUT (HTTP Method) .....	گذاشتن (روش درخواست).....
Password .....	گزروازه.....
Collection (Abstract Data Type) .....	گردآورد.....
Workflow .....	گردش کار.....
GraphQL .....	گرفکیوال.....
Rolling Stones .....	گروه موسیقی رولینگ استونز.....
Grace Murray Hopper .....	گریس موری هاپر.....
Scope (Programming) .....	گستره.....
Bottleneck .....	گلوگاه.....
Smartphone .....	گوشی هوشمند.....
Google .....	گوگل.....
Google Docs (Online Word Processor) .....	گوگل داکس.....
Google Chrome (SW) .....	گوگل کروم.....
Google Maps (Web Mapping Service) .....	گوگل مپ.....
Google News (News Aggregator Service) .....	گوگل نیوز.....
Git (Version Control System) .....	گیت.....
GitLab .....	گیتلب.....
GitHub .....	گیت‌هاب.....

## ل

Lambda (AWS) .....	لامیدا.....
Affordable Care Act .....	لایحه مراقبت مقررین به صرفه.....
Laptop .....	لپ‌تاپ.....
Wrapper (Programming) .....	لفافپیچ.....
Literal (Programming) .....	لفظ.....
Lua (Programming Language) .....	لوآ (زبان برنامه‌نویسی).....
Luiz Barroso .....	لوئیز باروسو.....
Lisp (Programming Language) .....	لیسپ (زبان برنامه‌نویسی).....
Linus Torvalds .....	لینوس توروالدز.....

## م

Marcus Vitruvius Pollio .....	مارکوس ویتروویوس پولیو.....
Therac-25 (radiation therapy machine) .....	ماشین پرتو درمانی تراک-۲۵.....
Lisp Machine .....	ماشین لیسپ.....
Virtual Machine .....	ماشین مجازی.....
Microsoft .....	مایکروسافت.....
Microsoft Azure .....	مایکروسافت آزور.....
Microsoft Edge .....	مایکروسافت اج.....

Microsoft Outlook (SW)	مایکروسافت اوت‌لوك
Microsoft Word	مایکروسافت ورد
Michael Swaine	مایکل سوین
RESTful	REST
Method (OOP)	متند
Accessor Method	متند دسترسی
Class Method (OOP)	متند کلاس
Instance Method (OOP)	متند نمونه
Variable (Programming)	متغیر
Global Variable (Programming)	متغیر سراسری
Class Variable (OOP)	متغیر کلاس
Local Variable (Programming)	متغیر محلی
Instance Variable (OOP)	متغیر نمونه
Body (of request/message/packet)	من
Open Source (Software)	متن‌باز
Virtualization	مجازی‌سازی
Internet Protocol Suite (TCP/IP)	مجموعهٔ پروتکل اینترنت
Codebase	مجموعهٔ کد
Integrated Development Environment (IDE)	محیط توسعهٔ یکپارچه
Repository (Version Control)	مخزن
Credentials (Security)	مدارک هویتی
Mars Climate Orbiter	مدارگرد اقلیمی مریخ
Model (MVC)	مدل
Waterfall Model	مدل آشیاری
Capability Maturity Model	مدل بلوغ قابلیت
Relational Model (Database Model)	مدل رابطه‌ای
Hierarchical Model (Database Model)	مدل سلسله‌مراتبی
Spiral Model	مدل مارپیچی
Business Modeling	مدل‌سازی کسب‌وکار
Model–View–Controller (Software Architecture)	مدل‌نما–کنترل‌گر
Chief Technology Officer	مدیر ارشد فناوری
Datacenter	مرکز داده
Browser (Web)	مرورگر
Documentation (Set of Documents)	مستندات
Documentation (Act of Documenting)	مستندسازی
Path (Computing)	مسیر (رایانش)
Specification	مشخصه
Service-oriented Architecture	معماری سرویس‌گرا
Software Architecture	معماری نرم‌افزار
Interpreter	تفسر
Constant Value (Programming)	مقدار ثابت
Sentinel Value (Programming)	مقدار قراول
Setter Method (OOP)	مقدارده
Getter Method (OOP)	مقدارگیر
Scale	مقیاس
Scalable	مقیاس‌پذیر
Scalability	مقیاس‌پذیری

Uniform Resource Locator (URL) .....	مکانیاب منبع یکنواخت
Floating Point .....	ممیز شناور
Resource (Computing) .....	منبع
Deprecation (Software) .....	منسوخ شدن
Agile Manifesto .....	منشور چاپک
Business/Application Logic .....	منطق کسب و کار
Dropdown Menu .....	منوی کشویی
Google App Engine .....	موتور اجرای برنامه گوگل
Search Engine .....	موتور جستجو
Motorola (Company) .....	موتورولا
Use case .....	مورد استفاده
Morris Udall .....	موریس اودال
NCSA Mosaic (SW) .....	موزاییک
Mozilla Firefox (SW) .....	موزیلا فایرفاکس
Massachusetts Institute of Technology .....	مؤسسه فناوری ماساچوست
Ariane 4 Rocket .....	موشک آریان <sup>۴</sup>
Ariane 5 Rocket .....	موشک آریان <sup>۵</sup>
MOCAS .....	موکاس
Component (Computing) .....	مؤلفه
Mona Lisa (Leonardo da Vinci's Painting) .....	مونا لیزا
Software Engineering .....	مهندسی نرم افزار
Legacy Code .....	میراث کد
Host (Computing) .....	میزبان
Localhost (Computing) .....	میزبان محلی
Mix-in (OOP) .....	میکس این
Makefile .....	میکافایل

## ن

False .....	نادرست
Virtual Machine Monitor .....	ناظر ماشین مجازی
Username .....	نام کاربری
Hostname (Computing) .....	نام میزبان
Netscape Navigator (SW) .....	ناتاسکیپ نویگیتور
Software .....	نرم افزار
Software as a Service .....	نرم افزار به صورت یک سرویس
Software as a Product .....	نرم افزار به صورت یک محصول
Virtual Case File (FBI Software) .....	نرم افزار پرونده مجازی افبی آی
Application (Software/Program) .....	نرم افزار کاربردی
Semantic Versioning .....	نسخه گذاری معنایی
Bookmark (Web) .....	نشانک
IP Address (Networking) .....	نشانه آی پی
Session (Computing) .....	نشست
Software Bloat .....	نفح نرم افزار
Endpoint (Communication) .....	نقطه پایانی

Breakpoint (Programming/Debugging) .....	نقطهٔ توقف .....
Watchpoint (Programming/Debugging) .....	نقطهٔ مراقبت .....
Camel Case (Programming) .....	نگارش شتری .....
Snake Case (Programming) .....	نگارش ماری .....
MapReduce (Programming Model) .....	نگاشت‌کاهش .....
Maintenance .....	نگهداری .....
View (MVC) .....	نما .....
Symbol (Programming Primitive Data Type) .....	نماد .....
Token (Security) .....	نمودافزار .....
Instance (OOP) .....	نمونه .....
Address Bar (Browser) .....	نوار آدرس .....
Node.js .....	نود چی‌اس .....
Type (Programming) .....	نوع .....
Abstract Data Type .....	نوع دادهٔ انتزاعی .....
Typing (Programming) .....	نوع دهنی .....
Duck Typing (Programming) .....	نوع دهنی اردکی .....
Static Typing (Type Checking) .....	نوع دهنی ایستا .....
Dynamic Typing (Type Checking) .....	نوع دهنی پویا .....
Weakly Typed (Programming Language) .....	نوع دهنی ضعیف .....
Strongly Typed (Programming Language) .....	نوع دهنی قوی .....
Novell (Company) .....	نوول .....
Data Hiding (Programming) .....	نهان‌سازی داده‌ها .....

Palindrome .....	واروخته .....
Buzzword .....	واژهٔ باب روز .....
Word Processor (Software) .....	واژه‌پرداز (نرم‌افزار) .....
Interface (Computing) .....	واسط (رایانش) .....
Application Programming Interface .....	واسط برنامه‌نویسی .....
Common Gateway Interface (CGI) .....	واسط دروازهٔ مشترک .....
Responsive (Web Design) .....	واکنش‌گرا .....
Weinberg .....	واینبرگ .....
Web (World Wide Web) .....	وب .....
Website .....	وبگاه .....
Weblog .....	وبلاگ .....
Inheritance (OOP) .....	وراثت .....
Werner Vogels .....	ورنر فوگلز .....
Input/Output (I/O) .....	ورودی/خروجی .....
Patch (Computing) .....	وصله .....
Clarity .....	وضوح .....
Editor (Programming) .....	ویرایشگر .....
Visual Studio Code (Editor) .....	ویژوال استودیو کد .....
Wikimedia Commons .....	ویکی‌انبار .....
Wikipedia .....	ویکی‌پدیا .....

Vue.js (JS Web Framework) ..... ویو جی اس

۵

Hotmail .....	هات میل .....
Kernel (Operating System) .....	هسته .....
Hash (Ruby Collection) .....	هش .....
Octet (Computing) .....	هشت تایی .....
Warning (Computing) .....	هشدار .....
Hacker .....	هکر .....
Topology (Networking) .....	هم بندی .....
Peer-to-peer (Networking) .....	هم تابه همتا .....
Croutine (Programming) .....	هم روال .....
Backward Compatibility .....	هم سازی با قبل .....

۶

Yahoo! .....	یاهو! .....
Integration .....	یک پارچه سازی .....
USS Hopper .....	یواس اس هاپر .....
YouTube (Online Video Sharing Platform) .....	یوتیوب .....
Eudora (Email Client) .....	یودارا .....
Eureka (SW) .....	یوریکا .....
Yukihiro "Matz" Matsumoto .....	یوکیهیرو ماتسوموتو .....
Unix (OS) .....	یونیکس .....





# فهرست اختصارات

## A

ACA .....	Affordable Care Act
ANSI .....	American National Standards Institute
API .....	Application Program Interface
ASCII .....	American Standard Code for Information Interchange
ASP.NET .....	Active Server Pages

## B

BDD .....	Behavior-Driven Development
BDUF .....	Big Design Up Front

## C

CGI .....	Common Gateway Interface
CHIPS .....	Coding/Hands-On Integrated Projects
CMM .....	Capability Maturity Model
COD .....	Content-Oriented Didactic
CORBA .....	Common Object Request Broker Architecture
CP/M .....	Control Program/Monitor
CRUD .....	Create, Read, Update, and Delete
CRUDI .....	Create, Read, Update, Delete, and Index
CSS .....	Cascading Style Sheets

## D

DCOM .....	Distributed Component Object Model
Dev/Ops .....	Software Development & IT Operations
DNS .....	Domain Name System
DOM .....	Document Object Model

DRY ..... Don't Repeat Yourself

## E

ECMA ..... European Computer Manufacturers Association  
EJB ..... Enterprise JavaBeans (Jakarta Enterprise Beans)

## F

FaaS ..... Function as a Service  
FM ..... Frequency modulation  
FTP ..... File Transfer Protocol

## H

HTML ..... HyperText Markup Language  
HTML 5 ..... HyperText Markup Language 5  
HTTP ..... Hypertext Transfer Protocol  
HTTPS ..... Hypertext Transfer Protocol Secure

## I

IDE ..... Integrated Development Environment  
IEEE ..... Institute of Electrical and Electronics Engineers  
IMAP ..... Internet Message Access Protocol  
IP ..... Internet Protocol  
IPv4 ..... Internet Protocol Version 4  
IPv6 ..... Internet Protocol Version 5  
IT ..... Information Technology

## J

JSON ..... JavaScript Object Notation  
JSP ..... JavaServer Pages (Jakarta Server Pages)

## M

MIT ..... Massachusetts Institute of Technology  
MOCAS ..... Mechanization of Contract Administration Services  
MS-DOS ..... Microsoft Disk Operating System

0

OO ..... Object-Oriented

P

POP ..... Post Office Protocol

PWA ..... Progressive Web Application

Q

QA ..... Quality Assurance

R

REPL ..... Read-Eval-Print Loop

REST ..... Representational State Transfer

RPC ..... Remote Procedure Call

RUP ..... Rational Unified Process

S

SaaS ..... Software as a Service

SaaS ..... Software as a Service

SGML ..... Standard Generalized Markup Language

SOA ..... Service Oriented Architecture

SOAP ..... Simple Object Access Protocol

T

TCP ..... Transmission Control Protocol

TCP/IP ..... Internet Protocol Suite (Transmission Control Protocol/Internet Protocol)

TDD ..... Test-Driven Development

TMDb ..... The Movie Database

U

UDDI ..... Universal Description, Discovery, and Integration

UML ..... Unified Modeling Language

URI ..... Uniform Resource Identifier

URL ..... Uniform Resource Locator

V

VB.NET ..... Visual Basic .NET  
VSCode ..... Visual Studio Code

W

W3C ..... World Wide Web Consortium  
WSDL ..... Web Services Description Language

X

XML ..... Extensible Markup Language  
XP ..... Extreme Programming

